

COHERENS

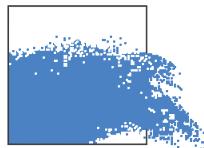
**A Coupled Hydrodynamical-Ecological Model
for Regional and Shelf Seas**

User Documentation

Version 2.12, March 23 2022

Patrick Luyten (Editor)

Royal Belgian Institute of Natural Sciences (RBINS)
Vautier street 29, 1000 Brussels, Belgium



Contributing authors

Luyten Patrick
Royal Belgian Institute of Natural Sciences (RBINS-MUMM)
Gulledelle 100
B-1200 Brussels
Belgium

Baetens Katrijn
Royal Belgian Institute of Natural Sciences (RBINS-MUMM)
Gulledelle 100
B-1200 Brussels
Belgium

Dulière Valérie
Royal Belgian Institute of Natural Sciences (RBINS-MUMM)
Gulledelle 100
111 B-1200 Brussels
Belgium

Breugem Alexander, Decrop Boudewijn, Delecluyse Kevin
International Marine and Dredging Consultants (IMDC nv)
Coveliersstraat 15
B-2600 Berchem (Antwerp)
Belgium

Heredia Gomez Marcelo and Rocabado Ivan
ANTEA Group
Poortakkerstraat 41
B-9051 Gent
Belgium

Rauwoens Pieter and Van Oyen Thomas
University of Ghent
Departement of Civil Engineering
Technologiepark 904
B-9052 Ghent
Belgium

COHERENS Licence Agreement

Copyright © 2014 MUMM <http://www.mumm.ac.be/coherens/>

The COHERENS licence agreement follows the EUPL free software licence agreement. Everyone is permitted to copy and distribute verbatim copies of the licence document, but changing is not allowed. A copy of the licence can be obtained at:

<http://ec.europa.eu/idabc/eupl>

COHERENS is a open-source multi-purpose model for regional, shelf and coastal seas, originally developed in the frame of European and national programs, by the Operational Directorate of the Royal Belgian Institute of Natural Sciences (RBINS)¹. Leading author is Patrick Luyten.

In order to boost the COHERENS users community, the COHERENS licence guarantees users the freedom to receive and run COHERENS free of charge. It also precises the terms and conditions to modify and redistribute COHERENS.

The licence also requests to make reference to the COHERENS documentation in any scientific publication presenting results obtained with COHERENS:

Luyten, P. (Editor) 2014. COHERENS — A Coupled Hydrodynamical-Ecological Model for Regional and Shelf Seas: User Documentation. Version 2.7. RBINS Report, Operational Directorate Natural Environment, Royal Belgian Institute of Natural Sciences.

¹Previously known as the Management Unit of the North Sea Mathematical Models (MUMM).

Contents

I	Introductory Manual	1
1	General overview	3
1.1	Introduction	3
1.2	Description of the program	5
1.3	User experience	9
1.3.1	Programming experience	9
1.3.2	Scientific background	10
1.4	Contents of the documentation	10
1.4.1	Structure	10
1.4.2	Suggestions for reading	12
2	Getting Started	15
2.1	Introduction	15
2.1.1	General requirements	15
2.1.2	Short Linux introduction	16
2.1.2.1	getting help	17
2.1.2.2	working with files	17
2.2	Running a test case	18
2.2.1	Installing a COHERENS test case	18
2.2.2	Compiling COHERENS	19
2.2.3	Running COHERENS	22
2.3	Post-processing the results	27
2.3.1	Visualising with Ncview	27
2.3.2	visualising with Ferret	29
2.3.3	visualising with Octave and Matlab	31
2.4	Modifying model setup	34
2.4.1	Modifying model setup via CIF	35
2.4.1.1	short CIF introduction	35
2.4.1.2	tutorial <i>river</i> test case	35
2.4.1.3	changing model setup through the CIF	36
2.4.2	Adapting model set up via the <i>Usrdef</i> files	45

2.4.2.1	installing and running the example test	45
2.4.2.2	changing model setup through the <i>Usrdef_</i> files	46
2.5	Format and specific syntax of a CIF	49
3	Compilation and installation	55
3.1	Installation	56
3.2	Compilation	58
3.2.1	C-preprocessing	59
3.2.2	Testing compilation	60
3.3	Installing and running a test case	61
3.4	Installing a user application	64
3.5	Running an application with external libraries	65
3.5.1	Parallel application	65
3.5.2	Using netCDF output format	66
3.6	Setting up a user application	66
3.7	Files for compilation and installation	68
3.7.1	<i>compilers.cmp</i>	68
3.7.2	The file <i>coherensflags.cmp</i>	69
3.7.3	The script <i>install_test</i>	72
II	Model description	73
4	Model grid	75
4.1	Model coordinates	75
4.1.1	Coordinate systems	75
4.1.2	Coordinate transforms in the horizontal	76
4.1.3	Rotated grids	79
4.1.4	Coordinate transforms in the vertical	82
4.1.4.1	σ -coordinates	82
4.1.4.2	generalised σ -coordinates	83
4.1.4.3	normalised vertical coordinate	86
4.2	Discretised model grid	87
4.2.1	Grid nodes and indexing system	87
4.2.2	Open boundaries	91
4.2.3	Conventions	92
4.2.4	Space discretisation	93
5	Hydrodynamic model	97
5.1	Hydrodynamic equations	97
5.1.1	3-D mode equations	97

5.1.1.1	Cartesian coordinates	97
5.1.1.2	transformed coordinates	101
5.1.2	2-D mode equations	104
5.1.3	Equation of state	108
5.2	Model equations on reduced grids	110
5.2.1	Water column (1-D) mode	110
5.2.2	Depth-averaged (2-D) mode	111
5.3	Turbulence schemes	112
5.3.1	Introduction	112
5.3.2	Algebraic schemes	116
5.3.2.1	Richardson number dependent formulations	116
5.3.2.2	flow-dependent formulations	117
5.3.3	RANS models	120
5.3.3.1	general form of the RANS equations	120
5.3.3.2	parameterisation of the RANS equations	122
5.3.3.3	stability functions	125
5.3.3.4	solution methods	131
5.3.3.5	mixing length formulations	133
5.3.3.6	background mixing	134
5.3.3.7	numerical lower bounds	137
5.4	Horizontal diffusion for scalars	142
5.4.1	General formulation using rotated diffusion tensor	142
5.4.2	Laplacian diffusion	142
5.4.3	Isolevel diffusion	143
5.4.4	Isopycnal diffusion	143
5.4.5	Gent-McWilliams eddy-induced transport	143
5.4.6	Limiting conditions	144
5.5	Astronomical tidal force	146
5.6	Surface boundary conditions	152
5.6.1	General form	152
5.6.2	Currents	153
5.6.3	Temperature	154
5.6.4	Salinity	156
5.6.5	Turbulence	157
5.6.6	Meteorological forcing data	159
5.7	Surface drag and exchange coefficients	159
5.7.1	Neutral formulations	160
5.7.2	Kondo's stratified formulation	161
5.7.3	Stratified case from Monin-Obukhov similarity theory	162
5.8	Solar radiation	168
5.9	Bottom boundary conditions	170

5.9.1	General form	170
5.9.2	Currents	171
5.9.3	Temperature and salinity	173
5.9.4	Turbulence	174
5.10	Lateral boundary conditions	175
5.10.1	Open boundary conditions for the 2-D mode	175
5.10.2	Open boundary conditions for the 3-D mode	184
5.10.2.1	baroclinic currents	184
5.10.3	Boundary conditions for cross-stream advective fluxes	186
5.10.3.1	3-D scalars	186
5.10.3.2	turbulence variables	187
5.10.4	Relaxation conditions	188
5.10.5	Coastal boundaries	189
5.11	Initial conditions	189
5.12	Harmonic analysis	190
5.12.1	Residuals, amplitudes and phases	190
5.12.2	Tidal ellipses	192
6	Current-wave interaction model	195
6.1	Introduction	195
6.2	Wave-current interaction at the sea bed	196
6.2.1	Soulsby-Clarke model	198
6.2.2	Malarkey-Davies model	200
6.2.3	Grant-Madsen model	200
6.2.4	Christofferson-Jonsson model	201
6.2.5	solution method	203
6.3	Wave-current interaction within the water column	204
7	Sediment transport model	205
7.1	Introduction	205
7.2	General aspects	205
7.2.1	Sediment concentrations	205
7.2.2	Density effects	206
7.2.2.1	Equation of state	206
7.2.2.2	Density stratification	206
7.2.3	Dimensionless parameters	207
7.2.4	Roughness length and bed shear stresses	208
7.3	Critical shear stress	214
7.3.1	Hiding and exposure	215
7.3.2	Bed slope	217
7.4	Settling velocity	217

7.4.1	Hindered settling	219
7.4.2	Influence of flocculation	220
7.5	Bed load transport	221
7.5.1	Introduction	221
7.5.2	Meyer-Peter and Mueller (1948)	222
7.5.3	Engelund and Fredsøe (1976)	223
7.5.4	Van Rijn (1984b)	223
7.5.5	Wu et al. (2000)	224
7.5.6	Soulsby (1997)	224
7.5.7	Van Rijn (1993)	226
7.5.8	Bed slope effects and coordinate transforms	226
7.6	Total load transport	228
7.6.1	Engelund and Hansen (1967)	228
7.6.2	Ackers and White (1973)	229
7.6.3	Madsen and Grant (1976)	229
7.6.4	Wu et al (2000)	230
7.7	Suspended sediment transport	231
7.7.1	Sediment transport equations	231
7.7.1.1	Three-dimensional transport	231
7.7.1.2	Two-dimensional sediment transport	232
7.7.2	Erosion and deposition	232
7.7.2.1	Erosion and deposition of cohesive sediment in 3-D	232
7.7.2.2	Erosion and deposition of sand in 3-D	233
7.7.2.3	Erosion-deposition of cohesive sediment in 2-D	235
7.7.2.4	Erosion-deposition of sand in 2-D	235
7.7.3	Sediment diffusivity	236
7.7.4	Numerical methods	238
7.7.4.1	Deposition flux	238
7.7.4.2	Settling velocity	239
7.7.4.3	Bed and total load	239
7.7.4.4	Median size diameter	239
7.7.4.5	Bartrnicky filter	240
7.7.4.6	Bed slope factors	240
7.7.4.7	Gaussian-Legendre quadrature	241
7.8	Flocculation model	242
7.8.1	Flocculation transport model	243
7.8.2	Floc properties	244

8 Morphological model	249
8.1 Introduction	249
8.2 Bed elevation equation	250
8.2.1 General aspects	250
8.2.2 Spatial integration	251
8.2.3 Time integration	252
8.3 Morphological acceleration	253
8.3.1 Morphological factor	254
8.3.2 Tidal averaging	254
8.4 Bed layers	256
8.4.1 Fixed layer	256
8.4.2 Active layer	257
8.4.3 Conservation of bed layer sediment	257
8.4.4 Mass balance	259
8.5 Vertical sorting	260
8.5.1 Deposition step	262
8.5.2 Erosion step	264
8.5.3 Sediment exchange fluxes	264
8.6 Avalanching	266
9 Biological model	269
9.1 Introduction	269
9.2 0-D formulation	269
9.2.0.1 Elemental ratio	271
9.2.0.2 Biomass equations	273
9.2.0.3 Organic matter equations	279
9.2.0.4 Nutrient equations	281
9.2.0.5 benthos-equations	283
10 Tracers and contaminant modules	287
10.1 Lagrangian tracer module	287
10.1.1 Lagrangian transport equations	288
10.1.1.1 floating particles	289
10.1.2 Numerical procedures	290
10.1.3 Model setup and functionalities	292
11 Structures, discharges and inundation	297
11.1 Dry cells	297
11.2 Thin dams	299
11.3 Weirs and barriers	300
11.3.1 Weirs	301

11.3.1.1	Free flow	302
11.3.1.2	Submerged flow	304
11.3.2	Barriers	305
11.3.3	Numerical implementation	307
11.4	Discharges	307
11.4.1	Continuity equation	310
11.4.2	Momentum equations	313
11.4.2.1	Discharge of scalars	313
11.4.2.2	Numerical implementation	313
11.5	Drying/wetting and inundation schemes	314
12	Numerical methods	319
12.1	Introduction	319
12.2	Time discretisation	321
12.3	Momentum equations	323
12.3.1	General procedure for the explicit case	323
12.3.1.1	predictor step	325
12.3.1.2	depth-integrated equations	327
12.3.1.3	corrector step	329
12.3.1.4	vertical current	330
12.3.2	General procedure for the implicit case	330
12.3.3	Advection schemes and time discretisation	333
12.3.3.1	introduction	333
12.3.3.2	mode splitting scheme for the 3-D momentum equations	342
12.3.3.3	mode splitting scheme for the 2-D momentum equations	344
12.3.4	Discretisation of 3-D horizontal advection	348
12.3.4.1	alongstream advection of u	348
12.3.4.2	cross-stream advection of u	349
12.3.4.3	cross-stream advection of v	350
12.3.4.4	alongstream advection of v	351
12.3.5	Discretisation of 2-D horizontal advection	352
12.3.5.1	alongstream advection of U	352
12.3.5.2	cross-stream advection of U	353
12.3.5.3	cross-stream advection of V	354
12.3.5.4	alongstream advection of V	355
12.3.6	Integrals of the baroclinic advection terms	356
12.3.7	Discretisation of vertical advection	356
12.3.7.1	vertical advection of u	356
12.3.7.2	vertical advection of v	357

12.3.8	Discretisation of 3-D horizontal diffusion	358
12.3.9	Discretisation of 2-D horizontal diffusion	359
12.3.10	Integrals of the baroclinic diffusion terms	360
12.3.11	Discretisation of vertical diffusion	360
12.3.12	Diffusion coefficients for momentum	361
12.3.12.1	horizontal diffusion coefficients	361
12.3.12.2	vertical diffusion coefficient	362
12.3.13	Discretisation of the baroclinic pressure gradient	363
12.3.13.1	second-order method	364
12.3.13.2	z -level method	365
12.3.13.3	cube-H method	365
12.3.14	Tidal force	368
12.3.15	Surface and bottom boundary conditions	370
12.3.15.1	surface boundary conditions	370
12.3.15.2	bottom boundary conditions	371
12.3.16	Lateral boundary conditions for the 2-D mode	372
12.3.16.1	open boundary conditions for transports	372
12.3.16.2	open boundary conditions for 2-D advective and diffusive fluxes	379
12.3.16.3	boundary conditions at closed lateral boundaries	381
12.3.17	Lateral boundary conditions for the 3-D currents	381
12.3.17.1	open boundary conditions for horizontal 3-D currents	381
12.3.17.2	open boundary conditions for the advective and diffusive fluxes of 3-D currents	384
12.3.17.3	boundary conditions for the 3-D mode at closed lateral boundaries	384
12.3.18	Solution of the discretised equations for momentum	384
12.3.18.1	composition of the tridiagonal matrix	385
12.3.18.2	solution of tridiagonal systems	388
12.3.19	Elliptic equation for the free surface correction	388
12.3.19.1	interior terms	388
12.3.19.2	open boundary terms	389
12.4	Scalar transport equations	394
12.4.1	General aspects of discretisation	394
12.4.2	Alternative formulation of the transport equation	395
12.4.3	Time discretisation	396
12.4.3.1	integration without advection	396
12.4.3.2	integration with advection but without operator splitting	397

12.4.3.3	integration with operator splitting	397
12.4.4	Discretisation of advection	398
12.4.4.1	advection in the X-direction	398
12.4.4.2	advection in the Y-direction	399
12.4.4.3	advection in the vertical direction	400
12.4.4.4	corrector terms	400
12.4.5	Discretisation of diffusion	401
12.4.5.1	diffusion in the X-direction	401
12.4.5.2	diffusion in the Y-direction	401
12.4.5.3	diffusion in the vertical direction	401
12.4.6	Diffusion coefficients for scalars	402
12.4.6.1	horizontal diffusion coefficients	402
12.4.6.2	vertical diffusion coefficient	402
12.4.7	Boundary conditions	402
12.4.7.1	surface boundary conditions	402
12.4.7.2	bottom boundary conditions	403
12.4.7.3	lateral boundary conditions	404
12.4.8	Solution of the discretised equations for scalars	405
12.5	Turbulence transport equations	409
12.5.1	Time discretisation	409
12.5.1.1	integration without advection	410
12.5.1.2	integration with advection but without operator splitting	410
12.5.1.3	integration with operator splitting	410
12.5.2	Discretisation of advection	411
12.5.2.1	advection in the X-direction	411
12.5.2.2	advection in the Y-direction	412
12.5.2.3	advection in the vertical direction	413
12.5.2.4	corrector terms	413
12.5.3	Discretisation of diffusion	414
12.5.3.1	diffusion in the X-direction	414
12.5.3.2	diffusion in the Y-direction	414
12.5.3.3	diffusion in the vertical direction	414
12.5.4	Diffusion coefficients for turbulence variables	414
12.5.4.1	horizontal diffusion coefficients	414
12.5.4.2	vertical diffusion coefficients	415
12.5.5	Production and sink terms	415
12.5.6	Boundary conditions	416
12.5.6.1	surface boundary conditions	416
12.5.6.2	bottom boundary conditions	417
12.5.6.3	lateral boundary conditions	418

12.5.7	Solution of the discretised equations for turbulent transport variables	418
12.6	Discretisations on reduced grids	422
12.6.1	Discretised 1-D mode equations	422
12.6.2	Discretised depth-integrated equations	423
12.7	Solution procedure	424
III	Description of the model code	427
13	Program conventions and techniques	429
13.1	Implementation of FORTRAN 90	429
13.1.1	COHERENS programming conventions	430
13.1.2	Data types	434
13.1.3	Allocatable arrays	436
13.1.4	Derived types	438
13.1.5	Modules	441
13.1.6	Generic procedures	443
13.1.7	Internal documentation and structured layout of the code	444
13.2	Specific program features	455
13.2.1	Date and time formats	455
13.2.2	Data flags	457
14	Model input and output	459
14.1	Classification of model files	459
14.2	Default file names	461
14.2.1	title	461
14.2.2	pid	461
14.2.3	form	462
14.2.4	filedesc	462
14.2.5	filenum	464
14.2.6	freqnum	465
14.2.7	dim	465
14.3	Formats of monitoring files	465
14.3.1	Log files	465
14.3.2	Error files	468
14.3.3	Warning file	471
14.3.4	Timer report file	471
14.4	Central input file	476
14.4.1	Syntax of a CIF	476

14.4.2 CIF blocks	477
14.4.3 Order of definitions	478
14.5 Forcing files	482
14.5.1 General aspects	482
14.5.2 Data contents of forcing files	487
14.5.3 Standard format of forcing files	494
14.5.3.1 ASCII files	494
14.5.3.2 unformatted binary files	500
14.5.3.3 netCDF files	501
14.6 User output files	506
14.6.1 General aspects	506
14.6.2 Structure of user output files	510
14.6.3 Format of files with user-defined output	512
14.6.3.1 ASCII files	512
14.6.3.2 unformatted binary files	519
14.6.3.3 netcdf files	519
15 Model grid and spatial interpolation	525
15.1 Model grid arrays	525
15.1.1 Array shapes	525
15.1.2 Parameters and arrays related to the model grid	526
15.1.2.1 definition of the model grid	526
15.1.2.2 definition of the open boundaries	528
15.1.2.3 grid spacings	531
15.1.2.4 pointer arrays	533
15.2 Interpolation of model arrays at a different node	535
15.2.1 Interpolation without land flags	535
15.2.2 Interpolation with land flags	537
15.3 Curvilinear, index and relative coordinates	538
15.4 Interpolation of a 2-D external data grid at the model grid	540
15.4.1 General description of the procedure	540
15.4.2 Implementation	541
15.5 Interpolation of model data at external locations	544
15.5.1 General description of the procedure	544
15.5.2 Implementation	548
16 Aspects of parallelisation	553
16.1 Basic principles	553
16.1.1 Implementation of MPI	553
16.1.2 Principles of the parallel code	555
16.2 Domain decomposition	555

16.2.1	Definition	555
16.2.2	Local grid indexing system	556
16.3	Halos	559
16.4	Communications	560
16.4.1	Send and receive in MPI	560
16.4.2	Sort of communications	562
16.4.3	Implementation	565
16.4.3.1	all-to-all operations	565
16.4.3.2	exchange operations	566
16.4.3.3	program routines for communications	567
16.5	Local versus global array indexing	568
17	Structure of the model code	571
17.1	Source code files	571
17.2	Structure diagrams	576
17.2.1	General structure	576
17.2.2	Initialisation procedures	577
17.2.3	Time loop	579
17.2.4	Open boundary and surface forcing data input	581
17.2.5	Finalisation procedures	584
IV	User manual	595
18	Introduction	597
18.1	Setup methods	597
18.2	usrdef routines	598
18.3	Preparing a simulation	602
18.4	Setup conventions	604
18.4.1	File formats	604
18.4.2	Time parameters	604
18.4.3	Key ids	606
18.4.4	Data types	606
18.4.5	Named constants and array dimensions	606
18.4.6	Data flags	608
18.4.7	Variable units	608
18.4.8	Dimensions of setup arrays	609
18.5	Selecting method for model setup	609
18.6	Some recommendations for programming	609

19 General and physical model parameters	617
19.1 Parameters for monitoring	617
19.1.1 Cold start	618
19.1.2 Log files	618
19.1.3 Error files	619
19.1.4 Warning file	619
19.1.5 Timer file	619
19.1.6 Monitoring files	620
19.1.7 General switches	620
19.1.8 Number of processes for each sub-model	621
19.2 Model switches	621
19.2.1 Model grid	622
19.2.2 Interpolation	623
19.2.3 Hydrodynamics	624
19.2.4 Density	625
19.2.5 Sediment and tracers modules	626
19.2.6 Advection	627
19.2.7 Horizontal diffusion	629
19.2.8 Vertical diffusion	630
19.2.9 Turbulence schemes	631
19.2.10 Bottom boundary conditions	633
19.2.11 Surface meteo	634
19.2.12 Surface fluxes	635
19.2.13 Open boundary conditions	636
19.2.14 Nesting	638
19.2.15 1-D applications	638
19.2.16 Tides	638
19.2.17 Surface waves	639
19.2.18 Drying/wetting scheme	640
19.2.19 Structures	640
19.2.20 Numerical switches	641
19.2.21 MPI mode	642
19.2.22 User output	644
19.2.23 NetCDF	644
19.2.24 Various	645
19.2.25 Internal switches	645
19.3 Model parameters	646
19.3.1 Process numbers for the domain decomposition	646
19.3.2 Grid parameters	647
19.3.2.1 Bathymetry and geographical location	648
19.3.2.2 Grid transformation	648

19.3.3 Date and time parameters	649
19.3.4 Parameters for diffusion	650
19.3.5 Tidal parameters	651
19.3.6 Open boundary conditions and nesting	652
19.3.7 Bottom boundary conditions	653
19.3.8 Surface boundary conditions	653
19.3.9 Reference values for physical variables	654
19.3.10 Optical parameters	654
19.3.11 Parameters for the structure module	655
19.3.12 Parameters for the drying-wetting scheme	655
19.3.13 Parameters used in numerical schemes	656
19.3.14 Physical model constants	657
19.3.15 Turbulence model parameters	657
19.3.16 Parameters for user-defined output	660
19.3.17 Formats used for ASCII output	661
19.4 Attributes of forcing files	662
19.4.1 Forcing attributes for input data (<i>itype</i> =1)	670
19.4.2 Forcing attributes for output data (<i>itype</i> =2)	672
19.5 Parameters for surface data grids	675
19.5.1 Grid descriptors	675
19.5.2 Attributes of the horizontal model grid	675
19.5.3 Attributes of an external data grid	676
20 Model grid and initial conditions	679
20.1 Model grid and bathymetry	679
20.2 Domain decomposition	682
20.3 Initial physical conditions	684
21 Open boundary conditions and nesting	691
21.1 2-D open boundary conditions	692
21.1.1 Open boundary specifications for the 2-D mode	692
21.1.1.1 Type of 2-D open boundary conditions conditions	693
21.1.1.2 Data file specifications	695
21.1.1.3 Amplitudes and phases	698
21.1.2 Open boundary data for the 2-D mode	699
21.1.3 2-D open boundary conditions: code example	703
21.1.2 3-D open boundary conditions	708
21.2.1 Open boundary specifications for the 3-D mode	708
21.2.2 Type of 3-D open boundary conditions	710
21.2.3 Data file specifications	711

21.2.4	Open boundary data for the 3-D mode	713
21.2.5	3-D open boundary conditions: code example	716
21.3	Nesting	722
21.3.1	Specifications of the sub-grid open boundaries	724
21.3.2	Locations of the sub-grid open boundaries	725
22	Surface forcing	727
22.1	Water column surface forcing	727
22.1.1	Surface forcing specifiers for the 1-D mode	727
22.1.2	Surface forcing data for the 1-D mode	729
22.2	2-D surface forcing	730
22.2.1	Surface grid in absolute coordinates	730
22.2.2	Surface grid in relative coordinates	732
22.2.3	Surface forcing data	734
22.2.3.1	Meteorological forcing data	735
22.2.3.2	Sea surface temperature data	737
22.2.3.3	Surface wave data	738
22.2.3.4	PAR data	739
23	Sediment and morphology manual	741
23.1	Sediment transport and flocculation model setup	741
23.1.1	Switches and parameters for the multi-fraction sediment module	742
23.1.1.1	Switches for the multi-fraction model	742
23.1.1.2	Parameters for the multi-fraction model	744
23.1.2	Switches and parameters for the flocculation model	745
23.1.2.1	Switches for the flocculation model	745
23.1.2.2	Parameters for the flocculation model	747
23.1.3	Forcing file parameters	748
23.1.4	Switches and parameters per fraction	748
23.1.5	Initial conditions for the sediment module	752
23.1.5.1	multi-fraction model	752
23.1.5.2	Flocculation model	753
23.1.6	Sediment open boundary conditions	754
23.1.7	Sediment nesting	754
23.2	Morphological model setup	755
23.2.1	Parameters for the morphological module	755
23.2.2	Parameters for the morphological module	757
23.2.3	monitoring info	759
23.2.4	Initial conditions for the morphology	759
23.3	Sediment output	761

24 Biology manual	763
24.1 Switches and parameters for the biology	763
24.1.1 Switches	763
24.1.2 Parameters	764
24.2 Initial conditions for the biology module	769
24.3 Biology open boundary conditions	770
24.4 Biological nesting	771
25 Tracers and contaminant manual	773
25.1 Lagrangian transport	773
25.1.1 Switches and model parameters	774
25.1.1.1 switches	774
25.1.1.2 model parameters	776
25.1.1.3 forcing files	778
25.1.2 Initial conditions	778
25.1.3 Output in user-defined format	780
25.1.4 Properties depending on the particle label and cloud attributes	781
25.1.4.1 Properties of particle labels	781
25.1.4.2 cloud attributes	782
25.1.5 Particle trajectory output	783
25.1.6 Particle cloud releases	786
25.1.7 External model grid for off-line applications	787
25.1.8 External hydrodynamic data for off-line applications	789
26 Structure and discharge manual	793
26.1 Dry cells	793
26.2 Thin dams	794
26.3 Weirs and barriers	795
26.4 Specifiers for discharges	797
26.5 Discharge data	798
27 User output	801
27.1 Introduction	801
27.2 Output attributes	802
27.2.1 Output variables	802
27.2.2 Output files	805
27.2.3 Output space and time grid	806
27.2.4 Output data stations	808
27.2.5 Output procedures	808
27.3 Time series output	809

27.3.1	Specifiers for time series output	809
27.3.2	Time series output data	811
27.3.2.1	0-D time series data	811
27.3.2.2	2-D time series data	812
27.3.2.3	3-D time series data	813
27.4	Time averaged output	814
27.4.1	Specifiers for time averaged output	815
27.4.2	Time averaged output data	816
27.4.2.1	0-D time averaged data	817
27.4.2.2	2-D time averaged data	817
27.4.2.3	3-D time averaged data	818
27.5	Harmonic analysis and output	819
27.5.1	Harmonic frequencies	820
27.5.2	Specifiers for harmonic output	821
27.5.3	Harmonic output data	824
27.5.3.1	0-D harmonic data	825
27.5.3.2	2-D harmonic data	825
27.5.3.3	3-D harmonic data	826
27.6	Output data grid, metadata and output formats	827
27.6.1	Output data grid	827
27.6.2	Data file info	828
27.6.2.1	metadata	829
27.6.2.2	data	831
27.7	User-defined output	832
28	Central Input File	833
28.0.1	Syntax of a CIF	833
28.1	CIF blocks	835
28.2	Special procedures	836
28.2.1	Key ids	836
28.2.2	Derived type arrays	838
28.2.3	CIF flags	839
28.3	Example	841
29	Layout of standard forcing files	851
29.1	Introduction	851
29.2	Structure of a non-COHERENS standard forcing file	853
29.2.1	ASCII or unformatted binary forcing files	853
29.2.2	netCDF forcing files	856
29.3	Structure of forcing files with multi-variable arrays	860
29.3.1	Recognition by index	860

29.3.2 Recognition by name	862
30 Contents of standard forcing files	865
30.1 Model grid	866
30.2 Domain decomposition	868
30.3 Initial and final conditions	868
30.3.1 Physical module	869
30.3.2 Multi-fraction sediment module	873
30.3.3 Flocculation module	874
30.3.4 Morphological module	875
30.3.5 Biological module	877
30.3.6 Particle tracer module	878
30.4 Surface grids	880
30.4.1 Absolute coordinates of surface grids	880
30.4.2 Relative coordinates of surface grids	881
30.5 Nesting	883
30.5.1 Number of nesting locations	883
30.5.2 Locations of sub-grid open boundaries	884
30.5.3 Sediment particle attributes	886
30.5.4 Properties of particle labels and clouds	889
30.6 Surface boundary conditions for 1-D applications	891
30.6.1 Tidal surface forcing	891
30.6.2 Surface forcing data	892
30.7 Open boundary conditions specifiers	893
30.7.1 Specifiers for 2-D normal boundary conditions	893
30.7.2 Specifiers for 2-D tangential open boundary conditions	895
30.7.3 Specifiers for 3-D normal open boundary conditions . .	897
30.7.3.1 Without sub-variables	897
30.7.3.2 With sub-variables	899
30.7.4 Specifiers for 3-D tangential open boundary conditions	900
30.8 Open boundary data	901
30.8.1 2-D normal open boundary data	901
30.8.2 2-D tangential open boundary data	902
30.8.3 Profile data for 3-D variables	902
30.8.3.1 Physical variables	902
30.8.4 Profile data for the multi-fraction sediments model .	903
30.8.5 Profile data for the flocculation model	904
30.8.6 Profile data for the biological model	905
30.9 Surface forcing data	906
30.9.1 Meteorological forcing	906
30.9.1.1 Single point grid	906

30.9.1.2	Multiple point grid	907
30.9.2	Surface temperature forcing	908
30.9.3	Surface wave forcing	908
30.9.3.1	Single point grid	909
30.9.3.2	Surface data grid	910
30.9.4	Surface biological (PAR) data	910
30.10	Structures	911
30.10.1	Dry cells	911
30.10.2	Thin dams	911
30.10.3	Weirs and barriers	912
30.11	Discharges	913
30.11.1	Discharge locations	913
30.11.2	Discharged volume	914
30.11.3	Discharge area and direction	914
30.11.4	Discharged salinity	915
30.11.5	Discharged temperature	915
30.11.6	Discharged sediment concentrations	916
30.12	Particle model	916
30.12.1	Particle clouds	916
30.12.2	Particle model grid	917
30.12.3	Physical data for the particle module	919
A	Transformed model equations	921
A.1	Horizontal transformation	921
A.2	Vertical transformation	922
B	Solutions of the RANS equations	927
B.1	Non-equilibrium method	927
B.2	Quasi-equilibrium method	928
B.3	Equilibrium method	930
C	Discretisation of the Coriolis force	931
D	Energy balance equation	933
E	List of standard output variables	935
	Bibliography	939

List of Figures

2.1	Screenshot of using <code>NcView</code>	28
2.2	Screenshot of using <code>Ferret</code>	31
2.3	Map of the Bohai sea	32
2.4	Example figure generated with <code>Octave</code>	34
3.1	COHERENS file directory tree	57
4.1	Example of a uniform (left) and non-uniform rectangular grid (right).	77
4.2	Example of a curvilinear grid	78
4.3	Example of a model grid with ragged boundaries	80
4.4	Example of a rotated grid	81
4.5	Transformed coordinate $\hat{\sigma} = F(\sigma)$ (a) and vertical grid spacing normalised to the total water depth $\Delta\hat{\sigma} = \partial F/\partial\sigma/N$ (b)	84
4.6	Distribution of vertical levels along a transect from Denmark to Norway: uniform σ -coordinates (a), non-uniform σ -coordinates using (4.35) (b).	86
4.7	Layout of the (global) computational grid in the horizontal.	88
4.8	Grid indexing in the horizontal plane.	89
4.9	Layout of the computational grid in the vertical.	90
4.10	Grid indexing in three-dimensional space.	92
5.1	Stability coefficients S_u , S_b as a function of the Richardson number using the equilibrium method for different RANS models	129
5.2	Surface drag and exchange coefficients as function of wind speed and according to different formulations	167
5.3	View of an open boundary section where the total discharge is distributed.	182
5.4	Scheme of a model with three open boundaries: water level and Neumann boundary conditions (Delft3D, 2009).	183

5.5	Time evolution of a concentration at the open boundary using (5.378).	188
7.1	Ripple roughness (m) according to three formulations using a bottom excursion amplitude A_b of 1.45 m. The Li <i>et al.</i> (1996) scheme is not shown since it depends also on the shear stress.	212
7.2	Critical shear stress τ_{cr} (Pa) according to the three different formulations using $\nu=10^{-6}\text{m}^2/\text{s}$, $s=2.65$	216
7.3	Settling velocity (m/s) according to the five different formulations. Parameters are the same as in Figure 7.2.	218
7.4	Bed load transport (kg/m/s) as function of velocity for the six available schemes without wave effects, using a particle size of 1 mm and a critical Shields parameter $\theta_{cr} = 0.03$ (Wu <i>et al.</i> , 2000).	225
7.5	Total load transport (kg/m/s) as function of velocity for the five available schemes using a particle size of 400 μm	230
7.6	Numerically determined time scale and fitted polynomial	236
8.1	The deposition algorithm for two non-empty/empty bottom layers (left frame) or an empty bottom layer combination (right frame).	263
8.2	The erosion algorithm with creation of a fictive fixed layer with thickness and bed fractions equal to zero.	265
9.1	(a) Mass flows of the water quality module, with SR the sedimentation–resuspension module. (b) the sedimentation–resuspension module (b) with WQ module the water quality module	272
11.1	Dry cell defined at grid location (m,n).	298
11.2	Groups of dry cells defined in a computational domain.	298
11.3	Definition of thin dams at V- and U-nodes.	299
11.4	Scheme of a barrier (submerged structure).	300
11.5	Blocking of flow when the water depth is less than the crest level.	301
11.6	Scheme of a “weir” in 3-D mode for non-submerged and submerged conditions (compared with the height of a structure). .	302
11.7	Scheme of free flow over a weir.	303
11.8	Scheme of submerged flow over a weir.	304
11.9	Sketch of a barrier.	305
11.10	Scheme of a submerged orifice flow.	306
11.11	Schematization of a discharge in a 3-D simulation	308

11.12	Location of a discharge point in a grid cell.	308
11.13	2-D normal discharge	310
11.14	2-D momentum discharge	310
11.15	3-D normal discharge distributed uniformly over all layers	311
11.16	3-D normal discharge defined at a specific height	311
11.17	3-D momentum discharge distributed uniformly over all layers	312
11.18	3-D momentum discharge defined at a specific height	312
12.1	Numerical grid for the 1-D advection problem.	335
12.2	Illustration of the z -level interpolation scheme.	366
15.1	Illustration of the indexing for X- and Y-node open boundaries. U- and V-node open boundaries are indicated by empty circles, X- and Y-node open boundaries by solid circles. The corresponding indices are given by numbers in normal font (U- and V-points) or in bold (X- and Y-points).	532
15.2	Interpolation of model grid arrays at a different node.	536
15.3	Curvilinear versus index coordinates.	539
15.4	Interpolation of external data to the model grid.	540
15.5	Illustration of the nesting procedure in the horizontal plane. .	546
15.6	Definition of vertical relative coordinates.	548
16.1	CPU efficiency, defined as the CPU time for a serial run divided by the one obtained for the parallel run, as function of the number of processes.	554
16.2	Examples of allowed and non-allowed domain decompositions. .	556
16.3	Domain decomposition for the North Sea area with 128 processes based on a 10×19 domain grid.	557
16.4	Horizontal layout of the computational grid on a local sub-domain.	558
16.5	Illustration of an halo for an array defined on a local sub-domain. .	559
16.6	Partitioning of a sub-domain halo.	560
16.7	Communication pattern for exchange operations.	566
16.8	Example of index mapping between global and local arrays. .	568
17.1	General structure of COHERENS.	576
17.2	Schematic diagram of all initialisation procedures.	585
17.3	Structure diagram of the time loop.	586
17.4	Diagram of routine <code>current_2d</code> which solves the 2-D mode equations.	587

17.5	Diagrams of the routines <code>current_pred</code> and <code>current_corr</code> which solve the 3-D momentum equation at the predictor and corrector step.	588
17.6	Diagrams of the routines <code>temperature_equation</code> and <code>salinity_equation</code> which solve the temperature and salinity equations.	589
17.7	Diagram of the routine <code>sediment_equation</code> which is the “main” routine of the sediment transport model.	590
17.8	Diagram of the routine <code>sediment_advdiffequation</code> which solves the transport equations for sediments.	591
17.9	Diagrams of the routines used for defining and updating 2-D open boundary conditions and data.	592
17.10	Diagrams of the routines used for defining and updating 3-D open boundary conditions and data.	592
17.11	Diagram of the routines used for defining and updating data from an external 2-D grid.	593
21.1	Example showing how to define the arrays <code>no2dobuv</code> and <code>index2dobuv</code>	697
21.2	Example how to define the open boundary specifier arrays.	714

List of Tables

3.1	Test case descriptions	62
4.1	Upper bounds for the grid indices (i,j,k) as function of nodal type.	91
4.2	Subscript and superscript notation used in the numerical discretisation formulae.	95
5.1	Values of the empirical parameters in the McDougall <i>et al.</i> (2003) equation of state.	109
5.2	Values of the turbulence constants in the RANS equations according to the different models implemented in COHERENS.	123
5.3	Values of critical parameters according to different RANS models.	130
5.4	Parameters used in different turbulence schemes (except those listed in Tables 5.2 and 5.3) and their default values.	139
5.5	Doodson numbers, frequencies (degrees/h), amplitudes (cm) and Greenwich arguments (degrees) of the tidal constituents which can be used for astronomical and open boundary forcing.	150
5.6	Doodson numbers, origin, frequencies (degrees/h) and Greenwich arguments (degrees) of the over tides which can be used for the open boundary forcing.	152
5.7	Empirical parameters used in the Kondo (1975) formulations for the neutral surface drag and exchange coefficients.	162
7.1	Overview of applicability of sediment transport formula, i.e. conditions of the measurements used as calibration data (if available).	222
9.1	The state variables of the ecological model	270
9.2	The elemental ratio's of the state variables of biological origin; n denotes element (C, Si, N or P)	273

9.3	The parameters used to describe biomass flows of the water column. * means the value is calculated within the model, ** means the value changes over time and is given as time dependent input from on-site measurements	280
9.4	The parameters used to describe nutrient and organic matter flows of the water column	284
9.5	The parameters used to describe flows of the sediment	285
11.1	Conditions for the determination of free or submerged	303
12.1	Parameters and variables used in the numerical description. Global and local FORTRAN names refer to the variables as defined on respectively the global and local (parallel) grid.	323
12.2	Overview of the operators used in the numerical discretisations.	338
12.3	Definitions of the fluxes used in the numerical discretisations. .	346
12.4	Discretisation schemes for each of the available surface and bottom boundary conditions for turbulent variables.	417
13.1	Model data types.	435
14.1	Key ids for error coding and associated error messages.	469
14.2	Timer key ids and their meaning.	472
14.3	Data contents for each type of input forcing file. In the last column 'R', 'T', 'C', 'D' denote respectively real, integer, character and derived type data.	488
15.1	Model grid parameters and arrays.	529
15.2	Open boundary parameters and arrays.	530
17.1	List of declaration module files.	572
17.2	List of module routine files.	573
17.3	List of files with external procedures.	574
17.4	List of user-defined routine files.	575
18.1	Overview of all <i>Usrdef</i> _ files and <i>usrdef</i> _ routines in the program.	599
18.2	List of available key classes	607
18.3	List of type names and KIND parameters.	607
18.4	List of constants used in the declaration of model setup arrays and strings.	613
18.5	Units of principal variables.	614
18.6	Dimension names used in the model setup for forcing arrays. .	615

19.1 Listing of all forcing types, the <code>usrdef_</code> routine where they are defined if the status attribute equals 'N' and their purpose.	663
22.1 Key ids for surface metereorological data	737
22.2 Key ids for surface wave data	739
24.1 Key ids for biological state variables.	772
27.1 List of available elliptic parameters (number,variable key id, description of the parameter and the dimension of the vector which determines the ellipse).	823
28.1 List of available CIF blocks	837
28.2 List of all derived type arrays used in the model setup. The last column indicates whether the values of a array component are array-wise (A) or element-wise (E).	840
30.1 Values of the switches <code>iopt_mode_2D</code> and <code>iopt_mode_3D</code>	870
E.1 Key ids of the variables which can be used for defining standard output variables	935
E.2 Key ids of the variables which can be used for defining standard output variables from the sediment module	938

Preface

This report is the User Documentation of COHERENS version V2.5. Changes with respect to earlier release(s) are documented in the accompanying Release Notes.

Version 1.0

COHERENS is a three-dimensional multi-purpose numerical model, designed for application in coastal and shelf seas, estuaries, lakes and reservoirs. The model is available as a free source code for the scientific community and can be considered as a tool for a better understanding of the physical and ecological processes and for the prediction and monitoring of waste material in shelf seas, the coastal zone and estuaries. The design, testing and validation of a three-dimensional model required years of efforts. Its ease of implementation across a range of computing platforms means that it is attractive to groups who have a sufficient modelling expertise and have need for sophisticated model products. Important advantages of the model are its transparency due to its modular structure and its flexibility because of the possibility to select different processes, specific schemes and different types of forcing for a particular application. The program structure allows users to perform process as well as predictive and operational setups without knowledge of the detailed model structure.

The first version of the program, now denoted as COHERENS V1 was developed over the period 1990-1998 by a multinational group, as part of several European projects. The physical core part was written and coupled with biological and resuspension modules during the PROFILE project. The aim, at that time, was to apply the model for the study of coastal regions of freshwater influence (ROFIs). A Lagrangian particle tracer and Eulerian contaminant module were added as part of the model intercomparison experiments conducted in NOMADS. The question arose at that time whether the model could be released as a public domain community model. This was the start of the COHERENS project. The objectives were to reshape the model

into portable code, develop verification tests, perform further validation and provide an extensive documentation which covers all aspects of the program so that users can make their setup independently. Partners in the project were MUMM-RBINS (Brussels, Belgium), Proudman Oceanographic Laboratory (POL, Liverpool, UK), Napier University (NU, Edinburgh, UK) and the British Oceanographic Data Centre (BODC, Liverpool, UK).

The model code and documentation have been made available for the scientific community and management authorities in April 2000. Since then, users can register on the **COHERENS** Web site

<http://www.mumm.ac.be/coherens>

The program was, in the first years after the release, distributed through CD-ROMs, but can now easily be downloaded after electronic registration. The registration list now counts more than thousand users.

Acknowledgements

Principal authors of **COHERENS** V1 are Patrick Luyten (MUMM-RBINS), Eric Jones and Roger Proctor (POL), Paul Tett and Karen Wild-Allen (NU) and Andy Tabor (BODC). The following persons, who made an important contribution to the development of **COHERENS** V1 are acknowledged: : Judith Wolf (general concept of the program), Kevin Ruddick (circulation and transport modules), Eric Deleersnijder (numerical methods), Claire Smith (biology), Sarah Jones (resuspension and sediment module) and Kajo Mirbach (Lagrangian tracers). The work was funded by the European Union under contracts MAST-0050-C (Profile I), MAST-900064, MAS2-CT93-0054 (Profile II) and MAS3-CT97-0088 (Coherens).

Version 2.0

More than 10 years passed since the official release of **COHERENS** V1. In the mean time the computing power increased significantly by the introduction of parallel clusters. Improvements and additions of new schemes (such as a drying and wetting algorithm) have been suggested. Preparation for a new version of **COHERENS** V2.0 started in 2003 during the European FP5 project ODON (2003-2005) and continued in the FP6 project ECOOP (2007-2009). The code has been completely rewritten in **FORTRAN 90** and can be used both in sequential and parallel mode using the **MPI** parallel communication library. Output can (optionally) be provided in portable **netCDF** format from within the program. Both curvilinear and generalised σ -coordinates (also called s -coordinates) are available for creating the model grid. A more

detailed listing of all new features is given in Chapter 1. Several years were needed for debugging and testing of the code, in particular for all aspects of the parallelisation.

The program is now ready for release as a free software and is given the official name of COHERENS V2.0. Modules for biology, sediments and particle tracers are not yet implemented in the current version. Contrary however to COHERENS V1 which has not been updated in more than ten years, V2.0 is a more dynamic version than V1. To this date several COHERENS users provide regular updates in the framework of ongoing national and international projects. Current activities include the implementation of a generic biological model, a more extended sediment transport module which takes account of different size classes, a particle tracer module, structures for engineering applications in portuaries and water ways, an implicit scheme which removes the mode splitting scheme, various inundation schemes,

Acknowledgements

COHERENS V2.0 has been written by Patrick Luyten (RBINS-MUMM). Roger Proctor (POL) is acknowledged for providing access to supercomputing facilities at CSAR (Univ. of Manchester) and the POL cluster at POL during the ODON project. Useful advice for writing the code in parallel was given by Mike Ashworth (STFC, Daresbury Laboratory). Testing and debugging was facilitated by the assistance of several beta-users. Special thanks are for Kaat Vandekerckhove and Philippe Hyde (ANTEA, Ghent, Belgium) for their assistance in writing this User Documentation. The following persons at MUMM are thanked for reviewing the documentation,: Katrijn Baetens, Valérie Duliere, Sébastien Legrand, José Ozer and Stéphanie Ponsar. The construction of V2.0 has been made possible thanks to the funding by the European Union within the ODON (contract no. EVK3-CT-2002-00082) and ECOOP (contract no. 36355) projects and by the VLABEL project funded by the Flemish Region in Belgium (Uitbouw numeriek modelinstrument voor de Noordzeehavens, Bestek 16EF/2008/02).

February 2011

Version 2.1.2

The following features have been implemented:

1. CIF (Central Input File) utility. An option is foreseen which permits to define model parameters (switches, time parameters, grid parameters, model constants, forcing attributes, definitions for user output) through input from a CIF, instead of making calls to `usrdef` interface routines (Sections 14.4, 18.3).
2. A large series of “standard” variables can be selected by the user for output using their key id value. In that case, the variable’s attributes and output values are automatically generated by the program (Chapter 27).
3. Additional boundary conditions for the baroclinic current can be selected (Sections 5.10.2.1 and 12.3.17.1).

November 2011

Version 2.2.1

This is technical version, intended for future implementation of structures (thin dams, current deflection walls, weires, groines).

December 2011

Version 2.3

- The existing drying and wetting algorithm has been extended by the implementation of additional mask criteria.
- The program can be used to simulate the inundation of land areas by (e.g.) an incoming tide or storm surge.
- Two new test cases for drying and wetting have been implemented.

Developers are the ANTEA Group in collaboration with RBINS-MUMM.

April 2012

Version 2.4

A first version of a semi-implicit algorithm has been implemented for the free surface slope term in the momentum equation. With this method the mode-splitting technique is no longer needed and all transport equations are updated with the same (larger) 3-D time step.

The new code has been developed by Pieter Rauwoens and Thomas Van Oyen (University of Ghent, Belgium).

June 2012

Version 2.5

An extended sediment transport module has been implemented. This version can therefore be considered as a major update of the code. Details are described in Chapter 7 of the User Documentation. The main features are:

- a module for the advective-diffusive transport of suspended sediments
- different fractions for the simulation of graded sediments
- near-bed boundary conditions for sand as well as mud
- various methods for bed and total load transport
- different formulae for the settling velocity (including hindered settling, flocculation), critical shear stress,
- turbulence damping caused by vertical stratification due to sediment concentrations
- turbidity flows caused by horizontal sediment concentration gradients.

A first version of a surface wave module has been implemented. A full current-wave interaction module is foreseen for a future COHERENS version. The aim here is to provide wave parameters used in some of the formulations for bed and total load transport.

The new code has been developed by Alexander Breugem, Boudewijn Decrop and Kevin Delecluyse (IMDC, Belgium).

June 2014

Version 2.6

The following model units for the simulation of hydraulic structures and discharges have been added:

- Cells can be selected which are taken as permanently dry during the simulation.
- Thin dams can be defined as infinitely thin vertical walls. They are located at velocity nodes and prohibit flow exchange and fluxes of scalars between the two adjacent computational grid cells without reducing the total wet surface and the volume of the model.
- Weirs are similar to thin dams, except that a weir can be inundated, in which case an energy loss is generated. This structure will work as a thin dam in cases where the total water depth upstream of the structure is less than the crest level of the structure. In this case a blocking of flow exchange is imposed by the module. Groynes are typical examples of weirs.
- Barriers are defined as barriers with an additional opening (also called “orifices”), generally taken close to the bottom
- A discharge module has been implemented. Discharges are represented as sources or sinks in the continuity, momentum or scalar equations supplied at specified (fixed or moving) locations at the surface, bottom or within the water column (e.g. discharge structures, pumping stations, discharge from moving ships ...) by adding water to the water column with a specified salinity, temperature or contaminant concentration. The discharge may or may not have a preferential direction.

June 2014

Version 2.7

New open boundary conditions for scalars and momentum (including tangential boundary conditions for currents) are added.

December 2014

Part I

Introductory Manual

Chapter 1

General overview

1.1 Introduction

During the last 25 years several two- and three-dimensional numerical model codes have been developed and made available for applications in the global ocean, shelf and regional seas, the coastal zones and estuaries. Without being complete we can mention public domain codes such as the global ocean MOM model (Modular Ocean Model; [Pacanowski & Griffies, 2000](#)), the POM (Princeton Ocean Model; [Blumberg & Mellor, 1987](#)) and ROMS (Regional Ocean Model System) models for regional seas, FVCOM ([Chen et al., 2006](#)) and GETM ([Burchard & Bolding, 2002](#)) for application in coastal and estuarine environments. Besides these free software codes there exist several licensed commercial softwares like TELEMAC, MIKE, Delft3d. The advantage of a public domain code is that the source code is freely available and can be modified by an experienced user. This is obviously not the case for commercial codes. On the other hand, developers of free codes have no obligation for providing user support which should, obviously, be the case for the commercial ones. However, the lack of a substantial support can be compensated by providing a detailed manual and model documentation to the users.

A special category of the open source codes are the integrated “Community Models” which couple a core physical component with modules for biology, sediments, waves, The construction of such models is a multi-disciplinary task which requires the collaboration of a variety of specialists and consumes a substantial number of years of scientific and programming effort for designing, developing and testing the model code and to validate the model against observational data.

COHERENS (coupled hydrodynamical-ecological model for regional and

shelf seas) is a three-dimensional hydrodynamic multi-purpose model for coastal and shelf seas, which resolves mesoscale to seasonal scale processes. The program has been developed initially over the period of 1990–1998 by a multinational European group as part of the MAST projects PROFILE, NOMADS and COHERENS funded by the European Union. During its development it was applied for studies in the North Sea and regions of fresh water influence (ROFIs) (Huthnance, 1997; Proctor, 1997; Luyten, 1999).

The first version, COHERENS V1¹, has been released in April 2000 as open source code. Besides the source code, the release contains an extensive User Documentation (Luyten *et al.*, 1999) and a series of test cases which can be used to test the installation and as an illustration of the different options available in the program.

The program was written in FORTRAN 77 and has four major components:

1. A physical part with a general module for solving advection-diffusion equations.
2. A simple microbiological module which deals with the dynamics of microplankton, detritus, dissolved inorganic nitrogen and oxygen.
3. An Eulerian sediment module which deals with deposition and resuspension of inorganic as well as organic particles.
4. A component with both an Eulerian and a Lagrangian transport model for contaminant distributions.

Its ease of implementation across a range of computing platforms made the program attractive to groups with a sufficient expertise in modelling and in need of sophisticated model products. Important advantages of the model are its transparency because of its modular structure and its flexibility because of the possibility to select different processes, specific schemes or different types of forcing for a particular application. The program structure allows users to perform process as well as predictive and operational setups without knowledge of the detailed model structure. Since the official release more than 1000 registrations have been recorded on the COHERENS home Web page.

<http://www.mumm.ac.be/coherens>

¹The original version number 8.4 has been replaced by V1 to be compatible with the currently adopted versioning system.

where a publication list can be found with applications of COHERENS.

The work on a fully updated version of COHERENS started in 2003 as part of the EU-FP5 and FP6 projects ODON (2003–2005) and ECOOP (2006–2009). The first objectives were to produce an optimised code so that it can be implemented on parallel machines and to provide user interfaces for model setup independent of the main source code and for user-defined formats for external files representing model forcing. A first version was presented at the 2006 EUROGOOS meeting (Luyten *et al.*, 2006). In response to various user requests new algorithms have been implemented or existing ones have been improved (e.g. drying/wetting scheme, additional types of open boundary conditions, turbulence schemes, baroclinic pressure gradient, astronomical forcing, one-way nesting …). More flexible model grids can be selected using orthogonal curvilinear coordinates in the horizontal and extended σ -coordinates (also known as s -coordinates) in the vertical.

The new version, denoted as COHERENS V2.0, is now available for the scientific community as open source code. Despite the numerous additional implementations the new code only contains a physical component, in contrast to COHERENS V1. In the framework of ongoing projects the development of a fully integrated version is now in progress. This includes in the first place the coupling with a generic biological model, a sediment transport model taking account of different size classes and a bottom boundary layer, a morphological module, a Langrangian tracer module which can also be applied for the dispersion of oil slicks and contaminants in general, and with surface wave models such as WAM and SWAN. Additional improvements are an implicit time integration scheme and schemes allowing to represent inundation over a land topography.

1.2 Description of the program

The characteristics of COHERENS V2.0 are summarised below where a (*) marks a new implementation or a change with respect to COHERENS V1.

1. Model code

- The code has been re-written in FORTRAN 90(*) replacing the FORTRAN 77 used in the previous version. Important advantages are:
 - dynamic allocation of model arrays. This permits a more efficient usage of internal memory and makes it easier to implement a domain decomposition for parallel applications.

- modules replacing the **INCLUDE** statements in **FORTRAN 77** and module routines used to create “internal libraries”.
- derived type variables for storing information of different “entities” (files, model variables, external grid information) in a structured format.
- array instead of element-wise assignment within **DO** loops.
- generic routines.
- Simulations can (optionally) be performed on parallel clusters using the **MPI** (Message Passing Interface) communication library (*). Advantages are a much faster computing speed and the possibility to distribute internal memory over different processes.
- CPP preprocessing for conditional compilation via **#ifdef** statements (*). The method is primarily used for an easy implementation of external libraries. In this way the user can decide to include the **MPI** (for parallel processing) and **netCDF** libraries (for portable data formats) by setting a compiler switch.
- The program is (in principle) compiled with the **UNIX/LINUX** **make** utility. A generic (machine-independent) **Makefile** is provided (*). The program can, in principle, be compiled on a **Windows** platform. This is however not recommended.
- Model setup has been significantly changed with respect to **COHERENS V1**. Definitions of model parameters and input of model forcing data are made by calls to “**usrdef_**” routines (*) which can be considered as a kind of user interfaces. These routines have to be provided by the user. Although this requires some programming skill of the user, model forcing can now be given in any kind of user-defined format and independently from the main source code. An option is foreseen to re-write each kind of forcing into a standard **COHERENS** format, including **netCDF** files (*). Most setup parameters have default values which, for most applications, do not need to be defined explicitly.
- The definitions of all model parameters (including default ones) can optionally be obtained from a Central Input File (CIF) instead of calling **usrdef_** routines (*).
- A total of 84 switches are implemented for the selection of different physical processes (barotropic or baroclinic, turbulence closures, density effects, tidal forcing, nesting . . .), type of model grid, numerical schemes (advection and diffusion, open boundary conditions, surface and bottom fluxes, drying and wetting algorithm,

time-integration, . . .), type of parallel communication, `netCDF` output.

- A series of monitoring utilities is optionally available:
 - The execution of the program can be traced in a “log” file.
 - The program is checked for setup errors. When the program traps an error, the program aborts with an explanatory message. The level of error trapping can be selected by the user.
 - Suspicious values of setup parameters are reported in a “warning” file (*).
 - The program optionally writes a timer report containing the % of time spent by different model compartments (*). The utility can be applied to optimise parallel applications.
- Several simulations can be set up and executed within one run (*).
- Restart times can be defined where the program writes initial conditions for an eventual restart of the program (*).

2. Model grid

- A rectangular or a curvilinear (*) grid can be selected in the horizontal directions.
- The program uses σ -coordinates in the vertical. The grid size in σ -space can either be uniform, non-uniform vertically and non-uniform in both vertical and horizontal directions (*) (making it equivalent to the “so-called” s -coordinate system).
- The model can either be set up on a 3-D grid, in 2-D (depth-averaged) mode (*) or in 1-D (water column) mode.
- One-way sub-grid nesting (*) can be applied for 2-D and 3-D model quantities using an off-line procedure. The advantage is that multiple sub-grids can be simultaneously defined without any restrictions of grid sizes and sub-grid time steps.
- Interpolation of model grid arrays at a different node is performed either with uniform or non-uniform (*) weight factors.

3. Numerics

- The model equations are discretised on an Arakawa C-grid using either Cartesian or spherical coordinates as selected by the user.

- The mode-splitting technique is applied to solve the 2-D and 3-D momentum and continuity equations.
- The same time step is used for 3-D currents and 3-D scalars.
- Two different advection schemes are implemented: a first order upwind and a more accurate TVD (Total Variation Diminishing) scheme. A separate selection can be made for 2-D transports, 3-D currents, 3-D scalars and turbulence transport variables.

4. Physics

- The program runs by default in barotropic mode, i.e. without density effects. Temperature and salinity can be included via separate switches.
- Density effects in the momentum and turbulence equations are included via an equation of state. A correction has been made with respect to COHERENS V1 since the simulated temperature field represents potential temperature instead of the physical “in situ” one.
- The absorption of solar radiation in the upper part of the water column is implemented by an optical module.
- The program incorporates a variety of turbulence schemes ranging from simple algebraic formulations to one- or two-equation schemes (Mellor-Yamada, $k - \varepsilon$). Additional RANS models have been implemented (*).
- On request of the users, additional types of open boundary conditions have been implemented (*), while corrections have been made to some of the existing ones.
- Different formulations for the wind stress and surface heat fluxes are available. Additional schemes have been added (*).
- A drying/wetting algorithm is implemented in Version V2.0. An extended scheme for simulating the flooding of land areas and the flows over obstacles has been added in Version 2.3.
- A module has been added to simulate effects of hydraulic structures, i.e. dry cells, thin dams, weirs and barriers (*).
- A discharge module has been implemented (*).
- A wave-current interaction module has not yet been implemented in COHERENS V2.0.

5. Other compartments

- An extended sediment model has been implemented in Version 2.5.

1.3 User experience

The COHERENS program has been designed such that it can be applied by users with a “low” or a “high” level of model experience. In the former case the program is set up by defining a limited number of parameters (e.g. time steps, a few model switches, attributes for model forcing, ...) whereas default values can be taken for most model parameters. Bathymetry, initial conditions, open boundary and surface forcing conditions are provided by the user via the `usrdef_` model setup routines. To facilitate its use for experienced modellers, the program is supplied in modular form. This allows to replace a module by a user-defined version without affecting the core of the program. It is clear that this practice is not without danger especially when the module is linked with other program units. Changing the default settings of switches and model parameters is easily performed but not always without risk.

For a user it is important to know which specific experience is required to run COHERENS for a particular type of application. There are two basic criteria: programming experience and scientific background. They are further discussed below.

1.3.1 Programming experience

A basic knowledge of FORTRAN 90 is required. As explained in Part **IV** (User manual) a series of FORTRAN source code files need to be created for a user application. Generic example files have been made available to facilitate this. In these files the user only needs to replace or add a number of assignment statements or insert READ instructions for data input. All aspects of model setup are, in principle, covered by the `usrdef_` interfaces. This makes this version more flexible than COHERENS V1, since there is no need to change the main code. However, if it is the intention to implement new schemes or to add new modules, the user must have, besides a good skill in programming with FORTRAN 90 have a good understanding of the general concept, including the parallelisation of the program, and internal structure of the COHERENS code. For this reason Part **III** (Model code) has been written for developers and covers all different aspects of the model code.

Installing and compiling COHERENS V2.0 only requires a limited knowledge of the UNIX/LINUX operating system. For installation and linking

with external libraries the user is referred to the manuals which are provided by the software developers and are freely available through the internet.

1.3.2 Scientific background

The default settings make it easier to run the model without a detailed knowledge of the underlying scientific basis. Only a limited number of default values need to be changed for most applications. Detailed instructions are given in this documentation together with a large number of examples (see Part ??). A more specific scientific background is required if for example the user intends to perform experiments with different turbulence or numerical schemes or with alternative settings of model parameters. It is then recommended to read first the appropriate chapters of Part II (Model Description).

1.4 Contents of the documentation

1.4.1 Structure

This documentation is composed of six parts covering different aspects of the COHERENS program.

I. Introductory manual.

Chapter 2 is a tutorial manual for beginning users. This chapter also presents various free-software for displaying COHERENS results. More detailed instructions for installing, preparing and running a pre-defined test case and a user application are given in Chapter 3.

II. Model description.

The scientific background of COHERENS is described in Chapter 5. Numerical aspects are the subject of Chapter 12. In this chapter the discretized forms of the model equations are written out in full detail. The implementation of hydraulic structures and discharges is discussed in Chapter 11. The sediment model is described in Chapter 7.

III. Description of the model code.

This part is mainly intended for model developers, although some aspects can be useful for beginning users as well. Program conventions and model layout are explained in Chapter 13. Chapter 14 deals with

different aspects of model input and output: purpose of the files connected to the program (monitoring, forcing, model output defined by the user), implementation in the model, structure of a file in standard COHERENS format. Chapter 15 explains how arrays are defined on the model C-grid and the types, principles and implementation of interpolation (internal on the model grid, from an external grid to the model grid, from the model to external data locations as used in the nesting procedures). Chapter 16 is devoted to the parallelisation of the code (general principles, domain decomposition, array halos, implementation of process communication with the MPI library, index mapping). Chapter 17 provides a listing of all source code files with their purpose and illustrates the general structure of the code with flow diagrams.

IV. User manual.

This part is undoubtedly the most important one of the documentation. The meaning and contents of each `usrdef_` routine are discussed in the following chapters:

- Chapter 18: general principles of model setup, listing of all `usrdef_` routines and files
- Chapter ???: switches, model parameters, forcing attributes, setup of monitoring and the parallel mode
- Chapter 20: model grid, bathymetry, initial conditions
- Chapter 21: open boundary conditions
- Chapter 26: structures and discharges
- Chapter ???: surface boundary conditions and nesting
- Chapter 27: different kinds of user output
- Chapter ???: sediment model

V. Test cases.

The test cases experiments which are provided with the distribution of the source code, are presented. The aim is to test the portability of the code, to illustrate how model setup is performed and to show different types of applications. The results are briefly discussed and illustrated with figures and tables which may be used by the user for comparison.

- Chapter ???: purpose of the test cases, CPU time table obtained on different computing platforms
- Chapter ???: numerical tests for advection schemes

- Chapter ??: turbulence schemes and heat flux formulations
- Chapter ??: density fronts in a channel, river plumes
- Chapter ??: shelf sea applications
- Chapter ??: hydraulic structures and discharges
- Chapter ??: tests for the sediment model

VI. Reference manual.

This part provides a listing of all program routines with their syntax and purpose and of all program variables with a global scope in the program.

- Chapter ??: external routines
- Chapter ??: module routines which are mainly used by the program as internal libraries
- Chapter ??: `usrdef_` routines
- Chapter ??: program variables defined in the FORTRAN 90 modules
- Chapter ??: sediment routines (external, module, `usrdef_`) and sediment program variables

1.4.2 Suggestions for reading

The documentation is written, as much as possible, in a self-contained form. Appropriate links are made, where necessary, to chapters and sections where a specific topic is explained in more detail.

Beginning users whose prime intentions are to perform simple simulations with the model and have less interest in learning all the features of COHERENS, are not obliged to read this voluminous documentation as a whole, but may proceed as follows:

1. Read Part I.
2. Select a few test cases from Part ?? to test the portability of the model. Have a look at the source code files used to set up the selected test cases.
3. Read those sections of Part IV which are of interest for the intended application.
4. Create the source files for the application using the setup of an analogous test case. For more information about the model setup or the name of certain variables for model output consult the reference manual.

On the other hand, if the user wants to learn about the full model's capacities or has the intention to extend the code with a new implementation it is recommended to read this manual in more detail, especially Part **III** and Chapter **12** about the discretisation methods used in the program.

Conventions used in the document

The text is typesetted in L^AT_EX2e. The following numbering conventions are adopted:

- Chapters and pages are numbered continuously over all Parts.
- Sections are numbered in a two-number format (Chapter-Section).
- Subsections are numbered in a three-number format (Chapter-Section-Subsection).
- Equations, figures and tables are numbered in a two-number format (Chapter-Equation/Figure/Table).

Specific names and keywords are outlined in different L^AT_EX fonts:

- **sans serif** style for the **FORTRAN** names of program variables and specific keywords
- **bold** for the names of directories
- *slanted* style for the names of files
- ***emphasised bold*** for the names of test cases
- **type-writer** font for sections of model code and UNIX command names

Chapter 2

Getting Started

2.1 Introduction

In this tutorial manual, an introduction is given how to work with COHERENS. We present two COHERENS test cases (river and bohai) and show how to compile the code, run the test case, view the results with various free software visualisation tools and show how model setup is arranged and can be modified.

The objective of this chapter is to give a first introduction to COHERENS for a beginning user. A more extended description with more technical details about installation and compilation of the code are found in the User Documentation.

2.1.1 General requirements

We will make the following assumptions on your system and the installation of COHERENS.

- You use COHERENS V2.4 or higher. Older versions of COHERENS have a slightly different syntax.
- You have managed to install COHERENS correctly and to set the compiler options correctly in *compilers.cmp* (see below).
- You have a working installation of netCDF on your computer or work station. Note that if you install a compiler, you have to do this before you install netCDF. Otherwise, the netCDF installation will not generate the correct binary files for your compiler.

- You have some software for doing post processing. In this manual, we will use the free software packages **NcView**, **Ferret** and **Octave**. However, it is very well possible to use other (free or commercial) post-processing software such as **IDL** or **Matlab**, as long as they can read the **netCDF** format.

Some other external libraries, which you can use with **COHERENS** (such as **MPI**), will not be used in this introduction. The used software and libraries can be found at:

- **MPI**: <http://www.mcs.anl.gov/research/projects/mpich2/>
- **netCDF**: <http://www.unidata.ucar.edu/software/netcdf/>
- **NcView**: http://meteora.ucsd.edu/~pierce/ncview_home_page.html
- **Ferret**: <http://ferret.wrc.noaa.gov/Ferret/>
- **Octave**: <http://www.gnu.org/software/octave>

2.1.2 Short Linux introduction

This section will make you familiar with working under **Linux**. If you are already familiar with **Linux**, you can skip this section without problem.

The first thing to do is to open a terminal in which we can execute the commands we need. Before we continue, there are three important things to know when working on the **Linux** command line:

1. Linux commands and file names are **case sensitive**. Thus **Run** is not the same as **run**.
2. Linux can help you finish commands (especially file names and directories) when you press on the tab key. Try using this as often as possible.
3. In Linux, files and folders are separated by a forward slash (/) not a backslash (\) that is used in **Windows**. Using the wrong slash can give very unexpected results!

The following commands are very useful when working with **Linux**.

2.1.2.1 getting help

If you do not know which command to use, use `apropos`. It will tell you which commands to use. For example, if you want to know how to copy a file, type

```
apropos copy
```

This gives you many results, including the one you need, which is `cp`. In order to understand how to use this command, use `man`. Type

```
man cp
```

2.1.2.2 working with files

To see which files are in a directory use `ls` or `ls -l1`. The latter gives more information. Try them both

```
ls  
ls -l
```

Aliases for some commands may be defined on your computer. A common alias is `ll` which is the same as `ls -l`. The command for copying a file is

```
cp sourcefile destination
```

The command for moving a file

```
mv sourcefile destination
```

The command for deleting a file is

```
rm filetodelete
```

In case you want to use any of these commands on directories, you must add `-r`

```
rm -r directorytodelete
```

In order to know how much free space there is on the hard disk, type

```
df -h
```

Finally it is important to know that you can use the `|` symbol to send output from one program to another program. For example, you can use the command `less` to view the output of the previous command

```
df -h | less
```

Exit less by typing the letter `q`.

¹note that ‘l’ is the letter 1 and not the number ‘1’.

2.2 Running a test case

2.2.1 Installing a COHERENS test case

First, you will need to go to the directory where the COHERENS source files are installed. We will assume here that we have them installed on the directory `~/V2.5` where `~` stands for your home directory². When installing a COHERENS test case, such as the `river` test, we always start by making a new folder in which the results are placed. In order to do so, type the following commands

```
cd
mkdir river
cd river
```

We follow by making a link to the COHERENS directory, in which the source code and installation scripts are placed. The name of the link **must** be **COHERENS**. We can make the link by typing the following command

```
ln -s ../V2.5/ COHERENS
```

Here, the two arguments are first the location to which the link points and secondly the name of the link. In this command, the two dots (..) are used to refer to the directory that is above the current directory. In case the COHERENS source files are located in another directory, you must adapt the first argument (`../V2.5`) such that it is the location of the directory containing your COHERENS files.

The first test case that we are going to do is called `river`. It describes the evolution of an estuarine salinity front advected by a tidal current and the corresponding estuarine circulation in an open non-rotating channel (as described in the User Documentation). We run the installation script `install_test` in the COHERENS directory, which copies the necessary files to our directory

```
COHERENS/install_test -t river
```

In this command, the argument of the script is the name of the test cases³. There are many test cases delivered with COHERENS. Their model setups can be found in the directory `COHERENS/setup`.

In order to examine the files that are generated, we use the command `ls`. Type

²The examples below can be run in the same way with other versions of COHERENS as long as they are not older than 2.4.

³The `-t` option is new from COHERENS V2.4 on. In older versions, this option must not be used.

```
ls
```

Now, you should see the following output

BCOMPS	deffigs.txt	postparsD	SETUP
BSOURCE	defposts	river.txt	SOURCE
cifruns	defruns	Run	SSOURCE
COHERENS	files.vis	Run_hpd_par	Usrdef_Harmonic_Analysis.f90
coherensflags.cmp	Makefile	Run_hpd_ser	Usrdef_Model.f90
COMPS	postparsA	Run_vic	Usrdef_Output.f90
con_sub_river	postparsB	SCOMPS	Usrdef_Time_Series.f90
DATA	postparsC	SCR	

The files have different meanings. They are not all necessary for our purposes.

We will now examine the most important of these files.

The names in capitals are symbolic links to important locations for COHERENS. For example, the link **SOURCE** points to the FORTRAN source files of the model. The link **COMPS** points to the files needed for the compilation. The model setup is given in the four files whose name starts with *Usrdef*. For our purposes, the two most important ones are *Usrdef_Model.f90*, which contains the model input and *Usrdef_Time_Series.f90*, which contains information on the data that should be exported as time series. We will discuss this later in the tutorial.

After installing the test case, we are now ready to compile the model. This will be explained in the next section.

2.2.2 Compiling COHERENS

Before we compile the model, we have a look at the files that determine the compilation process. There are three. First there is *Makefile*, which contains instructions about the order in which the model is compiled. Second, there is the file *coherensflags.cmp*. This file contains some statements, which determine which options are used in the compilation of the model. Thus you can make the program a little bit different by changing the options in this file. We will edit this file. An easy to use text editor with a graphical user interface in Linux is called **gedit**. Open the file by typing

```
gedit coherensflags.cmp
```

Note that, if **gedit** is not installed on your system, or if you prefer other text editors, such as **vi**, **nano** or **emacs**, you can of course use these instead of **gedit**.

Now, we are going to add two options. Change the line that starts with **CPPDFLAGS** = into the following

```
CPPFLAGS = -DALLOC -DCDF
```

The meaning of these two switches is the following. The first one (-DALLOC) changes the location in the memory that is used in **COHERENS** (more variables are placed on the heap instead of on the stack). The first advantage of using this option is that there is a much lower risk that the model crashes, because the amount of memory is too low. The reason is that there is much more memory available on the heap than on the stack. The second advantage is that less CPU memory is used by the model. The eventual disadvantage of using this option is that the model might become slightly slower. The second option (-DCDF) enables the **netCDF** module in **COHERENS**. Without this option, it is not possible to use **netCDF** input and output. If you do not set this option and you are trying to generate **netCDF** output, **COHERENS** will run normally, but the files are not generated! Another option that can be set in this way and which will not further discussed in this tutorial, is the enabling of parallel computations by setting the switch (-DMPI).

Further, you need to select the paths of the **netCDF** installation (usually, this is in **/usr/local**). You can do this by removing the comments (indicated by a hash #) in the lines related to **netCDF**, and changing (if necessary) the location of your **netCDF** files. The result should look something like this

```
# netCDF directory path
NETCDF_PATH = /usr/local

# netCDF library file
NETCDF_LIB_FILE = netcdf

# netCDF include options
FCFLAGS_NETCDF = -I$(NETCDF_PATH)/include

# netCDF library options
FLIBS_NETCDF = -L$(NETCDF_PATH)/lib -l$(NETCDF_LIB_FILE)
```

Now save and close the file. The third file that is important for the course is **compilers.cmp** which is located in the folder **COHERENS/code/physics/comps** given by the symbolic link **COMPS**. We will open this file in **gedit** to see its contents

```
gedit COMPS/compilers.cmp
```

In this file, there is a list of “targets”. Each target defines a set of options for different compilers. If one these compilers has been installed on your machine, there is, in principle, no need to change this file. We will use here

the Intel Fortran compiler (`ifort`), or the GNU Fortran compiler (`gfortran`). We will look up the settings for this compiler in `compilers.cmp`. In this file, you should see the following statements (they may be different because of local settings on your computer)

```
# Linux gfortran (GNU Fortran)
linux-gfort:
$(MAKE) $(EXEFILE) "FC=gfortran" "FCOPTS= -O3" "FCDEFS=" "FCDEBUG=" \
"CPP=" "CPPF=cpp" "CPPOPTS==traditional-cpp" "CPPDEFS=$(CPPFLAGS)"

#Intel fortran compiler v 9 without MPI, dynamically linked
linux-iforts:
$(MAKE) $(EXEFILE) "FC=ifort" "FCOPTS==cpp1 -i-dynamic" \
"FCDEFS=$(CPPFLAGS)" "FCDEBUG=" \
"FCIFLAGS_COMP==Wl,-rpath=/usr/local/intel/lib" \
"CPP=" "CPPF=@cp" "CPPOPTS=" "CPPDEFS="
```

The first word before the colon (e.g. `linux-iforts`) is the name of the compiler setting, also called the “target”. There are two settings for the Intel compiler in this file. `linux-iforts` is the normal option for using the Intel Fortran Compiler. This is the recommended setting when performing simulations. In this tutorial, we will make a few changes in this file in order to speed up the compilation process. This is done by disabling the optimization that is performed by the compiler. This optimization takes a lot of time. On our machine, the compilation with optimization takes about 25 minutes. Without optimization, it takes only one minute. However, the optimized version executes much faster (up to 5 times faster). Therefore, for real simulations, one should always use this optimization.

If you are using the `gfortran` compiler, you can switch off the optimization by setting (take care with the difference between, the capital O and the number 0!)

```
"FCOPTS= -O0"
```

For the Intel Fortran compiler, you can disable the optimization by setting

```
"FCDEBUG= -O0"
```

In order to compile the code, close `gedit` and type (for `gfortran`)

```
make linux-gfort
```

In order to compile the model using the Intel Compiler, type

```
make linux-iforts
```

This command will print some information on the screen mainly showing, which files were compiled. It may also show some warnings or errors. In case, the compilation was successful, the last output line should be something like

```
make[1]: Leaving directory '/home/coherens/rivertest'
```

The compilation generated many additional files. Use the command `ls` in order to examine the files that are present in the working directory.

We are now ready to run the simulation.

2.2.3 Running COHERENS

In COHERENS, the different runs that are done during a simulation are defined in the file `defruns`, which is located in the working directory. Open this file in `gedit` in order to examine the runs that are being performed

```
gedit defruns
```

In this file, all the runs are defined, with the following syntax

```
Run name,CIF options, CIF file name
```

Note that the last two inputs (CIF option and CIF file name) are optional, which means that you can leave them empty. However, you must **always** put the two commas on this line. Otherwise, COHERENS will generate an error. Comments can be set in the `defruns` file by adding an exclamation mark (!). There are two options for CIF files. In order to read a CIF file, one must add the letter `R` (note that this option is case sensitive). In order to write a CIF file, one must add the letter `W`. We will change the `defruns` file. We do so by adding `W` after the first comma on the first two lines and inserting an exclamation mark at the beginning of the next lines. The result looks like

```
river0A,W,
river1A,W,
!river0B,,
!river1B,,
!river0C,,
!river1C,,
!river0D,,
!river1D,,
```

Note that in the test case ***river***, each run actually exists of two different runs with the numbers 0 and 1. In the run with number zero, a spin up calculation is done without salinity in order to determine the initial conditions for the actual run. In the run with number one, the calculation of the estuarine circulation is performed.

After closing **gedit**, we can run the model with the command:

```
./Run &
```

This command will run the model. The ampersand (&) makes the model run in the background, which means that we can keep working on the command line. We can for example use the command **ls** to see which files are being made by **COHERENS**. We can check whether **COHERENS** is still running with the command **top**. Type:

```
top
```

This will show a list of the processes that are occurring on the computer and how much processor time they are using. You can close this program by typing the letter **q** (from quit).

We can also examine how far the run has advanced using the command **grep** which searches a text file for the occurrence of a word given by the user. We apply it here to the *runlog* file, which is a text file **COHERENS** produces and in which the time step is written. We can search for these time steps in the file, and then send it to another program, called **tail** which returns only the last lines of data. For sending information from one program to the next (this is called a pipe) we use the vertical line (|). Thus in order to find the last occurrence of the string “2003” (the calculations in this file have a start date and end date in 2003) in the *runlog* file, we use the command:

```
grep 2003 river0A.runlogA | tail -1
```

You can also use this command without tail, in order to get all occurrences of the string “2003”:

```
grep 2003 river0A.runlogA
```

After the model has finished, something like this will be written to the terminal

```
Main program terminated
```

```
real    0m19.678s
user    0m18.490s
sys     0m1.154s
```

Let's have a look at the files that are generated by COHERENS. All these files have a name that is generated from the name of the run (defined in the file *defruns*).

```
ls river*
```

This should give the following result

```
river0A.cifmodA
river0A.inilogA
river0A.runlogA
river0A.timingA
river0A.warlogA
river1A.cifmodA
river1A.inilogA
river1A.runlogA
river1A.timingA
river1A.warlogA
riverA_1.resid3I
riverA_1.resid3N
riverA_1.tsout2I
riverA_1.tsout2N
riverA_1.tsout3I
riverA_1.tsout3N
riverA_2.tsout0A
riverA_2.tsout0I
riverA.2uvobc1U
riverA.modgrdU
riverA.phsicsU
river.txt
```

These files have the following meanings

- Files ending at *.inilogA* are the initialization log files. The name of each subroutine that is called during the initialization is written to this file. It is mainly important for debugging of a model setup and code developers.
- Files with suffix *.runlogA* are log files writing information during the actual run of the model. This files may become quite large for long runs. We have already seen it before when we used this file to examine the progress of the simulations. Otherwise, it is also more important

for code developers than for model users. However, it sometimes occurs that an error is written to a log file, in addition to the *errlog* file discussed below.

- The files ending with *.timingA* contain information on the time it took for the model to run. Open the file *river0A.timingA* in **gedit** and examine the results. This file shows how long the simulation took and it also shows the percentages of the time that different parts of the calculation took (such as 2D calculation, 3D calculation, input and output, ...).
- Files ending at *.warlogA* are files with warnings about settings that were automatically changed by **COHERENS**. Before the start of the simulations, **COHERENS** checks whether some parameters in the model have appropriate values, and if not, **COHERENS** resets them automatically. When **COHERENS** changes these values it writes a warning in this file. It may be useful after each simulation to inspect the *.warlogA* files! They can give information on some model settings that were not intended by the user and they may explain unexpected results in the simulations. Open *river0A.warlogA* in **gedit** and examine the warnings. In this case, there are no important changes made to the model.
- A file that should **not** be present is the *.errlogA* file. In this file, error messages are written and the execution of the program stops. Examples of this are incompatible options which can not be automatically reset. At the end of the simulation, **COHERENS** automatically removes this file. This means that you know for certain that something went wrong with the simulations if this file is present at the end of the simulation. You should then examine this file and fix the problem. Make sure that there is no *errlog* file by typing

```
ls *.errlogA
```

If an error log file exists, but is empty, you must check the *runlog* or *inilog* files to see firstly whether some error message was written to one of these files. If no error message has been written, the program crashed by a type of error not detected by **COHERENS**.

- The file with suffix *.cifmodA* contains an automatically generated central input file (CIF). Open the file *river1A.cifmodA* in **gedit**. It contains some lists of the different parameters in the model. Each of these lists is separated from the others with a hash sign (#). Comments can

be given with an exclamation mark (!). The CIF file will be discussed in more detail in the next chapter.

- There are files with the results of the simulation. Files whose names contain the string *.tsout* and end with the letter **N** contain time series of some variables. For this simulation, there are three of them, *riverA_1.tsout2N*, *riverA_1.tsout3N* and *riverA_2.tsout0A*. The number after *tsout* indicates the dimension which can be 0, 1 or 2. As an example, *riverA_1.tsout3N* contains three-dimensional data. The last letter of the extension indicates the type of data. The letter **N** stands for netCDF. Other possibilities are **A** for ASCII and **U** for binary. Files ending at **I** are ASCII files containing information about the output data, but do not contain the data themselves. Files with the string *resid* in their name have been generated for harmonic analysis. Their meaning is similar to the *tsout* files discussed above.
- The file whose names contain the string *2uvobc*, *modgrd* and *phsics* are created by COHERENS for internal use only and are of no interest for the present discussion.
- Finally, the file with the suffix *.txt* is a small ASCII file describing the setups of the different experiments which can be conducted with the test case *river*.

Because the netCDF output contains binary files, we need a special tool for examining the data. It is called **ncdump**. In order to examine the results, use the following command

```
ncdump riverT_1.tsout2N | less
```

Here, the command **less** is used as a viewer. You can close this program by typing **q**. Using **ncdump**, you can see which variables are in the file and which values they have. However, it is normally more instructive to visualize the data. We will do that in the next sections.

An alternative of the **less** command is to send the ASCII output to another data file. For example, the three dimensional output can be examined by typing

```
ncdump riverT_1.tsout3N >> outdat
gedit outdat
```

Compare the variables that are in this file with those in the two-dimensional file. What are the differences in the dimensions and the variables? Also notice the meaning of the different variables by observing the **long_name** and **units** attributes.

2.3 Post-processing the results

2.3.1 Visualising with Ncview

Ncview is a tool for visualizing netCDF data files. It is very easy to use, because of its graphical user interface. However, its possibilities are limited. In order to start this program for a certain data file, type the following command

```
ncview riverA_1.tsout2N
```

When using NcView, you always **must** give the filename of the data that is plotted in the command to start the graphical user interface. An example of the user interface in NcView is given in Figure 2.1. The first thing to do when working with NcView is selecting the variable to plot. This can be done by clicking on the name of the variable you want to select. We start by plotting the sea surface elevation. In order to do so, click on the button with **zeta**, which is the name used on COHERENS for the sea surface elevation. In order to get more information on a variable, click on the button with the question mark **?**.

You should now see a plot of the sea surface elevation, as function of the **x** location (on the x-axis) and time (on the y-axis). We can change this by clicking on the **Axes** button. If you do so, a dialog box will appear that lets you select the axes to plot. In this case, there are only two possible options, the time and the x-dimension. Hence, it is not necessary to change the axes. Close the dialog box by clicking on **cancel**. To make the plot larger, click on the button that says **M xn** where **n** is the magnification scale. As you can see, the image is magnified, and the name of this button is changed to the new magnification. Press this button again to make the plot even larger.

When can change the colors in the graph by clicking the leftmost button (its name is one of the color maps, probably **3gauss**). Click this button and see how the colors change. Try out the different colormaps, and select one you find useful to work with. A useful selection may be **ssec**. The ranges of the colorbar are set by the button with **Range**. Click on it in order to get a dialog box, in which you can set the limits of the colorbar. You can also choose to highlight some special ranges by clicking on the button **linear**. The name of the button will change in **low** and the resolution in the color bar will increase for low values of the water level. By clicking it again, it will say **Hi** and have more resolution for high values of the water level. The button which says **Bi-lin** will influence the way in which the colors are interpolated (default is bi-linear interpolation). By clicking it, it will say **Repl**, which gives nearest neighbor interpolation. The latter is less visually appealing, but is

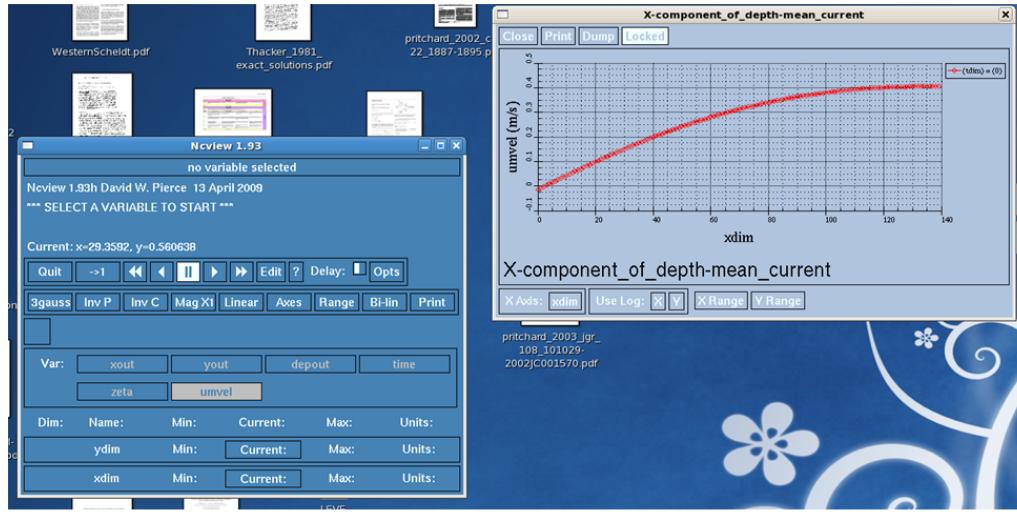


Figure 2.1: Screenshot of using NcView

is sometimes useful near the edges. Finally, we are going to make some files of the data. This can be done by clicking on the button **Print**. A dialog box will appear. Click on the button **File** in order to make a file with the data and click on **OK**. Use **nautilus** or **konqueror** (the Linux programs that do the same as windows explorer) to look up the file. Click on it to see it. Now click on **umvel** to visualize the depth averaged velocities (in the x-direction) and also make an output file with these data. Finally, close NcView by clicking on **Quit**.

We continue by visualizing the 3-D data. In order to open the 3-D file, type

```
ncview riverA_1.tsout3N
```

In this file, three-dimensional data are stored. In fact, because the river test case is two-dimensional (there is no information in the y-direction), only the data in the x-z direction can be displayed. We will start by visualizing the salinity. Click on **sal** to visualize the salinity. Now, you will see the initial

salinity profile in the x - z plane. We can go to a different moment in time by clicking the arrow to the right. By clicking the two arrows to the right, an animation is shown. You can change the speed of the animation by sliding the slider to the right of the text `delay`. Watch the animation and observe how the salinity intrusion develops.

Close `NcView`. We will now continue to make visualizations in `Ferret`.

2.3.2 visualising with Ferret

`Ferret` is a more advanced tool for displaying graphics, which has much more possibilities than `NcView`. However, it is somewhat more difficult to use than `NcView`, since options are given through the command line. Nevertheless, no real program is necessary to view output, which is different from for example `Matlab`.

In order to visualize the data, we must first start `Ferret` by typing

```
ferret
```

Now we are inside `Ferret`. We can see this, because the command prompt has changed in `yes?`. The first thing to do is load the data, with the `use` command. Type

```
use riverA_1.tsout2N
```

This will load the file `riverT_1.tsout2N`, with the two-dimensional data. Now we inspect the variables that are in that file, by typing

```
show data
```

You should now see a list with the variables that were saved in the file `riverA_1.tsout2N`. We can make contour plots of these data (x and time axis) by typing

```
contour zeta
contour umvel
```

We can also make a graph of the velocity at one location or at one moment in time by typing respectively

```
plot umvel[x=50]
plot umvel[t=200]
```

In this command, the part between the square brackets is used to tell `Ferret` which data has to be plotted.

It is also possible to plot various of these data together by typing

```
plot umvel[x=50], zeta[x=50]
plot umvel[x=50], umvel[x=100]
```

Saving files is somewhat cumbersome in **Ferret**. The first thing you need to do is tell **Ferret** to save every graph it makes as a meta file, which is special a text file containing the data of the plot. You can do this by typing

```
set mode metafile
```

Now make the same contour plots as you did before. When you do not want to plot any more files, you can type

```
cancel mode metafile
```

Now close **Ferret** by typing

```
exit
```

The metafiles that are made by **Ferret** are text files, with the name *metafile.plt.nr*, in which *nr* is the number of the file. In order to transform them to graphical files, which you can use in reports, you have to use the command **Fprint**

```
Fprint -o outputfile00.ps -p portrait metafile*.plt
Fprint -o outputfile01.ps -p portrait metafile*.plt.^01^
```

Note that we used an asterisk (*) in this command, because **Fprint** changes the filename every time it is invoked. Note that **Fprint** prints to a file, so in order to use it, a default printer must be selected.

Open **Ferret** again to visualize the three dimensional data

```
ferret
```

We first load the three dimensional datafile and see which data it contains

```
use riverA_1.tsout3N
show data
```

In this file, we can make a vector plot of the velocity field at time step 20 using (an example is given in Figure 2.2)

```
vector uvel[t=20], wphys[t=20]
```

We can also make a contour plot of the salinity at this moment in time

```
contour sal[t=20]
```

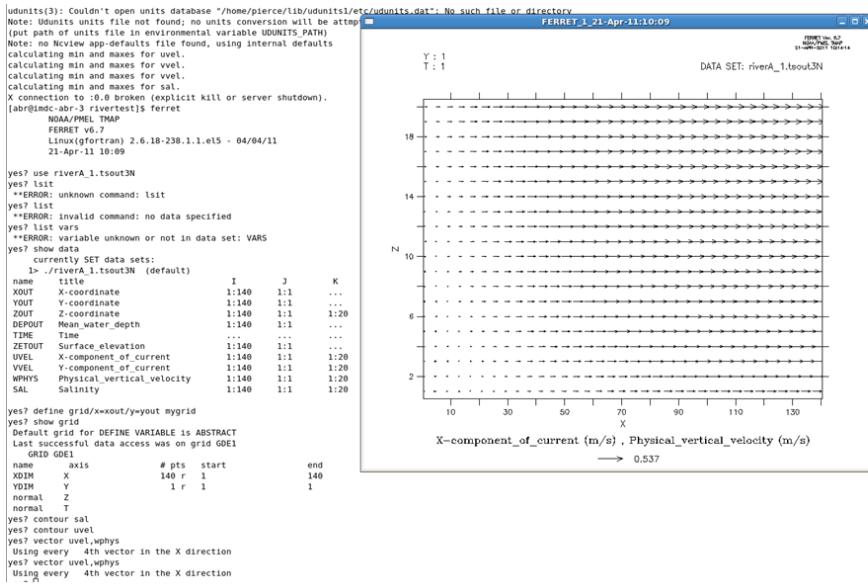


Figure 2.2: Screenshot of using Ferret

If we want to plot the vectors and contours together, we must add the overlay option

```
vector/overlay uvel[t=20],wphys[t=20]
```

Once again, we can plot time series or profiles, but now we need to provide two axis locations between brackets

```
plot sal[z=1,t=20]
```

Finally, we make an animation using the repeat statement of the first 50 time steps. For the animation of the salinity contours, type

```
repeat/t=1:50/animate/loop=1 (contour sal)
```

For the animation of the flow vectors, type

```
repeat/t=1:50/animate/loop=1 (vector uvel,wphys)
```

2.3.3 visualising with Octave and Matlab

Now we are going to run a second test case, which is called *bohai*. This test case is a simulation of the tides in the Bohai Sea (northern part of the Yellow Sea, see Figure 2.3). The commands are the same as for the previous *river* test. Take care of the following



Figure 2.3: Map of the Bohai sea

- Edit the `defruns` file by commenting all lines (i.e. inserting a '!' in the first column) except the first one such that only the first run is made. The name of this run is ***bohaiA***. Note that, contrary to the ***river*** there is no spin up run.
- The date of the simulation, which you need to check the advance of the calculations is in the year 1999 (look this up in the file `Usr-def_Model.f90`). Note that running this test takes about 10–15 minutes depending on the capacity of your machine.
- When visualizing the data with **Ferret**, you must define the `x` and `y` data to make time series. Because this case is two-dimensional, there is only a `tsout2N` file.

We are using the file `bohai_1.1amplt2N` with the harmonically analysed surface M_2 -amplitudes as example for making visualisation with **Octave** or **Matlab**.

1. Download the **octcdf** package, for **Ubuntu** this is rather straightforward using

```
sudo apt-get install octave-octcdf
```

For other platforms such as **Windows** a useful wiki might be: http://modb.oce.ulg.ac.be/mediawiki/index.php/NetCDF_toolbox_for_Octave

2. In this package the `ncdump` function is included, which provides information about all the contents of the `netCDF` file. In our example file

we see that there are 3 dimensions and 7 variables. The variable we are interested in is `zeta`, we are also interested in the dimensions `xdim` (longitude), `ydim` (latitude) and time (depth is constant in this example so we disregard it).

3. To load a variable to the `Octave` environment, firstly check whether the file `ncparsen.m` has been copied from the `scr` to your working directory and type in the `qtOctave` terminal

```
[locallat] = ncparsen('bohaiA_1.1amplt2N','xout')
[locallon] = ncparsen('bohaiA_1.1amplt2N','yout')
[localzeta] = ncparsen('bohaiA_1.1amplt2N','zeta')
```

to load the spatial coordinate arrays `xout`, `yout` and the elevation data `zeta`.

4. In Matlab (R2008a and higher), you can load variables from the `netCDF` file using the `ncread` function

```
[locallat] = ncread('bohaiA_1.1amplt2N','xout')
[locallon] = ncread('bohaiA_1.1amplt2N','yout')
[localzeta] = ncread('bohaiA_1.1amplt2N','zeta')
```

5. To let your graphs appear in a workable screen format, set the display to the correct environment (only for `Octave`)

```
setenv("GNUTERM", "wxt")
```

6. Now you can make your first graph

```
zetafig = localzeta(:,:,1)
contourf(localxout,localyout,zetafig)
```

7. Add a colorbar by typing

```
colorbar
```

8. Put a title and axis legends

```
title("M2-amplitude","fontsize",20,"fontname","Arial_black")
xlabel("longitude","fontsize",12,"fontname","Arial")
ylabel("latitude","fontsize",12,"fontname","Arial")
```

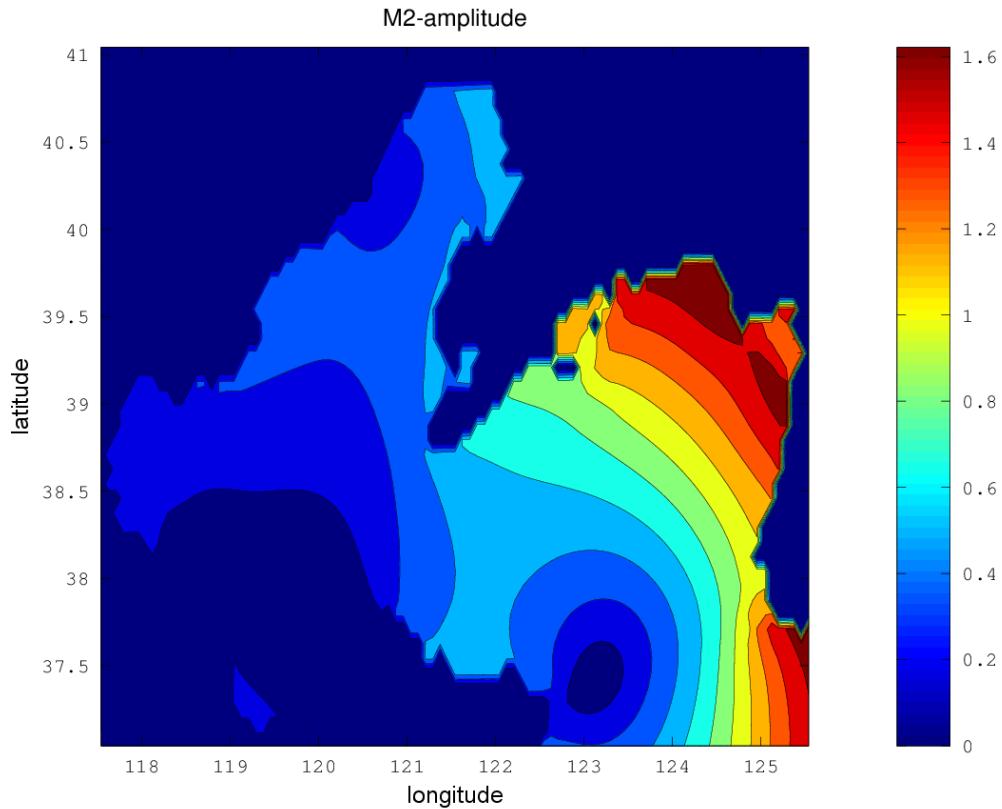


Figure 2.4: Example figure generated with Octave

9. To export the file

```
print -dpng bohai_M2amp.png
```

The result is shown in Figure 2.4.

10. To improve pixel size

```
print -dpng "-S800,800" "bohai_M2amp.png"
```

11. For more information on a topic type `help topic`, for example `help print`.

2.4 Modifying model setup

Two methods are available to adapt the model settings. In the first one the model parameters are changed in the CIF. The second one consists in

editing the *Usrdef_* FORTRAN files. The two methods are described in the subsections below.

2.4.1 Modifying model setup via CIF

2.4.1.1 short CIF introduction

The central input file (CIF) is a functionality introduced in COHERENS V2.1 aimed to define or re-define parameters for model setup without having to do a recompilation. More information on the format of the CIF file can be found in the User Documentation. In addition, all forcing data can be provided in a standard COHERENS format. These formats can be read by the program without the need to change the model setup code (and recompile the program) in the *Usrdef_* files.

2.4.1.2 tutorial *river* test case

The following procedure needs to be used to install the tutorial version of the *river* test case

1. Select a working directory, e.g.

```
cd /home/coherens/riverCIF
```

2. Create a link with the COHERENS root directory (a path name can be defined instead of $\sim/\mathbf{V2.5}$), e.g.

```
ln -s ../V2.5 COHERENS
```

3. Install the tutorial version of the *river* test case

```
COHERENS/install_test -t tutorial/river
```

4. Adapt your settings by changing *coherensflags.cmp*.

5. Compile (e.g for Intel Fortran)

```
make linux-iforts
```

The following procedure is recommended to run the river test case with the CIF functionality

1. Open the file *defruns*

```
river0T,,  
river1T,,
```

which shows that no CIF will be read or produced.

2. Run the test case

```
./Run
```

3. The simulation has now created the following input files in standard COHERENS format:

- *riverT.modgrdN*: model grid arrays
- *riverT.2uvobc1A*: open boundary conditions
- *riverT.phsfinN*: initial conditions produced by the initialisation run for the final one

4. Run the test case again now using the CIF functionality

```
cp cifruns defruns
```

Open the file *defruns*

```
river0T,R,river0T.CifModA  
river1T,R,river1T.CifModA
```

5. You may open the CIF files to see how information is passed to COHERENS.

2.4.1.3 changing model setup through the CIF

Now, we are going to change the model by adapting the CIF file and the forcing files. We will do the following

1. Change the simulation time.
2. Adjust the time step.
3. Modify settings (e.g. roughness length).
4. Change the form of the output file (e.g. from netCDF to ASCII).
5. Modify the grid resolution (number of rows and columns, grid size).
6. Modify the open boundary conditions.

Changing simulation time The first modification is the modification of the simulation time by editing the CIF file

1. Change the *defruns* file.

```
gedit defruns
```

We will switch off the first of the two simulations for the moment, and only do changes in the second file

```
!river0T,R,river0T.CifModA
river1T,R,river1T.CifModA
```

2. Open and edit the CIF.

```
gedit river1T.CifModA
```

Set the simulation time to 1 day by changing the line

```
CENDDATETIME = 2003/01/06;00:00:00:000
```

into

```
CENDDATETIME = 2003/01/04;00:00:00:000
```

The previous re-setting can also be made by inserting the comment character ! in the first column of the line with the old definition

```
!CENDDATETIME = 2003/01/06;00:00:00:000
CENDDATETIME = 2003/01/04;00:00:00:000
```

3. Run the model with the new simulation time.

```
./Run
```

Note that the time steps for the output data are adapted automatically. This is seen by opening the *warlog* file

```
gedit river1T.warlogA
```

This will show a warning, that the last time step for output is equal to the last time step in the model. However, if you would have increased the calculation time, you would have had to adapt the number of time steps that is outputted manually

```
WARNING: value of integer component %tlims(2) and element 1
of array tsrgpars is set from 8640 to 2880
```

Check in the output data, that you now have indeed only one day of output data. You can do this easily using `ncdump` by typing (use `man ncdump` to look up what `-v` means)

```
ncdump -v time riverT_1.tsout2N | less
```

Modification of the time step This exercise is focused on the modification of the time step by editing the CIF file, the instructions are given below

1. Open and edit the CIF.

```
gedit river1T.CifModA
```

2. Set the 2-D time step to 15 seconds. COHERENS also has a 3-D time step for barotropic calculations, which is given as the number of 2-D time steps that pass before a 3-D step is done. This is given by the variable `IC3D`. In order to leave the 3-D time step unchanged, we modify this value as well by changing

```
DELT2D = 30.
IC3D = 10
```

into

```
DELT2D = 15.
IC3D = 20
```

3. Run the model with the new time step.

```
./Run
```

Visualize the results.

Checking of errors in the CIF This exercise we will make some errors in the CIF file, such that an error message is generated by COHERENS. In this way, you learn to find the error messages and solve them

1. Open and edit the CIF.

```
gedit river1T.CIF
```

2. Create an error in the CIF by re-editing the line

```
IOPT_GRID_NODIM = 3
```

to

```
IOPT_GRID_NODIM
```

3. Save the file and run the model

```
./Run
```

An error message will be generated in the *river1T.errlogA* file. Open this file and verify the error message. It will look like this

```
Error occurred on line 29 of CIF file river1T.CifModA
Variable name is not defined
A total of 1 errors occurred in read_cif_params
Error type 7 : Invalid initial values for model parameters or arrays
PROGRAM TERMINATED ABNORMALLY
```

This message tells you that the syntax in the CIF is not correct. Now we try to correct this message, but make another mistake. Edit the CIF and type:

```
IOPT_GRID_NODIM = 4
```

This is obviously an error, because the maximum number of dimensions in COHERENS is 3. When you inspect the file *river1T.errlogA*, you will see the following

```
Invalid value for integer parameter iopt_grid_nodim: 4
Must be between: 1 and 3
A total of 1 errors occurred in check_mod_filepars
Error type 7 : Invalid initial values for model parameters or arrays
PROGRAM TERMINATED ABNORMALLY
```

Here COHERENS tells you that the number of grid dimensions should be between 1 and 3, and thus the value of 4 is not allowed.

Try to perform additional changes in the CIF file to generate some other error messages. Fix the errors and move on to the next exercise.

Form of the output file In this exercise, we modify the format of the output file by editing the CIF. The instructions are given below

1. Open and edit the CIF.

```
gedit river1T.CifModA
```

2. Change the format of the output file from netCDF to ASCII. Change the third character of the string

```
TSR2D = 1,T,N,riverT_1.tsout2N,T,,2
```

from N to A:

```
TSR2D = 1,T,A,riverT_1.tsout2N,T,,2
```

3. Run the model with the new output format

```
./Run
```

Now you have a new file, called *riverT_1.tsout2N*, containing ASCII output data. Open the file in **gedit** to inspect the data.

4. Change the current default filename for easy management on the same input line

```
TSR2D = 1,T,A,riverT.txt,T,,2
```

5. Run the model with the new output filename.

```
./Run
```

We strongly recommend you to use the netCDF format for your output. Therefore, undo the changes done during this exercise and continue to the next one.

Modification of model settings This exercise is focused on the modification of some settings of the model setup by editing the CIF, in the present exercise we will modify the roughness height, the instructions are given below:

1. Open and edit the CIF

```
gedit river1T.CifModA
```

2. Set the roughness to 0.3E-02 [m]

```
ZROUGH_CST = 0.3E-02
```

3. Run the model with the new roughness

```
./Run
```

Visualize the results, and see what the effects is of the changed bed roughness. You can modify many things in **COHERENS**, including the numerical scheme, the turbulence model and many physical parameters. Try to modify some other settings, by looking them up in the user manual and seeing what they do. Interesting settings to change are:

- The numerical scheme for advection, using **IOPT_ADV_SCAL**, **IOPT_ADV_2D** and **IOPT_ADV_3D**
- Horizontal viscosities using **IOPT_HDIV_2D**, **IOPT_HDIV_3D**, **HDIFMOM_CST** and **HDIFSCAL_CST**
- Vertical turbulence model using **IOPT_VDIF_COEF** and the various turbulence switches **IOPT_TURB_***.

Grid resolution This exercise is focused on the modification of the grid resolution by editing the CIF, as well as the input files. In this way, you get already a first idea how to set up a new model application. The instructions are given below:

1. Adapt the *defruns* file in order to make sure that both simulations are run (i.e. remove the ! on the first line)

```
river0T,R,river0T.CifModA
river1T,R,river1T.CifModA
```

2. Open and edit both CIFs

```
gedit river0T.CifModA
gedit river1T.CifModA
```

3. We are now resetting the setup so that the test runs with half the current grid size. Firstly, change the number of columns of the computational grid from `nc=141` to `nc=281`⁴

```
NC = 281
NR = 2
```

4. Secondly, change the grid size from 1000 to 500 m. Change the fifth and sixth fields in the line

```
SURFACEGRIDS = 1,1,0,0,1000.,1000.,0.,0.
```

in both CIFs to

```
SURFACEGRIDS = 1,1,0,0,500.,500.,0,0
```

5. Use a new file to called `riverTnew.modgrdN` for the bathymetry. We will generate this file in a moment.

```
MODFILES = modgrd,1,1,N,R,riverTnew.modgrdN,0,0,0,0,F,F,
```

6. Also change the eleventh field for the time series output grid (parameter `TSRGPARS`, in `river1T.CifModA` only, from 140 to 280

```
TSRGPARS = 1,T,F,F,T,2003/01/03:00:00:00:000,3,0,0,1,280,1,1,1,1,1,20
```

Now save the files. We will continue by making a new bathymetry file.

7. Generate the new bathymetry file. Convert the COHERENS netCDFfile an ASCII text file that can easily be edited, using `ncdump`

```
ncdump riverT.modgrdN > riverTmodgrd.txt
```

⁴The actual number of “active” in the X-direction is given by `nc-1`. Halving the grid size means then that `nc` is reset to $2*(nc-1)+1=281$.

In this command the `>` sign is used to send output from one program (in this case `ncdump`) to a file `riverTmodgrd.txt`. Edit the `riverTmodgrd.txt` file generated for a new grid resolution

```
gedit riverTmodgrd.txt
```

The changes that you have to make are

- Change the file name

```
netcdf riverTnew {
```

- Change the number of cells

X001 = 280 ;

- Change the bathymetry, by copying all values (20) after `depmean-glb`, such that are now 280 numbers

depmeanglb =

- Change the grid index location of the open boundary, to make it compliant with the new grid

iobu = 1, 281 ;

- Convert the text file back to a netCDF file using `ncgen` (look up how it works using `man ncgen`)

```
ncgen -b riverTmodgrd.txt -o riverTnew.modgrdN
```

8. The new grid file can also be created alternatively through the CIF

- Edit the *defruns* file and disable the second run

```
river0T,R,river0T.CifModA
!river1T,R,river1T.CifModA
```

- In *river0T.CifModA* change the line

```
MODFILES = modgrd,1,1,N,R,riverT.modgrdN,0,0,0,0,F,F,
```

into

```
MODFILES = modgrd,1,1,N,W,riverTnew.modgrdN,0,0,0,0,F,F,
```

- Run the test case and replace the 'W' again to 'R' on the same line in the *river0T.CifModA*. The new grid file has now been generated automatically.

9. Run the model now with the new grid resolution.

```
./Run
```

Note that for this exercise, it is necessary to run both simulations (*river0T* and *river1T*), because the initial conditions that are generated in run *river0T* are changed with the new grid resolution and need to be provided for *river1T*.

Modification of open boundaries This exercise is focused on the modification of the definitions of the open boundaries by editing the open boundary file. The instructions are given below

1. Edit both CIFs files, in order to use a new file with boundary conditions:

```
MODFILES = 2uvobc,1,1,A,R,riverTnew.2uvobc1A,0,0,0,0,F,F,
```

2. Open and edit the *riverT.2uvobc1A* file generated previously, which contains the open boundaries in COHERENS ASCII format

```
gedit riverT.2uvobc1A
```

3. Modify the type of the upstream open boundary condition from the characteristic method (11) to the method of Flather (8)

```
ityp2dobu
```

4. Modify the the water level amplitude from 0.8 to 1.0 m

```

ud2obu_amp
 14.00714      0.000000
zetobu_amp
 1.000000      0.000000

```

Note that the amplitude of the depth-integrated velocity is proportional to the water level amplitude in the **river** test case and needs therefore be divided by 0.8.

5. Save the file under the new name *riverTnew.2uvobc1A*
6. Run the model with the new boundary definitions

```
./Run
```

2.4.2 Adapting model set up via the *Usrdef*_ files

We are now going to repeate the model setup changes by adapting the FORTRAN code in the *Usrdef*_ files.

2.4.2.1 installing and running the example test

We re-install, compile and run the tutorial version of **river** test again, now without making use of the CIF. The steps are similar to ones shown in the previous section, but repeated here for clarity

1. Select a working directory, e.g.

```
cd /home/coherens/rivertest
```

2. Create a link with the COHERENS root directory (a path name can be defined instead of $\sim/V2.5$), e.g.

```
ln -s ../V2.5 COHERENS
```

3. Install the tutorial version of the **river** test case

```
COHERENS/install_test -t tutorial/river
```

4. Adapt your settings by changing *coherensflags.cmp*.

5. Compile (e.g for Intel Fortran)

```
make linux-iforts
```

6. Run the test case

```
./Run
```

2.4.2.2 changing model setup through the *Usrdef*_ files

The modifications in the model setup, discussed with the CIF method in the previous section, can equally well be performed, by modifying the setups defined in the *Usrdef*_ files. Main difference is that you need to recompile the code each time a *Usrdef*_ file has been changed. This recompilation only takes a few seconds of time.

Changing model setup parameters As before, switch off the first of the two simulations in *defruns*

```
!river0T,,  
river1T,,
```

1. Edit the file *Usrdef_Model.f90*

```
gedit Usrdef_Model.f90
```

2. To change the simulation time, modify the following line in routine *usrdef_mode_params* from

```
CendDateTime = 2003/01/06;00:00:00:000
```

into

```
CendDateTime = 2003/01/04;00:00:00:000
```

3. Note that, lower case characters are used preferentially for variable names contrary to the CIF where all parameter names are given in upper case.
4. Since the comment character ‘!’ has the same meaning in FORTRAN as for the CIF, you may also comment the old definition

```
!CendDateTime = 2003/01/06;00:00:00:000  
CendDateTime = 2003/01/04;00:00:00:000
```

5. Recompile and run the modified test

```
make linux-iforts  
./Run
```

6. A new time step is taken by changing the following lines in *Usrdef_Model.f90* (routine `usrdef_mod_params`) from

```
delt2d = 30.0
...
ic3d = 10
```

to

```
delt2d = 15.0
...
ic3d = 20
```

7. The roughness length parameter `zrough_cst` can be changed in the same routine.
8. The switch `iopt_grid_nodim` is not defined in *Usrdef_Model.f90*. COHERENS will therefore take the default value which is 3. You can reset this value to the erroneous value of 4 in routine `usrdef_mod_params` (preferentially in section “Switches”) within the file *Usrdef_Model.f90*

```
...
SUBROUTINE usrdef_mod_params
...
iopt_grid_nodim = 4
...
END SUBROUTINE usrdef_mod_params
```

9. Compile and run. The same error message as in the CIF example is issued.

Form of the output file The instructions for modifying the format of an output file are as follows

1. Edit the file *Usrdef_Time_Series.f90*

```
gedit Usrdef_Time_Series.f90
```

2. Change the output format by adding the two lines

```

...
SUBROUTINE usrdef_tsr_params
...
tsr2d(1)%form = 'A'
tsr2d(2)%filename = 'riverT.txt'
...
END SUBROUTINE usrdef_tsr_params

```

in routine `usrdef_tsr_params`. The second line changes the name of the output file as well.

3. Compile and run.

Grid resolution Halving the grid size (in the X-direction) is easily performed by modifying the following lines in `Usrdef_Model.f90`

```

...
SUBROUTINE usrdef_mod_params
...
nc = 281; nr = 2; nz = 20
...
surfacegrids(igrd_model,1)%delxdat = 500.0
surfacegrids(igrd_model,1)%delydat = 500.0
...
END SUBROUTINE usrdef_mod_params

```

Contrary to the CIF case, the output grid, the grid index of the open boundary locations and the bathymetry are automatically adapted by the code to the new grid.

Modification of open boundaries The upstream open boundary condition is reset from the characteristic (11) to the Flather (8) method and the tidal amplitude of the water level is reset from 0.8 to 1.0 m in `Usrdef_Model.f90`

```

...
SUBROUTINE usrdef_2dcbc_spec
...
ityp2dcbc(1) = 8; ityp2dcbc(2) = 13
...
amp = 1.0; phase = -halfpi
...
END SUBROUTINE usrdef_2dcbc_spec

```

Note that only the parameter `amp` needs to be changed. The amplitudes of the water elevation and depth-integrated current are automatically adapted by the code in routine `usrdef_2dcbc_spec`.

2.5 Format and specific syntax of a CIF

Example of a CIF file for the test case River The CIF used in this tutorial is shown here as an example what the CIF looks like

```
!*****
! This is an example CIF file, for running the test case river
! It is the second of two files, which runs the simulation of a
! density current.
! This file uses the tutorial settings
! Written by Alexander Breugem
! April 2012
!*****
```

```
! Monitoring parameters
!*****
! In this part, parameters are defined that determine the
! parameters that are written to the logfiles
!
! Get reasonable amount of logdata. Increase the number to obtain more data
LEVPROCS_INI = 7
LEVPROCS_RUN = 3

! Generate a detailed timing file
LEVTIMER = 3

#
!*****
! Switches and parameters.
! Do not remove the # above
!
! In this part, parameters are set for the run

! Define grid
IOPT_GRID_NODIM = 3
```

```
! Use open boundary conditions
IOPT_OBC_2D = 1

! Switching on simulation of baroclinic currents
IOPT_DENS = 1
IOPT_DENS_GRAD = 1

! Calculate salinity
IOPT_SAL = 2

! Turbulence switch for limiting conditions
IOPT_TURB_IWLIM = 1

! Set up model grid
! Note that you always define an extra dummy cell on the edges
NC = 141
NR = 2
NZ = 20

! Define number of sea boundary conditions
NOSBU = 2
NOSBV = 0
! Define number of river boundary conditions
NRVBU = 0
NRVBV = 0

! Define number of tidal constituents at the boundary
NCONOBC = 1

! Define type of constituent (S2 = 51)
INDEX_OBC = 51

! Set start and endtime and Time step
CSTARTDATETIME = 2003/01/03;00:00:00:000
CENDDATETIME = 2003/01/06;00:00:00:000
DELT2D = 30.
IC3D = 10

! Select constant water depth
DEPMEAN_CST = 20.
```

```
! Accelaration of gravity
GACC_REF = 9.81

! Bottom friction coefficient
ZROUGH_CST = 0.6E-02

! Define restart files
NORESTARTS = 0

! define runtitle
INTITLE = riverT
OUTTITLE = riverT

! Define number of outputs for time series
NOSETSTSR = 1
NOVARSTSR = 6

! Model files
!*****
! The format is: file descriptor,file number,input(1)/output(2),
!                 format(A=ascii,N=netcdf,U=binary),
!                 status(R=read,W=write,N=usrdef), filename,
!                 tlim1,tlim2,tlim3,endfile,info,time_regular,path

! Grid input
MODFILES = modgrd,1,1,N,R,riverT.modgrdN,0,0,0,0,F,F,

! Open boundary conditions input
MODFILES = 2uvobc,1,1,A,R,riverT.2uvobc1A,0,0,0,0,F,F,

! Initial conditions input (restart file from simulation 0)
MODFILES = inicon,1,1,N,N,riverT.phsfinN,0,0,0,0,F,F,

! Horizontal coordinates
SURFACEGRIDS = 1,1,0,0,1000.,1000.,0.,0.

#
!*****
! Do not remove the # above
!
```

```

! Parameters for time series output
!

! Definition of the output variables
! The format is: number (output), key id, dimension (0/2/3),
!                 oopt,klev,dep, node,fortran name, long name, unit
! Only the first 3 are required

! Surface elevation (92)
TSRVARS = 1,92,2,,,,,
! Depth avg U velocity (104)
TSRVARS = 2,107,2,,,,,

! U velocity
TSRVARS = 3,109,3,,,,,
! V velocity
TSRVARS = 4,121,3,,,,,
! Vertical velocity
TSRVARS = 5,123,3,,,,,
! Salinity
TSRVARS = 6,128,3,,,,,

! Matching the variables to sets
! Format: iset, ivar1, ivar2,etc.
IVARSTSR = 1,2,3,4,5,6

! Definition of the output files
! Format: set number, defined (T/F), format (A=ascii,N=netCDF, U=binary),
!           filename, info (T/F), path, header_type
TSR2D = 1,T,N,riverT_1.tsout2N,T,,2
TSR3D = 1,T,N,riverT_1.tsout3N,T,,2

! Definition of output grid
! Format: set number,gridded (T/F),gridfile (T/F),land.mask (T/F),
!           timegrid (T/F),ref_date,number dimensions,number stations,
!           start x,end x,stepx,start y,end y,step y, ,start z,end z,
!           step z,start time,end time,timestep
! note that the end time may have to change if you change the time step or
! time interval of the simulation
TSRGPARS = 1,T,F,F,T,2003/01/03;00:00:00:000,3,0,0,1,140,1,1,1,1,1,20,1,0,
           8640,180

```

```
#  
!*****  
! Do not remove the # above  
!  
! Parameters for time averaged output  
  
#  
!*****  
! Do not remove the # above  
!  
! Parameters for harmonic analysis  
  
#  
!*****  
! Do not remove the # above  
!  
! Parameters for harmonic output  
  
#
```


Chapter 3

Compilation and installation

The objectives of this chapter are to explain program installation and compilation, how to run one of the built-in test cases and to summarise the different steps for setting up a user-defined application. It is assumed that the operating system is either Unix or Linux¹.

This chapter is organised as follows:

- Instructions for installation and a description of the program's file system are given in Section 3.1.
- Compilation is explained in Section 3.2.
- The different steps needed to run a test case are explained in Section 3.3. For a detailed description of all test cases the reader is referred to Part ??.
- An example for installing and running a user-defined application is given in Section 3.4.
- Implementation of external libraries is discussed in Section 3.5.
- A general outline explaining how to set up a realistic application of the model, is presented in Section 3.6. For a complete description the user is referred to the User Manual (Part IV).
- More detailed technical information for installing and compiling COHERENS is provided in Section 3.7.

¹Installation and compilation of COHERENS under a WINDOWS platform is not excluded *a priori* but is beyond the scope of this User Documentation.

3.1 Installation

In the following it is assumed that the compressed file with the model code, e.g. `coherensV2.5.tar` is downloaded on the user's home directory. To retrieve the files use

```
tar -xvf coherensV2.5.tar
```

This creates a file directory tree, as shown in Figure 3.1. The root directory is **V2.5** which contains the following subtrees

1. **code** This directory contains subdirectories for each model compartment

- **physics**: “main” physical compartment
- **sediment**: sediment model compartment
- **biology**: biological model compartment (currently not yet available)

Each of these compartments has a **source** directory with the source code and a **comps** directory with the files for compilation.

2. **setups**

- **examples**: example code for setting up a user application
- **ptests**: contains subdirectories each representing a different computing platform. In each directory there are check-up tables for each test case experiment (see Section 3.3).
- **tutorial**: setups of test(s) discussed in the tutorial manual (Chapter 2).
- **cones, ...**: setups of pre-defined test cases

3. **data**: data files used in some of the test case applications

4. **scr**: examples script for running the code on a Unix/Linux platform. For serial runs **Run** is the most obvious choice.

5. **utils**: utility programs

- **decomp**: utility program for creating a domain decomposition
- **post**: (non-portable) postprocessing program (only used by some developers)

The file **install_test** in the root directory is a script, primarily intended for installing a test case. Its purpose is further explained below.

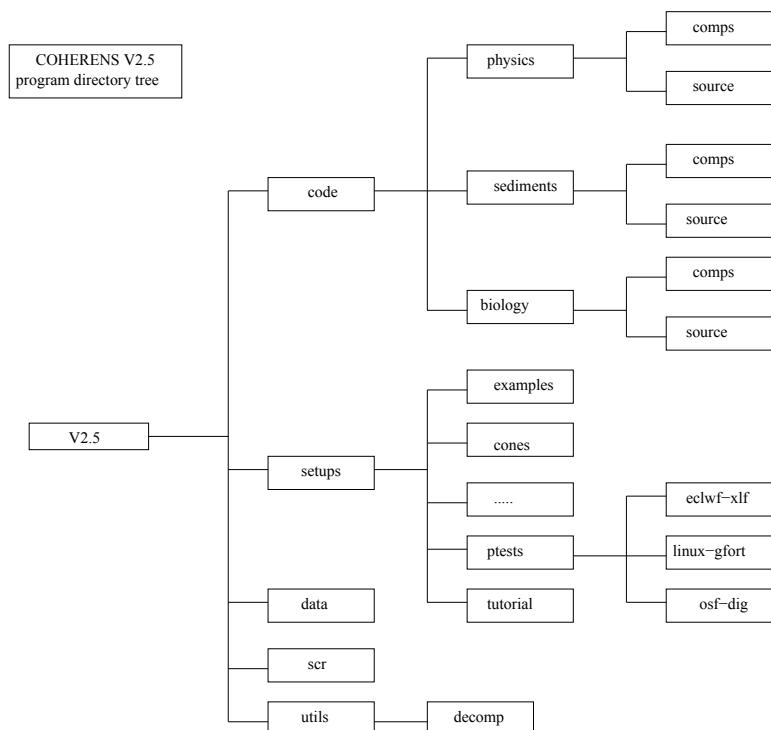


Figure 3.1: COHERENS file directory tree

3.2 Compilation

To compile the COHERENS source code, the following tools are essential

- FORTRAN 90 compiler
- C-preprocessor, usually `cpp`
- a compiled MPI library which is needed (only) for simulations in parallel mode. Note that MPI is not supported by all compilers.
- a compiled netCDF library (version 3.6 or higher) which is needed (only) to read and write data in portable netCDF format.

Compilation is performed with the Unix/Linux `make` utility. The rules for making the executable file `coherens` are defined in the file `Makefile` located in the `comps` directory. Contrary to COHERENS V1 this file is given in a portable format, i.e. independent of the type of compiler or operating system. `Makefile` reads input from a series of additional files:

- `objects.cmp`: defines macros with a listing of objects (`*.o`) files.
- `objects_bio.cmp`, `objects_sed.cmp`: defines macros with a listing of object files for the biology, respectively sediment model.
- `dependencies.cmp`: describes all file dependencies for the compiler.
- `dependencies_bio.cmp`, `dependencies_sed.cmp`: describes all file dependencies for the compiler and for the biology, respectively sediment model.
- `compilers.cmp`: defines targets for different compilers and computing platforms.
- `coherensflags.cmp`: list of compiler options for the C-preprocessor, links with external libraries². For further details see section 3.7.1.

The object and dependency files are portable and should not be changed unless the main source code in `source` has been modified by the user. The last two files are user dependent.

²This file is a more extended version of the previous file `options.cpp` file, which has been removed in Version 2.4.

3.2.1 C-preprocessing

Besides links for using the `netCDF` library, further discussed below, the user can define options in `coherensflags.cmp` for the CPP through the macro `CPPFLAGS`. The syntax of this macro is:

```
CPPFLAGS = -Dname1 -Dname2 ...
```

where `name` corresponds to a C-language statement in the source code of the form

```
#ifdef name
...
#endif
```

or

```
#ifdef name
...
#else
...
#endif
```

If `name` is defined in `CPPFLAGS`, the C-preprocessor retains all statements between `#ifdef` and `#endif` (first form) or between `#ifdef` and `#else` (second form) and removes all statements between `#else` and `#endif` (second form). If `name` is not defined, all statements are removed between `#ifdef` and `#endif` (first form) or between `#ifdef` and `#else` (second form) and all statements are retained between `#else` and `#endif` (second form). Finally, the lines starting with `#` are removed. The remaining code is then passed to the FORTRAN compiler.

Several CPP options are implemented. The most relevant for COHERENS users are:

-DMPI

Needed for parallel execution of the program. An error is issued by the compiler if the MPI library is not installed and linked to the main source code.

-DCDF

Needed for data input/output in `netCDF` format. An error occurs if the `netCDF` library is not installed and linked to the main source code. If this option is defined, all user output is written in `netCDF` format by default.

-DALLOC

Option for allocation of most local arrays in the model. This will provide a more efficient memory management.

In the default version of *coherensflags.cmp*, supplied with the source code, **CPPDFLAGS** is left undefined.

3.2.2 Testing compilation

Compilation can be tested by the following recommended procedure

1. Select a working directory, e.g.

```
cd /home/test
```

2. Create a link with the COHERENS root directory

```
ln -s path_name COHERENS
```

where **path_name** is the path name of the **coherens/V2.5** root directory, e.g. **/home/coherens/V2.5**.

3. Install the standard COHERENS code

```
COHERENS/install_test
```

The script **install_test** creates links to subtrees of **coherens/V2.5**, copies the *Makefile* and *coherensflags.cmp* files and all the files in the **scr** subdirectory to the current directory.

4. Compile

```
make target_name
```

where **target_name** equals one of the targets defined in *compilers.cmp*.

The *Makefile* script reads the *coherensflags.cmp* file residing in the working directory, which, by default, includes no compiler options. Implementation of the MPI or netCDF library can be tested by inserting the appropriate options in this file. See Section 3.5 below.

3.3 Installing and running a test case

In addition to the main source code a total of 19 pre-defined test cases are supplied to the user. Their aim is to:

- test the installation and compilation of the code
- provide examples of model setups
- show how model results are affected by different setups (e.g. numerical schemes, turbulence closures, different kinds of processes, . . .)
- provide a debugging tool.

Table 3.1 provides a list of all test cases. Each case is composed of a series of experiments. Each experiment has a name given by the name of the test case followed by an upper case letter. For example, the **cones** test consists of four experiments each representing a separate simulation: **conesA**, **conesB**, **conesC**, **conesD**. A total of 107 experiments are defined in this way.

The procedure for installing and running a specific test case is analogous to the one presented in Section 3.2.2. The complete procedure, including running the application, is as follows.

1. Select a working directory, e.g.

```
cd /home/test
```

2. Create a link with the COHERENS root directory

```
ln -s path_name COHERENS
```

where *path_name* is the path name of e.g. the **coherens/V2.5** root directory, e.g. **/home/coherens/V2.5**.

3. Install the test case

```
COHERENS/install_test -t test_name
```

where **test_name** is the name of the test case (e.g. **cones**). This creates links to subtrees of **coherens/V2.5**, copies the *Makefile* and *coherensflags.cmp* files, all the files in the **scr** subdirectory and the setup files in **setups/test_name** to the current directory.

4. Compile

Table 3.1: Test case descriptions

Name	Experiments	Description
<i>cones</i>	A–D	advection of a uniformly rotating “cone” shaped contaminant distribution
<i>front</i>	A–D	advection of a layered contaminant distribution by a tidal slope current
<i>seich</i>	A–E	propagation of an internal wave within a closed channel
<i>freddy</i>	A–D	generation of baroclinic instabilities by a fresh water distribution immersed in a rotating basin
<i>pycno</i>	A–G	deepening of an initially stratified surface layer by action of a uniform wind stress (1-D application)
<i>csnsp</i>	A–I	evolution of temperature and seasonal stratification at station CS in the central North Sea (1-D application)
<i>river</i>	A–D	propagation of a salinity front in a tidal channel
<i>plume</i>	A–G	formation and evolution of a tidally modulated river plume
<i>rhone</i>	A–G	simulation experiments of the Rhone plume in the Gulf of Lions (Mediterranean Sea)
<i>bohai</i>	A–F	barotropic tidal simulations of the Bohai Sea (northern part of the Yellow Sea)
<i>flood2d</i>	A–D	flooding and drying experiments in a channel using different bathymetries
<i>flood3d</i>	A–D	flooding and drying experiments in a rectangular basin using different bathymetries
<i>bedload</i>	A–F	experiments for bed load transport
<i>totload</i>	A–F	experiments for total load transport
<i>wavload</i>	A–D	experiments for bed/total load transport including wave effects
<i>sedvprof</i>	A–H	diffusion and settling of sediments in a water column (1-D application)
<i>sedhprof</i>	A–G	experiments simulating the transition between a erodable and a non-erodable sea bed
<i>seddens</i>	A–E	turbidity flow experiments in a channel due to an horizontal sediment concentration gradient
<i>thacker</i>	A–D	flow experiments, including suspended sediments, in a rotating parabolic basin with moving boundaries

```
make target_name
```

where `target_name` equals one of the targets defined in `compilers.cmp`.

5. Run all experiments

```
./Run
```

or equivalently,

```
./coherens
```

Instead of running all experiments of a test case at once, the user can make a selection by editing the file `defruns`. This file contains the names of all experiments for the specific test case on different lines. If a `!` is inserted at the beginning of a line, the corresponding experiment will be skipped.

To illustrate the use of the CIF utility, the test case runs can be set up in two modes, depending on different choices for the `defruns` file. In the first case, the `defruns` file located in the test case directory is taken and the setup is as before. In the second case, instructions for installation are the same as before except that the following copy has to be made in the working directory

```
cp cifruns defruns
```

If `Run` is executed, two simulations are performed for each experiment

1. The program creates a CIF and a series of forcing files in COHERENS standard format. No calculations are performed.
2. The test is run again. The program first reads all model setup parameters and forcing data from the previously created CIF and standard forcing files and then performs the actual calculations.

The CIF utility is further discussed in Sections 14.4 and 18.3.

The following checks can be made for a succesfull run:

1. The `Run` script terminates with exit status 0.
2. The program terminates with the following message, sent to standard output (screen or batch file):

```
Main program terminated
```

3. At the start of each numerical experiment, the program creates a file whose name equals the experiment's name (as listed in `defruns`) followed by the suffix `.errlogA` (e.g. `conesA.errlogA`). The file is used for the writing of eventual error messages and is automatically deleted at the end of the simulation. An error occurred if the file still exists, even without any contents, after completion of the run.
4. The program writes run-time information to a “log”-file with suffix `.runlogA` (e.g. `conesA.runlogA`). The last line of this file should read

`Close file log_file on unit 1 (A)`

where `log_file` is the name of the “log”-file.

It is clear that even when the program terminates without any noticeable error, the results can still be incorrect. This can easily be verified. Each experiment produces a file with suffix `.tst` (e.g. `conesA.tst`). The file contains values of some critical parameters produced by the simulation. These can be compared to the ones obtained from a reference run and listed in a file with the same name located in one of the `setups/ptests` subdirectories³.

3.4 Installing a user application

The procedure below, analogous to the one followed for installing a test case, is to be considered as an example

1. As discussed below a series of setup files needs to be created by the user. Assume now that they are located on some user directory, say `/home-/mytest`
2. Select a working path for compilation and running of the application, e.g.

`cd /tmp/mytest`

3. Create a link with the COHERENS root directory where `path_name` is the path name of the `coherens/V2.5` root directory, e.g. `/home-/coherens/V2.5`.

`ln -s path_name COHERENS`

³The parameter `sdev` defined in some test cases is only used as a measure of rounding errors and should not be considered as critical.

4. Install the user application on the current directory

```
COHERENS/install_test -u /home/mytest
```

This creates links to subtrees of **coherens/V2.5**, copies the *Makefile* and *coherensflags.cmp* files, all the files in the **scr** subdirectory and all the setup files in **/home/mytest** to the current directory.

5. Run the application

```
./Run
```

or equivalently,

```
./coherens
```

3.5 Running an application with external libraries

3.5.1 Parallel application

The procedure is analogous to the previous ones with the following additional steps:

1. Make sure that the MPI library is properly installed.
2. Insert the appropriate library options in *compilers.cmp* (if needed).
3. Insert -DMPI as CPP compiling option in *coherensflags.cmp*.
4. The parameter **nprocs**, defined in the routine **usrdef_mod_params** (file *Usrdef_Model.f90*) must be set to the number of processes used in the parallel application.
5. Edit the **Run** script according to the guidelines for running a MPI application on the user machine, e.g.

```
mpirun -n 4 ./coherens
```

to run the application using 4 processes with MPICH.

3.5.2 Using netCDF output format

The following additional steps are needed:

1. Make sure that the `netCDF` library (Version 3.6 or later) is properly installed.
2. Insert `-DCDF` as compiling option in `coherensflags.cmp`.
3. Insert the appropriate library options in `coherensflags.cmp`.
4. In subroutine `usrdef_tsr_params` within the file `Usrdef_Time_Series.f90`, change the values of the parameters(s) ending with `%form` to 'N' or delete the corresponding code line or change the same parameters in the CIF in case the CIF utility has been selected (see Chapter 2). When the run is completed, all time series output will be available in `netCDF` format.

3.6 Setting up a user application

Model setup consists in defining, firstly, a series of model parameters for initialisation and, secondly, providing different kinds of input data at run-time. A first method for implementation is via calls to routines located in `Usrdef_*` files. These routines need to be programmed by the user. A summary of their contents is given below. A detailed discussion is given in Part IV. A second method, mentioned below, is through the use of the Central Input File (CIF) utility and forcing files in a standard format recognised by COHERENS.

1. `Usrdef_Model.f90`
 - parameters for “monitoring”
 - model switches
 - activate/deactivate program modules
 - selection of a specific numerical/physical scheme
 - parameters which determine the kind of forcing input (e.g. file name, type of file, ...)
 - parameters for parallel setup
 - define external surface data grid(s) if regular
 - model grid, bathymetry, locations of open boundaries

- initial conditions
 - open boundary conditions for the 2-D and 3-D mode
 - insert code to read the open boundary data
2. *Usrdef_Surface_Data.f90* : meteorological and/or SST forcing
 - define surface grid(s) if non-regular
 - insert code for reading forcing data
 3. *Usrdef_Nested_Grids.f90*
 - define locations of nested sub-grids
 4. *Usrdef_Sediment.f90*
 - sediment model parameters and switches
 - define initial conditions for the sediment model
 - define the particle properties for each fraction (size, density, . . .)
 5. *Usrdef_Time_Series.f90* : time series model output
 - define “metadata” information
 - define output resolution in space and time
 - define output parameters
 - define output data
 6. *Usrdef_Time_Averages.f90* : time averaged model output
 7. *Usrdef_Harmonic_Analysis.f90* : harmonic analysis and output (residuals, amplitudes, phases, elliptic parameters)
 8. *Usrdef_Output.f90* : output defined by the user in any kind of (non-COHERENS) format

Remarks

1. It is clear that the user only needs to define what is needed for the application.
2. (Almost) all parameters which can be defined, have default values. Only a few need to be re-defined by the user.

3. Options are provided to write all parameters and forcing data to files in a standard **COHERENS** format. These files can be used for model setup in subsequent runs.
4. An alternative, available since version V2.1.2, is to define model setup through a Central Input File (CIF) which is a parameter file read by the program during initialisation.
5. Some **usrdef**_ routines can be made redundant by defining forcing data through files in standard **COHERENS** format.
6. A whole series of “standard” output variables are available. They can be selected by the user via a so-called “key id” number in which case metadata and output data are automatically generated by **COHERENS**.

Procedures for creating *Usrdef*_ files.

1. The simplest way is to copy the setup (*Usrdef*_ files) of a suitable test case and make the necessary adaptations. The method is recommended for applications which do not require a too complicated setup.
2. A generic version of each *Usrdef*_ file is located in the **/setups/examples** subdirectory. All parameters are listed with their default values or with an undefined value (given by a ?). The user may re-define the default and either replace the question mark by a specific value or remove those lines. There are, however, no defaults for defining the input of forcing data unless the data are read from a file in standard **COHERENS** format.

3.7 Files for compilation and installation

3.7.1 *compilers.cmp*

If the user wants to add a new compiler, a new target has to be defined by inserting the following lines, using the format below, in *compilers.cmp*:

```
target\_name:
$(MAKE) $(EXEFILE) \
"FC=" "FCOPTS=" "FCDEFS=" "FCDEBUG=" \
"CPP=" "CPPF=" "CPPOPTS=" "CPPDEFS="
```

where **target_name** is arbitrarily defined by the user. Line 2 is intended by one TAB position, the following ones by blanks only. The macro definitions

on line 2 should not be changed, those on the next lines can either be defined by the user or remain undefined. The latter have the following meaning

FC

name of the Fortran 90 compiler, eventually preceded by its directory path, such as `f90`, `/usr/bin/gfortran`, `mpif90` (for parallel runs using MPICH). This macro has to be defined always!

FCOPTS

Optimisation options for the Fortran compiler.

FCDEFS

Set to `$(CPPDFLAGS)` if the C-preprocessor is implicitly invoked by the Fortran compiler, or left undefined otherwise.

FCDEBUG

Debugging options for the `FORTTRAN` compiler, e.g. `-g` (optional).

CPP

Name of the C-preprocessor including options (except `-D` options), such as `gcc -E`, if invoked implicitly by the Fortran compiler, undefined otherwise.

CPPF

Name of the C-preprocessor, if not invoked by the Fortran compiler or set to `@cp` otherwise.

CPPOPTS

Options for the C-preprocessor, excluding `-D` options, in case that the C-preprocessor is defined by `CPPF`, undefined otherwise.

CPPDEFS

Undefined if `FCDEFS` is defined, set to `$(CPPDFLAGS)` otherwise.

A number of standard targets are already defined such `linux-gfort` for the `gfortran` and `linux-iforts`, `linux-ifortp` for compilation with the `INTEL` compiler in serial, respectively parallel mode.

3.7.2 The file `coherensflags.cmp`

The file `coherensflags.cmp` is read by the `Makefile` and contains definitions of machine-dependent macros. A default (empty) version, located in the `comps` directory is listed below.

```
1 :#
2 :# Version - @COHERENScoherensflags.cmp      V2.5
3 :#
4 :# $Date: 2021-07-29 12:05:53 +0200 (Thu, 29 Jul 2021) $
5 :#
6 :# $Revision: 1378 $
7 :#
8:
9 :# options for compilation with CPP
10:## -DALLOC :allocates/deallocates local arrays
11:## -DMPI    :includes MPI library
12:## -DCDF    :includes netCDF library
13:## -DVERIF  :enables output for verification procedure
15:
16:CPPFLAGS =
17:
18:# physics directory path
19:PHYSMOD = COHERENS/code/physics
20:
21:# sediment directory path
22:# SEDMOD = $(PHYSMOD)
23:SEDMOD = COHERENS/code/sediment
24:
25:# netCDF directory path
26:#NETCDF_PATH = /usr/local
27:
28:# netCDF library file
29:#NETCDF_LIB_FILE = netcdf
30:
31:# netCDF include options
32:#FCFLAGS_NETCDF = -I$(NETCDF_PATH)/include
33:
34:# netCDF library options
35:#FLIBS_NETCDF = -L$(NETCDF_PATH)/lib -l$(NETCDF_LIB_FILE)
36:
37:# PETSc directories
38:#PETSC_DIR = /home/patrick/petsc/petsc-3.1-p5
39:#PETSC_ARCH = linux-gfort
40:
41:# PETSc include options
42:#CPPFLAGS = -I$(PETSC_DIR)/include -I$(PETSC_DIR)/include/mpiuni
```

```

-I$(PETSC_DIR)/$(PETSC_ARCH)/include
43:#FCIFLAGS_PETSC = -I$(PETSC_DIR)/include -I$(PETSC_DIR)/include/mpiuni
-I$(PETSC_DIR)/$(PETSC_ARCH)/include

44:
45:# environment variables for PETSc
46:# include $(PETSC_DIR)/conf/variables
47:
48:# PETSc library options
49:#FLIBS_PETSC = $PETSC_LIB

```

The macros, which can be defined by the user, are on the following lines

- Line 16: compiler options for the CPP. The following options are implemented
 - DALLOC Enables allocation of local arrays.
 - DMPI Allows the use of MPI routine calls.
 - DCDF Allows the use of netCDF routine calls.
 - DVERIF Used to run the test cases with the verification procedure.
- Line 19: path of the **physics** directory. This path should not be changed!
- Line 23: path of the **sediment** directory in case the user wants to enable the **COHERENS** sediment model. Alternative, as given on line 22, is to make this path the same as the **physics** path, in which case the sediment model is disabled and not compiled.
- Line 26: installation path of the **netCDF** library. The compiler then expects that the library file and the compiled **netCDF** modules are found in respectively the directories **\$NETCDF_PATH/lib** and **\$NETCDF_PATH/include**
- Line 29: name of the **netCDF** library file
- Line 32: compiler include options for **netCDF**
- Line 35: options for compilation with the **netCDF** library

The following changes are to be made by the user

- If **-DCDF** is defined on line 16, the lines 26, 29, 32 and 35 must be uncommented and changed where necessary.

3.7.3 The script `install_test`

Test cases or a user application can be installed on a working directory with the shell script `install_test` which can be invoked with optional arguments

```
install_test [-t test_name] [-u test_dir] [-o flag_file]
```

where

- t Installs the pre-defined test case `test_name`, e.g. `cones`.
- u Installs a user defined application. The setup `Usrdef_*` and `defruns` files are copied from directory `test_dir` to the directory where `install_test` is executed.
- o Copies the file `flag_file` with the user-specific compilation instructions (see above) to the file `coherensflags.cmp` in the working directory.

- The link **COHERENS** must be defined before using the script.
- The options `-t` and `-u` are mutually exclusive.
- If neither `-t` or `-u` are present, no application has been defined, but the script can be used for testing the compilation of **COHERENS** without a setup.
- If `-o` is not present, the file `coherensflags.cmp` in the **comps** directory is copied by default.

The script creates the following links

SOURCE	directory path of the “main” source code
BSOURCE	directory path of the biological source code
SSOURCE	directory path of the sediment source code
COMPS	directory path of the files for compilation of the “main” code
BCOMPS	directory path of the files for compilation of the biological source code
SCOMPS	directory path of the files for compilation of the sediment source code
SCR	directory path of the scr directory
SETUP	path of the directory where the files for the application are located
DATA	directory path of the data directory

Part II

Model description

Chapter 4

Model grid

4.1 Model coordinates

4.1.1 Coordinate systems

The coordinate units, used within the program, are either Cartesian (x, y, z) or spherical (λ, ϕ, z) , with the z -axis directed upwards along the vertical. The Cartesian coordinates are defined in a horizontal plane tangent at a location on the Earth's surface. In the spherical system λ and ϕ represent respectively the longitude (positive in the eastern, negative in the western hemisphere) and the latitude (positive in the northern, negative in the southern hemisphere). The vertical coordinate is chosen such that the surface $z = 0$ corresponds to the mean sea water level. This gives

$$z = \zeta(x, y, t) \quad \text{or} \quad z = \zeta(\lambda, \phi, t) \quad \text{at the free surface} \quad (4.1)$$

$$z = -h(x, y) \quad \text{or} \quad z = -h(\lambda, \phi) \quad \text{at the bottom} \quad (4.2)$$

where ζ is the sea surface elevation and h the mean water depth so that the total water depth H is given by $H = h + \zeta$.

Cartesian coordinates make it easier to set up a model grid in the horizontal and can be used for size-limited areas where the Earth's curvature is negligible and the Coriolis frequency can be considered as uniform in space. A further advantage is that the coordinate axes can be arbitrarily rotated in the horizontal. Rotation of a spherical coordinate system is less straightforward and is considered by the program as a particular case of a curvilinear grid (see below).

Coordinate units are meters in the Cartesian and decimal degrees in the spherical case with $-180^\circ \leq \lambda \leq 180^\circ$ and $-90^\circ \leq \phi \leq 90^\circ$. For convenience, a unifying notation will be used whereby (x_1, x_2) equals (x, y) in the Cartesian and (λ, ϕ) in the spherical case.

Switches

Coordinate systems are selected in the model with the switch `iopt_grid_sph`:

0 : Cartesian.

1 : Spherical. Default.

4.1.2 Coordinate transforms in the horizontal

The program allows to define horizontal grids in a more flexible way through the introduction of curvilinear coordinates. Consider firstly the following general horizontal coordinate transform

$$\xi_1 = f_1(x, y), \quad \xi_2 = f_2(x, y) \quad (4.3)$$

The inverse transform becomes

$$x = F_1(\xi_1, \xi_2), \quad y = F_2(\xi_1, \xi_2) \quad (4.4)$$

The distance between two neighbouring (grid) points is given by

$$\begin{aligned} \Delta d^2 &= \Delta x^2 + \Delta y^2 \\ &= \left[\left(\frac{\partial F_1}{\partial \xi_1} \right)^2 + \left(\frac{\partial F_2}{\partial \xi_1} \right)^2 \right] \Delta \xi_1^2 + \left[\left(\frac{\partial F_1}{\partial \xi_2} \right)^2 + \left(\frac{\partial F_2}{\partial \xi_2} \right)^2 \right] \Delta \xi_2^2 \\ &+ 2 \left(\frac{\partial F_1}{\partial \xi_1} \frac{\partial F_1}{\partial \xi_2} + \frac{\partial F_2}{\partial \xi_1} \frac{\partial F_2}{\partial \xi_2} \right) \Delta \xi_1 \Delta \xi_2 \end{aligned} \quad (4.5)$$

If

$$\frac{\partial F_1}{\partial \xi_1} \frac{\partial F_1}{\partial \xi_2} + \frac{\partial F_2}{\partial \xi_1} \frac{\partial F_2}{\partial \xi_2} = 0 \quad (4.6)$$

equation (4.5) can be rewritten as

$$\Delta d^2 = h_1^2 \Delta \xi_1^2 + h_2^2 \Delta \xi_2^2 \quad (4.7)$$

and (ξ_1, ξ_2) become orthogonal curvilinear coordinates. This means geometrically that the coordinate curve along which ξ_1 is a constant and the curve along which ξ_2 is constant intersect orthogonally.

Note that spherical coordinates can be considered as “pseudo”-curvilinear coordinates with respect to Cartesian coordinates with $h_1 = R \cos \phi$ and $h_2 = R$ where $R = 6371$ km is the mean radius of the Earth, defined as the radius of a sphere having the same volume as the Earth (see Appendix 2 of [Gill, 1982](#)).

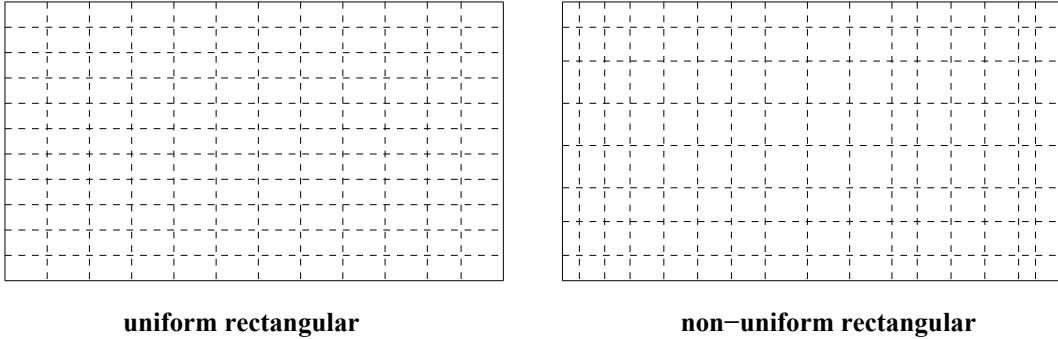


Figure 4.1: Example of a uniform (left) and non-uniform rectangular grid (right).

In practice $\Delta\xi_1, \Delta\xi_2$ are both normalised to 1, so that the metric coefficients h_1, h_2 now become the grid spacings along the curvilinear coordinate lines.

The following types of coordinate transformations are considered in COHERENS:

- “Fully” curvilinear: $h_1 = h_1(\xi_1, \xi_2)$ and $h_2 = h_2(\xi_1, \xi_2)$
- Non-uniform rectangular: $h_1 = h_1(\xi_1)$ and $h_2 = h_2(\xi_2)$, i.e. $\partial h_1 / \partial \xi_2 = \partial h_2 / \partial \xi_1 = 0$
- Uniform rectangular: h_1 and h_2 are independent of ξ_1 and ξ_2 .

A computational model grid in the horizontal is constructed at the “corner nodes” which are located at the orthogonal intersections of a series of coordinate lines along which ξ_1 is constant with coordinate lines along which ξ_2 is constant. The boxes bounded by four neighbouring corner nodes are called model “grid cells”. Figures 4.1 and 4.2 show examples of a uniform and non-uniform rectangular grid, respectively a curvilinear grid. Although not recommended, COHERENS offers the possibility to define model grids with “ragged” boundaries, obtained by removing grid cells from the physical domain where the actual calculations are performed. An example is given in Figure 4.3.

A model grid is defined in practice by supplying the coordinates of all grid nodes. In case of a fully curvilinear grid, these coordinates need to be provided by the user. For a rectangular grid, they are given by

$$\begin{aligned} x_1 &= x_r \\ x_i &= x_r + \sum_{k=1}^{i-1} \Delta x(k) \quad \text{for } i = 2, \dots, N_x + 1 \\ y_1 &= y_r \end{aligned}$$

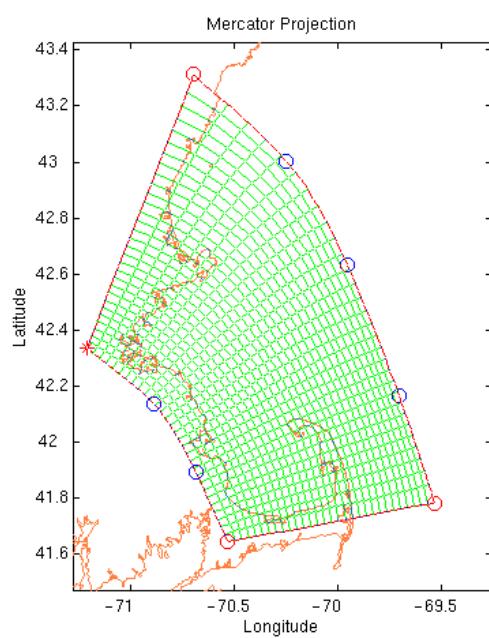


Figure 4.2: Example of a curvilinear grid

$$y_j = y_r + \sum_{k=1}^{j-1} \Delta y(k) \quad \text{for } j = 2, \dots, N_y + 1 \quad (4.8)$$

in Cartesian, or

$$\begin{aligned} \lambda_1 &= \lambda_r \\ \lambda_i &= \lambda_r + \sum_{k=1}^{i-1} \Delta \lambda(k) \quad \text{for } i = 2, \dots, N_x + 1 \\ \phi_1 &= \phi_r \\ \phi_j &= \phi_r + \sum_{k=1}^{j-1} \Delta \phi(k) \quad \text{for } j = 2, \dots, N_y + 1 \end{aligned} \quad (4.9)$$

in spherical coordinates, where N_x , N_y are the number of grid cells in the ξ_1 -, respectively ξ_2 -direction, (x_r, y_r) or λ_r, ϕ_r the coordinates of a reference point, $(\Delta x, \Delta y)$ the grid spacings along the X- and Y-direction in the Cartesian and $(\Delta \lambda, \Delta \phi)$ the grid spacings along longitude and latitude circles in the spherical case. Note that $(\Delta x, \Delta y)$ or $(\Delta \lambda, \Delta \phi)$ are always defined as positive.

The parameters which need to be defined by the user are $(x_r, y_r, \Delta x, \Delta y)$ in case of a rectangular grid and $(\lambda_r, \phi_r, \Delta \lambda, \Delta \phi)$ for a spherical grid.

Switches

The type of horizontal grid is selected with the switch `iopt_grid_htype`

- 1: Uniform rectangular. Default.
- 2: Non-uniform rectangular.
- 3: Fully curvilinear.

4.1.3 Rotated grids

To avoid that the coordinate lines of a spherical rectangular grid are restrained to latitude and longitude circles, **COHERENS** allows to apply a grid rotation. This is affected by displacing the North pole for the geographic coordinates to a new position. In this way the grid can be made more aligned with the coast line or open boundaries. An example of a rotated grid is shown in Figure 4.4

If (λ_p, ϕ_p) are the longitude and latitude of the displaced North Pole, the transformation formulae to the new coordinates (λ', ϕ') become

$$\phi' = \arcsin \left[\sin \phi_p \sin \phi + \cos \phi_p \cos \phi \cos(\lambda - \lambda_p) \right] \quad (4.10)$$

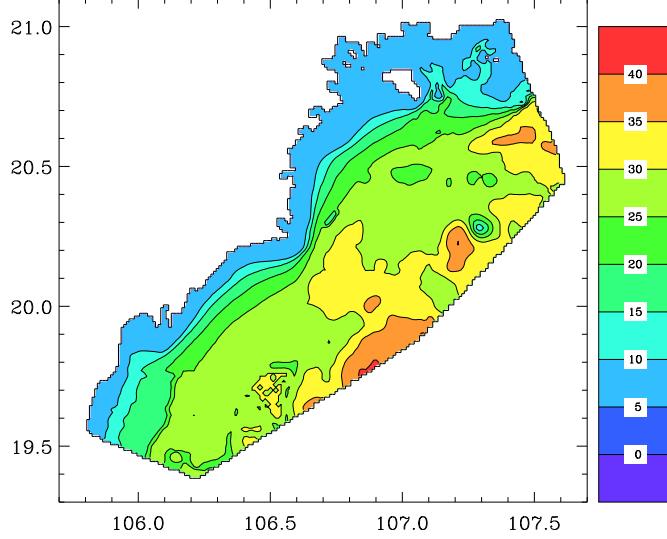


Figure 4.3: Example of a model grid with ragged boundaries

$$\lambda' = \mathcal{S}(\sin(\lambda_p - \lambda)) \arccos \left[\frac{\cos \phi_p \sin \phi - \sin \phi_p \cos \phi \cos(\lambda - \lambda_p)}{\cos \phi'} \right] \quad (4.11)$$

where $\mathcal{S}(x)$ is the Sign function¹. The backward transformation formulae are

$$\phi = \arcsin \left[\sin \phi_p \sin \phi' + \cos \phi_p \cos \phi' \cos \lambda' \right] \quad (4.12)$$

$$\lambda = \mathcal{S}(\sin \lambda') \arccos \left[\frac{\sin \phi_p \cos \phi' \cos \lambda' - \cos \phi_p \sin \phi'}{\cos \phi} \right] + \lambda_p - \mathcal{S}(\lambda_p) \pi \quad (4.13)$$

The coordinates of the grid nodes in the new coordinate grid are determined by (4.9) with $(\lambda, \phi, \Delta\lambda, \Delta\phi)$ replaced by $(\lambda', \phi', \Delta\lambda', \Delta\phi')$. The location of the new North pole is obtained by defining two additional parameters.

- The first is the grid rotation angle α defined as the angle between the geographical and the new equator. It is easily seen that $\alpha = 90^\circ - \phi_p$. Note that $0 < \alpha < 180^\circ$.

¹ $\mathcal{S}(x)$ equals 1 if $x \geq 0$, 0 otherwise.

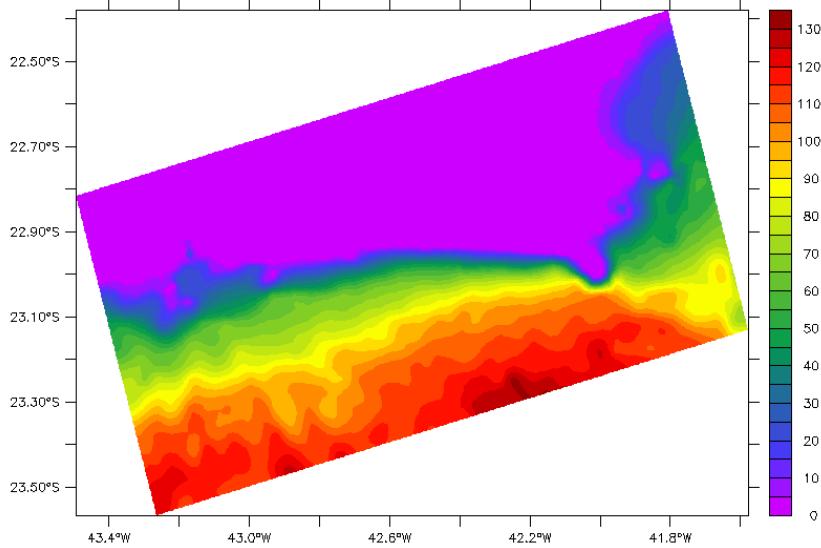


Figure 4.4: Example of a rotated grid

- The second parameter is the reference latitude ϕ'_r from which the longitude of the new North pole can be determined using (4.10):

$$\lambda_p = \lambda_r - \text{Sign}(\cos \alpha) \left| \arccos \left(\frac{\sin \phi'_r - \cos \alpha \sin \phi_r}{\sin \alpha \cos \phi_r} \right) \right| \quad (4.14)$$

where $\text{Sign}(x) = 1$ if $x \geq 0$ and -1 otherwise. The reason for taking ϕ'_r as a user-defined parameter is to allow more flexibility for selecting the grid spacing. A uniform rectangular grid with $h_1 \simeq h_2$ can be generated by letting $\Delta \lambda' = \Delta \phi'$ and $\phi'_r = 0$.

A rotated grid in Cartesian coordinates is defined by rotating the axes over the grid angle α and taking the origin of the new Cartesian frame at the reference point (x_r, y_r) . The coordinate transformations are given by

$$x' = (x - x_r) \cos \alpha + (y - y_r) \sin \alpha \quad (4.15)$$

$$y' = (y - y_r) \cos \alpha - (x - x_r) \sin \alpha \quad (4.16)$$

or

$$x = x_r + x' \cos \alpha - y' \sin \alpha \quad (4.17)$$

$$y = y_r + y' \cos \alpha + x' \sin \alpha \quad (4.18)$$

The parameters to be defined by the user are now $(x_r, y_r, \Delta x, \Delta y, \alpha)$ in case of a rectangular grid and $(\lambda_r, \phi_r, \Delta \lambda, \Delta \phi, \alpha, \phi'_r)$ for a spherical grid.

4.1.4 Coordinate transforms in the vertical

4.1.4.1 σ -coordinates

The σ -coordinate is defined by [Phillips \(1957\)](#)

$$\sigma = \frac{z + h}{H} = \frac{z + h}{h + \zeta} \quad (4.19)$$

where σ varies between 0 at the bottom and 1 at the surface². The reverse formula is obviously

$$z = \sigma H - h \quad (4.20)$$

so that the grid spacing in the vertical becomes

$$\Delta z = H \Delta \sigma \quad (4.21)$$

The spacings of vertical σ -points $\Delta \sigma$ are horizontally uniform, but can be taken as either uniform or non-uniform in the vertical.

Advantages are:

- much simpler boundary conditions at the surface and bottom
- a better resolution of surface and bottom layers

However there are well-known disadvantages of using σ -coordinates:

- areas with steep bathymetric gradients are difficult to present
- large errors can be produced by discretisation of the baroclinic pressure gradient

²Note that the definition is different from the traditional one $\sigma = (z - \zeta)/H$ with $-1 \leq \sigma \leq 0$.

A non-uniform σ -grid can be obtained by means of a transformation of the form

$$\hat{\sigma} = F(\sigma) \quad \text{or its inverse} \quad \sigma = G(\hat{\sigma}) \quad (4.22)$$

where F and G are increasing functions and $\hat{\sigma}$ equals 0 at the bottom and 1 at the surface. [Davies & Jones \(1991\)](#) defined the following logarithmic transformations

$$\sigma = \frac{1}{\alpha} \left[\ln\left(1 + \frac{\hat{\sigma}}{\sigma_0}\right) + \frac{\hat{\sigma}}{\sigma_*} \right] \quad (4.23)$$

$$\sigma = 1 - \frac{1}{\alpha} \left[\ln\left(1 + \frac{1 - \hat{\sigma}}{\sigma_0}\right) + \frac{1 - \hat{\sigma}}{\sigma_*} \right] \quad (4.24)$$

where

$$\alpha = \ln\left(1 + \frac{1}{\sigma_0}\right) + \frac{1}{\sigma_*} \quad (4.25)$$

The first (second) form provides a more refined resolution at the bottom (surface). The extent of the logarithmic grid is set by the tunable parameter σ_* .

[Burchard & Bolding \(2002\)](#) considered a formulation with refined resolutions near both the bottom and surface

$$\hat{\sigma} = \frac{\tanh[(d_l + d_u)\sigma - d_l] + \tanh d_l}{\tanh d_l + \tanh d_u} \quad (4.26)$$

Increasing the values of the (positive) parameters d_l or d_u will provide a higher resolution in respectively the bottom or surface layer at the expense of a coarser resolution in the remaining parts of the water column.

A vertical grid is then constructed by firstly taking a series of uniformly spaced σ -levels, i.e. $\sigma_k = (k - 1)/N$, $k = 1, N + 1$ where N is the number of vertical layers. In the case of a non-uniform grid, the corresponding values of $\hat{\sigma}_k$ are obtained from the transformation formula. Examples are given in Figure 4.5a-b. The first one shows that the vertical grid positions are more densely packed and the grid spacings are smaller in the bottom (surface) layer for a logarithmic transformation concentrated at the bottom (surface). The [Burchard & Bolding \(2002\)](#) formulation (with $d_l = d_u$) has enhanced resolutions both near the surface as near the bottom but a coarser resolution in the middle of the water column.

4.1.4.2 generalised σ -coordinates

Instead of using the traditional σ -coordinate a generalised vertical “ s ” coordinate can be defined by

$$z = F(x_1, x_2, s, t) \quad (4.27)$$

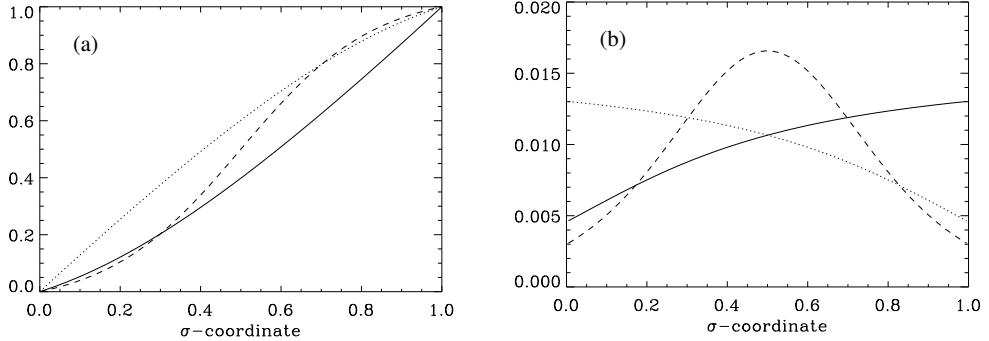


Figure 4.5: Transformed coordinate $\hat{\sigma} = F(\sigma)$ (a) and vertical grid spacing normalised to the total water depth $\Delta\hat{\sigma} = \partial F / \partial \sigma / N$ (b): formulation (4.23) with $\sigma_* = 0.25$, $\sigma_0 = 0.1$ (solid), (4.24) with the same parameter values (dots), (4.26) with $d_l = d_u = 1.5$ (dashes).

with $(x_1, x_2) = (x, y)$, (λ, ϕ) or (ξ_1, ξ_2) and where $s = 0$ at the bottom and $s = 1$ at the surface so that

$$F(x_1, x_2, 0, t) = -h, \quad F(x_1, x_2, 1, t) = \zeta \quad (4.28)$$

The vertical grid spacing now becomes

$$\Delta z = \frac{\partial F}{\partial s} \Delta s = h_3 \Delta s \quad (4.29)$$

where Δs is taken as a constant.

The distance between two neighbouring points in 3-D space now becomes

$$\Delta d^2 = h_1^2 \Delta \xi_1^2 + h_2^2 \Delta \xi_2^2 + h_3^2 \Delta s^2 \quad (4.30)$$

Song & Haidvogel (1994) related the s -coordinate to the σ -coordinate by letting

$$F = sH - h + hF_*(x_1, x_2, s), \quad F_*(x_1, x_2, 0) = F_*(x_1, x_2, 1) = 0 \quad (4.31)$$

Equation (4.29) is re-written as

$$h_3 = H + h \frac{\partial F_*}{\partial s} = H \left(1 + \frac{h}{H} \frac{\partial F_*}{\partial s} \right) \simeq H \left(1 + \frac{\partial F_*}{\partial s} \right) \quad (4.32)$$

where the approximation is made that $h \simeq H$ or $|\zeta| \ll h$. The assumption is reasonable since the s -coordinate is designed for non-shallow areas with

large bathymetric gradients, such as shelf breaks. The s -coordinate, defined by (4.31) is related to the σ -coordinate by

$$\sigma = s + \frac{h}{H} F_*(x_1, x_2, s) \simeq s + F_*(x_1, x_2, s) \quad (4.33)$$

and

$$\Delta\sigma = \frac{\Delta z}{H} \simeq (1 + \frac{\partial F_*}{\partial s}) \Delta s \quad (4.34)$$

which means that the Sung-Haidvogel s -coordinate can be seen as a generalised σ -coordinate with non-uniform spacings in the horizontal if $F_* \neq 0$.

The new coordinate should be defined so that it can represent surface and bottom layers in shallow as well as deep waters and can deal with areas with a steep topography. [Song & Haidvogel \(1994\)](#) proposed the following expression for $F_*(s)$:

$$\begin{aligned} F_*(x_1, x_2, s) &= \max \left[0, \frac{h - h_c}{h} (C(s) + 1 - s) \right] \\ C(s) &= \frac{(1 - b) \sinh [\theta(s - 1)]}{\sinh \theta} + \frac{b}{2} \left[\frac{\tanh [\theta(s - 0.5)]}{\tanh(0.5\theta)} - 1 \right] \end{aligned} \quad (4.35)$$

where h_c is a critical water depth below which the s -coordinate reduces to the σ -coordinate and b and θ are tunable parameters. The vertical grid is defined by taking uniformly spaced s -levels, i.e. $s_k = (k - 1)/N, k = 1, N + 1$ and calculating the corresponding generalised σ -levels using (4.33). Figure 4.6 compares the distribution of vertical levels for a uniform σ -spacing with the s -coordinate levels obtained from (4.35), using $b = 1$, $h_c = 200$ and $\theta = 8$, for a transect across the Norwegian trench in the North Sea. While the σ -coordinate provides an accurate resolution in the shallow waters near the Danish coast (left side of the figure), the layer thickness is ~ 30 m in the deepest part which is clearly insufficient. A (vertically) non-uniform σ -grid will not resolve the problem since any improvement for the deepest parts will deteriorate the solution in the coastal areas. The s -coordinate has a much more accurate resolution in deep water as seen in the figure on the right and reduces to the σ -coordinate when $h < h_c$ near the coasts.

A transformed vertical grid can also be constructed through the inverse relation

$$s = G(x_1, x_2, z, t) \quad (4.36)$$

or, after substituting from (4.20)

$$s = G_*(x_1, x_2, \sigma, t) \quad (4.37)$$

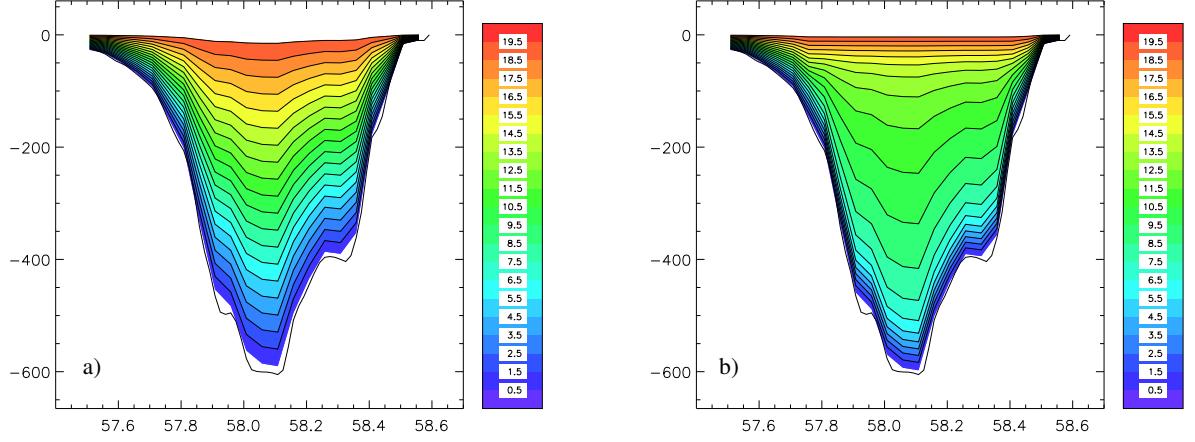


Figure 4.6: Distribution of vertical levels along a transect from Denmark to Norway: uniform σ -coordinates (a), non-uniform σ -coordinates using (4.35) (b).

with $G_*(x_1, x_2, 0, t) = 0$, $G_*(x_1, x_2, 1, t) = 1$ and $\partial G_*/\partial\sigma > 0$. For example, [Burchard & Bolding \(2002\)](#) proposed

$$s_k = \alpha\sigma_k + (1 - \alpha)\hat{\sigma}_k \quad (4.38)$$

where

$$\alpha = \min\left[\frac{(\hat{\sigma}_k - \hat{\sigma}_{k-1}) - (\sigma_k - \sigma_{k-1})h_c/h}{(\hat{\sigma}_k - \hat{\sigma}_{k-1}) - (\sigma_k - \sigma_{k-1})}, 1\right] \quad (4.39)$$

and $\hat{\sigma}$ is obtained from (4.26) with h_c a critical water depth below which $s = \sigma$. Other transformation formulae can be defined by the user as part of the model setup.

4.1.4.3 normalised vertical coordinate

In analogy with the horizontal curvilinear coordinate system the vertical s -coordinate is normalised using

$$H\Delta\sigma = h_3\Delta s \quad (4.40)$$

Setting $\Delta s = 1$ between neighbouring grid points in the (transformed) vertical direction and using similar normalised coordinates in the horizontal one has

$$\Delta d^2 = h_1^2\Delta\xi_1^2 + h_2^2\Delta\xi_2^2 + h_3^2\Delta s^2 = h_1^2 + h_2^2 + h_3^2 \quad (4.41)$$

so that h_1 , h_2 , h_3 become the grid spacings in the three (transformed) coordinate directions.

Switches

The type of vertical grid is selected with the switch `iopt_grid_vtype`

1 : Uniform σ -grid. Default.

2 : Non-uniform σ -grid in the vertical. Uniform grid in the horizontal.

3 : Non-uniform σ -grid in the horizontal and the vertical.

In the following, the general vertical coordinate will be represented by its normalised value s . This implies that

$$\frac{\partial}{\partial z} = \frac{1}{H} \frac{\partial}{\partial \sigma} = \frac{1}{h_3} \frac{\partial}{\partial s} \quad (4.42)$$

4.2 Discretised model grid

Conservative finite differences (equivalent to a finite volume technique for the Cartesian mesh) are used to discretise the mathematical model in space. The grid chosen for horizontal discretisation is the well known Arakawa “C” grid ([Mesinger & Arakawa, 1976](#)) which staggers the currents and pressure/elevation nodes to give a good representation of the crucial gravity waves and provides simple representations of open and coastal boundaries. As discussed in Section 4.1 the model equations are solved on a rectangular or curvilinear grid in the horizontal and a σ - or extended σ -coordinate grid in the vertical, whereby varying surface and bottom boundaries are transformed into constant surfaces. This provides for accurate representation of surface and bottom boundary processes. It also results in an equal number of cells in each vertical water column.

4.2.1 Grid nodes and indexing system

Figure 4.7 shows the horizontal layout of the C-grid domain as it appears in curvilinear coordinates (ξ_1, ξ_2) . A normalisation is applied so that $\Delta\xi_1=\Delta\xi_2=1$. For convenience, the notations X and Y will be used for ξ_1 and ξ_2 . It is remarked that X and Y do not refer to Cartesian axes in general. The following nodes can be distinguished:

- C-nodes (empty circles): located at the centers of the grid cells, used for 2-D and 3-D scalar quantities (elevations, water depths, ...) and wind components
- U-nodes (horizontal bars): at the centers of the left (West) and right (East) cell faces, used for the X-components of vectors except the surface wind (currents, horizontal fluxes of scalars, bottom stress, ...)

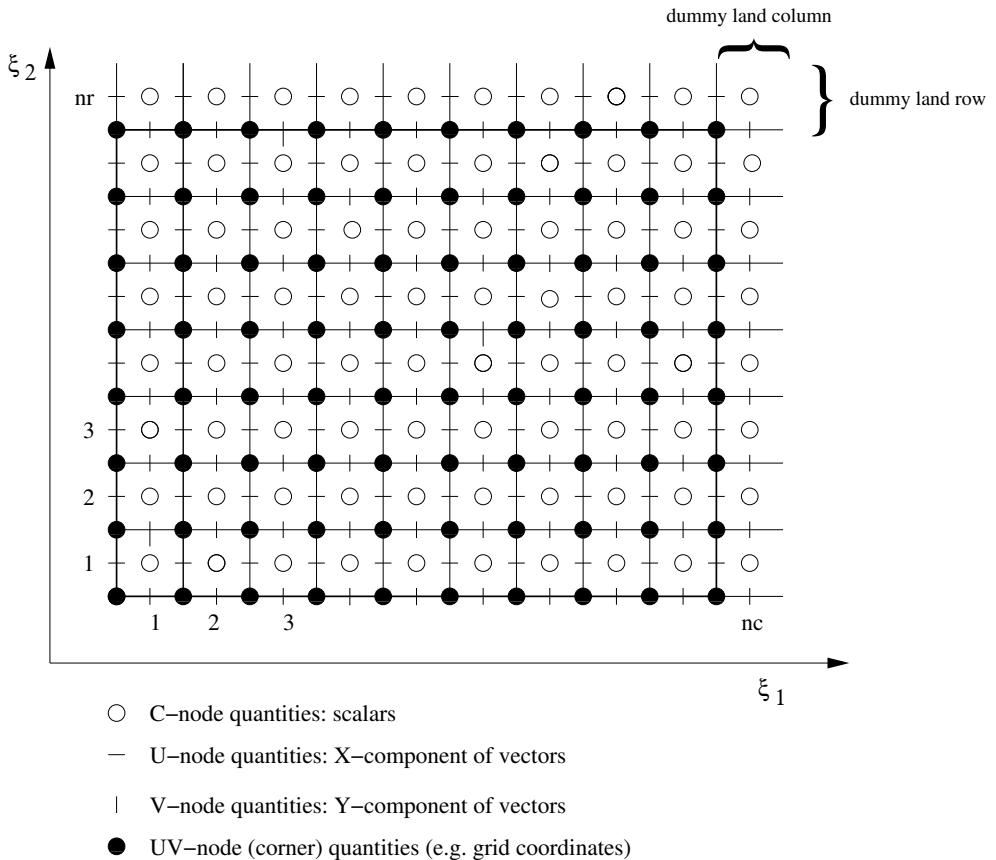


Figure 4.7: Layout of the (global) computational grid in the horizontal.

- V-nodes (vertical bars): at the centers of the lower (South) and upper (North) cell faces, used for the Y-components of vectors except the surface wind (currents, horizontal fluxes of scalars, bottom stress, ...)
- UV-nodes (solid circles): at the corners of the grid cells, used for the horizontal coordinate arrays which determine the geographical location of the grid

Each horizontal grid cell has an index, generally denoted by 'i', in the X-direction between 1 and nc and an index ('j') in the Y-direction between 1 and nr. The indices refer to the position of a variable at its “natural” node (C-, U-, V-, UV-node). This is illustrated in Figure 4.8.

As shown in Figure 4.7, the last column (to the East) and the last row (to the North) are open ended. In this way the domain contains the same number of C-, U-, V- and UV-nodes. This was not implemented in COHERENS V1 but introduced in the new version to allow a more efficient domain decom-

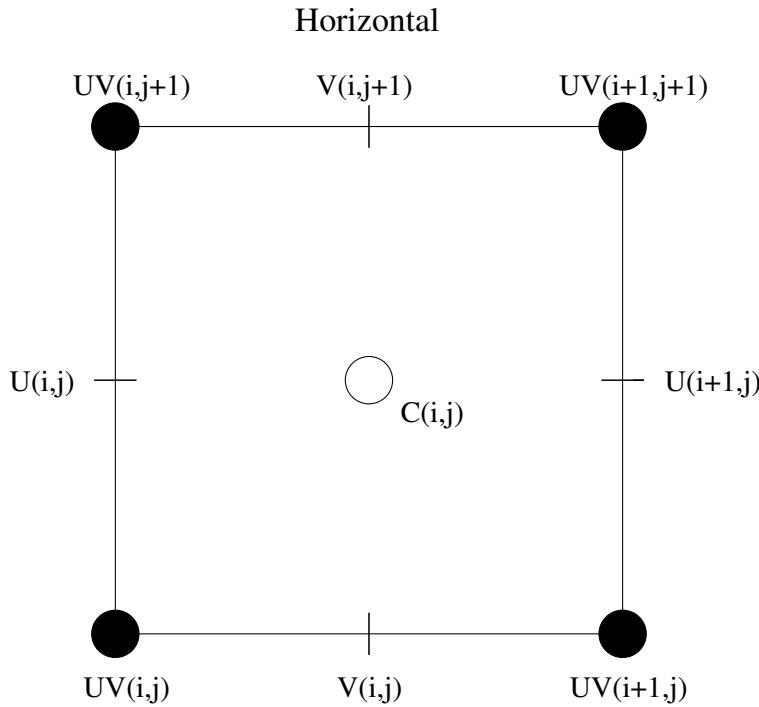


Figure 4.8: Grid indexing in the horizontal plane.

position in case of a parallel application. The drawback is that the C-node grid points with X-index `nc` or Y-index `nr` have to be declared as spurious dry cells. This means in practice that, whereas the computational size of the domain is $nc \times nr$, the physical size is $(nc-1) \times (nr-1)$ for C-node, $nc \times (nr-1)$ for U-node, $(nc-1) \times nr$ for V-node and $nc \times nr$ for UV-node quantities.

In analogy with the horizontal directions, a staggered grid is used in the vertical as well. The water column is divided into `nz` layers. The layers, which in transformed vertical coordinates have equal sizes, are illustrated in Figure 4.9. The previous C-nodes are vertically located at the midst of each layer. A new type of node, the W-node, is introduced located at the layer itself, i.e. vertically between the C-nodes and at the bottom and the surface. The vertical position of a 3-D model variable is determined by the vertical (Z)-index (“`k`”) which varies between 1 and `nz` for C-node and between 1 and `nz+1` for W-node quantities.

The grid indexing system for the full 3-D mode is shown in Figure 4.10. Combining horizontal and vertical nodes, new types of “combined” nodes arise. The following nodal types are considered in the program:

- C-nodes: at the center of a 3-D grid cell
- U-nodes: at the center of a West/East lateral face

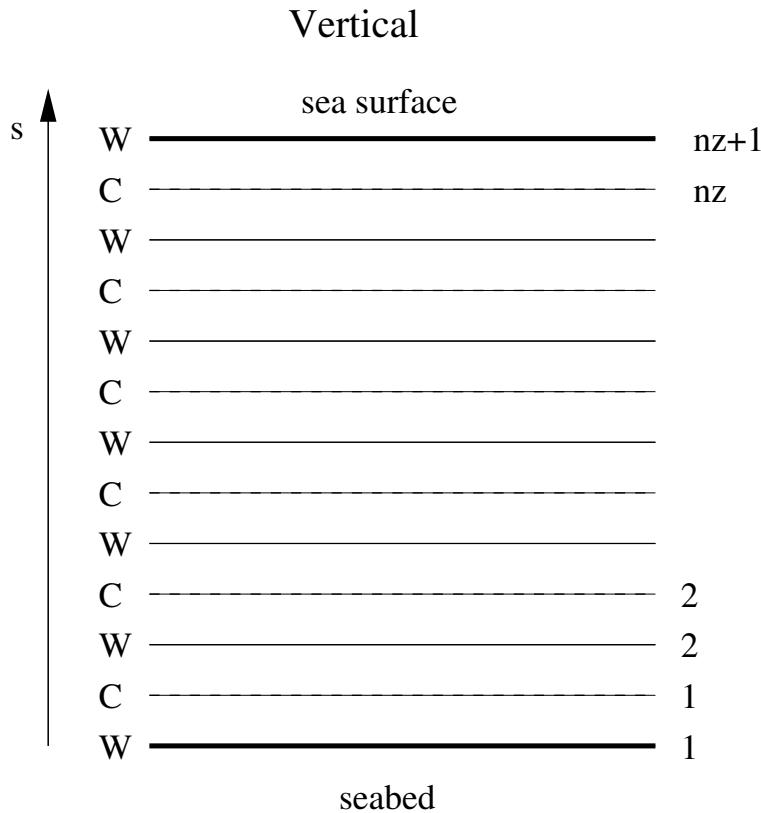


Figure 4.9: Layout of the computational grid in the vertical.

- V-nodes: at the center of a South/North lateral face
- UV-nodes: along the intersection lines of the lateral faces horizontally, halfway between the lower and upper surface vertically
- W-nodes: at the centers of the lower and upper boundary faces
- UW-nodes: as the U-nodes horizontally, as the W-nodes vertically
- VW-nodes: as the V-nodes horizontally, as the W-nodes vertically
- UVW-nodes: at the corners of a 3-D grid cell (as UV-nodes horizontally and as W-nodes vertically)

The W-nodes are used for the (transformed) vertical current and for turbulence variables (vertical diffusion coefficients and related variables). The UW-, VW- and UVW-nodes are only needed by the program for local internal variables.

Table 4.1: Upper bounds for the grid indices (i,j,k) as function of nodal type.

Node	Computational			Physical		
	X-index	Y-index	Z-index	X-index	Y-index	Z-index
C	nc	nr	nz	nc-1	nr-1	nz
U	nc	nr	nz	nc	nr-1	nz
V	nc	nr	nz	nc-1	nr	nz
UV	nc	nr	nz	nc	nr	nz
W	nc	nr	nz+1	nc-1	nr-1	nz+1
UW	nc	nr	nz+1	nc	nr-1	nz+1
VW	nc	nr	nz+1	nc-1	nr	nz+1
UVW	nc	nr	nz+1	nc	nr	nz+1

The lower bound of all grid indices is 1, the upper boundary depends on the nodal type and on whether it is taken along the computational or physical domain. A complete listing is given in Table 4.1.

4.2.2 Open boundaries

Open boundaries are defined as locations on the model grid where the solution of the discretised model equations requires values of the transport variable(s) located outside the physical domain. Open boundary conditions have to be specified at those locations. The program distinguishes four types of open boundaries:

- U-open boundaries at U-nodes needed to determine the values of the component of the horizontal current and the advective/diffusive fluxes of scalars in the X-direction.
- V-open boundaries at V-nodes needed to determine the values of the component of the horizontal current and the advective/diffusive fluxes of scalars in the Y-direction.
- X-open boundaries at UV-nodes needed to determine the cross-stream advective/diffusive fluxes of the Y-component of currents.
- Y-open boundaries at UV-nodes needed to determine the cross-stream advective/diffusive fluxes of the X-component of currents.

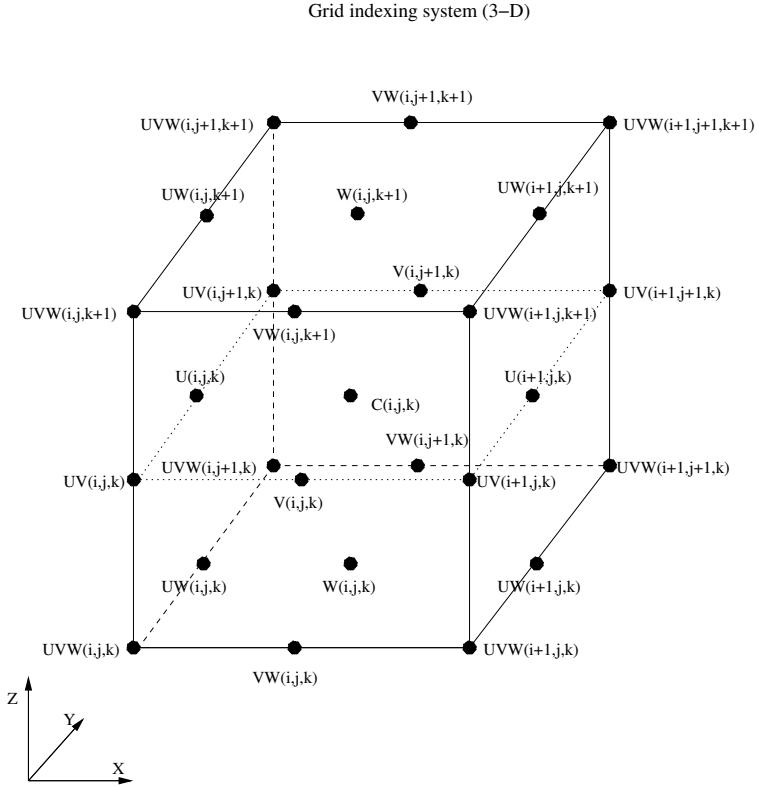


Figure 4.10: Grid indexing in three-dimensional space.

4.2.3 Conventions

A quantity taken at a grid point on its natural node is written as Q_{ij} for a 2-D or Q_{ijk} for a 3-D variable. To simplify the notations, the indices i, j, k are omitted if no confusion is possible. This means e.g. that $Q_{i,j+1,k}$ (3-D quantity) can be written as Q_{j+1} or that \bar{Q}_{i-1} (2-D quantity) is the same as $\bar{Q}_{i-1,j}$.

If a quantity needs to be evaluated at a point, different from its natural position, its value is determined by taking an average over the neighbouring points. This is indicated by one of the superscripts $^c, ^u, ^v, ^w, \dots$ referring to the point at which the quantity is interpolated. The program allows to use uniform averaging with equal weight factors or non-uniform averaging with unequal weights (see Section 15.2). To illustrate the convention, uniform averaging is assumed here for simplicity. The Coriolis terms in the momentum equations require a 4-point interpolation of the u and v velocities:

$$u_{ijk}^v = \frac{1}{4}(u_{ijk} + u_{i,j-1,k} + u_{i+1,j,k} + u_{i+1,j-1,k})$$

$$v_{ijk}^u = \frac{1}{4}(v_{ijk} + v_{i-1,jk} + v_{i,j+1,k} + v_{i-1,j+1,k}) \quad (4.43)$$

The next example is a centered quantity Q evaluated at respectively the U-, V-, W-, UW- and VW-node with the same index values:

$$\begin{aligned} Q_{ijk}^u &= \frac{1}{2}(Q_{i-1,jk} + Q_{ijk}) \\ Q_{ijk}^v &= \frac{1}{2}(Q_{i,j-1,k} + Q_{ijk}) \\ Q_{ijk}^w &= \frac{1}{2}(Q_{ij,k-1} + Q_{ijk}) \\ Q_{ijk}^{uw} &= \frac{1}{4}(Q_{ij,k-1} + Q_{ijk} + Q_{i-1,j,k-1} + Q_{i-1,j,k}) \\ Q_{ijk}^{vw} &= \frac{1}{4}(Q_{ij,k-1} + Q_{ijk} + Q_{i,j-1,k-1} + Q_{i,j-1,k}) \end{aligned} \quad (4.44)$$

A double index notation of the form $i_1 : i_2$ or $j_1 : j_2$ is sometimes introduced in expressions related to open boundary conditions, where the first index i_1 (j_1) is used at western (southern) boundaries and the second index i_2 (j_2) at eastern (northern) boundaries, such as in the following example expressions

$$u_{i+1:i-1,jk}, v_{i,j:j-1,k}$$

4.2.4 Space discretisation

The grid is defined by specifying the following three arrays:

- the x_1 -coordinates (in Cartesian or spherical coordinates) $x_{1;ij}$ of the cell corners (represented by the 2-D array `gxcoordglb(nc, nr)`)
- the x_2 -coordinates (in Cartesian or spherical coordinates) $x_{2;ij}$ of the cell corners (represented by the 2-D array `gycoordglb(nc, nr)`)
- the σ -coordinates σ_{ijk} of the W-nodes (represented by the array `gscoordglb(nc, nr, nz+1)`). Note that $\sigma_{ij1} = 0$ (bottom) and $\sigma_{ij,N_z+1} = 1$ (surface).

As discussed in Sections 4.1.2-4.1.4, the grid spacings Δx_1 , Δx_2 , Δz are set equal to respectively the metric coefficients h_1 , h_2 , h_3 by normalisation. The latter notation will be used for convenience in the following.

Spatial differences in the x_1 -, x_2 - or vertical direction are represented respectively by the operators Δ_x , Δ_y , Δ_z . The superscript c , u , v , w , uw , vw

or uv indicates the grid (nodal) location of the result. This is illustrated with the following examples (where Q represents a centered quantity in the third example):

$$\begin{aligned}\Delta_x^c u_{ijk} &= u_{i+1,j,k} - u_{ijk} \\ \Delta_y^v u_{ijk}^c &= \frac{1}{2}(u_{ijk} + u_{i+1,jk} - u_{i,j-1,k} - u_{i+1,j-1,k}) \\ \Delta_z^w Q_{ijk} &= Q_{ijk} - Q_{ij,k-1} \\ \Delta_y^c V_{ij} &= V_{i,j+1} - V_{ij}\end{aligned}\tag{4.45}$$

Grid spacings are “naturally” evaluated at the cell centre. Conforming the previous rules interpolated values at other grid locations are indicated by a superscript, e.g. $h_{1;ij}^u$, $h_{3;ijk}^w$. Note that the grid indices on the left hand side of the expressions (4.45) refer to the destination node and not the source node of the interpolation. An overview of all subscript and superscript notations, used in this chapter, is given in Table 4.2.

Table 4.2: Subscript and superscript notation used in the numerical discretisation formulae.

Type	Purpose
subscripts	
i	X-index of the variable on the model grid (between 1 and either $\mathbf{nc}-1$ or \mathbf{nc})
j	Y-index of the variable on the model grid (between 1 and either $\mathbf{nr}-1$ or \mathbf{nr})
k	vertical index of the variable on the model grid (between 1 and either \mathbf{nz} or $\mathbf{nz}+1$)
$i_1:i_2$	expression used in the spatial discretisation of open boundary conditions, whereby the first index is taken at the western and the second index at the eastern boundary
$j_1:j_2$	expression used in the spatial discretisation of open boundary conditions, whereby the first index is taken at the southern and the second index at the northern boundary
superscripts	
c	quantity evaluated or interpolated at the cell centre
u	quantity evaluated or interpolated at the U-node
v	quantity evaluated or interpolated at the V-node
uv	quantity evaluated or interpolated at the UV-node
w	quantity evaluated or interpolated at the W-node
uw	quantity evaluated or interpolated at the UW-node
vw	quantity evaluated or interpolated at the VW-node

Chapter 5

Hydrodynamic model

5.1 Hydrodynamic equations

5.1.1 3-D mode equations

5.1.1.1 Cartesian coordinates

The model equations are derived with the following (classic) assumptions.

1. The Boussinesq approximation is applied which means that the density is constant except in the Earth's gravity force term.
2. The vertical component of the momentum equations reduces to the hydrostatic balance between the vertical pressure gradient and the gravity force.
3. The horizontal component of the Earth's rotation vector is set to zero. The assumption becomes invalid for non-hydrostatic water masses.

The equations for the “3-D” mode consist of the continuity equation, the momentum equations and the equations of temperature and salinity. In Cartesian coordinates and using the previous assumptions these are given by

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} = 0 \quad (5.1)$$

$$\begin{aligned} \frac{\partial u}{\partial t} + u \frac{\partial u}{\partial x} + v \frac{\partial u}{\partial y} + w \frac{\partial u}{\partial z} - fv \\ = -\frac{1}{\rho_0} \frac{\partial p}{\partial x} + F_x^t + \frac{\partial}{\partial z} \left(\nu_T \frac{\partial u}{\partial z} \right) + \frac{\partial}{\partial x} \tau_{xx} + \frac{\partial}{\partial y} \tau_{xy} \end{aligned} \quad (5.2)$$

$$\begin{aligned} \frac{\partial v}{\partial t} + u \frac{\partial v}{\partial x} + v \frac{\partial v}{\partial y} + w \frac{\partial v}{\partial z} + fu \\ = -\frac{1}{\rho_0} \frac{\partial p}{\partial y} + F_y^t + \frac{\partial}{\partial z} \left(\nu_T \frac{\partial v}{\partial z} \right) + \frac{\partial}{\partial x} \tau_{yx} + \frac{\partial}{\partial y} \tau_{yy} \end{aligned} \quad (5.3)$$

$$\frac{\partial p}{\partial z} = -\rho g \quad (5.4)$$

$$\begin{aligned} \frac{\partial T}{\partial t} + u \frac{\partial T}{\partial x} + v \frac{\partial T}{\partial y} + w \frac{\partial T}{\partial z} \\ = \frac{1}{\rho_0 c_p} \frac{\partial I}{\partial z} + \frac{\partial}{\partial z} \left(\lambda_T \frac{\partial T}{\partial z} \right) + \frac{\partial}{\partial x} \left(\lambda_H \frac{\partial T}{\partial x} \right) + \frac{\partial}{\partial y} \left(\lambda_H \frac{\partial T}{\partial y} \right) \end{aligned} \quad (5.5)$$

$$\begin{aligned} \frac{\partial S}{\partial t} + u \frac{\partial S}{\partial x} + v \frac{\partial S}{\partial y} + w \frac{\partial S}{\partial z} \\ = \frac{\partial}{\partial z} \left(\lambda_T \frac{\partial S}{\partial z} \right) + \frac{\partial}{\partial x} \left(\lambda_H \frac{\partial S}{\partial x} \right) + \frac{\partial}{\partial y} \left(\lambda_H \frac{\partial S}{\partial y} \right) \end{aligned} \quad (5.6)$$

where (u, v) are the horizontal components of the current, w the vertical current, f the Coriolis frequency given by $2\Omega \sin \phi$ where $\Omega = \pi/43082$ radians/s is the Earth's rotation frequency, p the pressure, ρ the density, ρ_0 a uniform reference density, g the acceleration of gravity, (F_x^t, F_y^t) the components of the astronomical tidal force, ν_T and λ_T the vertical turbulent diffusion coefficients, λ_H the horizontal diffusion coefficient for scalars, τ_{ij} the horizontal friction tensor, T potential temperature, I the solar irradiance within the water column, c_p the specific heat capacity of sea water at constant pressure, and S salinity.

Note that T is not the *in situ* temperature but potential temperature, defined as the temperature of a fluid parcel, moved adiabatically to a certain level (usually taken at or near the surface). The reason is that (5.5) is derived from the conservation equation of heat. This equation contains an extra term due to compressibility, which vanishes if T is interpreted as potential temperature (Gill, 1982).

Since no ice model has currently been implemented in the model, the temperature must stay above the freezing point of seawater, i.e.

$$T > \alpha_f S, \quad \alpha_f = -0.0575 \text{ } ^\circ\text{C/PSU} \quad (5.7)$$

The horizontal diffusion tensor is introduced to represent horizontal sub-grid scale processes not resolved by the model and is parameterised as follows

$$\tau_{xx} = -\tau_{yy} = \nu_H D_T, \quad \tau_{xy} = \tau_{yx} = \nu_H D_S \quad (5.8)$$

$$D_T = \frac{\partial u}{\partial x} - \frac{\partial v}{\partial y} \quad D_S = \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \quad (5.9)$$

where D_T and D_S are called the horizontal tension and shearing strain and ν_H denotes the horizontal diffusion coefficient which is either given as a constant or taken as proportional to the local rate of strain

$$\nu_H = C_m \Delta x \Delta y \sqrt{D_T^2 + D_S^2} \quad (5.10)$$

Similarly, the scalar diffusion coefficient is either a constant or given by

$$\lambda_H = C_s \Delta x \Delta y \sqrt{D_T^2 + D_S^2} \quad (5.11)$$

Equations (5.10) and (5.11) are the well-known Smagorinsky (1963) parameterisations. The coefficients C_m and C_s usually have the same value of the order of 0.1–0.2. The sub-grid parameterisation (5.8) and (5.9) differs from the one implemented in COHERENS V1 and several other ocean models. The present formulation has been introduced in the Modular Ocean Model (Pacanowski & Griffies, 2000; Griffies, 2004) on general considerations about basic symmetry properties of the physical system.

In the formulation above it is assumed that “horizontal” diffusion of momentum and scalars takes place along horizontal planes. Diffusion along the vertical is parameterised by the vertical diffusion coefficients ν_T and λ_T . It is, however, physically more meaningful to replace these notions of horizontal mixing, produced by two-dimensional meso-scale eddies, and vertical mixing, representing small scale turbulence on scales of 10^{-3} to 10 m, by mixing along and across isopycnals. Since $\nu_T \ll \nu_H$ and $\lambda_T \ll \lambda_H$, the horizontal mixing scheme may produce an excessive diapycnal diffusion in the presence of lateral fronts. Complex schemes for isopycnal mixing of scalars have been developed and applied to global ocean models (e.g. Griffies *et al.*, 1998). An option for using isopycnal instead of geopotential mixing has been implemented in COHERENS. For a further discussion see Section 5.4.4.

The pressure can be eliminated from the above equations by rewriting (5.4) as

$$\frac{\partial p}{\partial z} = -\rho_0(g - b) \quad (5.12)$$

where the buoyancy b is defined by

$$b = -\left(\frac{\rho - \rho_0}{\rho_0}\right)g \quad (5.13)$$

Integrating (5.12) using the surface boundary condition $p = P_a$ at $z = \zeta$ where P_a is the surface atmospheric pressure, the horizontal pressure gradient

terms in (5.2) and (5.3) can be written as

$$-\frac{1}{\rho_0} \frac{\partial p}{\partial x_i} = -g \frac{\partial \zeta}{\partial x_i} - \frac{1}{\rho_0} \frac{\partial P_a}{\partial x_i} - \frac{\partial q}{\partial x_i} \quad (5.14)$$

where x_i equals x or y and q is the vertically integrated buoyancy

$$q = - \int_z^\zeta b dz \quad (5.15)$$

The first two terms on the right of (5.14) represent the barotropic, the third one the baroclinic component.

The following additional remarks are to be given:

- The vertical diffusion term and the diffusion coefficients ν_T and λ_T are obtained from a turbulence model. Various schemes, including simple algebraic schemes and more complex second order closure schemes, are implemented (see Section 5.3).
- The acceleration due to gravity can be set in the program to a constant value or obtained from the geodetic formula as function of latitude (see Appendix 2 of [Gill, 1982](#))

$$g = 9.78032 + 0.005172 \sin^2 \phi - 0.00006 \sin^2 2\phi \quad (5.16)$$

- The absorption of solar irradiance within the water column is generally a function of solar wavelength and the penetration depth of solar light. The formulation chosen in the model follows the one given by [Paulson & Simpson \(1977\)](#) whereby I is given by

$$I(x_1, x_2, z) = Q_{rad} (Re^{-z/\lambda_1} + (1-R)e^{-z/\lambda_2}) \quad (5.17)$$

where R represents the absorption of the red end of the solar spectrum in the upper (1–2) meters of the water column, $1 - R$ the absorption of blue-green light over larger depths and Q_{rad} the solar radiance incident on the surface. Since turbidity effects are not explicitly taken into account by the physical model, values of R , λ_1 , $\lambda_2 \gg \lambda_1$ depend on the optical properties of the water masses and can be selected following e.g. the optical classification scheme of [Jerlov \(1968\)](#). Solar radiance is further discussed in Section 5.8.

- The astronomical force is only relevant in deep ocean waters and can be neglected on the shelf and in the coastal zone. Expressions for its components are given as a sum of tidal harmonics (see Section 5.5).

5.1.1.2 transformed coordinates

The forms of the model equations in orthogonal curvilinear and s -coordinates (ξ_1, ξ_2, s) are derived in Appendix A. The equations of continuity and momentum, written in conservative and operator format, become

$$\frac{1}{h_3} \frac{\partial h_3}{\partial t} + \frac{1}{h_1 h_2 h_3} \left[\frac{\partial}{\partial \xi_1} (h_2 h_3 u) + \frac{\partial}{\partial \xi_2} (h_1 h_3 v) \right] + \frac{1}{h_3} \frac{\partial \omega}{\partial s} = 0 \quad (5.18)$$

$$\begin{aligned} & \frac{1}{h_3} \frac{\partial}{\partial t} (h_3 u) + \mathcal{A}_{h1}(u, u) + \mathcal{A}_{h2}(v, u) + \mathcal{A}_v(\omega, u) + \frac{v}{h_1 h_2} \left(u \frac{\partial h_1}{\partial \xi_2} - v \frac{\partial h_2}{\partial \xi_1} \right) - 2\Omega v \sin \phi \\ &= -\frac{g}{h_1} \frac{\partial \zeta}{\partial \xi_1} - \frac{1}{\rho_0 h_1} \frac{\partial P_a}{\partial \xi_1} + F_1^b + F_1^t + \mathcal{D}_{mv}(u) + \mathcal{D}_{mh1}(\tau_{11}) + \mathcal{D}_{mh2}(\tau_{12}) \end{aligned} \quad (5.19)$$

$$\begin{aligned} & \frac{1}{h_3} \frac{\partial}{\partial t} (h_3 v) + \mathcal{A}_{h1}(u, v) + \mathcal{A}_{h2}(v, v) + \mathcal{A}_v(\omega, v) + \frac{u}{h_1 h_2} \left(v \frac{\partial h_2}{\partial \xi_1} - u \frac{\partial h_1}{\partial \xi_2} \right) + 2\Omega u \sin \phi \\ &= -\frac{g}{h_2} \frac{\partial \zeta}{\partial \xi_2} - \frac{1}{\rho_0 h_2} \frac{\partial P_a}{\partial \xi_2} + F_2^b + F_2^t + \mathcal{D}_{mv}(v) + \mathcal{D}_{mh1}(\tau_{21}) + \mathcal{D}_{mh2}(\tau_{22}) \end{aligned} \quad (5.20)$$

where ω represents the transformed vertical velocity, further discussed below. The spherical case is recovered from the above equations by letting $h_1 = R \cos \phi$ (or $R \cos \phi'$ in case of a rotated grid) and $h_2 = R$.

Apart from a constant factor of proportionality, which can be set to 1 without loss of generality, the metric coefficients h_i are related to the model grid spacings along the horizontal coordinate directions $(\Delta x, \Delta y)$ and the vertical (Δz) by

$$\Delta x = h_1, \quad \Delta y = h_2, \quad \Delta z = h_3 \Delta s \quad (5.21)$$

The advection and diffusion operators are defined by

$$\mathcal{A}_{h1}(u, F) = \frac{1}{h_1 h_2 h_3} \frac{\partial}{\partial \xi_1} (h_2 h_3 u F) \quad (5.22)$$

$$\mathcal{A}_{h2}(v, F) = \frac{1}{h_1 h_2 h_3} \frac{\partial}{\partial \xi_2} (h_1 h_3 v F) \quad (5.23)$$

$$\mathcal{A}_v(\omega, F) = \frac{1}{h_3} \frac{\partial}{\partial s} (\omega F) \quad (5.24)$$

$$\mathcal{D}_{mh1}(F) = \frac{1}{h_1 h_2 h_3} \frac{\partial}{\partial \xi_1} (h_2^2 h_3 F) \quad (5.25)$$

$$\mathcal{D}_{mh2}(F) = \frac{1}{h_1^2 h_2 h_3} \frac{\partial}{\partial \xi_2} (h_1^2 h_3 F) \quad (5.26)$$

$$\mathcal{D}_{mv}(F) = \frac{1}{h_3} \frac{\partial}{\partial s} \left(\frac{\nu_T}{h_3} \frac{\partial F}{\partial s} \right) \quad (5.27)$$

The parameterised form of the horizontal shear stress tensor in orthogonal curvilinear coordinates is given by

$$\tau_{11} = -\tau_{22} = \nu_H D_T, \quad \tau_{12} = \tau_{21} = \nu_H D_S \quad (5.28)$$

$$\begin{aligned} D_T &= \frac{h_2}{h_1} \frac{\partial}{\partial \xi_1} \left(\frac{u}{h_2} \right) - \frac{h_1}{h_2} \frac{\partial}{\partial \xi_2} \left(\frac{v}{h_1} \right) \\ D_S &= \frac{h_1}{h_2} \frac{\partial}{\partial \xi_2} \left(\frac{u}{h_1} \right) + \frac{h_2}{h_1} \frac{\partial}{\partial \xi_1} \left(\frac{v}{h_2} \right) \end{aligned} \quad (5.29)$$

The quantity ω in (5.18) and (5.24) is the transformed vertical current normal to the s -coordinate surfaces. Physical and transformed vertical current are related by

$$w = \omega - h_3 \left(\frac{\partial s}{\partial t} + \frac{u}{h_1} \frac{\partial s}{\partial \xi_1} + \frac{v}{h_2} \frac{\partial s}{\partial \xi_2} \right) \quad (5.30)$$

An alternative form, more useful for numerical discretisation, is

$$w = \frac{1}{h_3} \frac{\partial}{\partial t} (h_3 z) + \mathcal{A}_{h1}(u, z) + \mathcal{A}_{h2}(v, z) + \mathcal{A}_v(\omega, z) \quad (5.31)$$

The continuity equation can be split into a barotropic (depth-integrated) and baroclinic part. The first one is obtained by integrating (5.18) over the vertical giving

$$\frac{\partial \zeta}{\partial t} + \frac{1}{h_1 h_2} \left[\frac{\partial}{\partial \xi_1} (h_2 U) + \frac{\partial}{\partial \xi_2} (h_1 V) \right] = 0 \quad (5.32)$$

where U, V are the depth-integrated currents

$$(U, V) = \int_0^1 (u, v) h_3 \, ds \quad (5.33)$$

The baroclinic component is derived by multiplying (5.18) with h_3 and subtracting (5.32) multiplied by the transformed grid spacing h_3/H :

$$\frac{1}{h_1 h_2} \left[\frac{\partial}{\partial \xi_1} (h_2 h_3 \delta u) + \frac{\partial}{\partial \xi_2} (h_1 h_3 \delta v) \right] + \frac{U}{h_1} \frac{\partial}{\partial \xi_1} \left(\frac{h_3}{H} \right) + \frac{V}{h_2} \frac{\partial}{\partial \xi_2} \left(\frac{h_3}{H} \right) + \frac{\partial \omega}{\partial s} = 0 \quad (5.34)$$

where $\delta u, \delta v$ are the baroclinic parts of the current defined by

$$(\delta u, \delta v) = \left(u - \frac{U}{H}, v - \frac{V}{H} \right) \quad (5.35)$$

The third and fourth terms in (5.34) only arise if the transformed grid spacing varies in the horizontal, i.e. in case of the most general vertical coordinate transformation. Note also that this equations involves only the baroclinic part of the current.

The horizontal vector F_i^t represents the components of the astronomical tidal force, discussed in Section 5.5. The expression for the baroclinic pressure gradient now contains two terms as a consequence of the vertical coordinate transformation

$$F_i^b = -\frac{1}{h_i h_3} \left[\frac{\partial}{\partial \xi_i} (h_3 q) - \frac{\partial}{\partial s} \left(q \frac{\partial z}{\partial \xi_i} \right) \right] \quad (5.36)$$

where

$$q = - \int_s^1 b h_3 \, ds \quad (5.37)$$

The equations for potential temperature and salinity can be cast in a more general form, representing the transport of an arbitrary concentration ψ (T , S , sediment, contaminant, biological state variable)

$$\begin{aligned} \frac{1}{h_3} \frac{\partial}{\partial t} (h_3 \psi) &+ \mathcal{A}_{h1}(u, \psi) + \mathcal{A}_{h2}(v, \psi) + \mathcal{A}_v(\omega - w_s, \psi) \\ &= \mathcal{P}(\psi) - \mathcal{S}(\psi) + \mathcal{D}_{sv}(\psi) + \mathcal{D}_{sh1}(\psi) + \mathcal{D}_{sh2}(\psi) \end{aligned} \quad (5.38)$$

where $\mathcal{P}(\psi)$, $\mathcal{S}(\psi)$ represent all source, respectively sinks terms and w_s is a vertical sinking or rising velocity (taken as positive downwards) in case that ψ represents a concentration of particulate matter.

Note that the formulation for vertical sinking/rising term is, in principle, physically incorrect, since it assumes that this vertical current is directed along the normal of the s -coordinate surface. In a correct formulation additional terms should be added to the horizontal current components

$$(u, v, \omega) \rightarrow (u - S_1 w_s, v - S_2 w_s, \omega - w_s) \quad (5.39)$$

where S_i are defined by (5.196) using the small qslope approximation, i.e. $S_i \ll 1$. The user can include these extra terms although they are seldom usefull.

The horizontal diffusion operators are, in approximated form, given by

$$\mathcal{D}_{sh1}(\psi) = \frac{1}{h_1 h_2 h_3} \frac{\partial}{\partial \xi_1} \left(\lambda_H \frac{h_2 h_3}{h_1} \frac{\partial \psi}{\partial \xi_1} \right) \quad (5.40)$$

$$\mathcal{D}_{sh2}(\psi) = \frac{1}{h_1 h_2 h_3} \frac{\partial}{\partial \xi_2} \left(\lambda_H \frac{h_1 h_3}{h_2} \frac{\partial \psi}{\partial \xi_2} \right) \quad (5.41)$$

In deriving the above equations, the assumption has been made that horizontal diffusion takes place along constant s -surfaces instead of along z -coordinate surfaces, which is only valid for smooth bathymetries. The exact formulations, including isopycnal mixing, are discussed in Section 5.4.

The vertical diffusion operator is defined by

$$\mathcal{D}_{sv}(\psi) = \frac{1}{h_3} \frac{\partial}{\partial s} \left(\frac{\lambda_T^\psi}{h_3} \frac{\partial F}{\partial s} \right) \quad (5.42)$$

where λ_T^ψ is the vertical diffusion coefficient for the scalar ψ ¹.

Smagorinsky's diffusion coefficients in curvilinear coordinates are given by

$$\nu_H = C_m h_1 h_2 \sqrt{D_T^2 + D_S^2} \quad (5.43)$$

$$\lambda_H = C_s h_1 h_2 \sqrt{D_T^2 + D_S^2} \quad (5.44)$$

Applying (5.38) for temperature and salinity one has

$$\begin{aligned} \frac{1}{h_3} \frac{\partial}{\partial t} (h_3 T) &+ \mathcal{A}_{h1}(u, T) + \mathcal{A}_{h2}(v, T) + \mathcal{A}_v(\omega, T) \\ &= \frac{1}{\rho_0 c_p h_3} \frac{\partial I}{\partial s} + \mathcal{D}_{sv}(T) + \mathcal{D}_{sh1}(T) + \mathcal{D}_{sh2}(T) \end{aligned} \quad (5.45)$$

$$\frac{1}{h_3} \frac{\partial}{\partial t} (h_3 S) + \mathcal{A}_{h1}(u, S) + \mathcal{A}_{h2}(v, S) + \mathcal{A}_v(\omega, S) = \mathcal{D}_{sv}(S) + \mathcal{D}_{sh1}(S) + \mathcal{D}_{sh2}(S) \quad (5.46)$$

5.1.2 2-D mode equations

The 3-D continuity and momentum equations presented in the previous section need to be supplemented by additional 2-D equations for the depth-integrated current and surface elevation in the following cases

1. The model is set up in depth-integrated (2-D) mode. This means that currents are assumed to be vertically uniform.

¹The same vertical diffusion coefficient is taken for temperature and salinity, i.e. $\lambda_T^T = \lambda_T^S = \lambda_T$.

2. The numerical solution of the 3-D momentum equations are constrained by the CFL limit which poses a severe limit on the time step used in the numerical discretisations. Two different numerical algorithms, described in Chapter 12, are available in COHERENSto circumvent this numerical stability problem.
- The 2-D momentum equations are integrated with a number of smaller 2-D time steps. The results are then inserted into the 3-D equations which can now be solved with a larger 3-D time step. The method is known as the mode splitting technique.
 - The 3-D momentum and the depth-integrated momentum equations are integrated using an implicit algorithm which removes the CFL stability constraint on the time step. This avoids a separate integration of the 2-D equations (except for the 2-D continuity equation (5.32) needed for updating the surface elevation).

The 2-D mode equations consists of the 2-D continuity equation (5.32) and equations for the depth-integrated currents

$$\begin{aligned} \frac{\partial U}{\partial t} &+ \overline{\mathcal{A}}_{h1}(U, U) + \overline{\mathcal{V}, \mathcal{A}}_{h2}(V, U) + \frac{V}{Hh_1h_2} \left(\frac{\partial h_1}{\partial \xi_2} U - \frac{\partial h_2}{\partial \xi_1} V \right) - 2\Omega V \sin \phi \\ &= -\frac{gH}{h_1} \frac{\partial \zeta}{\partial \xi_1} - \frac{H}{\rho_0 h_1} \frac{\partial P_a}{\partial \xi_1} + \overline{F_1^b} + HF_1^t + \frac{1}{\rho_0} (\tau_{s1} - \tau_{b1}) \\ &+ \overline{\mathcal{D}}_{mh1}(\overline{\tau_{11}}) + \overline{\mathcal{D}}_{mh2}(\overline{\tau_{12}}) - \overline{\delta A}_{h1} + \overline{\delta D}_{h1} \end{aligned} \quad (5.47)$$

$$\begin{aligned} \frac{\partial V}{\partial t} &+ \overline{\mathcal{A}}_{h1}(U, V) + \overline{\mathcal{A}}_{h2}(V, V) + \frac{U}{Hh_1h_2} \left(\frac{\partial h_2}{\partial \xi_1} V - \frac{\partial h_1}{\partial \xi_2} U \right) + 2\Omega U \sin \phi \\ &= -\frac{gH}{h_2} \frac{\partial \zeta}{\partial \xi_2} - \frac{H}{\rho_0 h_2} \frac{\partial P_a}{\partial \xi_2} + \overline{F_2^b} + HF_2^t + \frac{1}{\rho_0} (\tau_{s2} - \tau_{b2}) \\ &+ \overline{\mathcal{D}}_{mh1}(\overline{\tau_{21}}) + \overline{\mathcal{D}}_{mh2}(\overline{\tau_{22}}) - \overline{\delta A}_{h2} + \overline{\delta D}_{h2} \end{aligned} \quad (5.48)$$

where $\overline{F_i^b}$ are the depth-integrated components of the baroclinic pressure gradient F_i^b , τ_{si} and τ_{bi} are the components of the surface and bottom stress.

$$(\tau_{s1}, \tau_{s2}) = \rho_0 \frac{\nu_T}{h_3} \frac{\partial(u, v)}{\partial s} \Big|_{sur} \quad (5.49)$$

$$(\tau_{b1}, \tau_{b2}) = \rho_0 \frac{\nu_T}{h_3} \frac{\partial(u, v)}{\partial s} \Big|_{bot} \quad (5.50)$$

Additional source/sink terms, due to rainfall and evaporation (see (5.240)) or the discharge of water from an external source, are added to the right hand side of (5.32) depending on the type of application.

The advection and diffusion operators for transports are given by

$$\bar{\mathcal{A}}_{h1}(U, F) = \frac{1}{h_1 h_2} \frac{\partial}{\partial \xi_1} \left(h_2 \frac{UF}{H} \right) \quad (5.51)$$

$$\bar{\mathcal{A}}_{h2}(V, F) = \frac{1}{h_1 h_2} \frac{\partial}{\partial \xi_2} \left(h_1 \frac{VF}{H} \right) \quad (5.52)$$

$$\bar{\mathcal{D}}_{mh1}(F) = \frac{1}{h_1 h_2^2} \frac{\partial}{\partial \xi_1} (h_2^2 F) \quad (5.53)$$

$$\bar{\mathcal{D}}_{mh2}(F) = \frac{1}{h_1^2 h_2} \frac{\partial}{\partial \xi_2} (h_1^2 F) \quad (5.54)$$

The 2-D equivalents of the shear stress tensor can be written as

$$\bar{\tau}_{11} = -\bar{\tau}_{22} = \bar{\nu}_H \bar{D}_T, \quad \bar{\tau}_{12} = \bar{\tau}_{21} = \bar{\nu}_H \bar{D}_S \quad (5.55)$$

$$\bar{\nu}_H = \int_0^1 \nu_H h_3 \, ds \quad (5.56)$$

$$\begin{aligned} \bar{D}_T &= \frac{h_2}{h_1} \frac{\partial}{\partial \xi_1} \left(\frac{U}{H h_2} \right) - \frac{h_1}{h_2} \frac{\partial}{\partial \xi_2} \left(\frac{V}{H h_1} \right) \\ \bar{D}_S &= \frac{h_1}{h_2} \frac{\partial}{\partial \xi_2} \left(\frac{U}{H h_1} \right) + \frac{h_2}{h_1} \frac{\partial}{\partial \xi_1} \left(\frac{V}{H h_2} \right) \end{aligned} \quad (5.57)$$

The last two terms on the right of (5.47)–(5.48), which are only used for the mode splitting algorithm and set to zero if the model is set up in 2-D mode, contain only the baroclinic part of the 3-D current. The operators $\bar{\delta A}_{hi}$ and $\bar{\delta D}_{hi}$ are defined by

$$\begin{aligned} \bar{\delta A}_{h1} &= \frac{1}{h_1 h_2} \int_0^1 \left[\frac{\partial}{\partial \xi_1} (h_2 h_3 \delta u^2) + \frac{\partial}{\partial \xi_2} (h_1 h_3 \delta u \delta v) \right. \\ &\quad \left. + h_3 \frac{\partial h_1}{\partial \xi_2} \delta u \delta v - h_3 \frac{\partial h_2}{\partial \xi_1} \delta v^2 \right] ds \end{aligned} \quad (5.58)$$

$$\begin{aligned} \bar{\delta A}_{h2} &= \frac{1}{h_1 h_2} \int_0^1 \left[\frac{\partial}{\partial \xi_1} (h_2 h_3 \delta u \delta v) + \frac{\partial}{\partial \xi_2} (h_1 h_3 \delta v^2) \right. \\ &\quad \left. + h_3 \frac{\partial h_2}{\partial \xi_1} \delta u \delta v - h_3 \frac{\partial h_1}{\partial \xi_2} \delta u^2 \right] ds \end{aligned} \quad (5.59)$$

$$\overline{\delta D}_{h1} = \frac{1}{h_1 h_2} \left[\frac{1}{h_2} \frac{\partial}{\partial \xi_1} \left(h_2^2 \int_0^1 \nu_H \delta D_T h_3 ds \right) + \frac{1}{h_1} \frac{\partial}{\partial \xi_2} \left(h_1^2 \int_0^1 \nu_H \delta D_S h_3 ds \right) \right] \quad (5.60)$$

$$\overline{\delta D}_{h2} = \frac{1}{h_1 h_2} \left[\frac{1}{h_2} \frac{\partial}{\partial \xi_1} \left(h_2^2 \int_0^1 \nu_H \delta D_S h_3 ds \right) - \frac{1}{h_1} \frac{\partial}{\partial \xi_2} \left(h_1^2 \int_0^1 \nu_H \delta D_T h_3 ds \right) \right] \quad (5.61)$$

where δD_T and δD_S are given by (5.29) with (u, v) replaced by $(\delta u, \delta v)$.

Switches

The solution method of the previous set of equations is controlled by the following model switches

iopt_curr	Type of update for the currents.
	0: Currents and elevations are set to zero.
	1: Currents are constant in time but may be non-uniform in space, elevations are zero.
	2: Currents and elevations are updated in time. Default.
iopt_temp	Type of update for the temperature field.
	0: Temperature is uniform in space and time. Default.
	1: Temperature is uniform in time, but non-uniform in space.
	2: Temperature is non-uniform in space and time and obtained by solving equation (5.45).
iopt_sal	Type of update for the salinity field.
	0: Salinity is uniform in space and time. Default.
	1: Salinity is uniform in time, but non-uniform in space.
	2: Salinity is non-uniform in space and time and obtained by solving equation (5.46).
iopt_grid_nodim	Dimension of the model grid.
	1: 1-D vertical water column model grid (i.e. horizontal gradients are ignored except for the surface elevation)
	2: 2-D vertically integrated grid

3: 3-D grid. Default.

`iopt_hydro_impl` Selects type of time integration scheme.

0: Explicit scheme scheme using mode splitting. Default.

1: Implicit scheme using a multigrid algorithm (but without mode splitting)

`iopt_curr_wfall` When enabled, a correction term is added to the horizontal current components in case a scalar quantity represent particulate matter with a fall/rusing velocity.

0: Disabled. Default.

1: Enabled.

5.1.3 Equation of state

The equations given in the previous sections form a complete set of equations for u , v , ω , ζ , \bar{U} , \bar{V} , T and S , provided that the density ρ , which enters the equations through the baroclinic gradient and the turbulent diffusion coefficients ν_T , λ_T (see Section 5.3 below), is known. Contrary to temperature and salinity, the density is not obtained by an additional transport equation but by means of an equation of state (EOS).

The International EOS (Millero *et al.*, 1980), adopted in the previous version of COHERENS relates the density to the three state variables T , S and p where T is the *in situ* temperature. A more appropriate formulation still based on the International EOS, but with *in situ* temperature replaced by potential temperature was considered by Jacket & McDougall (1995). More recently, McDougall *et al.* (2003) proposed an EOS using potential temperature, which according to the authors is more accurate than the International EOS and computationally more efficient. The latter formulation has therefore been implemented in COHERENS V2.0.

With a precision of 0.003 kg/m³ the density is given by

$$\rho(S, T, p) = P_1(S, T, p)/P_2(S, T, p) \quad (5.62)$$

$$\begin{aligned} P_1 = a_0 &+ a_1 T + a_2 T^2 + a_3 T^3 + a_4 S + a_5 S T + a_6 S^2 \\ &+ a_7 p + a_8 p T^2 + a_9 p S + a_{10} p^2 + a_{11} p^2 T^2 \\ P_2 = 1 &+ b_1 T + b_2 T^2 + b_3 T^3 + b_4 T^4 + b_5 S + b_6 S T + b_7 S T^3 \\ &+ b_8 S^{3/2} + b_9 S^{3/2} T^2 + b_{10} p + b_{11} p^2 T^3 + b_{12} p^3 T \end{aligned} \quad (5.63)$$

Table 5.1: Values of the empirical parameters in the [McDougall *et al.* \(2003\)](#) equation of state.

a_0	999.843699	b_1	$7.28606739 \times 10^{-3}$
a_1	7.3521284	b_2	$-4.60835542 \times 10^{-5}$
a_2	$-5.45928211 \times 10^{-2}$	b_3	$3.68390573 \times 10^{-7}$
a_3	$3.98476704 \times 10^{-4}$	b_4	$1.80809186 \times 10^{-10}$
a_4	2.96938239	b_5	$2.14691708 \times 10^{-3}$
a_5	$-7.23268813 \times 10^{-3}$	b_6	$-9.27062484 \times 10^{-6}$
a_6	$2.12382341 \times 10^{-3}$	b_7	$-1.78343643 \times 10^{-10}$
a_7	$1.04004591 \times 10^{-2}$	b_8	$4.76534122 \times 10^{-6}$
a_8	$1.03970529 \times 10^{-7}$	b_9	$1.63410736 \times 10^{-9}$
a_9	5.1876188×10^{-6}	b_{10}	$5.30848875 \times 10^{-6}$
a_{10}	$-3.24041825 \times 10^{-8}$	b_{11}	$-3.03175128 \times 10^{-16}$
a_{11}	$-1.2386936 \times 10^{-11}$	b_{12}	$-1.27934137 \times 10^{-17}$

where a_i and b_i are empirical parameters listed in Table 5.1. Neglecting density variations in the water column, p can be approximated by

$$p \simeq P_a - \rho g(z - \zeta) \quad (5.64)$$

The expansion coefficients for temperature and salinity are obtained from (5.62)–(5.63)

$$\beta_T = -\frac{1}{\rho} \frac{\partial \rho}{\partial T} = \frac{1}{P_2} \frac{\partial P_2}{\partial T} - \frac{1}{P_1} \frac{\partial P_1}{\partial T} \quad (5.65)$$

$$\beta_S = \frac{1}{\rho} \frac{\partial \rho}{\partial S} = \frac{1}{P_1} \frac{\partial P_1}{\partial S} - \frac{1}{P_2} \frac{\partial P_2}{\partial S} \quad (5.66)$$

For compatibility with the previous COHERENS version and simple case studies, the model allows to use a simpler linear equations of state, obtained by expanding (5.62)–(5.66) around a reference state

$$\rho = \rho_0 (1 + \beta_S(S - S_r) - \beta_T(T - T_r)) \quad (5.67)$$

where T_r and S_r are constant reference values, and $(\rho_0, \beta_T, \beta_S)$ are obtained from (5.62)–(5.63) with $T = T_r$, $S = S_r$, $p = 0$.

Switches

The following switches, used for the evaluation of density and density gradients, are available:

iopt_dens Selects type of equation of state.

- 0 : The density is set to a uniform value, obtained from (5.62)-(5.63) using constant reference values for T , S and $p = 0$. The expansion coefficients are set to zero. Default.
- 1 : Density is calculated from the linear EOS (5.67). Constant values are taken for the expansion coefficients.
- 2 : ρ , β_T and β_S are calculated from (5.62)–(5.63) with $p = 0$.
- 3 : ρ , β_T and β_S are calculated from (5.62)–(5.63) with a non-zero pressure.

`iopt_dens_grad` Selects the numerical algorithm for discretisation of the baroclinic pressure gradient (see Section 12.3.13 for details).

- 0 : The gradient is set to zero in all momentum equations. Default if `iop_dens=0`.
- 1 : Traditional σ -coordinate (second order) method. Default if `iop_dens>0`.
- 2 : Using the z -level method.
- 3 : The method of [Shchepetkin & McWilliams \(2003\)](#)

5.2 Model equations on reduced grids

5.2.1 Water column (1-D) mode

In case of a water column application the horizontal grid reduces to one singular point so that the grid becomes one-dimensional. The following simplifications are made:

1. Advective and horizontal diffusion terms are set to zero.
2. All components of the horizontal pressure gradient are neglected except for the barotropic surface slope term.
3. The continuity equation is not solved. This means in particular that the physical vertical current is set to zero.

In the absence of an horizontal grid, the model equations can be written using Cartesian coordinates in the horizontal and σ -coordinates in the vertical. The momentum equations (5.19), (5.20) then reduce to

$$\frac{1}{h_3} \frac{\partial}{\partial t} (h_3 u) - 2 f v = -g \frac{\partial \zeta}{\partial x} + \frac{1}{h_3} \frac{\partial}{\partial s} \left(\frac{\nu_T}{h_3} \frac{\partial u}{\partial s} \right) \quad (5.68)$$

$$\frac{1}{h_3} \frac{\partial}{\partial t} (h_3 v) + 2 f u = -g \frac{\partial \zeta}{\partial y} + \frac{1}{h_3} \frac{\partial}{\partial s} \left(\frac{\nu_T}{h_3} \frac{\partial v}{\partial s} \right) \quad (5.69)$$

where

- the Coriolis frequency is defined by specifying the latitude of the location, i.e. $f = 2\Omega \sin(\phi_{ref})$.
- the surface slope and the surface elevation ζ , needed to calculate the total water depth H and the vertical grid spacing h_3 are specified as external “surface” forcing conditions.

The surface slope and elevations are prescribed as the sum of a non-harmonic and harmonic part

$$q^e(t) = q_0^e(t) + \sum_{n=1}^N A_n f_n(t) \cos(V_n(t) + u_n(t) - \varphi_n) \quad (5.70)$$

where q^e represents the external value of $\partial\zeta/\partial x$, $\partial\zeta/\partial y$ or ζ . As explained in Section 5.5, (f_n, u_n) are the nodal amplitude and phase factors, $V_n(t)$ the astronomical phases at Greenwich and (A_n, φ_n) the amplitudes and phases with respect to Greenwich². The one-dimensional version of the scalar transport equation (5.38) is given by

$$\frac{1}{h_3} \frac{\partial}{\partial t} (h_3 \psi) - \frac{1}{h_3} \frac{\partial}{\partial s} (w_s \psi) = \mathcal{P}(\psi) - \mathcal{S}(\psi) + \frac{1}{h_3} \frac{\partial}{\partial s} \left(\frac{\lambda_T^\psi}{h_3} \frac{\partial \psi}{\partial s} \right) \quad (5.71)$$

5.2.2 Depth-averaged (2-D) mode

In depth-averaged mode it is assumed that the 3-D current and all 3-D scalar quantities are depth-independent.

From (5.34) it follows that $\omega = 0$. The hydrodynamic equations are then given by (5.32) and the equations in Section 5.1.2 without the baroclinic terms, i.e.

$$\overline{\delta A}_{h1} = \overline{\delta A}_{h2} = \overline{\delta D}_{h1} = \overline{\delta D}_{h2} = 0 \quad (5.72)$$

The depth-integrated form of the transport equation for a depth-independent scalar ψ is obtained by integrating (5.38) over the vertical

$$\begin{aligned} \frac{\partial}{\partial t} (H \overline{\psi}) + \overline{\mathcal{A}}_{h1}(H \overline{\psi}) + \overline{\mathcal{A}}_{h2}(H \overline{\psi}) &= -(w_s \psi)|_{-h} + F_s^\psi - F_b^\psi + H \left(\mathcal{P}(\overline{\psi}) - \mathcal{S}(\overline{\psi}) \right) \\ &+ \frac{1}{h_1 h_2} \left[\frac{\partial}{\partial \xi_1} \left(\overline{\lambda_H} \frac{h_2}{h_1} \frac{\partial \overline{\psi}}{\partial \xi_1} \right) + \frac{\partial}{\partial \xi_2} \left(\overline{\lambda_H} \frac{h_1}{h_2} \frac{\partial \overline{\psi}}{\partial \xi_2} \right) \right] \end{aligned} \quad (5.73)$$

²Expression (5.70) is analogous to (5.333) applied at open boundaries.

where $(w_s\psi)|_{-h}$ is the bottom deposition flux, F_s^ψ and F_b^ψ are the fluxes at respectively the surface and the bottom, and

$$\overline{\lambda_H} = \int_0^1 \lambda_H h_3 \, ds \quad (5.74)$$

The vertically integrated horizontal diffusion coefficients take the form

$$\overline{\nu_H} = C_m h_1 h_2 H \sqrt{\overline{D}_T^2 + \overline{D}_S^2}, \quad \overline{\lambda_H} = C_s h_1 h_2 H \sqrt{\overline{D}_T^2 + \overline{D}_S^2} \quad (5.75)$$

where \overline{D}_T , \overline{D}_S are defined by (5.57).

5.3 Turbulence schemes

5.3.1 Introduction

The objective of a turbulence scheme is to parameterise the effects of turbulent motions. It is assumed that turbulence is fully developed and in a quasi-equilibrium state. The main characteristics can be described as follows (see e.g. Ferziger, 2005; Kantha & Clayson, 2000a):

- Three-dimensional. In contrast to the mean flow, which may be two-dimensional, turbulent motions are fully three-dimensional. This definition excludes the two-dimensional turbulence of geophysical flows on the meso-scale, mentioned in Section ?? which is parameterised by a horizontal mixing scheme.
- Randomness. Turbulence has a “short-time” memory. This means that turbulent states arising from slight changes in initial, boundary or forcing conditions become uncorrelated in time within short time intervals. As a consequence, the only meaningful way to analyse turbulence is through its statistical properties.
- Broad spectrum. Turbulent motions span a large spectrum of scales in space and time.
- Vorticity. In contrast to waves, turbulent motions are characterised by random vorticity fluctuations. This explains why turbulence can be visualised in laboratory experiments as a spectrum of eddies on different spatial scales. On the largest scales, the eddies extract energy from the mean flow, whereas on the shortest (so-called Kolmogorov) scales this energy is converted into heat by molecular dissipation.

The spatial scales of turbulence, which need to be taken into account in models for the ocean, shelf seas or coastal areas, range from 10^{-3} – 10^2 m and are, except eventually for the largest ones, not resolved by the model. Turbulence schemes need to be developed, based upon the statistical properties of the turbulence spectrum. Starting point are the Navier-Stokes equations and the equations of continuity and temperature³, written for convenience in Cartesian coordinates and tensorial notation

$$\frac{\partial U_i}{\partial t} + U_j \frac{\partial U_i}{\partial x_j} + \epsilon_{ijk} f_j U_k = -\frac{1}{\rho_0} \frac{\partial P}{\partial x_i} + \delta_{i3} b + \nu \sum_j \frac{\partial^2 U_i}{\partial x_j^2} \quad (5.76)$$

$$\frac{\partial U_i}{\partial x_i} = 0 \quad (5.77)$$

$$\frac{\partial T}{\partial t} + U_i \frac{\partial T}{\partial x_i} = \mathcal{S}(T) + k_T \sum_i \frac{\partial^2 T}{\partial x_i^2} \quad (5.78)$$

where summation is performed within each term over repeating indices, U_i are the velocity components, ϵ_{ijk} is 1(-1) if (i,j,k) are in cyclic (anticyclic) order and 0 if any two indices are equal, δ_{ij} is the Kronecker symbol (1 if $i=j$, 0 otherwise), ν and k_T are the kinematic viscosity and molecular diffusivity of heat, P the dynamic pressure⁴ and $f_i = 2\Omega(\cos\phi, 0, \sin\phi)$ is twice the Earth's rotation vector.

All quantities are then decomposed into a mean (designated by an overbar) and a fluctuating turbulent part or

$$U_i = \bar{U}_i + u_i, \quad T = \bar{T} + \theta, \quad P = \bar{P} + \rho_0 \pi, \quad b = \bar{b} + \beta \quad (5.79)$$

where the fluctuating parts have zero means. The averages can be considered as ensemble averages over a large number of turbulent states or as a statistical mean over the full turbulence spectrum. Since the fluctuations of density can be taken as small with respect to their mean value, β and θ can be related by the linearised equation of state

$$\beta = g\beta_T \theta \quad (5.80)$$

The mean flow equations are obtained by substituting (5.79) into (5.76)–(5.78) and taking the average. This gives

$$\frac{\partial \bar{U}_i}{\partial t} + \bar{U}_j \frac{\partial \bar{U}_i}{\partial x_j} + \epsilon_{ijk} f_j \bar{U}_k = -\frac{1}{\rho_0} \frac{\partial \bar{P}}{\partial x_i} + \delta_{i3} \bar{b} + \nu \sum_j \frac{\partial^2 \bar{U}_i}{\partial x_j^2} - \frac{\partial}{\partial x_j} \bar{u}_i u_j \quad (5.81)$$

³In the absence of double-diffusive mixing, which is not implemented in the current model code, the procedure is similar if temperature is replaced by salinity.

⁴The dynamic pressure is defined as the pressure minus its homogeneous hydrostatic part, i.e. $P = p + \rho_0 g(z - \zeta) - P_a$.

$$\frac{\partial \bar{U}_i}{\partial x_i} = 0 \quad (5.82)$$

$$\frac{\partial \bar{T}}{\partial t} + \bar{U}_i \frac{\partial \bar{T}}{\partial x_i} = \mathcal{P}(\bar{T}) + k_T \sum_i \frac{\partial^2 \bar{T}}{\partial x_i^2} - \frac{\partial}{\partial x_i} \bar{u}_i \bar{\theta} \quad (5.83)$$

since $\overline{\mathcal{P}(\bar{T})} = \mathcal{P}(\bar{T})$ in view of (5.45). Equations (5.81)–(5.83) are the same as (5.76)–(5.78) except for the last terms in the momentum and temperature equations which represent the exchange of momentum and heat between the mean and turbulent flows. The two terms appear as a divergence of fluxes of momentum $\bar{u}_i \bar{u}_j$ and temperature $\bar{u}_i \bar{\theta}$.

It remains to find suitable parameterisations for these turbulent fluxes. The oldest approach — dating back from the time of Boussinesq in 1877 — and also the most commonly used, is to model the momentum fluxes like the viscous stresses in laminar flows

$$-\bar{u}_i \bar{u}_j = \nu_T \left(\frac{\partial \bar{U}_i}{\partial x_j} + \frac{\partial \bar{U}_j}{\partial x_i} \right) - \frac{2}{3} k \delta_{ij} \quad (5.84)$$

where

$$k = \frac{1}{2} (\bar{u}^2 + \bar{v}^2 + \bar{w}^2) \quad (5.85)$$

is the kinetic energy of turbulent motions and ν_T is called the eddy viscosity (diffusion) coefficient. In analogy with (5.84) the temperature fluxes are usually parameterised using the down-gradient diffusion hypothesis

$$-\bar{u}_i \bar{\theta} = \lambda_T \frac{\partial \bar{T}}{\partial x_i} \quad (5.86)$$

Despite the similarity with laminar flows, there are fundamental differences between turbulent and laminar diffusion

- For fully developed turbulence, ν_T and λ_T are larger than their laminar counterparts by several orders of magnitude.
- The turbulent diffusion coefficients are of the same order of magnitude whereas the molecular coefficients for momentum, T and S are of the order of $(\nu, k_T, k_S) \sim (10^{-6}, 10^{-7}, 10^{-9}) \text{ m}^2/\text{s}$.
- Turbulence is initiated by instabilities of the mean flow. This means in practice that ν_T and λ_T are not constant but depend on the mean flow properties generating and controlling those instabilities, i.e. the current shear $\partial U_i / \partial x_j$ and the density gradient $\partial \rho / \partial x_i$.

The vertical diffusion terms in the model equations (5.2), (5.3) and (5.5)–(5.6) are then derived by evaluating the flux divergences using (5.84)–(5.86) and making the shallow water approximation. This condition is compatible with the assumption of hydrostatic balance and states that the horizontal (mean flow) scales are larger than the vertical one, or $\partial/\partial x, \partial/\partial y \ll \partial/\partial z$. This gives

$$\begin{aligned} -\frac{\partial}{\partial x} \bar{u^2} - \frac{\partial}{\partial y} \bar{uv} - \frac{\partial}{\partial z} \bar{uw} &\simeq -\frac{\partial}{\partial z} \bar{uw} \simeq \frac{\partial}{\partial z} \nu_T \frac{\partial \bar{U}}{\partial z} \\ -\frac{\partial}{\partial x} \bar{uv} - \frac{\partial}{\partial y} \bar{v^2} - \frac{\partial}{\partial z} \bar{vw} &\simeq -\frac{\partial}{\partial z} \bar{vw} \simeq \frac{\partial}{\partial z} \nu_T \frac{\partial \bar{V}}{\partial z} \\ -\frac{\partial}{\partial x} \bar{u\theta} - \frac{\partial}{\partial y} \bar{v\theta} - \frac{\partial}{\partial z} \bar{w\theta} &\simeq -\frac{\partial}{\partial z} \bar{w\theta} \simeq \frac{\partial}{\partial z} \lambda_T \frac{\partial \bar{T}}{\partial z} \end{aligned} \quad (5.87)$$

since

$$-\bar{uw} \simeq \nu_T \frac{\partial U}{\partial z}, \quad -\bar{vw} \simeq \nu_T \frac{\partial V}{\partial z} \quad (5.88)$$

by application of the shallow water approximation to (5.84). A similar expression applies for the diffusion of salinity. It is clear that (u, v, T, S) in (5.1)–(5.3), (5.5)–(5.6) are now interpreted as statistical averages.

The turbulence, as stated in the beginning of this subsection, is now reduced to the implementation of suitable expressions for the turbulent diffusion coefficients. A large number of schemes, applied for hydraulic engineering and in the geophysical context, are available from the scientific literature. For detailed overviews, the reader is referred to the text books of [Kantha & Clayson \(2000a\)](#); [Pope \(2001\)](#) and the reviews by [Rodi \(1984\)](#); [Burchard \(2002\)](#).

The turbulence models, implemented in COHERENS, fall in three categories of increasing complexity

1. The simplest formulation is to set the diffusion coefficients to constant values

$$\nu_T = \nu_c, \quad \lambda_T = \lambda_c \quad (5.89)$$

where ν_c and λ_c are set by the user. Despite its simplicity, it is recommended not to use this form. Turbulence usually occurs in the surface and bottom boundary layers, which requires spatially varying coefficients.

2. Models using simplified empirical or semi-empirical (algebraic) relations, not derived from a turbulence closure theory.

3. Models obtained from the Reynolds averaged Navier-Stokes (RANS) equations. These models are physically more robust, but have a larger computational overhead. The basic assumptions (5.84) and (5.86) are then not assumed *a priori* but derived *a posteriori* from theory.

Switches

The general type of turbulence scheme is selected with the switch `iopt_vdif_coef`:

- 0 : Vertical diffusion is set to zero.
- 1 : Constant diffusion coefficients.
- 2 : Algebraic schemes (Section 5.3.2).
- 3 : RANS models (Section 5.3.3).

5.3.2 Algebraic schemes

5.3.2.1 Richardson number dependent formulations

The Richardson number is defined by

$$Ri = \frac{N^2}{M^2} \quad (5.90)$$

where

$$N^2 = \frac{1}{h_3} \frac{\partial b}{\partial s} = \frac{g}{h_3} \left(\beta_T \frac{\partial T}{\partial s} - \beta_S \frac{\partial S}{\partial s} \right) \quad (5.91)$$

$$M^2 = \frac{1}{h_3^2} \left[\left(\frac{\partial u}{\partial s} \right)^2 + \left(\frac{\partial v}{\partial s} \right)^2 \right] \quad (5.92)$$

are the squared buoyancy and shear frequencies. The first one measures the degree of stratification which is stable if $N^2 > 0$ ($Ri > 0$) and unstable if $N^2 < 0$ ($Ri < 0$). The formulations below are based on theoretical and observational evidence that turbulence decreases in the first case and increases in the second one.

Pacanowski & Philander (1981) proposed the following formulation

$$\nu_T = \nu_{0p} f_p^{n_p}(Ri) + \nu_{bp} \quad (5.93)$$

$$\lambda_T = \nu_T f_p(Ri) + \lambda_{bp} \quad (5.94)$$

$$f_p(Ri) = \min \left[(1 + \alpha_p Ri)^{-1}, \nu_{max}^{1/n_p} \right] \quad (5.95)$$

An upper limit has been imposed on f_p to prevent that turbulence becomes too large in the case of unstable stratification ($Ri < 0$).

The following default values are used⁵

$$\nu_{0p} = 10^{-2}, n_p = 2, \alpha_p = 5, \nu_{bp} = 10^{-4}, \lambda_{bp} = 10^{-5}, \nu_{max} = 3 \quad (5.96)$$

The upper bounds for ν_T and λ_T are then given by 0.03, respectively 0.052.

The scheme has been primarily developed for application in global ocean models (e.g. [Semtner & Chervin, 1988](#)). It has the advantage of being less sensitive to vertical resolution than the more advanced turbulence closures discussed in Section 5.3.3. In the absence of stratification the coefficients take uniform values which makes the scheme less reliable for the study of neutral tidal and wind-driven flows. Test simulations in the Rhine plume ([Ruddick et al., 1995](#)) showed that the results are sensitive to a calibration of the model constants. [Peters et al. \(1988\)](#) derived a similar formulation using different values of the parameters calibrated from microstructure measurements in the Pacific Ocean.

The second scheme use the historical empirical relations proposed by [Munk & Anderson \(1948\)](#):

 $\nu_T = \nu_{0m} f_m(Ri) + \nu_b \quad (5.97)$

$$\lambda_T = \nu_{0m} g_m(Ri) + \lambda_b \quad (5.98)$$

with

$$f_m(Ri) = \min[(1 + \alpha_m Ri)^{-n_1}, \nu_{max}] \quad (5.99)$$

$$g_m(Ri) = \min[(1 + \beta_m Ri)^{-n_2}, \lambda_{max}] \quad (5.100)$$

and ν_b and λ_b are uniform background mixing coefficients selected by the user. The following default parameter values taken

$$\nu_{0m} = 0.06, \alpha_m = 10, \beta_m = 3.33, n_1 = 0.5, n_2 = 1.5, \nu_{max} = 3, \lambda_{max} = 4 \quad (5.101)$$

5.3.2.2 flow-dependent formulations

In shelf and coastal seas tides are a prominent source of turbulence. Observations in the Irish Sea ([Bowden et al., 1959](#)) indicate that the eddy viscosity is proportional to the magnitude of the tidal current. A suitable parameterisation for tidal flow can then be written as

$$\nu_T = (\alpha(\xi_1, \xi_2, t) \Phi(\sigma) + \nu_w) f_m(Ri) + \nu_b \quad (5.102)$$

⁵Note that ν_{0p} , ν_{bp} , λ_{bp} and ν_{0m} in (5.96) and (5.101) are expressed in the same unit as ν_T and λ_T , i.e. m^2/s , whereas ν_{max} , λ_{max} are dimensionless.

$$\lambda_T = (\alpha(\xi_1, \xi_2, t)\Phi(\sigma) + \nu_w)g_m(Ri) + \lambda_b \quad (5.103)$$

The flow field is represented by the depth-independent factor α . Explicit forms are described below. In the absence of stratification the vertical variation of turbulence is presented by a prescribed profile for $\Phi(\sigma)$ which takes account of the reduction of turbulence in the near bottom and surface layers. Following [Davies \(1990\)](#) the following piecewise linear profile is adopted

$$\begin{aligned} \Phi(\sigma) &= \left((1 - r_1)\sigma/\delta_1 + r_1 \right)/D & \text{for } 0 \leq \sigma \leq \delta_1 \\ \Phi(\sigma) &= 1/D & \text{for } \delta_1 \leq \sigma \leq 1 - \delta_2 \\ \Phi(\sigma) &= \left(r_2 - (r_2 - 1)(1 - \sigma)/\delta_2 \right)/D & \text{for } 1 - \delta_2 \leq \sigma \leq 1 \end{aligned} \quad (5.104)$$

where

$$D = 1 + \frac{1}{2}\delta_1(r_1 - 1) + \frac{1}{2}\delta_2(r_2 - 1) \quad (5.105)$$

is a normalisation factor such that the depth-integral of $\Phi(\sigma)$ equals 1. The parameters δ_1, δ_2 are the fractional depths of the bottom and surface layers, and r_1, r_2 the ratios of the bottom and surface values of Φ with respect to the interior value. Default values for the parameters are

$$\delta_1 = \delta_2 = 0, \quad r_1 = r_2 = 1 \quad (5.106)$$

giving a uniform vertical profile. More details about a proper selection of these parameters can be found in e.g. [Davies \(1993\)](#).

In analogy with the formulation used by [Naimie *et al.* \(1994\)](#) for simulating the circulation around Georges Bank the damping functions $f_m(Ri)$ and $g_m(Ri)$ take the form given by the Munk-Anderson expressions (5.99)–(5.100). Following [Glorioso & Davies \(1995\)](#) wind-induced turbulence is related to the surface friction velocity using the simple form

$$\nu_w = \lambda_* u_{*s} \quad (5.107)$$

where λ_* is a constant tunable parameter and the surface friction velocity u_{*s} is given by

$$u_{*s} = \tau_s^{1/2} = (\tau_{s1}^2 + \tau_{s2}^2)^{1/2} \quad (5.108)$$

The last terms on the right of (5.102)–(5.103) are the uniform background eddy viscosity ν_b and diffusivity λ_b .

The following three formulations for the flow factor α can be selected

$$\alpha = K_1(U^2 + V^2)^{1/2} \quad (5.109)$$

$$\alpha = K_2(U^2 + V^2)/(H^2\omega_1) \quad (5.110)$$

$$\alpha = K_1(U^2 + V^2)^{1/2} \Delta_b / H \quad (5.111)$$

where Δ_b measures the thickness of the bottom boundary layer as a function of the bottom friction velocity u_{*b} :

$$\Delta_b = \min(C_\nu u_{*b} / \omega_1, H) \quad (5.112)$$

$$u_{*b} = \tau_b^{1/2} = (\tau_{b1}^2 + \tau_{b2}^2)^{1/2} \quad (5.113)$$

and ω_1 is a characteristic frequency. In shallow areas $\Delta_b = H$ so that (5.111) reduces to (5.109). The following default values are taken

$$K_1 = 2.5 \times 10^{-3}, K_2 = 2 \times 10^{-5}, C_\nu = 2.0, \omega_1 = 10^{-4} \text{ s}^{-1} \quad (5.114)$$

The eddy viscosity parameterisation (5.102) without stratification has been used for the prediction of tidal currents and surface elevations in the Northwest European Continental Shelf (e.g. Davies, 1990; Davies *et al.*, 1997), the Irish and Celtic Seas (e.g. Davies & Jones, 1992; Davies, 1993) and the shelf edge off the West coast of Scotland (Proctor & Davies, 1996).

A parabolic eddy viscosity/diffusivity has been implemented for simplified test case studies

$$\begin{aligned} \nu_T &= \kappa f_m(Ri) u_{*b} H \sigma(1 - \sigma) \\ \lambda_T &= \kappa g_m(Ri) u_{*b} H \sigma(1 - \sigma) \end{aligned} \quad (5.115)$$

where f_m, g_m are the damping functions defined by (5.99)–(5.100).

Switches

An algebraic scheme is taken if `iopt_vdif_coef=2`. The type of scheme is further selected by the switch `iopt_turb_alg`:

- 1 : Pacanowski-Philander
- 2 : Munk-Anderson
- 3 : Flow dependent scheme with α given by (5.109) 
- 4 : Flow dependent scheme with α given by (5.110)
- 5 : Flow dependent scheme with α given by (5.111) 
- 6 Parabolic profile (5.115) 

5.3.3 RANS models

Contrary to the previous schemes, RANS models do not take the eddy viscosity-diffusivity concept as a prior assumption. The turbulent fluxes are derived from a series of transport equations (Section 5.3.3.1). These equations contain unknown second and third-order correlations which need to be parameterised (Section 5.3.3.2). Analytical solutions for geophysical flows and expressions for the eddy viscosity and diffusivity coefficients are derived in Section 5.3.3.3. The solutions contain the turbulent energy k and its dissipation ε as unknown variables. Alternative formulations using a mixing length are given in Section 5.3.3.5. Different schemes are presented in Section 5.3.3.4. Alternative formulations using a mixing length are defined in Section 5.3.3.5. Background mixing schemes are discussed in Section 5.3.3.6.

5.3.3.1 general form of the RANS equations

Equations for the turbulent fluctuations u_i and β are obtained by subtracting the mean equations (5.81)–(5.83) from the non-averaged equations (5.76)–(5.78) and making use of (5.80). One obtains

$$\frac{\partial u_i}{\partial t} + \frac{\partial}{\partial x_j} (U_i u_j + u_i U_j + u_i u_j) + \epsilon_{ijk} f_j u_k = -\frac{\partial \pi}{\partial x_i} + \beta \delta_{i3} + \frac{\partial}{\partial x_j} \overline{u_i u_j} + \nu \sum_j \frac{\partial^2 u_i}{\partial x_j^2} \quad (5.116)$$

$$\frac{\partial u_i}{\partial x_i} = 0 \quad (5.117)$$

$$\frac{\partial \beta}{\partial t} + \frac{\partial}{\partial x_i} (U_i \beta + g \beta_T T u_i + u_i \beta) = k_T \sum_i \frac{\partial^2 \beta}{\partial x_i^2} + \frac{\partial \overline{u_i \beta}}{\partial x_i} \quad (5.118)$$

where the overbar has been omitted for convenience above mean quantities and the temperature equation is converted to an equation for the perturbed buoyancy β by multiplication with the factor $g \beta_T$.

Adding the i -component of (5.116) multiplied by u_j and the j -component multiplied by u_i and taking the average, the following system of equations is obtained for the Reynolds stresses $\overline{u_i u_j}$, making use of the zero divergence condition (5.117)

$$\begin{aligned} \frac{d}{dt} \overline{u_i u_j} + \frac{\partial}{\partial x_k} \overline{u_i u_j u_k} + f_k (\epsilon_{ikl} \overline{u_l u_j} + \epsilon_{jkl} \overline{u_l u_i}) = \\ - \frac{\partial}{\partial x_i} \overline{u_j \pi} - \frac{\partial}{\partial x_j} \overline{u_i \pi} - \overline{u_i u_k} \frac{\partial U_j}{\partial x_k} - \overline{u_j u_k} \frac{\partial U_i}{\partial x_k} \\ + \delta_{i3} \overline{u_j \beta} + \delta_{j3} \overline{u_i \beta} + \pi \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) + \nu \frac{\partial^2}{\partial x_k^2} \overline{u_i u_j} - 2\nu \frac{\partial u_i}{\partial x_k} \frac{\partial u_j}{\partial x_k} \end{aligned}$$

(5.119)

where use is made of the zero divergence condition (5.117) and the total derivative is defined by

$$\frac{d}{dt} = \frac{\partial}{\partial t} + U_k \frac{\partial}{\partial x_k} \quad (5.120)$$

The transport equation for the buoyancy fluxes $\overline{u_i \beta}$ is derived by multiplying (5.116) by β , (5.118) by u_i adding and making the average

$$\begin{aligned} \frac{d}{dt} \overline{u_i \beta} + \frac{\partial}{\partial x_j} \overline{u_i u_j \beta} + \epsilon_{ijk} f_j \overline{u_k \beta} = \\ - \frac{\partial}{\partial x_i} \overline{\beta \pi} - \overline{u_i u_j} \frac{\partial b}{\partial x_j} - \overline{u_j \beta} \frac{\partial U_i}{\partial x_j} + \delta_{i3} \overline{\beta^2} + \overline{\pi} \frac{\partial \beta}{\partial x_i} \\ + \nu \frac{\partial}{\partial x_j} \overline{\beta \frac{\partial u_i}{\partial x_j}} + k_T \frac{\partial}{\partial x_j} \overline{u_i \frac{\partial \beta}{\partial x_j}} - (\nu + k_T) \frac{\partial u_i}{\partial x_j} \frac{\partial \beta}{\partial x_j} \end{aligned} \quad (5.121)$$

The β^2 -equation is obtained by multiplying (5.118) with 2β and averaging

$$\frac{d}{dt} \overline{\beta^2} + \frac{\partial}{\partial x_i} \overline{u_i \beta^2} = -2 \overline{u_i \beta} \frac{\partial b}{\partial x_i} + k_T \sum_i \frac{\partial^2 \overline{\beta^2}}{\partial x_i^2} - 2k_T \sum_i \overline{\left(\frac{\partial \beta}{\partial x_i} \right)^2} \quad (5.122)$$

Equations (5.119) and (5.121)–(5.122) form a complete set of equations for all second-order correlations. An important equation is the one for turbulent kinetic energy k , defined by (5.85) and obtained from (5.119) by taking half its trace

$$\begin{aligned} \frac{dk}{dt} + \frac{\partial}{\partial x_j} \overline{u_i \left(\frac{1}{2} u_i u_j + \pi \right)} - \nu \frac{\partial^2 k}{\partial x_i^2} &= -\overline{u_i u_j} \frac{\partial U_i}{\partial x_j} + \delta_{i3} \overline{u_i \beta} - \nu \overline{\left(\frac{\partial u_i}{\partial x_j} \right)^2} \\ &= P + G - \varepsilon \end{aligned} \quad (5.123)$$

where summation is taken over all indices. The terms on the right have the following meaning:

- P is a source term and equals the energy withdrawn by the “energy containing eddies” at the largest spatial scales of the turbulence spectrum
- G is a buoyancy term which can be shown to be (see below) a sink term for stable stratification ($Ri > 0$) and a source term for an unstable stratification ($Ri < 0$).
- ε represents the dissipation of turbulent energy which occurs at the smallest scales of turbulence.

One main properties of fully developed turbulence is that it adjusts rapidly to a state of equilibrium. This means that the terms on the right hand side are the dominant ones and $P + G \simeq \varepsilon$. A further advantage is that P and G don't contain unknown correlations.

5.3.3.2 parameterisation of the RANS equations

The following parameterisations are adopted for the unknown correlations in (5.119), (5.121) and (5.122).

- All Coriolis terms are set to zero. The main reason is that rotation introduces a large level of complexity. The simplification can be considered as reasonable when the Coriolis period is much larger than the decay time of turbulence or $f\varepsilon/k \ll 1$. For an account of Coriolis effects see [Galperin *et al.* \(1989\)](#); [Kantha *et al.* \(1989\)](#).
- Pressure-strain correlation

$$\begin{aligned} \overline{\pi \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right)} &= -c_1 \frac{\varepsilon}{k} (\overline{u_i u_j} - \frac{2}{3} \delta_{ij} k) - c_{21} (P_{ij} - \frac{2}{3} \delta_{ij} P) \\ &\quad - c_{22} k \left(\frac{\partial U_i}{\partial x_j} + \frac{\partial U_j}{\partial x_i} \right) - c_{23} (D_{ij} - \frac{2}{3} \delta_{ij} D) \\ &\quad - c_3 (G_{ij} - \frac{2}{3} \delta_{ij} G) \end{aligned} \quad (5.124)$$

The first term represents the [Rotta \(1951\)](#) hypothesis of return to isotropy. The tensors in the remaining terms are defined by

$$P_{ij} = -\overline{u_i u_k} \frac{\partial U_j}{\partial x_k} - \overline{u_j u_k} \frac{\partial U_i}{\partial x_k} \quad (5.125)$$

$$D_{ij} = -\overline{u_i u_k} \frac{\partial U_k}{\partial x_j} - \overline{u_j u_k} \frac{\partial U_k}{\partial x_i} \quad (5.126)$$

$$G_{ij} = \delta_{i3} \overline{u_j \beta} + \delta_{j3} \overline{u_i \beta} \quad (5.127)$$

$$P = \frac{1}{2} P_{ii} = -\overline{u_i u_k} \frac{\partial U_i}{\partial x_k} \quad (5.128)$$

$$G = \frac{1}{2} G_{ii} = \delta_{i3} \overline{u_i \beta} = \overline{w \beta} \quad (5.129)$$

- Pressure-buoyancy gradient correlation

$$\overline{\pi \frac{\partial \beta}{\partial x_i}} = -c_{1\beta} \frac{\varepsilon}{k} \overline{u_i \beta} + c_{21\beta} \overline{u_k \beta} \frac{\partial U_i}{\partial x_k} - c_{22\beta} \overline{u_k \beta} \frac{\partial U_k}{\partial x_i} - c_{3\beta} \delta_{i3} \overline{\beta^2} \quad (5.130)$$

Table 5.2: Values of the turbulence constants in the RANS equations according to the different models implemented in COHERENS.

	c_1	c_{21}	c_{22}	c_{23}	c_3	$c_{1\beta}$	$c_{21\beta}$	$c_{22\beta}$	$c_{3\beta}$	R_β
MY82	3.01	0	-0.16	0	0	3.74	0	0	0	0.61
KC94	3.01	0	-0.16	0	0	3.74	0.7	0	0.2	0.61
BB95	1.8	0.6	0	0	0.6	3.0	0.33	0	0.333	0.8
HR82	2.2	0.55	0	0	0.55	3.0	0.5	0	0.5	0.8
CA01	2.49	0.777	0.256	0.207	0.402	5.95	0.8	0.2	0.333	0.72
CA02	2.1	0.803	0.257	0.183	0.576	5.6	0.8	0.2	0.333	0.477

- Dissipation terms

$$2\nu \overline{\frac{\partial u_i}{\partial x_k} \frac{\partial u_j}{\partial x_k}} = \frac{2}{3} \delta_{ij} \varepsilon \quad (5.131)$$

$$2k_T \left(\overline{\frac{\partial \beta}{\partial x_i}} \right)^2 = \chi = \frac{\varepsilon}{k} \overline{\beta^2} \quad (5.132)$$

$$\nu \overline{\frac{\partial}{\partial x_j} \beta \frac{\partial u_i}{\partial x_j}} = k_T \overline{\frac{\partial}{\partial x_j} u_i \frac{\partial \beta}{\partial x_j}} = (\nu + k_T) \overline{\frac{\partial u_i}{\partial x_j} \frac{\partial \beta}{\partial x_j}} = 0 \quad (5.133)$$

Since the laminar diffusion scales are much smaller than the scales of the largest eddies, the laminar terms can be neglected

$$\nu \overline{\frac{\partial^2}{\partial x_k^2} u_i u_j} = 0, \quad k_T \overline{\frac{\partial^2 \beta^2}{\partial x_i^2}} = 0 \quad (5.134)$$

Equation (5.131) states that turbulence energy is dissipated isotropically. Equation (5.132) assumes that the time scale for the buoyancy variance $\overline{\beta^2}/\chi$ is proportional to the one for dissipation of turbulence kinetic energy k/ε . The ratio is given by the parameter R_β .

- Pressure transport is neglected

$$\overline{\frac{\partial}{\partial x_i} u_j \pi} = 0, \quad \overline{\frac{\partial}{\partial x_i} \beta \pi} = 0 \quad (5.135)$$

The expressions for the pressure-strain and pressure-buoyancy gradient are compiled from different sources and presented in their most general form. The parameterisations contain 10 parameters $c_1, c_{21}, c_{22}, c_{23}, c_3, c_{1\beta}, c_{21\beta}, c_{22\beta}, c_{3\beta}, R_\beta$. The values used in COHERENS are taken from different sources: [Mellor & Yamada \(1982\)](#); [Kantha & Clayson \(1994\)](#); [Burchard & Baumert](#)

(1995); Hossain & Rodi (1982) and two schemes taken from Canuto *et al.* (2001). The six models are further denoted by MY82, KC94, BB95, HR82, CA01 and CB01. Values of the parameters are listed in Table 5.2. For readers, familiar with the A_i , B_i , C_i parameters used in MY82 and KC94, the following conversion rules apply

$$\begin{aligned} c_1 &= \frac{B_1}{6A_1}, \quad c_{21} = 0, \quad c_{22} = -2C_1, \quad c_{23} = 0, \quad c_3 = 0 \\ c_{1\beta} &= \frac{B_1}{6A_2}, \quad c_{21\beta} = C_2, \quad c_{22\beta} = 0, \quad c_{3\beta} = C_3, \quad R_\beta = \frac{B_2}{B_1} \end{aligned} \quad (5.136)$$

Different schemes are available for the modelisation of the third-order correlations, total derivative (i.e. time derivative and advective terms) in (5.119) and (5.121)–(5.123). They are based on the classification scheme introduced by Mellor & Yamada (1974, 1982) and Galperin *et al.* (1988).

1. Non-equilibrium or “level 3” method. The left hand sides of (5.121) and (5.122) are set to zero while

$$\frac{d}{dt}\overline{u_i u_j} + \frac{\partial}{\partial x_k}\overline{u_i u_j u_k} = \frac{2}{3}\delta_{ij}(P + G - \varepsilon) \quad (5.137)$$

in (5.119) and

$$\frac{1}{2}\frac{\partial}{\partial x_j}\overline{u_i^2 u_j} = -c_{sk}\frac{\partial}{\partial x_j}\left(\frac{k}{\varepsilon}\overline{u_j u_k}\frac{\partial k}{\partial x_k}\right) \quad (5.138)$$

in (5.123). The expression in the last equation was proposed by Daly & Harlow (1970) and differs somewhat from the one given in the Mellor-Yamada papers. This will be further discussed below.

2. Quasi-equilibrium or “level 2.5” method after Galperin *et al.* (1988). This is the same as previous except that $P+G$ is set to ε in all equations except in the one for turbulent energy. This mean that the left hand side of (5.119) becomes zero as well.
3. Equilibrium or “level 2” method. A full equilibrium, i.e. $P+G=\varepsilon$ in all equations including the k -equation.

A major simplification is additionally achieved by making the boundary layer approximation. This means that horizontal derivatives of mean quantities and the mean vertical current are all set to zero.

5.3.3.3 stability functions

Substituting the parameterisations and approximations, presented in the previous subsection into the equations for the second-order correlations, the problem reduces to solving a system of linear equations. Solutions are presented below for each of the three equilibrium levels.

Non-equilibrium method

The following system of linear equations is obtained where a vertical derivative is denoted by a subscript z and $b_z = N^2$ by its definition (5.91)

$$\begin{aligned}
 \overline{u^2} &= \frac{2}{3} \left[k - \frac{k}{\varepsilon c_1} \left((2 - 2c_{21} + c_{23}) \overline{uw} U_z - (1 - c_{21} - c_{23}) \overline{vw} V_z + (1 - c_3) \overline{w\beta} \right) \right] \\
 \overline{v^2} &= \frac{2}{3} \left[k + \frac{k}{\varepsilon c_1} \left((1 - c_{21} - c_{23}) \overline{uw} U_z - (2 - 2c_{21} + c_{23}) \overline{vw} V_z - (1 - c_3) \overline{w\beta} \right) \right] \\
 \overline{w^2} &= \frac{2}{3} \left[k + \frac{k}{\varepsilon c_1} \left((1 - c_{21} + 2c_{23}) (\overline{uw} U_z + \overline{vw} V_z) + 2(1 - c_3) \overline{w\beta} \right) \right] \\
 \overline{uv} &= -\frac{k}{\varepsilon c_1} (1 - c_{21}) (\overline{uw} V_z + \overline{vw} U_z) \\
 \overline{uw} &= -\frac{k}{\varepsilon c_1} \left[(1 - c_{21}) \overline{w^2} U_z + c_{22} k U_z - c_{23} (\overline{u^2} U_z + \overline{uv} V_z) - (1 - c_3) \overline{u\beta} \right] \\
 \overline{vw} &= -\frac{k}{\varepsilon c_1} \left[(1 - c_{21}) \overline{w^2} V_z + c_{22} k V_z - c_{23} (\overline{v^2} V_z + \overline{uv} U_z) - (1 - c_3) \overline{v\beta} \right] \\
 \overline{u\beta} &= -\frac{k}{\varepsilon c_{1\beta}} \left[\overline{uw} N^2 + (1 - c_{21\beta}) \overline{w\beta} U_z \right] \\
 \overline{v\beta} &= -\frac{k}{\varepsilon c_{1\beta}} \left[\overline{vw} N^2 + (1 - c_{21\beta}) \overline{w\beta} V_z \right] \\
 \overline{w\beta} &= -\frac{k}{\varepsilon c_{1\beta}} \left[\overline{w^2} N^2 + c_{22\beta} (\overline{u\beta} U_z + \overline{v\beta} V_z) - (1 - c_{3\beta}) \overline{\beta^2} \right] \\
 \overline{\beta^2} &= -2 \frac{k}{\varepsilon} R_\beta N^2 \overline{w\beta}
 \end{aligned} \tag{5.139}$$

The solutions for the vertical fluxes of momentum and buoyancy, obtained from (5.139), can be written as

$$-\overline{uw} = S_u \frac{k^2}{\varepsilon} \frac{\partial U}{\partial z}, \quad -\overline{vw} = S_v \frac{k^2}{\varepsilon} \frac{\partial V}{\partial z}, \quad -\overline{w\beta} = S_b \frac{k^2}{\varepsilon} N^2 \tag{5.140}$$

Comparing with (5.88) and (5.86) the eddy viscosity and diffusivity are given by

$$\nu_T = S_u \frac{k^2}{\varepsilon}, \quad \lambda_T = S_b \frac{k^2}{\varepsilon} \quad (5.141)$$

Note that these expressions are obtained from the RANS theory and not postulated *a priori*.

The coefficients S_u and S_b are the so-called stability functions and can be expressed as function of the stability parameters

$$\alpha_M = \frac{k^2}{\varepsilon^2} M^2, \quad \alpha_N = \frac{k^2}{\varepsilon^2} N^2 \quad (5.142)$$

The two parameters are the squares of the ratio of respectively the shear and buoyancy frequency with respect to the turbulence frequency and represent the influence of shear and density gradients on the turbulent fluxes. The solutions for S_u and S_b can be written as

$$\begin{aligned} S_u &= (C_{a1} + C_{a2}\alpha_M + C_{a3}\alpha_N)/D \\ S_b &= (C_{a4} + C_{a5}\alpha_M + C_{a6}\alpha_N)/D \\ D &= 1 + C_{a7}\alpha_M + C_{a8}\alpha_N + C_{a9}\alpha_M^2 + C_{a10}\alpha_M\alpha_N + C_{a11}\alpha_N^2 \end{aligned} \quad (5.143)$$

Explicit expressions for the coefficients C_{ai} as function of the RANS parameters are given in Appendix B.

When applying the boundary layer approximation, the diffusion term (5.138) in the k -equation can be written as

$$c_{sk} \frac{\partial}{\partial x_j} \left(\frac{k}{\varepsilon} \overline{u_j u_k} \frac{\partial k}{\partial x_k} \right) \simeq c_{sk} \frac{\partial}{\partial z} \left(\frac{k}{\varepsilon} \overline{w^2} \frac{\partial k}{\partial z} \right) = \frac{\partial}{\partial z} \left(\nu_k \frac{\partial k}{\partial z} \right) \quad (5.144)$$

where the diffusion coefficient is given by

$$\nu_k = c_{sk} \frac{k^2}{\varepsilon} \frac{\overline{w^2}}{k} = S_k \frac{k^2}{\varepsilon} \quad (5.145)$$

The stability coefficient for turbulent energy diffusion is given by

$$S_k = c_{sk} \frac{\overline{w^2}}{k} = \frac{2}{3} c_{sk} \left[1 - \frac{1}{c_1} \left((1 - c_{21} + 2c_{23})\alpha_M S_u + 2(1 - c_3)\alpha_N S_b \right) \right] \quad (5.146)$$

Quasi-equilibrium method

The equations for the second order correlations are the same as in (5.139) except for the components of turbulent energy

$$\overline{u^2} = \frac{2}{3} k \left(1 - \frac{1 - c_{21} - c_{23}}{c_1} \right) - \frac{2k}{\varepsilon c_1} \left[(1 - c_{21}) \overline{u w} U_z + \frac{1}{3} (c_{21} + c_{23} - c_3) \overline{w \beta} \right]$$

$$\begin{aligned}\overline{v^2} &= \frac{2}{3}k\left(1 - \frac{1 - c_{21} - c_{23}}{c_1}\right) - \frac{2k}{\varepsilon c_1}\left[(1 - c_{21})\overline{vw}V_z + \frac{1}{3}(c_{21} + c_{23} - c_3)\overline{w\beta}\right] \\ \overline{w^2} &= \frac{2}{3}k\left(1 - \frac{1 - c_{21} + 2c_{23}}{c_1}\right) + \frac{2k}{3\varepsilon c_1}\left((3 - c_{21} + 2c_{23} - 2c_3)\overline{w\beta}\right)\end{aligned}\quad (5.147)$$

The solutions for the vertical fluxes can be written in the form (5.140)–(5.142), obtained with the non-equilibrium method. Difference is that the stability functions now depend only on α_N . Two cases can be distinguished

1. Case $c_{22\beta} = 0$.

$$\begin{aligned}S_u &= \frac{C_{b1} + C_{b2}\alpha_N}{(1 + C_{b3}\alpha_N)(1 + C_{b4}\alpha_N)} \\ S_b &= \frac{C_{b5}}{1 + C_{b3}\alpha_N}\end{aligned}\quad (5.148)$$

2. Case $c_{22\beta} \neq 0$.

$$S_u = \frac{C_{c1} + C_{c2}\alpha_N S_b}{1 + C_{c3}\alpha_N} \quad (5.149)$$

S_b is obtained as solution of the quadratic equation

$$(C_{c4} + C_{c5}\alpha_N)\alpha_N S_b^2 + (C_{c6} + C_{c7}\alpha_N)S_b + C_{c8} = 0 \quad (5.150)$$

giving

$$\begin{aligned}S_b &= -\frac{(C_{c6} + C_{c7}\alpha_N) + sD^{1/2}}{2(C_{c4} + C_{c5}\alpha_N)\alpha_N} \\ D &= (C_{c6} + C_{c7}\alpha_N)^2 - 4C_{c8}(C_{c4} + C_{c5}\alpha_N)\alpha_N\end{aligned}\quad (5.151)$$

where s is the sign of $C_{c6} + C_{c7}\alpha_N$.

The expression for the turbulent energy stability coefficient now becomes

$$S_k = c_{sk} \frac{\overline{w^2}}{k} = \frac{2c_{sk}}{3c_1} \left(c_1 - 1 + c_{21} - 2c_{23} - (3 - c_{21} + 2c_{23} - 2c_3)\alpha_N S_b \right) \quad (5.152)$$

Equilibrium method

In this case equilibrium between production and dissipation of turbulent energy is assumed in all second moment equation and in the equation of turbulent energy. This means that

$$P + G = \varepsilon \quad \text{or in dimensionless form} \quad S_u\alpha_M - S_b\alpha_N = 1 \quad (5.153)$$

since

$$\frac{P+G}{\varepsilon} = -\frac{\bar{u}\bar{w}U_z + \bar{v}\bar{w}V_z}{\varepsilon} + \frac{\bar{w}\beta}{\varepsilon} = S_u \frac{k^2}{\varepsilon^2} M^2 - S_b \frac{k^2}{\varepsilon^2} N^2 = S_u \alpha_M - S_b \alpha_N \quad (5.154)$$

Using (5.153) and the expressions for S_u and S_b , the following relation between the stability parameters can be derived

$$1 + C_{d1}\alpha_M + C_{d2}\alpha_N + C_{d3}\alpha_M^2 + C_{d4}\alpha_M\alpha_N + C_{d5}\alpha_N^2 = 0 \quad (5.155)$$

which can be rewritten as a quadratic equation for the squared inverse time scale $\kappa^2 = \varepsilon^2/k^2$

$$\kappa^4 + (C_{d1}M^2 + C_{d2}N^2)\kappa^2 + C_{d3}M^4 + C_{d4}M^2N^2 + C_{d5}N^4 = 0 \quad (5.156)$$

If the limit $\kappa = 0$ is taken, then $\alpha_M = \alpha_N = \infty$ so that $S_u = S_b = 0$. From (5.155) one therefore deduces that turbulence ceases when the Richardson number exceeds a critical value Ri_c obtained by setting $\kappa^2 = 0$. This gives

$$\begin{aligned} Ri_c &= -\frac{C_{d4}}{C_{d5}} \quad \text{if} \quad c_{22\beta} = 0 \\ Ri_c &= \frac{\sqrt{C_{d4}^2 - 4C_{d3}C_{d5}} - C_{d4}}{2C_{d5}} \quad \text{if} \quad c_{22\beta} \neq 0 \end{aligned} \quad (5.157)$$

From Table 5.3 it is seen that Ri_c strongly depends on the type of model.

Since $\alpha_M = \alpha_N/Ri$ equation (5.155) can be rewritten as

$$(1 + C_{d2}\alpha_N + C_{d5}\alpha_N^2)Ri^2 + \alpha_N(C_{d1} + C_{d4}\alpha_N)Ri + C_{d3}\alpha_N^2 = 0 \quad (5.158)$$

Solving (5.158) for Ri as a function of α_N and substituting into the expressions (5.148)–(5.151), the stability functions can be expressed as function of Ri only. The dependence of S_u and S_b on Ri is shown in Figure 5.1 (upper row) for the six RANS models. Also shown is the case of stable stratification with limiting conditions which is further discussed in Section 5.3.3.6.

Alternative methods

Besides the expressions given above, the COHERENS program allows to use simpler formulations for the stability coefficients. In the first one the quasi-equilibrium expressions (5.148)–(5.151) are reset to their constant neutral values obtained in the absence of stratification ($\alpha_N = 0$)

$$S_u = S_{u0}, \quad S_b = S_{b0} \quad (5.159)$$

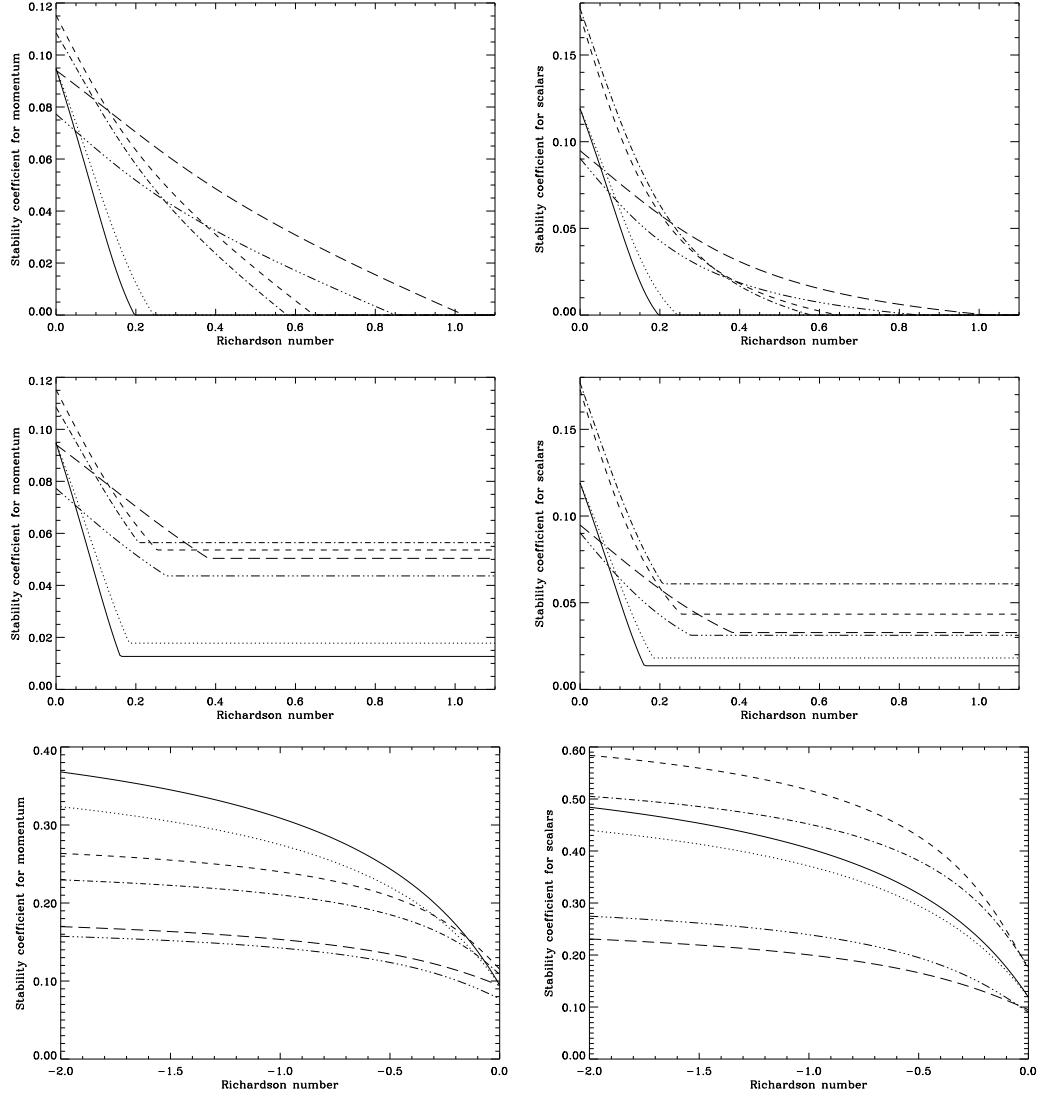


Figure 5.1: Stability coefficients S_u (left column), S_b (right column) as a function of the Richardson number using the equilibrium method for RANS model MY82 (solid), KC94 (dots), BB95 (dashes), HR82 (dash-dots), CA01 (dash and 3 dots), CA02 (long dashes): stable case without limiting condition (upper row), stable case with limiting condition (middle row), unstable case (bottom row).

Table 5.3: Values of critical parameters according to different RANS models.

	S_{u0}	S_{b0}	Ri_c	Ri_c^*	α_{max}	ν_{0l}	λ_{0l}
MY82	0.095	0.119	0.197	0.161	15.35	0.050	0.053
KC94	0.095	0.119	0.244	0.183	12.69	0.063	0.065
BB95	0.115	0.173	0.647	0.252	5.92	0.130	0.106
HR82	0.108	0.177	0.579	0.207	4.72	0.123	0.132
CA01	0.077	0.090	0.851	0.281	7.91	0.123	0.088
CA02	0.094	0.095	1.023	0.388	10.07	0.160	0.104

Values of S_{u0} and S_{b0} are given in Table 5.3⁶. In the second form, S_u and S_b are set to their neutral values, multiplied by the Munk-Anderson damping/amplification factors, defined by (5.99) and (5.100) or

$$S_u = S_{u0}f_m(Ri), \quad S_b = S_{b0}g_m(Ri) \quad (5.160)$$

The latter scheme resembles the one adopted in the earlier standard version of the $k - \varepsilon$ model (Rodⁱ, 1984).

It is remarked that these schemes are physically less robust, but have been implemented in the code for historical reasons or to perform sensitivity experiments related to specific details of the turbulence schemes.

Besides the general expressions (5.146) or (5.152) for the diffusion of turbulent energy, the following simpler alternative options for the stability coefficient S_k are implemented

$$S_k = S_{k0} \quad (5.161)$$

and

$$S_k = S_u/\sigma_k \quad (5.162)$$

where S_{k0} and σ_k are model constants. The first form was introduced in $k - l$ model of Mellor & Yamada (1982), the second in the “standard” $k - \varepsilon$ model (Rodⁱ, 1984), further discussed below. In the former case S_{k0} is related to the Mellor-Yamada parameter S_q by

$$S_{k0} = 2^{1/2}\epsilon_0 S_q \quad (5.163)$$

with $\epsilon_0 = S_{u0}^{3/4}$ (see equation (5.164) below).

⁶Note that S_{u0} is the same as the parameter c_μ used in the standard $k - \varepsilon$ theory (Rodⁱ, 1984).

5.3.3.4 solution methods

The theories presented in Section 5.3.3.3 define the eddy coefficients as function of two turbulent parameters: turbulence energy k and its dissipation rate ε . Such theories, primarily used in hydraulic modelling, are called $k - \varepsilon$ models. The former is determined by solving either the equation for turbulent energy or by the equilibrium relation (5.156). It remains to determine ε appearing in the expressions for the eddy coefficients, stability parameters, k -equation and the equilibrium relation (5.156). Different methods, using either a prescribed analytical expression or a parameterised transport equation, are given below.

Other theories, like the one advocated by Mellor & Yamada (1974, 1982), prefer to use the mixing length l , representative of the spatial scales of the largest (energy-containing) eddies, instead. Such theories are called $k - l$ (or $q^2 - q^2 l$ with $q^2 = 2k$ in the Mellor-Yamada terminology) models. The $k - \varepsilon$ and $k - l$ models are separately discussed below.

$k - \varepsilon$ models

The schemes are divided into three categories depending on the number of transport equations which need to be solved.

1. Zero-equation model. The dissipation rate is determined through the relation

$$\varepsilon = \epsilon_0 \frac{k^{3/2}}{l} \quad (5.164)$$

where l is the mixing length and $\epsilon = S_{u0}^{3/4}$ where S_{u0} is the neutral value of the momentum stability coefficient S_u . The mixing length is determined from one of the available analytical expressions, given below. Turbulence energy is calculated from the equilibrium relation (5.156). The method has the advantage that no transport equation need to be solved in time. The problem is, however, that it may produce numerical instabilities in the time-integration of the model equations. This is illustrated in Section ?? for the test case **pycno**.

2. One-equation model. Turbulence energy is obtained from the k -equation. Inserting the parameterisations of the previous subsection into the source and sinks terms P and G and the diffusion term, this equation, written in transformed coordinates and in its most general form, becomes

$$\frac{1}{h_3} \frac{\partial}{\partial t} (h_3 k) + \mathcal{A}_{h1}(k) + \mathcal{A}_{h2}(k) + \mathcal{A}_v(k) = \frac{1}{h_3} \frac{\partial}{\partial s} \left(\frac{\nu_k}{h_3} \frac{\partial k}{\partial s} \right)$$

$$+ \nu_T M^2 - \lambda_T N^2 - \varepsilon + \mathcal{D}_{sh1}(k) + \mathcal{D}_{sh2}(k) \quad (5.165)$$

with ε given by (5.164) and the advection and diffusion operators defined by (5.22)–(5.24), (5.40)–(5.41). The last two terms represent horizontal diffusion of k . It is noted that advection and horizontal diffusion are usually neglected. This is achieved in the model by the settings of two switches (see below).

3. Two-equation model. Besides the k -equation (5.165), a second transport equation is solved for ε . In transformed coordinates, this equation, including extra horizontal diffusion terms which are usually neglected, is given by (e.g. Rodi, 1984)

$$\begin{aligned} \frac{1}{h_3} \frac{\partial}{\partial t} (h_3 \varepsilon) + \mathcal{A}_{h1}(\varepsilon) + \mathcal{A}_{h2}(\varepsilon) + \mathcal{A}_v(\varepsilon) &= \frac{1}{h_3} \frac{\partial}{\partial s} \left(\frac{\nu_k}{h_3 \sigma_\varepsilon} \frac{\partial \varepsilon}{\partial s} \right) \\ &+ c_{1\varepsilon} \frac{\varepsilon}{k} \left(\nu_T M^2 - c_{3\varepsilon} \lambda_T N^2 \right) - c_{2\varepsilon} \frac{\varepsilon^2}{k} + \mathcal{D}_{sh1}(\varepsilon) + \mathcal{D}_{sh2}(\varepsilon) \end{aligned} \quad (5.166)$$

It should be remarked that, contrary to the k -equation, the ε -equation is derived from a general equation after many parameterisations. The parameters appearing in the (5.166) must satisfy the following constraint, derived from wall layer theory

$$c_{1\varepsilon} - c_{2\varepsilon} = - \frac{\kappa^2 S_{k0}}{\epsilon_0^2 \sigma_\varepsilon} \quad (5.167)$$

where S_{u0} and S_{k0} are the neutral values of S_u and S_k .

$k - l$ models

In $k - l$ theory all equations are written explicitly as function of k and l . This is achieved by substituting l for ε through the relation (5.164). The expressions for the eddy viscosity and diffusion coefficients then take the form

$$\nu_T = S_m k^{1/2} l, \quad \lambda_T = S_h k^{1/2} l, \quad \nu_k = \tilde{S}_k k^{1/2} l \quad (5.168)$$

where the stability functions $(S_m, S_h, \tilde{S}_k) = (S_u, S_b, S_k)/\epsilon_0$ are function of the stability parameters $(G_m, G_h) = (M^2, N^2) l^2/k = \epsilon_0^2 (\alpha_M, \alpha_N)$. Equations (5.143), (5.148), (5.149)–(5.151) and (5.155) remain valid with (α_M, α_N) replaced by (G_m, G_h) , (S_u, S_b) by (S_m, S_h) , κ by $\tilde{\kappa}$ and $(C_{ai}, C_{bi}, C_{ci}, C_{di})$ replaced by $(\tilde{C}_{ai}, \tilde{C}_{bi}, \tilde{C}_{ci}, \tilde{C}_{di})$ using

$$(\tilde{C}_{a1}, \tilde{C}_{a4}) = \epsilon_0 (C_{a1}, C_{a4}), \quad (\tilde{C}_{a2}, \tilde{C}_{a3}, \tilde{C}_{a5}, \tilde{C}_{a6}) = \frac{1}{\epsilon_0} (C_{a2}, C_{a3}, C_{a5}, C_{a6})$$

$$\begin{aligned}
(\tilde{C}_{a7}, \tilde{C}_{a8}) &= \frac{1}{\epsilon_0^2}(C_{a7}, C_{a8}), \quad (\tilde{C}_{a9}, \tilde{C}_{a10}, \tilde{C}_{a11}) = \frac{1}{\epsilon_0^4}(C_{a9}, C_{a10}, C_{a11}) \\
(\tilde{C}_{b1}, \tilde{C}_{b5}) &= \epsilon_0(C_{b1}, C_{b5}), \quad \tilde{C}_{b2} = \frac{C_{b2}}{\epsilon_0}, \quad (\tilde{C}_{b3}, \tilde{C}_{b4}) = \frac{1}{\epsilon_0^2}(C_{b3}, C_{b4}) \\
(\tilde{C}_{c1}, \tilde{C}_{c8}) &= \epsilon_0(C_{c1}, C_{c8}), \quad \tilde{C}_{c6} = C_{c6} \\
(\tilde{C}_{c2}, \tilde{C}_{c3}, \tilde{C}_{c7}) &= \frac{1}{\epsilon_0^2}(C_{c2}, C_{c3}, C_{c7}), \quad \tilde{C}_{c4} = \frac{1}{\epsilon_0^3}C_{c4}, \quad \tilde{C}_{c5} = \frac{1}{\epsilon_0^5}C_{c5} \\
(\tilde{C}_{d1}, \tilde{C}_{d2}) &= \frac{1}{\epsilon_0^2}(C_{d1}, C_{d2}), \quad (\tilde{C}_{d3}, \tilde{C}_{d4}, \tilde{C}_{d5}) = \frac{1}{\epsilon_0^4}(C_{d3}, C_{d4}, C_{d5}) \quad (5.169)
\end{aligned}$$

In case of a two-equation model, the ϵ -equation is replaced by an equation for the quantity kl obtained from the Mellor-Yamada q^2l -equation by setting $q^2 = 2k$:

$$\begin{aligned}
\frac{1}{h_3} \frac{\partial}{\partial t} (h_3 kl) + \mathcal{A}_{h1}(kl) + \mathcal{A}_{h2}(kl) + \mathcal{A}_v(kl) &= \frac{1}{h_3} \frac{\partial}{\partial s} \left(\frac{\nu_k}{h_3 \sigma_{kl}} \frac{\partial}{\partial s} (kl) \right) \\
+ \frac{1}{2} l \left(E_1 \nu_T M^2 - E_3 \lambda_T N^2 \right) - \frac{1}{2} \epsilon_0 k^{3/2} \tilde{W} &+ \mathcal{D}_{sh1}(kl) + \mathcal{D}_{sh2}(kl) \quad (5.170)
\end{aligned}$$

with the wall proximity function \tilde{W} defined by

$$\tilde{W} = 1 + E_2 \left[\frac{l}{\kappa} \left(\frac{1}{H(1-\sigma) + z_{0s}} + \frac{1}{\sigma H + z_{0b}} \right) \right]^2 \quad (5.171)$$

where z_{0s} and z_{0b} are roughness lengths at the surface, respectively the bottom (see below). In analogy with the ϵ -equation the parameters in (5.170)–(5.171) satisfy the constraint

$$E_2 - E_1 + 1 = \frac{2\kappa^2 S_{k0}}{\epsilon_0^2 \sigma_{kl}} \quad (5.172)$$

5.3.3.5 mixing length formulations

A mixing length formulation is needed in the case of a zero- or one-equation model. Four formulations are implemented in the program. The basic requirement in each formulation is that l reduces to the following forms near, respectively, the bottom and the surface

$$l \simeq l_1 = \kappa(\sigma H + z_{0b}), \quad l \simeq l_2 = \kappa(H - \sigma H + z_{0s}) \quad (5.173)$$

where z_{0s} and z_{0b} are surface and bottom roughness lengths.

The first and simplest expression is the parabolic law

$$\frac{1}{l} = \frac{1}{l_1} + \frac{1}{l_2} \quad (5.174)$$

having a maximum at $\sigma \simeq 0.5$.

The second is the “quasi-parabolic” law given by

$$\frac{1}{l} = \frac{1}{l_1} \left(\frac{l_1 + l_2}{l_2} \right)^{1/2} \quad (5.175)$$

which differs from the first one in that l has a maximum at $\sigma \simeq 2/3$ closer to the surface.

The third, recommended by [Xing & Davies \(1996\)](#) has the same form as (5.174) but with l_1 replaced by

$$l_1 = \kappa(\sigma H e^{-\beta_1 \sigma} + z_{0b}) \quad (5.176)$$

allowing for a larger reduction of the mixing length in the lower parts of the water column.

The fourth formulation, initially proposed by [Blackadar \(1962\)](#), has the form

$$\frac{1}{l} = \frac{1}{l_1} + \frac{1}{l_2} + \frac{1}{l_a} \quad (5.177)$$

so that $l \rightarrow l_a$ far from the boundaries. [Mellor & Yamada \(1974\)](#) defined l_a as the ratio of the first to the zeroth moment of the vertical profile of the turbulent velocity scale $k^{1/2}$. Hence

$$l_a = \alpha_1 \int_0^1 (1 - \sigma) k^{1/2} H d\sigma / \int_0^1 k^{1/2} H d\sigma \quad (5.178)$$

5.3.3.6 background mixing

The theory, presented above, is valid for the case of a nearly-isotropic, fully developed turbulence under homogeneous or weakly stratified conditions, but becomes invalid for strongly stratified flows. The reasons are as follows:

- It is known from theory and laboratory experiments that vertical turbulent excursions are impeded by stable stratification. The consequence is that turbulence becomes anisotropic and the assumption of nearly-isotropic turbulence can no longer be maintained.
- Even when turbulence decays for an increasing stable stratification, additional turbulence is generated by the shear and breaking of internal waves which are not resolved by the model.

In the absence of a comprehensive theory of both effects, simple parameterisations have been designed and implemented in the COHERENS code. The methods consist in adding background mixing coefficients to the ones calculated by the RANS scheme. Different schemes are available.

first method

The simplest case are uniform background coefficients for momentum and scalars, selected by the user. The program allows to reset the background value for momentum to the kinematic viscosity ν which is either a user defined or given as function of temperature using the [ITTC \(1978\)](#) expression (7.13).

second method

The second one is based on limiting conditions for turbulence variables. The limitation of vertical overturns by stable stratification is expressed by a limiting condition of the form

$$L_T/L_O < R_l \quad (5.179)$$

where the constant R_l is of $\mathcal{O}(1)$,

$$L_T = -\sqrt{\beta^2}/N^2 \quad (5.180)$$

is the Thorpe scale measuring the extent of the vertical excursions, and

$$L_O = (\varepsilon/N^3)^{1/2} \quad (5.181)$$

the Ozmidov scale at which buoyancy and inertial forces are of comparable magnitude. From theory, laboratory and measurements at sea (see [Luyten et al., 2002](#), and references therein), it is found that $R_l \simeq 1 - 1.3$. Using the last equation of (5.139), (5.140), (5.142) one obtains

$$\left(\frac{L_T}{L_O}\right)^2 = \frac{\beta^2}{N\varepsilon} = 2R_\beta S_b \alpha_N^{3/2} \quad (5.182)$$

Assuming a state of quasi-equilibrium, it can be shown that the right hand side of (5.182) is an increasing function of α_N . The upper limit in (5.179) then implies a maximum value α_{max} for α_N . Its value is obtained by solving

$$2R_\beta S_b \alpha_N^{3/2} = R_l^2 = 2R_\beta R_\star \quad (5.183)$$

for $Z = \alpha_N^{1/2}$ after substitution of (5.148) or (5.151). The result is the polynomial equation

$$C_{b5}Z^3 - R_\star C_{b3}Z^2 - R_\star = 0 \quad (5.184)$$

if $c_{22\beta} = 0$ or

$$\begin{aligned} C_{c5}C_{c8}Z^6 &+ R_*C_{c5}C_{c7}Z^5 + (C_{c4}C_{c8} + R_*^2C_{c5}^2)Z^4 + R_*(C_{c4}C_{c7} + C_{c5}C_{c6})Z^3 \\ &+ 2R_*^2C_{c4}C_{c5}Z^2 + R_*C_{c4}C_{c6}Z + R_*^2C_{c4}^2 = 0 \end{aligned} \quad (5.185)$$

if $c_{22\beta} \neq 0$. Since S_u and S_b are decreasing functions of α_N , the stability functions are limited from below by the constants S_{ulim} and S_{blim} . The theory is illustrated in Figure 5.1 where the evolution of S_u and S_b as function of the Richardson number is shown for the simplified equilibrium method and using an upper limit for α_N .

Substituting the previous limits into the definitions of the eddy coefficients and α_N , the following limiting or background values are obtained

$$\begin{aligned} \nu_T > \nu_{Tlim} &= S_{ulim}\alpha_{max}^{1/2} \frac{k}{N} = \nu_{0l} \frac{k}{N} \\ \lambda_T > \lambda_{Tlim} &= S_{blim}\alpha_{max}^{1/2} \frac{k}{N} = \lambda_{0l} \frac{k}{N} \\ \varepsilon > \varepsilon_{lim} &= \alpha_{max}^{-1/2}kN \\ l < l_{max} &= \epsilon_0\alpha_{max}^{1/2} \frac{k^{1/2}}{N} \end{aligned} \quad (5.186)$$

As explained in [Luyten *et al.* \(2002\)](#) a second limiting condition needs to be imposed for the turbulence energy

$$k > k_{lim} \quad (5.187)$$

yielding background values of the form $\nu_T \sim N^{-1}$, $\lambda_T \sim N^{-1}$ and $\varepsilon \sim N$.

The effect of the limiting condition can be further clarified by making the local equilibrium assumption (5.153). Substituting α_{max} for α_N into equation (5.155) and setting $\alpha_M = \alpha_{max}/Ri$, this equation can be solved for Ri yielding a second critical Richardson number $Ri_c^* < Ri_c$. The eddy coefficients are then given by the previous closure schemes or by the background values (5.186) depending on whether Ri is lower or higher than Ri_c^* . Values of the critical parameters α_{max} , ν_{0l} , λ_{0l} and Ri_c^* are listed in Table 5.3.

third method

The third method is a semi-empirical formulation originally proposed by [Large *et al.* \(1994\)](#)

$$\begin{aligned} \nu_{bT} &= \nu_{T0} + \nu_0^s \left(1 - Ri/Ri_0\right)^{p_1} \\ \lambda_{bT} &= \lambda_{T0} + \nu_0^s \left(1 - Ri/Ri_0\right)^{p_1} \end{aligned} \quad (5.188)$$

where ν_{bT} and λ_{bT} are background mixing coefficients added to the eddy coefficients calculated by the model. The first terms on the right hand side represent mixing due to unresolved internal shear, the second one mixing due to internal wave braking. The parameters have the following default values

$$Ri_0 = 0.7, \quad p_1 = 3, \quad (\nu_0^s, \nu_{T0}, \lambda_{T0}) = (0.005, 10^{-4}, 0.5 \times 10^{-4}) \text{ m}^2/\text{s} \quad (5.189)$$

5.3.3.7 numerical lower bounds

Besides the physical limiting conditions, discussed above, the following numerical lower bounds are always imposed

$$k_{min} = 10^{-14}, \quad \varepsilon_{min} = 10^{-12}, \quad l_{min} = 1.7 \times 10^{-10} \quad (5.190)$$

The reason for adopting this lower limits is to prevent that unrealistically large eddy coefficients are created by rounding errors within the intermittent zone. With the above values and taking $S_u \sim S_b \sim 0.1$ the values of ν_T and λ_T are of the order of $\sim 10^{-17}$ which are clearly much smaller than the corresponding laminar viscosity and diffusivity coefficients.

Switches

A RANS model is selected if `iopt_vdif_coef=3`. A series of additional switches are implemented to determine the specifications of the model.

<code>iopt_kinvisc</code>	Formulation for kinematic viscosity.
	0: user-defined uniform value <code>kinvisc_cst</code>
	1: ITTC (1978) relation (7.13)
<code>iopt_turb_dis_bbc</code>	Type of bottom boundary condition for the dissipation rate ε .
	1: Neumann condition (5.332)
	2: Dirichlet condition (5.330)
<code>iopt_turb_dis_sbc</code>	Type of surface boundary condition for the dissipation rate ε .
	1: Neumann condition (5.248)
	2: Dirichlet condition (5.245)
<code>iopt_turb_iwlim</code>	Type of background mixing scheme as described in Section 5.3.3.6.

	0: using uniform background coefficients 1: using limiting conditions for turbulence parameters 2: the Large <i>et al.</i> (1994) scheme given by (5.188)–(5.189)
iopt_turb_kinvisc	Selects type of background mixing mixing. 0: user-defined constant value vdifmom_cst, vdifscal_cst 1: kinematic viscosity as selected by iopt_kinvisc
iopt_turb_lmix	Mixing length formulation as described in Section 5.3.3.5. 1: parabolic law (5.174) 2: “modified” parabolic law (5.175) 3: “Xing” formulation (5.176) 4: “Blackadar” asymptotic formulation (5.177)
iopt_turb_ntrans	Number of transport equations as described in Section 5.3.3.4. 0: zero-equation model (equilibrium or Mellor-Yamada level 2 method) with a mixing length selected by iopt_turb_lmix 1: turbulence energy equation with a mixing length selected by iopt_turb_lmix 2: $k-\varepsilon$ of $k-kl$ equation depending on the value of iopt_turb_param
iopt_turb_param	Selects type of second turbulent variable. 1: mixing length l ($k-l$ scheme) 2: dissipation rate ε ($k-\varepsilon$ scheme)
iopt_turb_stab_form	Selects type of stability function. 1: constant value (5.159) 2: Munk-Anderson form (5.160) 3: from RANS model as explained in Section 5.3.3.3
iopt_turb_stab_lev	Selects level for stability functions if iopt_turb_stab_form = 3. 1: quasi-equilibrium method (Section 5.3.3.3) 2: non-equilibrium method (Section 5.3.3.3)
iopt_turb_stab_mod	Selects type of closure (RANS) model. 1: MY82-model (Mellor & Yamada, 1982)

	2: KC94-model (Kantha & Clayson, 1994) 3: BB95-model (Burchard & Baumert, 1995) 4: HR82-model (Hossain & Rodi, 1982) 5: CA01-model (Canuto <i>et al.</i>, 2001) 6: CA02-model (Canuto <i>et al.</i>, 2001)
iopt_turb_stab_tke	Formulation for the turbulent diffusion coefficient ν_k (or stability coefficient S_k) of turbulent energy. 1: constant value for S_k as given by equation (5.161) 2: S_k is taken as proportional to momentum stability function S_u as given by (5.162) 3: using the formulation of Daly & Harlow (1970) as given by (5.146) or (5.152) depending on the value of iopt_turb_stab_lev
iopt_turb_tke_bcc	Type of bottom boundary condition for turbulence energy. 1: Neumann condition (5.331) 2: Dirichlet condition (5.330)
iopt_turb_tke_sbc	Type of surface boundary condition for turbulence energy. 1: Neumann condition (5.247) 2: Dirichlet condition (5.245)

Table 5.4 gives an overview of all parameters used in the different schemes and their default values.

Table 5.4: Parameters used in different turbulence schemes (except those listed in Tables 5.2 and 5.3) and their default values.

name	value	unit	purpose
<i>k-l</i> theory			
E_1	1.8	–	constant in the shear production term of the <i>kl</i> -equation (5.170)
E_2	1.33	–	constant in the wall proximity term (5.171) of the <i>kl</i> -equation (5.170)
E_3	1.0	–	constant in the buoyancy source/sink term of the <i>kl</i> -equation (5.170)

(Continued)

Table 5.4: Continued

<i>k-ε theory</i>			
$c_{1\varepsilon}$	1.44	–	constant in the shear production term of the ε -equation (5.166)
$c_{2\varepsilon}$	1.92	–	constant in the dissipation term of the ε -equation (5.166)
$c_{3\varepsilon}$	0.2	–	constant in the buoyancy sink term of the ε -equation (5.166) in case of stable stratification ($N^2 > 0$)
$c_{3\varepsilon}$	1.0	–	constant in the buoyancy source term of the ε -equation (5.166) in case of unstable stratification ($N^2 < 0$)
diffusion coefficients for turbulence variables			
c_{sk}	0.15	–	Daly-Harlow parameter in (5.138)
S_{k0}	0.09	–	neutral value of the stability coefficient S_k in the $k-\varepsilon$ model (see equation (5.161))
S_q	0.2	–	used to determine S_{k0} in the Mellor-Yamada model (see equation (5.163))
σ_k	1.0	–	used to define S_k in (5.162)
σ_{kl}	1.83	–	ratio of the diffusion coefficients in the k - and kl -equations, calculated from (5.172)
σ_ε	1.01	–	ratio of diffusion coefficients in the k - and ε -equations, calculated from (5.167)
limiting conditions			
k_{lim}	10^{-6}	J/kg	background limit for k (see equation (5.187))
k_{min}	10^{-14}	J/kg	numerical lower limit for k
l_{min}	1.7×10^{-10}	m	numerical lower limit for l
ε_{min}	10^{-12}	W/kg	numerical lower limit for ε
background mixing			
Ri_0	0.7	–	critical Richardson number in the Large et al. (1994) background mixing scheme (5.188)
λ_{T0}	5×10^{-5}	m^2/s	internal wave breaking diffusion coefficient for scalars in the Large et al. (1994) background mixing scheme (5.188)
ν_{T0}	10^{-4}	m^2/s	internal wave breaking diffusion coefficient for momentum in the Large et al. (1994) background mixing scheme (5.188)
ν_0^s	0.005	m^2/s	maximum mixing due to unresolved vertical shear in the Large et al. (1994) background mixing scheme (5.188)
boundary conditions			
c_w	0.0	–	surface wave factor used in the surface flux condition (5.247) for turbulent energy
z_{0b}	0.0	m	bottom roughness length in the mixing length formulation (5.173)

(Continued)

Table 5.4: Continued

z_{0s}	0.0	m	surface roughness length in the mixing length formulation (5.173)
<i>mixing length formulations</i>			
α_1	0.2	—	constant in the Blackadar (1962) mixing length formulation (5.178)
β_1	2.0	—	attenuation factor in the Xing & Davies (1996) mixing length formulation (5.176)
<i>algebraic schemes</i>			
n_p	2.0	—	Pacanowski & Philander (1981) scheme (5.93)–(5.95)
α_p	5.0	—	Pacanowski & Philander (1981) scheme (5.93)–(5.95)
λ_{bp}	10^{-5}	m^2/s	Pacanowski & Philander (1981) scheme (5.93)–(5.95)
ν_{bp}	10^{-4}	m^2/s	Pacanowski & Philander (1981) scheme (5.93)–(5.95)
ν_{max}	3.0	—	Pacanowski & Philander (1981) scheme (5.93)–(5.95)
ν_{0p}	0.01	m^2/s	Pacanowski & Philander (1981) scheme (5.93)–(5.95)
n_1	0.5	—	Munk & Anderson (1948) scheme (5.97)–(5.100)
n_2	1.5	—	Munk & Anderson (1948) scheme (5.97)–(5.100)
α_m	10.0	—	Munk & Anderson (1948) scheme (5.97)–(5.100)
β_m	3.33	—	Munk & Anderson (1948) scheme (5.97)–(5.100)
λ_{max}	4.0	—	Munk & Anderson (1948) scheme (5.97)–(5.100)
ν_{max}	3.0	—	Munk & Anderson (1948) scheme (5.97)–(5.100)
ν_{0m}	0.06	m^2/s	Munk & Anderson (1948) scheme (5.97)–(5.100)
C_ν	2.0	—	see equation (5.112)
K_1	0.0025	—	see equations (5.109) and (5.111)
K_2	2×10^{-5}	—	see equation (5.110)
r_1	1.0	—	see equation (5.104)
r_2	1.0	—	see equation (5.104)
δ_1	0.0	—	see equation (5.104)
δ_2	0.0	—	see equation (5.104)
λ_*	0.0	m	see equation (5.107)
ω_1	10^{-4}	s^{-1}	see equation (5.112)

5.4 Horizontal diffusion for scalars

5.4.1 General formulation using rotated diffusion tensor

The horizontal and vertical diffusion terms in the transport equation for a scalar variable ψ can be generally written as

$$\begin{aligned}\mathcal{D}(\psi) &= \nabla \cdot \mathbf{F} \\ &= -\frac{1}{h_1 h_2 h_3} \left(\frac{\partial}{\partial \xi_1} (F_1 h_2 h_3) + \frac{\partial}{\partial \xi_2} (F_2 h_1 h_3) \right) - \frac{1}{h_3} \frac{\partial F_3}{\partial s}\end{aligned}\quad (5.191)$$

where F_i are the diffusive fluxes. In stead of using the terms “horizontal” and “vertical”, diffusion can be generally defined as taking place along and across given surfaces (e.g. constant z -coordinate or density surfaces). The diffusion fluxes can then be written as (Redi, 1982).

$$-F_i = \sum_{j=1}^3 K_{ij} (\nabla \psi)_j \quad (5.192)$$

where

$$\nabla \psi = \left(\frac{1}{h_1} \frac{\partial \psi}{\partial \xi_1}, \frac{1}{h_2} \frac{\partial \psi}{\partial \xi_2}, \frac{1}{h_3} \frac{\partial \psi}{\partial s} \right) \quad (5.193)$$

and \mathbf{K} is the rotated diffusion tensor obtained by a transformation from the geopotential or isopycnal to the model coordinate system

$$\mathbf{K} = \begin{pmatrix} \lambda_H & 0 & -\lambda_H S_1 \\ 0 & \lambda_H & -\lambda_H S_2 \\ \lambda_H S_1 & \lambda_H S_2 & \lambda_H (S_1^2 + S_2^2) + \lambda_T^\psi \end{pmatrix} \quad (5.194)$$

where S_1, S_2 are the normalised components of the slope vector, defined as the unit vector normal to the surface along which diffusion takes place. In deriving (5.194), the small slope approximation, i.e. $S_1, S_2 \ll 1$, has been used except for the K_{33} term, since $\lambda_H \gg \lambda_T^\psi$.

Three different schemes are implemented.

5.4.2 Laplacian diffusion

The simplest scheme is to take

$$S_1 = 0, \quad S_2 = 0 \quad (5.195)$$

giving a diagonal diffusion tensor. This implies that horizontal diffusion takes place along s -coordinate in stead of geopotential surfaces. The assumption is only therefore only valid for smooth bathymetries.

5.4.3 Isolevel diffusion

Isolevel diffusion means that horizontal diffusion occurs along horizontal geopotential (constant z) surfaces. In that case

$$S_1 = \frac{1}{h_1} \frac{\partial z}{\partial \xi_1}, \quad S_2 = \frac{1}{h_2} \frac{\partial z}{\partial \xi_2} \quad (5.196)$$

5.4.4 Isopycnal diffusion

It is now generally believed (Iselin, 1939; Montgomery, 1940) that mixing caused by mesoscale eddies in the ocean takes place along surfaces of constant (potential) density. This type of mixing is usually referred as isopycnal or neutral diffusion. The notions of horizontal and vertical diffusion are now replaced by respectively alongpycnal and diapycnal diffusion. The slopes in the diffusion tensor are then defined by

$$S_1 = -\frac{1}{h_1} \frac{\partial \rho}{\partial \xi_1} \left/ \frac{1}{h_3} \frac{\partial \rho}{\partial s} \right., \quad S_2 = -\frac{1}{h_2} \frac{\partial \rho}{\partial \xi_2} \left/ \frac{1}{h_3} \frac{\partial \rho}{\partial s} \right. \quad (5.197)$$

Previous studies (Cox, 1987; Griffies *et al.*, 1998; Mathieu & Deleersnijder, 1998) showed that the numerical discretisation of isopycnal mixing is a complex issue. This is further discussed in Section ???.

Since $\lambda_T^\psi \ll \lambda_H$, isopycnal diffusion may also be relevant for applications in regional seas where it avoids diffusion with a large diffusion coefficient across salinity plume fronts (in case of weak thermal strafication) and across thermal fronts (in case of weak salinity stratification).

5.4.5 Gent-McWilliams eddy-induced transport

Gent & McWilliams (1990) demonstrated that the parameterisation of mixing by non-resolved eddies in the ocean requires two additional terms in the scalar transport equations. The first is the isopycnal diffusion term given above, the second is an additional advective term with an eddy-induced transport velocity. As shown by Griffies (1998), this transport term can be reformulated in a more convenient form as a skew flux added to the non-diagonal parts of isopycnal diffusion tensor. This approach is adapted in COHERENS as well as in other ocean models (MOMM, ROMS, NEMO).

Combining isopycnal mixing and eddy-induced transport, the diffusion tensor takes the form

$$\mathbf{K} = \begin{pmatrix} \lambda_H & 0 & -\lambda_H S_1 + \kappa_{GM} S_1^* \\ 0 & \lambda_H & -\lambda_H S_2 + \kappa_{GM} S_2^* \\ \lambda_H S_1 - \kappa_{GM} S_1^* & \lambda_H S_2 - \kappa_{GM} S_2^* & \lambda_H (S_1^2 + S_2^2) + \lambda_T^\psi \end{pmatrix} \quad (5.198)$$

where κ_{GM} is a user-defined constant with the unit of a diffusion coefficient. The slope components S_i^* are defined as before by equation (5.197), now referenced with respect to the geopotential surfaces

$$S_i^* = S_i + \frac{1}{h_i} \frac{\partial z}{\partial \xi_i} \quad (5.199)$$

5.4.6 Limiting conditions

The CFL limiting condition is applied for the diagonal components of the diffusion tensor

$$\lambda_H \leq \frac{\min(h_1, h_2)^2}{\Delta t} \quad (5.200)$$

To satisfy the small slope approximation an upper limit of S_{max} is imposed for $|S_i|$ and $|S_i^*|$. Default value is 0.1 which can be considered as reasonable for ocean applications.

In case of isolevel diffusion the slopes are further limited by applying the CFL condition to the cross-diagonal terms

$$S_{1;lim} \leq \frac{h_1 h_3}{2\lambda_H S_1} \quad S_{2;lim} \leq \frac{h_2 h_3}{2\lambda_H S_2} \quad (5.201)$$

A different approach is taken for isopycnal diffusion. Firstly, since it is difficult to determine the values of the slopes in the surface mixed layer (SML) while their values are zero at the surface, a linear interpolation is made within the SML

$$\begin{aligned} S_{i;lim}^* &= -z/d_m S_i^*(z_m) & \text{if } z > z_m \\ S_{i;lim}^*(z) &= S_i^*(z) & \text{if } z \leq z_m \\ S_{i;lim}(z) &= S_{i;lim}^*(z) - \frac{1}{h_i} \frac{\partial z}{\partial \xi_i} \end{aligned} \quad (5.202)$$

The mixed layer depth $d_m = -z_m$ is defined as the first point, taken from the bottom where the difference of the density with respect to its surface value $\rho(z_m) - \rho_s$ becomes smaller than a critical value ρ_{crit} . This is the approach followed in the NEMO model. One of the following limiting conditions can be optionally applied:

1. As used in the NEMO model

$$S_i(z) \leq S_{i;lim}^2(z) / S_{i;old}(z) \quad (5.203)$$

where $S_{i;old}(z)$ are the slopes obtained from (5.197).

2. The [Gerdes *et al.* \(1991\)](#) condition, as described in [Griffies *et al.* \(1998\)](#) and used in the MOMM model

$$\begin{aligned} S_i(z) &= \frac{\Delta_i^2}{S_{i;lim}} && \text{if } |S_{i;lim}| > \Delta_i \\ S_i(z) &= S_{i;lim} && \text{if } |S_{i;lim}| \leq \Delta_i \end{aligned} \quad (5.204)$$

with

$$\Delta_i = \frac{h_i h_3}{2\lambda_H \Delta t} \quad (5.205)$$

Further information can be obtained from the NEMO and MOMM manuals.

Switches

`iopt_hdif_coef` Selects the type of scheme for horizontal diffusion coefficients

- 0: Horizontal diffusion is disabled. Default.
- 1: Constant diffusion coefficients.
- 2: [Smagorinsky \(1963\)](#) formulation.

`iopt_hdif_scal` Selects the type of diffusion scheme for scalars

- 0: Disabled. Default.
- 1: Laplacian diffusion.
- 2: Diffusion along geopotential (z -coordinate) surfaces.
- 3: Diffusion along isoneutral (isopycnal) surfaces.

`iopt_hdif_lim` Selects the type of limiting condition for isopycnal diffusion

- 0: Disabled.
- 1: NEMO limiting condition. Default.
- 2: [Gerdes *et al.* \(1991\)](#).

`iopt_hdif_2D` Disables (0) or enables (1) horizontal in the 2-D momentum equations. Default is 0.

`iopt_hdif_3D` Disables (0) or enables (1) horizontal in the 3-D momentum equations. Default is 0.

5.5 Astronomical tidal force

Tides are generated by the combined gravitational attraction of the sun and the moon. The total force is calculated as the gradient of the tidal potential Φ_{tid} . The potential can be written as a series of tidal harmonics

$$\begin{aligned}
 \frac{\Phi_{tid}}{g} &= \zeta_e = \left(\frac{3}{2} \cos^2 \phi - 1 \right) \sum_{n=1}^{N_0} A_{0n}(t) \cos(V_{0n}(t) + u_{0n}(t)) \\
 &\quad + \sin 2\phi \sum_{n=1}^{N_1} A_{1n}(t) \cos(\lambda + V_{1n}(t) + u_{1n}(t)) \\
 &\quad + \cos^2 \phi \sum_{n=1}^{N_2} A_{2n}(t) \cos(2\lambda + V_{2n}(t) + u_{2n}(t)) \\
 &\quad + \cos^3 \phi \sum_{n=1}^{N_3} A_{3n}(t) \cos(3\lambda + V_{3n}(t) + u_{3n}(t)) \\
 &= \sum_{q=0}^3 G_q(\phi) \sum_{n=1}^{N_q} A_{qn}(t) \cos(q\lambda + V_{qn}(t) + u_{qn}(t)) \quad (5.206)
 \end{aligned}$$

where ζ_e is the equilibrium tide, representing the change of sea level due to the tidal attraction only, i.e. in absence of all other forces (pressure, Coriolis, ...), N_0 , N_1 , N_2 , N_3 are the number of respectively long-period, diurnal, semi-diurnal and terdiurnal tides and q is the species index. Higher order harmonics can be shown to be negligible and are therefore not included in the expansion. Once the tidal potential is known, the components of the tidal force in the momentum equations are obtained using

$$\begin{aligned}
 F_1^t &= \frac{g}{h_1} \frac{\partial \zeta_e}{\partial \xi_1} = \frac{g}{h_1} \left(\frac{\partial \zeta_e}{\partial \lambda} \frac{\partial \lambda}{\partial \xi_1} + \frac{\partial \zeta_e}{\partial \phi} \frac{\partial \phi}{\partial \xi_1} \right) \\
 F_2^t &= \frac{g}{h_2} \frac{\partial \zeta_e}{\partial \xi_2} = \frac{g}{h_2} \left(\frac{\partial \zeta_e}{\partial \lambda} \frac{\partial \lambda}{\partial \xi_2} + \frac{\partial \zeta_e}{\partial \phi} \frac{\partial \phi}{\partial \xi_2} \right) \quad (5.207)
 \end{aligned}$$

The tidal amplitudes are the product of three factors

$$A_{qn}(t) = a_{qn} \alpha_{qn} f_{qn}(t) \quad 0 \leq q \leq 3 \quad (5.208)$$

The first factor a_{qn} is the astronomical tidal amplitude due to the lunar and solar attractive forces. The Earth can be considered as an elastic body and is deformed by the tidal force as well. The effect of the Earth tide is to reduce the astronomical tide and is represented by the second factor α_{qn} . Values

are taken from [Foreman *et al.* \(1993\)](#). The third term is the so-called nodal factor and arises from modulations of the the lunar orbit. The term is always close to 1 and varies with a period of 18.6 years.

The tidal phases are the sum of the geographical factor $q\lambda$, the astronomical argument V_{qn} and the nodal correction factor u_{qn} (further discussed below). The astronomical phase is evaluated at the longitude of Greenwich ($\lambda = 0$). Its time dependence is given by the astronomical ephemerides⁷

$$V_{qn}(t) = i\tau + js + kh + lp + mN + np_s \quad (5.209)$$

where $i = q$ and j, k, l, m, n are integers, called the Doodson numbers ([Doodson, 1921](#)), characterising the constituent, and

τ the mean lunar time

s the mean longitude of the moon

h the mean longitude of the sun

p the mean longitude of the lunar perigee

N the negative mean longitude of the ascending lunar node

p_s the mean longitude of the solar perigee

Since the six astronomical parameters are almost linear time, one can write

$$V_{qn}(t) \simeq V_{qn}(t_{ref}) + (t - t_{ref})\omega_{qn} \quad (5.210)$$

where t_{ref} is some reference time and ω_{qn} the frequency of the tidal constituent. The approximation is valid as long as $|t_{ref} - t| \ll 18.6$ years⁸. In practice, V_{qn} is evaluated using the linear form (5.210) whereby the first term is updated (for a new reference time) with a much larger time step than the second one.

The mean lunar time τ is related to the mean solar time (H) by

$$\tau = H - s + h \quad (5.211)$$

The reference time t_{ref} must always be updated at the same solar time, say H_0 which is usually taken at mid-night Greenwich time when $H_0 = 0$. From (5.209) one obtains

$$V_{qn}(t_{ref}) = H_0 + (j - i)s_0 + (k + i)h_0 + lp_0 + mN_0 + np_{s0} \quad (5.212)$$

⁷An additional phase lag of $\pm 90^\circ$ has to be added for diurnal tides and 0° or 180° for diurnal tides.

⁸Note that the time t must be referenced with respect to Greenwich mean time (GMT). An automatic conversion is made by the program if needed.

and

$$\omega_{qn} = i\dot{\tau}_0 + j\dot{s}_0 + k\dot{h}_0 + l\dot{p}_0 + m\dot{N}_0 + n\dot{p}_{s0} \quad (5.213)$$

where a \cdot (dot) represents a time derivative and the subscript $_0$ evaluation at midnight (GMT). Since (5.213) is only valid when t_{ref} is always taken at the same solar time, the update time interval must be a multiple of days.

The astronomical ephemerides are calculated in time using the reference values at the first of January 0h (GMT) of the year 1900. Explicit expressions (in degrees), taken from [Kantha & Clayson \(2000b\)](#), are

$$\begin{aligned} s_0 &= 270.434358 + 481267.88314137T - 0.001133T^2 + 1.9 \cdot 10^{-6}T^3 \\ h_0 &= 279.69668 + 36000.768925485T + 3.03 \cdot 10^{-4}T^2 \\ p_0 &= 334.329653 + 4069.0340329575T - 0.10325T^2 - 1.2 \cdot 10^{-5}T^3 \\ N_0 &= -259.16 + 1934.14T - 0.0021T^2 \\ p_{s0} &= 281.22083 + 1.71902T + 0.00045T^2 + 3.0 \cdot 10^{-6}T^3 \end{aligned} \quad (5.214)$$

The number of Julian centuries T is given as function of the current year y and the day number within the current year d (between 1 and 366):

$$\begin{aligned} T &= (27393.500528 + 1.0000000356D)/36525 \\ D &= d + 365(y - 1975) + \text{INT}(y - 1973)/4 \end{aligned} \quad (5.215)$$

if the current year is 1975 or later, or

$$\begin{aligned} T &= (0.5 + D)/36525 \\ D &= d + 365(y - 1900) + \text{INT}(y - 1901)/4 \end{aligned} \quad (5.216)$$

for years before 1975. The rate of change of the astronomical ephemerides are obtained from the above equations (e.g. [Schureman, 1941](#))

$$\begin{aligned} \dot{\tau}_0 &= 14.492052^0/\text{h}, \quad \dot{s}_0 = 0.549017^0/\text{h}, \quad \dot{h}_0 = 0.041068^0/\text{h} \\ \dot{p}_0 &= 0.004642^0/\text{h}, \quad \dot{N}_0 = -0.002206^0/\text{h}, \quad \dot{p}_{s0} = 0.000002^0/\text{h} \end{aligned} \quad (5.217)$$

from which the frequencies ω_{qn} are obtained using (5.213).

A total of 56 astronomical tidal constituents are defined within the program. The user is free to select a subset of these frequencies as part of the model setup. The characteristics of all constituents (name, Doodson numbers, frequency, amplitude, Greenwich phase) are given in Table 5.5.

Besides the “main” astronomical constituents, defined in Table 5.5, the harmonic expansion of the tidal potential shows a large number of additional

harmonics with frequencies close to some “main” frequency, but with amplitudes much smaller than the main constituent. Their effect is taken into account through the nodal amplitude and phase factors f_{qn} and u_{qn} . They are determined as follows. Let

$$\zeta_{en} = a_{n0} \cos \theta + \sum_{k=1}^N a_{nk} \cos(\theta + \Delta\theta_k) \quad (5.218)$$

be a cluster of constituents around the main component “ n ” (after omission of the common factor G_q), with amplitude a_{n0} and total phase θ . Setting $\varepsilon = a_{nk}/a_{n0} \ll 1$, one obtains

$$\begin{aligned} \zeta_{en} &= a_{n0} \left(\cos \theta + \sum_k \varepsilon_k \cos(\theta + \Delta\theta_k) \right) \\ &= a_{n0} \left(\cos \theta + \sum_k \varepsilon_k (\cos \theta \cos \Delta\theta_k - \sin \theta \sin \Delta\theta_k) \right) \\ &= a_{n0} \left(\cos \theta (1 + \sum_k \varepsilon_k \cos \Delta\theta_k) - \sin \theta \sum_k \varepsilon_k \sin \Delta\theta_k \right) \\ &= a_{n0} f_n \cos(\theta + u_n) \end{aligned} \quad (5.219)$$

Defining

$$\rho_1 = 1 + \sum_k \varepsilon_k \cos \Delta\theta_k, \quad \rho_2 = \sum_k \varepsilon_k \sin \Delta\theta_k \quad (5.220)$$

the nodal factors are then given by

$$f_n = \sqrt{\rho_1^2 + \rho_2^2}, \quad u_n = \arctan(\rho_2/\rho_1) \quad (5.221)$$

and making a first order Taylor expansion with respect to ε_k . Values of a_{nk} are taken from the tables given by [Cartwright & Tayler \(1971\)](#); [Cartwright & Edden \(1973\)](#).

When the tidal forcing is included in the momentum equations, the tidal solutions for currents and elevations contain additional higher frequencies components. These so-called “overtides” are produced by non-linearities in the model equations and do not appear in the expansion of the tidal potential. For applications in shelf seas, where the astronomical force becomes negligible compared to the bottom stress, the tidal forcing is introduced as an open boundary condition for currents and/or elevations or as a surface boundary condition in case of a water column application. For details see Sections 5.2.1 and 5.10.1. The external forcing is usually presented by an expansion of tidal harmonics which may include over-tides. A list of the most relevant over-tides is given Table 5.6.

Table 5.5: Doodson numbers, frequencies (degrees/h), amplitudes (cm) and Greenwich arguments (degrees) of the tidal constituents which can be used for astronomical and open boundary forcing.

Name	Doodson numbers					ω_{qn}	a_{qn}	α_{qn}	$V_{qn}(t_0)$	
	i	j	k	l	m	n	Long-period tides ($q=0$)			
<i>Long-period tides ($q=0$)</i>										
S_0	0	0	0	0	0	0	0.0000000	19.8419	0.693	0.0
Sa	0	0	1	0	0	-1	0.0410667	0.3103	0.693	$h - p_s$
Ssa	0	0	2	0	0	0	0.0821373	1.9542	0.693	$2h$
058	0	0	3	0	0	-1	0.123204	0.1142	0.693	$3h - p_s$
$Msm0$	1	-2	1	0	0	0	0.4715211	0.4239	0.693	$s - 2h - p$
Mm 0	1	0	-1	0	0	0	0.5443747	2.2191	0.693	$s - p$
Msf 0	2	-2	0	0	0	0	1.0158958	0.3677	0.693	$2s - 2h$
Mf 0	2	0	0	0	0	0	1.0980331	4.2016	0.693	$2s$
083	0	3	-2	1	0	0	1.5695541	0.1526	0.693	$3s - 2h + p$
Mt	0	3	0	-1	0	0	1.6424078	0.8049	0.693	$3s - p$
093	0	4	-2	0	0	0	2.1139288	0.1287	0.693	$4s - 2h$
095	0	4	0	-2	0	0	2.1867825	0.1066	0.693	$4s - 2p$
<i>Diurnal tides ($q=1$)</i>										
α_1	1	-4	2	1	0	0	12.3827651	0.0749	0.693	$-5s + 3h + p - 90^0$
$2Q_1$	1	-3	0	2	0	0	12.8542862	0.2565	0.693	$-4s + h + 2p - 90^0$
σ_1	1	-3	2	0	0	0	12.9271398	0.3098	0.693	$-4s + 3h - 90^0$
Q_1	1	-2	0	1	0	0	13.3986609	1.9387	0.6946	$-3s + h + p - 90^0$
ρ_1	1	-2	2	-1	0	0	13.4715145	0.3685	0.6948	$-3s + 3h - p - 90^0$
O_1	1	-1	0	0	0	0	13.9430356	10.1266	0.695	$-2s + h - 90^0$
τ_1	1	-1	2	0	0	0	14.0251729	0.1325	0.6956	$-2s + 3h + 90^0$
β_1	1	0	-2	1	0	0	14.4145567	0.0749	0.693	$-s - h + p + 90^0$
M_1	1	0	0	1	0	0	14.4966939	0.7965	0.6962	$-s + h + p + 90^0$
χ_1	1	0	2	-1	0	0	14.5695476	0.1522	0.6994	$-s + 3h - p + 90^0$
π_1	1	1	-3	0	0	1	14.9178647	0.2754	0.7027	$-2h + p_s - 90^0$
P_1	1	1	-2	0	0	0	14.9589314	4.7129	0.7059	$-h - 90^0$
S_1	1	1	-1	0	0	1	15.0000000	0.1116	0.7126	$p_s + 90^0$
K_1	1	1	0	0	0	0	15.0410686	14.2408	0.7364	$h + 90^0$
ψ_1	1	1	1	0	0	-1	15.0821353	0.1132	0.5285	$2h - p_s + 90^0$
ϕ_1	1	1	2	0	0	0	15.1232059	0.2028	0.6657	$3h + 90^0$
θ_1	1	2	-2	1	0	0	15.5125897	0.1526	0.6784	$s - h + p + 90^0$
J_1	1	2	0	-1	0	0	15.5854433	0.7965	0.6911	$s + h - p + 90^0$
SO_1	1	3	-2	0	0	0	16.0569644	0.1321	0.693	$2s - h + 90^0$
OO_1	1	3	0	0	0	0	16.1391017	0.4361	0.6925	$2s + h + 90^0$

(Continued)

Table 5.5: Continued

KQ_1	1	4	0	-1	0	0	16.6834764	0.0834	0.693	$3s + h - p + 90^0$
<i>Semi-diurnal tides ($q=2$)</i>										
OQ_2	2	-3	0	3	0	0	27.3509801	0.0695	0.693	$-5s + 2h + 3p$
ε_2	2	-3	2	1	0	0	27.4238337	0.1804	0.693	$-5s + 4h + p$
$2N_2$	2	-2	0	2	0	0	27.8593548	0.6184	0.693	$-4s + 2h + 2p$
μ_2	2	-2	2	0	0	0	27.9682084	0.7463	0.693	$-4s + 4h$
238	2	-2	3	0	0	-1	28.0092751	0.0502	0.693	$-4s + 5h - p_s$
244	2	-1	-1	1	0	1	28.3986628	0.0394	0.693	$-3s + h + p + p_s + 180^0$
N_2	2	-1	0	1	0	0	28.4397295	4.6735	0.693	$-3s + 2h + p$
246	2	-1	1	1	0	-1	28.4807962	0.0436	0.693	$-3s + 3h + p - p_s$
ν_2	2	-1	2	-1	0	0	28.5125831	0.8877	0.693	$-3s + 4h - p$
248	2	-1	3	-1	0	-1	28.5536498	0.0409	0.693	$-3s + 5h - p - p_s$
γ_2	2	0	-2	2	0	0	28.9112506	0.0734	0.693	$-2s + 2p + 180^0$
H_1	2	0	-1	0	0	1	28.9430375	0.0842	0.693	$-2s + h + p_s + 180^0$
M_2	2	0	0	0	0	0	28.9841042	24.4102	0.693	$-2s + 2h$
H_2	2	0	1	0	0	-1	29.0251709	0.0746	0.693	$-2s + 3h - p_s$
λ_2	2	1	-2	1	0	0	29.4556253	0.18	0.693	$-s + p + 180^0$
L_2	2	1	0	-1	0	0	29.5284789	0.6899	0.693	$-s + 2h - p + 180^0$
T_2	2	2	-3	0	0	1	29.9589333	0.6636	0.693	$-h + p_s$
S_2	2	2	-2	0	0	0	30.0000000	11.3572	0.693	0.0
R_2	2	2	-1	0	0	-1	30.0410667	0.095	0.693	$h - p_s + 180^0$
K_2	2	2	0	0	0	0	30.0821373	3.0875	0.693	2h
η_2	2	3	0	-1	0	0	30.6265120	0.1727	0.693	$s + 2h - p$
295	2	4	0	0	0	0	31.1801703	0.0452	0.693	$2s + 2h$
<i>Ter-diurnal tides ($q=3$)</i>										
M_3	3	0	0	0	0	0	43.4761563	0.3455	0.693	$-3s + 3h$

Switches

`iopt_astro_tide` Disables (0) or enables (1) the inclusion of the tidal force in the momentum equations.

Table 5.6: Doodson numbers, origin, frequencies (degrees/h) and Greenwich arguments (degrees) of the overtides which can be used for the open boundary forcing.

Name	Doodson numbers						source	ω_{qn}	$V_{qn}(t_0)$
	i	j	k	l	m	n			
<i>Semi-diurnal tides (q=2)</i>									
$2SM_2$	2	4	-4	0	0	0	$2S_2 - M_2$	31.0158958	$2s - 2h$
<i>Ter-diurnal tides (q=3)</i>									
$2MK_3$	3	-1	0	0	0	0	$2M_2 - K_1$	42.9271398	$-4s + 3h - 90^0$
SO_3	3	1	-2	0	0	0	$S_2 + O_1$	43.9430356	$-2s + h - 90^0$
MK_3	3	1	0	0	0	0	$M_2 + K_1$	44.0251729	$-2s + 3h + 90^0$
SK_3	3	3	-2	0	0	0	$S_2 + K_1$	45.0410686	$h + 90^0$
<i>Quarter-diurnal tides (q=4)</i>									
MN_4	4	-1	0	1	0	0	$M_2 + N_2$	57.4238337	$-5s + 4h$
M_4	4	0	0	0	0	0	$2M_2$	57.9682084	$-4s + 4h$
MS_4	4	2	-2	0	0	0	$M_2 + S_2$	58.9841042	$-2s + 2h$
MK_4	4	2	0	0	0	0	$M_2 + K_2$	59.0662415	$-2s + 4h$
S_4	4	4	-4	0	0	0	$2S_2$	60.0000000	0.0
<i>Sixth-diurnal tides (q=6)</i>									
$2MN_6$	6	1	0	1	0	0	$2M_2 + N_2$	86.4079380	$-7s + 6h + p$
M_6	6	0	0	0	0	0	$3M_2$	86.4079380	$-6s + 6h$
MSN_6	6	1	-2	1	0	0	$M_2 + S_2 + N_2$	87.4238337	$-5s + 4h + p$
$2MS_6$	6	2	-2	0	0	0	$2M_2 + S_2$	87.9682084	$-4s + 4h$
$2SM_6$	6	4	-4	0	0	0	$2S_2 + M_2$	88.9841042	$-2s + 2h$
S_6	6	6	-6	0	0	0	$3S_2$	90.0000000	0.0
<i>Eighth-diurnal tides (q=8)</i>									
M_8	8	0	0	0	0	0	$4M_2$	115.9364169	$-8s + 8h$
$2MSN_8$	1	-2	1	0	0	0	$2M_2 + S_2 + N_2$	116.4079380	$-7s + 6h + p$
$3MS_8$	8	2	-2	0	0	0	$3M_2 + S_2$	116.9523127	$-6s + 6h$
$2(MS)_8$	4	-4	0	0	0	0	$2M_2 + 2S_2$	117.9682084	$-4s + 4h$
S_8	8	8	-8	0	0	0	$4S_2$	120.0000000	0.0

5.6 Surface boundary conditions

5.6.1 General form

Most of the surface boundary conditions discussed in this section are Neumann type conditions for the surface fluxes and can be written into one of the two following general forms

- A prescribed (upwards) surface flux F_s^ψ

$$\frac{\lambda_T^\psi}{h_3} \frac{\partial \psi}{\partial s} = F_s^\psi \quad (5.222)$$

- A surface flux describing the transfer across the surface

$$\frac{\lambda_T^\psi}{h_3} \frac{\partial \psi}{\partial s} = C_s^\psi (\psi_s^e - \psi_s^i) \quad (5.223)$$

where C_s is the transfer rate (with the dimension of a velocity) and ψ_s^e , ψ_s^i are the values of ψ just above and below the surface.

A second form of surface boundary condition is the Dirichlet type where the value of ψ at the surface or at the first interior grid point is specified. Examples are the conditions (5.245) for turbulence.

Note that when the model equations are given in depth-averaged mode (Section 5.2.2), the surface boundary condition enters as an additional flux (source or sink) term in the transport equations. In that case only a Neumann flux conditions is allowed.

5.6.2 Currents

The surface condition for the horizontal current is as usual obtained by specifying the surface stress as function of the wind components

$$\rho_0 \frac{\nu_T}{h_3} \left(\frac{\partial u}{\partial s}, \frac{\partial v}{\partial s} \right) = (\tau_{s1}, \tau_{s2}) = \rho_a C_{ds} W_r (U_{wr}, V_{wr}) \quad (5.224)$$

where $\rho_a = 1.2 \text{ kg/m}^3$ is the air density, C_{ds} the surface drag coefficient discussed in Section 5.7. The wind components and wind speed are taken relative to the sea surface current

$$U_{wr} = U_w - u_s, \quad V_{wr} = V_w - v_s, \quad W_r = \sqrt{U_{wr}^2 + V_{wr}^2} \quad (5.225)$$

where U_w , V_w are the (externally defined) wind components and u_s , v_s the currents at the sea surface.

The boundary condition for the transformed vertical velocity takes the simple form

$$\omega = 0 \quad (5.226)$$

5.6.3 Temperature

The surface boundary condition for temperature can either be taken as a Dirichlet condition in which case T_s is specified directly at (or near) the surface or a Neumann condition in which case the surface temperature flux is given as

$$\rho_0 c_p \lambda_T \frac{1}{h_3} \frac{\partial T}{\partial s} = Q_s \quad (5.227)$$

where Q_s is the downwards directed heat flux at the surface and c_p the specific heat of seawater at constant pressure. The net total heat flux into the water column is composed of a term $-Q_{nsol}$ of all non-solar contributions plus the radiative flux Q_{rad} . Only the former contributes to the surface flux of temperature, since solar radiance is absorbed within the water column.

The (upwards directed) non-solar heat flux has three components, i.e.

$$Q_{nsol} = Q_{la} + Q_{se} + Q_{lw} \quad (5.228)$$

where Q_{la} is the latent heat flux released by evaporation, Q_{se} the sensible heat flux due to the turbulent transport of temperature across the air/sea interface and Q_{lw} the net long-wave radiation emitted at the sea surface. The first two terms are related to the turbulent fluxes of humidity and temperature

$$Q_{la} = \rho_a L_\nu C_e W_r (q_s - q_a) = L_\nu E_{vap} \quad (5.229)$$

$$Q_{se} = \rho_a c_{pa} C_h W_r (T_s - T_a) \quad (5.230)$$

where T_s , q_s and T_a , q_a are the temperature and specific humidity at respectively the sea surface and a reference height, usually taken at 10 m, E_{vap} the evaporation rate and

$$c_{pa} = 1004.6(1 + 0.8375q_a) \text{ J kg}^{-1} (\text{ }^0\text{C})^{-1} \quad (5.231)$$

the specific heat of air at constant pressure. The latent heat of vaporization is given as a function of the sea surface temperature

$$L_\nu = 2.5008 \times 10^6 - 2300T_s \text{ J/kg} \quad (5.232)$$

The air humidity q_a is calculated using

$$q_a = \frac{\epsilon_R v_{pa}}{P_a - (1 - \epsilon_R)v_{pa}} \quad (5.233)$$

where P_a is the atmospheric pressure (in mbar) and $\epsilon_R = 0.62197$ the ratio of molecular weight of water to dry air. The saturated vapour pressure v_{pa} (in

mbar) is provided by [Montheith & Unsworth \(2008\)](#) for positive and [Murray \(1967\)](#) for negative temperatures using Tetens' formula

$$v_{pa} = 6.1078RH \exp\left(\frac{17.269T_a}{T_a + 273.29}\right) \quad \text{if } T_a > 0 \quad (5.234)$$

$$v_{pa} = 6.1078RH \exp\left(\frac{21.875T_a}{T_a + 265.49}\right) \quad \text{if } T_a < 0 \quad (5.235)$$

where RH is the relative humidity (between 0 and 1). The specific humidity at the sea surface q_s is calculated in the same way with T_a, v_{pa} replaced by T_s, v_{ps} and setting $RH = 1$ (relative humidity of 100%).

The surface fluxes of momentum and heat involve the surface drag coefficient C_{ds} and two dimensionless parameters C_e, C_h sometimes referred as the Dalton and Stanton numbers. Various empirical schemes for these transfer coefficients have been presented in the literature (see e.g. [Blanc, 1985](#); [Geernaert, 1990](#)). A few formulations are available in the program. They are further discussed in Section 5.7.

Three formulations are available for the (upward) long-wave radiation flux

1. [Brunt \(1932\)](#)

$$Q_{lw} = \epsilon_s \sigma_{rad} T_{sk}^4 (0.39 - 0.05v_{pa}^{1/2})(1 - 0.6f_c^2) \quad (5.236)$$

where T_{sk} is the sea surface temperature in ^0K , $\epsilon_s = 0.985$ is the emissivity at the sea surface and $\sigma_{rad} = 5.67 \times 10^{-8} \text{ W m}^{-2} \text{ K}^{-4}$ Stefan's constant.

2. [Clark *et al.* \(1974\)](#)

$$Q_{lw} = \epsilon_s \sigma_{rad} T_{sk}^4 (0.39 - 0.05v_{pa}^{1/2})(1 - \beta f_c^2) + 4\epsilon_s \sigma_{rad} T_{sk}^3 (T_s - T_a) \quad (5.237)$$

with $\beta = 0.00422|\phi| + 0.5$ a correction factor for cloud cover where ϕ is the latitude in degrees.

3. [Bignami *et al.* \(1995\)](#)

$$Q_{lw} = \epsilon_s \sigma_{rad} T_{sk}^4 - \sigma T_{ak}^4 (0.635 + 0.00535v_{pa})(1 + 0.1762f_c^2) \quad (5.238)$$

where T_{ak} is the air temperature in ^0K .

Switches

The following model switches are implemented

- iopt_temp_sbc** Selects the type of surface condition for temperature.
- 1: Neumann (flux) condition
 - 2: Dirichlet condition at the first grid point below the surface
 - 3: Dirichlet condition at the surface itself
- iopt_sflux_qlong** Selects the scheme for the long-wave radiation flux
- 1: [Brunt \(1932\)](#)
 - 2: [Clark *et al.* \(1974\)](#)
 - 3: [Bignami *et al.* \(1995\)](#)

5.6.4 Salinity

The surface salinity flux is determined using the formula given by [Steinhorn \(1991\)](#):

$$\rho_0 \frac{\lambda_T}{h_3} \frac{\partial S}{\partial s} = \frac{S_s(E_{vap} - P_{rec})}{1 - 0.001S_s} \quad (5.239)$$

where $E_{vap} = Q_{la}/L_\nu$ and P_{rec} are the evaporation and precipitation rates in $\text{kg m}^{-2} \text{ s}^{-1}$ and S_s the surface salinity in PSU. The evaluation of the surface salinity flux requires the input of precipitation data as an additional meteorological parameter.

The evaporation minus precipitation balance has also an impact on the amount of water in the water column since (fresh) water is lost by evaporation and added by precipitation. This is implemented by (optionally) adding an extra source/sink to the right side of equation (5.32)

$$\frac{\partial \zeta}{\partial t} + \frac{1}{h_1 h_2} \left[\frac{\partial}{\partial \xi_1} (h_2 U) + \frac{\partial}{\partial \xi_2} (h_1 V) \right] = \frac{P_{rec} - E_{vap}}{\rho_0} \quad (5.240)$$

Switches

- iopt_sflux_precip** Selects how precipitation and evaporation data are used in the model
- 0: Disabled. Default.
 - 1: As surface flux for salinity.
 - 2: As source and sink in the 2-D continuity equation.

- 3: As surface flux for salinity and source and sink in the 2-D continuity equation.

The switch `iopt_sal_sbc` enables (1) or disables (0) the surface flux condition (5.239).

5.6.5 Turbulence

The surface boundary conditions for turbulence variables are derived by making the “wall”- (or “log”-) layer approximation. The following assumptions are made

1. The layer is neutrally stratified ($N^2 = 0$).
2. The shear stress is taken as vertically uniform. Neglecting the Coriolis force, taking the X-axis along the flow direction and using equation (5.140) one has

$$u_{\star s}^2 = \frac{1}{\rho_0} \sqrt{\tau_{s1}^2 + \tau_{s2}^2} = -S_{w0} \frac{k^2}{\varepsilon} \frac{\partial U}{\partial z} = \text{constant} \quad (5.241)$$

3. The mixing length is proportional to the distance d from the “wall” boundary as given by equation (5.173):

$$l = l_2 = \kappa d_s = \kappa(\zeta - z + z_{0s}) = \kappa(H(1 - \sigma) + z_{0s}) \quad (5.242)$$

where $\kappa = 0.4$ is von Kármán’s constant and z_{0s} a surface roughness length.

4. The velocity shear is given by

$$\frac{\partial U}{\partial z} = -\frac{u_{\star s}}{l} = -\frac{u_{\star s}}{\kappa d_s} \quad (5.243)$$

Integration of this equation gives the familiar linear U versus $\log z$ dependence.

5. Turbulence is assumed to be in equilibrium, i.e.

$$P = -u_{\star s}^2 \frac{\partial U}{\partial z} = \varepsilon = \epsilon_0 \frac{k^{3/2}}{\kappa d_s} \quad (5.244)$$

From (5.241)–(5.244) the following Dirichlet type surface conditions are derived for the k , ε and kl transport equations

$$k = \frac{u_{*s}^2}{S_{u0}^{1/2}}, \quad \varepsilon = \frac{u_{*s}^3}{\kappa d_s}, \quad l = \kappa d_s \quad (5.245)$$

with d_s defined through (5.242). The relation

$$\epsilon_0 = S_{u0}^{3/4} \quad (5.246)$$

can be readily obtained in addition to the previous relations.

The program allows to use Neumann type condition for k and ε as well. They are given by

$$\frac{\nu_k}{h_3} \frac{\partial k}{\partial s} = c_w u_{*s}^3 \quad (5.247)$$

$$\frac{\nu_k}{h_3 \sigma_\varepsilon} \frac{\partial \varepsilon}{\partial s} = \frac{\nu_k}{h_3 \sigma_\varepsilon} \frac{\epsilon_0 k^{3/2}}{\kappa d_s^2} \quad (5.248)$$

The first condition was proposed by [Craig & Banner \(1994\)](#) with $c_w \sim 100$ and a non-zero surface roughness to represent the energy input of breaking surface waves. The second one, introduced by [Burchard & Petersen \(1999\)](#), can be derived from the Dirichlet conditions and has, according to these authors, a better numerical performance for applications with a coarse vertical resolution. A modification of the flux condition for ε which takes account of wave breaking, was considered by [Burchard \(2001\)](#) but is currently not implemented in the code.

Switches

The surface boundary condition for turbulence are selected by the following switches:

`iopt_turb_tke_sbc` Type of condition for k

- 1: Neumann condition (5.247)
- 2: Dirichlet condition (5.245)

`iopt_turb_dis_sbc` Type of condition for ε

- 1: Neumann condition (5.248)
- 2: Dirichlet condition (5.245)

5.6.6 Meteorological forcing data

A number of meteorological forcing data are required from an external data source. They can be supplied either in non-flux ($U_w, V_w, P_a, T_a, RH, f_c, P_{rec}$) or flux ($\tau_{s1}, \tau_{s2}, P_a, Q_{nsol}, Q_{qsol}$) format.

Switches

iopt_meteo	0: Meteo forcing is disabled. All surface fluxes and surface solar radiation are set to zero. 1: Meteo forcing is enabled.
iopt_meteo_data	Format of meteo input 1: non-flux format 2: flux format
iopt_meteo_stres	Disables (0) or enables (1) input of wind or surface stress data.
iopt_meteo_pres	Disables (0) or enables (1) input of atmospheric pressure data.
iopt_meteo_heat	Disables (0) or enables (1) input of temperature or heat flux data.
iopt_meteo_precip	Selects type of input for precipitation and evaporation 0: No input for precipitation. Evaporation may be calculated from the latent heat flux. 1: Precipitation is provided from input. Evaporation may be calculated from the latent heat flux. 2: Evaporation minus precipitation rate is provided from input.

5.7 Surface drag and exchange coefficients

The values of C_{ds} , C_e and C_h depend on conditions in the lower atmosphere and are, in general, functions of W_r , T_a , T_s , RH (relative humidity) and P_a . Several (mainly empirical) formulations are implemented and can be divided into neutral schemes, depending on wind speed only, and stratified ones which take additionally account of at least the effect of the air minus sea temperature difference.

5.7.1 Neutral formulations

The following formulations for C_{ds} are implemented

1. Generic case

$$\begin{aligned} C_{ds} &= c_1 & \text{if} & \quad W_r < c_4 \\ C_{ds} &= c_2 + c_3 W_r & \text{if} & \quad W_r \geq c_4 \\ C_e &= C_e^s, \quad C_h = C_h^s & \text{if} & \quad T_a \geq T_s \\ C_h &= C_e^u, \quad C_h = C_h^u & \text{if} & \quad T_a < T_s \end{aligned} \quad (5.249)$$

where $c_1, c_2, c_3, c_4, C_e^s, C_e^u, C_h^s, C_h^u$ are user-defined constants and W the relative wind speed at the reference height usually taken at 10 m.

2. Large & Pond (1981)

$$\begin{aligned} C_{ds} &= 0.0014 & \text{if} & \quad W_r \leq 10 \\ C_{ds} &= 10^{-3}(0.49 + 0.065W_r) & \text{if} & \quad W_r > 10 \\ C_e &= 0.00115 \\ C_h &= 0.00066 & \text{if} & \quad T_a \geq T_s \\ C_h &= 0.0013 & \text{if} & \quad T_a < T_s \end{aligned} \quad (5.250)$$

3. Smith & Banke (1975)

$$C_{ds} = 10^{-3}(0.63 + 0.066W_r) \quad (5.251)$$

4. Geernaert *et al.* (1986)

$$C_{ds} = 10^{-3}(0.43 + 0.097W_r) \quad (5.252)$$

C_e and C_h are given by (5.249)

5. Kondo (1975)

$$\begin{aligned} C_{ds} &= 10^{-3}(a_d + b_d W_r^{p_d}) \\ C_e &= 10^{-3}\left(a_e + b_e W_r^{p_e} + c_e(W_r - 8)^2\right) \\ C_h &= 10^{-3}\left(a_h + b_h W_r^{p_h} + c_h(W_r - 8)^2\right) \end{aligned} \quad (5.253)$$

where $a_d, b_d, p_d, a_e, b_e, c_e, p_e, a_h, b_h, c_h$ and p_h are function of the (relative) wind speed and given in Table 5.7.

6. Wu (1980)

$$\begin{aligned}
 C_{ds} &= 0.0012 R_w^{0.15} \\
 C_e &= 0.001 * R_w^{0.11} \\
 C_h &= C_e \\
 \log_{10} R_w &= 0.137 W_r - 0.616
 \end{aligned} \tag{5.254}$$

7. Charnock (1955) relation

$$\begin{aligned}
 z_{0s} g / u_{*s}^2 &= a_{ch} \\
 u_{*s}^2 = C_{ds} W_r^2 &= \left(\frac{\kappa W_r}{\ln(z_{0s}/10)} \right)^2
 \end{aligned} \tag{5.255}$$

where z_{0s} is the surface roughness length, u_* the surface friction velocity and $a_{ch}=0.014$ Charnock's constant. Equations (5.255) have to be solved by iteration. C_e and C_h are given by (5.249)

8. Tuned formulation for the North Sea

$$\begin{aligned}
 C_{ds} &= 0.000621 & \text{if } & W_r \leq 5 \\
 C_{ds} &= 10^{-3}(-0.1325 + 0.151W_r) & \text{if } & 5 < W_r \leq 19.22 \\
 C_{ds} &= 0.002764 & \text{if } & W_r > 19.22
 \end{aligned} \tag{5.256}$$

C_e and C_h are given by (5.249).

9. Large & Yeager (2004)

$$\begin{aligned}
 C_{ds} &= 0.00558 & \text{if } & W_r < 0.5 \\
 C_{ds} &= 10^{-3}(2.7/W + 0.142 + 0.0764W_r) & \text{if } & W_r \geq 0.5 \\
 C_e &= 0.0346 C_{ds}^{1/2} \\
 C_h &= 0.0327 & \text{if } & T_a > T_s \\
 C_h &= 0.0327 & \text{if } & T_a \leq T_s
 \end{aligned} \tag{5.257}$$

5.7.2 Kondo's stratified formulation

Kondo (1975) extended the neutral formulation (5.253) for stratified conditions. The method consists in multiplying the neutral values by a factor depending on the air minus sea temperature difference. The procedure is as follows

$$R_0 = (T_s - T_a)/W_{10}^2 \tag{5.258}$$

Table 5.7: Empirical parameters used in the [Kondo \(1975\)](#) formulations for the neutral surface drag and exchange coefficients.

	a_d, b_d, p_d	a_e, b_e, c_e, p_e	a_h, b_h, c_h, p_h
$W_r < 2.2$	0.0, 1.08, -0.15	0.0, 1.23, 0.0, -0.16	0.0, 1.185, 0.0, -0.157
$2.2 \leq W_r < 5$	0.771, 0.0858, 1.0	0.969, 0.0521, 0.0, 1.0	0.927, 0.0546, 0.0, 1.0
$5 \leq W_r < 8$	0.867, 0.0667, 1.0	1.18, 0.0, 0.0, 1.0	1.15, 0.01, 0.0, 1.0
$8 \leq W_r < 25$	1.2, 0.025, 1.0	1.196, 0.008, -0.0004, 1.0	1.17, 0.0075, -0.00045, 1.0
$25 \leq W_r$	0.0, 0.073, 1.0	1.68, -0.016, 0.0, 1.0	1.652, -0.017, 0.0, 1.0

$$R = R_0 \frac{|R_0|}{|R_0| + 0.01} \quad (5.259)$$

In case of stable conditions ($T_s < T_a$)

$$\begin{aligned} f(R) &= 0.1 + 0.03R + 0.9e^{4.8R} & \text{if } -3.3 < R < 0 \\ f(R) &= 0 & \text{if } R \leq -3.3 \end{aligned} \quad (5.260)$$

$$C_{ds} = C_{dn}f(R), \quad C_e = C_{en}f(R), \quad C_h = C_{hn}f(R) \quad (5.261)$$

For unstable conditions ($T_s > T_a$)

$$\begin{aligned} C_{ds} &= C_{dn}(1.0 + 0.47R^{1/2}) \\ C_e &= C_{en}(1.0 + 0.63R^{1/2}) \\ C_h &= C_{hn}(1.0 + 0.63R^{1/2}) \end{aligned} \quad (5.262)$$

where C_{dn} , C_{en} , C_{hn} are the neutral values in the absence of stratification.

5.7.3 Stratified case from Monin-Obukhov similarity theory

The most general way, but also the most complex one, to include stratification is based on the Monin-Obukhov similarity theory as described in e.g. [Geernaert \(1990\)](#); [Kantha & Clayson \(2000a\)](#). Some details of the physical theory are given here to understand how it is implemented in the program.

The surface values of the momentum, latent and sensible heat fluxes at the air-sea interface depend on the turbulent structure of the lower atmosphere, usually called the planetary boundary layer. It is generally assumed that the fluxes of momentum, heat and specific humidity have nearly constant values within this layer. Following [Monin & Obukhov \(1954\)](#) the structure of this layer can be described by means of a velocity scale u_* (the surface friction velocity), a temperature scale T_* and a humidity scale q_* defined by

$$u_* = (\langle u'w' \rangle^2 + \langle v'w' \rangle^2)^{1/2} \quad (5.263)$$

$$u_* T_* = -\langle w' T' \rangle \quad (5.264)$$

$$u_* q_* = -\langle w' q' \rangle \quad (5.265)$$

where (u', v', w') , T' and q' are the turbulent fluctuations of the wind velocity, temperature and humidity, and $\langle \rangle$ denotes an ensemble average. The vertical gradient of the wind speed U is written as the ratio of u_{*s} to a mixing length, or

$$\frac{\partial U}{\partial z} = \frac{u_*}{l} \quad (5.266)$$

where z is the height above the sea surface. For neutral stratification one has

$$l = \kappa z \quad (5.267)$$

Since l decreases or increases with respect to its neutral value, according as the stratification is stable or unstable, equation (5.266) can be rewritten in the more general form

$$\frac{\partial U}{\partial z} = \frac{u_*}{\kappa z} \phi_m \quad (5.268)$$

The dimensionless function ϕ_m describes the effect of stratification and is smaller (or larger) than 1 for unstable (stable) stratification. In a similar way, the gradients of temperature and relative humidity are given by

$$\frac{\partial T_a}{\partial z} = \frac{T_*}{\kappa z} \phi_h \quad (5.269)$$

$$\frac{\partial q_a}{\partial z} = \frac{q_*}{\kappa z} \phi_q \quad (5.270)$$

The functions ϕ_m , ϕ_h and ϕ_q are expressed in terms of the dimensionless height $\xi = z/L_{mo}$ with the Monin-Obukhov length L_{mo} defined by

$$L_{mo}^{-1} = -\frac{g\kappa}{T_v u_*^3} (\langle w' T' \rangle + 0.61 T_k \langle w' q' \rangle) \quad (5.271)$$

where T_k represents the air temperature in degrees Kelvin and the virtual temperature T_v is given by

$$T_v = T_k (1 + 0.61 q) \quad (5.272)$$

Note that $\xi > 0$ for stable and $\xi < 0$ for unstable stratification. Based upon atmospheric measurements (Businger *et al.*, 1971; Kondo, 1975), the following parameterisations are adopted

$$\begin{aligned} \phi_m &= 1 + \frac{\alpha \xi}{1 + \xi} & \text{for } \xi > 0 \\ \phi_m &= (1 - \beta \xi)^{-1/4} & \text{for } \xi < 0 \end{aligned} \quad (5.273)$$

and

$$\begin{aligned}\phi_h &= \phi_m & \text{for } \xi > 0 \\ \phi_h &= \phi_m^2 & \text{for } \xi < 0\end{aligned}\quad (5.274)$$

with $\alpha = 5$, $\beta = 6$, while it is further assumed that $\phi_q = \phi_h$. Integrating (5.268)–(5.270) one obtains

$$U = \frac{u_*}{\kappa} \left(\ln \frac{z}{z_{0U}} - \psi_m \right) \quad (5.275)$$

$$T_a - T_s = \frac{T_*}{\kappa} \left(\ln \frac{z}{z_{0T}} - \psi_h \right) \quad (5.276)$$

$$q - q_s = \frac{q_*}{\kappa} \left(\ln \frac{z}{z_{0q}} - \psi_h \right) \quad (5.277)$$

where

$$\psi_m = \int_0^\xi \frac{1 - \phi_m(\xi)}{\xi} d\xi \quad (5.278)$$

$$\psi_h = \int_0^\xi \frac{1 - \phi_h(\xi)}{\xi} d\xi \quad (5.279)$$

The subscript s indicates a quantity evaluated at the sea surface. Expressions (5.275)–(5.277) are valid for $z \gg z_{0U}, z_{0T}, z_{0q}$ while it is further assumed that U is much greater than its value at the sea surface. The roughness lengths z_{0U} , z_{0T} and z_{0q} prevent the quantities becoming too large near the surface. Evaluating the integrals (5.278)–(5.279) one has

$$\begin{aligned}\psi_m &= -\alpha \ln(1 + \xi) & \text{for } \xi \geq 0 \\ \psi_m &= 2 \ln(1 + \phi_m^{-1}) + \ln(1 + \phi_m^{-2}) - 2 \arctan(\phi_m^{-1}) + \frac{\pi}{2} - 3 \ln 2 & \text{for } \xi < 0\end{aligned}\quad (5.280)$$

and

$$\begin{aligned}\psi_h &= \psi_m & \text{for } \xi > 0 \\ \psi_h &= 2 \ln(1 + \phi_h^{-1}) - 2 \ln 2 & \text{for } \xi < 0\end{aligned}\quad (5.281)$$

The surface drag coefficient is obtained as follow. From (5.275) one has

$$u_* = \kappa \left(\ln \frac{z}{z_{0U}} - \psi_m \right)^{-1} U = C_{ds}^{1/2} U \quad (5.282)$$

or

$$C_{ds}^{1/2} = \kappa \left(\ln \frac{z}{z_{0U}} - \psi_m \right)^{-1} \quad (5.283)$$

In the neutral case this reduces to

$$C_{dn}^{1/2} = \kappa \left(\ln \frac{z}{z_{0n}} \right)^{-1} \quad (5.284)$$

Following Charnock (1955) one assumes that the roughness length scales with the wind stress or

$$z_{0U} = a_{ch} u_*^2 / g \quad (5.285)$$

so that

$$\frac{z_{0U}}{z_{0n}} = \frac{u_*^2}{u_{*n}^2} = \frac{C_{ds}}{C_{dn}} \quad (5.286)$$

where u_{*n} is the surface friction velocity in neutral conditions. Combining (5.284) and (5.285) the roughness height z_{0U} can be eliminated from (5.283) giving

$$\ln \frac{z}{z_{0U}} = \ln \frac{C_{dn}}{C_{ds}} + \frac{\kappa}{C_{dn}^{1/2}} = \sigma \quad (5.287)$$

The drag coefficient is obtained by substituting (5.287) into (5.283)

$$C_{ds} = \kappa^2 (\sigma - \psi_m)^{-2} \quad (5.288)$$

Expressions for the heat exchange coefficients C_h and C_e are obtained in a similar way. From (5.275) and (5.276) one has

$$u_* T_* = \kappa^2 (\ln \frac{z}{z_{0U}} - \psi_m)^{-1} (\ln \frac{z}{z_{0T}} - \psi_h)^{-1} U (T_a - T_s) = C_h U (T_a - T_s) \quad (5.289)$$

or

$$C_h = \kappa^2 (\ln \frac{z}{z_{0U}} - \psi_m)^{-1} (\ln \frac{z}{z_{0T}} - \psi_h)^{-1} \quad (5.290)$$

giving

$$C_{hn} = \kappa^2 (\ln \frac{z}{z_{0U}})^{-1} (\ln \frac{z}{z_{0T}})^{-1} \quad (5.291)$$

for the neutral case. Using (5.287) the last equation can be rewritten as

$$(\ln \frac{z}{z_{0T}})^{-1} = \frac{C_{hn}}{\kappa^2} \ln \frac{z}{z_{0U}} = \sigma \frac{C_{hn}}{\kappa^2} \quad (5.292)$$

The equation for C_h is derived by substituting (5.291) and (5.292) into (5.290):

$$C_h = C_{hn} \left(1 - \frac{\psi_m}{\sigma} - \frac{C_{hn}}{\kappa^2} \sigma \psi_h + \frac{C_{hn}}{\kappa^2} \psi_m \psi_h \right)^{-1} \quad (5.293)$$

In the same way one finds that

$$C_e = C_{en} \left(1 - \frac{\psi_m}{\sigma} - \frac{C_{en}}{\kappa^2} \sigma \psi_h + \frac{C_{en}}{\kappa^2} \psi_m \psi_h \right)^{-1} \quad (5.294)$$

The dimensionless height ξ can be rewritten as

$$\xi = \frac{g \kappa z_{ref}}{T_v U^2} \frac{C_h (T_a - T_s) + 0.61 T_k C_e (q_a - q_s)}{(C_{ds})^{3/2}} \quad (5.295)$$

The reference height z_{ref} in (5.295) is the height above the sea surface where the meteorological data are supplied. If the reference height for the wind speed (z_r^w) differs from the one for temperature and humidity (z_r^t), wind speed and the neutral exchange coefficients are first reset to their values at $z = z_r^t$:

$$C_{dn}(z_r^t) = \left(\frac{C_{dn}(z_r^w)}{1 + \ln(z_r^t/z_r^w) C_{dn}^{1/2}(z_r^w) / \kappa} \right)^2 \quad (5.296a)$$

$$C_{en}(z_r^t) = C_{en}(z_r^w) \left(\frac{C_{dn}(z_r^t)}{C_{dn}(z_r^w)} \right)^{1/2} \left(1 + \ln \frac{z_r^t}{z_r^w} \frac{C_{en}}{\kappa C_{dn}(z_r^w)^{1/2}} \right)^{-1} \quad (5.296b)$$

$$C_{hn}(z_r^t) = C_{hn}(z_r^w) \left(\frac{C_{dn}(z_r^t)}{C_{dn}(z_r^w)} \right)^{1/2} \left(1 + \ln \frac{z_r^t}{z_r^w} \frac{C_{hn}}{\kappa C_{dn}(z_r^w)^{1/2}} \right)^{-1} \quad (5.296c)$$

$$U(z_r^t) = U(z_r^w) \frac{C_{dn}(z_r^w)}{C_{dn}(z_r^t)} \quad (5.296d)$$

The three coefficients C_{ds} , C_h , C_e are calculated in two steps

1. The neutral values are determined using one of the formulations described in Section 5.7.1.
2. Equations (5.288), (5.293), (5.294) are solved using (5.295) for the three coefficients by iteration. (Only a few iterations are needed).

The various schemes introduced in this section are compared in Figure 5.2. The following observations can be made

- The [Smith & Banke \(1975\)](#) and [Charnock \(1955\)](#) formulations are highly similar. Larger differences between the different schemes, up to a factor 2, are seen for the surface drag coefficient in the case of high wind speeds.
- Stratification effects measured by the air-sea temperature difference are important at wind speeds below 10 m/s. A significant decrease of the exchange coefficients is observed in case of a stable ($T_a > T_s$) stratification, whereas the coefficients increase in the unstable ($T_a < T_s$) case.
- The Kondo and Monin-Obukhov formulations are qualitatively similar.
- The effect of relative humidity is less significant compared to the one produced by stratification.

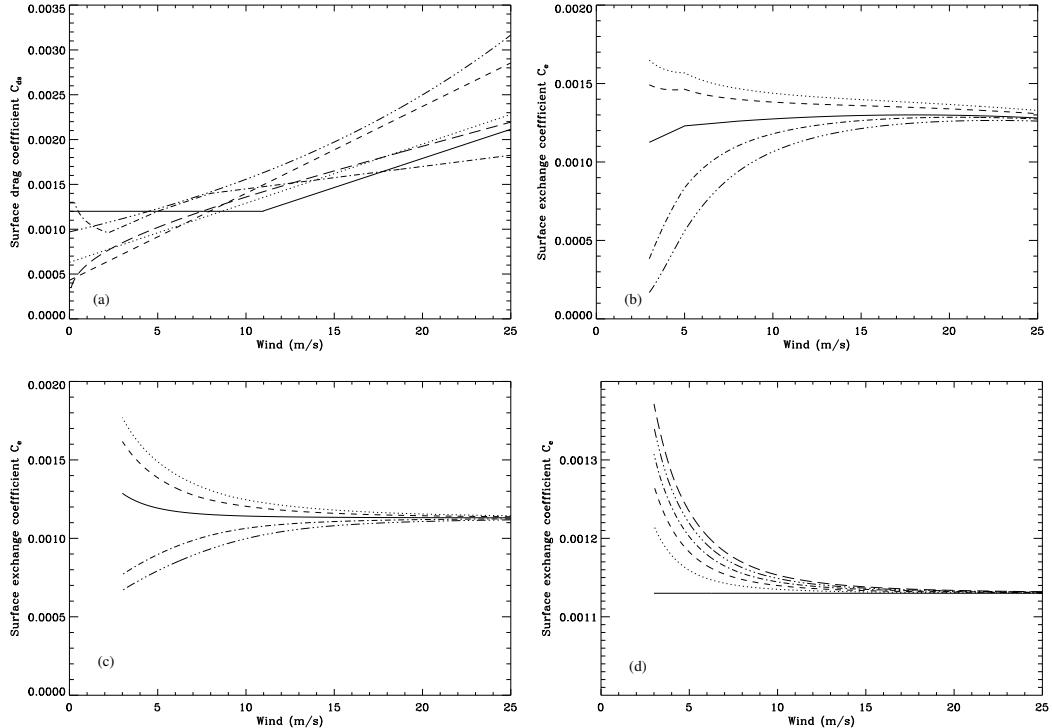


Figure 5.2: (a) Surface drag coefficient C_{ds} as function of wind speed according to (5.250) (solid), (5.251) (dots), (5.252) (dashes), (5.258)–(5.261) (dash-dots), (5.254) (dash and 3 dots), (5.255) (long dashes). (b) Surface exchange coefficient C_e as function of wind speed according to the Kondo (1975) formulation and $\Delta T = T_a - T_s = 0^0\text{C}$ (solid), -5^0C (dots), -2.5^0C (dashes), 2.5^0C (dash-dots), 5^0C (dash and 3 dots). (c) Surface exchange coefficient C_e as function of wind speed according to Monin-Obukhov theory, using $RH = 75\%$, $T_s = 12^0\text{C}$ and $\Delta T = T_a - T_s = 0^0\text{C}$ (solid), -5^0C (dots), -2.5^0C (dashes), 2.5^0C (dash-dots), 5^0C (dash and 3 dots). (d) Surface exchange coefficient C_e as function of wind speed according to Monin-Obukhov theory, using $T_a = T_s = 12^0\text{C}$ and $RH = 100\%$ (solid), 90% (dots), 80% (dashes), 70% (dash-dots), 60% (dash and 3 dots), 50% (long dashes).

Switches

Evaluation of the surface drag and exchange coefficients is selected with the following switches

iop_sflux_pars Formulation for the neutral surface drag and exchange coefficients

- 1 : generic case as given by equation (5.249)
- 2 : equation (5.250) from Large & Pond (1981)
- 3 : equation (5.251) from Smith & Banke (1975)
- 4 : equation (5.252) from Geernaert *et al.* (1986)
- 5 : equation (5.253) from Kondo (1975)
- 6 : equation (5.254) from Wu (1980)
- 7 : equation (5.255) from Charnock (1955)
- 8 : equation (5.256)
- 9 : equation (5.257) from Large & Yeager (2004)

iop_sflux_strat Disables (0) or enables (1) stratification dependence for surface transfer coefficients (as described in Section 5.7.2 or 5.7.3)

5.8 Solar radiation

Deriving a suitable expression for the solar radiation flux is not straightforward in view of its dependence on atmospheric parameters (atmospherical absorption and reflection, water vapour, cloud coverage, albedo of the sea surface). Three formulations, described below, are available within COHERENS.

The radiation entering at the top of the atmosphere is given by

$$Q_s = Q_0 p_{cor} \mathcal{H}(\sin \gamma_{\odot}) \quad (5.297)$$

where $Q_0 = 1367.0 \text{ W/m}^2$ is the solar constant, γ_{\odot} the altitude of the sun and \mathcal{H} the Heaviside function

$$\begin{aligned} \mathcal{H}(x) &= 0 & \text{if } x \leq 0 \\ \mathcal{H}(x) &= x & \text{if } x > 0 \end{aligned} \quad (5.298)$$

The factor p_{cor} represents a correction term due to the elliptical orbit of the earth and is usually expressed as a function of the day number of the year:

$$p_{cor} = a_{10} + \sum_{i=1}^2 a_{1i} \cos \tau + \sum_{i=1}^2 a_{2i} \sin \tau \quad (5.299)$$

with

$$\begin{aligned}(a_{10}, a_{11}, a_{12}) &= (1.00011, 0.034221, 0.000719) \\ (a_{21}, a_{22}) &= (0.00128, 0.000077) \\ \tau &= \frac{2\pi}{365+L}d\end{aligned}\quad (5.300)$$

where L equals 0 for a non-leap and 1 for a leap year and d is the day number of the year (between 0 and 365+ L). The altitude of the sun is calculated from

$$\sin \gamma_{\odot} = \sin \phi \sin \delta_{\odot} + \cos \phi \cos \delta_{\odot} \cos H_{\odot} \quad (5.301)$$

where δ_{\odot} is the declination of the sun, H_{\odot} the sun's hour angle and ϕ the latitude. The angle δ_{\odot} , in radians, is obtained from the series expansion

$$\delta_{\odot} = b_{10} + \sum_{i=1}^3 b_{1i} \cos \tau' + \sum_{i=1}^3 b_{2i} \sin \tau' \quad (5.302)$$

with

$$\begin{aligned}(b_{10}, b_{11}, b_{12}, b_{13}) &= (0.006918, -0.399912, -0.006758, -0.002697) \\ (b_{21}, b_{22}, b_{23}) &= (0.070257, 0.000907, 0.00148) \\ \tau' &= \frac{2\pi}{365+L}(d-1)\end{aligned}\quad (5.303)$$

The sun's hour angle, measured in hours, is computed by

$$H_{\odot} = \frac{\pi}{12}(t_h - 12 + TE) + \frac{\lambda_h}{15} \quad (5.304)$$

where t_h is the hour of the day in GMT, λ_h the longitude and TE the equation of time which can be written as

$$TE = \sum_{i=1}^3 c_{1i} \cos \tau + \sum_{i=1}^3 c_{2i} \sin \tau \quad (5.305)$$

with

$$\begin{aligned}(c_{10}, c_{11}, c_{12}, c_{13}) &= (0.0072, -0.0528, -0.0012) \\ (c_{21}, c_{22}, c_{23}) &= (-0.1229, -0.1565, -0.0041)\end{aligned}\quad (5.306)$$

The downward solar radiation at the surface is calculated from either of the following formulations

- [Rosati & Miyakoda \(1988\)](#)

$$Q_{sol} = \frac{1}{2}(1 - A)Q_s(0.7^{1/\sin\gamma_\odot} + 0.91)(1 - 0.62f_c + 0.0019 \sin\gamma_{\odot,max}) \quad (5.307)$$

where $\gamma_{\odot,max}$ is the sun's altitude at noon, f_c the fractional cloud cover and A the surface albedo split into a clear and oversky value

$$A = A_{cs} + A_{os} = \frac{0.05(1 - f_c)}{1 + \sin\gamma_\odot^{1.4} + 0.15} + 0.06f_c \quad (5.308)$$

- [Zillman \(1972\)](#)

$$Q_{sol} = (1 - A)Q_s \sin\gamma_\odot \left(1.085 \sin\gamma_\odot + 0.001v_{pa}(2.7 + \sin\gamma_\odot) + 0.1 \right)^{-1} \quad (5.309)$$

where v_{pa} is the saturated vapour pressure obtained from (5.235).

- [Shine \(1984\)](#)

$$Q_{sol} = Q_{cs} + Q_{os} \quad (5.310)$$

where the clear and overcast values are given by

$$\begin{aligned} Q_{cs} &= (1 - f_c)(1 - A_{cs})Q_s \sin\gamma_\odot \left(1.2 \sin\gamma_\odot + 0.001v_{pa}(1 + \sin\gamma_\odot) + 0.0455 \right)^{-1} \\ Q_{os} &= f_c(1 - A_{os}) \frac{(53.5 + 1274.5 \sin\gamma_\odot) \sqrt{\sin\gamma_\odot}}{1 + 0.139(1 - 0.935A_{os})d_c} \end{aligned} \quad (5.311)$$

where $d_c = 6.35$ is the dimensionless cloud optical depth. This formula is recommended ([Key et al., 1996](#)) above sea-ice at high latitudes.

Switches

The scheme for the solar (short-wave) radiation is selected with `iopt_sflux_qshort`

- 1 [Rosati & Miyakoda \(1988\)](#)
- 2 [Zillman \(1972\)](#)
- 3 [Shine \(1984\)](#)

5.9 Bottom boundary conditions

5.9.1 General form

In analogy with Section 5.6.1 most of the bottom boundary conditions are flux (Neumann type) conditions and can be written into one of the two following general forms

- A prescribed (upwards) bottom flux F_b^ψ

$$\frac{\lambda_T^\psi}{h_3} \frac{\partial \psi}{\partial s} = F_b^\psi \quad (5.312)$$

- A bottom flux describing the transfer across the sea bed

$$\frac{\lambda_T^\psi}{h_3} \frac{\partial \psi}{\partial s} = C_b^\psi (\psi_b^i - \psi_b^e) \quad (5.313)$$

where C_b^ψ is the transfer rate (with the dimension of a velocity) and ψ_b^e, ψ_b^i are the values of ψ just below and above the sea bed.

For example, the bottom conditions (5.315) and (5.319) for u and v are of the form (5.313) with

$$C_b^u = C_b^v = C_{db}(u_b^2 + v_b^2)^{1/2}, \quad u_b^e = v_b^e = 0 \quad (5.314)$$

An alternative form is a Dirichlet boundary condition where the value of ψ at the bottom or the first interior point is specified. Examples are the conditions (5.330) for turbulence.

Note that when the model equations are given in depth-averaged mode (Section 5.2.2), the bottom boundary condition enters as an additional flux (source or sink) term in the transport equations. It is obvious that in that case only a Neumann flux condition is allowed.

5.9.2 Currents

A slip boundary condition is applied for the horizontal current at the bottom which takes the form

$$\rho_0 \frac{\nu_T}{h_3} \left(\frac{\partial u}{\partial s}, \frac{\partial v}{\partial s} \right) = (\tau_{b1}, \tau_{b2}) \quad (5.315)$$

The following formulations have been implemented

- zero stress condition

$$(\tau_{b1}, \tau_{b2}) = (0, 0) \quad (5.316)$$

- linear friction law, either in 3-D as in 2-D mode

$$(\tau_{b1}, \tau_{b2}) = \rho_0 k_{lin}(u_b, v_b) \quad (5.317)$$

or

$$(\tau_{b1}, \tau_{b2}) = \rho_0 k_{lin}(\bar{u}, \bar{v}) \quad (5.318)$$

- quadratic friction law, either in 3-D or in 2-D mode

$$(\tau_{b1}, \tau_{b2}) = \rho_0 C_{db} (u_b^2 + v_b^2)^{1/2} (u_b, v_b) \quad (5.319)$$

or

$$(\tau_{b1}, \tau_{b2}) = \rho_0 C_{db} (\bar{u}^2 + \bar{v}^2)^{1/2} (\bar{u}, \bar{v}) \quad (5.320)$$

where the bottom currents (u_b, v_b) are evaluated at the grid point nearest to the bottom and (\bar{u}, \bar{v}) are the depth-mean currents. A constant value is taken for the linear friction coefficient k_{lin} .

The quadratic law for the 3-D case is obtained using the boundary layer approximation of a vertically uniform shear stress (see equation (5.328)), yielding a logarithmic profile for the current

$$|\mathbf{u}(z)| = \frac{u_{*b}}{\kappa} \ln\left(\frac{z_*}{z_0}\right) \quad (5.321)$$

where $u_{*b}^2 = \tau_b$, z_* the height above the sea bed and z_0 the bottom roughness length. The quadratic bottom drag coefficient can then be expressed as a function of the roughness length z_0 and the location of the bottom cell. This gives

$$C_{db} = \left(\kappa / \ln(z_r/z_0) \right)^2 \quad (5.322)$$

where z_r is a reference height taken at the grid centre of the bottom cell. The value of z_0 which may vary in the horizontal directions, depends on the geometry and composition of the seabed. Values of z_0 , measured from logarithmic current profiles can be found in [Heathershaw \(1981\)](#); [Soulsby \(1983, 1997\)](#) for various bed type forms.

When COHERENS is used in 2-D mode, the drag coefficient is determined by averaging (5.321) over the water depth. Assuming $z_0 \ll H$, one obtains

$$C_{db} = \left[\kappa / \ln(H/(ez_0)) \right]^2 \quad (5.323)$$

Note that in depth-averaged (2-D) mode, the only allowed formulations for the bottom stress are (5.318), (5.320) and (5.323).

The log-layer approximation is only valid if $z_b \gg z_0$. This may create a problem in case the grid cell is drying and $z_b \rightarrow z_0$, $C_{db} \rightarrow \infty$. To prevent too large drag coefficients, a lower limit has been imposed of the form $z_b/z_0 > \xi_{min}$. The parameter ξ_{min} can be defined by the user. The default value 2 yields a maximum of 0.333 for C_{db} .

In analogy with the surface condition (5.226) the bottom value of the transformed vertical velocity equals zero, i.e.

$$\omega = 0 \quad (5.324)$$

Switches

Evaluation of the bottom stress is controlled by the following switches

`iopt_bstres_form` Type of bottom stress formulation

- 0: Zero bottom stress
- 1: Linear friction law
- 2: Quadratic friction law

`iopt_bstres_nodim` Type of currents used in the bottom stress formulation

- 2: Depth mean currents
- 3: 3-D current at the bottom grid cell

`iopt_bstres_drag` Type of formulation for C_{db}

- 0: Set to zero
- 1: Constant prescribed value
- 2: Prescribed horizontally non-uniform value
- 3: Using (5.322) or (5.323) and a constant roughness length
- 4: Using (5.322) or (5.323) and a spatially dependent roughness length

5.9.3 Temperature and salinity

The bottom boundary conditions for temperature and salinity are obtained by considering a zero flux normal to the seabed:

$$\frac{\lambda_T}{h_3} \frac{\partial T}{\partial s} = 0, \quad \frac{\lambda_T}{h_3} \frac{\partial S}{\partial s} = 0 \quad (5.325)$$

A similar assumption is applied for the absorption term in the temperature equation (5.5) where solar radiance I is set to zero at the sea bottom. It is remarked that the non-allowance of any heat exchange at the bottom interface may not be realistic but is only imposed in the absence of a useful parameterisation which takes account of a bottom exchange (e.g. release of geothermal energy).

5.9.4 Turbulence

The bottom boundary conditions are obtained using the same boundary layer assumptions as for the surface case. Equations (5.241)–(5.244) are replaced by equations (5.326)–(5.329) below. The bottom friction velocity is defined by

$$u_{\star b}^2 = \sqrt{\tau_{b1}^2 + \tau_{b2}^2} = S_{u0} \frac{k^2}{\varepsilon} \frac{\partial U}{\partial z} = \text{constant} \quad (5.326)$$

The mixing length is proportional to the distance d_b from the “wall” boundary as given by equation (5.173):

$$l = l_1 = \kappa d_b = \kappa(h + z + z_{0b}) = \kappa(H\sigma + z_{0b}) \quad (5.327)$$

where z_{0b} a bottom roughness length.

$$\frac{\partial U}{\partial z} = \frac{u_{\star b}}{l} = \frac{u_{\star b}}{\kappa d_b} \quad (5.328)$$

$$P = u_{\star b}^2 \frac{\partial U}{\partial z} = \varepsilon = \epsilon_0 \frac{k^{3/2}}{\kappa d_b} \quad (5.329)$$

The following Dirichlet conditions are derived from the previous equations

$$k = \frac{u_{\star b}^2}{S_{u0}^{1/2}}, \quad \varepsilon = \frac{u_{\star b}^3}{\kappa d_b}, \quad l = \kappa d_b \quad (5.330)$$

In analogy with the surface case the conditions for k and ε can be replaced by conditions for the bottom flux

$$\frac{\nu_k}{h_3} \frac{\partial k}{\partial s} = 0 \quad (5.331)$$

$$\frac{\nu_k}{h_3 \sigma_\varepsilon} \frac{\partial \varepsilon}{\partial s} = \frac{\nu_k}{h_3 \sigma_\varepsilon} \frac{\epsilon_0 k^{3/2}}{\kappa d_b^2} \quad (5.332)$$

The first condition states that there is no energy flux across the bottom.

Switches

Bottom boundary conditions for turbulence are selected by the following switches:

`iopt_turb_tke_bbc` Type of condition for k

1: Neumann condition (5.331)

2: Dirichlet condition (5.330)

`iopt_turb_dis_bbc` Type of condition for ε

1: Neumann condition (5.332)

2: Dirichlet condition (5.330)

5.10 Lateral boundary conditions

5.10.1 Open boundary conditions for the 2-D mode

The model uses a Arakawa C-grid (see Section 4.2). The 2-D mode equations contain the three unknown variables U , V and ζ . The surface elevation is obtained at the C-nodes from the continuity equation which does not explicitly require knowledge of ζ at the open boundaries. This means that open boundary conditions only need to be supplied for the transports U and V at respectively the U- and V-nodes. However, a robust scheme needs to take account of the information entering or leaving the domain which involves all three parameters and should therefore include ζ as well.

The implemented schemes can be divided in four categories⁹:

1. Conditions without externally imposed values for transports and elevations (0,1,2,5,6,7,10,13).
2. Conditions with imposed elevations (3,9,12) or surface slopes (17). An “external” value for the transport is obtained by solving a local solution of the momentum equations.
3. Conditions with imposed transports, currents or discharges (4,14,15,16).
4. Conditions with specified transports and elevations (8,11).

External values (U^e, V^e, ζ^e) can be expressed as the sum of a non-harmonic and an harmonic part

$$\psi^e(\xi_1, \xi_2, t) = \psi_0^e(\xi_1, \xi_2, t) + \sum_{n=1}^N A_n(\xi_1, \xi_2) f_n(t) \cos(V_{qn}(t) + i_n \lambda_{ref}^o - \varphi_n(\xi_1, \xi_2)) \quad (5.333)$$

where ψ_0^e represents the non-harmonic part, f_n are the nodal amplitude factors, $V_{qn}(t)$ the astronomical phases at Greenwich, obtained from (5.210), A_n , φ_n the space-dependent amplitudes and phases at the open boundaries

⁹The numbers in parentheses refer to the numbering of the descriptions given below.

with respect to a user-defined reference longitude $\lambda = \lambda_{ref}^o$. The parameter i_n depends on the type of harmonic (diurnal, semi-diurnal, ...) and is defined in the second column of Table 5.5. The amplitudes A_n and phases φ_n are constant in time but non-uniform in space and needed to determine the harmonic input at the open boundaries. The function ψ_0^e depends on space and time. Values for A_n , φ_n and ψ_0^e need to be supplied by the user. By default, phases are determined with respect to Greenwich ($\lambda_{ref}^o = 0$).

In COHERENS the astronomical phases may alternatively be defined using the simpler form

$$V_{qn} = V_{0n} + \omega_n t \quad (5.334)$$

where V_{0n} is a user-defined constant and $\lambda_{ref}^o = 0$. This option should only be used only for idealised numerical experiments and not for real-time applications.

Variations in atmospheric pressure cause a displacement of the sea level. On the other hand, when an external surface elevation is specified, usually in the form of an harmonic expansion, the harmonic amplitudes are obtained with respect to a reference atmospheric pressure P_{ref} . To take account of this “inverse barometer” an (optionally) correction term is added to ζ^e , i.e.

$$\zeta^e = \text{expression (5.333)} + (P_{ref} - P_a)/(g\rho_0) \quad (5.335)$$

A relaxation condition in time can (optionally) be applied for all exterior 2-D data (transports and elevation) in case the model is set up with the default initial conditions (zero transports and elevations). In that case the exterior data function $\psi^e(\xi_1, \xi_2, t)$ is multiplied by the factor

$$\alpha_r(t) = \min((t - t_0)/T_r, 1) \quad (5.336)$$

where T_r is a user-defined relaxation period and t_0 the initial time. The method avoids the development of discontinuities during the initial propagation of (e.g.) a tidal wave into the domain.

All available schemes for 2-D open boundary conditions are briefly described below. Details are not given but can be found in the appropriate references. Comparison of different schemes are discussed in e.g. [Palma & Matano \(1998\)](#); [Jensen \(1998\)](#); [Røed & Cooper \(1987\)](#). Note that the conditions are applied after solving the 2-D mode equations for U , V , ζ at all interior points.

The following notations are adopted

- \pm or \mp : upper (lower) sign applies at western/southern (eastern/northern) boundaries

- the gravity wave speed c is defined by $c = \sqrt{gH}$
- s equals 1 if ζ^e is defined at an exterior node or 2 if ζ^e is defined at the open boundary (U- or V-) node

0. Clamped.

The transports are uniform in time and determined by the initial conditions.

$$\frac{\partial U}{\partial t} = 0, \quad \frac{\partial V}{\partial t} = 0 \quad (5.337)$$

This is the default condition.

1. Zero slope.

The 2-D momentum equations are solved without surface slope, advection, horizontal diffusion, pressure gradient and astronomical force.

$$\frac{\partial U}{\partial t} = fV + HF_1^t + \frac{1}{\rho_0}(\tau_{s1} - \tau_{b1}), \quad \frac{\partial V}{\partial t} = -fU + HF_2^t + \frac{1}{\rho_0}(\tau_{s2} - \tau_{b2}) \quad (5.338)$$

2. Zero volume flux.

This is a reflective boundary condition whereby the transport is set equal to its nearest interior value.

$$\frac{\partial U}{\partial \xi_1} = 0, \quad \frac{\partial V}{\partial \xi_2} = 0 \quad (5.339)$$

3. Specified elevation.

The 2-D momentum equations are solved without advection, horizontal diffusion, atmospheric and baroclinic pressure gradient.

$$\begin{aligned} \frac{\partial U}{\partial t} &= -\frac{c^2}{h_1} \frac{\partial \zeta}{\partial \xi_1} + fV + HF_1^t + \frac{1}{\rho_0}(\tau_{s1} - \tau_{b1}) \\ \frac{\partial V}{\partial t} &= -\frac{c^2}{h_2} \frac{\partial \zeta}{\partial \xi_2} - fU + HF_2^t + \frac{1}{\rho_0}(\tau_{s2} - \tau_{b2}) \end{aligned} \quad (5.340)$$

The slope term is calculated by the spatial difference of the specified ζ value, either at the open boundary or outside the model grid, and its nearest interior value. The solutions are called “local” since all other horizontal gradients are suppressed. This condition is the easiest to use if ζ is the only available data parameter.

4. Specified transport.

$$U = U^e, \quad V = V^e \quad (5.341)$$

This is the simplest and most appropriate condition to be used at river boundaries.

5. Radiation condition using the shallow water wave speed.

Several types of radiation conditions, which allow the propagation of waves approaching the open boundary, are implemented. Oblique waves are not considered so that a normal incidence on the boundary is assumed. The methods use a Sommerfeld type of equation of the form

$$\frac{\partial \psi}{\partial t} \mp \frac{C}{h_i} \frac{\partial \psi}{\partial \xi_i} = 0 \quad (5.342)$$

where C is an appropriate wave speed. The first condition uses the surface gravity wave speed for shallow water

$$\frac{\partial U}{\partial t} \mp \frac{c}{h_1} \frac{\partial U}{\partial \xi_1} = 0, \quad \frac{\partial V}{\partial t} \mp \frac{c}{h_2} \frac{\partial V}{\partial \xi_2} = 0 \quad (5.343)$$

6. [Orlanski \(1976\)](#) condition.

This is the most popular radiation scheme using

$$C = c_r = h_i \frac{\partial \psi / \partial t}{\partial \psi / \partial \xi_i} \quad (5.344)$$

so that

$$\begin{aligned} \frac{\partial U}{\partial t} \mp \frac{c_r}{h_1} \frac{\partial U}{\partial \xi_1} &= 0, \quad c_r = \pm \frac{\partial U}{\partial t} / \left(\frac{1}{h_1} \frac{\partial U}{\partial \xi_1} \right) \\ \frac{\partial V}{\partial t} \mp \frac{c_r}{h_2} \frac{\partial V}{\partial \xi_2} &= 0, \quad c_r = \pm \frac{\partial V}{\partial t} / \left(\frac{1}{h_2} \frac{\partial V}{\partial \xi_2} \right) \end{aligned} \quad (5.345)$$

The numerical implementation (further discussed in Section [12.3.16.1](#)) involves the values from two previous time steps and at the nearest two interior grid points.

7. [Camerlengo & O'Brien \(1980\)](#).

The scheme is a mixture of the clamped and zero flux condition and can be considered as a simplified case of the Orlanski condition. Details are given in Section [12.3.16.1](#).

8. Flather (1976) with specified elevation and transport.

This is based on (5.343) combined with the continuity equation using only the volume flux normal to the open boundary. The condition then requires that $U \pm c\zeta$ or $V \pm c\zeta$ is continuous across the boundary or

$$U = U^e \mp \frac{1}{2}sc(\zeta - \zeta^e), \quad V = V^e \mp \frac{1}{2}sc(\zeta - \zeta^e) \quad (5.346)$$

9. Flather (1976) with specified elevation.

The formulation is the same as (5.346) with U^e , V^e replaced by the local solutions U^L , V^L obtained by solving (5.340)

$$U = U^L \mp \frac{1}{2}sc(\zeta - \zeta^L), \quad V = V^L \mp \frac{1}{2}sc(\zeta - \zeta^L) \quad (5.347)$$

10. Røed & Smedstad (1984).

The condition is the same as Flather's condition but with all specified exterior values replaced by local solutions

$$U = U^L \mp c(\zeta - \zeta^L), \quad V = V^L \mp c(\zeta - \zeta^L) \quad (5.348)$$

where

$$\frac{\partial \zeta^L}{\partial t} = -\frac{1}{h_1 h_2} \frac{\partial}{\partial \xi_2} (h_1 V), \quad \frac{\partial \zeta^L}{\partial t} = -\frac{1}{h_1 h_2} \frac{\partial}{\partial \xi_1} (h_2 U) \quad (5.349)$$

and U^L , V^L are obtained from (5.340) using ζ^L .

11. Characteristic method with specified elevation and transport.

The method is perhaps the most robust, but also the most complex one. The scheme is based on the theory of characteristics. The principle is to determine which information propagates into or out of the domain. The former is calculated using external data, the latter from the model equations. The method is discussed in Hedstrom (1979); Hirsch (1990) and applied in modified form to a barotropic ocean model by Røed & Cooper (1987).

The characteristic variables are defined by

$$R_{\pm}^u = U \pm c\zeta \quad \text{or} \quad R_{\pm}^v = V \pm c\zeta \quad (5.350)$$

At a western (eastern) boundary R_{-}^u (R_{+}^u) is the outgoing and R_{+}^u (R_{-}^u) the incoming characteristic. Let $R_i^u = U \pm c\zeta$, $R_o^u = U \mp c\zeta$ denote the

incoming and outgoing characteristics at a western or eastern boundary and $R_i^v = V \pm c\zeta$, $R_o^v = V \mp c\zeta$ their counterparts at a southern or northern boundary. The outgoing characteristics are obtained by solving

$$\frac{\partial R_o^u}{\partial t} \mp \frac{c}{h_1} \frac{\partial R_o^u}{\partial \xi_1} = \pm \frac{c}{h_1 h_2} \left(\frac{\partial}{\partial \xi_2} (h_1 V) + \frac{\partial h_2}{\partial \xi_1} U \right) + fV + HF_1^t + \frac{1}{\rho_0} (\tau_{s1} - \tau_{b1}) \quad (5.351)$$

and

$$\frac{\partial R_o^v}{\partial t} \mp \frac{c}{h_2} \frac{\partial R_o^v}{\partial \xi_2} = \pm \frac{c}{h_1 h_2} \left(\frac{\partial}{\partial \xi_1} (h_2 U) + \frac{\partial h_1}{\partial \xi_2} V \right) - fU + HF_2^t + \frac{1}{\rho_0} (\tau_{s2} - \tau_{b2}) \quad (5.352)$$

The equations are obtained by adding the continuity equation (5.32), multiplied by $\mp c$, to the two momentum equations (5.47)–(5.48), where the advective and horizontal diffusion terms, the baroclinic and atmospheric pressure gradient are neglected. These equations are solved using values of the involved parameters evaluated inside the domain.

The incoming characteristic is prescribed by

$$R_i^u = U^e \pm c\zeta^e, \quad R_i^v = V^e \pm c\zeta^e \quad (5.353)$$

The transports are then obtained by

$$U = \frac{1}{2} (R_i^u + R_o^u), \quad V = \frac{1}{2} (R_i^v + R_o^v) \quad (5.354)$$

12. Characteristic method with specified elevation.

The method is as previous with U^e , V^e replaced by the local solution U^L , V^L from (5.340).

13. Characteristic method using a zero normal gradient.

Following Røed & Cooper (1987) the incoming characteristic is, in the absence of available data, obtained from (5.351) or (5.352) with $\partial R_i^u / \partial \xi_1 = 0$ and $\partial R_i^v / \partial \xi_2 = 0$. This gives

$$\frac{\partial R_i^u}{\partial t} = \mp \frac{c}{h_1 h_2} \left(\frac{\partial}{\partial \xi_2} (h_1 V) + \frac{\partial h_2}{\partial \xi_1} U \right) + fV + HF_1^t + \frac{1}{\rho_0} (\tau_{s1} - \tau_{b1}) \quad (5.355)$$

$$\frac{\partial R_i^v}{\partial t} = \mp \frac{c}{h_1 h_2} \left(\frac{\partial}{\partial \xi_1} (h_2 U) + \frac{\partial h_1}{\partial \xi_2} V \right) - fU + HF_2^t + \frac{1}{\rho_0} (\tau_{s2} - \tau_{b2}) \quad (5.356)$$

14. Specified depth-mean current

$$U = \bar{u}^e H, \quad V = \bar{v}^e H \quad (5.357)$$

15. Specified discharge

Instead of imposing transport or depth-mean current one may specify a volume discharge condition (appropriate at e.g. river boundaries)

$$U = Q/h_2, \quad V = Q/h_1 \quad (5.358)$$

where Q is a volume discharge (m^3/s) and h_2 or h_1 the width of the open boundary cell.

16. Distributed discharge

This is an extended version of the previous one whereby the discharge is distributed along one or more open boundary sections. Each section may contain one or more open boundary nodes.

$$\begin{aligned} U &= \frac{H^{1.5} C_z}{\sum_{m=1}^N h_{2;m} H_m^{1.5} C_{zm}} Q \\ V &= \frac{H^{1.5} C_z}{\sum_{m=1}^N h_{1;m} H_m^{1.5} C_{zm}} Q \end{aligned} \quad (5.359)$$

where Q is the total discharge (m^3/s), N the number of open nodes along the section and C_z the Chézy coefficient ($\text{m}^{1/2}/\text{s}$) which is related to the bottom drag coefficient by

$$C_z = \sqrt{\frac{g}{C_{db}}} \quad (5.360)$$

An outline of the method is shown in Figure 5.3. Care should be taken to use this type of boundary condition since several numerical tests showed that this may lead to instabilities when applied to sections with strong discontinuities in the bottom profile.

17. Imposed surface slope

The method can be considered as an extended version of the zero slope boundary condition or a modified version of the condition using specified elevations. Equations (5.338) or (5.340) are replaced by

$$\frac{\partial U}{\partial t} = -gH \left(\frac{1}{h_1} \frac{\partial \zeta}{\partial \xi_1} \right)^e + fV + HF_1^t + \frac{1}{\rho_0} (\tau_{s1} - \tau_{b1})$$

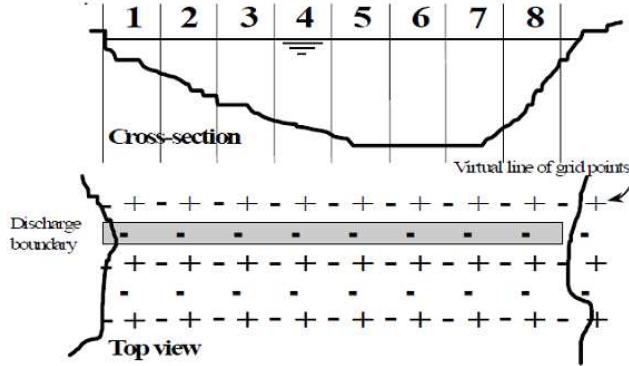


Figure 5.3: View of an open boundary section where the total discharge is distributed.

$$\frac{\partial V}{\partial t} = -gH \left(\frac{1}{h_2} \frac{\partial \zeta}{\partial \xi_2} \right)^e - fU + HF_2^t + \frac{1}{\rho_0} (\tau_{s2} - \tau_{b2}) \quad (5.361)$$

The advantage of this type of (“Neumann”) open boundary condition is that no discretisation is required for the slope term. Assuming that the surface elevation at the open boundary is available from an harmonic expression of the form

$$\zeta^e(\xi_1, \xi_2, t) = \zeta_0^e(\xi_1, \xi_2, t) + \sum_{n=1}^N A_n(\xi_1, \xi_2) f_n(t) \cos(V_n(t) + u_n(t) - \varphi_n(\xi_1, \xi_2)) \quad (5.362)$$

the surface slope, obtained by taking the derivative of (5.362) is given by

$$\left(\frac{1}{h_i} \frac{\partial \zeta}{\partial \xi_i} \right)^e = \left(\frac{1}{h_i} \frac{\partial \zeta}{\partial \xi_i} \right)_0 + \sum_{n=1}^N A'_n(\xi_1, \xi_2) f_n(t) \cos(V_n(t) + u_n(t) - \varphi'_n(\xi_1, \xi_2)) \quad (5.363)$$

where

$$\begin{aligned} A'_n &= \frac{1}{h_i} \sqrt{\left(\frac{\partial A_n}{\partial \xi_i} \right)^2 + \left(A_n \frac{\partial \varphi_n}{\partial \xi_i} \right)^2} \\ \tan \varphi'_n &= \frac{\partial \varphi_n}{\partial \xi_i} \left(A_n / \frac{\partial A_n}{\partial \xi_i} \right) \end{aligned} \quad (5.364)$$

A motivation for using Neumann type of open boundary conditions is given by [Roelvink & Walstra \(2005\)](#). The problem of specifying boundary conditions at lateral boundaries is that due to a combination of

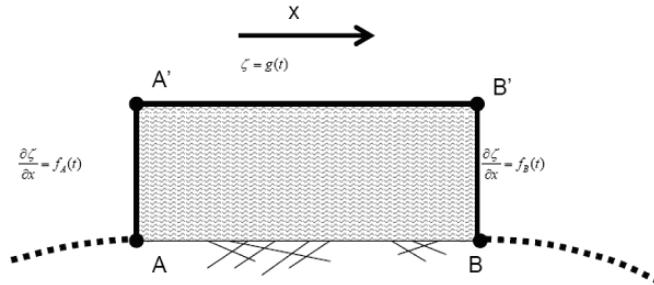


Figure 5.4: Scheme of a model with three open boundaries: water level and Neumann boundary conditions (Delft3D, 2009).

processes acting on the model domain, a certain water level or velocity distribution will develop in the cross-shore direction. For the boundary conditions to match this distribution it has to be known beforehand; if not, boundary disturbances will develop.

Therefore, these authors suggest to let the model determine the correct solution at the boundaries by imposing the alongshore water level gradient instead of a fixed water level or velocity. In many cases the gradient can be assumed to be zero; only in tidal cases or in cases where a storm surge travels along a coast the alongshore gradient varies in time, but in a model with a limited cross-shore extent, the alongshore gradient of the water level does not vary much in cross-shore direction.

The alongshore water level gradient can be assumed to be zero in case of a steady wind field/wave field, or vary periodically for a tidal wave traveling along the coast. In the latter case, the alongshore gradient varies in time at the boundary. A constant periodical water level gradient for a boundary can be applied to models with a limited cross-shore extent, under the assumption that the alongshore gradient of the water level does not vary much in the cross-shore direction. Neumann boundaries can only be applied on cross-shore boundaries in combination with a water level boundary at the seaward boundary, which is needed to make the solution of the mathematical boundary value problem well-posed (Deltares, 2011). An example of the prescription of Neumann boundary conditions is depicted in Figure 5.4.

Switches

`iopt_astro_pars` Type of formulation for evaluating the astronomical phases $V_{qn}(t)$.

- 1: Using the linear format (5.334) with a user-defined initial phase. Should be used for special cases only, but is not recommended in general.
- 2: Using (5.210) with t_{ref} either defined initially with no further updates or updated at a regular time interval selected by the user (which must be a multiple of days). Default.

5.10.2 Open boundary conditions for the 3-D mode

5.10.2.1 baroclinic currents

Since U and V are obtained from the 2-D open boundary conditions, only their baroclinic parts $\delta u = U - u/H$ and $\delta v = V - v/H$ need to be determined. The following conditions can be selected

0. Zero gradient.

$$\frac{1}{h_1} \frac{\partial}{\partial \xi_1} (h_2 h_3 \delta u) = 0, \quad \frac{1}{h_2} \frac{\partial}{\partial \xi_2} (h_1 h_3 \delta v) = 0 \quad (5.365)$$

This is the default condition.

1. Specified external profile.

$$\delta u = \delta u^e, \quad \delta v = \delta v^e \quad (5.366)$$

2. Second order gradient condition. In case of ragged open boundaries the (first order) zero gradient condition may yield spurious discontinuities of the vertical current at the first interior node. The effect is reduced when using the second order condition

$$\frac{1}{h_1} \frac{\partial}{\partial \xi_1} \left[\frac{1}{h_1 h_2} \frac{\partial}{\partial \xi_1} (h_2 h_3 \delta u) \right] = 0, \quad \frac{1}{h_2} \frac{\partial}{\partial \xi_2} \left[\frac{1}{h_1 h_2} \frac{\partial}{\partial \xi_2} (h_1 h_3 \delta v) \right] = 0 \quad (5.367)$$

at respectively U- and V-node open boundaries.

3. Local solution. The equation is derived from the 3-D and 2-D momentum equations without advection and horizontal diffusion:

$$\frac{1}{h_3} \frac{\partial}{\partial t} (h_3 \delta u) - 2\Omega \sin \phi \delta v = F_1^b - \frac{\overline{F_1^b}}{H} + \frac{1}{h_3} \frac{\partial}{\partial s} \left(\frac{\nu_T}{h_3} \frac{\partial \delta u}{\partial s} \right) + \frac{1}{\rho_0 H} (\tau_{b1} - \tau_{s1}) \quad (5.368)$$

at U-nodes and

$$\frac{1}{h_3} \frac{\partial}{\partial t} (h_3 \delta v) + 2\Omega \sin \phi \delta u = F_2^b - \frac{\overline{F_2^b}}{H} + \frac{1}{h_3} \frac{\partial}{\partial s} \left(\frac{\nu_T}{h_3} \frac{\partial \delta v}{\partial s} \right) + \frac{1}{\rho_0 H} (\tau_{b2} - \tau_{s2}) \quad (5.369)$$

at V-node open boundaries. The diffusive fluxes at the surface and bottom are determined by respectively (5.224) and (5.315) with u, v , replaced by $\delta u, \delta v$.

4. Radiation condition using the baroclinic wave speed.

$$\frac{\partial \delta u}{\partial t} \mp c_i \frac{1}{h_1} \frac{\partial \delta u}{\partial \xi_1} = 0, \quad \frac{\partial \delta v}{\partial t} \mp c_i \frac{1}{h_2} \frac{\partial \delta v}{\partial \xi_2} = 0 \quad (5.370)$$

The baroclinic wave speed c_i is generally unknown and has to be specified by the user.

5. Orlanski condition.

$$\begin{aligned} \frac{\partial \delta u}{\partial t} \mp c_r \frac{1}{h_1} \frac{\partial \delta u}{\partial \xi_1} &= 0, \quad c_r = \pm \frac{\partial \delta u}{\partial t} / \left(h_1 \frac{\partial \delta u}{\partial \xi_1} \right) \\ \frac{\partial \delta v}{\partial t} \mp c_r \frac{1}{h_2} \frac{\partial \delta v}{\partial \xi_2} &= 0, \quad c_r = \pm \frac{\partial \delta v}{\partial t} / \left(h_2 \frac{\partial \delta v}{\partial \xi_2} \right) \end{aligned} \quad (5.371)$$

6. Discharge condition

Instead of providing a full vertical profile for the baroclinic current, a discharge condition can be imposed at one or several vertical locations.

$$u = \frac{q}{h_2 h_3}, \quad v = \frac{q}{h_1 h_3} \quad (5.372)$$

so that

$$\delta u_k = \frac{1}{h_2} \left(\frac{q_k}{h_3} - \frac{Q}{H} \right), \quad \delta v_k = \frac{1}{h_1} \left(\frac{q_k}{h_3} - \frac{Q}{H} \right) \quad (5.373)$$

where $Q = \sum_{k=1}^{N_z} q_k^e = h_2 U$ or $= h_1 V$ is the discharge over the whole column.

Important to note is that a discharge condition must be imposed for the 2-D mode as well at the same boundary location (see equation (5.358) with the same value of Q).

5.10.3 Boundary conditions for cross-stream advective fluxes

Several boundary conditions are implemented for the calculation of cross-stream advective fluxes at open boundaries. They are further discussed in Sections 12.3.16.2 and 12.3.17.2.

5.10.3.1 3-D scalars

Open boundary conditions are needed for calculation of the horizontal advective fluxes inside the domain. Since advective fluxes at open boundaries are determined by an upwind scheme, this means that an “outside” profile must be defined in case of an inflow condition, whereas the advective flux is determined by the (known) upstream value of the concentration during outflow. Different formulations for determining the outside profile are implemented in COHERENS.

0. Zero gradient.

$$\frac{1}{h_1} \frac{\partial}{\partial \xi_1} (h_2 h_3 \psi) = 0, \quad \frac{1}{h_2} \frac{\partial}{\partial \xi_2} (h_1 h_3 \psi) = 0 \quad (5.374)$$

This is the default condition.

1. Specified external profile.

$$\psi = \psi^e \quad (5.375)$$

2. Radiation condition using the baroclinic wave speed.

$$\frac{\partial \psi}{\partial t} \mp c_i \frac{1}{h_1} \frac{\partial \psi}{\partial \xi_1} = 0, \quad \frac{\partial \psi}{\partial t} \mp c_i \frac{1}{h_2} \frac{\partial \psi}{\partial \xi_2} = 0 \quad (5.376)$$

The baroclinic wave speed c_i is generally unknown and has to be specified by the user.

3. Orlanski condition.

$$\begin{aligned} \frac{\partial \psi}{\partial t} \mp c_r \frac{1}{h_1} \frac{\partial \psi}{\partial \xi_1} &= 0, \quad c_r = \pm \frac{\partial \psi}{\partial t} / \left(h_1 \frac{\partial \psi}{\partial \xi_1} \right) \\ \frac{\partial \psi}{\partial t} \mp c_r \frac{1}{h_2} \frac{\partial \psi}{\partial \xi_2} &= 0, \quad c_r = \pm \frac{\partial \psi}{\partial t} / \left(h_2 \frac{\partial \psi}{\partial \xi_2} \right) \end{aligned} \quad (5.377)$$

4. Thatcher-Harleman condition.

The transition of a scalar concentration at the boundary from its outflow value to the inflow value may take some time (Delft3D, 2009; Guan & Wolanski, 2005; Liu *et al.*, 2005). This depends on the refreshment of the water in the boundary region. The transition time is called the ‘return time’. In case of a tidal flow the functional relationship describing the variation in concentration from the slack-water value to the present value is arbitrary. Usually, in similar models, a half-cosine variation is used. After the return time, the boundary value remains constant until outward flow begins (Thatcher & Harleman, 1972). This boundary condition allows for the possibility that some of the water that leaves the estuary on the ebb tide may re-enter the estuary with the following flood tide. The mathematical formulation of this memory effect is given as follows:

$$\psi(t) = \psi^{out} + \frac{1}{2}(\psi^{bnd} - \psi^{out}) \max \left[\cos \left(\pi \frac{T_{ret} - t_{out}}{T_{ret}} \right) + 1, 0 \right]$$

for $0 \leq t_{out} \leq T_{ret}$ (5.378)

where ψ^{out} is the computed open boundary concentration at the last time of outward flow, ψ^{bnd} is the background concentration that should be prescribed by the user as a reference concentration, t_{out} is the elapsed time since the last outflow and T_{ret} is the constituent return period. When the flow turns inward ($t_{out} = 0$), the concentration is set equal to ψ^{out} . During the interval $0 \leq t_{out} \leq T_{ret}$ the concentration will return to the background concentration ψ^{bnd} . For $t_{out} \geq T_{ret}$, ψ remains equal to ψ^{bnd} until the next output time.

For a stratified flow the return time of the upper layer will be longer than for the bottom layer. Hence, the return period will vary along the vertical direction. Similar models, offer the opportunity to prescribe return times for the surface layer and the bed layer. Hence, the return time of the intermediate layers is determined using linear interpolation. In COHERENS the return time can be specified by the user at each vertical level.

5.10.3.2 turbulence variables

Advection of turbulence is considered of minor importance and has been disabled by default. The only available open boundary condition for k , ε or kl therefore consists of a zero gradient condition.

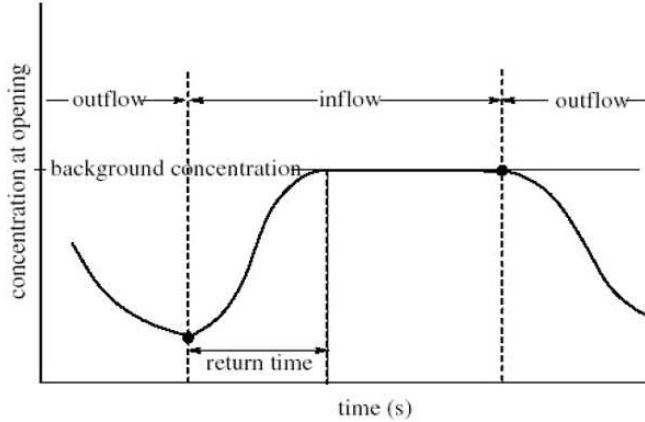


Figure 5.5: Time evolution of a concentration at the open boundary using (5.378).

5.10.4 Relaxation conditions

A known problem with open boundary conditions is that the imposed value of a quantity at the open boundary is not always compatible with its value calculated by the model inside. For example, the thermocline depth obtained from an imposed vertical profile of temperature may be different from the one calculated by the model just inside the domain, creating unrealistic discontinuities. A known solution is the creation of a sponge layer near the open boundaries where the calculated interior solution is allowed to relax towards its value imposed at the open boundary.

The method implemented in the program is the flow relaxation scheme proposed by [Martinsen & Engedahl \(1987\)](#). A relaxation zone is defined near the open boundary where the value of a model quantity ψ is written as

$$\psi_i = \alpha\psi_e + (1 - \alpha)\tilde{\psi}_i \quad (5.379)$$

where ψ_e is the open boundary value, $\tilde{\psi}_i$ the calculated interior value, prior to relaxation, and α a weighting factor which varies between 1 at the open boundary and 0 at the inner edge of the relaxation zone. [Martinsen & Engedahl \(1987\)](#) showed for a simplified case that the procedure is equivalent to add a relaxation term $\alpha(\psi - \psi_e)/\Delta t/(1 - \alpha)$ to the right hand side of the transport equation where Δt is the model time step. In this way, ψ relaxes to its open boundary value if $\alpha \rightarrow 1$ and to the internal solution if $\alpha \rightarrow 0$.

The method can be applied in the program for temperature, salinity, baroclinic currents and other scalars but is not implemented for the 2-D mode. The following interpolation schemes can be selected

1. linear

$$\alpha = 1 - \frac{d}{D} \quad (5.380)$$

2. quadratic

$$\alpha = \left(1 - \frac{d}{D}\right)^2 \quad (5.381)$$

3. hyperbolic

$$\alpha = 1 - \tanh\left(\frac{d}{2\Delta h}\right) \quad (5.382)$$

where d is the distance to the boundary, D the width of the relaxation zone and Δh the grid spacing. Note that the interpolation assumes a uniform grid spacing within the relaxation zone.

5.10.5 Coastal boundaries

At coastal boundaries currents and fluxes of scalars are set to zero, or

$$U = 0, \quad \delta u = 0, \quad u\psi = 0 \quad (5.383)$$

at western and eastern boundaries, and

$$V = 0, \quad \delta v = 0, \quad v\psi = 0 \quad (5.384)$$

at southern and northern boundaries.

5.11 Initial conditions

The default initial conditions are

- 2-D hydrodynamics

$$U = 0, \quad V = 0, \quad \zeta = 0 \quad (5.385)$$

- 3-D hydrodynamics

$$u = v = 0 \quad (5.386)$$

- scalars

$$T = T_{ref}, \quad S = S_{ref} \quad (5.387)$$

where T_{ref} , S_{ref} are uniform values selected by the user. These are also the values taken over time if `iopt_temp=0` or `iopt_sal=0`.

- turbulence

$$k = 10^{-6} \text{ J/kg} \quad (5.388)$$

for turbulent energy while l is obtained from one of the mixing length prescriptions given in Section 5.3.3.5 and ε from (5.164).

Although the initial conditions for turbulence cannot be considered as realistic, they are of lesser importance since turbulence is assumed to be in quasi-equilibrium and adjusts itself rapidly to changes in the forcing conditions, given by the vertical current shear and stratification. It is clear that realistic initial conditions cannot be given in general. In practice, they need to be obtained by the user, or from a previous COHERENS run.

5.12 Harmonic analysis

5.12.1 Residuals, amplitudes and phases

The program offers the possibility to apply an harmonic analysis on a given number of user-defined quantities. An harmonic expansion approximates a function $F(\xi_1, \xi_2, \sigma, t)$ by a series of the form

$$F(\xi_1, \xi_2, \sigma, t) = A_0 + \sum_{n=1}^{N_h} A_n(\xi_1, \xi_2, \sigma) \cos(V_{qn}(t) + i_n \lambda_{ref}^a - \varphi_n(\xi_1, \xi_2, \sigma)) \quad (5.389)$$

where N_h is the number of frequencies used in the expansion, V_{qn} are the astronomical phases, A_0 is the residual part, A_n are the amplitudes and φ_n the phases with respect to a user-defined reference longitude $\lambda = \lambda_{ref}^o$. In analogy with the formulation used at open boundaries the astronomical phases are defined either using the quasi-linear formulation (5.210) or with the simplified formulation (5.334) where V_{qn} is supplied by the user and $\lambda_{ref}^a = 0$.

Equation (5.389) can be rewritten as

$$F(t) = a_0 + \sum_{n=1}^{N_h} f_n(a_n \cos V_{qn} + b_n \sin V_{qn}) \quad (5.390)$$

The residual part, amplitudes and phases are then given by

$$A_0 = a_0, \quad A_n = (a_n^2 + b_n^2)^{1/2}, \quad \varphi_n = \text{mod}(\arctan(b_n/a_n), 2\pi) \quad (5.391)$$

The procedure for obtaining the coefficients a_0 , a_n , b_n is as follows. Firstly, the following parameters are defined by the user

1. A start time t_1 for the first analysis.
2. An end time t_2 for the last analysis.
3. The period of analysis T .

The number of periods for analysis is then defined by $L = (t_2 - t_1)/T$. Analysis is then performed for the consecutive intervals $[t_1 + (l-1)T, t_1 + lT]$ with $1 \leq l \leq L$.

A least squares method is then applied by minimising the following integral for each period

$$R_l = \int_{(l-1)T}^{lT} \left[F(t) - a_0 - \sum_n (a_n \cos V_{qn} + b_n \sin V_{qn}) \right]^2 dt \quad (5.392)$$

or in time-discretised form

$$R_l = \sum_{k=1}^M \left[F_k - a_0 - \sum_n a_n \cos V_{kn} - \sum_n b_n \sin V_{kn} \right]^2 \Delta t \quad (5.393)$$

where $F_k = F(t_k)$, $V_{kn} = V_{qn}(t_k)$ and t_k the time at the k^{th} time interval with $k = 1, \dots, M$ where M is the number of time intervals. Minimising (5.393) with respect to a_0 , a_n , b_n with $n = 1, \dots, N_h$ yields the following linear system of equations with $1 \leq m \leq N_f$:

$$Ma_0 + \sum_n a_n \sum_k \cos V_{kn} + \sum_n b_n \sum_k \sin V_{kn} = \sum_K F_k \quad (5.394a)$$

$$\begin{aligned} \sum_k a_0 \cos V_{km} + \sum_n a_n \sum_k \cos V_{kn} \cos V_{km} + \sum_n b_n \sum_k \sin V_{kn} \cos V_{km} \\ = \sum_k \cos V_{km} F_k \end{aligned} \quad (5.394b)$$

$$\begin{aligned} \sum_k a_0 \sin V_{km} + \sum_n a_n \sum_k \cos V_{kn} \sin V_{km} + \sum_n b_n \sum_k \sin V_{kn} \sin V_{km} \\ = \sum_k \sin V_{km} F_k \end{aligned} \quad (5.394c)$$

The coefficients a_0 , a_m , b_m are obtained by solving the linear system using LU-decomposition.

Important to note is that the number and values of the frequencies ω_n used in the harmonic analysis do not need to be the same as the ones appearing in the tidal forcing, as given by the harmonic expansions (5.206) or (5.333). For example, if the model is forced with the M_2 -tide only, higher order harmonics (M_4, M_6, \dots) are generated by the non-linearities of the model equations. These higher order terms can be investigated by applying an harmonic analysis. The method can be also used to analyse the evolution of a M_2 -tide during a spring-neaps cycle, e.g. by forcing the model with a M_2 - and S_2 -tide and performing the analysis for the M_2 -frequency only for consecutive periods comparable with the M_2 period (say 1 day)(e.g. [Luyten, 1997](#)). [Luyten et al. \(2003\)](#) applied the same procedure for the analysis of internal waves in the North Sea.

The period T needs to be selected with some care. A general rule is that it must be of the same order or larger than any of the analysed periods $2\pi/\omega_n$ and should increase with the number of frequencies. A more stringent restriction is imposed if two frequencies ω_i and ω_j are nearly equal to avoid aliasing effects, in which case T should larger than $2\pi/|\omega_i - \omega_j|$.

5.12.2 Tidal ellipses

Besides amplitudes and phases the tidal characteristics of tidal currents can be further analysed using tidal ellipse parameters. During the course of a tidal cycle the current vector describes a curve, known as the “tidal ellipse”. Characteristic parameters of tidal ellipses are the semi-major axis, semi-minor axis, ellipticity, orientation and elliptic angle. They can optionally be derived by the program in the usual way (e.g. [Godin, 1972](#)) once the harmonic parameters of the current are available using the analysis of the preceding section. The n^{th} harmonic component of the horizontal current during the tidal cycle can be written as

$$u_n = u_{an} \cos(\omega_n t - \varphi_{nu}), \quad v_n = v_{an} \cos(\omega_n t - \varphi_{nv}) \quad (5.395)$$

where the astronomical phase is taken to be zero at the initial time. Introducing the complex notation

$$U = u_a e^{-i\varphi_u}, \quad V = v_a e^{-i\varphi_v} \quad (5.396)$$

where the subscript n has been omitted for simplicity, the complex current can then be decomposed into a cyclonically and an anticyclonically rotating component

$$u + iv = \frac{1}{2}(Ue^{i\omega t} + \tilde{U}e^{-i\omega t}) + \frac{i}{2}(Ve^{i\omega t} + \tilde{V}e^{-i\omega t})$$

$$= S_+ e^{i\omega t} + \tilde{S}_- e^{-i\omega t} \quad (5.397)$$

where a \sim denotes the complex conjugate and

$$S_{\pm} = \frac{1}{2}(U \pm iV) = |S_{\pm}|e^{i\alpha_{\pm}} \quad (5.398)$$

The semi-major axis A , semi-minor axis B and ellipticity e of the ellipse, described by the current, are then given by

$$A = |S_+| + |S_-|, B = ||S_+| - |S_-||, e = (|S_+| - |S_-|)/(|S_+| + |S_-|) \quad (5.399)$$

The inclination Θ of the ellipse with respect to the X-axis and the elliptic angle Φ , i.e. the angle between the initial current at $t = 0$ and its position when the current achieves its first maximum, are obtained using

$$\Theta = (\alpha_+ - \alpha_-)/2, \Phi = -(\alpha_+ + \alpha_-)/2 \quad (5.400)$$

with the restriction that

$$0 \leq \Theta, \Phi \leq \pi \quad (5.401)$$

The orientation of the ellipse is determined by the sign of the ellipticity. A positive value of e means that the current vector rotates cyclonically (anti-clockwise/clockwise in the northern/southern hemisphere) and $|S_+| > |S_-|$ whereas a negative value indicates anticyclonic rotation (clockwise/anticlockwise in the northern/southern hemisphere) and $|S_+| < |S_-|$. If $e = 0$, the flow is rectilinear and $|S_+| = |S_-|$. A useful discussion of cyclonic and anticyclonic components and their impact on the depth of the tidal bottom layer can be found in e.g. [Prandle \(1982\)](#); [Soulsby \(1983\)](#); [Luyten \(1996\)](#).

Switches

The following switches are available for harmonic analysis

iopt_out_anal Switch to disable (0) or enable (1) harmonic analysis. Default is 0.

iopt_astro_pars Type of formulation for evaluating the Greenwich phases $V_{qn}(t)$.

- 1: Using the linear format (5.334) with a user-defined initial phase. Should be used for special cases only, but is not recommended in general.
- 2: Using (5.210) with t_{ref} either defined initially with no further updates or updated at a regular time interval selected by the user (which must be a multiple of days). Default.

Chapter 6

Current-wave interaction model

6.1 Introduction

The wave-current interaction theory described in this chapter consists of two parts:

- A tiny wave boundary layer of a few centimeters develops near the sea bed if waves are present. Within this layer waves and currents interact non-linearly through bottom friction. The result is an enhancement of the bottom drag coefficient and shear stress. The effect has an influence on the profile of the current near the sea bed but also on resuspension of sediments (see chapter 7). Since the wave layer is not resolved by the model, several wave-current interaction schemes, obtained from literature have been implemented. They are described in Section 6.2.
- Non-linear interaction within the water column produces extra source/sink terms in the momentum and scalar transport equations such as radiation stress, additional transport by Stokes drift, wave dissipation near the surface and sea bed. This is further discussed in Section ??.

In order to use these wave-current models a number of wave field parameters are needed. They are obtained from an external wave model and either supplied off-line from a forcing file or on-line by activating the COHERENS-SWAN coupler module. The basic wave parameters, obtained in the wave model by integration over the wave spectrum are the significant wave height H_s , wave period T_w and wave direction ϕ_w . The wave amplitude A_w and frequency are defined by

$$A_w = H_s/2, \omega_w = 2\pi/T_w \quad (6.1)$$

The near-bottom wave orbital velocity determined from linear wave theory is given by

$$U_w = \frac{A_w \omega_w}{\sinh k_w H} \quad (6.2)$$

with the wavenumber k_w obtained from the dispersion relation

$$\omega_w^2 = gk_w \tanh k_w H \quad (6.3)$$

The latter equation is solved for k_w using the approximate formula of [Hunt \(1979\)](#)

$$k_w = \omega_w \sqrt{\frac{f(\alpha)}{gH}} \quad (6.4)$$

with

$$f(\alpha) = \alpha + (1.0 + 0.666\alpha + 0.445\alpha^2 - 0.105\alpha^3 + 0.272\alpha^4)^{-1} \quad (6.5)$$

and

$$\alpha = \frac{\omega_w^2 H}{g} \quad (6.6)$$

It is further assumed that the near bed wave period is equal to the mean wave period, such that the near bed wave excursion amplitude can be written as

$$A_b = \frac{U_w}{\omega_w} p \quad (6.7)$$

6.2 Wave-current interaction at the sea bed

During the last decades several schemes have been developed describing the interaction of waves and currents near the sea bed and their influence on the bottom shear stress. Four of them are implemented in COHERENS:

- the [Soulsby & Clarke \(2005\)](#) model (SC)
- the [Malarkey & Davies \(2012\)](#) model (MD)
- the [Grant & Madsen \(1979, 1986\)](#) model (GM)
- the [Christofferson & Jonsson \(1985\)](#) model (CJ)

Although currents and waves interact non-linearly in the wave boundary layer (WBL), the basic assumption is made that the current and bottom

shear stress in the logarithmic bottom layer are decomposed into a mean and time-dependend part oscillating with the wave frequency

$$\mathbf{u} = \mathbf{u}_c + \mathbf{u}_w, \quad \boldsymbol{\tau} = \boldsymbol{\tau}_m + \boldsymbol{\tau}_w \quad (6.8)$$

The decomposition is valid for time scales comparable to the wave period but small compared to the forcing time scales (e.g. tidal or inertial period).

In the following a distinction will be made between three types of bottom stress:

- the mean stress τ_m representing the stress averaged over the wave cycle
- the maximum value of the shear stress τ_{cw} during the wave cycle
- the peak stress τ_p defined as the amplitude of wave-dependent part τ_w inside the WBL

The corresponding friction velocities are defined by

$$u_{*m} = \left(\frac{\tau_m}{\rho} \right)^{1/2}, \quad u_{*cw} = \left(\frac{\tau_{cw}}{\rho} \right)^{1/2}, \quad u_{*p} = \left(\frac{\tau_p}{\rho} \right)^{1/2} \quad (6.9)$$

The mean stress τ_m is used in the hydrodynamics as bottom boundary condition for the current as given by equation (5.315). The maximum stress τ_{cw} is used within the sediment model to determine the amount of resuspension and for the bed and total load formulations. In the absence of waves there is no distinction between the types of bottom stress since then $\tau_m = \tau_{cw} = \tau_b$ while $\tau_p = 0$, with τ_b given by equations (5.319) and (5.322).

Detailed descriptions of the schemes can be found in the relevant publications. They are presented here in somewhat detail in a unified notation. Some adaptations have been made with respect to the original schemes.

In the following the current is taken along the X-axis and the vertical coordinate z is defined as the distance from the sea bed. The bottom values of u and z will be taken at the lowest grid cell and denoted by u_b and z_b respectively¹.

Above the WBL or $z > z_0 + \delta_w$, the mean shear stress τ_m is assumed to be constant whereas the wave stress τ_w is set to zero. Assuming an eddy viscosity of the form

$$\nu_T = \kappa u_{*m} z \quad (6.10)$$

¹The use of u_b , z_b instead of H and $|\bar{\mathbf{u}}|$ used in the SC, MD and CJ schemes are more appropriate since it avoids the inclusion of unnecessary surface layer effects such as wind stress.

the mean current u_c is determined by solving

$$\tau_m = \rho u_{*m}^2 = \rho \kappa z u_{*m} \frac{\partial u_c}{\partial z} \quad (6.11)$$

giving

$$u_c = \frac{u_{*m}}{\kappa} \ln\left(\frac{z}{\delta_w + z_0}\right) + u_c|_{z=\delta_w+z_0} \quad (6.12)$$

The last term on the right will be determined below by fitting the solution with the one obtained inside the WBL. In the absence of a wave stress, there is no interaction with wave and currents above the WBL so that the wave component reduces to the linear wave form

$$\mathbf{u}_w = \mathbf{u}_w^o = U_w e^{i\omega_w t} \mathbf{1}_w \quad (6.13)$$

where

$$\mathbf{1}_w = (\cos \phi_w, \sin \phi_w) \quad (6.14)$$

is the unit vector in the wave direction. The solution inside the WBL depends on further assumptions and are derived below.

The maximum stress within the WBL is defined by

$$\tau_{cw} = \max |\boldsymbol{\tau}_m + \boldsymbol{\tau}_w| \quad (6.15)$$

where the maximum is taken over a wave cycle. This gives

$$u_{*cw}^2 = \left(u_{*m}^4 + 2 \cos \phi_w u_{*m}^2 u_{*p}^2 + u_{*p}^4 \right)^{1/2} \quad (6.16)$$

6.2.1 Soulsby-Clarke model

Inside the WBL ($z_0 \leq z \leq \delta_w + z$) the mean shear stress τ_m takes the same value as outside the WBL assuming now an eddy viscosity of the form

$$\nu_T = \kappa u_{*e} z \quad (6.17)$$

where

$$u_{*e}^2 = (u_{*c}^4 + u_{*w}^4)^{1/2} \quad (6.18)$$

The current-only, respectively wave-only friction velocities u_{*c} , u_{*w} are given by

$$u_{*c}^2 = \left(\frac{\kappa}{\ln(z_b/z_0)} \right)^2 u_b^2 \quad (6.19a)$$

$$u_{*w}^2 = 0.695 U_w^2 \left(\frac{U_w}{\omega_w z_0} \right)^{-0.52} \quad (6.19b)$$

The equation for the current now becomes

$$\kappa u_{*e} z \frac{\partial u_c}{\partial z} = u_{*m}^2 \quad (6.20)$$

The solution which becomes zero at $z = z_0$ is given by

$$u_c = \frac{u_{*m}^2}{\kappa u_{*e}} \ln\left(\frac{z}{z_0}\right) \quad (6.21)$$

The outside solution (6.12) which matches the inner one then becomes

$$u_c = \frac{u_{*m}}{\kappa} \ln\left(\frac{z}{\delta_w + z_0}\right) + \frac{u_{*m}^2}{\kappa u_{*e}} \ln\left(\frac{\delta_w + z_0}{z_0}\right) \quad (6.22)$$

The mean friction velocity evaluated at $z = z_b$ is determined from (6.22) and (6.21):

$$\begin{aligned} z_0 < z_b \leq \delta_w + z_0 : \quad u_{*m} &= \sqrt{\frac{\kappa u_{*e} u_b}{\ln(z_b/z_0)}} \\ z_b > \delta_w + z_0 : \quad u_{*m} &= \frac{1}{2a} (\sqrt{b^2 + 4a\kappa u_b} - b) \end{aligned} \quad (6.23)$$

with

$$a = \frac{1}{u_{*e}} \ln\left(\frac{\delta_w + z_0}{z_0}\right), \quad b = \ln\left(\frac{z_b}{\delta_w + z_0}\right) \quad (6.24)$$

Instead of obtaining the peak friction velocity by solving a separate equation for the wave component u_w inside the WBL, u_p is defined in the SC model by

$$u_{*p}^2 = u_{*e} u_{*m} \quad (6.25)$$

The WBL thickness δ_w is defined in the SC model by

$$\delta_w = a_r \kappa \frac{u_{*w}}{\omega_w} \quad (6.26)$$

where a_r is given the value of 0.65.

The outer solution for the mean current, given by (6.22), can be rewritten in the usual logarithmic form with the roughness length z_0 replaced by the apparent roughness z_{0a} :

$$u_c = \frac{u_{*m}}{\kappa} \ln\left(\frac{z}{z_{0a}}\right) \quad (6.27)$$

where

$$z_{0a} = (\delta_w + z_0) \left(\frac{z_0}{\delta_w + z_0} \right)^R, \quad R = \frac{u_{*m}}{u_{*e}} \quad (6.28)$$

6.2.2 Malarkey-Davies model

The model is similar to the Soulsby-Clarke model except that equations (6.18) and (6.16) are replaced by

$$u_{*e}^2 = \left(u_{*c}^4 + 2 \cos \phi_w u_{*c}^2 u_{*w}^2 + u_{*w}^4 \right)^{1/2} \quad (6.29)$$

$$u_{*cw}^2 = \left(\varepsilon u_{*m}^4 + 2\varepsilon^{1/2} \cos \phi_w u_{*m}^2 u_{*p}^2 + u_{*p}^4 \right)^{1/2} \quad (6.30)$$

where

$$\varepsilon = 1 + \frac{u_{*w}^2}{(u_{*c}^2 + u_{*w}^2)^2} \left(u_{*c}^2 \cos \phi_w + \frac{1}{4} u_{*w}^2 \right) \quad (6.31)$$

For more details see [Malarkey & Davies \(2012\)](#).

6.2.3 Grant-Madsen model

In the GM model, the solution for the mean current inside the WBL is the same as for the SC case except that u_{*e} is now replaced by u_{*cw} . This means that equations (6.17) and (6.20)–(6.24) remain valid after substituting u_{*cw} for u_{*e} . The inside and outside solutions for the mean current then become

$$u_c = \frac{u_{*m}^2}{\kappa u_{*cw}} \ln\left(\frac{z}{z_0}\right) \quad (6.32a)$$

$$u_c = \frac{u_{*m}}{\kappa} \ln\left(\frac{z}{\delta_w + z_0}\right) + \frac{u_{*m}^2}{\kappa u_{*cw}} \ln\left(\frac{\delta_w + z_0}{z_0}\right) \quad (6.32b)$$

whereas the mean friction velocity is given by

$$z_0 < z_b \leq \delta_w + z_0 : \quad u_{*m} = \sqrt{\frac{\kappa u_{*cw} u_b}{\ln(z_b/z_0)}} \quad (6.33)$$

$$z_b > \delta_w + z_0 : \quad u_{*m} = \frac{1}{2a} \left(\sqrt{b^2 + 4a\kappa u_b} - b \right)$$

with

$$a = \frac{1}{u_{*cw}} \ln\left(\frac{\delta_w + z_0}{z_0}\right), \quad b = \ln\left(\frac{z_b}{\delta_w + z_0}\right) \quad (6.34)$$

Main difference is that the oscillatory component of the current u_w inside the WBL is now explicitly calculated by solving the equation

$$\frac{\partial \mathbf{u}_w^i}{\partial t} = \frac{\partial}{\partial z} \left(\kappa z u_{*cw} \frac{\partial \mathbf{u}_w^i}{\partial z} \right) \quad (6.35)$$

where $\mathbf{u}_w^i = \mathbf{u}_w - \mathbf{u}_w^o$ with \mathbf{u}_w^o given by the outside solution (6.13). The general solution of (6.35) which equals $-\mathbf{u}_w^o$ at $z = z_0$ and zero at $z = \delta_w + z_0$, has a complex form involving Kelvin functions. This can, however, be recast into a simpler logarithmic form by making the realistic assumption $\delta_w \gg z_0$ (see above). The peak wave stress, derived from the approximate solution, is then given by

$$u_{*p}^2 = \kappa u_{*cw} U_w \left[\left(\ln \left(\frac{\kappa u_{*cw}}{z_0 \omega_w} \right) - 1.15 \right)^2 + \left(\frac{\pi}{2} \right)^2 \right]^{-1/2} \quad (6.36)$$

The WBL thickness is now given by

$$\delta_w = a_r \kappa \frac{u_{*cw}}{\omega_w} \quad (6.37)$$

with $a_r = 2$. The apparent roughness z_a is given by

$$z_{0a} = (\delta_w + z_0) \left(\frac{z_0}{\delta_w + z_0} \right)^R, \quad R = \frac{u_{*m}}{u_{*cw}} \quad (6.38)$$

6.2.4 Christofferson-Jonsson model

Of the four models implemented in COHERENS the one given by [Christofferson & Jonsson \(1985\)](#) has the most complex form, although there are many similarities with the GM model. Inside the WBL, two models are used for obtaining the current and shear stress component, differing by the choice of the eddy viscosity coefficient inside the WBL. They are discussed separately below.

First method

The first method uses an eddy viscosity of the form

$$\nu_T = \beta_* z_0 u_{*cw} \quad (6.39)$$

with $\beta_* = 2.241$. The WBL thickness is defined as

$$\delta_w = \sqrt{\frac{\beta_* z_0 u_{*cw}}{\omega_w}} \quad (6.40)$$

Using (6.39), the solutions for mean current inside and outside the WBL are given by

$$z_0 < z \leq \delta_w + z_0 : \quad u_c = \frac{u_{*m}^2}{\beta_* z_0 u_{*cw}} (z - z_0) \quad (6.41)$$

$$z > \delta_w + z_0 : \quad u_c = \frac{u_{*m}}{\kappa} \ln \left(\frac{z}{\delta_w + z_0} \right) + \frac{u_{*m}^2 \delta_w}{\beta_* z_0 u_{*cw}} \quad (6.42)$$

The mean friction velocity is then obtained from the previous equations

$$\begin{aligned} z_0 < z_b \leq \delta_w + z_0 : \quad u_{*m} &= \left(\frac{u_{*cw} z_0 \beta_* u_b}{z_b} \right)^{1/2} \\ z_b > \delta_w + z_0 : \quad u_{*m} &= \frac{1}{a} \left(\sqrt{b^2 + 4au_b} - b \right) \end{aligned} \quad (6.43)$$

with

$$a = \frac{\delta_w + z_0}{\beta_* z_0 u_{*cw}}, \quad b = \frac{1}{\kappa} \ln \left(\frac{z_b}{\delta_w + z_0} \right) \quad (6.44)$$

The peak wave friction velocity is obtained by solving the equation

$$\frac{\partial \mathbf{u}_w^i}{\partial t} = \frac{\partial}{\partial z} \left(\beta_* z_0 u_{*cw} \frac{\partial \mathbf{u}_w^i}{\partial z} \right) \quad (6.45)$$

giving

$$u_{*p}^2 = U_w \sqrt{u_{*cw} \beta_* \omega_w z_0} \quad (6.46)$$

The apparent roughness becomes

$$z_a = (\delta_w + z_0) \exp \left(- \frac{\kappa u_{*m} \delta_w}{\beta_* z_0 u_{*cw}} \right) \quad (6.47)$$

Second method

The eddy viscosity selected in the second model is the same as the one used in the GM model

$$\nu_T = \kappa u_{*cw} z \quad (6.48)$$

The solutions for the mean current and shear stress u_{*m} are therefore the same as the ones for the GM model.

The solution for the wave part of the current inside the WBL takes the same form as the one in the GM model. After making some approximations the following form for the peak wave stress is derived in the CJ model

$$u_{*p}^2 = \frac{U_w u_{*cw}}{5.76} \left[\log \left(\frac{5.76 u_{*cw}}{U_w} \right) - 0.1164 + \log \left(\frac{U_w}{30 z_0 \omega_w} \right) \right]^{-1} \quad (6.49)$$

The WBL thickness is again given by (6.37) with $a_r = 0.367$. The apparent roughness z_a has the same form as in (6.38).

According to Christofferson & Jonsson (1985), the choice between method I or II depends on the value of the parameter J defined by

$$J = \frac{u_{*cw}}{30z_0\omega_w} \quad (6.50)$$

The first or second method is taken depending on whether $J < J_{crit}$ or $J > J_{crit}$ with $J_{crit} = 3.47$.

6.2.5 solution method

The formulae for the different models are applied with u_b and z_b taken at the bottom grid cell using values from the previous time step. For the SC and MD models u_{*m} , u_{*cw} , u_{*p} , δ_w and z_{0a} are obtained directly from the equation in Sections 6.2.1 and 6.2.2. For the GM and CJ models they are obtained using an iteration scheme.

The procedure for the GM model is as follows

1. Initialise u_{*m} and $u_{*p} = u_{*m}$ to their values without waves, given by equation (6.19a)–(6.19b).
2. Set $u_{*m,old} = u_{*m}$.
3. Calculate u_{*p} from (6.36), u_{cw} from (6.16), δ_w from (6.37).
4. Update u_{*m} using equations (6.34), (6.34).
5. If there is no convergence, i.e. $|u_{*m} - u_{*m,old}| > \epsilon$, then steps 2 to 4 are repeated.
6. Otherwise, z_{0a} is obtained from (6.38).

A similar iteration procedure is applied for the CJ model.

1. Obtain all parameters using the method I.
2. Evaluate J using (6.50).
3. If $J > J_{crit}$, the procedure is restarted now using method II.

6.3 Wave-current interaction within the water column

Two different types of formalisms have been developed during the past decades for describing the interaction between current and surface gravity waves within the water column. In the first one the current is decomposed into a steady and a oscillating wave part. After applying phase averaging, the effects of surface gravity waves on the current is presented by the divergence of a radiation stress in the current equations (e.g. Hasselmann, 1971; Phillips, 1977; Svendsen, 2006; Mellor, 2003). McWilliams *et al.* (2004) used an alternative approach whereby the hydrodynamic variables are split into a short-wave a long-wave and a steady part. A further asymptotic expansion is made with respect to the wave slope. Two extra (non-dissipative) terms appear on the right hand side of the horizontal current equations. The first one is the vortex force term, the second the Bernoulli pressure head term. A variant of the previous formulation is to make use of Generalized Lagrangian means (GLM), defined by Andrews & McIntyre (1978), whereby the phase-averaged mean is obtained by averaging over the displaced position during the wave cycle. As shown by Ardhuin *et al.* (2008), results are in agreement with the ones ontained with the Eulerian method. Lane *et al.* (2007) compared the radiation-stress and vortex-force (VF) representations. Although both methods appeared equivalent, preference should be given to the VF formulations.

Warner et al implemented the radiation-stress approach of Mellor (2003) in the ROMS model, but take not account of certain inconsistencies in the theory. The VF formalism was introduced in ROMS by Uchiyama *et al.* (2009, 2010) and in the MARS3D model by Bennis *et al.* (2011). In view of the previous studies it has been decided the implement the VF method in COHERENS.

Chapter 7

Sediment transport model

7.1 Introduction

This chapter describes the sediment transport model implemented in COHERENS. The multi-fraction sediment model is presented in Sections 2 to 7: general aspects (definitions, sediment density, skin roughness), critical shear stress, settling velocity, bed load transport, total load transport and suspended transport. The flocculation model is discussed in Section 8.

7.2 General aspects

7.2.1 Sediment concentrations

A sediment concentration consists of a number of particles with different sizes and masses. In the model, these concentrations are approximated by assuming a finite number N_f of size classes. Each sediment particle within a size class n has the same diameter $d_{p,n}$ and mass density $\rho_{p,n}$. A sediment concentration for class n can be presented either as a volumetric concentration c_n in units of m^3/m^3 or as a mass concentration $c_{m,n}$ in kg/m^3 . The two forms are related by

$$c_{m,n} = \rho_{p,n} c_n \tag{7.1}$$

The volumetric form is taken, for convenience, within the COHERENS code and in this documentation. Since mass concentrations are more useful than volume concentrations for post-processing, volume is converted to mass when sediment concentrations are written to the user-defined output files.

The total (volume and mass) sediment concentrations are defined by

$$c_t = \sum_{n=1}^{N_f} c_n, \quad c_{mt} = \sum_{n=1}^{N_f} \rho_{p,n} c_n \quad (7.2)$$

Another important parameter in the sediment model is the fraction of particles of class n at the sea bed, denoted by f_n . The distribution of the bed fractions is determined in the morphological module discussed in Chapter 8. However, if morphology is disabled by the user, f_n is assumed to be constant in time and should be supplied by the user as initial condition. Default is $f_n = 1/N_f$. Note that by definition one must have $\sum_{n=1}^{N_f} f_n = 1$, which is checked by the program.

Besides particle diameter and density, a series of properties can be defined for each particle class such as type of transport (bed load, total load, suspended load), formulation for critical shear stress and settling velocity and type of sediment (sand or mud). A further discussion is given below.

7.2.2 Density effects

7.2.2.1 Equation of state

The fluid density is an important parameter for calculating settling velocities and entrainment rates in the sediment transport. The equation of state that is used to calculate the “pure” (i.e. without sediment particles) water density ρ_w as function of the water temperature, salinity and pressure is discussed in Section 5.1.3. When sediment is present, the sediment mass concentrations are added to the water density

$$\rho = (1 - \sum_{n=1}^{N_f} c_n) \rho_w + \sum_{n=1}^{N_f} c_n \rho_{p,n} = (1 - c_t) \rho_w + c_{mt} \quad (7.3)$$

Here ρ is the total density of the mixture. A stable vertical sediment stratification leads to damping of turbulence and affects the settling velocity of the sediment.

7.2.2.2 Density stratification

To a large extent, the two-way coupling effects between flow and sediment transport are due to density stratification effects. [Villaret & Trowbridge \(1991\)](#) showed that the effects of the stratification from suspended sediment are very similar to the ones produced by temperature and salinity gradients. This means that the effects of sediment-turbulence interaction can be

implemented within the existing turbulence models in the same way as T and S .

The (squared) buoyancy frequency in the presence of vertical sediment stratification becomes:

$$N^2 = -\frac{g}{\rho} \frac{\partial \rho}{\partial z} = N_w^2 - g \sum_{n=1}^{N_f} \beta_{c,n} \frac{\partial c_n}{\partial z} \quad (7.4)$$

where N_w^2 is the value in the absence of sediment stratification, as given by (5.91) and $\beta_{c,n}$, the expansion coefficient for sediment fraction n , which is calculated from the density of each sediment fraction $\rho_{p,n}$ using:

$$\beta_{c,n} = \frac{1}{\rho} \frac{\partial \rho}{\partial c_n} = \frac{\rho_{p,n} - \rho_w}{\rho} \quad (7.5)$$

The baroclinic component of the horizontal pressure gradient contains an additional term due to horizontal sediment stratification. Equation (12.220) now becomes

$$-\frac{\partial q}{\partial x_i} \simeq g \int_z^\zeta \left(\beta_T \frac{\partial T}{\partial x_i} - \beta_S \frac{\partial S}{\partial x_i} - \sum_{n=1}^N \beta_{c,n} \frac{\partial c_n}{\partial x_i} \right) dz' \quad (7.6)$$

In transformed coordinates the following term is added to right hand side of (12.221) for each fraction n :

$$F_i^{c,n} = -g \int_z^\zeta \beta_{c,n} \left(\frac{\partial c_n}{\partial x_i} \Big|_s - \frac{\partial c_n}{\partial z'} \frac{\partial z'}{\partial x_i} \Big|_s \right) dz' \quad (7.7)$$

The numerical methods, described in Section 12.3.13 for discretising the baroclinic gradient are easily extended to include sediment stratification.

7.2.3 Dimensionless parameters

A dimensional analysis by Yalin (1977) shows that the sediment transport can be expressed by a number of dimensionless parameters:

$$Re_{*p,n} = \frac{u_* d_{p,n}}{\nu} \quad (7.8)$$

$$s_n = \frac{\rho_{p,n}}{\rho_w} \quad (7.9)$$

$$\theta_n = \frac{\rho_w u_*^2}{(\rho_{p,n} - \rho_w) g d_{p,n}} = \frac{u_*^2}{(s_n - 1) g d_{p,n}} \quad (7.10)$$

$$d_{*p,n} = d_{p,n} \left[(s_n - 1) \frac{g}{\nu^2} \right]^{1/3} \quad (7.11)$$

$$\Phi_n = \frac{q_n}{\sqrt{(s_n - 1) g d_{p,n}^3}} \quad (7.12)$$

where u_* is the bottom friction velocity, $Re_{*p,n}$ is the turbulent particle Reynolds number, θ_n the dimensionless shear stress or Shields parameter (Shields, 1936), s_n the relative density and q_n the sediment (bed or total) load per unit width in m^2/s (see Sections 7.5 and 7.6). The non-dimensional parameters are often used to describe sediment properties, such as critical shear stresses or settling velocities, which are discussed below.

As explained in Chapter 6, different types of bottom stress can be identified in the presence of wave-current interaction. The Shields parameters, obtained by replacing u_* in (7.10) with u_{*m} , u_{*cw} , u_{*p} (representing respectively, mean, maximum and peak values) or τ_m , τ_{cw} , τ_p are notated by respectively $\theta_{m,n}$, $\theta_{cw,n}$, $\theta_{p,n}$. In the absence of waves, $\theta_{m,n} = \theta_{cw,n} = \theta_n$ and $\theta_{p,n} = 0$. The convention is made that, if a parameter appears without subscript, the value corresponding to the mean stress (subscript m) is assumed in the wave case. When waves are present, the mean Shields parameter θ_m or mean bottom stress τ_m , is used for bed load transport (Section 7.5) or sea bed erosion (Section 7.7.2).

The kinematic viscosity ν is an important parameter that has a significant influence on the settling velocity and the critical bed stress. Its value can be selected either as a user-defined constant or as a temperature dependent value using the [ITTC \(1978\)](#) equation for sea water

$$\nu = 10^{-6} [1.7688 + (0.659 \cdot 10^{-3} (T - 1) - 0.05076) (T - 1)] \quad (7.13)$$

Here T is the water temperature in $^{\circ}\text{C}$ and ν is given in m^2/s .

7.2.4 Roughness length and bed shear stresses

The bed shear stress is considered as the physical parameter which has the largest impact on sediment transport, since it controls how much of the sediment in the bed layer becomes suspended in the water column. In the hydrodynamic part of **COHERENS**, bottom stresses are calculated using either a linear or a quadratic friction law (see Section 5.9). In the sediment transport module, however, a quadratic friction law is always taken, as given by equation (5.319) in 3-D or (5.320) in 2-D mode.

The roughness length z_0 can be calculated using the following relation:

$$z_0 = 0.11 \frac{\nu}{u_{*b}} + \frac{k_b}{30} \quad (7.14)$$

which reduces for a rough bed to

$$z_0 = \frac{k_b}{30} \quad (7.15)$$

Here, k_b is the roughness height from Nikuradse, ν the kinematic viscosity and u_{*b} the friction velocity¹.

Different kinds of roughness height are considered:

- Physical or “form” roughness k_b representing the heights of the elements composing the bottom roughness. The corresponding physical bottom stress is used as boundary condition for the momentum flux at the bottom.
- “Skin” roughness k_s usually related to the median particle size d_{50} . Skin stress is used in the sediment module to calculate the amount of sediment material resuspended from the sea bed and the bed load transport, once the stress exceeds a critical value (see below).
- The main effect of surface waves on the bottom stress is a significant increase of both the form and skin roughness heights. A new roughness length and height, denoted by z_a and $k_a = 30z_a$ are obtained from formulae for current-wave interaction theories (see Section 6.2).

In the following, a superscript “s” of “f” denotes a quantity determined using a “skin” or “form” roughness, e.g. C_{db}^s or τ_b^f . Since skin roughness will be used in most of the formulae below, the convention is made that, when the superscript is omitted, the value based on skin roughness will be taken.

The skin roughness length is taken as proportional to median grain size, i.e.

$$z^s = a_{sg} d_{50} = z_{sg} \quad (7.16)$$

where z_{sg} is the grain roughness. Default value for a_{sg} is 0.0833.

Following Soulsby (1997) the form roughness length is defined as the sum of the grain roughness, wave ripple roughness and bedload roughness

$$z^f = z_{sg} + z_{sr} + z_{st} \quad (7.17)$$

Formulations for the ripple and bedload roughness are given below.

¹Since the friction velocity always refers to the bottom in this chapter, the subscript b will be omitted in the following.

ripple roughness models

Wave ripples are generated at the sea bed by the action of a wave stress and are characterised by three parameters: ripple height η_r , ripple length λ_r and ripple slope $\vartheta_r = \eta_r/\lambda_r$. The ripple parameters are determined from a wave ripple model. Four schemes, based on laboratory data and measurements in coastal marine waters, have been implemented.

1. Wiberg & Harris (1994)

The Wiberg & Harris (1994) ripple predictor formulae need to be solved by an iterative procedure. The version, implemented in COHERENS uses the adapted (non-iterative) version described in Malarkey & Davies (2003). Ripple length and height are related by

$$\frac{d_0}{\eta_r} = \exp \left[7.59 - \sqrt{33.6 - 10.53 \ln \left(\frac{d_0}{\lambda_r} \right)} \right] \quad (7.18)$$

where $d_0 = 2A_b$ is the bottom wave orbital displacement and A_b the bottom excursion amplitude defined by (6.7). The procedure is as follows:

- Equation (7.18) is first solved for $\eta_r = \eta_{ano}$ with $\lambda_r = \lambda_{ano} = 535d_{50}$ giving a value for η_{ano} where the subscript *ano* stands for anorbital 3-D ripples.
- The ripple length is then calculated by

$$\begin{aligned} \frac{d_0}{\lambda_r} &= \frac{1}{0.62} \quad \text{if} \quad \frac{d_0}{\eta_{ano}} < 20 \\ \frac{d_0}{\lambda_r} &= \frac{d_0}{535d_{50}} e^{-f} \quad \text{if} \quad 20 \leq \frac{d_0}{\eta_{ano}} \leq 100 \\ \frac{d_0}{\lambda_r} &= \frac{d_0}{535d_{50}} \quad \text{if} \quad \frac{d_0}{\eta_{ano}} > 100 \end{aligned} \quad (7.19)$$

with

$$f = \ln(\lambda_{ano}/(0.62d_0))/\ln(0.01d_0/\eta_{ano})/\ln 5 \quad (7.20)$$

- The ripple height is then obtained by inserting the value for d_0/λ_r from (7.19) into (7.18). The ripple slope ϑ_r is then given by the ratio η_r/λ_r .

2. Li *et al.* (1996)

The formulation is an adapted version of the original scheme proposed

by [Grant & Madsen \(1982\)](#). Ripple parameters depend on the values of three Shield parameters: peak wave stress θ_p , critical stress θ_{cr} for the initiation of bed load and a critical stress θ_{bf} for the ripple breakoff. The latter is given by

$$\theta_{bf} = 1.8\theta_{cr}S_*^{0.6} \quad \text{with} \quad S_* = \frac{d_{50}}{4\nu}\sqrt{(s-1)gd_{50}} \quad (7.21)$$

Fractional dependency has been removed by averaging over all fractions

$$s = \sum_{n=1}^{N_f} f_n s_n, \quad \theta_{cr} = \sum_{n=1}^{N_f} f_n \theta_{cr,n} \quad (7.22)$$

The wave-ripple predictor is as follows. For $\theta_{cr} \leq \theta_p < \theta_{bf}$ one has

$$\begin{aligned} \eta_r &= 0.101A_b(\theta_p/\theta_{cr})^{-0.16} \\ \lambda_r &= 0.364A_b(\theta_p/\theta_{cr})^{-0.12} \\ \vartheta_r &= 0.278(\theta_p/\theta_{cr})^{-0.04} \end{aligned} \quad (7.23)$$

while if $\theta_{bf} \leq \theta_p$

$$\begin{aligned} \eta_r &= 0.356A_bS_*^{-0.8}(\theta_p/\theta_{cr})^{-1.5} \\ \lambda_r &= 1.079A_bS_*^{-1.4}(\theta_p/\theta_{cr})^{-0.5} \\ \vartheta_r &= 0.33S_*^{0.6}(\theta_p/\theta_{cr})^{-1} \end{aligned} \quad (7.24)$$

When $\theta_p < \theta_{cr}$ the ripple parameters are the same as the ones obtained at the previous time step.

3. Soulsby & Whitehouse (2005)

$$\begin{aligned} \lambda_r &= A_b \left[1 + 0.00187 \frac{A_b}{d_{50}} \left(1 - \exp(-0.0002(A_b/d_{50})^{1.5}) \right) \right]^{-1} \\ \vartheta_r &= 0.15 \left[1 - \exp(-(5000d_{50}/A_b)^{3.5}) \right] \\ \eta_r &= \lambda_r \vartheta_r \end{aligned} \quad (7.25)$$

4. A wave-ripple predictor model was developed by [Goldstein et al. \(2013\)](#) using machine learning from field and laboratory data sets

$$\lambda_r = \frac{d_0}{1.12 + 2180d_{50}}$$

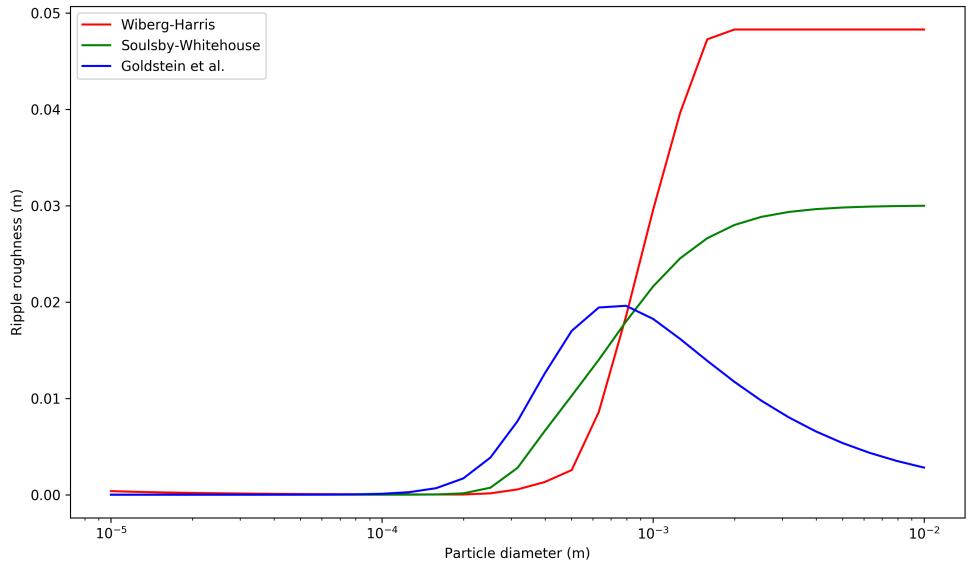


Figure 7.1: Ripple roughness (m) according to three formulations using a bottom excursion amplitude A_b of 1.45 m. The [Li et al. \(1996\)](#) scheme is not shown since it depends also on the shear stress.

$$\begin{aligned}\vartheta_r &= \frac{3.42}{22 + (\lambda_r/(1000d_{50}))^2} \\ \eta_r &= \lambda_r \vartheta_r\end{aligned}\quad (7.26)$$

with d_0 and d_{50} given in m. The formulae should be valid for median grain sizes between $10\mu\text{m}$ and $400\mu\text{m}$.

Values of the ripple roughness are displayed as function of particle diameter in Figure 7.1 for three ripple models. It is seen, as mentioned above, that the [Goldstein et al. \(2013\)](#) formula deviates from the other formulations when d_p becomes larger than $\simeq 400\mu\text{m}$.

Once the ripple parameters have been defined the ripple roughness is given by

$$z_{sr} = a_{sr} \frac{\eta^2}{\lambda} \quad (7.27)$$

where a_{sr} is a user-defined constant. [Grant & Madsen \(1982\)](#) proposed the value 0.923 for a_{sr} which is also taken as the default.

bedload roughness models

1. Grant & Madsen (1982)

$$z_{st} = 5.32 \frac{s + 0.5}{s - 1} \frac{\tau_{cr}}{g} \left[\sqrt{\frac{\tau}{\tau_{cr}}} - 0.7 \right]^2 \quad (7.28)$$

where the fractional averages s and τ_{cr} are defined by (7.22).

2. Wiberg & Rubin (1989)

$$\begin{aligned} z_{st} &= 0.0381 \frac{d_{50} \tau_*}{1 + a_{wr} \tau_*} \\ \tau_* &= \frac{\theta}{\theta_{cr}} \\ a_{wr} &= 0.0204(\ln d_{50})^2 + 0.022(\ln d_{50}) + 0.0709 \end{aligned} \quad (7.29)$$

3. Nielsen (1992)

$$z_{st} = 5.667 d_{50} \max(0, \sqrt{\theta - \theta_{cr}}) \quad (7.30)$$

Switches

The following switches are available²:

iopt_kinvisc	Selects type of kinematic viscosity.
	0: User-selected uniform value. Default.
	1: From the ITTC (1978) equation.
iopt_sed	The multi-fraction sediment model is activitated by setting this switch to 1.
iopt_sed_dens_grad	Disables/enables the inclusion of sediment stratification in the formulations for the buoyancy frequency and baroclinic pressure gradient.
	0: Disabled. Default.
	1: Enabled.
iopt_sed_rough	Selects formulation for the skin bottom roughness length.
	0: Same value as the form roughness.
	1: User-defined spatially uniform value.

²A * means that different values can be selected for different fractions.

- 2: As function of the median grain size, given by (7.16).

`iopt_sed_rough_btr` Selects type of model for the bedload roughness length.

- 0: Disabled. Default.
- 1: [Grant & Madsen \(1982\)](#).
- 2: [Wiberg & Rubin \(1989\)](#).
- 3: [Nielsen \(1992\)](#).

`iopt_sed_rough_rip` Selects type of wave ripple model.

- 0: Disabled. Default.
- 1: [Wiberg & Harris \(1994\)](#).
- 2: [Li *et al.* \(1996\)](#).
- 3: [Soulsby & Whitehouse \(2005\)](#).
- 4: [Goldstein *et al.* \(2013\)](#).

`iopt_type*` Selects type of sediment.

- 1: Sand.
- 2: Mud.

7.3 Critical shear stress

In many transport relations, a critical shear stress τ_{cr} is required, which is the value of the bed shear stress at which the sediment particles start to move (the threshold of motion).

The sediment transport model in COHERENS has an option available for the user to manually set the critical shear stress. The user can either set the critical shear stress to a uniform value (depending on size class) throughout the whole domain, or to a spatially varying value.

The value of the bed shear stress is in engineering practice obtained from the Shields curve, which relates the non-dimensional critical shear stress θ_{cr} to the particle Reynolds number $Re_{*p,n}$. For numerical models, some fits to this curve are made, expressing θ_{cr} as a function of d_{*p} rather than $Re_{*p,n}$, in order to avoid the iteration process necessary for the determination of the threshold of motion in the original curve obtained by [Shields \(1936\)](#), as both θ_n and $Re_{*p,n}$ are a function of u_* . The following formulations are available.

1. A constant value is selected by the user.

2. Brownlie (1981) obtained the following relation from a fit through the data:

$$\theta_{cr,n} = 0.22d_{*p,n}^{-0.9} + 0.06 \times 10^{(-7.7d_{*p,n}^{-0.9})} \quad (7.31)$$

3. An alternative form, also obtained by fitting of the Shields curve, is proposed by Soulsby & Whitehouse (1997)

$$\theta_{cr,n} = \frac{0.3}{1 + 1.2d_{*p,n}} + 0.055 (1 - e^{-0.02d_{*p,n}}) \quad (7.32)$$

4. A simpler formulation, used by Wu *et al.* (2000), is obtained by assuming a constant value for the critical Shields stress

$$\theta_{cr} = 0.03 \quad (7.33)$$

This equation should, in principle, only be used in combination with the bed load and total load equations from Wu *et al.* (2000).

The critical shear stress is shown as function of the particle diameter in Figure 7.2 for the three schemes.

The final value of the critical shear stress is obtained by multiplying its value $\theta_{cr0,n}$ from one of the above expressions with two additional factors. The first $\xi_{he,n}$ represents the effect of hiding and exposure, the second $\xi_{sl,n}$ the effect of the bed slope

$$\theta_{cr,n} = \xi_{he,n} \xi_{sl,n} \theta_{cr0,n} \quad (7.34)$$

A discussion of the two effects is given below. By default, the two effects are disabled, i.e. $\xi_{he,n} = \xi_{sl,n} = 1$.

7.3.1 Hiding and exposure

Hiding and exposure effects can be included for the critical shear stress provided that at least two sediment fractions are present. Two methods are available in COHERENS. The first one uses the formulation of Wu *et al.* (2000) based upon a stochastic relation between size and gradation of bed materials and the hidden and exposed probabilities. The probability that particles with diameter d_m are in front of particles with diameter d_n can be assumed to be the fraction f_m in the bed material of particles with diameter d_m . The total hidden and exposed probabilities of particles with diameter d_n can be described by:

$$p_{h,n} = \sum_{m=1}^{N_f} f_m \frac{d_{p,m}}{d_{p,n} + d_{p,m}} \quad (7.35)$$

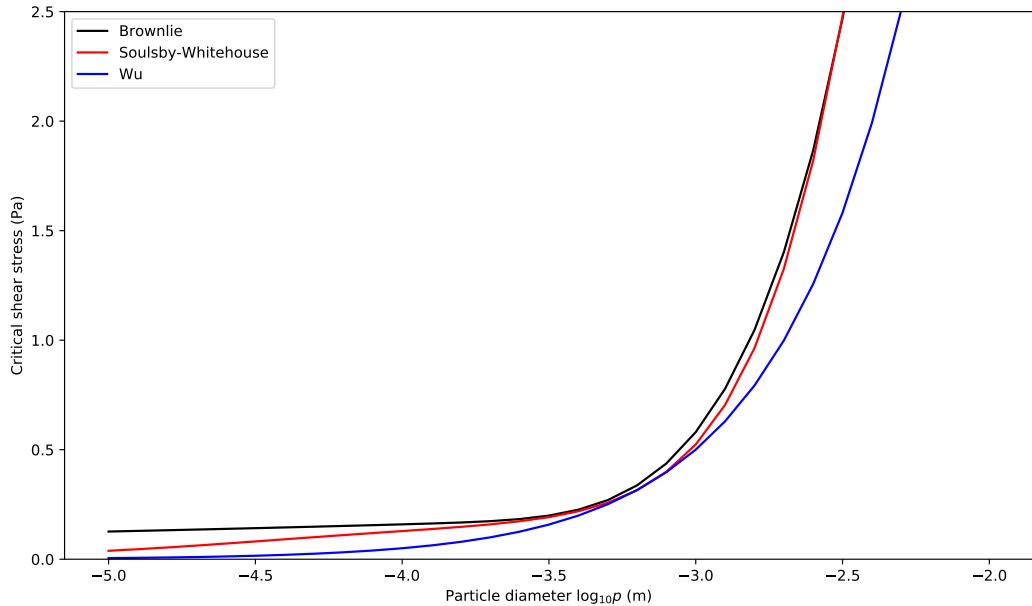


Figure 7.2: Critical shear stress τ_{cr} (Pa) according to the three different formulations using $\nu=10^{-6}\text{m}^2/\text{s}$, $s=2.65$.

$$p_{e,n} = \sum_{m=1}^{N_f} f_m \frac{d_{p,n}}{d_{p,n} + d_{p,m}} \quad (7.36)$$

so that $p_{h,n} + p_{e,n} = 1$. The hiding/exposure factor is then defined by:

$$\xi_{he,n} = \left(\frac{p_{e,n}}{p_{h,n}} \right)^{-m_h} = \left(\frac{1 - p_{h,n}}{p_{h,n}} \right)^{-m_h} \quad (7.37)$$

where m_h is an empirical parameter which [Wu et al. \(2000\)](#) determined as $m_h = 0.6$ in their study.

Alternatively, hiding and exposure can be calculated with the equation of [Ashida & Michiue \(1972\)](#), which is an adapted form of [Egiazaroff \(1965\)](#)

$$\xi_{he,n} = \begin{cases} 0.8429 \frac{d_{50}}{d_{p,n}} & \text{if } \frac{d_{50}}{d_{p,n}} < 0.38889 \\ \left[\frac{\log 19}{\log 19 - \log(d_{50}/d_{p,n})} \right]^2 & \text{if } \frac{d_{50}}{d_{p,n}} \geq 0.38889 \end{cases} \quad (7.38)$$

where d_{50} is the median grain size diameter, which, in case multiple size fractions are present, means that particles with sizes less than d_{50} account for 50% of the total mass.

7.3.2 Bed slope

The critical shear stress is also influenced by the bed slope in the current direction

$$\xi_{sl} = \left(1 - \frac{\partial h}{\partial c}\right) \quad (7.39)$$

where $\partial h / \partial c$ is the bed slope along the current direction, defined by (7.69) below. Since many of the bed boundary conditions for suspended sediment concentrations depend on the critical Shields parameter, this correction will also affect the amount of sediment in suspension.

Switches

The critical shear stress is evaluated by means of the following switches

iopt_bstres_cr* Selects the formulation for the critical bottom shear stress.

- 1: User-defined value for each fraction. Default.
- 2: [Brownlie \(1981\)](#) as given by (7.31).
- 3: [Soulsby & Whitehouse \(1997\)](#) as given by (7.32).
- 4: Constant value for the critical Shield parameter from [Wu et al. \(2000\)](#).

iopt_hidexp* Selects the type of model for hiding and exposure.

- 0: Disabled. Default.
- 1: [Wu et al. \(2000\)](#) as given by (7.37).
- 2: [Ashida & Michiue \(1972\)](#) as given by (7.38).

iopt_slope* Selects the type of slope factor.

- 0: Disabled. Default.
- 1: Enabled using (7.39).

7.4 Settling velocity

Different methods are available to calculate the settling velocity of sediment particles.

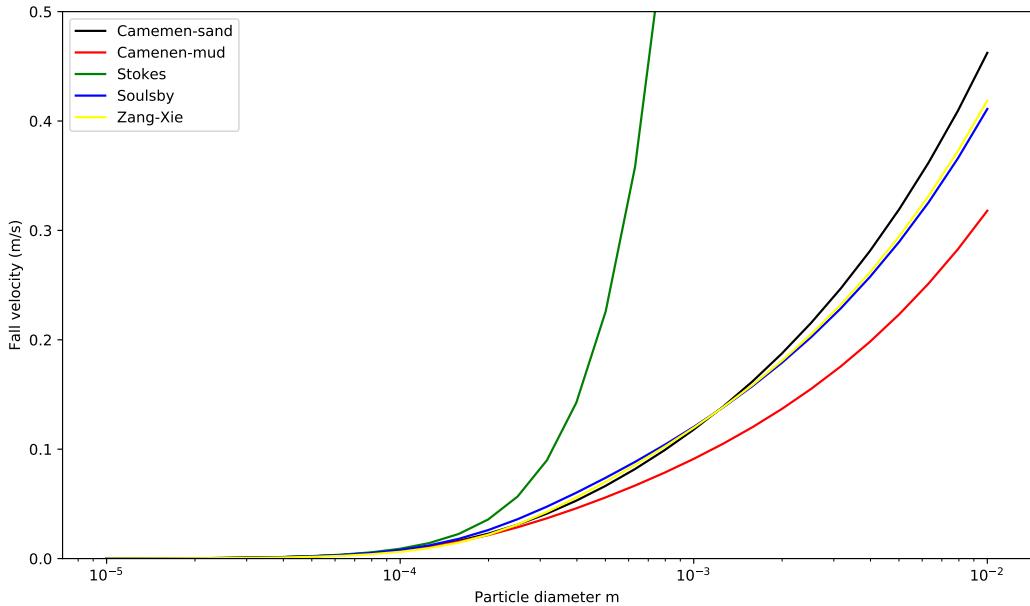


Figure 7.3: Settling velocity (m/s) according to the five different formulations. Parameters are the same as in Figure 7.2.

1. A general formula was derived by [Camenen \(2007\)](#), based on two formulations for the drag coefficient:

$$w_{s,n} = \frac{\nu}{d_{p,n}} Re_{p,n} = \frac{\nu}{d_{p,n}} \left[\sqrt{\frac{1}{4} \left(\frac{A}{B} \right)^{2/m} + \left(\frac{4 d_{*p,n}^3}{3 B} \right)^{1/m}} - \frac{1}{2} \left(\frac{A}{B} \right)^{1/m} \right]^m \quad (7.40)$$

with $d_{*p,n}$ defined by (7.11) and $Re_{p,n} = w_{s,n} d_{p,n} / \nu$ a non-turbulent particle Reynolds number using the fall velocity instead of the bottom friction velocity used in the definition (7.8) for the turbulent Reynolds number. The coefficients A , B and m have been given different values in the literature. [Camenen \(2007\)](#) recalibrated these coefficients using results of [Dietrich \(1982\)](#) and data from 11 previous studies. They found different values depending on the type of the sediment. In case of sand, $A = 24.6$, $B = 0.96$, $m = 1.53$, whereas for mud $A = 26.8$, $B = 2.11$ and $m = 1.19$.

2. For small Reynolds numbers ($Re_{p,n} \ll 1$), the settling velocity can be calculated from the Stokes equation ([Stokes, 1847](#)), valid for small spherical particles

$$w_{s,n} = \frac{(s_n - 1) g d_{p,n}^2}{18 \nu} \quad (7.41)$$

where s_n is the relative density defined by (7.9), g the acceleration due to gravity, $d_{p,n}$ the particle diameter and ν the viscosity.

3. The following formulation is recommended by [Soulsby \(1997\)](#) for natural sands

$$w_{s,n} = \frac{\nu}{d_{p,n}} \left[\sqrt{10.36^2 + 1.049d_{*p,n}^3} - 10.36 \right] \quad (7.42)$$

4. Another equation for the settling velocity is the one by [Zhang & Xie \(1993\)](#)

$$\begin{aligned} w_{s,n} &= \sqrt{\left(13.95 \frac{\nu}{d_{p,n}}\right)^2 + 1.09(s-1)gd_{p,n} - 13.95 \frac{\nu}{d_{p,n}}} \\ &= \frac{\nu}{d_{p,n}} \left[\sqrt{194.6 + 1.09d_{*p,n}^3} - 13.95 \right] \end{aligned} \quad (7.43)$$

Settling velocities are plotted as function of particle diameter for the different formulations in Figure 7.3. Note that the Stokes formula largely exceeds the values of the other formulations if d_p is larger than $\sim 400\mu\text{m}$.

The final value of the settling velocity is obtained by multiplying its value $w_{s0,n}$ from one of the expressions above with one or two additional factors, representing respectively the effect of hindered settling and flocculation

$$w_{s,n} = f_{hs,n} f_{fl,n} w_{s0,n} \quad (7.44)$$

A discussion of the two effects is given below. By default, the two effects are disabled, i.e. $f_{hs,n} = f_{fl,n} = 1$.

7.4.1 Hindered settling

In case of high concentrations (mass concentrations larger than 3 g/l) the settling of particles is not only influenced by the surrounding fluid and the weight and shape of the particles, but also by the other particles in suspension. The settling of the particles is reduced by a number of processes, e.g. return flow, particle collisions, changed mixture viscosity, buoyancy due to increased mixture density and wake formation.

For sand, hindered settling can be calculated with the equation of [Richardson & Zaki \(1954\)](#). They described the settling velocity in high sediment concentrations as a function of the sediment concentration

$$f_{hs,n} = (1 - c_t)^{n_h} \quad (7.45)$$

In this equation, c_t is the total volume sediment concentration given by (7.2) and n_h a user-defined constant. Note that this formula is not advisable for cohesive sediments because it does not take into account the maximum sediment concentration (i.e. the settling velocity should reduce to zero at the maximum packing concentration c_{max}). This is not a severe disadvantage, because in most practical applications, the concentrations are not so high, except close to the sea bed. Note that some hindered settling effects are already calculated when the fluid density depends itself on the sediment concentrations, and that (7.45) may become less valid when a net deposition occurs (Breugem, 2012).

Winterwerp & van Kesteren (2004) introduced a different formula for mud

$$f_{hs,n} = \frac{(1 - c_{f,n})(1 - c_t)}{1 + 2.5c_{f,n}} \quad (7.46)$$

where $c_{f,n} = c_n/c_{gel}$, c_{gel} is the gelling concentration (the mass concentration at which the mud flocs form a space filling network). A constant value is taken for c_{gel} in COHERENS. Default value is 0.075.

7.4.2 Influence of flocculation

In COHERENS, the influence of turbulence and sediment concentration on the flocculation of mud flocs can be simulated. For simulating the influence of turbulence the empirical model by Van Leussen (1994) is used for the fall velocity of settling flocs

$$f_{fl,n} = \frac{1 + aG}{1 + bG^2} \quad (7.47)$$

Here a , b are empirical constants and G is the shear rate, defined as

$$G = \sqrt{\varepsilon/\nu} \simeq \sqrt{\frac{\nu_T M^2 - \lambda_T N^2}{\nu}} \quad (7.48)$$

A similar empirical approach to flocculation, which considers the influence of the sediment concentration is given in Van Rijn (2007b)

$$f_{fl,n} = [4 + \log_{10}(2c_n/c_{gel})]^{\alpha_{fl}} \quad (7.49)$$

with $1 \leq f_{fl,n} \leq 10$ and α_{fl} a user-defined constant with a minimum of 0 and a maximum of 3, whose default value 2.19 in COHERENS was obtained from calibration of flocculation data in the Scheldt and harbour of Zeebrugge. Note that Van Rijn (2007b) suggests $\alpha_{fl} = d_{sand}/d_{50} - 1$, with d_{sand} the diameter of the transition from sand to silt (62 μm).

It is remarked that flocculation effects are only allowed for mud fractions. This is checked by the program.

Switches

The following switches are available for evaluating the settling velocity²

iopt_ws*	Type of method for the settling velocity.
1:	User-defined value for each fraction.
2:	Camenen (2007) formulation (7.40). The coefficients A , B , m depending on the type of sediment.
3:	Stokes formula (7.41).
4:	Soulsby (1997) formula (7.42).
5:	Zhang & Xie (1993) equation (7.43).
iopt_ws_floc*	Type of flocculation factor for the settling velocity
0:	Flocculation effect disabled.
1:	Van Leussen (1994) equation (7.47).
2:	Van Rijn (2007b) equation (7.49).
iopt_ws_hindset*	Formulation for hindered settling.
0:	Hindered settling disabled.
1:	Richardson & Zaki (1954) equation (7.45).
2:	Winterwerp & van Kesteren (2004) formula (7.46).

7.5 Bed load transport

7.5.1 Introduction

In this section an overview of the available models for bed load transport is given. These models are applicable to non-cohesive sediment transport and can be used in combination with suspended transport equations. Bed load is active in the near-bed layer with thickness often taken as $z_{sb,n} \approx 2d_{p,n}$.

Bed load is usually expressed as the *intensity of solid discharge*, which can be written in non-dimensional form (see also equation (7.12)):

$$\Phi_{b,n} = \frac{q_{b,n}}{\sqrt{(s_n - 1)gd_{p,n}^3}} \quad (7.50)$$

where $q_{b,n}$ is the bed load solid discharge for size class n and per unit width in m^2/s . Note that in the model output files, bed load, suspended load and

total load are multiplied by the particle density. The corresponding unit then becomes kg/m/s.

The bed load transport is assumed to occur below the roughness level z_0 , i.e. below the lowest grid cell. Except stated otherwise, the bed load transport takes place along the current direction

$$\mathbf{q}_{b,n} = q_{b,n} \mathbf{1}_c \quad (7.51)$$

where

$$\mathbf{1}_c = (\cos \phi_c, \sin \phi_c) \quad (7.52)$$

is the unit vector in the current direction and ϕ_c the angle between the X-axis and the current direction.

applicability of the bed load formulae

Every sediment transport formulation has been developed and calibrated under controlled conditions where a certain range of parameters such as particle diameters has been covered. The formulations will therefore be valid under the conditions covered by the experiments that lead to the formula with tuned parameters. The model user should take these limitations into account when applying a formula. The range of applicability of a number of transport formula has been given in table 7.1.

Table 7.1: Overview of applicability of sediment transport formula, i.e. conditions of the measurements used as calibration data (if available).

	$d_{p,n}$ [mm]	d_x , for non-uniform granulate
Meyer-Peter & Müller (1948)	3.1-28.6	d_{50}
Engelund & Fredsøe (1976)	0.19-0.93	d_{50}
Van Rijn (1993)	0.2-2.0	d_{50}
Wu <i>et al.</i> (2000)	0.06-32	graded

Six formulations are available for bed load transport. They are described below.

7.5.2 Meyer-Peter and Mueller (1948)

The formula of Meyer-Peter & Müller (1948) relates the intensity of solid discharge Φ_n to the the Shields parameter θ_n , with the following formula

$$\Phi_{b,n} = 8f_n(\theta_n - \theta_{cr,n})^{3/2} \quad (7.53)$$

or

$$q_{b,n} = \frac{8f_n}{(s_n - 1)g} \left(\frac{\tau_b - \tau_{cr,n}}{\rho} \right)^{3/2} \quad (7.54)$$

where $\theta_{cr,n}$ is the critical dimensionless shear stress, the threshold above which motion of the sediment particles begins.

7.5.3 Engelund and Fredsøe (1976)

From an equilibrium of agitating and stabilizing forces, a dimensionless expression was found for the bed load transport. The agitating forces consist of drag and lift, while the stabilising forces are reduced gravity and friction.

Using measurements to determine the parameters, and using the number of particles in the bed load layer per unit area, which is based on a particle probability p_n , the following bed load transport equation is found

$$\begin{aligned} q_{b,n} &= 9.3f_n \frac{\pi}{6} d_{p,n} p_n u_* (1 - 0.7\sqrt{\theta_{cr,n}/\theta_n}) \\ &= 4.9f_n d_{p,n} p_n (u_* - 0.7u_{*cr,n}) \end{aligned} \quad (7.55)$$

or, in non-dimensional form:

$$\Phi_{b,n} = 4.9p_n f_n (\sqrt{\theta_n} - 0.7\sqrt{\theta_{cr,n}}) \quad (7.56)$$

The probability of mobility of a sediment particle p_n is expressed as follows:

$$\begin{aligned} p_n &= \left[1 + \left(\frac{\pi f_d / 6}{\theta_n - \theta_{cr,n}} \right)^4 \right]^{-1/4} \quad \text{if } \theta_n > \theta_{cr,n} \\ &= 0 \quad \text{otherwise} \end{aligned} \quad (7.57)$$

where f_d is the dynamic friction coefficient of a grain, assumed to be equal to 0.51.

This relation was proven to be accurate for fine to medium sands, with experiments using sand diameters of 190, 270 and 930 μm .

7.5.4 Van Rijn (1984b)

Van Rijn (1984b) used the method of Bagnold (1954) to determine the bed load sediment transport. Bed load transport is here dominated by particle saltation with height δ_b causing a bed load concentration c_b with particles moving at a velocity equal to U_p . Unlike in the formulation of Engelund & Fredsøe (1976), only drag and lift forces as well as reduced weight are taken into account. The bed load transport was then assumed to be $q_b = U_p \delta_b c_b$.

The bed load transport by currents for particles with diameter in the range of 200 to 2000 μm is determined by:

$$\Phi_{b,n} = 0.053 f_n \tau_{*n}^{2.1} d_{*p,n}^{-0.3} \quad (7.58)$$

or

$$q_{b,n} = 0.053 f_n \sqrt{(s_n - 1) g d_{p,n}^3} \tau_{*n}^{2.1} d_{*p,n}^{-0.3} \quad (7.59)$$

where

$$\tau_{*n} = \max \left(\frac{\theta_n}{\theta_{cr,n}} - 1, 0 \right) \quad (7.60)$$

7.5.5 Wu et al. (2000)

This equation is specially recommended for the calculation of graded sediment, because in its determination, the different fractions were explicitly taken into account.

The bed load sediment transport is then determined with:

$$\Phi_{b,n} = 0.0053 f_n \tau_{*n}^{2.2} \quad (7.61)$$

or

$$q_{b,n} = 0.0053 f_n \sqrt{(s_n - 1) g d_{p,n}^3} \tau_{*n}^{2.2} \quad (7.62)$$

Note that in the [Wu et al. \(2000\)](#) paper, the critical shear stress θ_{cr} is set to 0.03.

7.5.6 Soulsby (1997)

In the previous formulae the wave effects are included implicitly through the Shields parameter. The bed load transport equation by [Soulsby \(1997\)](#) takes explicitly account of wave effects. The components Φ_{bc} and Φ_{bn} along the directions parallel, respectively perpendicular to the current direction are obtained as follows

$$\begin{aligned} \Phi_{bc,1n} &= 12 \sqrt{\theta_m} (\theta_m - \theta_{cr,n}) \\ \Phi_{bc,2n} &= 12 (0.95 + 0.19 \cos(2\phi_{cw})) \theta_m \sqrt{\theta_p} \\ \Phi_{bc,n} &= f_n \max(\Phi_{bc,1n}, \Phi_{bc,2n}) \\ \Phi_{bn,n} &= 12 f_n \frac{0.19 \theta_m \theta_p^2 \sin(2\phi_{cw})}{\theta_p^{1.5} + 1.5 \theta_m^{1.5}} \end{aligned} \quad (7.63)$$

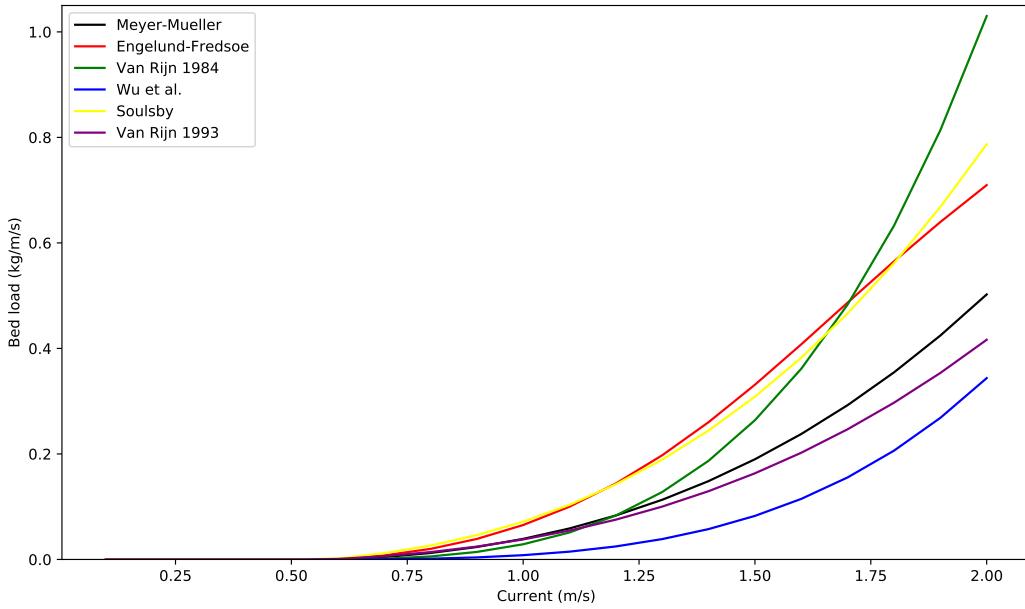


Figure 7.4: Bed load transport (kg/m/s) as function of velocity for the six available schemes without wave effects, using a particle size of 1 mm and a critical Shields parameter $\theta_{cr} = 0.03$ (Wu *et al.*, 2000).

where $\Phi_{bc,n} = \Phi_{bn,n} = 0$ if $\theta_m \leq \theta_{cr,n}$, ϕ_{cw} the angle between current and wave propagation direction. The dimensional form of (7.63) is given by

$$\begin{aligned}
 q_{bc,1n} &= 12 \frac{u_{*m}^2 (u_{*m}^2 - u_{*cr,n}^2)}{(s_n - 1)g} \\
 q_{bc,2n} &= 12 u_{*m}^2 u_{*p} \frac{0.95 + 0.19 \cos(2\phi_{cw})}{(s_n - 1)g} \\
 q_{bc,n} &= f_n \max(q_{bc,1n}, q_{bc,2n}) \\
 q_{bn,n} &= \frac{12 f_n}{(s_n - 1)g} \frac{0.19 u_{*m}^2 u_{*p}^4 \sin(2\phi_{cw})}{u_{*p}^3 + 1.5 u_{*m}^3}
 \end{aligned} \tag{7.64}$$

From the equations it can be seen that when the wave direction is perpendicular or along the current direction, the component at right angle to the current direction is zero. No wave asymmetry is taken into account.

Note that in the absence of waves, one has $\Phi_{bc,2n} = \Phi_{bn,n} = 0$ so that

$$\Phi_{b,n} = \Phi_{bc,n} = 12 f_n \sqrt{\theta_n} \max(\theta_n - \theta_{cr,n}, 0) \tag{7.65}$$

or

$$q_{b,n} = q_{bc,n} = \frac{12 f_n u_* \max(u_*^2 - u_{*cr,n}^2, 0)}{(s_n - 1)g} \tag{7.66}$$

The formula is implemented on the numerical grid by computing $\Phi_{bc,n}$ and $\Phi_{bn,n}$ both at U- and V-velocity points, after which they are projected on the X- and Y- grid axes. More details are given in Section 7.7.4.6.

7.5.7 Van Rijn (1993)

The bed load formula is the one given in [Van Rijn \(1993\)](#)

$$\Phi_{b,n} = f_n \frac{\gamma d_{p,n} d_{*p,n}^{-0.3} u_* \tau_{*n}^\eta}{\sqrt{(s_n - 1) g d_{p,n}^3}} \quad (7.67)$$

or

$$q_{b,n} = f_n \gamma d_{p,n} d_{*p,n}^{-0.3} u_* \tau_{*n}^\eta \quad (7.68)$$

with τ_{*n} defined by (7.60) and $\gamma = 0.5$, $\eta = 1$.

Bed load transport is plotted for all formulations in Figure 7.4 (without wave effects) as function of current magnitude, using a particle size of 1 mm.

7.5.8 Bed slope effects and coordinate transforms

The bed slope can have three effects:

- influence of local flow direction
- a different formulation for the critical shear stress (see Section 7.3.2)
- a change in the bed load transport rate and direction.

In COHERENS, the second and third effect can be modeled using the switch `iopt_sed_slope`. The second effect was discussed in Section 7.3. For the third effect, a correction for the transport rate vector magnitude has to be combined with a correction on the transport direction, i.e. the correction has a longitudinal component parallel to the current vector and a transverse component, normal to the current.

The longitudinal $\partial h / \partial c$ and transverse $\partial h / \partial n$ bed slopes are given by

$$\frac{\partial h}{\partial c} = \frac{1}{h_1} \frac{\partial h}{\partial \xi_1} \cos \phi_c + \frac{1}{h_2} \frac{\partial h}{\partial \xi_2} \sin \phi_c \quad (7.69)$$

$$\frac{\partial h}{\partial n} = \frac{1}{h_2} \frac{\partial h}{\partial \xi_2} \cos \phi_c - \frac{1}{h_1} \frac{\partial h}{\partial \xi_1} \sin \phi_c \quad (7.70)$$

where ϕ_c is the angle between the X-axis and the current direction.

The slope effects are modeled using the relation of [Koch & Flokstra \(1981\)](#). They developed the following relation for a correction factor α_s on the bed load transport rate

$$\alpha_s = 1 - \beta_s \frac{\partial h}{\partial c} \quad (7.71)$$

with $\beta_s = 1.3$. They included also the effect of the slope transverse to the flow by the change in direction of δ_s degrees:

$$\tan \delta_s = \beta_s \frac{\partial h}{\partial n} \quad (7.72)$$

Note that these corrections are only applied for bed load equations and not for total load.

The bed load formulae in the previous subsections were derived along the current direction and an additional transverse component in the presence of waves. The components along the coordinate directions are calculated by the following transformations

$$\begin{aligned} \Phi_{b1,n} &= \alpha_s \left(\Phi_{bc,n} \cos(\phi_c + \delta_s) - \Phi_{bn,n} \sin(\phi_c + \delta_s) \right) \\ \Phi_{b2,n} &= \alpha_s \left(\Phi_{bc,n} \sin(\phi_c + \delta_s) + \Phi_{bn,n} \cos(\phi_c + \delta_s) \right) \end{aligned} \quad (7.73)$$

where

- $\alpha_s = 1$, $\beta_s = 0$, $\delta_s = 0$ in the absence of bed slope effects
- $\Phi_{bn,n} = 0$ in the absence of waves.

Implementation

The bed load formulae in this section are selected with the following switches²

`iopt_bedeq*` Type of bedload equation.

- 0: Disabled. Default.
- 1: [Meyer-Peter & Müller \(1948\)](#).
- 2: [Engelund & Fredsøe \(1976\)](#).
- 3: [Van Rijn \(1984b\)](#).
- 4: [Wu *et al.* \(2000\)](#).
- 5: [Soulsby \(1997\)](#). This equation includes explicit wave effects.
- 6: [Van Rijn \(1993\)](#).

- `iopt_slope*` Disables or enables bed slope effects.
- 0: Disabled. Default.
 - 1: Slope effects enabled using the [Koch & Flokstra \(1981\)](#) formulation.

7.6 Total load transport

The total load transport $\mathbf{q}_{t,n}$ is defined as the sum of bed and suspended load. The former can be determined from one of the formulations of Section 7.5, the latter is given by the depth integrated suspended sediment flux

$$\mathbf{q}_{s,n} = \int_{-h}^{\eta} \mathbf{u}_h c_n dz \quad (7.74)$$

where c_n is calculated by solving the transport equations (7.85) or (7.88) for suspended sediment. The total load can alternatively be determined using one of the four formulations below without solving the time-consuming transport equations and is directed along the current direction:

$$\mathbf{q}_{t,n} = q_{t,n} \mathbf{1}_c \quad (7.75)$$

7.6.1 Engelund and Hansen (1967)

The classical formula of [Engelund & Hansen \(1967\)](#) has been modified by [Chollet & Cunge \(1979\)](#) to take into account different transport regimes and to introduce a threshold for beginning of sediment particle motion:

$$q_{t,n} = 0.05 f_n \sqrt{(s-1) g d_{p,n}^3} \frac{\theta_{*n}^{5/2}}{C_{db}^f} \quad (7.76)$$

In the original formulation of [Engelund & Hansen \(1967\)](#), θ_{*n} is set to the Shields parameter θ_n . In the modified version of [Chollet & Cunge \(1979\)](#), θ_{*n} is defined depending on the transport regime:

$$\begin{aligned} \theta_{*n} &= 0 && \text{if } \theta_n < 0.06 && \text{(no transport),} \\ \theta_{*n} &= [2.5(\theta - 0.06)]^{0.5} && \text{if } 0.06 < \theta_n < 0.384 && \text{(dune regime)} \\ \theta_{*n} &= 1.065 \theta^{0.176} && \text{if } 0.348 < \theta_n < 1.08 && \text{(transition regime)} \\ \theta_{*n} &= \theta && \text{if } 1.08 < \theta_n && \text{(sheet flow regime)} \end{aligned} \quad (7.77)$$

The user can choose to use the original version as well as the modified version. The modified version shows deviations of the transport calculated for bed shear below the sheet flow regime, where the original version shows result more in line with other formulae.

7.6.2 Ackers and White (1973)

A direct determination of the total sediment transport rate is provided with the formula from [Ackers & White \(1973\)](#). It gives extra attention to the subdivision of non-cohesive sediment transport for coarser and finer fractions. It was developed using hydraulic considerations and dimensional analysis. The coefficients in the model have been determined by evaluation of nearly 1000 experiments in the laboratory and 250 experiments in the field. Nineteen different transport formulae have been studied in this work.

A parameter of mobility was defined, based on the Shields parameter:

$$F_{gr} = \frac{u_*^{n_w}}{\sqrt{(s-1)gd_{p,n}}} \left[\frac{|\bar{\mathbf{u}}|}{2.46 \ln(10H/d_{p,n})} \right]^{(1-n_w)} \quad (7.78)$$

The parameter of transport is then defined as

$$\begin{aligned} G_{gr} &= C_w \left(\frac{F_{gr}}{\alpha_w} - 1 \right)^{m_w} \quad \text{for} \quad F_{gr} > \alpha_w \\ G_{gr} &= 0 \quad \text{otherwise} \end{aligned} \quad (7.79)$$

The total load is then calculated as follows:

$$q_{t,n} = f_n G_{gr} d_{p,n} \left(\frac{|\bar{\mathbf{u}}|}{u_*} \right)^{n_w} \bar{\mathbf{u}} = f_n G_{gr} d_{p,n} (C_{db}^f)^{-n_w/2} \bar{\mathbf{u}} \quad (7.80)$$

This transport formula was derived by uniting bed load and suspended load transport relations in one function through a transition in the dimensionless d_* parameter. A revised set of values for the parameters n_w , m_w , c_w and C_w which are used in the current implementation, have been derived after experiments in 1990:

$$\begin{aligned} n_w &= 1 - 0.243 \ln \tilde{d}_{*p,n} \\ m_w &= 1.67 + 6.83/\tilde{d}_{*p,n} \\ \ln C_w &= 2.79 \ln \tilde{d}_{*p,n} - 0.426(\ln \tilde{d}_{*p,n})^2 - 7.97 \\ \alpha_w &= 0.14 + 0.23/\tilde{d}_{*p,n}^{1/2} \end{aligned} \quad (7.81)$$

with $\tilde{d}_{*p,n} = \min(60, \max(1, d_{*p,n}))$

7.6.3 Madsen and Grant (1976)

The [Madsen & Grant \(1976\)](#) formula is given by

$$q_{t,n} = 40 f_n w_{s,n} d_{p,n} \theta_n^3 \quad (7.82)$$

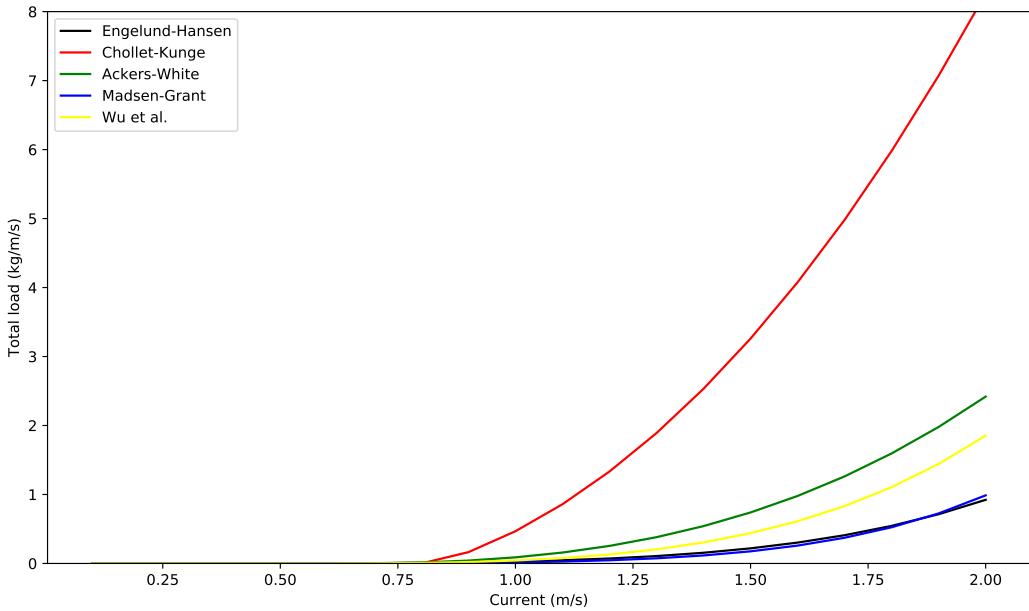


Figure 7.5: Total load transport (kg/m/s) as function of velocity for the five available schemes using a particle size of 400 μm .

7.6.4 Wu et al (2000)

Bed and suspended load are calculated separately so that

$$q_{t,n} = q_{b,n} + q_{s,n} \quad (7.83)$$

where $q_{b,n}$ is given by (7.62) and

$$q_{s,n} = 0.262 \cdot 10^{-4} f_n \sqrt{(s_n - 1) g d_{p,n}^3} \left(\tau_{*n} \frac{|\bar{\mathbf{u}}|}{w_{s,n}} \right)^{1.74} \quad (7.84)$$

In the paper of Wu *et al.* (2000), θ_{cr} is set to 0.03 and the settling velocity is calculated with equation (7.43) of Zhang & Xie (1993).

Total load transport is shown for the 5 available schemes in Figure 7.4 as function of current magnitude, using a particle size of 400 μm .

Implementation

The type of total load formulation is selected with the following switches²

`iopt_toteq*` Method for total load transport.

- 0: Disabled. Default.
- 1: [Engelund & Hansen \(1967\)](#) with $\theta_{*,n} = \theta_n$.
- 2: [Engelund & Hansen \(1967\)](#) using the modified version of [Chollet & Cunge \(1979\)](#) for $\theta_{*,n}$.
- 3: [Ackers & White \(1973\)](#).
- 4: [Madsen & Grant \(1976\)](#).
- 5: [Wu et al. \(2000\)](#). Total load is calculated as the sum of suspended and bed load.

Note that total load is incompatible with bed and suspended load. This means that if `iopt_toteq` is non-zero for a fraction, the switches `iopt_bedeq` and `iopt_suseq` (see below) must be zero. An error message is issued by the program if this is not the case. If the switches for bed and suspended load are non-zero, the total load is calculated by the program as the sum of bed and suspended load without setting the switch `iopt_toteq`.

7.7 Suspended sediment transport

7.7.1 Sediment transport equations

7.7.1.1 Three-dimensional transport

The transport equations for a suspended sediment concentration c_n is given by (see (5.38))

$$\begin{aligned} \frac{1}{h_3} \frac{\partial}{\partial t} (h_3 c_n) + \mathcal{A}_{h1}(u, c_n) + \mathcal{A}_{h2}(v, c_n) + \mathcal{A}_v(\omega - w_s, c_n) \\ = Q_{sed,n} + \mathcal{D}_{sv}(c_n) + \mathcal{D}_{sh1}(c_n) + \mathcal{D}_{sh2}(c_n) \end{aligned} \quad (7.85)$$

where the operators are defined in Section 5.1.1.2, $Q_{sed,n}$ is the rate of “dry” sediment discharge, when the discharge module has been activitated. For details see Section 11.4.2.1 and (11.22). Formulations for the sediment diffusion coefficient λ_T^{sed} are discussed in Section 7.7.3.

Open boundary conditions are obtained by selecting one of the formulations given in Section 5.10.3.1. The most common method is either a zero-gradient condition (default) or the specification of an external profile. The vertical boundary conditions are

$$w_{s,n} = 0, \quad \lambda_T^{sed} c_n = 0 \quad \text{at the surface} \quad (7.86)$$

$$w_{s,n} c_n = D_n, \quad -\lambda_T^{sed} c_n = E_n \quad \text{at the bottom} \quad (7.87)$$

where D_n and E_n are the deposition and erosion rates (in m/s) for the sediment fraction c_n . They are further discussed in Section 7.7.2.

7.7.1.2 Two-dimensional sediment transport

As explained in Section 5.2.2, the advection-diffusion equations for sediments can also be solved in depth-averaged form. In that case, erosion E_n and deposition D_n , used to determine the bottom boundary condition in the 3-D case (see below), are modeled as source and sink terms in the depth averaged sediment concentration equation

$$\frac{\partial \bar{c}_n}{\partial t} + \frac{1}{h_1 h_2} \left(\frac{\partial}{\partial \xi_1} h_2 \bar{c}_n \bar{u} + \frac{\partial}{\partial \xi_2} h_1 \bar{c}_n \bar{v} \right) = \frac{1}{h_1 h_2} \left(\frac{\partial}{\partial \xi_1} \frac{\lambda_H}{h_1} \frac{h_2}{\partial \xi_1} \partial \bar{c}_n + \frac{\partial}{\partial \xi_2} \frac{\lambda_H}{h_2} \frac{h_1}{\partial \xi_2} \partial \bar{c}_n \right) + \frac{E_n - D_n}{H} \quad (7.88)$$

For details see Section 5.2.2.

7.7.2 Erosion and deposition

Erosion and deposition have a double influence. On the one hand, erosion and deposition provide the bottom boundary conditions for the advection-diffusion equation of sediment transport. On the other hand, they provide important sink and source terms for the bed elevation equation in the morphological module (see Section 8.2). In this section, the modeling of the erosion and deposition terms is described. This is first done for three-dimensional situations, where the erosion and deposition terms are introduced through the near-bed boundary conditions. These terms are described separately for cohesive sediment (mud) and non-cohesive sediment (sand), because they are calculated in a different way. This is followed by a description of erosion and deposition in two-dimensional situations, where they are represented as sink and source terms.

7.7.2.1 Erosion and deposition of cohesive sediment in 3-D

Erosion of cohesive sediments is calculated with the equation based on the data of [Partheniades \(1965\)](#)

$$E_n = f_n \frac{M_p}{\rho_{s,n}} \tau_{*n}^{n_p} \quad (7.89)$$

Deposition is given by

$$D_n = w_{s,n}(z_b) c_n(z_b) \quad (7.90)$$

Here M_p is an erosion rate parameter ($\text{kg}/\text{m}^2/\text{s}$), τ_* , the excess shear stress defined by (7.60), n_p an empirical parameter (normally equal to unity) and z_b

is taken at the bottom grid cell. The parameter M_p needs to be determined, which may be difficult in practice, because it depends on the material, its consolidation as well as biological and chemical characteristics of the bed, and can therefore vary in space and time and f_n is the amount of this fraction in the bed. In the current version of COHERENS, M_p can only be defined as a constant. The critical shear stress is determined using one of the methods, described in Section 7.3.

The implementation of the three-dimensional erosion of cohesive sediment differs from sand in that no reference concentration is calculated, and no reference height has to be specified (see below). Just as for sand, wave effects are only included through the changes in the bed shear stress.

7.7.2.2 Erosion and deposition of sand in 3-D

The following equation, taken at a reference height a_n above the bed, is used as the boundary condition for erosion near the bed for the suspended sediment transport equation

$$E_n - D_n = -\lambda_T^\psi(a_n) \frac{\partial c_n}{\partial z}(a_n) - w_{s,n}(a_n)c_n(a_n) \quad (7.91)$$

The deposition flux is given as

$$D_n = w_{s,n}(a_n)c_n(a_n) \simeq w_{s,n}(z_b)c_n(z_b) \quad (7.92)$$

Using this expression, the bottom boundary condition for erosion reduces to the following Neumann boundary condition:

$$-\lambda_T^{sed}(a_n) \frac{\partial c_n}{\partial z}(a_n) = E_n \quad (7.93)$$

[Garcia & Parker \(1991\)](#) showed that the erosion flux can be expressed as a non-dimensional flux $E_{*n} \equiv E_n/w_{s,n}$ and that in situations not too far from equilibrium, this non-dimensional entrainment is equal to the near bed reference concentration (i.e. $E_{*n} = f_n c_{a,n}$). Therefore one can write:

$$E_n = f_n w_{s,n}(a_n)c_{a,n} \quad (7.94)$$

Here $c_{a,n}$ is the near-bed reference (equilibrium) sediment concentration at a reference height a_n for sediment size class n with fractional amount f_n at the bed.

The near bed reference concentration for sand is calculated either by the method of [Smith & McLean \(1977\)](#) or by the one in [Van Rijn \(1984a\)](#).

The near-bed boundary condition of [Smith & McLean \(1977\)](#) is given for each fraction n by:

$$c_{a,n} = 0.0024c_{max} \frac{\tau_{*n}}{1 + 0.0024\tau_{*n}}$$

$$a_n = k_s + 26.3 \max((\theta_n - \theta_{cr,n}), 0.0) d_{p,n}$$
(7.95)

Here, c_{max} is the maximum possible concentration, i.e. the concentration of a bed packed with sediment, τ_{*n} the normalised excess shear stress defined by [\(7.60\)](#), θ_n the Shield parameter [\(7.10\)](#) and k_s the skin roughness height.

The near bed boundary condition of [Van Rijn \(1984a\)](#) is given by:

$$c_{a,n} = 0.015 \frac{d_{p,n} \tau_{*n}^{1.5}}{a_n d_{*p,n}^{0.3}}$$
(7.96)

with $d_{*p,n}$ defined by [\(7.11\)](#). The reference level a_n (either half the size of the dunes or the roughness length scale k_s) is limited between $0.01H$ and $0.1H$.

The influence of waves on the erosion of sediment is included in this equation only by the change in the bed shear stress due to wave-current interaction.

A more practical formulation for the erosion rate is implemented in **COHERENS** whereby [\(7.94\)](#) is replaced by

$$E_n = f_n w_{s,n}(z_b) c_{eq}(z_b)$$
(7.97)

where z_b is the distance of the lowest C-grid cell to the sea bed and c_{eq} the equilibrium concentration obtained from the Rouse profile. This profile is determined from the transport equation assuming no time dependence, no horizontal gradients, an eddy diffusion coefficient of the form $\kappa u_* \beta_n(z + h)$, an equilibrium balance between erosion and deposition and takes the value of the reference concentration $c_{a,n}$ at the reference height a_n

$$c_{eq,n}(z) = c_{a,n} \left[\frac{\zeta - z}{z + h} \frac{a_n}{H - a_n} \right]^{R_n} \quad \text{for } -h + a_n \leq z \leq \zeta$$
(7.98)

where $c_{eq,n}$ must be lower than the user-defined value c_{max} , H the total water depth, $R_n = w_{s,n}/(\beta_n \kappa u_*)$ the Rouse number, $w_{s,n}$ the particle fall velocity at the sea bed, u_* the skin shear velocity, β_n the inverse of the Prandtl-Schmidt number (see [Section 7.7.3](#)) and κ the von Karman constant. The exponent expresses the relative importance of the suspended load, where it is equal to about 5 for bed load only and decreases to unity for suspended load up to the surface ([Van Rijn, 1993](#)).

7.7.2.3 Erosion-deposition of cohesive sediment in 2-D

For cohesive sediments, the erosion and deposition rates in 2-D are obtained in similar way as for the 3-D case. The erosion rate is given by (7.89), whereas deposition is now given by

$$D_n = w_{s,n}(z_b)\bar{c}_n \quad (7.99)$$

7.7.2.4 Erosion-deposition of sand in 2-D

In two-dimensional calculations, the erosion and deposition of sand are calculated from the difference between the depth averaged concentrations and the depth averaged equilibrium concentration. A time scale $T_{e,n}$ is used in which the sediment concentration adapts to the equilibrium concentration. This represents the fact that deposition depends on the near-bed sediment concentration, rather than the depth averaged value and thus that the sediment concentration profile needs some time to adapt to the equilibrium profile (e.g. [Galappatti & Vreugdenhil, 1985](#)). The equations that are used for the erosion deposition term are

$$E_n - D_n = \frac{H}{T_{e,n}}(\bar{c}_{e,n} - \bar{c}_n) \quad (7.100)$$

The equilibrium concentration $\bar{c}_{e,n}$ can, in the 2-D case, be calculated in different ways. The default option is to calculate \bar{c}_{eq} by taking the depth average of the Rouse profile (7.98) numerically (see Section 7.7.4.7 for the numerical method).

Another way to calculate the depth averaged equilibrium concentration is by using one of the formulae for total load (except the [Wu et al. \(2000\)](#) scheme). The depth averaged equilibrium concentration is then given by

$$\bar{c}_{e,n} = \frac{q_{t,n}}{|\mathbf{U}|} \quad (7.101)$$

Here $q_{t,n}$ is the total load and $|\mathbf{U}|$ the magnitude of the depth-integrated current (in m^2/s).

The adaptation time scale $T_{e,n}$ is given by:

$$T_{e,n} = \frac{H}{w_{s,n}}T_{*n} \quad (7.102)$$

Here, T_{*n} is a non-dimensional correction factor, which depends on the shape of the sediment concentration profile, parameterized by the modified Rouse

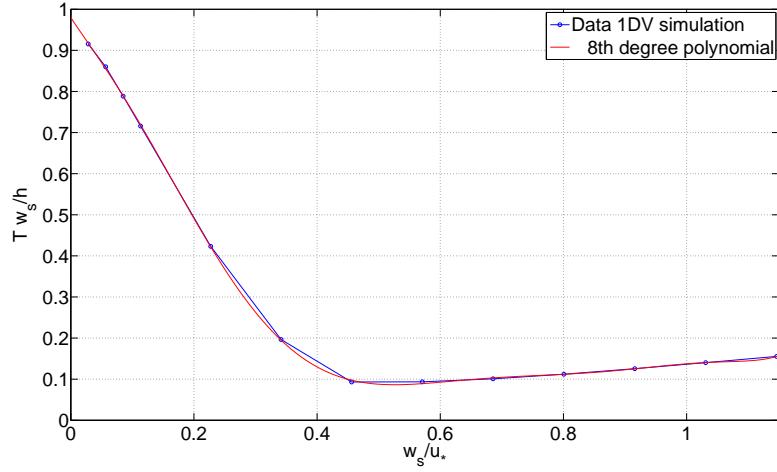


Figure 7.6: Numerically determined time scale and fitted polynomial

number \tilde{R} defined below. This relation was obtained by performing simulations with COHERENS in water column (1DV) mode and determining the adaptation time scale from the data (see Figure 7.6). Through this data an eight order polynomial was fitted:

$$T_{*n} = p_1 \tilde{R}_n^8 + p_2 \tilde{R}_n^7 + p_3 \tilde{R}_n^6 + p_4 \tilde{R}_n^5 + p_5 \tilde{R}_n^4 + p_6 \tilde{R}_n^3 + p_7 \tilde{R}_n^2 + p_8 \tilde{R}_n + p_9 \quad (7.103)$$

with:

$$\tilde{R}_n = \min(w_{sb,n}/u_*, 1.14) \quad (7.104)$$

and

$$\begin{array}{lll} p_1 = 67.342 & p_2 = -321.0 & p_3 = 614.14 \\ p_4 = -592.75 & p_5 = 292.15 & p_6 = -62.141 \\ p_7 = 3.667 & p_8 = -2.2571 & p_9 = 0.97978 \end{array} \quad (7.105)$$

and $w_{sb,n}$ the settling velocity at the bottom.

7.7.3 Sediment diffusivity

It has been found experimentally that the diffusivity of sediment is not always equal to the eddy diffusivity for T and S . For practical purposes, the sediment diffusivity $\lambda_{T,n}^{sed}$ is assumed to be proportional to the eddy viscosity with a proportionality factor depending on size class

$$\lambda_{T,n}^{sed} = \beta_n \nu_T \quad (7.106)$$

Here, β_n is the inverse of the Prandtl-Schmidt number σ_T and ν_T the eddy viscosity defined in Section 5.3. Different relations have been proposed for the β_n factor. In COHERENS, β_n can be either a constant or calculated with the equation of Van Rijn (1984b). Based on the work of (Coleman, 1970), he found that it can be taken as a function of the settling velocity

$$\beta_n = 1 + 2 \left(\frac{w_{s,n}}{u_*} \right)^2 \quad (7.107)$$

For practical applications, β_n is limited to values between 1 and 1.5.

Implementation

The following switches are used for the transport of suspended sediment²:

iopt_bbc_eq* Selects the formulation for the equilibrium sediment concentration.

- 1: Rouse profile. Default.
- 2: Using q_t/U determined with the equation of Engelund & Hansen (1967) and $\theta_{*,n} = \theta_n$.
- 3: Using q_t/U determined with the equation of Engelund & Hansen (1967) and using the modified version of Chollet & Cunge (1979) for $\theta_{*,n}$.
- 4: Using q_t/U determined with the equation of Ackers & White (1973).
- 5: Using q_t/U determined using (7.82) from Madsen & Grant (1976).

iopt_bbc_ref* Selects the formulation for the reference concentration $c_{a,n}$.

- 1: Smith & McLean (1977). Default.
- 2: Van Rijn (1984a).

iopt_suseq* Disables/enables suspended transport.

- 0: Disabled. Default.
- 1: Enabled.

iopt_sed_beta The type of equation used for β_n , the ratio between the eddy viscosity and eddy diffusivity.

- 1: $\beta_n = 1$.

- 2: User defined value (`beta_cst`).
- 3: [Van Rijn \(1984b\)](#).

`iopt_sed_nodim` The number of dimensions used in the sediment formulation.

- 2: Depth (2-D) averaged transport³
- 3: 3-D sediment transport. Default.

7.7.4 Numerical methods

The numerical methods are generally the same as the ones for a scalar quantity, as described in Section 12.4. This section describes a few numerical features, specific for the sediment transport module.

7.7.4.1 Deposition flux

The deposition flux is represented in the model as an vertical advective flux applied at the sea bed. Different types of discretisations are available

- The deposition flux is set to zero which means that all sediment remains inside the computation domain. This is useful only for simulating some idealised tests and should not be used for general applications.
- A first order discretisation using an upwind scheme

$$D_{n;ij} = w_{s,n;ij1} c_{n;ij1} \quad (7.108)$$

- A second order discretisation of the deposition flux using linear extrapolation to obtain an estimation of the sediment concentration at the lowest W-node. It is given by

$$D_{n;ij} = w_{s,n;ij1} \left[\left(1 + \frac{h_{3;ij1}^c}{2h_{3;ij2}^w} \right) c_{ij1} - \frac{h_{3;ij1}^c}{2h_{3;ij2}^w} c_{ij2} \right] \quad (7.109)$$

Note that, whereas the numerical discretisation of vertical advection is treated semi-implicitly in the water column (see Section 12.4.1), the bottom (deposition) flux is taken explicitly at the old time step for reasons explained in Section 8.4.3.

³Note that `iop_sed_nodim` is always set to 2 if `iop_grid_nodim` = 2.

7.7.4.2 Settling velocity

To prevent numerical instabilities in shallow waters (e.g. near tidal flats) with small water depths and vertical grid spacings, the settling velocity can optionally be limited by

$$w_{s,n} \leq w_{slim} \frac{\Delta z}{\Delta t} \quad (7.110)$$

where w_{slim} can be considered as a maximum value for the CFL number for vertical advection by settling (see equation (12.3)). Although the semi-implicit vertical advection scheme should, in principle, prevent this instability, it may still arise when the vertical displacement of sediment particle by settling becomes comparable or larger than the water depth.

7.7.4.3 Bed and total load

The bed and total load transports are determined at all wet velocity nodes, including nodes located at open boundaries. Both transports depend on the value of the bottom stress, which may be assumed to be less accurate at open boundaries. Significant errors can therefore be produced in the divergence of the transport normal to the open boundary with severe consequences for the sea bed morphology (see equation 8.2). An option has therefore been implemented to apply a zero-gradient condition for bed and total load transports at open sea boundaries.

An upwind-type scheme is applied for bed loads, meaning that the sediment fraction f_n in the formulae is taken at the C-node, located on the downstream side at the velocity node. This also prevents transport outside a cell containing no bed sediments.

7.7.4.4 Median size diameter

The median size diameter is defined by

$$d_{50} = d_{p,n} \quad \text{such that} \quad c_f(n-1) \leq 0.5 < c_f(n) \quad (7.111)$$

where the particle diameters are first sorted in ascending order and the cumulative fraction distribution c_f is defined by

$$\begin{aligned} c_f(n) &= 0 && \text{if } n = 1 \\ c_f(n) &= \sum_{m=1}^{n-1} f_m && \text{if } 1 < n \leq N_f \end{aligned} \quad (7.112)$$

7.7.4.5 Bartnicki filter

Negative concentrations may arise by application of the bottom boundary conditions. In order to prevent negative sediment concentrations, a [Bartnicki \(1989\)](#) filter can be used with the switch `iopt_sed_filter`. This filter eliminates negative sediment concentrations, while conserving the total sediment volume. It is based on the assumption that the amount of negative concentration is much smaller than the amount of positive concentrations. The Bartnicki filter performs, at each time step, the following steps iteratively until no negative concentrations exist in the model.

- The total volume of negative concentrations is determined: $V_- = \sum_{ijk} c_{ijk} (c_{ijk} < 0) \Delta V_{ijk}$ where $\Delta V_{ijk} = h_{1;ij} h_{2;ij} h_{3;ijk}$ is the volume of a grid cell.
- The total volume of the cells with a positive concentration (larger than 0) is determined: $V_+ = \sum_{ijk} (c_{ijk} > 0) \Delta V_{ijk}$.
- The volume fraction of negative concentrations is determined: $M = V_- / V_+$.
- The concentration in cells with a negative concentration is set to zero.
- M is distributed over cells with a positive concentration.

7.7.4.6 Bed slope factors

Using the notations of Section [7.5.8](#), one has

$$\begin{aligned} \cos(\phi_c + \delta_s) &= \cos \phi_c \cos \delta_s - \sin \phi_c \sin \delta_s \\ \sin(\phi_c + \delta_s) &= \sin \phi_c \cos \delta_s + \cos \phi_c \sin \delta_s \end{aligned} \quad (7.113)$$

Letting

$$\beta_* = \beta_s \frac{\partial H}{\partial n} = \tan \delta_s \quad (7.114)$$

one has

$$\begin{aligned} \cos \delta_s &= \frac{1}{\sqrt{1 + \beta_*^2}} \\ \sin \delta_s &= \frac{\beta_*}{\sqrt{1 + \beta_*^2}} \end{aligned} \quad (7.115)$$

so that

$$\cos(\phi_c + \delta_s) = \frac{\cos \phi_c - \beta_* \sin \phi_c}{\sqrt{1 + \beta_*^2}}$$

$$\sin(\phi_c + \delta_s) = \frac{\sin \phi_c + \beta_* \cos \phi_c}{\sqrt{1 + \beta_*^2}} \quad (7.116)$$

7.7.4.7 Gaussian-Legendre quadrature

Numerical integration is needed to determine the depth averaged equilibrium concentration by taking the vertical average of the Rouse profile (7.98). The integral is calculated by applying Gaussian quadrature. The integral of a function $f(x)$ between a and b is calculated as:

$$\int_a^b f(x) dx \simeq \frac{b-a}{2} \sum_{i=1}^n w_i f \left(\frac{b-a}{2} x_i + \frac{a+b}{2} \right) \quad (7.117)$$

Here, n is the number of points in the integration, x_i are the locations of the nodes where the function is evaluated and w_i are the weighting factors. The locations x_i are the roots of the n -th order Legendre polynomial P_n and w_i are determined by

$$w_i = \frac{2}{(1-x_i)^2} \left(\frac{dP_n(x_i)}{dx} \right)^{-2} \quad (7.118)$$

The equilibrium concentration \bar{c}_e is then given by

$$\bar{c}_e \simeq \frac{c_a}{H} \int_a^H \left[\frac{H-z'}{z'} \frac{a}{H-a} \right]^R dz' \quad (7.119)$$

where $z' = z + h$ is the distance to the sea bed. Letting $\phi = \ln(z/H)$, $a_* = a/H$ and applying (7.117) one obtains

$$\begin{aligned} \bar{c}_e &= c_a \left(\frac{a_*}{1-a_*} \right)^R \int_{\ln a_*}^0 e^\phi (e^{-\phi} - 1)^R d\phi \\ &\simeq -\frac{c_a}{2} \left(\frac{a_*}{1-a_*} \right)^R \ln a_* \sum_{i=1}^n w_i e^{\phi_i} (e^{-\phi_i} - 1)^R \end{aligned} \quad (7.120)$$

with $\phi_i = \frac{1}{2}(1-x_i) \ln a_*$.

Implementation

Numerical methods, described in this section, are selected with the following switches²:

iopt_sed_filter Disables/enables the application of the Bartrniki filter to prevent the occurrence of negative concentrations.

0: Disabled. Default.

1: Enabled.

`iopt_sed.obc_flux` Disables/enables a zero gradient condition for bed and total load transports at open boundaries.

0: Disabled. Default.

1: Enabled.

`iopt_sed.vadv` Selects the type of vertical advection scheme for settling in case of 1-D simulations.

0: Vertical settling disabled.

1: Upwind scheme.

2: Central scheme.

3: TVD scheme. Default.

`iopt_ws_lim*` Disables/enables the limitation of the settling velocity for shallow waters.

0: Disabled. Default.

1: Enabled.

7.8 Flocculation model

In the multi-fraction sediment transport model, presented in the previous sections, the sediment particles are distributed among different size classes, each having its own diameter, mass, fall velocity, critical shear stress. The size distributions are considered as independent of each other. If the flocculation model is used instead, larger size particles (flocs) are formed from smaller size particles (flocculi) by aggregation. The opposite effect is the breakage of larger size particles into smaller ones. Diameter, mass and fall velocity are now defined as function of the number of flocculi within flocs. The flocculation model, implemented in COHERENS, is the two-class bimodal population balance model, described in [Lee *et al.* \(2011, 2014\)](#).

The following remarks should be given concerning the use of the flocculation module with the other sediment modules:

- The flocculation and the multi-fraction model cannot be activated both within the same simulation.

- The morphology can only be activated in connection with the multi-fraction model. A morphological module coupled with the flocculation model, is not available in the current version of COHERENS.
- The flocculation model can only be used in 1-D water column or 3-D mode, but not in depth-averaged (2-D) mode.

7.8.1 Flocculation transport model

The model contains three state variables: the number concentrations⁴ of flocculi (N_P), flocs (N_F) and flocculi bound in flocs (N_T). They are calculated by solving the transport equations

$$\frac{1}{h_3} \frac{\partial}{\partial t} (h_3 N_P) + \mathcal{A}_{h1}(u, N_P) + \mathcal{A}_{h2}(v, N_P) + \mathcal{A}_v(\omega - w_{sP}, N_P) = \mathcal{D}_{sv}(N_P) + \mathcal{D}_{sh1}(N_P) + \mathcal{D}_{sh2}(N_P) + A_P + B_P \quad (7.121a)$$

$$\frac{1}{h_3} \frac{\partial}{\partial t} (h_3 N_F) + \mathcal{A}_{h1}(u, N_F) + \mathcal{A}_{h2}(v, N_F) + \mathcal{A}_v(\omega - w_{sF}, N_F) = \mathcal{D}_{sv}(N_F) + \mathcal{D}_{sh1}(N_F) + \mathcal{D}_{sh2}(N_F) + A_F + B_F \quad (7.121b)$$

$$\frac{1}{h_3} \frac{\partial}{\partial t} (h_3 N_T) + \mathcal{A}_{h1}(u, N_T) + \mathcal{A}_{h2}(v, N_T) + \mathcal{A}_v(\omega - w_{sF}, N_T) = \mathcal{D}_{sv}(N_T) + \mathcal{D}_{sh1}(N_T) + \mathcal{D}_{sh2}(N_T) + A_T + B_T \quad (7.121c)$$

where the advective and diffusive operators are defined in Section 5.1.1.2, w_{sP} and w_{sF} are the settling velocities for flocculi and flocs discussed below. The value of N_C is limited by $N_{cmin} \leq N_C \leq N_{max}$ to prevent numerical instabilities. Contrary to the multi-fraction sediment model, the vertical diffusion coefficient is the same as the one used for T and S .

The last two terms on the right hand side of (7.121a) and (7.121b) are source/sink terms due to respectively aggegration and breakage and are given by

$$A_P + B_P = -\frac{1}{2} \alpha \beta_{PP} \frac{N_P^2 N_C}{N_C - 1} - \alpha \beta_{PF} N_P N_F + f a_F N_C N_F \quad (7.122a)$$

$$A_F + B_F = \frac{1}{2} \alpha \beta_{PP} \frac{N_P^2}{N_C - 1} - \frac{1}{2} \alpha \beta_{FF} N_F^2 + a_F N_F \quad (7.122b)$$

$$A_T + B_T = -(A_P + B_P) \quad (7.122c)$$

where f is the fraction of flocculi generated by floc breakage, α the collision efficiency factor, β_{ij} the collision frequency function between size classes i and

⁴A number concentration is defined as the number of particles per unit volume.

j (where the indices i and j are equal to either P for flocculi or F for flocs), a_F the breakage kinetic factor and $N_C = N_T/N_F$. The value of N_C is limited by $N_{cmin} \leq N_C \leq N_{max}$ to prevent numerical instabilities. The collision function can be written as the sum of three collision factors, representing respectively the collision frequency due to Brownian motion, fluid shear and differential settling

$$\beta_{ij} = \beta_{BR,ij} + \beta_{SH,ij} + \beta_{DS,ij} \quad (7.123)$$

which are defined by

$$\beta_{BR,ij} = \frac{2k_B T_k}{3\rho\nu} \left(\frac{1}{d_i} + \frac{1}{d_j} \right) (d_i + d_j) \quad (7.124a)$$

$$\beta_{SH,ij} = \frac{1}{6} (d_i + d_j)^3 G \quad (7.124b)$$

$$\beta_{DS,ij} = 2\pi d_i^2 |w_{si} - w_{sj}| \quad (7.124c)$$

where k_B is Boltzmann's constant, T_k the water temperature in degrees K, d_P the diameter of the flocculi, d_F the floc diameter, defined below, and $G = \sqrt{\varepsilon/\nu}$ the turbulent shear. The factor a_i represents the rate of braking and is given by

$$a_i = E_b G \left(\frac{d_i - d_p}{d_p} \right)^p \left(\frac{\rho\nu G}{F_y/d_i^2} \right)^q \quad (7.125)$$

where E_b is the breakage efficiency factor and F_y the yield strength of flocs in Pa.

Open boundary conditions for the three transport variables are obtained by selecting one of the formulations given in Section 5.10.3.1. The most common method is either a zero-gradient condition (default) or the specification of an external profile.

The vertical boundary conditions at the surface are the same as for the multi-fraction model (zero advective and diffusive flux). At the bottom, deposition is given by (7.90). Erosion rate for flocculi is determined by the equilibrium formulation (7.97). Erosion is set to zero for flocs since it is assumed that the binding of the microflocs inside flocs is broken once the flocs reach the sea bed.

7.8.2 Floc properties

The floc diameter and density are determined from fractal theory taking account of floc packing and shaping

$$d_F = N_C^{1/n_F} d_P \quad (7.126)$$

$$\rho_F = \rho_w + (\rho_P - \rho_w) \left(\frac{d_P}{d_F} \right)^{3-n_F} \quad (7.127)$$

where ρ_P is the microfloc density and n_F the fractal dimension of flocs (between 1.7 and 2.3).

Two formulations are available for the settling velocity.

1. The first is the Stokes formula. For flocculi this gives

$$w_{sP} = \frac{(s_P - 1) g d_P^2}{18 \nu} \quad (7.128)$$

with $s_P = \rho_P / \rho_w$. The settling velocity for flocs is determined from the modified Stokes formula ([Winterwerp & van Kesteren, 2004](#))

$$w_{sF} = \frac{(s_P - 1) g}{18 \nu} \frac{d_P^{3-n_F} d_F^{n_F-1}}{1 + 0.15 Re_F^{0.687}} \quad (7.129)$$

where $Re_F = w_{sF} d_F / \nu$ is the particle Reynolds number for flocs. Note that, since w_{sF} appears both on the left and right side of (7.129), the equation should, in principle be solved by iteration. In COHERENS, this is avoided by taking the value of Re_F from the previous time step.

2. The second is the formula of [Camenen \(2007\)](#) now written in the form

$$w = \frac{\nu}{d} \left[\sqrt{\frac{1}{4} \left(\frac{A}{B} \right)^{2/m} + \left(\frac{4}{3} \frac{d_*^3}{B} \right)^{1/m}} - \frac{1}{2} \left(\frac{A}{B} \right)^{1/m} \right]^m \quad (7.130)$$

where

$$\begin{aligned} w &= w_P, d = d_P, d_* = d_P \left[(s_P - 1) \frac{g}{\nu^2} \right]^{1/3} \\ A &= 24.6, B = 0.96, m = 1.53 \end{aligned} \quad (7.131)$$

for flocculi and

$$\begin{aligned} w &= w_F, d = d_F, d_* = d_F \left[(s_F - 1) \frac{g}{\nu^2} \right]^{1/3}, s_F = \rho_F / \rho_w \\ A &= 26.8, B = 2.11, m = 1.19 \end{aligned} \quad (7.132)$$

for flocs.

Effects of hindered settling can optionally be taken into account by multiplying (7.128)–(7.129) with a factor f_{hs} for hindered settling, defined by (7.45).

In the current implementation of the flocculation model, only the [Richardson & Zaki \(1954\)](#) formulation is allowed.

Once the number densities are calculated from the transport equations [\(7.121a\)](#)–[\(7.121c\)](#) the volume, total and mass concentrations become:

$$c_P = \frac{\pi}{6} N_P d_P^3, \quad c_F = \frac{\pi}{6} d_P^3 N_F N_c^{3/n_F} \quad (7.133a)$$

$$c_t = c_P + c_F \quad (7.133b)$$

$$c_{mP} = \rho_P c_P \quad c_{mF} = \rho_F c_F \quad (7.133c)$$

Switches

The following switches are available:

<code>iopt_floc_bbc_ref</code>	Selects the formulation for the reference concentration $c_{a,n}$.
	1: Smith & McLean (1977) . Default.
	2: Van Rijn (1984a) .
<code>iopt_floc_bstres_cr</code>	Selects the formulation for the critical bottom shear stress.
	1: User-defined value for each fraction. Default.
	2: Brownlie (1981) as given by (7.31) .
	3: Soulsby & Whitehouse (1997) as given by (7.32) .
	4: Constant value for the critical Shield parameter from Wu et al. (2000) .
<code>iopt_floc_slope</code>	Selects the type of slope factor for the critical shear stress.
	0: Disabled. Default.
	1: Enabled using (7.39) .
<code>iopt_floc_ws</code>	Selects the formulation for the settling velocity.
	0: Disabled.
	1: Stokes formula.
	2: Camenen (2007) .
<code>iopt_floc_ws_hindset</code>	Formulation for hindered settling.
	0: Hindered settling disabled.
	1: Richardson & Zaki (1954) equation (7.45) .

<code>iopf_floc_ws_lim</code>	Disables/enables the limitation of the settling velocity for shallow waters. 0: Disabled. Default. 1: Enabled.
<code>iopf_kinvisc</code>	Selects type of kinematic viscosity. 0: User-selected uniform value. Default. 1: From the ITTC (1978) equation.
<code>iopf_sed</code>	The flocculation model is activated by setting this switch to 2.
<code>iopf_sed_dens_grad</code>	Disables/enables the inclusion of sediment stratification in the formulations for the buoyancy frequency and baroclinic pressure gradient. 0: Disabled. Default. 1: Enabled.
<code>iopf_sed_vadv</code>	Selects the type of vertical advection scheme for settling in case of 1-D simulations. 0: Vertical settling disabled. 1: Upwind scheme. 2: Central scheme. 3: TVD scheme. Default.

Chapter 8

Morphological model

8.1 Introduction

The sediment transport module, described in Chapter 7 and the morphological module can be considered as complementary since the first one deals with sediment transport in the water column and bed or total transport, whereas the second one deals with sediment processes in the bottom sediment layer. The two modules are connected since sediment matter eroded from or deposited at the sea bed are added or retrieved from the water column and bed load transport adds or removes sediment, depending on the transport direction, in the upper part of the bottom layer.

In the next section, the bed elevation equation, which governs the evolution of the bed under the influence of erosion/deposition and sediment transport, is introduced. This section also addresses the numerical schemes used to solve this equation. A next section covers the implemented schemes for morphological acceleration, a technique often used to calculate long-term morphodynamics without having to sacrifice a lot of computational time. In the following section, the notions of fixed and active layers are introduced and a scheme is presented which conserves sediment mass in both the water column and bed layer. Then, the mechanics and concepts behind vertical sorting - i.e. the modeling of the vertical sediment structure within the bed - are explained. A final section offers a description of the implemented avalanching scheme, with which geomechanical failure of bed material on critical slopes can be described.

8.2 Bed elevation equation

8.2.1 General aspects

The bed elevation equation, also called the “Exner” equation, forms the base of the morphological module. The equation expresses the conservation of sediment mass contained in the water column and bed layer and takes the form of a continuity equation for each fraction

$$(1 - p) \frac{\partial \eta_n}{\partial t} + \nabla \cdot \mathbf{q}_{tr,n} = D_n - E_n \quad (8.1)$$

where η is the elevation of the sea bed, p the porosity of the bed (i.e. the fractional amount of water) defined by the user, $\mathbf{q}_{tr,n}$ represents either the bed $\mathbf{q}_{b,n}$ or the total load $\mathbf{q}_{t,n}$ transport and D_n, E_n are respectively the deposition and erosion rates. In the model, the equation is first solved for each fraction separately and then summed over all fractions giving $\eta = \sum_{n=1}^{n_f} \eta_n$.

Once the bed level is updated, the mean water depth needs to be adapted as well since the amount of sediments added to or retrieved from the sea bed must be compensated by decrease, respectively increase of the mean water depth, i.e.

$$\frac{\partial}{\partial t} (h + \eta) = 0 \quad (8.2)$$

In choosing a numerical scheme to solve the bed elevation equation, some specific problems with respect to morphology need to be taken into consideration:

- Sediment transport is highly non-linear. Therefore, the bed level equation coupled with the hydrodynamic equations and a sediment transport model leads to a non-linear hyperbolic model. Such models can give rise to the occurrence of shock waves. The numerical scheme must be able to deal with these shock waves, without becoming unstable or generating spurious wiggles.
- Upwind schemes may lead to difficulties, because the propagation velocity of the bed disturbances might be difficult to determine. In particular, it is not necessarily true that the bed level disturbances propagate in the direction of the flow (Talstra, 2003). One could use the Jacobian of the hyperbolic system as an alternative. However, except in simple situations, there is no closed expression for the Jacobian, such that it needs to be estimated numerically, which may give some complications.

- The simultaneous occurrence of bed load and suspended load might lead to instabilities if these transports are not well balanced (Talstra, 2003).
- Since the morphological time scale is large compared to the hydrodynamic time scale, it might be advantageous to use time steps that are multiples of the hydrodynamic time step (i.e. morphological mode splitting). Care must be taken, when doing so, in particular in combination with morphological acceleration (Section 8.3), and it might be advantageous to use numerical schemes that have a higher order accuracy in time (see below).

These conditions led to the implementation of the schemes described below.

8.2.2 Spatial integration

The bed level elevation equation has the form of a hyperbolic conservation law. The general form of the hyperbolic conservation law is, in the absence of deposition and erosion, given by

$$\frac{\partial \eta}{\partial t} + \nabla \cdot \mathbf{F}(\eta) = 0 \quad (8.3)$$

The non-linearity can easily be seen in the function $\mathbf{F}(\eta)$, which represents either the bed or total load. Additionally, this equation does not have any dissipation.

A frequently used approach in morphological models is to use an artificial diffusion flux \mathbf{q}_h for solving the bed level equation. This should normally only be used close to discontinuities to keep the calculation stable, and be very small otherwise. Equation (8.1) (omitting the fraction index for simplicity) then becomes

$$(1 - p) \frac{\partial \eta}{\partial t} + \frac{1}{h_1 h_2} \left[\left(\frac{\partial}{\partial \xi_1} (h_2 q_1) + \frac{\partial}{\partial \xi_2} (h_1 q_2) \right) \right] = D - E \quad (8.4)$$

where $\mathbf{q} = \mathbf{q}_{tr} + \mathbf{q}_h$.

The discretisation of the diffusive flux is given by Jameson *et al.* (1981), who used a second order term (high dissipation) near strong gradients and a fourth order (low dissipation) term in smooth regions. The scheme is described below, for simplicity, for fluxes in the X-direction with the j - and fraction index omitted

$$q_{h,i} = c_{b,i} \left[\epsilon_i^{(2)} (\eta_i - \eta_{i-1}) + \epsilon_i^{(4)} (\eta_{i+1} - 3\eta_i + 3\eta_{i-1} - \eta_{i-2}) \right]$$

$$\begin{aligned}
c_{b;i} &= \frac{h_{1;i}^u}{h_{1;i-1}^c + h_{1;i}^c} \frac{q_{tr;i+1} - q_{tr;i-1}}{\eta_{i-1} - \eta_i} \\
\epsilon_i^{(2)} &= \min(0.5, k_2 \nu_i^u) \\
\epsilon_i^{(4)} &= \max(0, k_4 - \alpha \nu_i^u) \\
\nu_i^u &= \max(\nu_{i-1}^c, \nu_i^c) \\
\nu_i^c &= \left| \frac{\eta_{i-1} - 2\eta_i + \eta_{i+1}}{\eta_{i-1} + 2\eta_i + \eta_{i+1}} \right|
\end{aligned} \tag{8.5}$$

where $c_{b;i}$ is the propagation speed of the bed level disturbances, ν_i acts as a gradient detector, which determines which kind of diffusion (second order or fourth order) is used. Further $k_2 = 8$ (but can be tuned to give better shock absorption), $k_4 = 1/32$ and $\alpha = 2$. At the open boundaries, the diffusive flux is set to zero in order to preserve sediment mass.

8.2.3 Time integration

The time step selected for updating the morphology can take values larger than the one used for solving transport equations in the water column. To preserve accuracy, higher order time integrations using multi step methods (Adams-Bashforth schemes) can be selected by the user. The following schemes are available

- First order Euler method

$$\Delta\eta_n^{n+1} = -(1-p)\Delta t_{mor}(\nabla \cdot \mathbf{q}_n + E_n - D_n)^{n+1} \tag{8.6}$$

- Second order Adams-Bashforth method

$$\Delta\eta_n^{n+1} = -(1-p)\Delta t_{mor} \left[\alpha_1(\nabla \cdot \mathbf{q}_n + E_n - D_n)^{n+1} + \alpha_2(\nabla \cdot \mathbf{q}_n + E_n - D_n)^n \right] \tag{8.7}$$

with $\alpha_1 = 3/2$, $\alpha_2 = -1/2$.

- Third order Adams-Bashforth method

$$\begin{aligned}
\Delta\eta_n^{n+1} &= -(1-p)\Delta t_{mor} \left[\beta_1(\nabla \cdot \mathbf{q}_n + E_n - D_n)^{n+1} + \beta_2(\nabla \cdot \mathbf{q}_n + E_n - D_n)^n + \right. \\
&\quad \left. \beta_3(\nabla \cdot \mathbf{q}_n + E_n - D_n)^{n-1} \right]
\end{aligned} \tag{8.8}$$

with $\beta_1 = 23/12$, $\beta_2 = -4/3$, $\beta_3 = 5/12$.

In (8.6)–(8.8), Δt_{mor} is the morphological time step¹. The superscripts $n+1$ and n denote values at the new, respectively new time step.

¹The morphological time step must either be equal to the 3-D time step Δt in case suspended sediment transport is included or may be a multiple of Δt if only bed or total load is included.

Switches

The following switches are available:

`iopt_morph_hdiff` Disables/enables the use of numerical diffusion in the bed elevation equation.

0: Disabled.

1: Enabled.

`iopt_morph_time_int` Selects the kind of time integration scheme for the bed elevation equation.

1: First order Euler scheme.

2: Second order Adams-Bashforth scheme.

3: Third order Adams-Bashforth scheme.

`iopt_morph_time_init` Informs the program, in the case of a higher order time integration, whether the values of the bed elevation at previous times have been provided by the user at the initial time.

0: Not provided. A first order time integration scheme is used at the first time step, in case `iopt_time_int=2,3`. A second order time integration scheme is used at the second time step, in case `iopt_time_int=3`.

1: Provided as initial conditions.

8.3 Morphological acceleration

The evaluation of the bathymetry is a slow process, typically much slower than the changes that occur in the flow. This means that it is often necessary to simulate large periods of time. However, simulating these long periods on the time scale of the fluid flow would be a very costly operation. Therefore, different techniques have been proposed to accelerate the calculation of the bathymetry in a way that a smaller number of hydrodynamic calculations need to be performed, by extrapolating in some way the effect of the hydrodynamic calculations on the bed morphology during one representative period to the effect of many representative periods. Two methods are available in COHERENS: (1) use of a morphological factor and (2) the tidal averaging method.

8.3.1 Morphological factor

In this approach, the bed or total transport terms are multiplied at each time step by a constant morphological acceleration factor in order to accelerate the morphological evolution. Compared to the tidal averaging approach (Roelvink, 2006), discussed below, this method tends to be more stable and is simpler to use. Disadvantage is that the erosion and deposition rates in the bed level and the water column transport equations should be multiplied by the same factor by conservation of mass, yielding unrealistically high (and possibly numerically unstable) sediment concentrations in the water column. The method can therefore only be used in the absence of suspended transport (i.e. `iopt_suseq` equal to zero).

8.3.2 Tidal averaging

The procedure, used in the tidal averaging method, is as follows:

- The simulated period is divided into a number N_p of morphological periods.
- Each morphological period is further divided into N_{mor} cycles.
- During the first (hydrodynamic) cycle, hydrodynamic calculations are enabled. Values of physical parameters (depth-mean current, surface elevations and water depths) are stored at each morphological time step.
- The first (hydrodynamic) cycle is followed by $N_{mor} - 1$ morphological cycles without explicit hydrodynamic calculations, but using the larger morphological time step for sediment transport and morphology. During each cycle the hydrodynamic data are obtained from the values stored during the first cycle and adjusted to the bathymetric changes that might have occurred over time.
- After the end of each morphological period, a new period begins starting (again) with a hydrodynamic cycle followed by $N_{mor} - 1$ morphological cycles.

The following setup parameters need to be provided by the user

<code>morphsteps</code>	The number of morphological cycles N_{mor} .
<code>nstep_hydro</code>	The number of hydrodynamic time steps during the hydrodynamical cycle.

number_tidal_steps The number of morphological time steps during the hydrodynamical and morphological cycles.

The number of morphological periods N_p is then defined by dividing the total number of time steps in the simulation `nstep` by `nstep_hydro*morphsteps`.

The estimated depth-mean current used during the morphological cycles is related to the one which was previously stored during the hydrodynamical cycle

$$(\bar{u}^n, \bar{v}^n) = F_c(\bar{u}^l, \bar{v}^l) \quad (8.9)$$

where ⁿ denotes the time step during one of the morphological cycles, ^l, the corresponding morphological time step during the previous hydrodynamic cycle and F_c is the applied correction factor which can be obtained in four ways ([Fortunato & Oliveira, 2004](#)).

1. No correction is applied, i.e. $F_c = 1$.
2. The continuity correction. This velocity correction is based on the assumption that the water fluxes are only weakly affected by the bathymetric changes, and is given by:

$$F_c = F_{cont} = \frac{H^l}{H^n} \quad (8.10)$$

where H is the total water depth.

3. The friction correction. [Friedrichs & Madsen \(1992\)](#) showed that inertia terms are one to two orders of magnitude smaller than friction in shallow tidal embayments. When the former terms are neglected, the momentum equation becomes a balance between friction and the barotropic pressure gradient. The friction correction is expressed as:

$$F_c = F_{fric} = \sqrt{\frac{C_{db}^l H^n}{C_{db}^n H^l}} = \frac{1 + \ln(H^n/z_0)}{1 + \ln(H^l/z_0)} \sqrt{\frac{H^n}{H^l}} \quad (8.11)$$

where z_b is the distance of the lowest C-grid cell from the sea bed.

4. The two previous approximations can be combined into a mixed continuity-friction correction by taking an average of the previous two coefficients:

$$F_c = \sqrt{F_{cont} F_{fric}} = \left(\frac{C_{db}^l}{C_{db}^n} \right)^{1/4} \left(\frac{H^l}{H^n} \right)^{1/4} = \sqrt{\frac{1 + \ln(H^n/z_0)}{1 + \ln(H^l/z_0)}} \left(\frac{H^l}{H^n} \right)^{1/4} \quad (8.12)$$

Switches

The following switches are available:

`iopt_morph_accel` Selects the method for morphological acceleration.

- 0: Disabled.
- 1: Using a morphological acceleration factor.
- 2: Using the tidal averaging method.

`iopt_morph_tidal_scheme` Selects the type of velocity correction if tidal averaging is enabled.

- 0: No correction is applied.
- 1: Continuity correction (8.10).
- 2: Friction correction (8.11).
- 3: Combined continuity/friction correction (8.12).

8.4 Bed layers

8.4.1 Fixed layer

In the simplest case, the bed consists of an endless amount of sediment across the entire domain. In other words, all the sediment in the bed is considered as erodible. A fixed layer, on the other hand, is a layer that cannot be eroded by the flow; it is immobile and remains as such for the duration of the simulation. The mobile sediment that lies on top of this fixed layer is finite and can be exhausted. When all sediment on top of a fixed layer has been transported and the bed has reached the level of the fixed layer, sediment transport halts.

This principle is implemented in COHERENS in the following way. When the inclusion of fixed layers in the model is desired, a user-specified, spatially variable sediment layer thickness h_{sf} for the bed needs to be provided to the model for the entire domain. When at some point during the simulation, the thickness of this bed layer reaches zero in a certain cell, the properties of the layer change in this cell: all of the bed fractions as well as the layer thickness are set to zero.

8.4.2 Active layer

The active layer – first proposed by [Armanini \(1995\)](#) – represents the part of the bed that is exposed to and interacts with the flow, and is thus subject to entrainment and deposition. The sediment underneath the active layer is treated as an immobile, homogeneous substrate, in which no sediment motion occurs ([Hirano, 1971](#)).

Three equations for the active layer thickness h_{sa} are implemented in COHERENS:

1. Flat-bed conditions ([Armanini, 1995](#)): $h_{sa} = 4.5d_{50}$
2. In the presence of bedforms ([Ribberink, 1987](#); [Parker, 1991](#); [Armanini, 1995](#); [Blom, 2003](#)): $h_{sa} = 0.5z_{bh}$, where z_{bh} is a user-defined average bedform height.
3. A combination of sediment features and hydraulic conditions ([Harris & Wiberg, 1997](#)):

$$h_{sa} = \max \left[k_1 \rho (\tau_b^s - \tau_{cr}), 0 \right] + k_2 d_{50} \quad (8.13)$$

where k_1 and k_2 are empirical constants with values of 0.007 and 6.0 respectively, τ_b^s the skin-friction stress and τ_{cr} the critical shear stress for erosion, averaged over all sediment classes

$$\tau_{cr} = \sum_{n=1}^{N_f} f_n \tau_{cr,n} \quad (8.14)$$

8.4.3 Conservation of bed layer sediment

Exhaustion of the fixed or active layer may occur, if the bed level displacement $\Delta\eta$ obtained from (8.1) is negative and $\Delta\eta < -h_{av}$ where h_{av} is the amount of available sediment, given either by h_{sf} , h_{sa} or $\min(h_{sf}, h_{sa})$. A conservation scheme for sediment mass in both the water column and bed layer has been implemented in COHERENS.

Firstly, the bed level equation is written in the form

$$(1 - p) \frac{\partial \eta}{\partial t} = D - E - \nabla \cdot \mathbf{q}_{tr} = D - E + Q_+ - Q_- \quad (8.15)$$

where Q_+ and Q_- on the right are source and sink terms representing respectively the inflow and outflow of mass within the grid cell. To simplify the notation the fraction index has been removed.

The mass conservation algorithm depends on the type of time integration

- First order Euler.

Equation (8.15) can be rewritten as

$$\begin{aligned}\Delta\eta^{n+1} &= \frac{\Delta t_{morph}}{1-p}(D + Q_+) - \frac{\Delta t_{morph}}{1-p}(E + Q_-) \\ &= S_+^{n+1} - S_-^{n+1} = \Delta\eta_+ - \Delta\eta_-\end{aligned}\quad (8.16)$$

To preserve conservation of mass, the sink term is multiplied by a correction factor α_{cor} such that

$$S_+^{n+1} - \alpha_{cor} S_-^{n+1} = -h_{av} \quad (8.17)$$

giving

$$\alpha_{cor} = \frac{S_+^{n+1} + h_{av}}{S_-^{n+1}} \quad (8.18)$$

- Second-order Adams-Bashforth scheme.

Equations (8.16)–(8.18) are replaced by

$$\begin{aligned}\Delta\eta^{n+1} &= \alpha_1(S_+ - S_-)^{n+1} + \alpha_2(S_+ - S_-)^n \\ &= \Delta\eta_+ - \Delta\eta_-\end{aligned}\quad (8.19)$$

$$\alpha_1(S_+ - \alpha_{cor} S_-)^{n+1} + \alpha_2(S_+ - S_-)^n = -h_{av} \quad (8.20)$$

$$\alpha_{cor} = \left(\frac{1}{\alpha_1} (h_{av} + \alpha_2(S_+ - S_-)^n + S_+^{n+1}) \right) / S_-^{n+1} \quad (8.21)$$

- Third-order Adams-Bashforth scheme.

Equations (8.16)–(8.18) are replaced by

$$\begin{aligned}\Delta\eta^{n+1} &= \beta_1(S_+ - S_-)^{n+1} + \beta_2(S_+ - S_-)^n + \beta_3(S_+ - S_-)^{n-1} \\ &= \Delta\eta_+ - \Delta\eta_-\end{aligned}\quad (8.22)$$

$$\beta_1(S_+ - \alpha_{cor} S_-)^{n+1} + \beta_2(S_+ - S_-)^n + \beta_3(S_+ - S_-)^{n-1} = -h_{av} \quad (8.23)$$

$$\alpha_{cor} = \left(\frac{1}{\beta_1} (h_{av} + \beta_2(S_+ - S_-)^n + \beta_3(S_+ - S_-)^{n-1} + S_+^{n+1}) \right) / S_-^{n+1} \quad (8.24)$$

The mass conservation scheme is implemented by an iteration procedure.

1. A first iteration starts by identifying (marking) the critical grid cells with a mass deficit.
2. The bed elevation equation is solved at each critical cell.
3. The correction factor α_{cor} is defined for each critical cell.

4. Erosion rates at cell centers are multiplied by α_{cor} . The cell becomes empty.
5. If the bed elevation equation is solved without sediment transport (i.e. $q_{tr} = 0$), no further action is needed and the iteration stops.
6. Outward sediment transport terms at the velocity nodes of the grid cell are multiplied by α_{cor} .
7. Changing the outward transports at the velocity nodes affects the mass balance in the neighbouring cells where an inflow occurs from the previous critical cells. These adjacent cells are now marked as critical.
8. A next iteration starts from step 2. The iteration stops when there are no remaining critical cells or the number of iterations exceeds a user-defined value.

The algorithm is repeated for each fraction.

To prevent that different erosion rates are used in the transport equation for suspended sediment and the bed elevation equation, the following procedure is followed

1. The erosion rate is first defined in the suspended transport routine.
2. The morphology is updated. Erosion rates are adapted for mass conservation where necessary.
3. The suspended transport is updated in time with the (eventually) adapted erosion rates.

8.4.4 Mass balance

The conservation of sediment mass can (optionally) be tested by the user through the mass balance equation which is obtained by integrating the sediment transport equation over the water column, the bed elevation equation over the sediment layer and adding the two integrals. This gives

$$\begin{aligned}
 & \iiint_V \frac{\partial c_t}{\partial t} h_1 h_2 h_3 d\xi_1 d\xi_2 ds + \frac{\partial}{\partial t} \iint_S (1-p) \eta h_1 h_2 d\xi_1 d\xi_2 \\
 &= \int_W \overline{q_{tr,1}} h_2 d\xi_2 - \int_E \overline{q_{tr,1}} h_2 d\xi_2 + \int_S \overline{q_{tr,2}} h_1 d\xi_1 - \int_N \overline{q_{tr,2}} h_1 d\xi_1
 \end{aligned} \tag{8.25}$$

where $\overline{\mathbf{q}_{tr}}$ is the transport averaged over all fractions and the integrals at the right hand side are performed along respectively the western, eastern, southern and northern open boundaries. Equation (8.25) is integrated in time between $(n-1)T$ and nT where T is the ingration period for which mass conservation is tested and $n = 1, 2, \dots$. The results are used for monitoring. See Section 23.2.3 for details. Note that the procedure can only be used in case of a fixed layer depth and a first-order time integration.

Switches

The following switches, related to this section, have been implemented

`iopt_morph_active_layer` Selects the type of formulation for the active layer thickness.

- 0: Disabled.
- 1: Flat-bed conditions ([Armanini, 1995](#)).
- 2: Using the average bedform height ([Ribberink, 1987](#); [Parker, 1991](#); [Armanini, 1995](#); [Blom, 2003](#)).
- 3: Combination of sediment features and hydraulic conditions ([Harris & Wiberg, 1997](#)).

`iopt_morph_corr` Selects the type of action taken when more sediment is retrieved from the bed layer than available.

- 0: The active or fixed bed layer depth is set to zero without preserving sediment mass.
- 1: The mass conservation algorithm is used.

`iopt_morph_fixed_layer` Disables/enables a fixed (finite) sediment later depth.

- 0: Disabled.
- 1: Enabled.

8.5 Vertical sorting

For some applications, it is required to keep track of not only the bed level, but also of the properties - i.e. thickness and sediment fraction distribution - of each separate layer the bed is composed of. These properties can change through erosion, deposition and other vertical sorting mechanisms through

which sediment is exchanged between different bed layers². When the vertical sorting module is switched on (by setting a value for the user-defined parameter N_b which is larger than 1 and setting `iopt_morph_fixed_layer` to 1), the bed is subdivided into a number of layers N_b , each with a user-defined bed layer thickness and distribution of the N_f bed fractions³. The fixed layer depth is then defined as the sum of the layer depths. During computation, changes in both bed layer thickness and bed fraction distribution due to erosion and deposition are calculated and stored in memory, enabling the study of the evolution of the bed level as well as the structure and composition of the bed.

The vertical sorting mechanisms, described below, update the characteristics of the separate bed layers. Sorting is split up into erosion and deposition (the reasoning is explained below), with the mechanisms for erosion and deposition explained in sections 8.5.1–8.5.2. When the Ribberink exchange model is chosen, an extra exchange of sediment occurs between the two top layers. This is discussed in section 8.5.3.

In case of a single bed layer, a mass balance of deposited and eroded sediment yields a single bed displacement $\Delta\eta$ that is added to the bed level η .

However, in the case of multiple layers, the sediment is not uniformly spread over the bed and can vary from cell to cell and from bed layer to bed layer. In order to keep the vertical fraction distribution intact, the deposited sediment is not mixed with the sediment already present, but deposited on top of it in a new layer (see below). Knowing this, combining deposition and erosion into one single $\Delta\eta$ would disrupt the mass balance between layers: instead of eroding a certain mass of sediment m_1 of grain size d_p and then depositing a mass m_2 of the same sediment in the new top layer, nothing is eroded and the difference $m_2 - m_1$ is deposited in the new layer. Furthermore, if $m_1 > m_2$, the bed fraction of this specific grain size in the new layer becomes negative.

The above demonstrates that combining erosion and deposition into one $\Delta\eta$ has negative consequences for the fractional mass balance per layer. Therefore, erosion and deposition are processed separately, and the single bed displacement $\Delta\eta$ is subdivided into a bed displacement due to deposition $\Delta\eta_+$ and a bed displacement due to erosion $\Delta\eta_-$.

Note that, after choosing the amount of bed layers N_b at the model initialisation, this amount cannot change during computation. This means that

²In this section erosion and deposition stand for the source and sink terms S_+ and S_- defined in equation (8.16) which may include besides erosion/deposition also the outward/inward transport of sediment at velocity nodes.

³The bed layers are numbered downwards from 1 at the water column interface to N_b at the bottom of the sediment layer.

when a new layer is created during the sorting procedure, another layer needs to be deleted and vice versa.

The sorting mechanisms then consist of a deposition step followed by an erosion step.

8.5.1 Deposition step

When a new layer is created during the deposition step, the following procedure is followed (see Figure 8.1):

1. Sediment is deposited into a new layer, except when the fraction distribution of the deposited sediment is similar to the distribution within the existing upper layer (see below).
2. If the two bottom layers N_b and $N_b - 1$ are non-empty, these two layers merge together into the new layer N_b . If not, the old empty layer N_b is deleted.
3. An internal shift of the layers: the thickness and fraction distribution of an old layer k are transferred to a new layer $k + 1$ (with $k=1$ to $N_b - 2$ when the previous step resulted in a merged new bottom layer N_b , and $k=1$ to $N_b - 1$ otherwise).

However, the creation of a new layer is in some cases not the best solution. The reason is that as the creation of new layers is accompanied by the merging of others, a sequence of deposition steps leads to a loss of structure in the bed as all layers eventually merge together into one well-mixed layer. Since one of the objectives of the vertical sorting procedure is to study the evolution of the bed layer structure as detailed as possible, this is rather undesirable. A case could be made for the opposite, always merging the deposited sediment with the sediment in the top layer, and relying on the erosion mechanism to filter the transportable sediment fractions from the non-transportable fractions. While certainly a sound idea in many cases, this may lead to an erroneous bed composition with potentially large consequences.

Consider a channel with a bed constructed out of two halves: the top layer of the upstream half consists of rough sediment, while the top layer of the downstream half consists of fine sediment. At a certain moment, the flow is such that both sediment sizes are transported by the flow, resulting in a newly deposited layer of rough sediment on top of the layer of fine sediment. Now, consider a later stage of lower flow intensity during which the rough sediment can no longer be transported. If the thickness of the newly constructed rough layer is greater than the active layer thickness, the

flow cannot reach the fine sediment underneath and the fine sediment layer is considered immobile. The rough layer is protecting the underlying fine sediment, sheltering it from erosion. This process is called armouring, and can be important in cases where the bed consists of highly varying grain sizes, as bed deformation effectively stops at locations where armouring occurs.

If, alternatively, the rough sediment had been mixed with the fine sediment during deposition, the bed level would still erode at the lower stage of flow due to the continuous availability of fine sediment among the rough sediment in the well-mixed layer. Erosion would only cease when the amount of fine sediment in this layer is exhausted and only rough sediment remains. In case this well-mixed layer is rather thick, the consequential delay to the bed deformation stop could potentially introduce significant morphological errors.

In order to avoid this error while still preserving as much information as possible on the bed level structure, a similarity criterion is introduced in a way that which determines whether a new layer is created or all new deposited sediment is put into the upper layer.

$$\begin{cases} (1 - S_R)f_n \leq f_n^d \leq (1 + S_R)f_n & \text{deposition into layer} \\ f_n^d < (1 - S_R)f_n \quad \text{or} \quad (1 + S_R)f_n < f_n^d & \text{new layer} \end{cases} \quad (8.26)$$

where f_n^d is the n^{th} bed fraction of the deposited sediment, f_n is the n^{th} bed

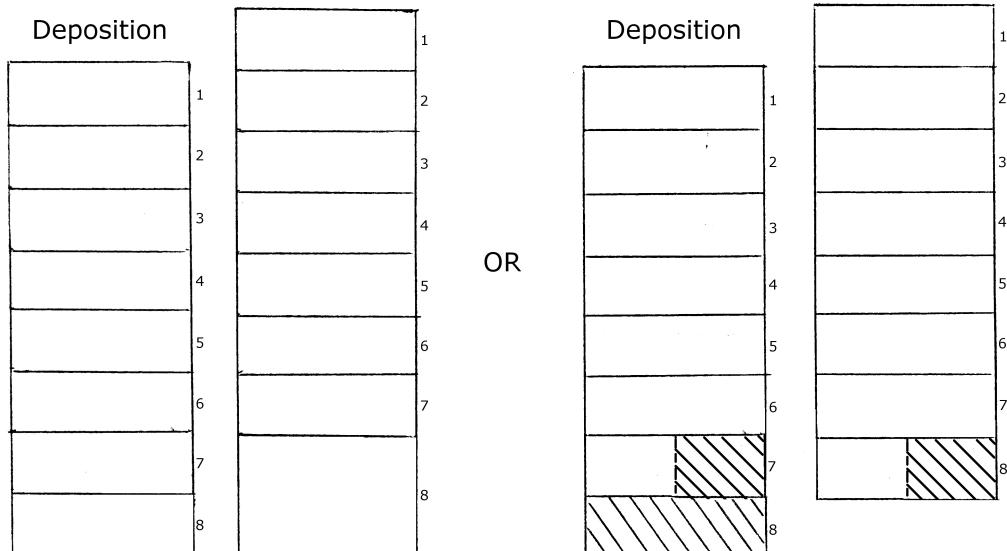


Figure 8.1: The deposition algorithm for two non-empty/empty bottom layers (left frame) or an empty bottom layer combination (right frame).

fraction of the sediment in the existing top layer and S_R is a user-defined parameter which defines the range of deposited fractions for which no new layer is created. Default value is 0.05.

8.5.2 Erosion step

When a layer is deleted during the erosion step, the following happens, as illustrated in Figure 8.2.

1. The number of layers removed by erosion N_r is determined.
2. If no layers are removed, only the thickness and fraction distribution of the upper layer need to be adapted.
3. When $N_r > 0$ layers are removed, thickness and fraction distribution of a new top layer is determined from the amount and fraction distribution of the eroded sediment.
4. An internal shift is made for the other layers: the thickness and fraction distribution of the old layers $N_r + 2$ to N_b are transferred to the new layers 2 to $N_b - N_r$.
5. Empty layers with index $N_b - N_r + 1$ to N_b are created to preserve the number of layers.
6. If an active layer has been defined, a second sorting procedure is performed such that the upper layer has a thickness equal to the active layer depth. Layers are added or removed depending on whether the active layer depth is smaller or larger than the depth of the first layer. One or more empty layers are added below in the first, bottom layers are fused in the second case.

8.5.3 Sediment exchange fluxes

After revealing the existence of an extra layer between the active layer and the underlying substrate during a series of equilibrium experiments, [Ribberink \(1987\)](#) developed a so-called two-layer model. This two-layer model consists of an active layer with the same characteristics as the active layer suggested by [Hirano \(1971\)](#). Underneath the active layer, an exchange layer of thickness $\delta_e = 1.22h_a$ is situated. This layer is only reached occasionally by deep bed form troughs, that cause a vertical sediment exchange between the two layers: exchange layer material is picked up and brought into transportation,

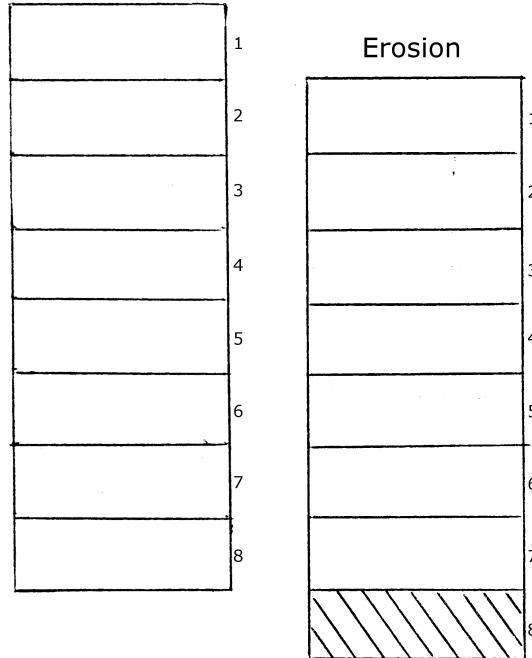


Figure 8.2: The erosion algorithm with creation of a fictive fixed layer with thickness and bed fractions equal to zero.

while active layer material is deposited in the exchange layer. This exchange process can lead to a resulting flux ψ_i of size fraction i from exchange layer to transport layer ($\psi_i > 0$ or $\psi_i < 0$) and, most importantly, induces the size fractions to become redistributed over bed elevations even in situations without net aggradation or degradation.

After analyzing the deposition and erosion that occurs along a series of bed forms with occasionally deep troughs, [Ribberink \(1987\)](#) derived the following formula for the sediment exchange flux between the exchange layer and the active layer:

$$\psi_i = \gamma_t \frac{q_a}{\lambda_a} (p_{ei} - p_{miD}) \quad (8.27)$$

where λ_a denotes the average bedform length, γ_t a proportionality factor depending on the subdivision between the active layer and the exchange layer ($\gamma_t \approx 0.06$), p_{ei} the volume fraction content of size fraction i in the exchange layer and p_{miD} the volume fraction content of size fraction i in the sediment deposited from the active layer into the exchange layer. This sediment is assumed to be somewhat coarser than the average composition of the active

layer because of the downward coarsening trend within the bedforms. For a mixture of two well-sorted sand fractions, Ribberink found that the volume fraction content of the finest fraction deposited from the active layer into the exchange layer is about 70% of that in the active layer:

$$p_{m1D} \approx 0.7p_{m1} \quad (8.28)$$

and thus, since there are only two sediment fractions:

$$p_{m2D} = 1 - p_{m1D} \quad (8.29)$$

Ribberink did not develop formulations for the grain-size specific deposition flux from the active layer to the exchange layer for mixtures of more than two fractions. Therefore, the application of this model is restricted to beds that consist of only two bed fractions.

Switches

`iopt_morph_vert_fluxes` Selects the kind of vertical exchange model

- 0: Disabled.
- 1: Enabled using the exchange layer concept of [Ribberink \(1987\)](#).

8.6 Avalanching

From geomechanics, it is known that every granular material possesses a so-called static angle of repose, i.e. the steepest angle of descent or dip of the slope relative to the horizontal plane when material on the slope face is on the verge of sliding. When this slope exceeds the static angle of repose S_{cr}^s , it is likely to collapse and decrease to a lower value denoted as the dynamic angle of repose S_{cr}^d . This collapsing process is called avalanching.

Avalanching is enabled by setting the appropriate switch. Firstly, user-defined static and dynamic angles of repose for the bed material need to be supplied for both wet conditions (i.e. submerged bed slope) and dry-wet conditions (i.e. bank slope). Avalanching takes place after the morphological update of the bed, and is implemented as follows:

1. Calculate the bed slopes. If no critical slopes (i.e. slopes larger than the static angle of repose) are found, go to step 9.
2. Identify the cells where the bottom slope exceeds the critical value S_{cr}^s at one (or more) of the surrounding velocity (U-or V-nodes).

3. Of these cells, take the cell with the greatest water depth. Let (i,j) be the grid indices of the selected cell.
4. Of the four surrounding velocity nodes, identify the one with the largest (critical) slope.
5. Assume that the largest slope occurs at the eastern velocity node with grid indices $(i+1,j)$. The other cases are treated in a similar way. From the two grid cells with C-node indices (i,j) and $(i+1,j)$, sediment will be eroded from the shallowest one and deposited in the deepest cell. Updates of the mean water depths are given by

$$\begin{aligned} U_1 &= \frac{A_{i+1,j}}{A_{ij} + A_{i+1,j}} \left(|h_{ij} - h_{i+1,j}| - \Delta x_{i+1,j}^u S_{cr}^d \right) \text{Sign}(1, h_{ij} - h_{i+1,j}) \\ U_2 &= -\frac{A_{ij}}{A_{ij} + A_{i+1,j}} \left(|h_{ij} - h_{i+1,j}| - \Delta x_{i+1,j}^u S_{cr}^d \right) \text{Sign}(1, h_{ij} - h_{i+1,j}) \end{aligned} \quad (8.30)$$

where A_{ij} and $A_{i+1,j}$ are the areas of the two cells and

$$\begin{aligned} \text{Sign}(x, y) &= |x| & \text{if } y \geq 0 \\ \text{Sign}(x, y) &= -|x| & \text{if } y < 0 \end{aligned} \quad (8.31)$$

The new mean water depths are given by

$$\begin{aligned} h_{ij}^{new} &= h_{ij}^{old} - U_1 \\ h_{i+1,j}^{new} &= h_{i+1,j}^{old} - U_2 \end{aligned} \quad (8.32)$$

giving a bed slope equal to S_{cr}^d . Since U_1 and U_2 have opposite signs, erosion/deposition occurs at the shallowest/deepest cell.

6. In case that a fixed layer has been defined, adjust the layer depths and fraction distributions at the eroded and deposition cells.
7. Reset the bed slopes affected by the adjusted water depths.
8. If there exists still an amount of critical slopes (i.e. slopes larger than the dynamic angle of repose), repeat from step 2 until no more critical slopes are found.
9. Algorithm terminates.

Switches

`iopt_morph_avalanching` Disables/enables avalanching.

0: Disabled.

1: Enabled.

Chapter 9

Biological model

9.1 Introduction

The mathematical formulation of the ecological model is described in this section. It is quite straightforward as its main goal is to provide maximum flexibility to future manipulation of the code. Because of this generalistic approach, it has the potential to be applicable for a wide range of different climatological circumstances when used with care and provided the different major groups of the ecosystem are represented in the model. The model code is open source, the authors warn the user to adapt the model to the specific needs of the environment they are modeling. The articles of [Flynn \(2005\)](#) and [Franks \(2009\)](#) point out some critical side-marks regarding the construction of Nutrient–Zooplankton–Phytoplankton (NZP) models. The mass flows of the model without the transport equations are described in the subsection 'Ecological model'.

9.2 0–D formulation

The model has a water quality module and a deposition–regeneration module. The deposition–regeneration module can be seen as an imaginary fluff layer. It provides the skeleton for future inclusion of a real sediment module. The water quality module has three main compartment that describe the mass flows: one for the organisms, one for the ditritic organic matter (OM) and one for the nutrients PO_4 , NO_3 , $NH4$, BSi (Biogenic Silica) and DSi (Dissolved Silica). The compartment for the organisms is split in two subcompartments (phytoplankton and zooplankton). The phytoplankton is in turn again subdivided in silica consumers (SC) and non silica consumers (NSC). Zooplankton (Z) has no further subdivision. The subdivision may seem a

Table 9.1: The state variables of the ecological model

State variable		Unit
SC	Silicium consuming phytoplankton	$mgC\ m^{-3}$
NSC	No silicium consuming phytoplankton	$mgC\ m^{-3}$
Z	Zooplankton	$mgC\ m^{-3}$
OM	Organic matter	$mgC\ m^{-3}$
PO_4	Phosphate	$mgP\ m^{-3}$
NO_3	Nitrate	$mgN\ m^{-3}$
NH_4	Ammonium	$mgN\ m^{-3}$
DSi	Dissolved silicon	$mgSi\ m^{-3}$
BSi	Biogenic silicon	$mgSi\ m^{-3}$
SOM	Sediment organic matter	$mgC\ m^{-3}$
$PO_{4,s}$	Phosphate in sediment	$mgP\ m^{-3}$
$NO_{3,s}$	Nitrate in sediment	$mgN\ m^{-3}$
$NH_{4,s}$	Ammonium in sediment	$mgN\ m^{-3}$
BSi_s	Biogenic silicon in sediment	$mgSi\ m^{-3}$

bit concise, but new divisions are easily included in the model. Possibilities for other constructions are the division of phytoplankton in size classes, in grazed vs. non-grazed, in tolerance for turbulent waters or for salinity. The OM box is quite straightforward and has no subdivisions. The silicon cycle is modeled separately from the C cycle. The nutrient compartment describes the flows of the main nutrients (PO_4 , NO_3 , NH_4) and biogenic and dissolved Silica (BSi and DSi). The deposition–regeneration module has an organic matter compartment (SOM) and a nutrient compartment with the main nutrients ($PO_{4,s}$, $NO_{3,s}$, $NH_{4,s}$ and biogenic silica of the sediment BSi_s). The 14 state variables described by the model are listed in table 9.1. Figure 9.1(a) illustrates the mass flows of the water quality module, Figure 9.1(b) illustrates the mass flows of the sedimentation–resuspension module.

Autotrophic phytoplankton uptakes nutrients dissolved in the water column (PO_4 , NO_3 , NH_4 and DSi). The NSC state variable has no need for Si , whereas the SC does. The growth rate of phytoplankton is influenced by the photosynthesis rate, temperature and nutrient availability. Phytoplankton is grazed by heterotrophic zooplankton. Part of the phytoplankton mortality and biomass loss is not caused by grazing. This part of the phytoplankton compartment flows to the organic matter compartment. The Si content of SC goes to the BSi statevariable. Zooplankton feeds on organic matter and phytoplankton. Part of the zooplankton compartment flows to the organic matter compartment due to mortality, exsudation and faecal pellets (= excretion). The OM and BSi sediments to the bottom or is recycled

inside the water column into dissolved nutrients available for uptake by SC and NSC . The processes in the fluff layer and the water column are linked by regeneration. Regeneration is a bulk representation of processes such as dilution and resuspension. Figure 9.1(a) illustrates the mass flows of the water quality module. In the deposition-regeneration module only nutrients and organic matter are taken into account. SOM receives nutrients from the water quality module through sinking of the organic matter from the water column. Part of the organic matter is buried, part of it is remineralized into nutrients. Nutrients are delivered back to the water column through regeneration. The mass flows of this module are illustrated in Figure 9.1(b).

[Ning et al. \(2004\)](#); [Ning \(2004\)](#) found that 63% of the phytoplankton in the South China Sea were diatoms and 24% were dinoflagellates. [Il'Yash et al. \(2004\)](#) found that diatoms made up to 60% of the total biomass in an area close to the studied area. They found that *Guinardia striata* was the most abundant phytoplankton species. No information on zooplankton dynamics of the studied region is available. For the Pearl River estuary in the South China Sea, both [Liu et al. \(2010b,a\)](#) and [Tan et al. \(2004\)](#) found that zooplankton in the South China Sea is mainly composed of copepods. According to [Tan et al. \(2004\)](#) the copepods *Acartia spinicauda*, *Pavocalanus crassirostris*, *Oithona rigida*, *Paracalanus aculeatus* and *Euterpina acutifrons*, numerically dominated zooplankton counts; the grazing impact of copepods was 85%. The same article also mentions the swarm behavior of groups of mixed copepods, especially at fronts (this is actually for the discussion part). Based on this information we searched in literature for functional grazing behavior of copepods. Based on these findings the parameters for SC , NSC and Z are based on parameters for diatoms, dinoflagellates and copepods, respectively.

9.2.0.1 Elemental ratio

Each biomass state variable is assigned a specific elemental ratio of $C : Si : N : P$. This ratio is referred to as elemental ratio of the statevariable ER_i (i equal to SC , NSC , Z , OM or SOM). As mentioned in the introduction of this section, the literature search for parameters appropriate for SC was based on diatoms, for NSC values on dinoflagellates and for Z on copepods. Table 9.2 shows the elemental ratios of each biomass state variable and the literature source on which this assignment was based.

The ratio of SC is based on the ratio's for some of the more abundant diatom species mentioned in [Il'Yash et al. \(2004\)](#). The ratios for C , N and Si for four of the mentioned species were found in a review article of [Sarthou](#)

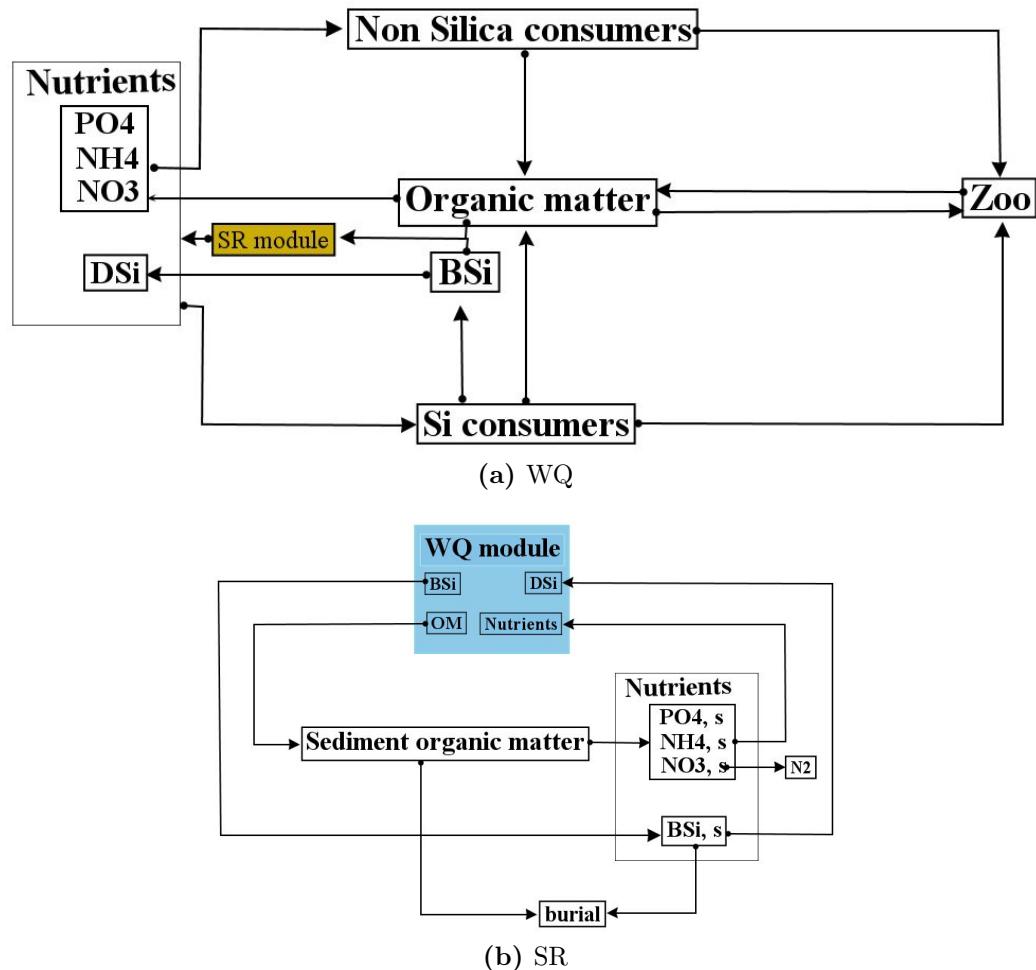


Figure 9.1: (a) Mass flows of the water quality module, with SR the sedimentation-resuspension module. (b) the sedimentation-resuspension module (b) with WQ module the water quality module

Table 9.2: The elemental ratio's of the state variables of biological origin; n denotes element (C, Si, N or P)

State variable	C	Si	N	P	Source
$ER_{SC,n}$	88	12.8	16	1	Sarthou et al. (200)
$ER_{NSC,n}$	101		14.25	1	Finkel et al. (2010)
$ER_{Z,n}$	88		18	1	Hannides et al. (2009)
$ER_{OM,n}$	106		15	1	Chen et al. (2001)
$ER_{SOM,n}$	106		15	1	Chen et al. (2001)

[et al. \(200\)](#), no information about P was available in this review. [Finkel et al. \(2010\)](#) found a ratio for $N:P$ between 4.9 and 17 for diatoms. Since no evidence is found for a correct value, the value of 16:1, which is the value of the classical redfield ratio, was taken.

The redfield ratio for NSC is based on the values found in literature for dinoflagellates. [Finkel et al. \(2010\)](#) found quite a broad range of redfield ratios for dinoflagellates, the average values were used in the model. [Finkel et al. \(2010\)](#) also found that the ratios are dependent on temperature and nutrient availability. This dependencies are not considered in this model.

The redfield ratio for Z are based on the values found in literature for copepods. The most common species of zooplankton in the South China Sea is *Acartia erythraea* ([Liu et al., 2010b,a](#)). [Hannides et al. \(2009\)](#) found for the stoichiometry of the standing stock of zooplankton $C:N:P$ values of 88:18:1. [Uye & Matsuda \(1988\)](#) mention that the elemental composition of species is region dependent. It was also mentioned in this article that the elemental ratio of zooplankton is more or less homeostatic and independent of nutrient availability. Because the lack of more appropriate data the ratio of [Hannides et al. \(2009\)](#) is included in the model.

Little information about the ER of OM was found. [Chen et al. \(2001\)](#) found for $N : P$ a value of 15. For the other elements nothing was found so the classical redfield ratio was maintained. There was no need to find a ratio for Si because the silicon cycle is modeled explicitly. The same elemental ratio was assigned to the sedimental fluff layer (i.e. SOM).

9.2.0.2 Biomass equations

Equations (9.1), (9.2) and (9.3) describe the mass flow of silicon consuming (SC) and no silicon consuming (NSC) phytoplankton and zooplankton (Z). The phytoplankton equations are analogue. The parameters found in literature for diatoms are assigned to the SC group, the parameters found

for dinoflagellates are assigned to the *NSC* group. The literature search for zooplankton was focused on the copepods, since this was the most abundant zooplankton group. This is more elaborated in the introduction of this section.

$$\frac{\partial SC}{\partial t} = b_{SC} SC - m_{SC} SC - g_{SC} Z \quad (9.1)$$

$$\frac{\partial NSC}{\partial t} = b_{NSC} NSC - m_{NSC} NSC - g_{NSC} Z \quad (9.2)$$

$$\frac{\partial Z}{\partial t} = \epsilon g_{SC} Z + \epsilon g_{NSC} Z + \epsilon g_{OM} Z - m_Z Z \quad (9.3)$$

where b_{SC} and b_{NSC} are growth rates of phytoplankton. They are dependent of temperature, rate of photosynthesis and nutrient availability. Most models refer to the relation found by [Eppley \(1972\)](#) to describe the influence of temperature on growth rate. About the influence of light on the growth rate is less concensus and many alternative equations and parameters are presented. [Behrenfeld & Falkowski \(1997\)](#) give a good overview on how to interpret and structure the huge amount of information. [Tian \(2006\)](#) gives a comprehensable overview on how this is incorporated in various ways in NPZD models. [Geider *et al.* \(1997\)](#) gives a detailed explanation on the mechanisms used to develop growth rates. Though a few assumption are made in this article, it is a widespread equation, it is well documented and validated and no other (theoretical and empirical) equations were available in the framework of this project. For this reason the birth rate presented by [Geider *et al.* \(1997\)](#) was adapted to the needs of this model. The general equation of growth looks as follows:

$$b_i = P_m^B D \theta \left(1 - \exp \left(-\frac{\alpha_i E}{P_m^B} \right) \right) \quad (9.4)$$

With P_m^B ($mgC\ mgChla^{-1}\ h^{-1}$) the specific photosynthesis rate at optimal illumination, D the fraction of daily light time, θ the Carbon to Chlorophyll a ratio ($mgC\ mgChla^{-1}$), α_i the maximum light irradiance coefficient in $gC\ gChla^{-1}\ h^{-1}\ (mole\ photons\ m^{-2}\ s^{-1})^{-1}$ with $i = SC$ or NSC and E irradiance ($mole\ photons\ m^{-2}\ s^{-1}$).

It is assumed that the carbon specific photosynthesis rate P_m^B depends on temperature and nutrient availability as follows:

$$P_m^B = P_{max}^B L_{Nu} f(T) \quad (9.5)$$

P_{max}^B is the maximum specific photosynthesis rate under nutrient depletion and at a reference temperature T_{ref} ($^{\circ}C$) in $mgC\ mgChla^{-1}\ h^{-1}$. L_{Nu}

is the nutrient limitation factor and described further in this section. $f(T)$ is a function of temperature and simulates the temperature dependence of phytoplankton growth. A widely used empirical equation comes from [Eppley \(1972\)](#):

$$f(T) = 0.59 \exp(0.0633T) \quad (9.6)$$

Recently this equation has been re-analyzed by [Bissinger et al. \(2008\)](#) and transformed to:

$$f(T) = 0.81 \exp(0.0631T) \quad (9.7)$$

[Bissinger et al. \(2008\)](#) predicted that models using the original equation would underestimate primary production by 30%. For now, this re-analysis will be withheld with the reservation that it is empirical and developed for temperate regions.

[Geider et al. \(1997\)](#) stated that the inclusion of D is an assumption. Because values are difficult to track down, the parameter D in equation (9.4) are included in equation (9.5) as follows:

$$P_m^B = P_{opt}^B L_{Nu} f(T) / D \quad (9.8)$$

With P_{opt}^B the optimal carbon specific photosynthesis rate ($mgC\ mgChla^{-1}\ h^{-1}$). A values for P_{opt}^B for the most dominant species (*Guinardia striata*) was not found, so a best guess assumption is based on information found in literature. [Ning et al. \(2004\); Ning \(2004\)](#) found that in winter 95% of the phytoplankton biomass were diatoms and in summer 79% of biomass was accounted to diatoms. [Siswanto et al. \(2006\)](#) recorded a P_{opt}^B of $1.29\ mgC\ mgChla^{-1}\ h^{-1}$ in winter and a P_{opt}^B of $6.13\ mgC\ mgChla^{-1}\ h^{-1}$ in summer. It is assumed that a linear relationship exists between P_{opt}^B and the species biomass and only the value in summer found by [Siswanto et al. \(2006\)](#) was selected. So a P_{opt}^B value of $6.13 * 0.79$ was selected for diatoms ($4.84\ mgC\ mgChla^{-1}\ h^{-1}$) and a value of 1.29 for dinoflagellates was selected. [Sarthou et al. \(2000\)](#) found an average of $2.6\ mgC\ mgChla^{-1}\ h^{-1}$ for diatoms with a range of P_{opt}^B between 1.18 and $11.4\ mgC\ mgChla^{-1}\ h^{-1}$. Many assumptions are made to estimate the P_{opt}^B value (linearity between P_{opt}^B and biomass and the values from [Siswanto et al. \(2006\)](#) were recorded for the East China Sea. [Il'Yash et al. \(2004\)](#) suggested that photoinhibition may also play a role in the growth curves of phytoplankton, this process was not taken into account in this model. More research on P_{opt}^B for the modeled area is necessary.

L_{Nu} in equation (9.4) is the nutrient limitation factor and is developed as a Michaelis–Menten limited function as follows:

$$L_{PO_4} = \frac{PO_4}{K_{PO_4} + PO_4}$$

$$L_{NH_4} = \frac{K_{NO_3}NH_4}{K_{NH_4}K_{NO_3} + K_{NO_3}NH_4 + K_{NH_4}NO_3}$$

$$L_{NO_3} = \frac{K_{NH_4}NO_3}{K_{NO_3}K_{NH_4} + K_{NH_4}NO_3 + K_{NO_3}NH_4}$$

$$L_{DSi} = \frac{DSi}{K_{DSi} + DSi}$$

$$L_{Nu,NSC} = \min\{L_{PO_4}, L_{NH_4} + L_{NO_3}\}$$

$$L_{Nu,SC} = \min\{L_{PO_4}, L_{NH_4} + L_{NO_3}, L_{DSi}\}$$

With K_i the saturation constant of nutrient i . The values for the Halong bay region, or the broader region of the South China Sea available in literature are limited. Most models developed for the South China See (Gan *et al.*, 2010; Lu *et al.*, 2010) use data from the ROMS model. We found data for the half saturation constant calibrated for models developed for mesocosm experiments (Dearman *et al.*, 2003), data from the literature review on diatoms (Sarthou *et al.*, 200) and dinoflagellates (Smayda, 1997) and data calibrated for specific regions (Christian *et al.*, 2002; Lacroix *et al.*, 2007; Menesguen *et al.*, 2007). It was decided to use the species specific data. For SC data the findings in the review of Sarthou *et al.* (200) were used (species specific if available, averages if no species information available) and the data reported for growth in the article of Smayda (1997). The half saturation constants found in literature for NH_4 and PO_4 are higher for diatoms than for dinoflagellates. For NO_3 it is the other way around. Sarthou *et al.* (200) suggest that diatoms may be the dominant species in nutrient rich waters. The values used for the half-saturation constants are listed in Table 9.3.

We followed several approaches to determine the value of α_i (i equals SC or NSC) or the slope of the EP curve at low light levels in this biogeochemical model (in $gC\ gChlA^{-1}\ m^2\ \mu mol^{-1}\ photons$). One approach was to determine the relationship between α_i , E and P_m^B (Jassby & Platt, 976; Grangere *et al.*, 2009). This is an empirical method, Sakshaug *et al.* (1997) demonstrates there is no physical relation between these parameters. Since no or little empirical investigation on this subject is done for the region of the Red River delta, this method is rejected. Biogeochemical models for the South China Sea included some standard parameters for α_i in ROMS model (Liu *et al.*, 2007, 2010b,a; Gan *et al.*, 2010). These values were expressed in energy units ($W\ m^{-2}$). Sakshaug *et al.* (1997) demonstrated that it is not possible to do a correct transformation towards mole photons unless the complete spectral curve is known. The input data for light irradiance E

in this model are in $mole\ photons\ m^{-2}\ s^{-1}$ so these values are not used in the model. Chauton *et al.* (1997) performed some mesocosm experiments and found several values for α_i . We selected two values obtained by Chauton *et al.* (1997) from experiments studying the succession of phytoflagellates. The value of 0.016 ($molephotonsm^{-2}s^{-1}$) $^{-1}$ was found when diatoms were the dominant species. This value is assigned to α_{SC} . The value of 0.018 ($molephotonsm^{-2}s^{-1}$) $^{-1}$ was found when dinoflagellates were the dominant species. This value is assigned to α_{NSC} . These values are in the range of values found by Moore *et al.* (2007).

We suggest there is a gap in our physical knowledge about photosynthesis and light. This is supported by the fact that although quite convincing empirical relations exist between α_i and P_m^B , no physical-mechanistic relationship exists and by the fact that Moore *et al.* (2007) found no clear relationship between the α 's measured with different methods.

θ_i is the Chlorophyll a to carbon ratio with i SC or NSC. This ratio is species dependent, but also depends on nutrient availability, depth and light irradiance, latitude and season (Taylor *et al.*, 1997). Cloern *et al.* (1995) derived an empirical relationship with temperature, light irradiance and nutrients:

$$Chla : C = 0.003 + 0.0154 \exp(0.050T) \exp(-0.059E) \min(L_{NH_4,i}, L_{DSi,i}) \quad (9.9)$$

With T temperature ($^{\circ}C$), E irradiance in $mole\ photons\ m^{-2}\ d^{-1}$. $L_{NH_4,i}$ and $L_{DSi,i}$ are the nutrient limitation factors for NH_4 and DSi for i (i equals SC or NSC) as described before. There are no useable data for the modeled area available. Faure *et al.* (2000) showed that a constant θ_i value tends to overestimate production, especially in nutrient poor regions. This poor estimation was also addressed by Buck *et al.* (1996). For these reasons θ_i is incorporated in the model as in equation 9.9, despite it was developed for the North Atlantic region. Buck *et al.* (1996) provided a relationship between Chlorophyll a and Carbon for the North Atlantic basin. Despite this relationship is not for the correct region, it can be used to recalibrate equation 9.9 if necessary.

For light irradiance E , typically only data on surface irradiance are available. The irradiance at other depths is deduced from Beer's law and dependent on the attenuation of seawater, the self shading effect and suspended matter. Beer's law looks as follows:

$$E = E_0 \exp(-k_E z) \quad (9.10)$$

With E the light irradiance, note that this value is the Photosynthetically Available Irradiance (PAR) and E_0 the light irradiance at the surface. k_E is

the light attenuation coefficient (m^{-1}) and z depth (m). The light attenuation coefficient can be split up to a term that account for the attenuation due to the suspended particles, a term due to the self shading effect of water and a term where pure sea water is responsible for the attenuation. Measurements for lumped light attenuation coefficients are available from the MODIS satellite. [Gan et al. \(2010\)](#) mentioned the lack of pelagic studies for the region, they used parameters developed for a different region. The formulation adopted for the attenuation coefficient here is for the attenuation coefficient

$$k_E = k_{E0} + k_{chl}chl \quad (9.11)$$

Two formulation are available for the light radiance at the surface. Either E_0 is taken as proportional to the solar radiance at the surface

$$E_0 = R_{par}Q_{sol} \quad (9.12)$$

or

$$E_0 = \left(\sin \gamma_{\odot} / \sin \gamma_{\odot}^{max} \right) E_p \quad (9.13)$$

where Q_{sol} is the surface solar radiation, γ_{\odot} the solar altitude, γ_{\odot} its value at noon and E_p the peak PAR at the surface.

The biomass loss of phytoplankton caused by processes other than grazing, is modeled as linear dependent on mortality rate (m_{SC} or m_{NSC}). It is limited by the abundance of the biomass itself, [Tian \(2006\)](#) discuss the different approaches to the inclusion of mortality in phytoplankton models. They also point out that this term is usually used as a closure term for the model. Little or no information could be found on direct estimates of m_i , the values used by [Gan et al. \(2010\)](#) and [\(Lu et al., 2010\)](#) used.

Phytoplankton is grazed by zooplankton (Z), with an assimilation efficiency ϵ of 0.75. No quantitative data for this value were found in literature, this value is used in many models ([Gan et al., 2010](#); [Lacroix et al., 2007](#); [Christian et al., 2002](#)). The predator response to changes in prey density is modeled as a functional response. [Tian \(2006\)](#) mentions three types of functional responses in his overview of the different modeling approaches used in studies of plankton dynamics. One of them is based on the Ivlev equation. This equation is applied in the ROMS model, hence the models available for the South China Sea use this equation ([Liu et al., 2007](#); [Gan et al., 2010](#)) and ([Liu et al., 2010b,a](#)). [Hansen & Bjornsen \(1997\)](#) provide information about half saturation constants for grazing and maximum ingestion rates of different groups of zooplankton. After conversion to C-specific parameters, these

data were used for modeling the functional response as a monod equation according to Moloney & Filed (1991):

$$g_i = \frac{g_{max} \ pref_i \ i}{K_g + \sum_{j,j \neq i} pref_j \ j}$$

With:

$$\begin{aligned} i, j &= \text{Grazed material (OM, SC and NSC)} \\ g_{max} &= \text{Maximum grazing rate of } Z \\ pref_i &= \text{Grazing preference for } i \\ K_g &= \text{Saturation constant for grazing} \end{aligned}$$

Little information about maximum grazing rates is available. According to Tan *et al.* (2004) the copepods *Acartia spinicauda*, *Pavocalanus crassirostris*, *Oithona rigida*, *Paracalanus aculeatus* and *Euterpina acutifrons*, numerically dominated zooplankton counts. None of these species were found in the study of Hansen & Bjornsen (1997). Therefor g_{max} was set to the average value found in this study which was 1.81 d^{-1} . This value lies in the range of grazing values found by Sarthou *et al.* (2000) for diatoms and by Calbet & Landry (2004). Hansen & Bjornsen (1997) mention that little information is available about food selectivity ($pref_i$), Kleppel *et al.* (1991) mention a preference for dinoflagellates, however in Kleppel *et al.* (1993) it is shown that diatoms are a substantial part of the diet of copepods (this is for the discussion). Based on this knowledge, the preference rates were conservatively assumed to be 0.4, 0.3 and 0.3 for *SC*, *NSC* and *OM* respectively. Since none of the most abundant copepods of this region was mentioned in the study of Hansen & Bjornsen (1997), the avarage value for K_g of 13.88 gC m^{-3} was used. It was difficult to compare with the values used in other models, because most models use different equations for the functional response (look up if you did the correct conversion!!).

The biomass loss (m_Z) of zooplankton accounts for excretion, lysis and grazing by larger trophic groups.

A total of 36 parameters is needed to solve the biomass equations. They are listed in table 9.3.

9.2.0.3 Organic matter equations

$$\frac{\partial OM}{\partial t} = (1 - \epsilon) (g_{SC} + g_{NSC} + g_{OM}) Z$$

Table 9.3: The parameters used to describe biomass flows of the water column.

* means the value is calculated within the model, ** means the value changes over time and is given as time dependent input from on-site measurements

Parameter	Description	Value	Unit
b_i	Growth rate of i , with i SC or NSC	*	h^{-1}
D	Fraction of day light	*	
E	Light irradiance	*	$mole\ photons\ m^{-2}\ s^{-1}$
E_0	Light irradiance at the surface	**	$mole\ photons\ m^{-2}\ s^{-1}$
g_i	Zooplankton grazing rate of i with i SC, NSC or OM	*	h^{-1}
g_{max}	Maximum zooplankton grazing rate	0.0754	h^{-1}
$L_{Nu,i}$	Nutrient limitation factor for i with i SC or NSC	*	—
k_{chl}	Coefficient in (9.11)		$m^2/mg/Chl$
k_{E0}	Light attenuation coefficient for pure water	0.35	m^{-1}
K_g	Half saturation constant for grazing	13.88 10^3	$mgCm^{-3}$
$K_{DSi,SC}$	Half saturation constant of nutrient DSi for SC	3.9	$mmolSi\ m^{-3}$
$K_{NH_4,SC}$	Half saturation constant of nutrient NH_4 for SC	1.7	$mmolN\ m^{-3}$
$K_{NH_4,NSC}$	Half saturation constant of nutrient NH_4 for NSC	0.6	$mmolN\ m^{-3}$
$K_{NO_3,SC}$	Half saturation constant of nutrient NO_3 for SC	0.4	$mmolN\ m^{-3}$
$K_{NO_3,NSC}$	Half saturation constant of nutrient NO_3 for NSC	1.02	$mmolN\ m^{-3}$
$K_{PO_4,SC}$	Half saturation constant of nutrient PO_4 for SC	2.5	$mmolP\ m^{-3}$
$K_{PO_4,NSC}$	Half saturation constant of nutrient PO_4 for NSC	0.18	$mmolP\ m^{-3}$
L_n	Limitation factor of nutrient n with n PO_4 , NH_4 , NO_3 or DSi	*	—
m_{NSC}	Mortality rate of NSC	0.00625	h^{-1}
m_{SC}	Mortality rate of SC	0.00625	h^{-1}
m_Z	Mortality rate of Z	0.00104	h^{-1}
$pref_{NSC}$	Zooplankton preference of NSC	0.3	—
$pref_{OM}$	Zooplankton preference of OM	0.3	—
$pref_{SC}$	Zooplankton preference of SC	0.4	—
$P_{opt,NSC}^B$	Optimal net primary production at nutrient saturation for NSC	1.29	$mgC\ mgChla^{-1}\ h^{-1}$
$P_{opt,SC}^B$	Optimal net primary production at nutrient saturation for SC	4.84	$mgC\ mgChla^{-1}\ h^{-1}$
t	time	**	h
T	Temperature	**	$^{\circ}C$
α_{NSC}	Maximum light utilization coefficient for NSC	0.016	$\frac{mgC\ mgChla^{-1}\ h^{-1}}{(mole\ photons\ m^{-2}\ s^{-1})^{-1}}$
α_{SC}	Maximum light utilization coefficient for SC	0.018	$\frac{mgC\ mgChla^{-1}\ h^{-1}}{(mole\ photons\ m^{-2}\ s^{-1})^{-1}}$
ϵ	Grazing efficiency of zooplankton C to $Chlorophyll\ a$ ratio, i is SC or NSC	0.75	—
θ_i		**	$mgC\ mgChla^{-1}$

$$+m_{SC} SC + m_{NSC} NSC + m_Z Z - \lambda_{OM}^* - g_{OM} Z - w_{OM} \frac{\partial OM}{\partial z} \quad (9.14)$$

The organic matter flow is increased by the exsudation of organic matter by zooplankton and mortality not caused by grazing of the organisms. It is decreased by remineralisation to nutrients with a remineralisation rate λ_{OM}^* , grazing of the organic matter by zooplankton and sinking. The remineralisation rate is modeled as a constant, so it is implicitly assumed that the water column never is depleted with oxygen. The sinking rate and the remineralisation rate of OM is little studied. The sinking rate is mainly a physical parameter and has little affinity with the specific features of the studied area. The value used in [Lu et al. \(2010\)](#) applied in the model. In [Lu et al. \(2010\)](#) remineralization rate of $0.03 d^{-1}$ was used for small organic matter, this value was included in this model. In the calibration and validation, special care is needed for the remineralization rate. The parameters used for the organic matter equation can be found in table 9.4.

9.2.0.4 Nutrient equations

When biomass flows from one compartment to the other, the elemental ratio of the biomass changes. The mass transitions are based on the C element. To make sure the mass balance of the other elements stays stable, a correction factor $ERCorrection$ is assigned to the nutrient equations. This factor is assumed to represent instantaneous remineralisation due to biological processes. There is no instantaneous remineralization to the NO_3 compartment. For the Si compartments there is no need for this correction factor, because the Si cycle is modeled explicitly. The correction factor for the elemental ratio is part of the remineralization process, for reasons of clarity it is described in two separate equations (9.16) and (9.17).

$$\begin{aligned}
 ERcorrection_{PO_4} = & m_{SC} SC (ER_{SC,PO_4} - ER_{OM,PO_4}) + \\
 & m_{NSC} NSC (ER_{NSC,PO_4} - ER_{OM,PO_4}) + \\
 & m_Z Z (ER_{Z,PO_4} - ER_{OM,PO_4}) + \\
 & (1 - \epsilon) Z g_{NSC} (ER_{NSC,PO_4} - ER_{OM,PO_4}) + \\
 & (1 - \epsilon) Z g_{SC} (ER_{SC,PO_4} - ER_{OM,PO_4}) + \\
 & \epsilon Z g_{NSC} (ER_{NSC,PO_4} - ER_{Z,PO_4}) + \\
 & \epsilon Z g_{SC} (ER_{SC,PO_4} - ER_{Z,PO_4}) + \\
 & \epsilon Z g_{OM} (ER_{OM,PO_4} - ER_{Z,PO_4}) \quad (9.15) \\
 & \quad (9.16)
 \end{aligned}$$

$$\begin{aligned}
ER_{correction_{NH_4}} = & m_{SC} SC (ER_{SC,NH_4} - ER_{OM,NH_4}) + \\
& m_{NSC} NSC (ER_{NSC,NH_4} - ER_{OM,NH_4}) + \\
& m_Z Z (ER_{Z,NH_4} - ER_{OM,NH_4}) + \\
& (1 - \epsilon) Z g_{NSC} (ER_{NSC,NH_4} - ER_{OM,NH_4}) + \\
& (1 - \epsilon) Z g_{SC} (ER_{SC,NH_4} - ER_{OM,NH_4}) + \\
& \epsilon Z g_{NSC} (ER_{NSC,NH_4} - ER_{Z,NH_4}) + \\
& \epsilon Z g_{SC} (ER_{SC,NH_4} - ER_{Z,NH_4}) + \\
& \epsilon Z g_{OM} (ER_{OM,NH_4} - ER_{Z,NH_4})
\end{aligned} \tag{9.17}$$

$$\frac{\partial PO_4}{\partial t} = -b_{SC} SC ER_{SC,PO_4} - b_{NSC} NSC ER_{NSC,PO_4} + \lambda_{OM}^* OM ER_{PO_4,OM} + ER_{correction_{PO_4}} \tag{9.18}$$

$$\begin{aligned}
\frac{\partial NH_4}{\partial t} = & -b_{SC} SC f_{NH_4} ER_{SC,NH_4} - b_{NSC} NSC f_{NH_4} ER_{NSC,NH_4} - nitrif NH_4 \\
& - nitrif NH_4 + \lambda_{OM}^* OM ER_{OM,NH_4} + ER_{correction_{NH_4}}
\end{aligned} \tag{9.19}$$

$$\frac{\partial NO_3}{\partial t} = -b_{SC} SC f_{NO_3} ER_{SC,NO_3} - b_{NSC} NSC f_{NO_3} ER_{NSC,NO_3} + nitrif NH_4 \tag{9.20}$$

$$\frac{\partial DSi}{\partial t} = -b_{SC} SC ER_{SC,DSi} + \lambda_{BSi}^* BSi \tag{9.21}$$

$$\frac{\partial BSi}{\partial t} = m_{SC} SC ER_{SC,BSi} + g_{SC} Z ER_{Z,BSi} - \lambda_{BSi}^* BSi - w_{BSi} \frac{\partial BSi}{\partial z} \tag{9.22}$$

Equations (9.18)–(9.22) describe the mass flows of the nutrients. The phosphate change (equation (9.18)) is decreased by the growth of *SC* and *NSC*. It is increased by remineralization of *OM* (λ^*) and regeneration of phosphate out of the sediment $PO_{4,s}$ into the water column with a regeneration rate r_{PO_4} . Ammonium flow is decreased by multiplication of phytoplankton and by nitrification of nitrate. The ammonium flow is increased by

remineralization of OM (λ^*OM) and regeneration of ammonium out of the sediment with a regeneration rate r_{NO_3} . Note that all the OM is remineralized into NH_4 . The nitrate concentration change is decreased by phytoplankton uptake. It is increased by nitrification. There is no regeneration of nitrate from of the sediment. It is assumed that phytoplankton has a stable preference for NH_4 incorporation by assigning the fractional parameter f_{NH_4} and $f_{NH_4} + f_{NO_3}$ equals 1. Dissolved Si (DSi) is decreased by SC growth and increased by regeneration of BSi out of the sediment into the water column. It is assumed that the regenerated Si from the bottom layer is readily available for SC take-up, r_{BSi} is the parameter which describes this process. BSi is assigned a sinking rate s_{BSi} . The nitrification rate is set to $0.066\ d^{-1}$ according to [Yool et al. \(200\)](#). They found that a study for the North Atlantic raised the nitrification rate significantly. The values of the nitrification rate in other studies matched better the lower nitrification rate, the calibration and validation part will investigate the influence of the nitrification rate with care. [Yool et al. \(200\)](#) found no significant relationship with time of the year or latitude. [Sutka et al. \(2004\)](#) found no significant denitrification in the oligotrophic water at the aloha station. Information about the sinking rates was mostly found for deep seas and a relationship with depth was mentioned (f.e. [Schmittner et al. \(2005\)](#)), the reported values are quite high ($5\ m\ d^{-1}$) in comparison to the values mentioned in [Lu et al. \(2010\)](#). For now, the values found in [Lu et al. \(2010\)](#) will be used, because no sound estimation for the modeled area was found. An overview of the 9 parameters used in the nutrient equations is provided in Table 9.4.

The vertical boundary conditions at the sediment water column interface are

- PO_4 : $r_{PO_4}\ PO_{4,sed}$
- NH_4 : $r_{NH_4}\ NH_{4,sed}$
- BSi : $r_{BSi}BSi_s$
- $\frac{SOM}{t}$: $OM\ s$

9.2.0.5 benthos-equations

The modeled sediment is a two dimensional fluff layer, no vertical dimensions are considered in this word, hence, the sediment state variables have units of $gC\ m^{-2}$ and the derivatives of time are fluxes.

$$\frac{\partial SOM}{\partial t} = w_{OM}\ OM(k=1) - b_{SOM}\ SOM - \lambda_s^* \quad (9.23)$$

Table 9.4: The parameters used to describe nutrient and organic matter flows of the water column

Parameter	Description	Value	Unit
f_{NH_4}	Preferencial uptake of NH_4 by phytoplankton	0.75	—
f_{NO_3}	Preferencial uptake of NO_3 by phytoplankton	0.25	—
$nitrif$	Nitrification	0.066/24	h^{-1}
r_{BSi}	Regeneration rate from sedimental BSi to Dsi	0.1/24	h^{-1}
r_n	Regeneration rate from sediment to water column of n with n PO_4 , NH_4 or NO_3	0.1/24	h^{-1}
s	Sinking velocity of organic matter	0.1/24	$m h^{-1}$
s_{BSi}	Sinking velocity of biogenic silicon	1/24	$m h^{-1}$
λ^*	Remineralization rate of OM	0.03/24	h^{-1}
λ_{BSi}^*	Remineralization rate of BSi	0.01/24	h^{-1}

The sediment organic matter (SOM) increases by sinking of OM from the water column, it decreases by sedimental remineralization with a rate of λ_s^* and burial. The burial of organic matter in this equation is a sink of the model (no recycling is foreseen at this stage of the model development).

$$\frac{\partial PO_{4,s}}{\partial t} = \lambda_s^* ER_{SOM,PO_{4,s}} - r_{PO_4} PO_{4,s} \quad (9.24)$$

$$\frac{\partial NH_{4,s}}{\partial t} = \lambda_s^* ER_{SOM,NH4,s} - r_{NH_4} NH_{4,s} - nitrif_s \quad (9.25)$$

$$\frac{\partial NO_{3,s}}{\partial t} = nitrif_s - denitrif_s - r_{NO_3} NO_{3,s} \quad (9.26)$$

$$\frac{\partial BSi_s}{\partial t} = w_{BSi} \frac{\partial BSi}{\partial z} - b_{BSi} BSi_s - r_{BSi} BSi_s \quad (9.27)$$

All nutrient flows increase because of remineralization of SOM except for NO_3 . All nutrients are resuspended in the water column. Ammonium decreases because of the process of nitrification ($nitrif_s$). Nitrate increases because of nitrification and solely decreases by denitrification. The denitrification process creates a sink for N in the model, at this stage, no recycling is included. BSi_s is made available to the water column by dissolution and resuspension. The burial parameter acts as a sink for the biogenic silicon of the model. The regeneration rates of all nutrients are assumed to be the same, the value is taken from [Lu et al. \(2010\)](#). [Bulleid \(1984\)](#) and others estimated that the nitrogen flux across the sediment provides 31% of the mean daily

Table 9.5: The parameters used to describe flows of the sediment

Parameter	Description	Value	Unit
b	Burial of SOM	0.1	h^{-1}
b_{BSi}	Burial of BSi_s	0.1	h^{-1}
$denitrif_s$	Denitrification in the sediment	1.2	$mg\ N\ m^{-2}h^{-1}$
$nitrif_s$	Nitrification in the sediment	6.8	$mg\ N\ m^{-2}h^{-1}$
λ_s^*	Remineralization rate of SOM	0.0671	$mgC\ m^{-2}h^{-1}$

phytoplankton requirement. [Odintsov *et al.* \(1993\)](#) reported rather low nitrification and denitrification values for the South China Sea. This seems to be in contradiction with articles about nutrient budgets of the South China Sea ([Liu *et al.*, 2007, 2010b,a](#); [Wong *et al.*, 2007](#)), who find a not closed N budget in the basin, they conclude that N fixation is the only process that can explain this. However, there seems no significant amount of N_2 fixating organisms present in the South China Sea. The values reported by [Odintsov *et al.* \(1993\)](#) will be used in this model. It is clear that the N budget needs further attention. [Chen *et al.* \(2001\)](#) deduced a high burial flux for carbon in the sediment ($0.2\ gC\ m^{-2}\ y^{-1}$) and estimated that 10% of the sediment gets buried. This was translated in the model as a burial rate for both BSi_s and SOM of $0.1\ h^{-1}$. [Chen *et al.* \(2001\)](#) estimated the remineralization rate of C in the South China Sea, this value was converted with the elemental ratio of SOM to a remineralization rate of all the elements. The parameters introduced in this section can be found in Table 9.5.

Chapter 10

Tracers and contaminant modules

10.1 Lagrangian tracer module

The use of Lagrangian particle tracking techniques in the marine environment, especially those in which tidal mixing is predominant, offers advantages over more traditional (Eulerian) solution methods of the advection-diffusion equation (e.g. finite differences). Computational effort is expended in areas where the particles are concentrated, allowing sources and sinks to be easily represented, and sharp gradients of the particle distributions to be resolved. This compares with the Eulerian approach in which all areas, regardless of concentration, are treated equally on a resolution defined by the grid structure, which is usually of a coarser resolution than needed, introducing numerical dispersion into the solution and a smoothing of frontal structure. [Hunter \(1997\)](#) showed that, when comparing particle tracking methods to second order finite difference methods, the ratio of the proportional errors is

$$\frac{E_p}{E_{fd}} = f^{(n/2+2)} N^{(n/2)} \quad (10.1)$$

where E_p is the error in particle tracking, E_{fd} the error in the finite difference solution, f the fraction of the computational domain covered by the contaminant, N the number of mesh points and n the number of dimensions (1, 2, or 3). The conclusion drawn is that for a given computational cost the particle tracking method is more accurate where the contaminant covers less than the whole domain ($f \ll 1$), and for cases of high dimension. The particle tracking algorithm is well suited for patch-like distributions such as oil spills, calculation of residence times of specific water masses within limited

areas, transport of living organisms such as fish larvae, tracking of specific substances such as radio-active nuclei, contaminants or floating debris.

10.1.1 Lagrangian transport equations

Particle transport can be simulated in two different modes: submerged (3-D case) or surface floats (2-D case)¹. The two cases are separately discussed below.

submerged particles

The evolution in time of the particle locations is obtained by solving the transport equations

$$\frac{dx_p}{dt} = \alpha_c u_a(\mathbf{x}_p, t) + u_d(\mathbf{x}_p, t) \quad (10.2a)$$

$$\frac{dy_p}{dt} = \alpha_c v_a(\mathbf{x}_p, t) + v_d(\mathbf{x}_p, t) \quad (10.2b)$$

$$\frac{dz_p}{dt} = w_a(\mathbf{x}_p, t) + w_d(\mathbf{x}_p, t) + w_b(\mathbf{x}_p, t) \quad (10.2c)$$

where \mathbf{x}_p is the particle's position vector and

- (u_a, v_a, w_a) are the components of the advective current obtained from the hydrodynamical model. When wave-current interaction within the water column is activated, particles are advected by the Lagrangian velocities given by the sum of the non-wave component and the Stokes drift velocity. Note that w is the physical (non-transformed) vertical current given by (5.31). The factor α_c represents the fraction of the surface current attributed to the surface drift. Default value is 1.
- (u_d, v_d, w_d) are the diffusive velocities defined below.
- w_b represents the fall or rising velocity due to buoyancy, vertical migration or swimming velocity, which can optionally be included.

The diffusive velocities are obtained from random walk theory

$$u_d = a_R \cos(\phi_R) \sqrt{\frac{r_h \nu_{Hx}}{\Delta t}} \quad (10.3a)$$

$$v_d = a_R \sin(\phi_R) \sqrt{\frac{r_h \nu_{Hy}}{\Delta t}} \quad (10.3b)$$

¹The partly-submerged case as used in e.g. oil dispersion models for which wind drift can be important, is not included in the current implementation.

$$w_d = a_R \sqrt{\frac{r_v \nu_{Vz}}{\Delta t}} \quad (10.3c)$$

where $(\nu_{Hx}, \nu_{Hy}, \nu_{Vz})$ are the diffusion coefficients in respectively the X-, Y-, Z-directions and a_R a random number between -1 and 1 with a standard deviation of 1, ϕ_R a random number between 0 and 360^0 and ν_{Hx} . Various values are found in the literature for the parameters r_h and r_v . [Visser \(1997\)](#) recommends $r_h = r_v = 2$ which are taken as default in the program. Alternative values are $r_h = 4$, $r_v = 2$ ([Wang *et al.*, 2008](#)) or $r_h = r_v = 6$ ([Maier-Reimer & Sündermann, 1982](#)).

The vertical buoyancy current is given by

$$w_b = \frac{(1-s)gd_p^2}{18\nu} \quad \text{if} \quad d_p \leq d_{crit} \quad \text{or} \quad \rho_p > \rho_w \quad (10.4a)$$

$$w_b = \sqrt{\frac{8}{3}gd_ps} \quad \text{if} \quad d_p > d_{crit} \quad \text{and} \quad \rho_p < \rho_w \quad (10.4b)$$

where $s = \rho_p/\rho_w$, ρ_p and ρ_w are the particle and water densities, d_p the particle diameter and ν is the kinematic viscosity. The critical diameter is defined by

$$d_{crit} = 9.52 \left(\frac{\nu^2}{gs} \right)^{1/3} \quad (10.5)$$

Note that (10.4a) is the same as Stokes's formula (7.41) defined in the sediment model. In the case of rising particles ($\rho_p < \rho_w$) the use of a critical diameter allows larger particles to be more buoyant and to remain longer near the surface while smaller particles are less buoyant and more easily transported downwards by diffusion.

10.1.1.1 floating particles

For particles floating at the surface the transport equations become

$$\frac{dx_p}{dt} = \alpha_c u_a(\mathbf{x}_p, t) + u_{wd}(\mathbf{x}_p, t) + u_d(\mathbf{x}_p, t) \quad (10.6a)$$

$$\frac{dy_p}{dt} = \alpha_c v_a(\mathbf{x}_p, t) + v_{wd}(\mathbf{x}_p, t) + v_d(\mathbf{x}_p, t) \quad (10.6b)$$

where x_p and y_p are the particle locations either in Cartesian coordinates or in longitude and latitude for the spherical case. The drift velocity on the right is divided in three components

- (u_a, v_a) are the components of the surface advective current obtained from the hydrodynamical model. When wave-current interaction within

the water column is activated, particles are advected by the Lagrangian velocities given by the sum of the non-wave component and the Stokes drift velocity. The parameter α_c represents the fraction of the surface current contributing to the particle drift. Default value is 1 (100%).

- (u_d, v_d) are the diffusive velocities defined by (10.3a)–(10.3b).
- (u_{wd}, v_{wd}) are the wind-driven currents defined by

$$u_{wd} = \alpha_{wd}\alpha_{lw}|\mathbf{U}_w| \cos(\phi_{wd} - \theta_{wd}), \quad v_{wd} = \alpha_{wd}\alpha_{lw}|\mathbf{U}_w| \sin(\phi_{wd} - \theta_{wd}) \quad (10.7)$$

where \mathbf{U}_w is the wind magnitude, ϕ_{wd} the wind direction, $\alpha_{wd} = 0.0315$ the fraction of the wind used for drifting and

$$\theta_{wd} = \alpha_{lw} \max(0^0, \theta_{d1} - \theta_{d2}\sqrt{|\mathbf{U}_w|}) \quad (10.8)$$

The crosswind factors θ_{d1} , θ_{d2} can be selected by the user from the US Coast Guard data base (Allen, 2005). Defaults are $\theta_{d1} = 40^0$, $\theta_{d2} = 8^0$. In addition, the crosswind drift component of floating objects has been observed to be either positive (right of the downwind direction) or negative (left of the downwind direction) and may change with a frequency of 4% per hour (Allen, 2005). This “Leeway” effect is presented in (10.8) by the additional factor

$$\alpha_{lw} = \text{Sign}(1, l_{wR} - 0.04\Delta t/3600) \quad (10.9)$$

where l_{wR} is a random number between 0 and 1, Δt the 3-D time step and the Sign function defined by (8.31).

10.1.2 Numerical procedures

horizontal displacements

Two methods of integration have been implemented. The first is a forward Euler method which is only first order in time. Letting $\tilde{u}_p = u_{ap}$, $\tilde{v}_p = v_{ap}$ in the submerged or $\tilde{u}_p = u_{ap} + u_{wd,p}$, $\tilde{v}_p = v_{ap} + v_{wd,p}$ in the floating case, where the subscript p means interpolation at the particle’s location. The new location is obtained from

$$x_p^{n+1} = x_p^n + \Delta t(u_{dp} + \tilde{u}_p), \quad y_p^{n+1} = y_p^n + \Delta t(v_{dp} + \tilde{v}_p) \quad (10.10)$$

The second method still uses a first order Euler method for the diffusive term, but now a fourth order Runge-Kutta scheme for the advective terms

$$x_p^{n+1} = x_p^n + \alpha_c \left(\Delta t(u_{dp} + \tilde{u}_p) + \frac{\Delta t}{6} (K_1^u + 2K_2^u + 2K_3^u + K_4^u) \right)$$

$$y_p^{n+1} = y_p^n + \alpha_c \left(\Delta t (v_{dp} + \tilde{v}_p) + \frac{\Delta t}{6} (K_1^v + 2K_2^v + 2K_3^v + K_4^v) \right) \quad (10.11)$$

where

$$\begin{aligned} K_1^u &= \tilde{u}_p(x^n, y^n, t^n) \\ K_1^v &= \tilde{v}_p(x^n, y^n, t^n) \\ K_2^u &= \tilde{u}_p(x^n + \frac{\Delta t}{2} K_1^u, y^n + \frac{\Delta t}{2} K_1^u, t^n + \frac{1}{2} \Delta) \\ K_2^v &= \tilde{v}_p(x^n + \frac{\Delta t}{2} K_1^v, y^n + \frac{\Delta t}{2} K_1^v, t^n + \frac{1}{2} \Delta) \\ K_3^u &= \tilde{u}_p(x^n + \frac{\Delta t}{2} K_2^u, y^n + \frac{\Delta t}{2} K_2^u, t^n + \frac{1}{2} \Delta) \\ K_3^v &= \tilde{v}_p(x^n + \frac{\Delta t}{2} K_2^v, y^n + \frac{\Delta t}{2} K_2^v, t^n + \frac{1}{2} \Delta) \\ K_4^u &= \tilde{u}_p(x^n + \Delta t K_3^u, y^n + \Delta t K_3^u, t^n + \Delta) \\ K_4^v &= \tilde{v}_p(x^n + \Delta t K_3^v, y^n + \Delta t K_3^v, t^n + \Delta) \end{aligned} \quad (10.12)$$

Despite the higher computing time, preference should be given to the Runge-Kutta method in view of its higher accuracy in time.

vertical displacements

The procedure for updating the vertical location of submerged particles is as follows

1. Determine the σ coordinate of the vertical position at the old time step

$$\sigma_p^n = \frac{z_p^n + h_p}{H_p^n} \quad (10.13)$$

2. Evaluate the advective displacement normal to the σ -coordinate surface in σ -coordinates

$$\Delta\sigma_a = \frac{\Delta t (\omega_{ap} + w_{bp})}{H_p^{n+1}} \quad (10.14)$$

where ω_{ap} is the transformed vertical current interpolated at the particle location².

²It is assumed here that the buoyant velocity acts along the normal of σ -coordinate surfaces instead of along the vertical. As explained in Section 5.1.1.2 this should be corrected by adding an additional term to the horizontal current. The effect can be considered as negligible.

3. Prevent that the particle leaves the water column

$$\Delta\sigma_a = \min(1 - \sigma_p^n, \max(\Delta\sigma_a, -\sigma_p^n)) \quad (10.15)$$

4. Define the displacement due to vertical diffusion

$$\Delta\sigma_d = \frac{\Delta w_{dp}}{H_p^{n+1}} \quad (10.16)$$

5. Determine the σ -coordinate of the vertical position at the new time step. In the absence of diffusion one has

$$\sigma_p^{n+1} = \sigma_p^n + \Delta\sigma_a \quad (10.17)$$

If vertical diffusion is enabled, a reflective boundary condition is applied to prevent that the particle leaves the water column

$$\begin{aligned} \tilde{\sigma} &= \sigma_p^n + \Delta\sigma_a + \Delta\sigma_d \\ \sigma_p^{n+1} &= \tilde{\sigma} \quad \text{if } 0 \leq \tilde{\sigma} \leq 1 \\ \sigma_p^{n+1} &= -\tilde{\sigma} \quad \text{if } \tilde{\sigma} < 0 \\ \sigma_p^{n+1} &= 2 - \tilde{\sigma} \quad \text{if } \tilde{\sigma} > 1 \end{aligned} \quad (10.18)$$

6. Transform for σ - to z -coordinate again

$$z_p^{n+1} = \sigma_p^{n+1} H_p^{n+1} - h_p \quad (10.19)$$

10.1.3 Model setup and functionalities

For each particle used in the simulation a number of attributes need or may be defined

- xpos** The particle's initial X-coordinate (Cartesian x -coordinate or longitude).
- ypos** The particle's initial Y-coordinate (Cartesian y -coordinate or latitude).
- kdistype** Selects the method (z -level, specific vertical grid cell, distance from the sea bed or sea surface) for defining the particle's initial location in the vertical.
- zpos** The particle's initial vertical location using the method selected by **kdistype**.

- starttime** The date/time when the particle is released and starts drifting. If this date is later than the initial date of the simulation, the release is postponed until the current date in the program matches the release date.
- state** Sets the type of particle drifting (surface floating or submerged).
- label** A label which can optionally be attached to the particle.
- diamconc** Particle diameter d_p needed if buoyancy effects are included. Its value depends on the particle label.
- rhoconc** Particle density ρ_p needed if buoyancy effects are included. Its value depends on the particle label.

A number of functionalities are provided and can be activated by the user as part of the setup.

- The particle module can be set up either in on-line or in off-line mode. In the latter case, COHERENS is run first without particle tracking. Hydrodynamical data (currents, ...), which are needed for a subsequent run without hydrodynamics, but with drifting enabled, are saved to a data file. In this way different particle setups and runs can be performed using the same hydrodynamical data.
- Particle tracking can be simulated either forward or backward in time. In the latter case, the particle module runs in off-line mode with a negative time step.
- The initial locations can be defined either individually for each particle or in the form of compact particle clouds each having a specific label. This allows to simulate patch-like distributions of various substances such as oil spills. Each cloud has its own label. Multiple release times can be defined for each cloud. The initial form of a cloud distribution has an quasi-ellipsoidal shape defined by the following procedure. Firstly, the coordinates x_c , y_c , z_c of the cloud centre are given for each cloud and release time. The particle locations are defined using a random distribution with an ellipsoidal shape

$$\begin{aligned}
 x_p &= x_c + \frac{1}{2}a_{Rp}L_{cl} \cos(\phi_{Rp}) \sin(\theta_{Rp}) \\
 y_p &= y_c + \frac{1}{2}a_{Rp}W_{cl} \sin(\phi_{Rp}) \sin(\theta_{Rp}) \\
 z_p &= z_c + \frac{1}{2}a_{Rp}T_{cl} \cos(\theta_{Rp})
 \end{aligned} \tag{10.20}$$

where L_{cl} , W_{cl} , T_{cl} are respectively the cloud's major axis, minor axis and thickness, a_{Rp} are random numbers between 0 and 1, ϕ_{Rp} and θ_{Rp} are random numbers between 0 and 360^0 . The horizontal distribution can be additionally rotated with a user-defined angle.

- Particle output can be set up by the user either in the form of time series of particle trajectories or as distributions on the model grid. The type of the distribution in the latter case can be selected as particle, volume or mass concentrations.

Switches

The following control switches can be defined for the particle module

iopt_part_model Selects the mode for particle tracking.

- 0: Particle module is disabled. Default.
- 1: Particle module runs on-line. COHERENS runs in serial mode.
- 2: Particle module runs on-line. COHERENS runs in parallel mode.
- 3: Particle module runs off-line in forward mode and reads data from an external file obtained from a previous (non)-COHERENS run.
- 4: Particle module runs offline in backward mode and reads data from an external file obtained from a previous (non)-COHERENS run.

iopt_part_write Disables/enables writing hydrodynamic data from a COHERENS run (without the particle module) to an output file which can be used for a subsequent off-line run with the particle module.

- 0: Disabled. Default
- 1: Enabled.

iopt_part_hadv Type of time integration scheme for horizontal advection.

- 0: Horizontal advection disabled.
- 1: First order forward Euler.
- 2: Fourth order Runge-Kutta. Default.

iopt_part_hdif Disables/enables horizontal diffusion.

	0: Horizontal diffusion disabled. Default.
	1: Horizontal diffusion enabled.
iopt_part_vadv	Disables/enables vertical advection.
	0: Vertical advection disabled. Default
	1: Vertical advection enabled.
iopt_part_vdif	Disables/enables vertical diffusion.
	0: Vertical diffusion disabled. Default.
	1: Using uniform diffusion coefficient.
	2: Using COHERENS diffusion coefficient.
iopt_part_wind	Disables/enables drifting of floating particles by the wind.
	0: Wind drifting enabled. Default.
	1: Wind drifting disabled.
iopt_part_leeway	Disables/enables the Leeway effect on surface drifting.
	0: Leeway effect disabled. Default.
	1: Leeway effect enabled.
iopt_part_dens	Type of density field for calculating the buoyant velocity.
	0: Buoyancy effects disabled. Default.
	1: Using reference value ρ_0 for the water density.
	2: Using the density field obtained from the equation of state.
iopt_part_cloud	Disables/enables cloud distributions.
	0: Cloud distributions disabled. Default.
	1: Cloud distributions enabled. Particle tracking outside the clouds are prohibited.
iopt_part_conc	Converts particle distributions to continuous concentrations over the model grid. The option is used for user-defined output only.
	0: Disabled. Default.
	1: As number concentration [number/m ³].
	2: As volume concentration [m ³ /m ³]. The particle diameter diamconc must be provided by the user.

- 3: As mass concentration [kg/m³]. The particle diameter **diamconc** and density **rhoconc** must be provided by the user.

iopt_part_out Disables/enables trajectory output.

0: Trajectory output disabled. Default.

1: Trajectory output enabled.

Chapter 11

Structures, discharges and inundation

This chapter deals with structures, discharges and with the drying-wetting and inundation schemes implemented in COHERENS. Four different structures are defined: dry cells, thin dams, weirs and barriers. The methods used for weirs and barriers have been obtained after a review of the technical literature and are similar to those used in similar models, such as [Delft3D \(2009\)](#), [SIMONA \(2009\)](#). The review showed a resemblance of the state of art in numerical modeling of hydraulic structures.

11.1 Dry cells

Land or permanently dry cells are defined at cells that are taken as permanently dry during the whole simulation, irrespective of the local water depth. They can be defined in two ways:

- The mean water depth is set to the flag `depmean_flag`.
- The user provides the dry cells by providing their grid locations at the cell centers. Flagging of the cells will be made by the code itself.

No flow exchange nor transport of scalars are allowed between dry cells and the adjacent grid cells. An example of a dry cell is given in Figure 11.1.

This functionality is used to define areas in the computational domain that will be permanently dry and excluded from computations. The goal of this functionality is to schematise or simulate the effect of some types of hydraulic structures which cannot be submerged. Dry cells can be defined as “isolated” units or groups representing different structures in a study area. An example of groups of dry cells is illustrated in Figure 11.2.

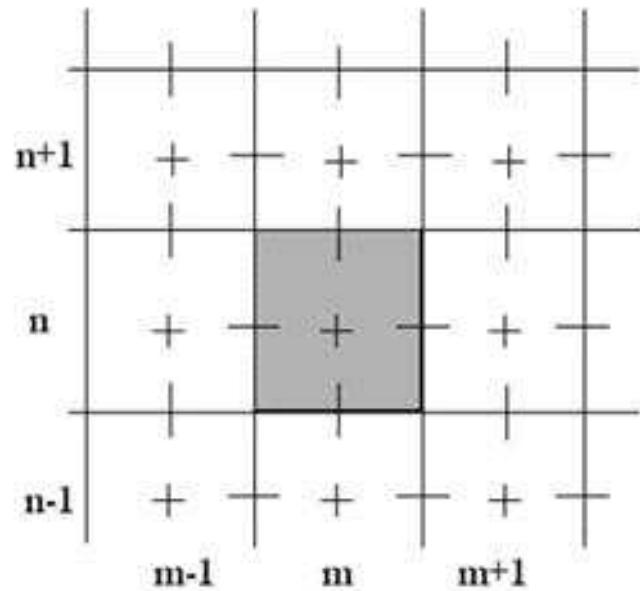


Figure 11.1: Dry cell defined at grid location (m,n) .

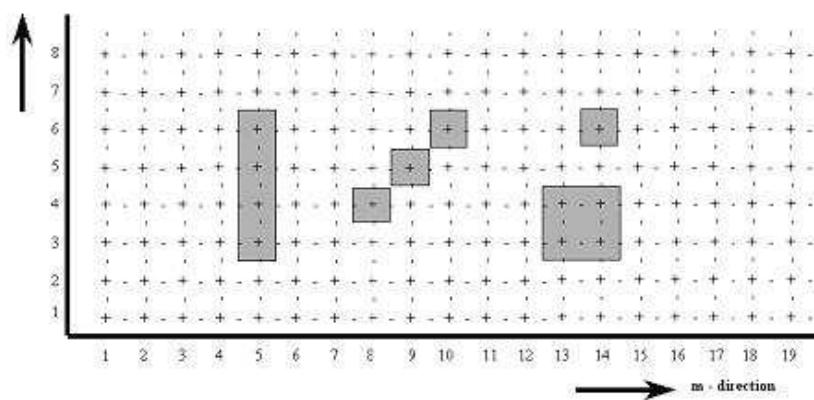


Figure 11.2: Groups of dry cells defined in a computational domain.

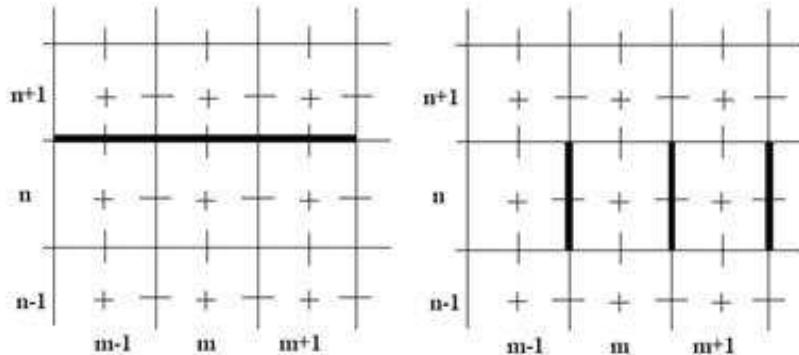


Figure 11.3: Definition of thin dams at V- and U-nodes.

11.2 Thin dams

Thin dams are defined as infinitely thin vertical walls. They are located at velocity nodes and prohibit flow exchange and fluxes of scalar quantities across the two adjacent computational grid cells without reducing the total wet surface and the volume of the model. This functionality can be used both in 2-D or 3-D mode applications. Note that, in 3-D mode, thin dams will block the flow and scalar exchange at all vertical layers. Particly blocked vertical walls are discussed in Section 11.3 below.

The purpose of a thin dam is to represent small obstacles (e.g. breakwaters, dams) in the model which have sub-grid dimensions, but still large enough so that they have an impact on the local flow pattern. They can be defined at the velocity (U- or V-)nodes, either as single elements or as a line of thin dams. An example is depicted in Figure 11.3.

The model defines the bathymetry at the centre of the grid cells (C-nodes). In this way it is possible to apply a thin dam while having a different bathymetry on both sides of the dam.

Moreover, some restrictions regarding application of thin dams are identified:

- Thin dams can only be specified along lines parallel to one of the numerical grid axes.
- No thin dams can (obviously) be defined at open boundaries or at the edges of the computational grid but thin dams perpendicular to open boundaries are allowed.

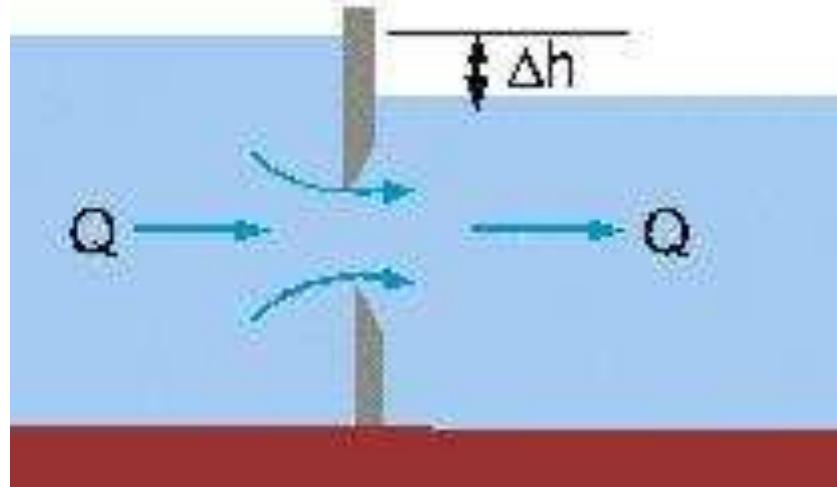


Figure 11.4: Scheme of a barrier (submerged structure).

11.3 Weirs and barriers

The aim of this model unit is twofold, the first aim is the schematization of structures that can be submerged like “levees” or “dikes”. These structures are denoted as “weirs”. Weirs are fixed non-movable constructions that generate energy losses to the flow due to contraction and expansion of the flow (see Figures 11.7 and 11.8). The second aim is a particular case of “weirs”, further denoted as “barriers”, oriented to schematize structures that can be submerged and present an opening close to the bottom, denoted as orifices (e.g. current deflecting walls), generating energy losses due to contraction and expansion of the flow (see Figure 11.4). Both energy losses are calculated using similar equations.

Upstream of the structure the flow is accelerated due to contraction and downstream the flow is decelerated due to expansion. This expansion introduces an important energy loss due to turbulent friction, and this loss needs to be calculated. This energy loss is added as an opposing force in the momentum equation by adding an extra term to the momentum equation.

The energy loss generated by the structure is not computed directly by the diffusive terms in the momentum equations, but is parameterized and added in the momentum equations by adding an extra term sink term on the right hand side of the 2-D and 3-D momentum equations, given by

$$\mathcal{S}_{wb}(U) = -g \frac{\Delta \eta}{\Delta x} \frac{U}{|\bar{u}|}, \quad \mathcal{S}_{wb}(V) = -g \frac{\Delta \eta}{\Delta y} \frac{V}{|\bar{v}|} \quad (11.1)$$

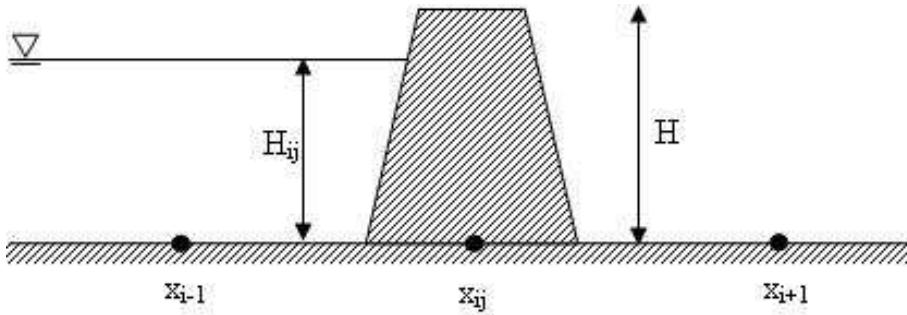


Figure 11.5: Blocking of flow when the water depth is less than the crest level.

in the 2-D case, and

$$\mathcal{S}_{wb}(u) = -g \frac{\Delta\eta}{\Delta x} \frac{u}{|\bar{u}|}, \quad \mathcal{S}_{wb}(v) = -g \frac{\Delta\eta}{\Delta y} \frac{v}{|\bar{v}|} \quad (11.2)$$

in the 3-D momentum equations, where η is the so-called “energy level” defined as $\eta = E/g$ so that the energy loss is given by $\Delta E = g\Delta\eta$.

11.3.1 Weirs

This type of structure is similar to a thin dam, except that a weir can be inundated, in which case an energy loss is generated. This structure will work as a thin dam in cases where the total water depth upstream of the structure is less than the crest level of the structure. In this case a blocking of flow exchange is imposed by the module (see Figure 11.5). The blocking process is present in 2-D as well as in 3-D mode simulations. Depending on the simulation mode, the blocking works differently. In the 2-D case, there is no necessity to define additional parameters since the complete water column is blocked. However for 3-D simulations, some additional considerations should be taken.

For the 2-D mode, the implementation of the model unit follows a certain criteria. In case of a U-node weir perpendicular to the X-axis, a “blocking” of the flow is imposed when the total water depth at the upstream side is less than the height of the crest of the weir, whereas a loss of energy is imposed when the total water depth is greater than the height of the crest. Therefore, the following scheme is used, where the verification of the water depth is performed at the C-nodes upstream (H_{ij} if $U > 0$ or $H_{i+1,j}$ if $U < 0$) and downstream ($H_{i+1,j}$ if $U > 0$ or $H_{i+1,j}$ if $U < 0$) of the location of the weir:

$$H_{i,j} > H_{weir} \text{ and } H_{i+1,j} > H_{weir} \rightarrow \text{loss of energy}$$

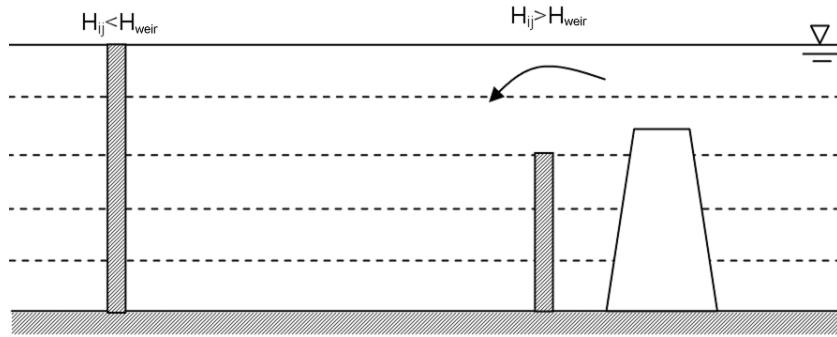


Figure 11.6: Scheme of a “weir” in 3-D mode for non-submerged and submerged conditions (compared with the height of a structure).

$$H_{i,j} > H_{weir} \text{ and } H_{i+1,j} < H_{weir} \rightarrow \text{loss of energy}$$

$$H_{i,j} < H_{weir} \text{ and } H_{i+1,j} > H_{weir} \rightarrow \text{loss of energy}$$

$$H_{i,j} < H_{weir} \text{ and } H_{i+1,j} < H_{weir} \rightarrow \text{blocking}$$

For 3-D simulations, it is necessary to make some additional considerations. When the total water depth is less than the height of the crest of the structure at the adjacent C-nodes, a “blocking” will be imposed. Then, all the layers will be blocked for flow exchange (see Figure 11.6 at the left side). When the total water depth is greater than the crest of the structure, a partial blocking process is imposed. A limited number of vertical grid layers will be blocked allowing flow over the structure. However, it is necessary to consider that the height that corresponds to the number of blocked layers will not be equal to the height of the structure (see Figure 11.6 at the right side). Hence, it is assumed that the layer that corresponds to the level of the crest will be blocked if less than 50% of this layer is above the crest and taken as open otherwise.

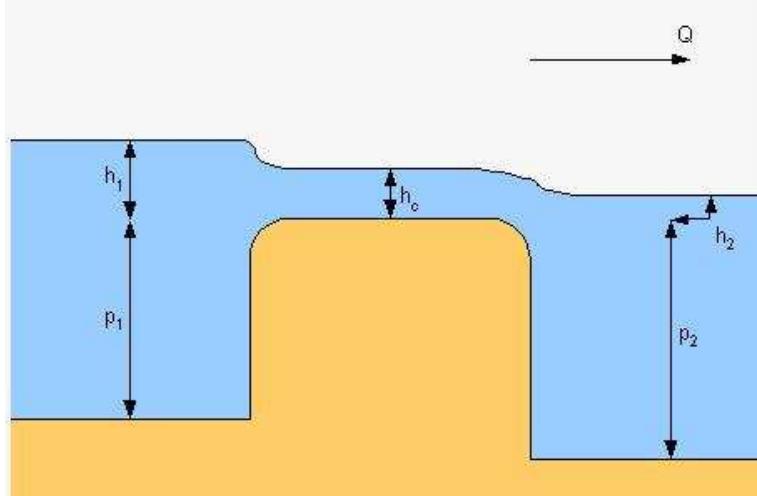
The loss of energy is calculated from the discharge rate defined for a weir. Two conditions are defined: “free flow” where the water depth downstream of the structure has no influence on the structure (see Figure 11.7), and “submerged flow”, where the downstream water depth has an influence on the structure (see Figure 11.8). The conditions for free and submerged flow are determined from the value of the modular limit m , see Table 11.1.

11.3.1.1 Free flow

In the case of a free flow, the flow rate is defined by:

Table 11.1: Conditions for the determination of free or submerged

Flow condition	$U > 0$	$U < 0$
Free flow	$H_i/H_{i-1} \leq m$	$H_{i-1}/H_i \leq m$
Submerged flow	$H_i/H_{i-1} > m$	$H_{i-1}/H_i > m$

**Figure 11.7:** Scheme of free flow over a weir.

$$Q = C_{st} b (H_1)^{3/2} \quad (11.3)$$

where Q denotes the discharge, b the width of the approaching channel, H_1 the water depth upstream of the structure (related to the energy head) and C_{st} is a coefficient that includes the shape of the approach channel and the geometry. Generally this coefficient is considered as a calibration coefficient and should be provided by the user. This coefficient was originally designed for open channel flow. Technical literature suggests different values for this coefficient (mostly around unity). However, preliminary tests of flow under tidal conditions showed that lower values are required to guarantee numerical stability. Hence, it is suggested to use values between 0 and 1.

For a weir defined at a U-node, one has $Q = bU$ where U is the depth-integrated current. Equation (11.3) can then be solved for H_1 :

$$H_1 = \left(\frac{U}{C_{st}} \right)^{2/3} \quad (11.4)$$

In free flows, it is assumed that the loss of energy equals the difference of water depths upstream of the weir so that

$$\Delta\eta = H_{up} - (h_{cr} + H_1) \quad (11.5)$$

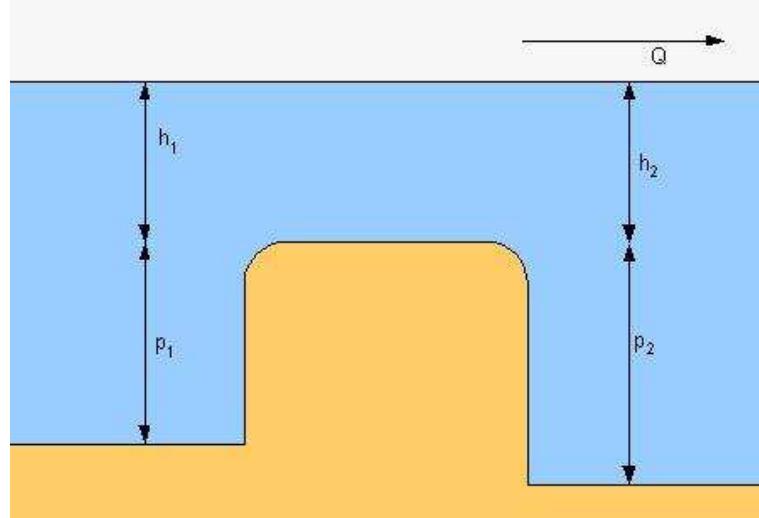


Figure 11.8: Scheme of submerged flow over a weir.

where H_{up} is the (total) water depth upstream of the weir and h_{cr} the height of the weir crest.

11.3.1.2 Submerged flow

For submerged flow the water depth downstream of the weir influences the water level upstream (see Figure 11.8). Its flow rate is defined by:

$$Q = C_{st} b H_1 \left(\frac{H_1 - H_2}{1 - m} \right)^{1/2} \quad (11.6)$$

where Q denotes the discharge, b the width of the approaching channel, H_1 , H_2 are the water depths (relative to the energy head) upstream, respectively downstream of the structure, C_{st} is a coefficient that includes the shape of the approach channel and the geometry and m denotes the modular limit (Bos, 1998). The modular limit, which is a user-defined parameter, relates the downstream energy level to the energy level over the weir crest, presenting values between 0.6 and 1.0.

The energy loss is calculated differently in submerged flows, since the downstream conditions are affecting the flow over the weir, and equals the difference of water depths (relative to the energy head) upstream and downstream the structure

$$\Delta\eta = H_1 - H_2 \quad (11.7)$$

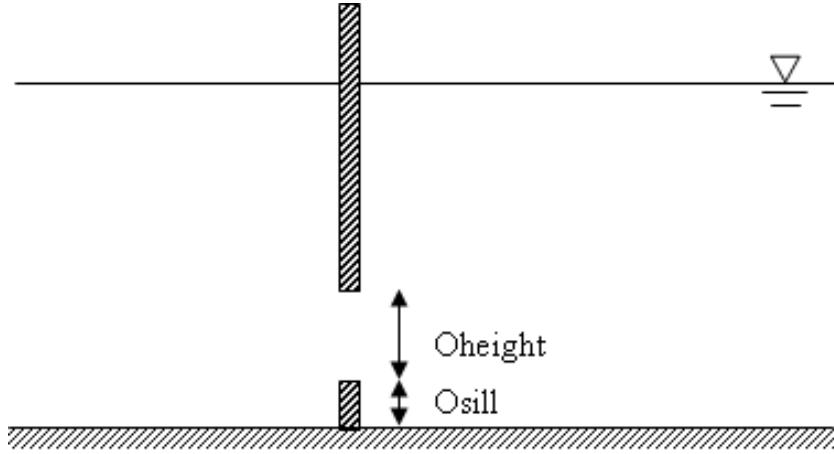


Figure 11.9: Sketch of a barrier.

Eliminating H_2 from (11.6) and (11.7) one has

$$\Delta\eta = (1 - m) \left(\frac{Q}{C_{st} b H_1} \right)^2 \quad (11.8)$$

Since $Q = bU$ for a weir at a U-node, this gives

$$\Delta\eta = (1 - m) \left(\frac{U}{C_{st} H_1} \right)^2 \quad (11.9)$$

In summary, the following parameters are needed to define a weir: weir crest height h_{cr} , discharge coefficient C_{st} and modular limit m .

11.3.2 Barriers

For structures of the barrier type, it is considered that there is an opening close to the bottom, where users can define the width of the opening and the height of the sill above the sea bed (see Figure 11.9). Since this structure defines a partial blocking of vertical layers, it is more oriented to be applied in 3-D mode simulations. The blocking process is similar to the process described for weirs. This functionality can alternate between weirs and barriers by defining the dimension of the opening close to the bottom. In case the opening is closed, a “weir” is defined. If the opening is different from zero, a “barrier” is defined.

The loss of energy is generated due to the contraction and expansion of the flow using the definition of flow in a submerged orifice (also denoted as a “current deflecting wall”), a schematic presentation of this type of structure is shown in Figure 11.10.

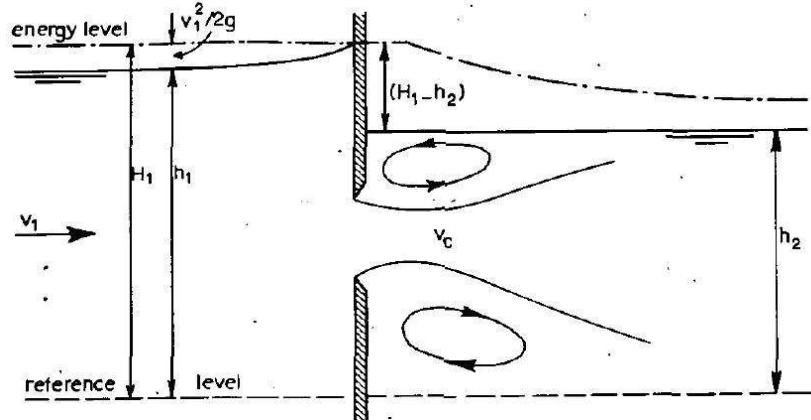


Figure 11.10: Scheme of a submerged orifice flow.

The flow rate is defined by

$$Q = C_e A (2g (H_1 - H_2))^{1/2} \quad (11.10)$$

where Q denotes the discharge, A the area of the opening, H_1, H_2 the water depths (relative to the energy head) upstream, respectively downstream of the structure and C_e is a coefficient, defined by the user, that includes the shape of the approach channel and the geometry. Similar to the C_{st} coefficient in the weir case, the C_e coefficient should take values between 0 and 1.

The calculation of the energy loss is similar to the weir case, since the downstream conditions are affecting the flow over the structure so that the loss equals the difference of the water depths upstream and downstream of the barrier

$$\Delta\eta = H_1 - H_2 \quad (11.11)$$

From (11.10) one obtains

$$\Delta\eta = \frac{1}{2g} \left(\frac{Q}{C_e A} \right)^2 \quad (11.12)$$

or, using $Q = bU$ at U-nodes,

$$\Delta\eta = \frac{1}{2g} \left(\frac{U}{C_e O_w} \right)^2 \quad (11.13)$$

where O_w is the width of the opening.

In summary, the following parameters are needed to define a barrier: discharge coefficient C_e , orifice width O_w and height of the sill above the sea bed O_h .

11.3.3 Numerical implementation

The sink terms in the momentum equations are discretised quasi-implicitly in time using the [Patankar \(1980\)](#) scheme, described in Section [12.5](#). For the sink terms at U-nodes, one then obtains from [\(11.1\)–\(11.2\)](#):

$$\mathcal{S}_{wb}(U) = -g \frac{\Delta\eta^*}{\Delta x} \frac{U^{n+1}}{|\bar{u}^n|}, \quad \mathcal{S}_{wb}(u) = -g \frac{\Delta\eta^*}{\Delta y} \frac{u^{n+1}}{|\bar{u}^n|} \quad (11.14)$$

with similar expressions at V-nodes.

To prevent oscillations, a relaxation parameter θ_{wrl} has been introduced for the energy loss $\Delta\eta$ and which can be set to a value between 0 and 1 (the default value is 1). The relaxation procedure is defined as follows

$$\Delta\eta^* = (1 - \theta_{wrl})\Delta\eta^{n-1} + \theta_{wrl}\Delta\eta^n \quad (11.15)$$

Water depths at weirs are determined using an upwind scheme, as described in [SIMONA \(2009\)](#). At U-nodes, this gives

$$\begin{aligned} H_{ij}^u &= h_{ij}^u + \zeta_{i-1,j} & \text{if } U_{ij} > 0 \\ H_{ij}^u &= h_{ij}^u + \zeta_{ij} & \text{if } U_{ij} < 0 \\ H_{ij}^u &= \max(\zeta_{i-1,j}, \zeta_{ij}) & \text{if } U_{ij} = 0 \end{aligned} \quad (11.16)$$

where h_{ij} is the mean water depth above the crest. In case of a barrier, the following value is added

$$\min(H_{ij} - h_{cr}, O_h + O_w) - O_h \quad (11.17)$$

To prevent negative depths, a minimum water depth is taken, given by the same parameter d_{min} used in the inundation schemes.

11.4 Discharges

Discharges are represented as sources or sinks in the continuity, momentum or scalar equations supplied at specified (fixed or moving) locations at the surface, bottom or within the water column (e.g. discharge structures, pumping stations, discharge from moving ships ...) by adding water to the water column with a specified salinity, temperature, sediment concentration, The discharge may or may not have a preferential direction. Figure [11.11](#) illustrates a discharge at the bottom of the sea defined in a 3-D simulation.

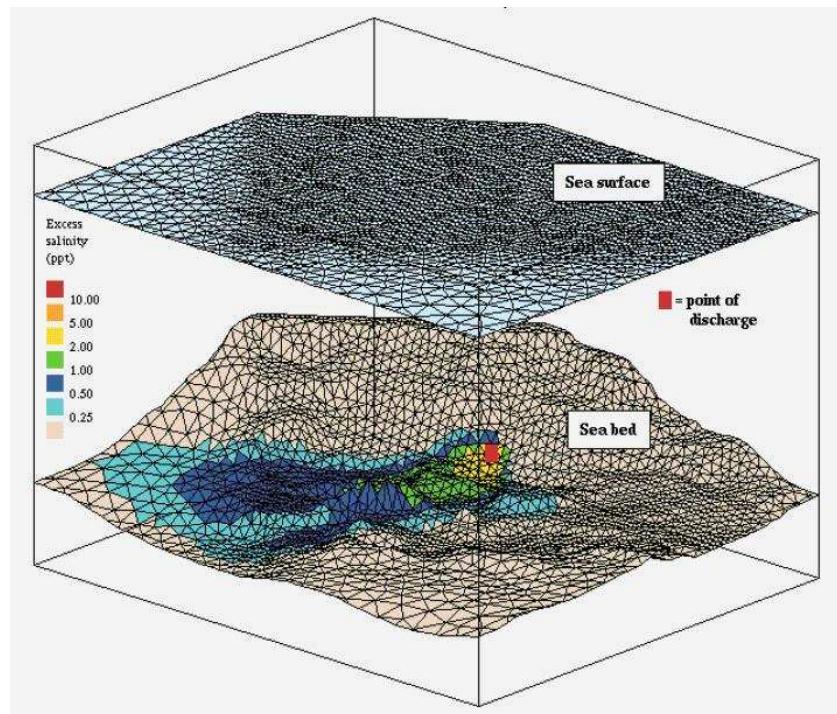


Figure 11.11: Schematization of a discharge in a 3-D simulation

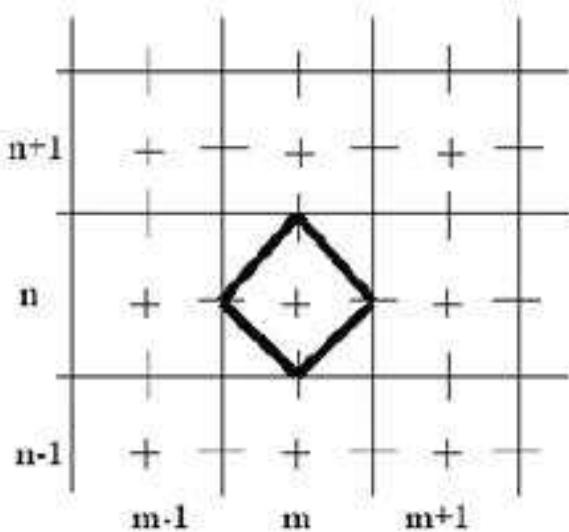


Figure 11.12: Location of a discharge point in a grid cell.

Discharges can be applied in 2-D as well 3-D mode. Discharge points are defined at C-nodes. In the 3-D case the discharge location can be specified at a vertical grid point or distributed uniformly over all layers. Figure 11.12 depicts the definition of a discharge point in a grid cell.

Considering the direction of the discharge, two types can be defined:

1. Normal: the discharge is defined without a specific discharge direction.
2. Momentum: the momentum of the discharge is taken into account, both discharge area and direction have to be provided.

As a result, depending on the simulation mode and the type of discharge, six types may be defined in COHERENS

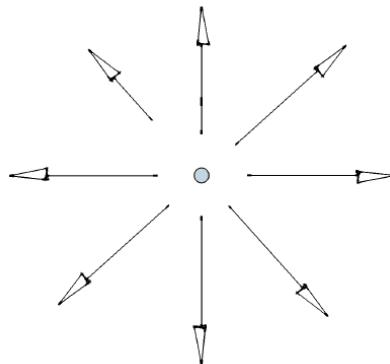
1. 2-D normal discharge (Figure 11.13)
2. 2-D momentum discharge (Figure 11.14)
3. 3-D normal discharge distributed uniformly over all layers (Figure 11.15)
4. 3-D normal discharge defined at a specific height (Figure 11.16)
5. 3-D momentum discharge distributed uniformly over all layers (Figure 11.17)
6. 3-D momentum discharge defined at a specific height (Figure 11.18).

The implementation of discharges requires

- a modification of the continuity equation by the addition of a source term
- an additional source or sink term in the 2-D and 3-D momentum equation, in case of a directional (momentum) discharge
- an (eventually) additional source term in the scalar transport equations. If no scalar discharge is provided by the user, the added water volume is assumed to have the same temperature, salinity or contaminant (sediment) concentration as the one at the discharge location.

The discharge locations are defined at specific heights or uniformly over the vertical and can be placed in several locations allowing the users to use several options for modelling of discharges. The following vertical locations can be used:

1. discharge, uniformly distributed over the vertical

**Figure 11.13:** 2-D normal discharge**Figure 11.14:** 2-D momentum discharge

2. discharge located at the bottom
3. discharge located at the surface
4. discharge located at a given distance from the sea bed
5. discharge located at a given distance below the surface.

The locations are either fixed or moving in time.

11.4.1 Continuity equation

A source term term is added to the right hand side of the 2-D continuity equation (5.32), given by

$$q_{dis} = \frac{Q_{dis}}{h_1 h_2} \quad (11.18)$$

where Q_{dis} represents the volume discharge in m^3/s and q_{dis} denotes the “discharge speed” (m/s).

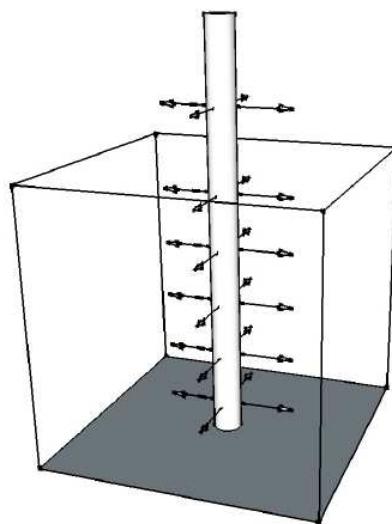


Figure 11.15: 3-D normal discharge distributed uniformly over all layers

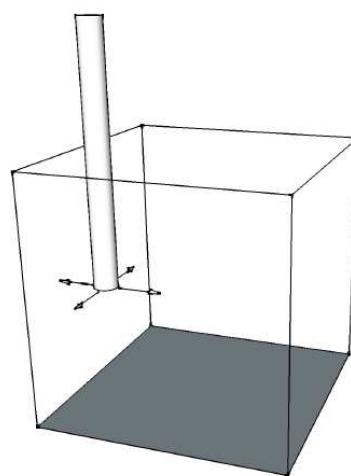


Figure 11.16: 3-D normal discharge defined at a specific height

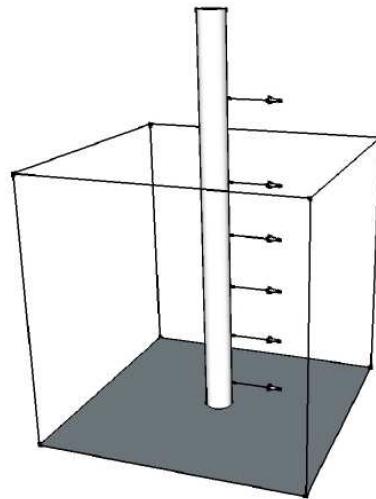


Figure 11.17: 3-D momentum discharge distributed uniformly over all layers

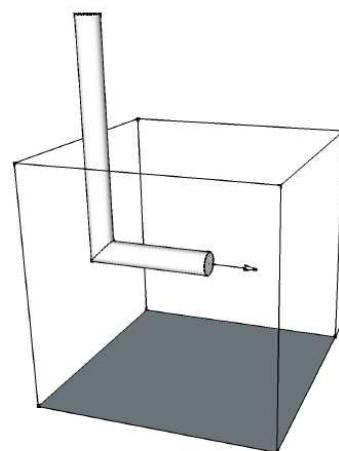


Figure 11.18: 3-D momentum discharge defined at a specific height

11.4.2 Momentum equations

A source (or sink) term is added to the right hand side of the momentum equations in case a directional discharge is specified. This is given by

$$\mathcal{Q}_m(U) = q_{dis} \left(\frac{Q_{dis}}{A_{dis}} \cos \theta_{dis} - \bar{u} \right), \quad \mathcal{Q}_m(V) = q_{dis} \left(\frac{Q_{dis}}{A_{dis}} \sin \theta_{dis} - \bar{v} \right) \quad (11.19)$$

in the 2-D momentum equation (5.47–5.48), and

$$\mathcal{Q}_m(u) = \frac{q_{dis}}{\Delta_{dis}} \left(\frac{Q_{dis}}{A_{dis}} \cos \theta_{dis} - u \right), \quad \mathcal{Q}_m(v) = \frac{q_{dis}}{\Delta_{dis}} \left(\frac{Q_{dis}}{A_{dis}} \sin \theta_{dis} - v \right) \quad (11.20)$$

in the 3-D momentum equations (5.19–5.20) where θ_{dis} is the discharge direction with respect to the reference axis (Eastward direction in the spherical case), A_{dis} the discharge area (in m^2) and Δ_{dis} the vertical distance over which the discharge takes place, given by H for a uniform or $h_3 = \Delta z$ for a non-uniform discharge.

11.4.2.1 Discharge of scalars

Two additional types of discharge can be defined for scalars. In the case of a “wet” discharge, water with a specified temperature, salinity or other concentration is added to the specified cells. The following source term is added to the right hand side of the scalar transport equation

$$\mathcal{P}(\psi) = \frac{q_{dis}}{\Delta_{dis}} \psi_{dis} \quad (11.21)$$

where ψ_{dis} is the user-specified temperature, salinity or concentration of the discharged water.

In the case of a “dry” discharge, no water is added. The source term takes the form

$$\mathcal{P}(\psi) = \frac{\psi_{dis}}{h_1 h_2 \Delta_{dis}} \quad (11.22)$$

where ψ_{dis} now represents the amount of the concentration added over the whole grid cell per unit of time.

Note that in the current version of COHERENS only wet discharges are permitted.

11.4.2.2 Numerical implementation

The numerical discretisation of the source terms in the continuity and scalar equations is straightforward. The discretisation of the discharge term in the

momentum equations requires some care since the discharge is defined at C-nodes whereas the components of the depth integrated and 3-D current are taken at the velocity (U- or V-) nodes. The discharge term is therefore taken as an average between adjacent velocity nodes. For the U - and u -momentum equations this gives

$$\begin{aligned}\mathcal{Q}_m(U)_{ij} &= q_{dis} \left(\frac{Q_{dis}}{A_{dis}} \cos \theta_{dis} - (m_{i-1,j} + m_{ij}) \bar{u}_{ij} \right) \\ \mathcal{Q}_m(u)_{ijk} &= \frac{q_{dis}}{\Delta_{dis;ijk}} \left(\frac{Q_{dis}}{A_{dis}} \cos \theta_{dis} - (m_{i-1,j} + m_{ij}) u_{ijk} \right) \quad (11.23)\end{aligned}$$

where $\Delta_{dis;ijk}$ equals H_{ij} for uniform and $h_{3;ijk}^u$ for non-uniform discharge. The weight factor is defined as follows

$$\begin{aligned}m_{ij} &= \frac{1}{2} \quad \text{if } (i-1,j) \text{ and } (i,j) \text{ are interior wet velocity points} \\ m_{ij} &= 1 \quad \text{if one of the velocity nodes } (i-1,j) \text{ and } (i,j) \text{ is interior wet and the other not} \\ m_{ij} &= 0 \quad \text{if none of the velocity nodes } (i-1,j) \text{ and } (i,j) \text{ is interior wet} \quad (11.24)\end{aligned}$$

The formulations at V-nodes are similar.

11.5 Drying/wetting and inundation schemes

The objective is to simulate areas where inundation and drying/wetting processes occur. This includes the simulation of land areas which are flooded by storm surges, flow over obstacles or tidal flats where drying or wetting takes place during ebb or flood tide. Grid cells of the computational domain will become “dry” or “wet” depending on the value of the total water depth.

Inundation schemes are focused on simulating dynamic processes. “Drying and wetting” refers to a formalism used to define “wet” and “dry” areas in the computational domain. **COHERENS** uses this formalism to define the coastal boundaries. The definition is performed at the initialisation stage and only once. In this way a distinction is made of three type of grid cells

- Permanent land cells which are always dry and where no calculation is performed. They are defined by the user as cells where the mean water depth has a flagged value.
- Cells which are ‘either temporarily dry (non-active) or temporarily wet when they get inundated by the rising water. Calculations in dry cells are disabled until they are inundated from a neighbouring cell. The

total water depth remains limited from below by d_{min} so that at least a small amount of water is present. In this way, dynamic coastal boundaries can be simulated. Note that land topography is represented by negative bathymetric values (see below) so that there exists no lower limit for the mean water depth except for values equal to the data flag.

- Cells which are permanently wet and where all calculations are enabled.

The drying and flooding algorithm implemented in COHERENS consists of the following steps which are consecutively executed by the program.

1. At the initial time, the total water depth at C-nodes is bounded below by a minimum water depth to prevent negative water depths

$$H_{ij}^c = \max(H_{ij}^c, d_{min}) \quad (11.25)$$

so that a (spurious) small amount of water remains if a cell becomes dry. Equation (11.25) implies the following condition for the surface height

$$\zeta_{ij} \geq d_{min} - h_{ij} \quad (11.26)$$

This means that if the value of ζ_{ij} , obtained from the continuity equation, falls below the minimum value (11.26), a (small) amount of water is added to the water column in violation of mass conservation. No correction is applied in the current version of COHERENS, but the spuriously added water is stored in the program variable $\delta_e H$ which measures the accumulated error in water depth and is increased by the amount $d_{min} - H$ at each time step when H drops below d_{min} . By its definition $\delta_e H$ is non-negative and cannot decrease in time.

2. In the absence of a drying mechanism the total water depth at the velocity nodes is calculated as the (weighted) average of the depths at the surrounding C-nodes. Experiments showed that this method produces large horizontal gradients for the vertical grid spacings for water depths close to the minimum value, and, hence, unrealistically high vertical current magnitudes after substitution in the baroclinic continuity equation (5.34). To smoothen the solution the depth at the velocity is taken as the minimum of the surrounding C-node values, i.e.

$$\begin{aligned} H_{ij}^u &= \min(H_{i-1,j}^c, H_{ij}^c) & \text{if} & \quad \min(H_{i-1,j}^c, H_{ij}^c) < d_{crit} \\ H_{ij}^v &= \min(H_{i,j-1}^c, H_{ij}^c) & \text{if} & \quad \min(H_{i,j-1}^c, H_{ij}^c) < d_{crit} \end{aligned} \quad (11.27)$$

where d_{crit} is defined below. Disadvantage is an unphysical retardation of the flow through the velocity interfaces.

3. Following (Burchard & Bolding, 2002), the advective, horizontal diffusion, Coriolis, curvature and baroclinic pressure gradient terms in the 2-D and 3-D momentum equations are multiplied by a “drying” factor α . For example, the u -equation becomes

$$\begin{aligned} \frac{1}{h_3} \frac{\partial}{\partial t} (h_3 u) + \alpha & \left[\mathcal{A}_{h1}(u) + \mathcal{A}_{h2}(u) + \mathcal{A}_v(u) \right. \\ & + \left. \frac{v}{h_1 h_2} \left(u \frac{\partial h_1}{\partial \xi_2} - v \frac{\partial h_2}{\partial \xi_1} \right) - 2\Omega v \sin \phi \right] \\ & = -\frac{g}{h_1} \frac{\partial \zeta}{\partial \xi_1} - \frac{1}{\rho_0 h_1} \frac{\partial P_a}{\partial \xi_1} + \alpha F_1^b + F_1^t + \mathcal{D}_{mv}(u) \\ & + \alpha \left(\mathcal{D}_{mh1}(\tau_{11}) + \mathcal{D}_{mh2}(\tau_{12}) \right) \end{aligned} \quad (11.28)$$

where α decreases from 1 when the total water depth becomes lower than a critical value d_{cr} and becomes 0 when H reaches the minimum value d_{min} .

$$\alpha = 1 \quad \text{if} \quad H > d_{crit} \quad (11.29a)$$

$$\alpha = \left(\frac{H - d_{min}}{d_{crit} - d_{min}} \right)^{n_{wd}} \quad \text{if} \quad d_{min} \leq H \leq d_{crit} \quad (11.29b)$$

$$\alpha = 0 \quad \text{if} \quad H < d_{min} \quad (11.29c)$$

An alternative formulation is available representing the asymptotic case $n_{wd} = \infty$ so that (11.29b) is replaced by

$$\alpha = 0 \quad \text{if} \quad d_{min} \leq H \leq d_{crit} \quad (11.30)$$

Using the α -factor, the momentum equations reduce to a balance between the time derivative, surface slope and vertical diffusion (bottom friction) terms when $H \rightarrow d_{min}$. The formulation (in the non-asymptotic case) provides a continuous transition from a wet to a dry condition in contrast to schemes, discussed below, using a “mask” function which sets cells to a dry or wet state depending on some drying criterium.

4. The following mask criterion is applied for all grid cells which are not permanently dry

$$\max(H_{ij}, H_{i-1,j}, H_{i+1,j}, H_{i,j-1}, H_{i,j+1}) < d_{th} \quad (11.31)$$

When the criterion evaluates to .TRUE. at a particular grid cell, the cell becomes dry and no hydrodynamic calculations will be performed within the cell. The following actions are taken in addition

- Currents will be set to zero at the four surrounding velocity nodes.
- The surface elevation and scalar quantities obtained from a transport equation retain their value from the last previous time when the cell was wet. Note that all cells, which are not permanently dry, are considered as wet at the initial time.
- Dry cells are excluded from the interpolation of model variables on the model grid (see Section 15.2.2).

If the criterion evaluates to `.FALSE.`, the cell becomes wet (again). The cell is reactivated again and water is allowed to enter through the side faces.

5. Currents and surface elevations are updated, after which step 1 is repeated. If the water depth at a wet cell equals the minimum value d_{min} , the cell can be considered as “virtually” empty. To prevent the outflow of non-existing water, the surface slopes are set to zero at the surrounding velocity nodes with an outflow condition.

Equations (5.322)–(5.323) show that the bottom drag coefficient and hence the bottom stress increase at small water depths. As mentioned in Section 5.9.2, the ratio of the reference depth to the roughness length in equation (5.322) is limited by a minimum value to prevent unrealistically large values for C_{db} when the water depth is close to the minimum value. Note that this effect is only present when the bottom drag coefficient is calculated as function of a roughness length.

6. In addition to the already existing dry cells, the cells with a water depth equal to the minimum value d_{min} are declared as dry. Scalar quantities such as temperature, salinity, sediment concentrations, ... are updated in time. No updates are made at dry cells (including those with a minimum water depth). If the cell is dry, the scalar quantities obtained from a transport equation retain their value from the last previous time when the cell was wet. Note that all cells, which are not permanently dry, are considered as wet at the initial time.

The following remarks are given for the user:

- The inundation algorithms can be used for tidal flats, i.e. areas below mean sea level becomes dry/wet during low/high tide, as well as for land areas above mean sea level. In the latter case, a topography of the land, located above the mean sea level, has to be supplied. Land topography is represented in the code by cells with a negative mean

water depth, sea areas by positive mean water depths. A data flag, given by the model parameter `depmean_flag`, has to be assigned to the locations, considered as permanent land.

- The minimum, threshold and critical water depth must satisfy the following conditions

$$d_{min} < d_{th} \leq d_{crit} \quad (11.32)$$

Default values are $d_{min}=0.02$, $d_{th}=0.1$, $d_{crit}=0.1$

- Although not recommended, the use of the α -factor and the mask criterion can be disabled by the user.

Switches

The following switches are defined for the schemes used in this section

<code>iopf fld</code>	Disables/enables the use of the drying/wetting scheme.
0:	Disabled.
1:	Enabled.
<code>iopf fld alpha</code>	Type of formulation for the α -factor used in the drying/wetting algorithm for momentum advection.
0:	No reduction applied ($\alpha = 1$).
1:	Power law given by (11.29a)–(11.29a).
2:	Asymptotic formulation given by (11.30).
<code>iopf fld crit</code>	Disables/enables the use of mask criterion.
0:	Disabled.
1:	Enabled.

Chapter 12

Numerical methods

12.1 Introduction

The numerical methods described in this chapter are based to a large extent upon previous work described in the COHERENS V1 manual ([Luyten *et al.*, 1999](#)).

Two options are available to solve the hydrodynamic equations. The original implementation in COHERENS used the mode-splitting technique as in the model of [Blumberg & Mellor \(1987\)](#) to solve the momentum equations. This method consists in solving the depth-integrated momentum and continuity equations for the “external” or barotropic mode with a small time step to satisfy the stringent CFL stability criterium for surface gravity waves, and the 3-D momentum and scalar transport equations for the “internal” or “baroclinic” mode with a larger time step. A “predictor” and a “corrector” step are applied for the horizontal momentum equations to satisfy the basic requirement that the depth-integrated currents obtained from the the 2-D and 3-D mode equations, have identical values.

Recently, the possibility to solve the momentum equations semi-implicitly as in the model of [Chen \(2003\)](#), based on the original work of [Casulli & Cheng \(1992\)](#) was implemented. With this method, there is no longer need to solve the depth-integrated momentum equations. The stringent CFL stability criterium is relaxed by treating the terms that provoke the barotropic mode in an implicit manner. After an explicit “predictor” step, velocities are corrected with the implicit free surface correction in the “corrector” step. In this method, the free surface correction follows from the inversion of the elliptic free surface correction equation obtained from the 2-D continuity equation.

Much effort has been made to implement suitable schemes for the advection of momentum and scalars. A variety of schemes are available from the lit-

erature, e.g. second and higher order central and upwind schemes (see [Hirsch, 1990](#), for a review), Flux Corrected Transport (FCT; [Boris & Book, 1979](#)), Total Variation Diminishing (TVD; [Roe, 1986](#); [Sweby, 1984](#)), Quadratic Upstream Interpolation for Convective Kinematics (QUICK; [Leonard, 1979](#)), Second Order Moments (SOM; [Prather, 1986](#); [Hofmann & Maqueda, 2006](#)), Piecewise Parabolic Method (PPM; [Colella & Woodward, 1984](#); [James, 1996](#)). Implementing different schemes within the same model code is a tedious task since most higher order schemes impose a coupling between space and time discretisation. The basic choice in the program will therefore be limited to the upwind and the TVD scheme to reduce the programming and computational overhead. The latter scheme is implemented with the symmetrical operator splitting method for time integration and can be considered as a useful tool for the simulation of frontal structures and areas with strong current gradients. The upwind scheme, on the other hand, is only first order accurate and therefore more diffusive, and should be used if CPU time is considered of more importance than accuracy.

The following additional issues are noted:

- When the mode-splitting method is used, scalar quantities are advected with a “filtered” velocity (u_f, v_f) derived from the “corrected” baroclinic currents and the depth-integrated current averaged over the internal time step ([Deleersnijder, 1993](#)).
- Sink terms are discretised explicitly in time for cell-centered scalars to make the scheme more conservative, whereas a quasi-implicit formulation is implemented for turbulence transport to ensure positivity ([Patankar, 1980](#)).

This chapter is organised as follows:

- The model grid, the grid indexing system and notational conventions are described in Section [4.2](#).
- The solution of the momentum equations is presented in Section [12.3](#).
- The scalar transport equations are discussed in Section [12.4](#).
- Numerical aspects of the turbulence module are given in Section [12.5](#).
- The discretisations for one-dimensional (water column) and two-dimensional (depth-averaged) applications are discussed in Section [12.6](#).
- The general solution procedure is summarised in Section [12.7](#).

12.2 Time discretisation

The time discretisation of the model equations is summarised below. A detailed description is given in the sections below.

- In case a mode-splitting technique is used (Blumberg & Mellor, 1987), separate time steps are taken for the 2-D “external” barotropic equations ($\Delta\tau$) and the “internal” baroclinic equations (Δt). The 2-D time step $\Delta\tau$ has to be small enough to satisfy the Courant-Friedrichs-Lowy (CFL) criterion (see equation (12.1) below). The 3-D time step is a multiple, M_t , of $\Delta\tau$ (typically of the order of 10–20) and the model is integrated forward in time for N_t baroclinic time steps (equal to $N_t M_t = M_{tot}$ barotropic time steps). From stability analysis for linear surface gravity waves

$$\Delta\tau \leq \frac{\Delta h_{min}}{2\sqrt{gh_{max}}} \quad (12.1)$$

and

$$\Delta t \leq \frac{\Delta h_{min}}{2\sqrt{g'h_{max}}} \quad (12.2)$$

where $\Delta h_{min} = \min(h_1, h_2)$ is the minimum horizontal grid spacing, $g' = g\Delta\rho/\rho_0$ the reduced gravity, h_{max} the maximum water depth and $\Delta\rho$ a typical value for the vertical density difference. Since $g' \ll g$ the second condition is less constraining than the first one. A more stringent condition for the 3-D mode, imposed by the explicit schemes for horizontal advection, is that the horizontal distance travelled by a fluid element during the internal time step Δt , must be smaller than the grid spacing, or

$$\left(\frac{u\Delta t}{h_1}, \frac{v\Delta t}{h_2} \right) \leq 1 \quad (12.3)$$

- The semi-implicit hydrodynamic scheme only uses one (3-D) timestep. In this case, $M_t = 1$ and $\Delta\tau = \Delta t$. Because of the implicit treatment of the free surface wave, there is no need for the 2-D CFL time step restriction (12.1) for stability. The convective CFL criterion, eq. (12.3), still needs to be satisfied in all cells at all times.
- All horizontal derivatives are evaluated explicitly while vertical diffusion is computed fully implicitly and vertical advection quasi-implicitly.
- A predictor-corrector method is used to solve the horizontal momentum equations (5.19)–(5.20). This satisfies the requirement (Blumberg & Mellor, 1987) that, when using a mode-splitting technique, the currents

in the 3-D equations should have the same depth integral as the ones obtained from the 2-D depth-integrated equations.

- A quasi-implicit method is implemented for the Coriolis terms.
- Time integration is performed with the operator splitting method in conjunction with the TVD scheme for advection, whereas a simpler forward scheme is considered when advection is discretised with the upwind scheme.
- The sink terms in the momentum and turbulent transport equations, representing e.g. the bottom stress in the momentum equation or the dissipation rate ε and work against stable density gradients in e.g. the k -equation (5.165), are discretised quasi-implicitly to ensure positivity (Patankar, 1980). The sink terms in all other transport equations will be taken explicitly for reasons of conservation.
- The time step at which a quantity is evaluated in the discretised equations, is represented by one of the following superscripts (see also Table 4.2):
 - n : 3-D quantity at the old baroclinic time level $t^n = n\Delta t$
 - $n+1$: 3-D quantity at the new baroclinic time level $t^{n+1} = (n+1)\Delta t$
 - m : 2-D quantity at the old barotropic time level $t^m = n\Delta t + m\Delta\tau$
 - $m+1$: 2-D quantity at the new barotropic time level $t^{m+1} = n\Delta t + (m+1)\Delta\tau$
 - p : horizontal current at the “predicted” time step

The superscript is omitted if no confusion is possible. If multiple superscripts appear separated by semicolons, the last superscript represents the spatial node, the one before last the time level. For example, $u^{n;c}$ denotes the value of u at time level n and node “C”. In case of multiple subscripts separated by semicolons, the last one(s) is (are) the spatial index (indices).

- The time step notations are the same in the implicit case except that there are no intermediate barotropic time steps. However, there is now a possibility to perform the hydrodynamic solution more than once every time step. Particularly in case of the use of the semi-implicit free surface correction method, the accuracy can be enhanced by applying extra iterations. The values at these extra iteration levels are addressed with the following superscripts (see also Table 4.2):

- it : quantity at the previous iteration level
- $it + 1$: quantity at the present iteration level

12.3 Momentum equations

12.3.1 General procedure for the explicit case

The 3-D momentum equations are solved by a predictor-corrector method in which the sequence of operations for each baroclinic time step is as follows:

1. An initial (predictor) estimate of the currents u^p , v^p is calculated from the equations of three-dimensional motion.
2. An implicit correction is added to the predicted values for the Coriolis terms.
3. The 2-D depth-integrated equations of continuity and momentum are solved for ζ , U and V . This involves M_t integrations in time.
4. An implicit correction is added for the Coriolis terms at each barotropic time step.
5. The 3-D horizontal current u^p and v^p are corrected yielding u^{n+1} and v^{n+1} by adjusting u^p and v^p to ensure that the integrated currents obtained from the 2-D and 3-D momentum equations are identical.
6. The transformed and physical vertical current are obtained by solving (5.34) and (5.31).

Table 12.1: Parameters and variables used in the numerical description. Global and local FORTRAN names refer to the variables as defined on respectively the global and local (parallel) grid.

Symbol	Global name	Local name	Purpose
N_x	nc	ncloc	number of grid cells in the X-direction
N_y	nr	nrloc	number of grid cells in the Y-direction
N_z	nz	nz	number of grid cells in the vertical direction
h_1	—	delxatc	grid spacing in the X-direction at the cell centre
h_2	—	delyatc	grid spacing in the Y-direction at the cell centre

(Continued)

Table 12.1: *Continued*

h_3	—	—	grid spacing in the vertical direction at the cell centre (calculated as $H\Delta\sigma_k$)
$\Delta\sigma_k$	—	delsatc	grid spacing in the vertical σ -space at the cell centre
$\Delta\tau$	delt2d	delt2d	(barotropic or external) time step for the 2-D mode equations. In case of an implicit scheme $\Delta\tau = \Delta t$.
Δt	delt3d	—	(baroclinic or internal) time step used for the update of 3-D momentum (3-D mode) and all scalar quantities
M_t	ic3d	ic3d	number of 2-D (barotropic) time step within one 3-D (baroclinic) time step ($= \Delta t/\Delta\tau$). In case of an implicit scheme $M_t=1$.
N_{tot}	—	—	total number of 3-D time steps used in the simulation
M_{tot}	nstep	nstep	total number of 2-D time steps used in the simulation ($= M_t N_{tot}$)
θ_c	theta_cor	theta_cor	implicity factor for the Coriolis force with a value between 0 (explicit) and 1 (implicit). The default value, currently used in the program, is 0.5.
θ_a	theta_vadv	theta_vadv	implicity factor for vertical advection with a value between 0 (explicit) and 1 (implicit). The default value, currently used in the program, is 0.501.
θ_v	theta_vdif	theta_vdif	implicity factor for vertical diffusion with a value between 0 (explicit) and 1 (implicit). The default value, currently used in the program, is 1.
it_{max}	maxitsimp	maxitsimp	maximum allowed number of outer iterations for the implicit scheme
ϵ_{imp}	dzetaresid_conv	dzetaresid_conv	convergence limit for the free surface correction as used in (12.39)

(Continued)

Table 12.1: *Continued*

$\Omega(r)$	—	—	weight function between the upwind and Lax-Wendroff (central) fluxes used in the evaluation of the horizontal (vertical) advective fluxes. Its value depends on the value of the advective switches <code>iop_adv_3D</code> (3-D currents), <code>iop_adv_2D</code> (2-D currents), <code>iop_adv_scal</code> (scalars) and <code>iop_adv_turb</code> (turbulent variables) as given by (12.47)–(12.50).
u_f	—	<code>ufvel</code>	X-component of the “filtered” advective velocity, used for the advection of scalar quantities
v_f	—	<code>vfvel</code>	Y-component of the “filtered” advective velocity, used for the advection of scalar quantities
U_f	—	<code>udfvel</code>	value of the depth-integrated current U averaged over one baroclinic time step, as given by (12.19)
V_f	—	<code>vdfvel</code>	value of the depth-integrated current V averaged over one baroclinic time step, as given by (12.19)

12.3.1.1 predictor step

1. Firstly, the following terms are evaluated using values of currents, T , S at the old time step (t^n):
 - the density ρ from the equation of state (see Section 5.1.3) if `iop_dens>0`
 - the coefficients of vertical diffusion if `iop_vdif_coef>0`
 - the baroclinic pressure gradient (see Section 12.3.13) if `iop_dens_grad>0`
 - the coefficients of horizontal diffusion if `iop_hdif_coef=2` (Smagorinsky scheme, see Section 12.3.12.1)
2. The 3-D momentum equations (5.19) and (5.20) are integrated in time at each (internal) grid point (i,j,k) . Their discretised forms without operator splitting, is given by

$$\frac{\tilde{u}^p - u^n}{\Delta t} = fv^n - \mathcal{A}_{h1}(u^n) - \mathcal{A}_{h2}(u^n) - \frac{v^{n;u}}{h_1^u h_2^u} (u^n \Delta_y^u h_1^{uv} - v^{n;u} \Delta_x^u h_2^c)$$

$$\begin{aligned}
& -\theta_a \mathcal{A}_v(\tilde{u}^p) - (1 - \theta_a) \mathcal{A}_v(u^n) + \theta_v \mathcal{D}_{mv}(\tilde{u}^p) + (1 - \theta_v) \mathcal{D}_{mv}(u^n) \\
& -g \frac{\Delta_x^u \zeta^n}{h_1^u} - \frac{\Delta_x^u P_a}{\rho_0 h_1^u} + F_1^{b;n} + F_1^{t;n+1} + \mathcal{D}_{mh1}(\tau_{11}^n) + \mathcal{D}_{mh2}(\tau_{12}^n)
\end{aligned} \tag{12.4}$$

$$\begin{aligned}
\frac{\tilde{v}^p - v^n}{\Delta t} = & -fu^n - \mathcal{A}_{h1}(v^n) - \mathcal{A}_{h2}(v^n) - \frac{u^{n;v}}{h_1^v h_2^v} (v^n \Delta_x^v h_2^{uv} - u^{n;v} \Delta_y^v h_1^c) \\
& -\theta_a \mathcal{A}_v(\tilde{v}^p) - (1 - \theta_a) \mathcal{A}_v(v^n) + \theta_v \mathcal{D}_{mv}(\tilde{v}^p) + (1 - \theta_v) \mathcal{D}_{mv}(v^n) \\
& -g \frac{\Delta_y^v \zeta^n}{h_2^v} - \frac{\Delta_y^v P_a}{\rho_0 h_2^v} + F_2^{b;n} + F_2^{t;n+1} + \mathcal{D}_{mh1}(\tau_{21}^n) + \mathcal{D}_{mh2}(\tau_{22}^n)
\end{aligned} \tag{12.5}$$

where $(\tilde{u}^p, \tilde{v}^p)$ are the “predicted” currents before implicit Coriolis correction, $f = 2\Omega \sin \phi$ is the Coriolis frequency, \mathcal{A}_{hi} , \mathcal{A}_v are the horizontal and vertical advection operators defined by (5.22)–(5.24) and \mathcal{D}_{mhi} , \mathcal{D}_{mv} the horizontal and vertical diffusion operators defined by (5.25)–(5.27).

3. The predictor currents are obtained by adding an implicit Coriolis correction:

$$\begin{aligned}
u^p = \tilde{u}^p & + \frac{f\theta_c \Delta t (\Delta v^u - f\theta_c \Delta t \Delta u)}{1 + (f\theta_c \Delta t)^2} \\
v^p = \tilde{v}^p & - \frac{f\theta_c \Delta t (\Delta u^v + f\theta_c \Delta t \Delta v)}{1 + (f\theta_c \Delta t)^2}
\end{aligned} \tag{12.6}$$

where

$$\Delta u = \tilde{u}^p - u^n, \quad \Delta v = \tilde{v}^p - v^n \tag{12.7}$$

For details see Appendix C.

4. The “predicted” values for the depth-integrated current are obtained by integrating u^p and v^p over the vertical

$$U^p = \sum_{k=1}^{N_z} u_k^p h_{3;k}^{n;u}, \quad V^p = \sum_{k=1}^{N_z} v_k^p h_{3;k}^{n;v} \tag{12.8}$$

The following features are to be noted:

- The forward (Euler) scheme for time discretisation in (12.4)–(12.5) is replaced by the operator splitting method, discussed in Section 12.3.3.2, in case the TVD scheme is applied for the advective terms.

- By default the Coriolis terms are evaluated semi-implicitly ($\theta_c=0.5$). The implicitity factors for vertical advection θ_a and diffusion θ_v are set to respectively 0.5, 0.501 (semi-implicit) and 1 (fully implicit method). This is further discussed in Section 12.3.3.1.
- The equations are solved at the predictor step with application of surface and bottom boundary conditions, but without open boundary conditions.

12.3.1.2 depth-integrated equations

1. The depth-integrated baroclinic advective and diffusive terms (5.58)–(5.61) are updated using values of the baroclinic current at the old time level t^n .
2. The astronomical tidal force is updated at the new time level t^{n+1} (Section 12.3.14) if `iopt_astro_tide=1`.
3. The 2-D continuity equation (5.32) for the surface elevation ζ and the depth-integrated momentum equations (5.47)–(5.48) for U , V are solved at each (internal) grid point (i,j) for $M_t = \Delta t / \Delta \tau$ barotropic time steps

$$\frac{\zeta^{m+1} - \zeta^m}{\Delta \tau} = -\frac{1}{h_1 h_2} \left(\Delta_x^c (h_2^u U^m) + \Delta_y^c (h_1^v V^m) \right) \quad (12.9)$$

$$\begin{aligned} \frac{\tilde{U}^{m+1} - U^m}{\Delta \tau} + \frac{k_{b2}^u}{H^{m;u}} \tilde{U}^{m+1} &= f V^{m;u} - \bar{\mathcal{A}}_{h1}(U^m) - \bar{\mathcal{A}}_{h2}(U^m) \\ - \frac{\bar{v}^{m;u}}{h_1^u h_2^u} (U^m \Delta_y^u h_1^{uv} - H^{m+1;u} \bar{v}^{m;u} \Delta_x^u h_2^c) - \frac{g H^{m+1;u}}{h_1^u} \Delta_x^u \zeta^{m+1} \\ - \frac{H^{m+1;u}}{\rho_0 h_1^u} \Delta_x^u P_a + \bar{F}_1^{b;n} + H^{m+1;u} F_1^{t;m+1} + \tau_{s1}^u - k_{b1}^u (u_b^p - \frac{U^p}{H^{n;u}}) \\ + \bar{\mathcal{D}}_{mh1}(\bar{\tau}_{11})^m + \bar{\mathcal{D}}_{mh2}(\bar{\tau}_{12})^m - \bar{\delta A}_{h1}^n + \bar{\delta D}_{h1}^n \end{aligned} \quad (12.10)$$

$$\begin{aligned} \frac{\tilde{V}^{m+1} - V^m}{\Delta \tau} + \frac{k_{b2}^v}{H^{m;v}} \tilde{V}^{m+1} &= -f U^{m;v} - \bar{\mathcal{A}}_{h1}(V^m) - \bar{\mathcal{A}}_{h2}(V^m) \\ - \frac{\bar{u}^{m;v}}{h_1^v h_2^v} (V^m \Delta_x^v h_2^{uv} - H^{m+1;v} \bar{u}^{m;v} \Delta_y^v h_1^c) - \frac{g H^{m+1;v}}{h_2^v} \Delta_y^v \zeta^{m+1} \\ - \frac{H^{m+1;v}}{\rho_0 h_2^v} \Delta_y^v P_a + \bar{F}_2^{b;n} + H^{m+1;v} F_2^{t;m+1} + \tau_{s2}^v - k_{b1}^v (v_b^p - \frac{V^p}{H^{n;v}}) \end{aligned}$$

$$+ \bar{\mathcal{D}}_{mh1}(\bar{\tau}_{21})^m + \bar{\mathcal{D}}_{mh2}(\bar{\tau}_{22})^m - \bar{\delta}A_{h2}^n + \bar{\delta}D_{h2}^n \quad (12.11)$$

where

$$(\bar{u}, \bar{v}) = (U/H^u, V/H^v) \quad (12.12)$$

are the depth-mean currents and \mathcal{A}_{hi} , \mathcal{D}_{mhi} are the 2-D advective and diffusion operators defined by (5.51)–(5.54).

A quasi-implicit formulation is used for the bottom stress in the U -equation of the form

$$\tau_{b1}^u = k_{b1}^u \left(u_b^p - \frac{U^p}{H^{n;u}} \right) + k_{b2}^u \frac{\tilde{U}^{m+1}}{H^{m;u}} \quad (12.13)$$

The friction velocities k_{b1} and k_{b2} depend on the formulation for the bottom stress (see equations (5.316)–(5.320)).

$$\begin{aligned} \text{no bottom stress} &: k_{b1}^u = k_{b2}^u = 0 \\ \text{linear bottom stress} &: k_{b1}^u = 0, k_{b2}^u = k_{lin} \\ \text{3-D quadratic law} &: k_{b1}^u = k_{b2}^u = C_{db}^u \left((u_b^n)^2 + (v_b^{n;u})^2 \right)^{1/2} \\ \text{2-D quadratic law} &: k_{b1}^u = 0, k_{b2}^u = C_{db}^u \left((\bar{u}^m)^2 + (\bar{v}^{m;u})^2 \right)^{1/2} \end{aligned} \quad (12.14)$$

The bottom drag coefficient C_{db}^u is calculated from (5.322), giving

$$C_{db;ij}^u = \kappa^2 \left[\ln \left(\max(0.5h_{3;ij1}^u/z_{0;ij}^u), \xi_{min} \right) \right]^{-2} \quad (12.15)$$

or by (5.323)

$$C_{db;ij}^u = \kappa^2 \left[\ln \left(\max(H_{ij}^u/(ez_{0;ij}^u)), \xi_{min} \right) \right]^{-2} \quad (12.16)$$

or by interpolating an externally supplied C-node value at the U-node. Note that the discretisations guarantee that C_{db} remains finite, in case of a drying condition (i.e. when $z_r \rightarrow z_0$).

The bottom stress at the V-node is treated similarly.

4. An implicit correction is applied for the Coriolis terms:

$$\begin{aligned} U^{m+1} &= \tilde{U}^{m+1} + \frac{f\theta_c \Delta\tau (\Delta V^u - f\theta_c \Delta\tau \Delta U)}{1 + (f\theta_c \Delta\tau)^2} \\ V^{m+1} &= \tilde{V}^{m+1} - \frac{f\theta_c \Delta\tau (\Delta U^v + f\theta_c \Delta\tau \Delta V)}{1 + (f\theta_c \Delta\tau)^2} \end{aligned} \quad (12.17)$$

where

$$\Delta U = \tilde{U}^{m+1} - U^m, \quad \Delta V = \tilde{V}^{m+1} - V^m \quad (12.18)$$

For details see Appendix C.

5. The 2-D open boundary conditions are applied (see Section 12.3.16.1).
6. After solving (12.9)–(12.11) M_t times, the solutions are averaged over the baroclinic time step, giving

$$U_f = \frac{1}{M_t} \sum_{m=1}^{M_t} U^m, \quad V_f = \frac{1}{M_t} \sum_{m=1}^{M_t} V^m \quad (12.19)$$

where $M_t = \Delta t / \Delta \tau$ is the number of barotropic time steps.

12.3.1.3 corrector step

1. Open boundary conditions are applied for the baroclinic part

$$(\delta u, \delta v) = (u^{n+1} - \bar{u}^{n+1}, v^{n+1} - \bar{v}^{n+1}) \quad (12.20)$$

2. The predicted values u^p, v^p of the horizontal current are corrected to ensure that the depth-integrated currents obtained from the 2-D mode equations (12.10)–(12.11) are identical to the depth-integrated values of the 3-D current. The corrected values are then given by

$$u^{n+1} = \frac{H^{n;u} u^p + U^{n+1} - U^p}{H^{n+1;u}} \quad (12.21)$$

$$v^{n+1} = \frac{H^{n;v} v^p + V^{n+1} - V^p}{H^{n+1;v}} \quad (12.22)$$

3. The “filtered” advective velocities u_f and v_f , used for the advection of scalar quantities (see Section 12.4), are obtained by adding the depth-integrated current averaged over the baroclinic time step to the baroclinic part of the 3-D corrected current:

$$u_f^{n+1} = \frac{H^{n;u} u^p + U_f - U^p}{H^{n+1;u}} \quad (12.23)$$

$$v_f^{n+1} = \frac{H^{n;v} v^p + V_f - V^p}{H^{n+1;v}} \quad (12.24)$$

For details of the procedures see Ruddick (1995).

12.3.1.4 vertical current

The transformed vertical current ω is obtained by integrating the “baroclinic” continuity equation (5.34) from the bottom. Omitting the i - and j -indices this gives

$$\begin{aligned}\omega_1^{n+1} &= 0 \\ \mathcal{F}_k &= \frac{1}{h_1^c h_2^c} \left[\Delta_x^c \left(h_2^u h_{3;k}^{n+1;u} (u_k^{n+1} - \bar{u}_k^{n+1}) \right) + \Delta_y^c \left(h_1^v h_{3;k}^{n+1;v} (v_k^{n+1} - \bar{v}_k^{n+1}) \right) \right] \\ &\quad + \frac{U^{n+1;c}}{h_1^c} \Delta_x^c \left(\frac{h_{3;k}^{n+1;u}}{H^{n+1;u}} \right) + \frac{V^{n+1;c}}{h_2^c} \Delta_y^c \left(\frac{h_{3;k}^{n+1;v}}{H^{n+1;v}} \right) \\ \omega_{k+1}^{n+1} &= \omega_k^{n+1} - \mathcal{F}_k \quad \text{for } 2 \leq k \leq N_z \\ \omega_{N_z+1}^{n+1} &= 0\end{aligned}\tag{12.25}$$

The procedure guarantees that $\omega_{N_z+1}^{n+1} = 0$.

The physical vertical current w is computed at the C-nodes from (5.31):

$$\begin{aligned}w_k^{n+1} &= \frac{2(H^{n+1;c} z_k^{n+1;c} - H^{n;c} z_k^{n;c})}{\Delta t (H^{n;c} + H^{n+1;c})} \\ &\quad + \frac{1}{h_1^c h_2^c h_{3;k}^{n+1;c}} \left[\Delta_x^c \left(h_2^u h_{3;k}^{n+1;u} u_k^{n+1} z_k^{n+1;u} \right) + \Delta_y^c \left(h_1^v h_{3;k}^{n+1;v} v_k^{n+1} z_k^{n+1;v} \right) \right] \\ &\quad + \frac{\Delta_z^c (z_k^{n+1;w} \omega_k^{n+1})}{h_{3;k}^{n+1;c}}\end{aligned}\tag{12.26}$$

where

$$z_k^{n+1;c} = H^{n+1;c} \sigma_k^c - h^c\tag{12.27}$$

and similar expressions at other nodes or time levels.

12.3.2 General procedure for the implicit case

With the implicit method, there is no longer need to solve the depth-integrated momentum equations (unless a 2-D grid has been selected). The stringent CFL stability criterium is relaxed by treating the terms that provoke the barotropic mode in an implicit manner. Difference with the previous explicit version is that the surface slope term is taken at the new time level. Horizontal advection and diffusion are calculated, as before, at the old time level.

After an explicit “predictor” step, velocities are corrected with the implicit free surface correction in the “corrector” step. In this method, the free

surface correction follows from the inversion of the elliptic free surface correction equation obtained from the 2-D continuity equation. Because of the non-linear dependency of the equations on the free surface height through the h_3 -term, an iterative scheme has been implemented in addition.

1. At the first iteration $\zeta^{n+1,1} = \zeta^n$ and $h_3^{n+1,1} = (h + \zeta^n)\Delta\sigma$.
2. The momentum equations are solved at the predictor step using the latest values for h_3 and ζ :

$$\begin{aligned} \frac{h_3^{n+1,it}\tilde{u}^p - h_3^n u^n}{h_3^n \Delta t} &= fv^n - \mathcal{A}_{h1}(u^n) - \mathcal{A}_{h2}(u^n) \\ &- \frac{v^{n;u}}{h_1^u h_2^u} (u^n \Delta_y^u h_1^{uv} - v^{n;u} \Delta_x^u h_2^c) - \theta_a \mathcal{A}_v(\tilde{u}^p) - (1 - \theta_a) \mathcal{A}_v(u^n) \\ &+ \theta_v \mathcal{D}_{mv}(\tilde{u}^p) + (1 - \theta_v) \mathcal{D}_{mv}(u^n) - g \frac{h_3^{n+1,it}}{h_3^n} \frac{\Delta_x^u \zeta^{n+1,it}}{h_1^u} \\ &- \frac{\Delta_x^u P_a}{\rho_0 h_1^u} + F_1^{b;n} + F_1^{t;n+1} + \mathcal{D}_{mh1}(\tau_{11}^n) + \mathcal{D}_{mh2}(\tau_{12}^n) \end{aligned} \quad (12.28)$$

$$\begin{aligned} \frac{1}{h_3^n} \frac{h_3^{n+1,it}\tilde{v}^p - h_3^n v^n}{\Delta t} &= -fu^n - \mathcal{A}_{h1}(v^n) - \mathcal{A}_{h2}(v^n) \\ &- \frac{u^{n;v}}{h_1^v h_2^v} (v^n \Delta_x^v h_2^{uv} - u^{n;v} \Delta_y^v h_1^c) - \theta_a \mathcal{A}_v(\tilde{v}^p) - (1 - \theta_a) \mathcal{A}_v(v^n) \\ &+ \theta_v \mathcal{D}_{mv}(\tilde{v}^p) + (1 - \theta_v) \mathcal{D}_{mv}(v^n) - g \frac{h_3^{n+1,it}}{h_3^n} \frac{\Delta_y^v \zeta^{n+1,it}}{h_2^v} \\ &- \frac{\Delta_y^v P_a}{\rho_0 h_2^v} + F_2^{b;n} + F_2^{t;n+1} + \mathcal{D}_{mh1}(\tau_{21}^n) + \mathcal{D}_{mh2}(\tau_{22}^n) \end{aligned} \quad (12.29)$$

where the surface slope is taken at the previous iteration level. The predicted currents (u^p, v^p) are obtained from $(\tilde{u}^p, \tilde{v}^p)$ after applying the implicit correction for the Coriolis terms, given by (12.6)–(12.7).

3. The free surface correction ζ' is defined as

$$\zeta' = \zeta^{n+1,it+1} - \zeta^{n+1,it} \quad (12.30)$$

The corrected depth-integrated current is then obtained by adding an implicit correction term

$$U^{n+1,it+1} = U^p - H^{n+1,it;u} \frac{\Delta t g}{h_1} \frac{\partial \zeta'}{\partial \xi_1} \quad (12.31)$$

$$V^{n+1, it+1} = V^p - H^{n+1, it; v} \frac{\Delta t g}{h_2} \frac{\partial \zeta'}{\partial \xi_2} \quad (12.32)$$

where (U^p, V^p) are the depth integrated values of (u^p, v^p) .

The values for ζ' follow from inversion of the elliptic equation that arises by introducing (12.31)–(12.32) into the 2-D continuity equation

$$\begin{aligned} \frac{\zeta^{n+1, it} - \zeta^n}{\Delta t} + \frac{\zeta'}{\Delta t} &= -\frac{1}{h_1 h_2} (\Delta_x^c (h_2^u U^p) + \Delta_y^c (h_1^v V^p)) \\ &+ \frac{1}{h_1 h_2} \left[\Delta_x^c \left(\frac{\Delta t h_2^u g^u H^{n+1, it; u}}{h_1^u} \Delta_x^u \zeta' \right) + \Delta_y^c \left(\frac{\Delta t h_1^v g^v H^{n+1, it; v}}{h_2^v} \Delta_y^v \zeta' \right) \right] \end{aligned} \quad (12.33)$$

Equation (12.33) can be written as a linear system of equations with non-zero values only on the diagonal and five sub-diagonals

$$A_{ij} \zeta'_{i-1, j} + B_{i, j} \zeta'_{i, j-1} + C_{ij} \zeta'_{i, j} + D_{ij} \zeta'_{i, j+1} + E_{ij} \zeta'_{i+1, j} = F_{ij} \quad (12.34)$$

Since the decomposition (12.31)–(12.32) can no longer be used at open boundaries, U^{n+1} or V^{n+1} are firstly written as a sum of explicit and implicit (involving ζ') terms which are then substituted into the continuity equation. This is further discussed in Section 12.3.19.1.

4. The free surface elevation is updated

$$\zeta^{n+1, it+1} = \zeta^{n+1, it} + \zeta' \quad (12.35)$$

5. The total water depth is updated

$$H^{n+1, it+1} = H^{n+1, it} + \zeta' \quad (12.36)$$

6. The depth-integrated velocity fields are corrected using (12.31)–(12.32).
7. The values of $U^{n+1, it+1}$ and $V^{n+1, it+1}$ are evaluated at the open boundaries by applying the appropriate boundary conditions.
8. The predicted values u^p, v^p of the horizontal current are corrected to ensure that the depth-integrated currents obtained from equations (12.31)–(12.32) are identical to the depth-integrated values of the 3-D current. The corrected values are then given by

$$u^{n+1} = \frac{H^{n+1, it; u} u^p + U^{n+1, it+1} - U^p}{H^{n+1, it+1; u}} \quad (12.37)$$

$$v^{n+1} = \frac{H^{n+1, it; v} v^p + V^{n+1, it+1} - V^p}{H^{n+1, it+1; v}} \quad (12.38)$$

9. A convergence check is performed by comparing the norm of ζ' with a threshold value ϵ , i.e.

$$\|\zeta'\|_\infty = \max(\zeta') \leq \epsilon_{imp} \quad (12.39)$$

A new iteration is started when the criterion is not satisfied, unless $it > it_{max}$ in which case no further iterations are taken.

10. After completing the iteration loop, the vertical current is obtained by integration of the “baroclinic” continuity equation, as described in Section 12.3.1.4. Since there are no barotropic time steps, one has $u_f = u^{n+1}$, $v_f = v^{n+1}$.

At present, no algorithm has been programmed within the COHERENS source code to solve the linear system, arising from the discretisation of the 2-D continuity equation.

In summary, application of the implicit scheme involves two iteration loops. The inner loop solves the linear system for ζ' and is controlled by the routines of the multi-grid scheme. The maximum number of iterations of the outer loop (needed for convergence of the h_3 -factor) is set by the user with the parameter `maxitsimp`.

12.3.3 Advection schemes and time discretisation

12.3.3.1 introduction

The time discretisation of the momentum equations depends on the type of advection scheme employed for the spatial discretisation of the horizontal and vertical advection terms. Several schemes are implemented in the program, selected with the model switches `iopt_adv_3D` and `iopt_adv_2D`. They may take the following values:

- 0 : horizontal and vertical advection of momentum disabled
- 1 : upwind scheme for horizontal and vertical advection
- 2 : Lax-Wendroff scheme for horizontal, central scheme for vertical advection¹
- 3 : TVD (Total Variation Diminishing) scheme using the superbee limiter as a weighting function between the upwind scheme and either the Lax-Wendroff scheme in the horizontal or the central scheme in the vertical

¹The “pure” Lax-Wendroff and central schemes have only been implemented for illustrative purposes and should be avoided in realistic simulations.

4 : as the previous case now using the monotonic limiter.

The discretisation of the different advection schemes is illustrated with the following simple example, describing the 1-D advection of a scalar ψ :

$$\frac{\partial \psi}{\partial t} + a \frac{\partial \psi}{\partial x} = 0 \quad (12.40)$$

where a is a constant advecting velocity and the equation is spatially integrated for the interval $x_a \leq x \leq x_b$. The equation can then be rewritten in flux form

$$\frac{\partial \psi}{\partial t} + \frac{\partial F}{\partial x} = 0 \quad (12.41)$$

where $F = a\psi$ is the advective flux. The discretised form of (12.41), using forward Euler time integration, is given by

$$\frac{\psi^{n+1} - \psi^n}{\Delta t} + \frac{F_{i+1} - F_i}{\Delta x} = 0 \quad (12.42)$$

where Δt is the time step and Δx a uniform grid spacing. The quantities ψ and F_i are evaluated on a uniform staggered grid (see Figure 12.1) with ψ -points located halfway between the F -points. Boundary conditions at x_a and x_b are needed to determine the fluxes F_1 and F_{N+1} . At interior points, i.e. for $2 \leq i \leq N+1$, the fluxes F_i are then written as a weighting between the upwind and Lax-Wendroff fluxes $F_{up;i}$ and $F_{lw;i}$:

$$F_i = \left((1 - \Omega(r_i)) F_{up;i} + \Omega(r_i) F_{lw;i} \right) \quad (12.43)$$

where

$$F_{up;i} = \frac{1}{2}a \left((1 + s_i)\psi_{i-1} + (1 - s_i)\psi_i \right) \quad (12.44)$$

$$F_{lw;i} = \frac{1}{2}a \left((1 + c_i)\psi_{i-1} + (1 - c_i)\psi_i \right) \quad (12.45)$$

where s_i and c_i are the sign and CFL number of the advecting current

$$s_i = \text{Sign}(a), \quad c_i = \frac{a\Delta t}{\Delta x} \quad (12.46)$$

The weight function Ω depends on the type of advection scheme:

- upwind

$$\Omega(r) = 0 \quad (12.47)$$

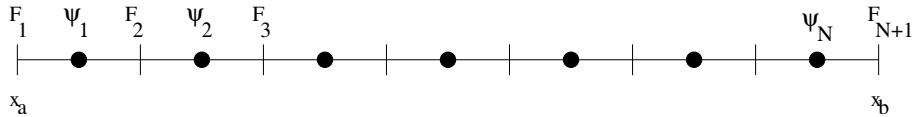


Figure 12.1: Numerical grid for the 1-D advection problem.

- Lax-Wendroff

$$\Omega(r) = 1 \quad (12.48)$$

- TVD with superbee limiter

$$\Omega(r) = \max(0, \min(2r, 1), \min(r, 2)) \quad (12.49)$$

- TVD with monotonic limiter

$$\Omega(r) = \frac{r + |r|}{1 + |r|} \quad (12.50)$$

The argument r of Ω is defined by

$$\begin{aligned} r_i &= \frac{(1 + s_i)\Delta F_{i-1} + (1 - s_i)\Delta F_{i+1}}{2\Delta F_i} \\ \Delta F_i &= F_{lw;i} - F_{up;i} \end{aligned} \quad (12.51)$$

The discretisation scheme for vertical advection is similar, except that the Lax-Wendroff flux $F_{lw;k}$ is replaced by the central flux

$$F_{ce;k} = \frac{1}{2}a(\psi_{k-1} + \psi_k) \quad (12.52)$$

The discretisation schemes, actually applied in the model, need to take account of the following additional complexities

- non-uniform grids
- space and time dependent currents
- grid staggering (adverted quantities and advecting currents can, for example, be located at the same locations)
- extensions to 2-D and 3-D grids
- time integration using operator splitting (see below) to improve the time accuracy of the TVD scheme

Explicit expressions of each discretisation will be presented below for the advective terms of all model equations.

The upwind scheme has the interesting property to preserve monotonicity, but has the disadvantage of being only first order accurate. The Lax-Wendroff scheme, on the other hand, is accurate to second order in space and time but non-monotone which means that spurious over- and undershootings are created in regimes of strong gradients. This is clearly illustrated by the results of the test cases *cones* and *front* described in Sections ?? and ???. The TVD scheme has the advantage of combining the monotonicity of the upwind scheme with the second order accuracy of the Lax-Wendroff scheme.

Horizontal advection is evaluated explicitly to prevent the solution of large-banded matrix systems. A necessary stability condition for both the upwind and the Lax-Wendroff scheme is given by the criterion (12.3) (see [Hirsch, 1990](#)). The restriction to explicit schemes does not apply for the vertical since the discretised equations can be written into a simpler tridiagonal form (see Section [12.3.18](#)). A semi-implicit scheme in the vertical allows to replace the Lax-Wendroff by the central scheme which is a monotone scheme and stable provided that the implicitity factor $\theta_a \geq 0.5$.

The aim of the limiter function is to reduce the numerical diffusion due to the upwind scheme in areas of low gradients and to provide sufficiently large diffusion in regions of large gradients so that over- and undershooting due to the non-monotonicity of the Lax-Wendroff scheme are suppressed. Both the superbee ([Roe, 1985](#)) as the monotonic limiter are available in the program. The *cones* and *front* test case simulations (see Sections ?? and ???) showed that the superbee limiter is the least diffusive and is therefore taken as the default formulation in the program. The spatial discretisation of the advective terms in the momentum equations and the form of the limiter function are further discussed in the subsections below.

In the absence of advection or when the upwind or Lax-Wendroff/central scheme is selected, the momentum equations are solved by forward time-stepping as given by the time-discretised forms (12.4)–(12.5) or (12.28)–(12.29). In case of the TVD scheme, the spatial discretisation of the advective terms involves the Lax-Wendroff and central schemes which are both second order accurate in space. The equations are then integrated in time with the aid of the “fractional step” or “operator splitting” method as proposed by [Yanenko \(1971\)](#). The procedure consists in splitting the time integration into three fractional steps. During the first and second step only the advection-diffusion terms in respectively the X- and Y-direction are taken into account. The vertical advection and diffusion terms and all other terms (Coriolis force, pressure gradient and tidal force) are included during the third time step. To preserve the second-order accuracy of the 1-D schemes in

the fractional step approach the method of symmetric splitting (e.g. [Hirsch, 1990](#)) is implemented. This means that the previous procedure (“A”-steps) is repeated now in reverse order (“B”-steps), i.e. vertical advection/diffusion and other terms followed by advection-diffusion in the Y-direction, followed by advection-diffusion in the X-direction. The final “predicted” value of u^p or v^p is then obtained by taking the average of the values at the end of the A- and B-steps. The same method is applied for scalar quantities.

The implicitity factors θ_a and θ_v have a range between 0 and 1 where a 0 corresponds to a fully explicit, 1 a fully implicit and 0.5 a semi-implicit (Crank-Nicholson) method. The schemes are stable provided that $\theta_a, \theta_v \geq 0.5$. To retain the same accuracy in time for horizontal as well as vertical advection the defaults are a semi-implicit option for vertical advection, i.e. $\theta_a = 0.501^2$ and a fully implicit treatment of vertical diffusion ($\theta_v = 1$). Contrary to **COHERENS V1**, these defaults can be changed by the user and can take any value between 0 and 1.

For a more detailed account of advection schemes and the time splitting method see [Ruddick \(1995\)](#).

²The central scheme is second accurate in time if $\theta_a = 0.5$.

Table 12.2: Overview of the operators used in the numerical discretisations.

Type	Purpose
difference operators	
Δ_x	difference operator in the X-direction
Δ_y	difference operator in the Y-direction
Δ_z	difference operator in the vertical direction
advective operators	
\mathcal{A}_{h1}	horizontal advection in the X-direction ³ $\mathcal{A}_{h1}(F) = \frac{1}{h_1 h_2 h_3} \frac{\partial}{\partial \xi_1} (h_2 h_3 u_f F)$
\mathcal{A}_{h2}	horizontal advection in the Y-direction ³ $\mathcal{A}_{h2}(F) = \frac{1}{h_1 h_2 h_3} \frac{\partial}{\partial \xi_2} (h_1 h_3 v_f F)$
\mathcal{A}_v	vertical advection (u, v and scalars) $\mathcal{A}_v(F) = \frac{1}{h_3} \frac{\partial}{\partial s} (\omega F)$
$\bar{\mathcal{A}}_{h1}$	horizontal advection in the X-direction (2-D mode) $\bar{\mathcal{A}}_{h1}(F) = \frac{1}{h_1 h_2} \frac{\partial}{\partial \xi_1} \left(\frac{h_2 U F}{H} \right)$
$\bar{\mathcal{A}}_{h2}$	horizontal advection in the Y-direction (2-D mode) $\bar{\mathcal{A}}_{h2}(F) = \frac{1}{h_1 h_2} \frac{\partial}{\partial \xi_2} \left(\frac{h_1 V F}{H} \right)$
extended advective operators for currents including curvature terms	

(Continued)

³Note that (u_f, v_f) is replaced by (u, v) if F represents u, v or a turbulent transport variable (k, ε, kl) .

Table 12.2: *Continued*

$\tilde{\mathcal{A}}_{h1}(u)$	extended horizontal advection of u in the X-direction
	$\tilde{\mathcal{A}}_{h1}(u) = \mathcal{A}_{h1}(u) - \frac{v^2}{h_1 h_2} \frac{\partial h_2}{\partial \xi_1}$
$\tilde{\mathcal{A}}_{h2}(u)$	extended horizontal advection of u in the Y-direction
	$\tilde{\mathcal{A}}_{h2}(u) = \mathcal{A}_{h2}(u) + \frac{uv}{h_1 h_2} \frac{\partial h_1}{\partial \xi_2}$
$\tilde{\mathcal{A}}_{h1}(v)$	extended horizontal advection of v in the X-direction
	$\tilde{\mathcal{A}}_{h1}(v) = \mathcal{A}_{h1}(v) + \frac{uv}{h_1 h_2} \frac{\partial h_2}{\partial \xi_1}$
$\tilde{\mathcal{A}}_{h2}(v)$	extended horizontal advection of v in the Y-direction
	$\tilde{\mathcal{A}}_{h2}(v) = \mathcal{A}_{h2}(v) - \frac{u^2}{h_1 h_2} \frac{\partial h_1}{\partial \xi_2}$
$\bar{\mathcal{A}}_{h1}(U)$	extended horizontal advection of U in the X-direction
	$\bar{\mathcal{A}}_{h1}(U) = \bar{\mathcal{A}}_{h1}(U) - \frac{\bar{v}V}{h_1 h_2} \frac{\partial h_2}{\partial \xi_1}$
$\bar{\mathcal{A}}_{h2}(U)$	extended horizontal advection of U in the Y-direction
	$\bar{\mathcal{A}}_{h2}(U) = \bar{\mathcal{A}}_{h2}(U) + \frac{\bar{v}U}{h_1 h_2} \frac{\partial h_1}{\partial \xi_2}$
$\bar{\mathcal{A}}_{h1}(V)$	extended horizontal advection of V in the X-direction
	$\bar{\mathcal{A}}_{h1}(V) = \bar{\mathcal{A}}_{h1}(V) + \frac{\bar{u}V}{h_1 h_2} \frac{\partial h_2}{\partial \xi_1}$
$\bar{\mathcal{A}}_{h2}(V)$	extended horizontal advection of V in the Y-direction
	$\bar{\mathcal{A}}_{h2}(V) = \bar{\mathcal{A}}_{h2}(V) - \frac{\bar{u}U}{h_1 h_2} \frac{\partial h_1}{\partial \xi_2}$
diffusion operators	
\mathcal{D}_{sh1}	horizontal diffusion in the X-direction (scalars)

(Continued)

Table 12.2: *Continued*

	$\mathcal{D}_{sh1}(\psi) = \frac{1}{h_1 h_2 h_3} \frac{\partial}{\partial \xi_1} \left(\lambda_H \frac{h_2 h_3}{h_1} \frac{\partial \psi}{\partial \xi_1} \right)$
\mathcal{D}_{sh2}	horizontal diffusion in the Y-direction (scalars) $\mathcal{D}_{sh2}(\psi) = \frac{1}{h_1 h_2 h_3} \frac{\partial}{\partial \xi_2} \left(\lambda_H \frac{h_1 h_3}{h_2} \frac{\partial \psi}{\partial \xi_2} \right)$
\mathcal{D}_{mh1}	horizontal diffusion in the X-direction (3-D momentum) $\mathcal{D}_{mh1}(F) = \frac{1}{h_1 h_2^2 h_3} \frac{\partial}{\partial \xi_1} \left(h_2^2 h_3 F \right)$
\mathcal{D}_{mh2}	horizontal diffusion in the Y-direction (3-D momentum) $\mathcal{D}_{mh2}(F) = \frac{1}{h_1^2 h_2 h_3} \frac{\partial}{\partial \xi_2} \left(h_1^2 h_3 F \right)$
τ_{ij}	3-D horizontal shear stress tensor $\begin{aligned} \tau_{11} &= -\tau_{22} = \nu_H D_T \\ \tau_{12} &= \tau_{21} = \nu_H D_S \\ D_T &= \frac{h_2}{h_1} \frac{\partial}{\partial \xi_1} \left(\frac{u}{h_2} \right) - \frac{h_1}{h_2} \frac{\partial}{\partial \xi_2} \left(\frac{v}{h_1} \right) \\ D_S &= \frac{h_1}{h_2} \frac{\partial}{\partial \xi_2} \left(\frac{u}{h_1} \right) + \frac{h_2}{h_1} \frac{\partial}{\partial \xi_1} \left(\frac{v}{h_2} \right) \end{aligned}$
\mathcal{D}_{sv}	vertical diffusion (scalars) $\mathcal{D}_{sv}(F) = \frac{1}{h_3} \frac{\partial}{\partial s} \left(\frac{\lambda_T^\psi}{h_3} \frac{\partial F}{\partial s} \right)$
\mathcal{D}_{mv}	vertical diffusion (momentum) $\mathcal{D}_{mv}(F) = \frac{1}{h_3} \frac{\partial}{\partial s} \left(\frac{\nu_T}{h_3} \frac{\partial F}{\partial s} \right)$
$\overline{\mathcal{D}}_{mh1}$	horizontal diffusion in the X-direction (2-D momentum) $\overline{\mathcal{D}}_{mh1}(F) = \frac{1}{h_1 h_2^2} \frac{\partial}{\partial \xi_1} \left(h_2^2 F \right)$
$\overline{\mathcal{D}}_{mh2}$	horizontal diffusion in the Y-direction (2-D momentum)

(Continued)

Table 12.2: *Continued*

$$\overline{\mathcal{D}}_{mh2}(F) = \frac{1}{h_1^2 h_2} \frac{\partial}{\partial \xi_2} \left(h_1^2 F \right)$$

$\overline{\tau}_{ij}$	2-D horizontal shear stress tensor
	$\begin{aligned}\overline{\tau}_{11} &= -\overline{\tau}_{22} = \overline{\nu_H} \overline{D}_T \\ \overline{\tau}_{12} &= \overline{\tau}_{21} = \overline{\nu_H} \overline{D}_S \\ \overline{D}_T &= \frac{h_2}{h_1} \frac{\partial}{\partial \xi_1} \left(\frac{\overline{u}}{h_2} \right) - \frac{h_1}{h_2} \frac{\partial}{\partial \xi_2} \left(\frac{\overline{v}}{h_1} \right) \\ \overline{D}_S &= \frac{h_1}{h_2} \frac{\partial}{\partial \xi_2} \left(\frac{\overline{u}}{h_1} \right) + \frac{h_2}{h_1} \frac{\partial}{\partial \xi_1} \left(\frac{\overline{v}}{h_2} \right)\end{aligned}$
	other operators
\mathcal{P}	production terms in the scalar transport equations
\mathcal{S}	sink terms in the scalar transport equations
\mathcal{T}	production minus sink terms in the scalar transport equations
	$\mathcal{T} = \mathcal{P} - \mathcal{S}$
\mathcal{C}_{s1}^f	X-corrector term in the scalar transport equations
	$\mathcal{C}_{s1}^f(\psi) = \frac{\psi}{h_1 h_2 h_3} \frac{\partial}{\partial \xi_1} (h_2 h_3 u_f)$
\mathcal{C}_{s2}^f	Y-corrector term in the scalar transport equations
	$\mathcal{C}_{s2}^f(\psi) = \frac{\psi}{h_1 h_2 h_3} \frac{\partial}{\partial \xi_2} (h_1 h_3 v_f)$
\mathcal{C}_{s3}	Z-corrector term in the scalar transport equations
	$\mathcal{C}_{s3}(\psi) = \frac{\psi}{h_3} \frac{\partial \omega}{\partial s}$

12.3.3.2 mode splitting scheme for the 3-D momentum equations

The time-discretised form of the u -equation (5.19) with mode splitting is given by

- Part A

$$\frac{u_A^{n+1/3} - u^n}{\Delta t} = -\mathcal{A}_{h1}(u^n) + \frac{(v^{n;u})^2 \Delta_x^u h_2^c}{h_1^u h_2^u} + \mathcal{D}_{mh1}(\nu_H D_T(u^n, v^n)) \quad (12.53)$$

$$\begin{aligned} \frac{u_A^{n+2/3} - u_A^{n+1/3}}{\Delta t} = & -\mathcal{A}_{h2}(u_A^{n+1/3}) - \frac{u_A^{n+1/3} v^{n;u} \Delta_y^u h_1^{uv}}{h_1^u h_2^u} \\ & + \mathcal{D}_{mh2}(\nu_H D_S(u_A^{n+1/3}, v^n)) \end{aligned} \quad (12.54)$$

$$\begin{aligned} \frac{\tilde{u}_A^p - u_A^{n+2/3}}{\Delta t} = & -\theta_a \mathcal{A}_v(\tilde{u}_A^p) - (1 - \theta_a) \mathcal{A}_v(u_A^{n+2/3}) \\ & + \theta_v \mathcal{D}_{mv}(\tilde{u}_A^p) + (1 - \theta_v) \mathcal{D}_{mv}(u_A^{n+2/3}) + \mathcal{O}_1 \end{aligned} \quad (12.55)$$

- Part B

$$\begin{aligned} \frac{u_B^{n+1/3} - u^n}{\Delta t} = & -\theta_a \mathcal{A}_v(u_B^{n+1/3}) - (1 - \theta_a) \mathcal{A}_v(u^n) \\ & + \theta_v \mathcal{D}_{mv}(u_B^{n+1/3}) + (1 - \theta_v) \mathcal{D}_{mv}(u^n) + \mathcal{O}_1 \end{aligned} \quad (12.56)$$

$$\begin{aligned} \frac{u_B^{n+2/3} - u_B^{n+1/3}}{\Delta t} = & -\mathcal{A}_{h2}(u_B^{n+1/3}) - \frac{u_B^{n+1/3} v^{n;u} \Delta_y^u h_1^{uv}}{h_1^u h_2^u} \\ & + \mathcal{D}_{mh2}(\nu_H D_S(u_B^{n+1/3}, v^n)) \end{aligned} \quad (12.57)$$

$$\frac{\tilde{u}_B^p - u_B^{n+2/3}}{\Delta t} = -\mathcal{A}_{h1}(u_B^{n+2/3}) + \frac{(v^{n;u})^2 \Delta_x^u h_2^c}{h_1^u h_2^u} + \mathcal{D}_{mh1}(\nu_H D_T(u_B^{n+2/3}, v^n)) \quad (12.58)$$

- Predictor value

$$\tilde{u}^p = \frac{1}{2}(\tilde{u}_A^p + \tilde{u}_B^p) \quad (12.59)$$

The \mathcal{O}_1 -terms are defined by

$$\mathcal{O}_1 = f v^{n;u} - g \frac{\Delta_x^u \zeta^n}{h_1^u} - \frac{\Delta_x^u P_a}{\rho_0 h_1^u} + F_1^{b;n} + F_1^{t;n+1} \quad (12.60)$$

A similar procedure is applied for the v -equation (5.20).

- Part A

$$\frac{v_A^{n+1/3} - v^n}{\Delta t} = -\mathcal{A}_{h1}(v^n) - \frac{u^{n;v} v^n \Delta_x^v h_2^{uv}}{h_1^v h_2^v} + \mathcal{D}_{mh1}(\nu_H D_S(u^n, v^n)) \quad (12.61)$$

$$\begin{aligned} \frac{v_A^{n+2/3} - v_A^{n+1/3}}{\Delta t} = & -\mathcal{A}_{h2}(v_A^{n+1/3}) + \frac{(u^{n;v})^2 \Delta_y^v h_1^c}{h_1^v h_2^v} \\ & - \mathcal{D}_{mh2}(\nu_H D_T(u^n, v_A^{n+1/3})) \end{aligned} \quad (12.62)$$

$$\begin{aligned} \frac{\tilde{v}_A^p - v_A^{n+2/3}}{\Delta t} = & -\theta_a \mathcal{A}_v(\tilde{v}_A^p) - (1 - \theta_a) \mathcal{A}_v(v_A^{n+2/3}) \\ & + \theta_v \mathcal{D}_{mv}(\tilde{v}_A^p) + (1 - \theta_v) \mathcal{D}_{mv}(v_A^{n+2/3}) + \mathcal{O}_2 \end{aligned} \quad (12.63)$$

- Part B

$$\begin{aligned} \frac{v_B^{n+1/3} - v^n}{\Delta t} = & -\theta_a \mathcal{A}_v(v_B^{n+1/3}) - (1 - \theta_a) \mathcal{A}_v(v^n) \\ & + \theta_v \mathcal{D}_{mv}(v_B^{n+1/3}) + (1 - \theta_v) \mathcal{D}_{mv}(v^n) + \mathcal{O}_2 \end{aligned} \quad (12.64)$$

$$\begin{aligned} \frac{v_B^{n+2/3} - v_B^{n+1/3}}{\Delta t} = & -\mathcal{A}_{h2}(v_B^{n+1/3}) + \frac{(u^{n;v})^2 \Delta_y^v h_1^c}{h_1^v h_2^v} \\ & - \mathcal{D}_{mh2}(\nu_H D_T(u^n, v_B^{n+1/3})) \end{aligned} \quad (12.65)$$

$$\begin{aligned} \frac{\tilde{v}_B^p - v_B^{n+2/3}}{\Delta t} = & -\mathcal{A}_{h1}(v_B^{n+2/3}) - \frac{u^{n;v} v_B^{n+2/3} \Delta_x^v h_2^{uv}}{h_1^v h_2^v} \\ & + \mathcal{D}_{mh1}(\nu_H D_S(u^n, v_B^{n+2/3})) \end{aligned} \quad (12.66)$$

- Predictor value

$$\tilde{v}^p = \frac{1}{2}(\tilde{v}_A^p + \tilde{v}_B^p) \quad (12.67)$$

The \mathcal{O}_2 -terms are defined by

$$\mathcal{O}_2 = -f u^{n;v} - g \frac{\Delta_y^v \zeta^n}{h_2^v} - \frac{\Delta_y^v P_a}{\rho_0 h_2^v} + F_2^{b;n} + F_2^{t;n+1} \quad (12.68)$$

Once \tilde{u}^p and \tilde{v}^p are obtained, an implicit correction is applied as described in Section 12.3.1.1.

Important to note again is that, compared to the simpler forward scheme, the computation using symmetrical operator splitting increases the CPU time for the circulation module by a factor two, but has the advantage of being more accurate which is an important property in regions of strong horizontal and vertical shear.

12.3.3.3 mode splitting scheme for the 2-D momentum equations

The operator splitting is applied for the 2-D case if the advective terms are discretised with the TVD scheme (iopt_adv_2D=3). Since the 2-D mode equations are solved with a much smaller time step than the 3-D mode, second-order accuracy is of less relevance. Contrary to the 3-D case, the simpler upwind scheme, using only a forward Euler time integration, can be recommended for 2-D applications.

The method is analogous to the 3-D case, but given here in detail for completeness. Firstly, the U -equation (5.47) is solved as follows:

- Part A

$$\begin{aligned} \frac{U_A^{m+1/3} - U_A^m}{\Delta\tau} &= -\bar{\mathcal{A}}_{h1}(U_A^m) + \frac{H^{m;u}(\bar{v}^{m;u})^2 \Delta_x^u h_2^c}{h_1^u h_2^u} \\ &\quad + \bar{\mathcal{D}}_{mh1}(\bar{\nu}_H \bar{D}_T(U_A^m, V^m)) \end{aligned} \quad (12.69)$$

$$\begin{aligned} \frac{U_A^{m+2/3} - U_A^{m+1/3}}{\Delta\tau} &= -\bar{\mathcal{A}}_{h2}(U_A^{m+1/3}) - \frac{U_A^{m+1/3} \bar{v}^{m;u} \Delta_y^u h_1^{uv}}{h_1^u h_2^u} \\ &\quad + \bar{\mathcal{D}}_{mh2}(\bar{\nu}_H \bar{D}_S(U_A^{m+1/3}, V^m)) \end{aligned} \quad (12.70)$$

$$\frac{\tilde{U}_A^{m+1} - U_A^{m+2/3}}{\Delta\tau} + \frac{k_{b2}^u}{H^{m;u}} \tilde{U}_A^{m+1} = \bar{\mathcal{O}}_1 \quad (12.71)$$

- Part B

$$\frac{U_B^{m+1/3} - U_B^m}{\Delta\tau} + \frac{k_{b2}^u}{H^{m;u}} U_B^{m+1/3} = \bar{\mathcal{O}}_1 \quad (12.72)$$

$$\begin{aligned} \frac{U_B^{m+2/3} - U_B^{m+1/3}}{\Delta\tau} &= -\bar{\mathcal{A}}_{h2}(U_B^{m+1/3}) - \frac{U_B^{m+1/3} \bar{v}^{m;u} \Delta_y^u h_1^{uv}}{h_1^u h_2^u} \\ &\quad + \bar{\mathcal{D}}_{mh2}(\bar{\nu}_H \bar{D}_S(U_B^{m+1/3}, V^m)) \end{aligned} \quad (12.73)$$

$$\begin{aligned} \frac{\tilde{U}_B^{m+1} - U_B^{m+2/3}}{\Delta\tau} &= -\bar{\mathcal{A}}_{h1}(U_B^{m+2/3}) + \frac{H^{m+1;u}(\bar{v}^{m;u})^2 \Delta_x^u h_2^c}{h_1^u h_2^u} \\ &\quad + \bar{\mathcal{D}}_{mh1}(\bar{\nu}_H \bar{D}_T(U_B^{m+2/3}, V^m)) \end{aligned} \quad (12.74)$$

- Value at new time step

$$\tilde{U}^{m+1} = \frac{1}{2}(\tilde{U}_A^{m+1} + \tilde{U}_B^{m+1}) \quad (12.75)$$

The $\overline{\mathcal{O}}_1$ -terms are defined by

$$\begin{aligned}\overline{\mathcal{O}}_1 &= fV^{m;u} - \frac{gH^{m+1;u}}{h_1^u} \Delta_x^u \zeta^{m+1} - \frac{H^{m+1;u}}{\rho_0 h_1^u} \Delta_x P_a + \overline{F}_1^{b;n} + H^{m+1;u} F_1^{t;m+1} \\ &\quad + \tau_{s1}^u - k_{b1}^{p;u} \left(u_b^p - \frac{U^p}{H^{n;u}} \right) - \overline{\delta A}_{h1}^n + \overline{\delta D}_{h1}^n\end{aligned}\quad (12.76)$$

A similar procedure is followed for the V -equation (5.48):

- Part A

$$\begin{aligned}\frac{V_A^{m+1/3} - V^m}{\Delta\tau} &= -\overline{\mathcal{A}}_{h1}(V^m) - \frac{\overline{u}^{m;v} V^m \Delta_x^v h_2^{uv}}{h_1^v h_2^v} \\ &\quad + \overline{\mathcal{D}}_{mh1}(\overline{\nu_H} \overline{D}_T(U^m, V^m))\end{aligned}\quad (12.77)$$

$$\begin{aligned}\frac{V_A^{m+2/3} - V_A^{m+1/3}}{\Delta\tau} &= -\overline{\mathcal{A}}_{h2}(V_A^{m+1/3}) + \frac{H^{m;v} (\overline{u}^{m;v})^2 \Delta_y^v h_1^c}{h_1^v h_2^v} \\ &\quad - \overline{\mathcal{D}}_{mh2}(\overline{\nu_H} \overline{D}_S(U^m, V_A^{m+1/3}))\end{aligned}\quad (12.78)$$

$$\frac{\tilde{V}_A^{m+1} - V_A^{m+2/3}}{\Delta\tau} + \frac{k_{b2}^v}{H^{m;v}} \tilde{V}_A^{m+1} = \overline{\mathcal{O}}_2 \quad (12.79)$$

- Part B

$$\frac{V_B^{m+1/3} - V^m}{\Delta\tau} + \frac{k_{b2}^v}{H^{m;v}} V_B^{m+1/3} = \overline{\mathcal{O}}_2 \quad (12.80)$$

$$\begin{aligned}\frac{V_B^{m+2/3} - V_B^{m+1/3}}{\Delta\tau} &= -\overline{\mathcal{A}}_{h2}(V_B^{m+1/3}) + \frac{H^{m;v} (\overline{u}^{m;v})^2 \Delta_y^v h_1^c}{h_1^v h_2^v} \\ &\quad - \overline{\mathcal{D}}_{mh2}(\overline{\nu_H} \overline{D}_S(U^m, V_B^{m+1/3}))\end{aligned}\quad (12.81)$$

$$\begin{aligned}\frac{\tilde{V}_B^{m+1} - V_A^{m+2/3}}{\Delta\tau} &= -\overline{\mathcal{A}}_{h1}(V^{m+2/3}) - \frac{\overline{u}^{m;v} V^{m+2/3} \Delta_x^v h_2^{uv}}{h_1^v h_2^v} \\ &\quad + \overline{\mathcal{D}}_{mh1}(\overline{\nu_H} \overline{D}_T(U^m, V_B^{m+2/3}))\end{aligned}\quad (12.82)$$

- Value at new time step

$$\tilde{V}^{m+1} = \frac{1}{2}(\tilde{V}_A^{m+1} + \tilde{V}_B^{m+1}) \quad (12.83)$$

The $\overline{\mathcal{O}}_2$ -terms are defined by

$$\begin{aligned}\overline{\mathcal{O}}_2 = & -fU^{m;v} - \frac{gH^{m+1;v}}{h_2^v} \Delta_y^v \zeta^{m+1} - \frac{H^{m+1;v}}{\rho_0 h_2^v} \Delta_y P_a + \overline{F}_2^{b;n} + H^{m+1;v} F_2^{t;m+1} \\ & + \tau_{s2}^v - k_{b1}^v \left(v_b^p - \frac{V^p}{H^{n;v}} \right) - \overline{\delta A}_{h2}^n + \overline{\delta D}_{h2}^n\end{aligned}\quad (12.84)$$

Once \tilde{U}^{m+1} and \tilde{V}^{m+1} are obtained, an implicit correction is applied as described in Section 12.3.1.2.

Table 12.3: Definitions of the fluxes used in the numerical discretisations.

Type	Purpose
advective fluxes	
F_1	advective flux of a scalar in the X-direction at the U-node $F_1 = u\psi$
F_2	advective flux of a scalar in the Y-direction at the V-node $F_2 = v\psi$
F_3	advective flux of a scalar in the vertical direction at the W-node $F_3 = \omega\psi$
F_{11}	advective flux of u in the X-direction at the C-node $F_{11} = uu$
F_{12}	advective flux of a u in the Y-direction at the UV-node $F_{12} = vu$
F_{21}	advective flux of a v in the X-direction at the UV-node $F_{21} = uv$
F_{22}	advective flux of v in the Y-direction at the C-node $F_{22} = vv$
F_{13}	advective flux of u in the vertical direction at the UW-node $F_{13} = \omega u$
F_{23}	advective flux of v in the vertical direction at the VW-node $F_{23} = \omega v$
\overline{F}_{11}	advective flux of U in the X-direction at the C-node $\overline{F}_{11} = \bar{u}U$
\overline{F}_{12}	advective flux of U in the Y-direction at the UV-node $\overline{F}_{12} = \bar{v}U$
\overline{F}_{21}	advective flux of V in the X-direction at the UV-node $\overline{F}_{21} = \bar{u}V$
\overline{F}_{22}	advective flux of V in the Y-direction at the C-node $\overline{F}_{22} = \bar{v}V$

(Continued)

Table 12.3: *Continued*

diffusive fluxes	
D_1	diffusive flux of a scalar in the X-direction at the U-node $D_1 = \frac{\lambda_H}{h_1} \frac{\partial \psi}{\partial \xi_1}$
D_2	diffusive flux of a scalar in the Y-direction at the V-node $D_2 = \frac{\lambda_H}{h_2} \frac{\partial \psi}{\partial \xi_2}$
D_3	diffusive flux of a scalar in the vertical direction at the W-node $D_3 = \frac{\lambda_T}{h_3} \frac{\partial \psi}{\partial s}$
D_{11}	diffusive flux in the X-direction (u -equation) at the C-node $D_{11} = h_2 \tau_{11} = \nu_H h_2 D_T$
D_{12}	diffusive flux in the Y-direction (u -equation) at the UV-node $D_{12} = h_1 \tau_{12} = \nu_H h_1 D_S$
D_{21}	diffusive flux in the X-direction (v -equation) at the UV-node $D_{21} = h_2 \tau_{21} = \nu_H h_2 D_S$
D_{22}	diffusive flux in the Y-direction (v -equation) at the C-node $D_{22} = h_1 \tau_{22} = -\nu_H h_1 D_T$
D_{13}	diffusive flux in the vertical direction (u -equation) at the UW-node $D_{13} = \frac{\nu_T}{h_3} \frac{\partial u}{\partial s}$
D_{23}	diffusive flux in the vertical direction (v -equation) at the VW-node $D_{23} = \frac{\nu_T}{h_3} \frac{\partial v}{\partial s}$
\bar{D}_{11}	diffusive flux in the X-direction (U -equation) at the C-node $\bar{D}_{11} = h_2 \bar{\tau}_{11} = \bar{\nu}_H h_2 \bar{D}_T$
\bar{D}_{12}	diffusive flux in the Y-direction (U -equation) at the UV-node $\bar{D}_{12} = h_1 \bar{\tau}_{12} = \bar{\nu}_H h_1 \bar{D}_S$
\bar{D}_{21}	diffusive flux in the X-direction (V -equation) at the UV-node $\bar{D}_{21} = h_2 \bar{\tau}_{21} = \bar{\nu}_H h_2 \bar{D}_S$
\bar{D}_{22}	diffusive flux in the Y-direction (V -equation) at the C-node $\bar{D}_{22} = h_1 \bar{\tau}_{22} = -\bar{\nu}_H h_1 \bar{D}_T$

(Continued)

Table 12.3: *Continued***12.3.4 Discretisation of 3-D horizontal advection**

The four horizontal advective terms in the 3-D momentum equations are written as the divergence of the horizontal fluxes F_{11} , F_{12} , F_{21} , F_{22} , defined in Table 12.3:

$$\mathcal{A}_{h1}(u) = \frac{1}{h_1 h_2 h_3} \frac{\partial}{\partial \xi_1} (h_2 h_3 u^2) = \frac{1}{h_1 h_2 h_3} \frac{\partial}{\partial \xi_1} (h_2 h_3 F_{11}) \quad (12.85)$$

$$\mathcal{A}_{h2}(u) = \frac{1}{h_1 h_2 h_3} \frac{\partial}{\partial \xi_2} (h_1 h_3 u v) = \frac{1}{h_1 h_2 h_3} \frac{\partial}{\partial \xi_2} (h_1 h_3 F_{12}) \quad (12.86)$$

$$\mathcal{A}_{h1}(v) = \frac{1}{h_1 h_2 h_3} \frac{\partial}{\partial \xi_1} (h_2 h_3 u v) = \frac{1}{h_1 h_2 h_3} \frac{\partial}{\partial \xi_1} (h_2 h_3 F_{21}) \quad (12.87)$$

$$\mathcal{A}_{h2}(v) = \frac{1}{h_1 h_2 h_3} \frac{\partial}{\partial \xi_2} (h_1 h_3 v^2) = \frac{1}{h_1 h_2 h_3} \frac{\partial}{\partial \xi_2} (h_1 h_3 F_{22}) \quad (12.88)$$

For simplicity, the k -index and time level will be omitted from the discretisation formulae.

Extended forms of the above operators which include the appropriate curvature term, are defined by (see Table 12.2):

$$\tilde{\mathcal{A}}_{h1}(u) = \mathcal{A}_{h1}(u) - \frac{v^2}{h_1 h_2} \frac{\partial h_2}{\partial \xi_1} \quad (12.89)$$

$$\tilde{\mathcal{A}}_{h2}(u) = \mathcal{A}_{h2}(u) + \frac{u v}{h_1 h_2} \frac{\partial h_1}{\partial \xi_2} \quad (12.90)$$

$$\tilde{\mathcal{A}}_{h1}(v) = \mathcal{A}_{h1}(v) + \frac{u v}{h_1 h_2} \frac{\partial h_2}{\partial \xi_1} \quad (12.91)$$

$$\tilde{\mathcal{A}}_{h2}(v) = \mathcal{A}_{h2}(v) - \frac{u^2}{h_1 h_2} \frac{\partial h_1}{\partial \xi_2} \quad (12.92)$$

12.3.4.1 alongstream advection of u

The alongstream advective term in the u -equation (5.19) is obtained by differencing the flux F_{11}^c at the U-node

$$\mathcal{A}_{h1}(u)_{ij}^u = \frac{h_{2;ij}^c h_{3;ij}^c F_{11;ij}^c - h_{2;i-1,j}^c h_{3;i-1,j}^c F_{11;i-1,j}^c}{h_{1;ij}^u h_{2;ij}^u h_{3;ij}^u} \quad (12.93)$$

The flux is calculated from

$$F_{11;ij}^c = \left(1 - \Omega(r_{ij}^c)\right) F_{up;ij}^c + \Omega(r_{ij}^c) F_{lw;ij}^c \quad (12.94)$$

where $F_{up;ij}^c$ and $F_{lw;ij}^c$ are the upwind and Lax-Wendroff fluxes at the C-node:

$$F_{up;ij}^c = \frac{1}{2}u_{ij}^c \left((1 + s_{ij})u_{ij} + (1 - s_{ij})u_{i+1,j} \right) \quad (12.95)$$

$$F_{lw;ij}^c = \frac{1}{2}u_{ij}^c \left((1 + c_{ij})u_{ij} + (1 - c_{ij})u_{i+1,j} \right) \quad (12.96)$$

where s_{ij} and c_{ij} are the sign and CFL number of the advecting current

$$s_{ij} = \text{Sign}(u_{ij}^c), \quad c_{ij} = \frac{u_{ij}^c \Delta t}{h_{1;ij}^c} \quad (12.97)$$

The form of the weighting function is given by (12.47)–(12.50), depending on the type of advection scheme, selected by the switch `iopt_adv_3D`. The argument r of the weight function is defined by

$$\begin{aligned} r_{ij}^c &= \frac{(1 + s_{ij})\Delta F_{i-1,j}^c + (1 - s_{ij})\Delta F_{i+1,j}^c}{2\Delta F_{ij}^c} \\ \Delta F_{ij}^c &= F_{lw;ij}^c - F_{up;ij}^c \end{aligned} \quad (12.98)$$

The extended advective term is discretised as

$$\tilde{\mathcal{A}}_{h1}(u)_{ij}^u = \mathcal{A}_{h1}(u)_{ij}^u - \frac{(v_{ij}^u)^2 \Delta_x^u h_{2;ij}^c}{h_{1;ij}^u h_{2;ij}^u} \quad (12.99)$$

12.3.4.2 cross-stream advection of u

The cross-stream advective term in the u -equation (5.19) is obtained by differencing the flux F_{12}^{uv} at the U-node

$$\mathcal{A}_{h2}(u)_{ij}^u = \frac{h_{1;ij+1}^{uv} h_{3;ij,j+1}^{uv} F_{12;ij,j+1}^{uv} - h_{1;ij}^{uv} h_{3;ij}^{uv} F_{12;ij}^{uv}}{h_{1;ij}^u h_{2;ij}^u h_{3;ij}^u} \quad (12.100)$$

The flux is calculated from

$$F_{12;ij}^{uv} = \left(1 - \Omega(r_{ij}^{uv}) \right) F_{up;ij}^{uv} + \Omega(r_{ij}^{uv}) F_{lw;ij}^{uv} \quad (12.101)$$

where $F_{up;ij}^{uv}$ and $F_{lw;ij}^{uv}$ are the upwind and Lax-Wendroff fluxes at the UV-node:

$$F_{up;ij}^{uv} = \frac{1}{2}v_{ij}^{uv} \left((\alpha_{ij} + s_{ij})u_{i,j-1} + (\beta_{ij} - s_{ij})u_{ij} \right) \quad (12.102)$$

$$F_{lw;ij}^{uv} = \frac{1}{2}v_{ij}^{uv} \left((\alpha_{ij} + c_{ij})u_{i,j-1} + (\beta_{ij} - c_{ij})u_{ij} \right) \quad (12.103)$$

where

$$s_{ij} = \text{Sign}(v_{ij}^{uv}), \quad c_{ij} = \frac{v_{ij}^{uv} \Delta t}{h_{2;ij}^{uv}} \quad (12.104)$$

$$\alpha_{ij} = \frac{h_{2;ij}^u}{h_{2;ij}^{uv}}, \quad \beta_{ij} = \frac{h_{2;i,j-1}^u}{h_{2;ij}^{uv}} \quad (12.105)$$

The form of the weighting function is given by (12.47)–(12.50), depending on the type of advection scheme, selected by the switch `iopt_adv_3D`. The argument r of the weight function is defined by

$$\begin{aligned} r_{ij}^{uv} &= \frac{(\alpha_{ij} + s_{ij}) \Delta F_{i,j-1}^{uv} + (\beta_{ij} - s_{ij}) \Delta F_{i,j+1}^{uv}}{2 \Delta F_{ij}^{uv}} \\ \Delta F_{ij}^{uv} &= F_{lw;ij}^{uv} - F_{up;ij}^{uv} \end{aligned} \quad (12.106)$$

The extended advective term is discretised as

$$\tilde{\mathcal{A}}_{h2}(u)_{ij}^u = \mathcal{A}_{h2}(u)_{ij} + \frac{u_{ij} v_{ij}^u \Delta_y^u h_{1;ij}^{uv}}{h_{1;ij}^u h_{2;ij}^u} \quad (12.107)$$

12.3.4.3 cross-stream advection of v

The cross-stream advective term in the v -equation (5.20) is obtained by differencing the flux F_{21}^{uv} at the V-node

$$\mathcal{A}_{h1}(v)_{ij}^v = \frac{h_{2;i+1,j}^{uv} h_{3;i+1,j}^{uv} F_{21;i+1,j}^{uv} - h_{2;ij}^{uv} h_{3;ij}^{uv} F_{21;ij}^{uv}}{h_{1;ij}^v h_{2;ij}^v h_{3;ij}^v} \quad (12.108)$$

The flux is calculated from

$$F_{21;ij}^{uv} = \left(1 - \Omega(r_{ij}^{uv})\right) F_{up;ij}^{uv} + \Omega(r_{ij}^{uv}) F_{lw;ij}^{uv} \quad (12.109)$$

where $F_{up;ij}^{uv}$ and $F_{lw;ij}^{uv}$ are the upwind and Lax-Wendroff fluxes at the UV-node:

$$F_{up;ij}^{uv} = \frac{1}{2} u_{ij}^{uv} \left((\alpha_{ij} + s_{ij}) v_{i-1,j} + (\beta_{ij} - s_{ij}) v_{ij} \right) \quad (12.110)$$

$$F_{lw;ij}^{uv} = \frac{1}{2} u_{ij}^{uv} \left((\alpha_{ij} + c_{ij}) v_{i-1,j} + (\beta_{ij} - c_{ij}) v_{ij} \right) \quad (12.111)$$

where

$$s_{ij} = \text{Sign}(u_{ij}^{uv}), \quad c_{ij} = \frac{u_{ij}^{uv} \Delta t}{h_{1;ij}^{uv}} \quad (12.112)$$

$$\alpha_{ij} = \frac{h_{1;ij}^v}{h_{1;ij}^{uv}}, \quad \beta_{ij} = \frac{h_{1;i-1,j}^v}{h_{1;ij}^{uv}} \quad (12.113)$$

The form of the weighting function is given by (12.47)–(12.50), depending on the type of advection scheme, selected by the switch `iopt_adv_3D`. The argument r of the weight function is defined by

$$\begin{aligned} r_{ij}^{uv} &= \frac{(\alpha_{ij} + s_{ij})\Delta F_{i-1,j}^{uv} + (\beta_{ij} - s_{ij})\Delta F_{i+1,j}^{uv}}{2\Delta F_{ij}^{uv}} \\ \Delta F_{ij}^{uv} &= F_{lw;ij}^{uv} - F_{up;ij}^{uv} \end{aligned} \quad (12.114)$$

The extended advective term is discretised as

$$\tilde{\mathcal{A}}_{h1}(v)_{ij}^v = \mathcal{A}_{h1}(v)_{ij}^v + \frac{u_{ij}^v v_{ij} \Delta_x^v h_{2;ij}^{uv}}{h_{1;ij}^v h_{2;ij}^v} \quad (12.115)$$

12.3.4.4 alongstream advection of v

The alongstream advective term in the v -equation (5.20) is obtained by differencing the flux F_{22}^c at the V-node

$$\mathcal{A}_{h2}(v)_{ij}^v = \frac{h_{1;ij}^c h_{3;ij}^c F_{22;ij}^c - h_{1;i,j-1}^c h_{3;i,j-1}^c F_{22;i,j-1}^c}{h_{1;ij}^v h_{2;ij}^v h_{3;ij}^v} \quad (12.116)$$

The flux is calculated from

$$F_{22;ij}^c = \left(1 - \Omega(r_{ij}^c)\right) F_{up;ij}^c + \Omega(r_{ij}^c) F_{lw;ij}^c \quad (12.117)$$

where $F_{up;ij}^c$ and $F_{lw;ij}^c$ are the upwind and Lax-Wendroff fluxes at the C-node:

$$F_{up;ij}^c = \frac{1}{2} v_{ij}^c \left((1 + s_{ij}) v_{ij} + (1 - s_{ij}) v_{i,j+1} \right) \quad (12.118)$$

$$F_{lw;ij}^c = \frac{1}{2} v_{ij}^c \left((1 + c_{ij}) v_{ij} + (1 - c_{ij}) v_{i,j+1} \right) \quad (12.119)$$

where s_{ij} and c_{ij} are the sign and CFL number of the advecting current

$$s_{ij} = \text{Sign}(v_{ij}^c), \quad c_{ij} = \frac{v_{ij}^c \Delta t}{h_{2;ij}^c} \quad (12.120)$$

The form of the weighting function is given by (12.47)–(12.50), depending on the type of advection scheme, selected by the switch `iopt_adv_3D`. The argument r of the weight function is defined by

$$\begin{aligned} r_{ij}^c &= \frac{(1 + s_{ij})\Delta F_{i,j-1}^c + (1 - s_{ij})\Delta F_{i,j+1}^c}{2\Delta F_{ij}^c} \\ \Delta F_{ij}^c &= F_{lw;ij}^c - F_{up;ij}^c \end{aligned} \quad (12.121)$$

The extended advective term is discretised as

$$\tilde{\mathcal{A}}_{h2}(v)_{ij}^v = \mathcal{A}_{h2}(v)_{ij}^v - \frac{(u_{ij}^v)^2 \Delta_y^v h_{1;ij}^c}{h_{1;ij}^v h_{2;ij}^v} \quad (12.122)$$

12.3.5 Discretisation of 2-D horizontal advection

The four horizontal advective terms in the 2-D momentum equations are written as the divergence of the horizontal fluxes \bar{F}_{11} , \bar{F}_{12} , \bar{F}_{21} , \bar{F}_{22} , defined in Table 12.3:

$$\bar{\mathcal{A}}_{h1}(U) = \frac{1}{h_1 h_2} \frac{\partial}{\partial \xi_1} (h_2 \bar{u} U) = \frac{1}{h_1 h_2} \frac{\partial}{\partial \xi_1} (h_2 \bar{F}_{11}) \quad (12.123)$$

$$\bar{\mathcal{A}}_{h2}(U) = \frac{1}{h_1 h_2} \frac{\partial}{\partial \xi_2} (h_1 \bar{v} U) = \frac{1}{h_1 h_2} \frac{\partial}{\partial \xi_2} (h_1 \bar{F}_{12}) \quad (12.124)$$

$$\bar{\mathcal{A}}_{h1}(V) = \frac{1}{h_1 h_2} \frac{\partial}{\partial \xi_1} (h_2 \bar{u} V) = \frac{1}{h_1 h_2} \frac{\partial}{\partial \xi_1} (h_2 \bar{F}_{21}) \quad (12.125)$$

$$\bar{\mathcal{A}}_{h2}(V) = \frac{1}{h_1 h_2} \frac{\partial}{\partial \xi_2} (h_1 \bar{v} V) = \frac{1}{h_1 h_2} \frac{\partial}{\partial \xi_2} (h_1 \bar{F}_{22}) \quad (12.126)$$

Extended forms of the above operators which include the appropriate curvature term, are defined by (see Table 12.2):

$$\bar{\tilde{\mathcal{A}}}_{h1}(U) = \bar{\mathcal{A}}_{h1}(U) - \frac{\bar{v} V}{h_1 h_2} \frac{\partial h_2}{\partial \xi_1} \quad (12.127)$$

$$\bar{\tilde{\mathcal{A}}}_{h2}(U) = \bar{\mathcal{A}}_{h2}(U) + \frac{\bar{v} U}{h_1 h_2} \frac{\partial h_1}{\partial \xi_2} \quad (12.128)$$

$$\bar{\tilde{\mathcal{A}}}_{h1}(V) = \bar{\mathcal{A}}_{h1}(V) + \frac{\bar{u} V}{h_1 h_2} \frac{\partial h_2}{\partial \xi_1} \quad (12.129)$$

$$\bar{\tilde{\mathcal{A}}}_{h2}(V) = \bar{\mathcal{A}}_{h2}(V) - \frac{\bar{u} U}{h_1 h_2} \frac{\partial h_1}{\partial \xi_2} \quad (12.130)$$

12.3.5.1 alongstream advection of U

The alongstream advective term in the U -equation (5.47) is obtained by differencing the flux \bar{F}_{11}^c at the U-node

$$\bar{\mathcal{A}}_{h1}(U)_{ij}^u = \frac{h_{2,ij}^c \bar{F}_{11;ij}^c - h_{2,i-1,j}^c \bar{F}_{11;i-1,j}^c}{h_{1,ij}^u h_{2,ij}^u} \quad (12.131)$$

The flux is calculated from

$$\bar{F}_{11;ij}^c = \left(1 - \Omega(r_{ij}^c)\right) \bar{F}_{up;ij}^c + \Omega(r_{ij}^c) \bar{F}_{lw;ij}^c \quad (12.132)$$

where $\bar{F}_{up;ij}^c$ and $\bar{F}_{lw;ij}^c$ are the upwind and Lax-Wendroff fluxes at the C-node:

$$\bar{F}_{up;ij}^c = \frac{1}{2} \bar{u}_{ij}^c \left((1 + s_{ij}) U_{ij} + (1 - s_{ij}) U_{i+1,j} \right) \quad (12.133)$$

$$\bar{F}_{lw;ij}^c = \frac{1}{2} \bar{u}_{ij}^c \left((1 + c_{ij}) U_{ij} + (1 - c_{ij}) U_{i+1,j} \right) \quad (12.134)$$

where s_{ij} and c_{ij} are the sign and CFL number of the advecting current

$$s_{ij} = \text{Sign}(\bar{u}_{ij}^c), \quad c_{ij} = \frac{\bar{u}_{ij}^c \Delta \tau}{h_{1;ij}^c} \quad (12.135)$$

The form of the weighting function is given by (12.47)–(12.50), depending on the type of advection scheme, selected by the switch `iopt_adv_2D`. The argument r of the weight function is defined by

$$\begin{aligned} r_{ij}^c &= \frac{(1 + s_{ij}) \Delta \bar{F}_{i-1,j}^c + (1 - s_{ij}) \Delta \bar{F}_{i+1,j}^c}{2 \Delta \bar{F}_{ij}^c} \\ \Delta \bar{F}_{ij}^c &= \bar{F}_{lw;ij}^c - \bar{F}_{up;ij}^c \end{aligned} \quad (12.136)$$

The extended advective term is discretised by

$$\bar{\mathcal{A}}_{h1}(U)_{ij}^u = \bar{\mathcal{A}}_{h1}(U)_{ij} - \frac{\bar{v}_{ij}^u V_{ij}^u \Delta_x^u h_{2;ij}^c}{h_{1;ij}^u h_{2;ij}^u} \quad (12.137)$$

12.3.5.2 cross-stream advection of U

The cross-stream advective term in the U -equation (5.47) is obtained by differencing the flux \bar{F}_{12}^{uv} at the U-node

$$\bar{\mathcal{A}}_{h2}(U)_{ij}^u = \frac{h_{1;i,j+1}^{uv} \bar{F}_{12;i,j+1}^{uv} - h_{1;ij}^{uv} \bar{F}_{12;ij}^{uv}}{h_{1;ij}^u h_{2;ij}^u} \quad (12.138)$$

The flux is calculated from

$$\bar{F}_{12;ij}^{uv} = \left(1 - \Omega(r_{ij}^{uv}) \right) \bar{F}_{up;ij}^{uv} + \Omega(r_{ij}^{uv}) \bar{F}_{lw;ij}^{uv} \quad (12.139)$$

where $\bar{F}_{up;ij}^{uv}$ and $\bar{F}_{lw;ij}^{uv}$ are the upwind and Lax-Wendroff fluxes at the UV-node:

$$\bar{F}_{up;ij}^{uv} = \frac{1}{2} \bar{v}_{ij}^{uv} \left((\alpha_{ij} + s_{ij}) U_{i,j-1} + (\beta_{ij} - s_{ij}) U_{ij} \right) \quad (12.140)$$

$$\bar{F}_{lw;ij}^{uv} = \frac{1}{2} \bar{v}_{ij}^{uv} \left((\alpha_{ij} + c_{ij}) U_{i,j-1} + (\beta_{ij} - c_{ij}) U_{ij} \right) \quad (12.141)$$

where

$$s_{ij} = \text{Sign}(\bar{v}_{ij}^{uv}), \quad c_{ij} = \frac{\bar{v}_{ij}^{uv} \Delta \tau}{h_{2;ij}^{uv}} \quad (12.142)$$

$$\alpha_{ij} = \frac{h_{2;ij}^u}{h_{2;ij}^{uv}}, \quad \beta_{ij} = \frac{h_{2;i,j-1}^u}{h_{2;ij}^{uv}} \quad (12.143)$$

The form of the weighting function is given by (12.47)–(12.50), depending on the type of advection scheme, selected by the switch `iopt_adv_2D`. The argument r of the weight function is defined by

$$\begin{aligned} r_{ij}^{uv} &= \frac{(\alpha_{ij} + s_{ij})\Delta\bar{F}_{i,j-1}^{uv} + (\beta_{ij} - s_{ij})\Delta\bar{F}_{i,j+1}^{uv}}{2\Delta\bar{F}_{ij}^{uv}} \\ \Delta\bar{F}_{ij}^{uv} &= \bar{F}_{lw;ij}^{uv} - \bar{F}_{up;ij}^{uv} \end{aligned} \quad (12.144)$$

The extended advective term is discretised by

$$\bar{\mathcal{A}}_{h2}(U)_{ij}^u = \bar{\mathcal{A}}_{h2}(U)_{ij} + \frac{\bar{v}_{ij}^u U_{ij} \Delta_y^u h_{1;ij}^{uv}}{h_{1;ij}^u h_{2;ij}^u} \quad (12.145)$$

12.3.5.3 cross-stream advection of V

The cross-stream advective term in the V -equation (5.48) is obtained by differencing the flux \bar{F}_{21}^{uv} at the V-node

$$\bar{\mathcal{A}}_{h1}(V)_{ij}^v = \frac{h_{2;i+1,j}^{uv} \bar{F}_{21;i+1,j}^{uv} - h_{2;ij}^{uv} \bar{F}_{21;ij}^{uv}}{h_{1;ij}^v h_{2;ij}^v} \quad (12.146)$$

The flux is calculated from

$$\bar{F}_{21;ij}^{uv} = \left(1 - \Omega(r_{ij}^{uv})\right) \bar{F}_{up;ij}^{uv} + \Omega(r_{ij}^{uv}) \bar{F}_{lw;ij}^{uv} \quad (12.147)$$

where $\bar{F}_{up;ij}^{uv}$ and $\bar{F}_{lw;ij}^{uv}$ are the upwind and Lax-Wendroff fluxes at the UV-node:

$$\bar{F}_{up;ij}^{uv} = \frac{1}{2} \bar{u}_{ij}^{uv} \left((\alpha_{ij} + s_{ij}) V_{i-1,j} + (\beta_{ij} - s_{ij}) V_{i,j} \right) \quad (12.148)$$

$$\bar{F}_{lw;ij}^{uv} = \frac{1}{2} \bar{u}_{ij}^{uv} \left((\alpha_{ij} + c_{ij}) V_{i-1,j} + (\beta_{ij} - c_{ij}) V_{i,j} \right) \quad (12.149)$$

where

$$s_{ij} = \text{Sign}(\bar{u}_{ij}^{uv}), \quad c_{ij} = \frac{\bar{u}_{ij}^{uv} \Delta\tau}{h_{1;ij}^{uv}} \quad (12.150)$$

$$\alpha_{ij} = \frac{h_{1;ij}^v}{h_{1;ij}^{uv}}, \quad \beta_{ij} = \frac{h_{1;i,j-1}^v}{h_{1;ij}^{uv}} \quad (12.151)$$

The form of the weighting function is given by (12.47)–(12.50), depending on the type of advection scheme, selected by the switch `iopt_adv_2D`. The argument r of the weight function is defined by

$$\begin{aligned} r_{ij}^{uv} &= \frac{(\alpha_{ij} + s_{ij})\Delta\bar{F}_{i-1,j}^{uv} + (\beta_{ij} - s_{ij})\Delta\bar{F}_{i+1,j}^{uv}}{2\Delta\bar{F}_{ij}^{uv}} \\ \Delta\bar{F}_{ij}^{uv} &= \bar{F}_{lw;ij}^{uv} - \bar{F}_{up;ij}^{uv} \end{aligned} \quad (12.152)$$

The extended advective term is discretised as

$$\bar{\mathcal{A}}_{h1}(V)_{ij}^v = \bar{\mathcal{A}}_{h1}(V)_{ij} + \frac{\bar{u}_{ij}^v V_{ij} \Delta_x^v h_{2;ij}^{uv}}{h_{1;ij}^v h_{2;ij}^v} \quad (12.153)$$

12.3.5.4 alongstream advection of V

The alongstream advective term in the V -equation (5.48) is obtained by differencing the flux \bar{F}_{22}^c at the V-node

$$\bar{\mathcal{A}}_{h2}(V)_{ij}^v = \frac{h_{1;ij}^c \bar{F}_{22;ij}^c - h_{1;ij-1}^c \bar{F}_{22;ij-1}^c}{h_{1;ij}^v h_{2;ij}^v} \quad (12.154)$$

The flux is calculated from

$$\bar{F}_{22;ij}^c = \left(1 - \Omega(r_{ij}^c)\right) \bar{F}_{up;ij}^c + \Omega(r_{ij}^c) \bar{F}_{lw;ij}^c \quad (12.155)$$

where $\bar{F}_{up;ij}^c$ and $\bar{F}_{lw;ij}^c$ are the upwind and Lax-Wendroff fluxes at the C-node:

$$\bar{F}_{up;ij}^c = \frac{1}{2} \bar{v}_{ij}^c \left((1 + s_{ij}) V_{ij} + (1 - s_{ij}) V_{i,j+1} \right) \quad (12.156)$$

$$\bar{F}_{lw;ij}^c = \frac{1}{2} \bar{v}_{ij}^c \left((1 + c_{ij}) V_{ij} + (1 - c_{ij}) V_{i,j+1} \right) \quad (12.157)$$

where s_{ij} and c_{ij} are the sign and CFL number of the advecting current

$$s_{ij} = \text{Sign}(\bar{v}_{ij}^c), \quad c_{ij} = \frac{\bar{v}_{ij}^c \Delta\tau}{h_{2;ij}^c} \quad (12.158)$$

The form of the weighting function is given by (12.47)–(12.50), depending on the type of advection scheme, selected by the switch `iopt_adv_2D`. The argument r of the weight function is defined by

$$r_{ij}^c = \frac{(1 + s_{ij})\Delta\bar{F}_{i,j-1}^c + (1 - s_{ij})\Delta\bar{F}_{i,j+1}^c}{2\Delta\bar{F}_{ij}^c}$$

$$\Delta \bar{F}_{ij}^c = \bar{F}_{lw;ij}^c - \bar{F}_{up;ij}^c \quad (12.159)$$

The extended advective term is discretised as

$$\bar{\tilde{\mathcal{A}}}_{h2}(V)_{ij}^v = \bar{\mathcal{A}}_{h2}(V)_{ij} - \frac{\bar{u}_{ij}^v U_{ij}^v \Delta_y h_{1;ij}^c}{h_{1;ij}^v h_{2;ij}^v} \quad (12.160)$$

12.3.6 Integrals of the baroclinic advection terms

The discretised versions of the advective integrals in the 2-D momentum equations at time step t^n are given by

$$\bar{\delta \mathcal{A}}_{h1;ij}^u = \sum_{k=1}^{N_z} \left(\tilde{\mathcal{A}}_{h1}(u)_{ijk}^u + \tilde{\mathcal{A}}_{h2}(u)_{ijk}^u \right) h_{3;ijk}^u - \bar{\tilde{\mathcal{A}}}_{h1}(U)_{ij}^u - \bar{\tilde{\mathcal{A}}}_{h2}(U)_{ij}^u \quad (12.161)$$

$$\bar{\delta \mathcal{A}}_{h2;ij}^v = \sum_{k=1}^{N_z} \left(\tilde{\mathcal{A}}_{h1}(v)_{ijk}^v + \tilde{\mathcal{A}}_{h2}(v)_{ijk}^v \right) h_{3;ijk}^v - \bar{\tilde{\mathcal{A}}}_{h1}(V)_{ij}^v - \bar{\tilde{\mathcal{A}}}_{h2}(V)_{ij}^v \quad (12.162)$$

12.3.7 Discretisation of vertical advection

The vertical advection terms in the 3-D momentum equations are written as the divergence of the vertical fluxes F_{13} , F_{23} , defined in Table 12.3:

$$\mathcal{A}_v(u) = \frac{1}{h_3} \frac{\partial}{\partial s} (\omega u) = \frac{1}{h_3} \frac{\partial F_{13}}{\partial s} \quad (12.163)$$

$$\mathcal{A}_v(v) = \frac{1}{h_3} \frac{\partial}{\partial s} (\omega v) = \frac{1}{h_3} \frac{\partial F_{23}}{\partial s} \quad (12.164)$$

12.3.7.1 vertical advection of u

The vertical advective term in the u -equation (5.19) is obtained by differencing the flux F_{13}^{uw} at the U-node

$$\mathcal{A}_v(u)_{ijk}^u = \frac{F_{13;ij,k+1}^{uw} - F_{13;ijk}^{uw}}{h_{3;ijk}^u} \quad (12.165)$$

The flux is calculated from

$$F_{13;ijk}^{uw} = \left(1 - \Omega(r_{ijk}^{uw}) \right) F_{up;ijk}^{uw} + \Omega(r_{ijk}^{uw}) F_{ce;ijk}^{uw} \quad (12.166)$$

where $F_{up;ijk}^{uw}$ and $F_{ce;ijk}^{uw}$ are the upwind and central fluxes at the UW-node:

$$F_{up;ijk}^{uw} = \frac{1}{2} \omega_{ijk}^{uw} \left((\alpha_{ijk} + s_{ijk}) u_{ij,k-1} + (\beta_{ijk} - s_{ijk}) u_{ijk} \right) \quad (12.167)$$

$$F_{ce;ijk}^{uw} = \frac{1}{2} \omega_{ijk}^{uw} (\alpha_{ijk} u_{ij,k-1} + \beta_{ijk} u_{ijk}) \quad (12.168)$$

where

$$s_{ijk} = \text{Sign}(\omega_{ijk}^{uw}), \quad \alpha_{ijk} = \frac{h_{3;ijk}^u}{h_{3;ijk}^{uw}}, \quad \beta_{ijk} = \frac{h_{3;ijk,k-1}^u}{h_{3;ijk}^{uw}} \quad (12.169)$$

The form of the weighting function is given by (12.47)–(12.50), depending on the type of advection scheme, selected by the switch `iopt_adv_3D`. The argument r of the weight function is defined by

$$\begin{aligned} r_{ijk}^{uw} &= \frac{(\alpha_{ijk} + s_{ijk}) \Delta F_{ij,k-1}^{uw} + (\beta_{ijk} - s_{ijk}) \Delta F_{ij,k+1}^{uw}}{2 \Delta F_{ijk}^{uw}} \\ \Delta F_{ijk}^{uw} &= F_{ce;ijk}^{uw} - F_{up;ijk}^{uw} \end{aligned} \quad (12.170)$$

12.3.7.2 vertical advection of v

The vertical advective term in the v -equation (5.20) is obtained by differentiating the flux F_{23}^{vw} at the V-node

$$\mathcal{A}_v(v)_{ijk}^v = \frac{F_{23;ij,k+1}^{vw} - F_{23;ijk}^{vw}}{h_{3;ijk}^v} \quad (12.171)$$

The flux is calculated from

$$F_{23;ijk}^{vw} = \left(1 - \Omega(r_{ijk}^{vw})\right) F_{up;ijk}^{vw} + \Omega(r_{ijk}^{vw}) F_{ce;ijk}^{vw} \quad (12.172)$$

where $F_{up;ijk}^{vw}$ and $F_{ce;ijk}^{vw}$ are the upwind and central fluxes at the VW-node:

$$F_{up;ijk}^{vw} = \frac{1}{2} \omega_{ijk}^{vw} \left((\alpha_{ijk} + s_{ijk}) v_{ij,k-1} + (\beta_{ijk} - s_{ijk}) v_{ijk} \right) \quad (12.173)$$

$$F_{ce;ijk}^{vw} = \frac{1}{2} \omega_{ijk}^{vw} (\alpha_{ijk} v_{ij,k-1} + \beta_{ijk} v_{ijk}) \quad (12.174)$$

where

$$s_{ijk} = \text{Sign}(\omega_{ijk}^{vw}), \quad \alpha_{ijk} = \frac{h_{3;ijk}^v}{h_{3;ijk}^{vw}}, \quad \beta_{ijk} = \frac{h_{3;ijk,k-1}^v}{h_{3;ijk}^{vw}} \quad (12.175)$$

The form of the weighting function is given by (12.47)–(12.50), depending on the type of advection scheme, selected by the switch `iopt_adv_3D`. The argument r of the weight function is defined by

$$\begin{aligned} r_{ijk}^{vw} &= \frac{(\alpha_{ijk} + s_{ijk}) \Delta F_{ij,k-1}^{vw} + (\beta_{ijk} - s_{ijk}) \Delta F_{ij,k+1}^{vw}}{2 \Delta F_{ijk}^{vw}} \\ \Delta F_{ijk}^{vw} &= F_{ce;ijk}^{vw} - F_{up;ijk}^{vw} \end{aligned} \quad (12.176)$$

12.3.8 Discretisation of 3-D horizontal diffusion

The four horizontal diffusion terms in the 3-D momentum equations are written as the divergence of the horizontal fluxes D_{11} , D_{12} , D_{21} , D_{22} , defined in Table 12.3:

$$\mathcal{D}_{mh1}(\tau_{11}) = \frac{1}{h_1 h_2^2 h_3} \frac{\partial}{\partial \xi_1} (h_2^2 h_3 \tau_{11}) = \frac{1}{h_1 h_2^2 h_3} \frac{\partial}{\partial \xi_1} (h_2 h_3 D_{11}) \quad (12.177)$$

$$\mathcal{D}_{mh2}(\tau_{12}) = \frac{1}{h_1^2 h_2 h_3} \frac{\partial}{\partial \xi_2} (h_1^2 h_3 \tau_{12}) = \frac{1}{h_1^2 h_2 h_3} \frac{\partial}{\partial \xi_2} (h_1 h_3 D_{12}) \quad (12.178)$$

$$\mathcal{D}_{mh1}(\tau_{21}) = \frac{1}{h_1 h_2^2 h_3} \frac{\partial}{\partial \xi_1} (h_2^2 h_3 \tau_{21}) = \frac{1}{h_1 h_2^2 h_3} \frac{\partial}{\partial \xi_1} (h_2 h_3 D_{21}) \quad (12.179)$$

$$\mathcal{D}_{mh2}(\tau_{22}) = \frac{1}{h_1^2 h_2 h_3} \frac{\partial}{\partial \xi_2} (h_1^2 h_3 \tau_{22}) = \frac{1}{h_1^2 h_2 h_3} \frac{\partial}{\partial \xi_2} (h_1 h_3 D_{22}) \quad (12.180)$$

Discretisations for the horizontal diffusion terms in the 3-D momentum equations are given below. For simplicity, the k -index and time level will be omitted.

- alongstream diffusion in the u -equation (5.19) at the U-node

$$\mathcal{D}_{mh1}(\tau_{11})_{ij}^u = \frac{h_{2;ij}^c h_{3;ij}^c D_{11;ij}^c - h_{2;i-1,j}^c h_{3;i-1,j}^c D_{11;i-1,j}^c}{h_{1;ij}^u (h_{2;ij}^u)^2 h_{3;ij}^u} \quad (12.181)$$

$$D_{11;ij}^c = \nu_{H;ij}^c h_{2;ij}^c \left[\frac{h_{2;ij}^c}{h_{1;ij}^c} \Delta_x^c \left(\frac{u_{ij}}{h_{2;ij}^u} \right) - \frac{h_{1;ij}^c}{h_{2;ij}^c} \Delta_y^c \left(\frac{v_{ij}}{h_{1;ij}^v} \right) \right] \quad (12.182)$$

- cross-stream diffusion in the u -equation (5.19) at the U-node

$$\mathcal{D}_{mh2}(\tau_{12})_{ij}^u = \frac{h_{1;i,j+1}^u h_{3;i,j+1}^u D_{12;i,j+1}^u - h_{1;ij}^u h_{3;ij}^u D_{12;ij}^u}{(h_{1;ij}^u)^2 h_{2;ij}^u h_{3;ij}^u} \quad (12.183)$$

$$D_{12;ij}^u = \nu_{H;ij}^{uv} h_{1;ij}^u \left[\frac{h_{1;ij}^{uv}}{h_{2;ij}^{uv}} \Delta_y^{uv} \left(\frac{u_{ij}}{h_{1;ij}^u} \right) + \frac{h_{2;ij}^{uv}}{h_{1;ij}^{uv}} \Delta_x^{uv} \left(\frac{v_{ij}}{h_{2;ij}^v} \right) \right] \quad (12.184)$$

- cross-stream diffusion in the v -equation (5.20) at the V-node

$$\mathcal{D}_{mh1}(\tau_{21})_{ij}^v = \frac{h_{2;i+1,j}^u h_{3;i+1,j}^u D_{21;i+1,j}^u - h_{2;ij}^u h_{3;ij}^u D_{21;ij}^u}{h_{1;ij}^v (h_{2;ij}^v)^2 h_{3;ij}^v} \quad (12.185)$$

$$D_{21;ij}^u = \nu_{H;ij}^{uv} h_{2;ij}^u \left[\frac{h_{1;ij}^{uv}}{h_{2;ij}^{uv}} \Delta_y^{uv} \left(\frac{u_{ij}}{h_{1;ij}^u} \right) + \frac{h_{2;ij}^{uv}}{h_{1;ij}^{uv}} \Delta_x^{uv} \left(\frac{v_{ij}}{h_{2;ij}^v} \right) \right] \quad (12.186)$$

- alongstream diffusion in the v -equation (5.20) at the V-node

$$\mathcal{D}_{mh2}(\tau_{22})_{ij}^v = \frac{h_{1;ij}^c h_{3;ij}^c D_{22;ij}^c - h_{1;i,j-1}^c h_{3;i,j-1}^c D_{22;i,j-1}^c}{(h_{1;ij}^v)^2 h_{2;ij}^v h_{3;ij}^v} \quad (12.187)$$

$$D_{22;ij}^c = \nu_{H;ij}^c h_{1;ij}^c \left[\frac{h_{1;ij}^c}{h_{2;ij}^c} \Delta_y^c \left(\frac{v_{ij}}{h_{1;ij}^v} \right) - \frac{h_{2;ij}^c}{h_{1;ij}^c} \Delta_x^c \left(\frac{u_{ij}}{h_{2;ij}^u} \right) \right] \quad (12.188)$$

12.3.9 Discretisation of 2-D horizontal diffusion

The four horizontal diffusion terms in the 2-D momentum equations are written as the divergence of the horizontal fluxes \bar{D}_{11} , \bar{D}_{12} , \bar{D}_{21} , \bar{D}_{22} , defined in Table 12.3:

$$\bar{D}_{mh1}(\bar{\tau}_{11}) = \frac{1}{h_1 h_2^2} \frac{\partial}{\partial \xi_1} \left(h_2^2 \bar{\tau}_{11} \right) = \frac{1}{h_1 h_2^2} \frac{\partial}{\partial \xi_1} \left(h_2 \bar{D}_{11} \right) \quad (12.189)$$

$$\bar{D}_{mh2}(\bar{\tau}_{12}) = \frac{1}{h_1^2 h_2} \frac{\partial}{\partial \xi_2} \left(h_1^2 \bar{\tau}_{12} \right) = \frac{1}{h_1^2 h_2} \frac{\partial}{\partial \xi_2} \left(h_1 \bar{D}_{12} \right) \quad (12.190)$$

$$\bar{D}_{mh1}(\bar{\tau}_{21}) = \frac{1}{h_1 h_2^2} \frac{\partial}{\partial \xi_1} \left(h_2^2 \bar{\tau}_{21} \right) = \frac{1}{h_1 h_2^2} \frac{\partial}{\partial \xi_1} \left(h_2 \bar{D}_{21} \right) \quad (12.191)$$

$$\bar{D}_{mh2}(\bar{\tau}_{22}) = \frac{1}{h_1^2 h_2} \frac{\partial}{\partial \xi_2} \left(h_1^2 \bar{\tau}_{22} \right) = \frac{1}{h_1^2 h_2} \frac{\partial}{\partial \xi_2} \left(h_1 \bar{D}_{22} \right) \quad (12.192)$$

Discretisations for the horizontal diffusion terms in the 2-D momentum equations are given below.

- alongstream diffusion in the U -equation (5.47) at the U-node

$$\bar{D}_{mh1}(\bar{\tau}_{11})_{ij}^u = \frac{h_{2;ij}^c \bar{D}_{11;ij}^c - h_{2;i-1,j}^c \bar{D}_{11;i-1,j}^c}{h_{1;ij}^u (h_{2;ij}^u)^2} \quad (12.193)$$

$$\bar{D}_{11;ij}^c = \bar{\nu}_{H;ij}^c h_{2;ij}^c \left[\frac{h_{2;ij}^c}{h_{1;ij}^u} \Delta_x^c \left(\frac{\bar{u}_{ij}}{h_{2;ij}^u} \right) - \frac{h_{1;ij}^c}{h_{2;ij}^c} \Delta_y^c \left(\frac{\bar{v}_{ij}}{h_{1;ij}^v} \right) \right] \quad (12.194)$$

- cross-stream diffusion in the U -equation (5.47) at the U-node

$$\bar{D}_{mh2}(\bar{\tau}_{12})_{ij}^u = \frac{h_{1;i,j+1}^{uv} \bar{D}_{12;i,j+1}^{uv} - h_{1;ij}^{uv} \bar{D}_{12;ij}^{uv}}{(h_{1;ij}^u)^2 h_{2;ij}^u} \quad (12.195)$$

$$\bar{D}_{12;ij}^{uv} = \bar{\nu}_{H;ij}^{uv} h_{1;ij}^{uv} \left[\frac{h_{1;ij}^{uv}}{h_{2;ij}^u} \Delta_y^{uv} \left(\frac{\bar{u}_{ij}}{h_{1;ij}^u} \right) + \frac{h_{2;ij}^{uv}}{h_{1;ij}^v} \Delta_x^{uv} \left(\frac{\bar{v}_{ij}}{h_{2;ij}^v} \right) \right] \quad (12.196)$$

- cross-stream diffusion in the V -equation (5.48) at the V-node

$$\bar{\mathcal{D}}_{mh1}(\bar{\tau}_{21})_{ij}^v = \frac{h_{2;i+1,j}^{uv} \bar{D}_{21;i+1,j}^{uv} - h_{2;ij}^{uv} \bar{D}_{21;ij}^{uv}}{(h_{1;ij}^v (h_{2;ij}^v)^2)} \quad (12.197)$$

$$\bar{D}_{21;ij}^{uv} = \bar{\nu}_{Hij}^{uv} h_{2;ij}^{uv} \left[\frac{h_{1;ij}^{uv}}{h_{2;ij}^{uv}} \Delta_y^{uv} \left(\frac{\bar{u}_{ij}}{h_{1;ij}^u} \right) + \frac{h_{2;ij}^{uv}}{h_{1;ij}^{uv}} \Delta_x^{uv} \left(\frac{\bar{v}_{ij}}{h_{2;ij}^v} \right) \right] \quad (12.198)$$

- alongstream diffusion in the V -equation (5.48) at the V-node

$$\bar{\mathcal{D}}_{mh2}(\bar{\tau}_{22})_{ij}^v = \frac{h_{1;ij}^c \bar{D}_{22;ij}^c - h_{2;i,j-1}^c \bar{D}_{22;i,j-1}^c}{(h_{1;ij}^v)^2 h_{2;ij}^v} \quad (12.199)$$

$$\bar{D}_{22;ij}^c = \bar{\nu}_{Hij}^c h_{1;ij}^c \left[\frac{h_{1;ij}^c}{h_{2;ij}^c} \Delta_y^c \left(\frac{\bar{v}_{ij}}{h_{1;ij}^v} \right) - \frac{h_{2;ij}^c}{h_{1;ij}^c} \Delta_x^c \left(\frac{\bar{u}_{ij}}{h_{2;ij}^u} \right) \right] \quad (12.200)$$

12.3.10 Integrals of the baroclinic diffusion terms

The discretised versions of the diffusion integrals in the 2-D momentum equations are given by

$$\bar{\delta D}_{h1;ij}^u = \sum_{k=1}^{N_z} \left(\mathcal{D}_{mh1}(\tau_{11})_{ijk}^u + \mathcal{D}_{mh2}(\tau_{12})_{ijk}^u \right) h_{3;ijk}^u - \bar{\mathcal{D}}_{mh1}(\bar{\tau}_{11})_{ij}^u - \bar{\mathcal{D}}_{mh2}(\bar{\tau}_{12})_{ij}^u \quad (12.201)$$

$$\bar{\delta D}_{h2;ij}^v = \sum_{k=1}^{N_z} \left(\mathcal{D}_{mh1}(\tau_{21})_{ijk}^v + \mathcal{D}_{mh2}(\tau_{22})_{ijk}^v \right) h_{3;ijk}^v - \bar{\mathcal{D}}_{mh1}(\bar{\tau}_{21})_{ij}^v - \bar{\mathcal{D}}_{mh2}(\bar{\tau}_{22})_{ij}^v \quad (12.202)$$

12.3.11 Discretisation of vertical diffusion

The vertical diffusion terms in the 3-D momentum equations are written as the divergence of the vertical fluxes D_{13} , D_{23} , defined in Table 12.3:

$$\mathcal{D}_{mv}(u) = \frac{1}{h_3} \frac{\partial}{\partial s} \left(\frac{\nu_T}{h_3} \frac{\partial u}{\partial s} \right) = \frac{1}{h_3} \frac{\partial D_{13}}{\partial s} \quad (12.203)$$

$$\mathcal{D}_{mv}(v) = \frac{1}{h_3} \frac{\partial}{\partial s} \left(\frac{\nu_T}{h_3} \frac{\partial v}{\partial s} \right) = \frac{1}{h_3} \frac{\partial D_{23}}{\partial s} \quad (12.204)$$

$$(12.205)$$

- The vertical diffusion term in the u -equation (5.19) is obtained by differencing the flux D_{13}^{uw} at the U-node

$$\mathcal{D}_{mv}(u)_{ijk}^u = \frac{D_{13;ij,k+1}^{uw} - D_{13;ijk}^{uw}}{h_{3;ijk}^u} \quad (12.206)$$

The flux is calculated from

$$D_{13;ijk}^{uw} = \frac{\nu_{T;ijk}^{uw}}{h_{3;ijk}^{uw}} \Delta_z^u u_{ijk} \quad (12.207)$$

- The vertical diffusion term in the v -equation (5.20) is obtained by differencing the flux D_{23}^{vw} at the V-node

$$\mathcal{D}_{mv}(v)_{ijk}^v = \frac{D_{23;ij,k+1}^{vw} - D_{23;ijk}^{vw}}{h_{3;ijk}^v} \quad (12.208)$$

The flux is calculated from

$$D_{23;ijk}^{vw} = \frac{\nu_{T;ijk}^{vw}}{h_{3;ijk}^{vw}} \Delta_z^v v_{ijk} \quad (12.209)$$

12.3.12 Diffusion coefficients for momentum

12.3.12.1 horizontal diffusion coefficients

This section describes the discretisation of the horizontal diffusion coefficients for the case that a Smagorinsky scheme has been selected. Firstly, the horizontal tension and shearing are calculated at their “natural” node

$$D_{T;ijk}^c = \frac{h_{2;ij}^c}{h_{1;ij}^c} \Delta_x^c \left(\frac{u_{ijk}}{h_{2;ij}^u} \right) - \frac{h_{1;ij}^c}{h_{2;ij}^c} \Delta_y^c \left(\frac{v_{ijk}}{h_{1;ij}^v} \right) \quad (12.210)$$

$$D_{S;ijk}^{uv} = \frac{h_{1;ij}^{uv}}{h_{2;ij}^{uv}} \Delta_y^{uv} \left(\frac{u_{ijk}}{h_{1;ij}^u} \right) + \frac{h_{2;ij}^{uv}}{h_{1;ij}^{uv}} \Delta_x^{uv} \left(\frac{v_{ijk}}{h_{2;ij}^v} \right) \quad (12.211)$$

The discretised values of the horizontal diffusion coefficient at the C- and corner nodes are obtained by applying (5.10)

$$\nu_{H;ijk}^c = C_m h_{1;ij}^c h_{2;ij}^c \sqrt{\left(D_{T;ijk}^c \right)^2 + \left(D_{S;ijk}^{uv} \right)^2} \quad (12.212)$$

$$\nu_{H;ijk}^{uv} = C_m h_{1;ij}^{uv} h_{2;ij}^{uv} \sqrt{\left(D_{T;ijk}^{uv} \right)^2 + \left(D_{S;ijk}^{uv} \right)^2} \quad (12.213)$$

Note that the (12.212) and (12.213) only require the interpolation of either D_S at the C-node or D_T at the UV-node but not both.

The 2-D coefficients are obtained by vertical integration

$$\overline{\nu}_{Hij}^c = \sum_{k=1}^{N_z} \nu_{H;ijk}^c h_{3;ijk}^c \quad (12.214)$$

$$\overline{\nu}_{Hij}^{uv} = \sum_{k=1}^{N_z} \nu_{H;ijk}^{uv} h_{3;ijk}^{uv} \quad (12.215)$$

12.3.12.2 vertical diffusion coefficient

The vertical diffusion coefficient for momentum ν_T is obtained from one of the available turbulence schemes, described in Section 5.3. Values are first stored at the W-nodes and interpolated afterwards at the UW- and VW-nodes for the calculation of the vertical diffusion fluxes in the momentum equations. The evaluation of ν_T only involves algebraic expressions so that the discretisation procedure is straightforward.

The following comments are to be given

- To avoid spurious numerical oscillations the squared buoyancy frequency N^2 is spatially discretised by averaging over the neighbouring cells in the horizontal:

$$\begin{aligned} \left(N_{ijk}^w \right)^2 = & \left[2w_{ij}(\tilde{N}_{ijk})^2 + w_{i-1,j}(\tilde{N}_{i-1,jk})^2 + w_{i+1,j}(\tilde{N}_{i+1,jk})^2 \right. \\ & \left. + w_{i,j-1}(\tilde{N}_{i,j-1,k})^2 + w_{i,j+1}(\tilde{N}_{i,j+1,k})^2 \right] \\ & \left[2w_{ij} + w_{i-1,j} + w_{i+1,j} + w_{i,j-1} + w_{i,j+1} \right]^{-1} \end{aligned} \quad (12.216)$$

where

$$\left(\tilde{N}_{ijk} \right)^2 = \frac{\beta_{T;ijk}^w (T_{ijk}^c - T_{ij,k-1}^c) - \beta_{S;ijk}^w (S_{ijk}^c - S_{ij,k-1}^c)}{h_{3;ijk}^w} \quad (12.217)$$

is the unfiltered value of N^2 and w_{ij} equals 0 on land and 1 on sea cells. The expansion coefficients β_T and β_S are first obtained from the equation of state at the C-node and then interpolated at the W-nodes.

- The squared shear frequency M^2 is discretised using

$$\left(M_{ijk}^w \right)^2 = \frac{(u_{ijk}^c - u_{ij,k-1}^c)^2 + (v_{ijk}^c - v_{ij,k-1}^c)^2}{(h_{3;ijk}^w)^2} \quad (12.218)$$

Note that the currents are interpolated first at the C-nodes before the vertical derivative is taken.

- The Richardson number is obtained using its definition $Ri = N^2/M^2$. An upper limit of 1000 is imposed to prevent division by zero if $N^2 \gg M^2$.
- If the vertical diffusion coefficient is derived from a RANS model (see Section 5.3.3), its value is not known at the surface and the bottom. Its evaluation involves the turbulent parameters k , ε or l which are obtained from algebraic relations or by solving additional transport equations. This is further discussed in Section 12.5.

12.3.13 Discretisation of the baroclinic pressure gradient

A known problem is the numerical treatment of the baroclinic pressure gradient for σ -coordinate models. Two types of errors may occur

- The two terms on the right hand side of (5.36) may have the same magnitude and different signs. Significant rounding errors may arise, especially in case of large bathymetric gradients.
- Violation of the hydrostatic consistency condition which states that a σ -surface immediately below (above) a given σ -surface remains below (above) the given σ -surface within a horizontal distance of one grid interval

$$\left| \frac{\sigma}{H} \frac{\partial H}{\partial x_i} \right| \Delta x_i < \Delta \sigma \quad (12.219)$$

where Δx_i is the grid resolution in the X- or Y-direction and $\Delta \sigma$ the vertical resolution in σ -space.

For further discussion and examples are found in [Haney \(1991\)](#); [Kliem & Pietrzak \(1999\)](#).

Several solutions have been proposed: fourth order ([McCalpin, 1994](#)) or sixth order ([Chu & Fan, 1997](#)) discretisations, “ z ”-level based methods ([Beckmann & Haidvogel, 1993](#); [Stelling & Van Kester, 1994](#); [Slørdal, 1997](#)), second order method using unequal weighting ([Song, 1998](#)), cubic polynomial interpolation using harmonic averaging ([Shchepetkin & McWilliams, 2003](#)). Three algorithms are implemented in the code: the “traditional” second-order discretisation, a simple z -level method and the [Shchepetkin & McWilliams \(2003\)](#) approach.

Before discussing the implemented algorithms, the baroclinic pressure gradient is rewritten in a more convenient form. In Cartesian coordinates, the last term on the right of (5.14) can be written as

$$\begin{aligned}
 -\frac{\partial q}{\partial x_i} &= -g \frac{\partial}{\partial x_i} \int_z^\zeta \left(\frac{\rho}{\rho_0} - 1 \right) dz' \\
 &= -g \int_z^\zeta \frac{\partial}{\partial x_i} \left(\frac{\rho}{\rho_0} - 1 \right) dz' - g \left(\frac{\rho_s}{\rho_0} - 1 \right) \frac{\partial \zeta}{\partial x_i} \\
 &\simeq -\frac{g}{\rho_0} \int_z^\zeta \frac{\partial \rho}{\partial x_i} dz' \\
 &\simeq g \int_z^\zeta \left(\beta_T \frac{\partial T}{\partial x_i} - \beta_S \frac{\partial S}{\partial x_i} \right) dz' \tag{12.220}
 \end{aligned}$$

where use is made of the Boussinesq approximation. The last line is obtained by applying the equation of state, taking account that T represents potential and not *in situ* temperature. The latter approximation is at least valid for non-oceanic waters.

In transformed coordinates, equation (12.220) becomes

$$\begin{aligned}
 F_i^b &= g \int_z^\zeta \beta_T \left(\frac{\partial T}{\partial x_i} \Big|_\sigma - \frac{\partial T}{\partial z'} \frac{\partial z'}{\partial x_i} \Big|_\sigma \right) dz' - g \int_z^\zeta \beta_S \left(\frac{\partial S}{\partial x_i} \Big|_\sigma - \frac{\partial S}{\partial z'} \frac{\partial z'}{\partial x_i} \Big|_\sigma \right) dz' \\
 &= F_i^T + F_i^S \tag{12.221}
 \end{aligned}$$

where a $\Big|_\sigma$ means a derivative along a surface of constant σ .

The implemented discretisations for F_i^T are discussed below. Algorithms for the salinity and Y-component are obtained in a similar way.

12.3.13.1 second-order method

The scheme uses a straightforward discretisation of (12.221)

$$\begin{aligned}
 F_{ijNz}^T &= \frac{g_{ij}^u}{2h_{1;ij}^u} \beta_{T;ijNz}^u \Delta_x^u(T_{ijNz}) h_{3;ijNz}^u \\
 F_{ij,k-1}^T &= F_{ijk}^T + \frac{g_{ij}^u}{h_{1;ij}^u} \beta_{T;ijk}^{uw} \left(h_{3;ijk}^{uw} \Delta_x^{uw}(T_{ijk}^w) - \Delta_z^{uw}(T_{ijk}^u) \Delta_x^{uw}(z_{ijk}^w) \right) \tag{12.222}
 \end{aligned}$$

for $N_z \geq k \geq 2$.

12.3.13.2 z -level method

The z -level scheme evaluates the horizontal gradient by vertically interpolating the values of T at the C-node to the vertical level of the U-node (see Figure 12.2). If z_{ijk}^u denotes the “physical” z -level of the U-node where the gradient has to be evaluated, the X-component of the gradient is obtained by vertically interpolating the C-node temperature values in the adjacent columns $(i-1,j)$ and (i,j) to the vertical level of the U-node. Denoting this values by respectively T_{ijk}^L and T_{ijk}^R , the pressure gradient is readily evaluated using (12.220):

$$\begin{aligned} F_{ijNz}^T &= \frac{g_{ij}^u}{2h_{1;ij}^u} \beta_{T;ijNz}^u (T_{ijk}^R - T_{ijk}^L) h_{3;ijNz}^u \\ F_{ij,k-1}^T &= F_{ijk}^T + \frac{g_{ij}^u}{h_{1;ij}^u} \beta_{T;ijk}^{uw} (T_{ijk}^R - T_{ijk}^L) h_{3;ijk}^{uw} \end{aligned} \quad (12.223)$$

The following restrictions apply:

- If the vertical position of T_{ijk}^L or T_{ijk}^R is below the lowest grid point, its value is set to T_{ij1} .
- If the vertical position of T_{ijk}^L or T_{ijk}^R is above the highest grid point, its value is set to T_{ijNz} .

Despite its simplicity and the fact that it avoids the truncation problem of the σ -grid, the scheme may produce unrealistic results near the bottom (surface) or adjacent to a sloping boundary.

12.3.13.3 cube-H method

The “cube-H” algorithm is probably the most robust scheme, but at the expense of a larger CPU time. The method uses a cubic spline formalism together with harmonic averaging. Details of the scheme are not given here but can be found in the paper of [Shchepetkin & McWilliams \(2003\)](#).

Omitting the j -index for simplicity, the following procedure is taken

1. Evaluate the integral at the W-nodes

$$FW_{ik} = \int_{\sigma_{ik}^c}^{\sigma_{i,k+1}^c} T \frac{\partial z}{\partial \sigma} d\sigma \quad (12.224)$$

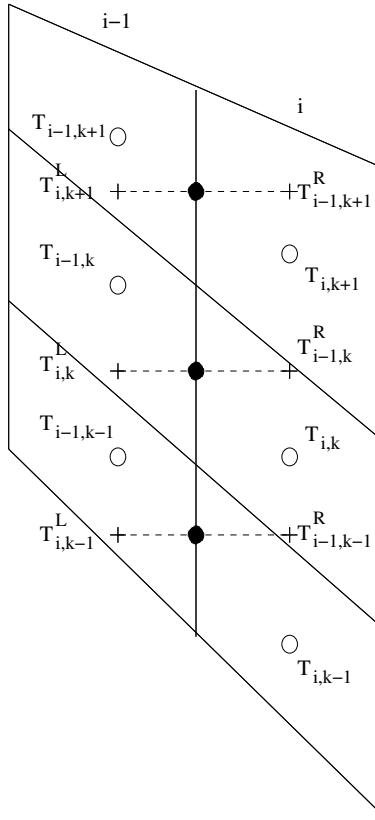


Figure 12.2: Illustration of the z -level interpolation scheme. C- and U-nodes are represented by empty, respectively solid circles.

giving

$$FW_{i,N_z+1} = \frac{1}{2} T_{iN_z} h_{3;iN_z}^c \quad (12.225)$$

$$\begin{aligned} FW_{ik} = & \frac{1}{2} (T_{i,k-1} + T_{ik}) (z_{ik}^c - z_{i,k-1}^c) \\ & - \frac{1}{10} \left[(d_z T_{ik} - d_z T_{i,k-1}) \left(z_{ik}^c - z_{i,k-1}^c - \frac{1}{12} (d_z z_{i,k-1}^c + d_z z_{ik}^c) \right) \right. \\ & \left. - (d_z z_{ik}^c - d_z z_{i,k-1}^c) \left(T_{ik} - T_{i,k-1} - \frac{1}{12} (d_z T_{i,k-1} + d_z T_{ik}) \right) \right] \end{aligned} \quad (12.226)$$

where $2 \leq k \leq N_z$.

2. Evaluate the integral at the UW-nodes

$$FU_{ik} = \int_{\xi_{1;ik}^c}^{\xi_{1;i+1,k}^c} T \frac{\partial z}{\partial \xi_1} d\xi_1 \quad (12.227)$$

giving

$$FU_{i,N_z+1} = 0 \quad (12.228)$$

$$\begin{aligned} FU_{ik} = & \frac{1}{2}(T_{i-1,k} + T_{ik})(z_{ik}^c - z_{i-1,k}^c) \\ & - \frac{1}{10} \left[(d_x T_{ik} - d_x T_{i-1,k}) \left(z_{ik}^c - z_{i-1,k}^c - \frac{1}{12}(d_x z_{i-1,k}^c + d_x z_{ik}^c) \right) \right. \\ & \left. - (d_x z_{ik}^c - d_x z_{i-1,k}^c) \left(T_{ik} - T_{i-1,k} - \frac{1}{12}(d_x T_{i-1,k} + d_x T_{ik}) \right) \right] \end{aligned} \quad (12.229)$$

where $2 \leq k \leq N_z$.

3. The algorithms for the derivatives in (12.226) and (12.229) are given by:

- Vertical derivatives

$$\begin{aligned} d_z f_{ik} &= \frac{2(f_{i,k+1} - f_{ik})(f_{ik} - f_{i,k-1})}{f_{i,k+1} - f_{i,k-1}} & \text{if } (f_{i,k+1} - f_{ik})(f_{ik} - f_{i,k-1}) > 0 \\ d_z f_{ik} &= 0 & \text{otherwise} \end{aligned} \quad (12.230)$$

Values at the boundaries are

$$\begin{aligned} d_z f_{i1} &= \frac{6}{5}(f_{i2} - f_{i1}) - \frac{7}{15}d_z f_{i2} \\ d_z f_{iN_z} &= \frac{6}{5}(f_{iN_z} - f_{i,N_z-1}) - \frac{7}{15}d_z f_{i,N_z-1} \end{aligned} \quad (12.231)$$

- Horizontal derivatives

$$\begin{aligned} d_x f_{ik} &= \frac{2(f_{i+1,k} - f_{ik})(f_{ik} - f_{i-1,k})}{f_{i+1,k} - f_{i-1,k}} & \text{if } (f_{i+1,k} - f_{ik})(f_{ik} - f_{i-1,k}) > 0 \\ d_x f_{ik} &= 0 & \text{otherwise} \end{aligned} \quad (12.232)$$

Boundary conditions are

$$\begin{aligned} i-1 \text{ dry, } i+1 \text{ wet} &: d_x f_{ik} = \frac{6}{5}(f_{i+1,k} - f_{ik}) - \frac{7}{15}d_x f_{i+1,k} \\ i-1 \text{ wet, } i+1 \text{ dry} &: d_x f_{ik} = \frac{6}{5}(f_{ik} - f_{i-1,k}) - \frac{7}{15}d_x f_{i-1,k} \\ i-1 \text{ dry, } i+1 \text{ dry} &: d_x f_{ik} = 0 \end{aligned} \quad (12.233)$$

4. Evaluate the “temperature Jacobian” at the UW-node

$$\mathcal{I}(T, z) = \frac{1}{h_1} \frac{\partial T}{\partial \xi_1} \Big|_{\sigma} \frac{\partial z}{\partial \sigma} - \frac{1}{h_1} \frac{\partial T}{\partial \sigma} \frac{\partial z}{\partial \xi_1} \Big|_{\sigma} \quad (12.234)$$

giving

$$\mathcal{I}_{ik}^{uw} = \frac{g_i^u \beta_{T;ik}^{uw}}{h_{1;i}^u} (FW_{ik} - FW_{i-1,k} - FU_{ik} + FU_{i,k-1}) \quad (12.235)$$

for $2 \leq k \leq N_z$.

5. Evaluate the baroclinic gradient

$$F_{ik}^T = \sum_{k'=k+1}^{N_z+1} \mathcal{I}_{ik'}^{uw} \quad (12.236)$$

for $1 \leq k \leq N_z$, or

$$\begin{aligned} F_{iN_z}^T &= \frac{g_i^u}{2h_{1;i}^u} \beta_{T;iN_z}^u \Delta_x^u(T_{iN_z}) h_{3;iN_z}^u \\ F_{i,k-1}^T &= F_{ik}^T + \mathcal{I}_{ik} \end{aligned} \quad (12.237)$$

for $2 \leq k \leq N_z$.

12.3.14 Tidal force

If the astronomical tidal force is included in the momentum equations⁴, the components of the force are updated at each internal (3-D) time step. Tidal constituents for the harmonic expansion are selected by the user.

The procedure is the following:

- The astronomical Greenwich arguments and nodal factors are updated using the expressions given in Section 5.5 if requested. Otherwise

$$\begin{aligned} f_{qn}(t^{m+1}) &\simeq f_{qn}(t^m) \\ P_{qn}^{m+1} &\simeq P_{qn}^m + \Delta\tau\omega_{qn} \end{aligned} \quad (12.238)$$

where $P_{qn}^m = V_{qn}(t^m) + u_{qn}(t^m)$ and ω_{qn} are the frequencies of the tidal constituents⁵⁶.

⁴The astronomical tidal force can only be taken into account if a spherical grid is selected.

⁵Time is converted to GMT if necessary.

⁶Equation (12.238) is applied in double precision arithmetic to avoid truncation errors.

- The tidal amplitudes A_{qn}^{m+1} are calculated using (5.208).
- The components of the tidal force are determined using (5.207) and (5.206)

$$\begin{aligned}
F_{1;ij}^{t;u} = & -\frac{g_{ij}^u}{h_{1;ij}^u} \left[\frac{3}{2} \sin(2\phi_{ij}^u) \Delta_x^u(\phi_{ij}^c) \sum_{n=1}^{N_0} A_{0n} \cos(P_{0n;ij}) \right. \\
& + \sin(2\phi_{ij}^u) \Delta_x^u(\lambda_{ij}^c) \sum_{n=1}^{N_1} A_{1n} \sin(\lambda_{ij}^u + P_{1n;ij}) \\
& - 2 \cos(2\phi_{ij}^u) \Delta_x^u(\phi_{ij}^c) \sum_{n=1}^{N_1} A_{1n} \cos(\lambda_{ij}^u + P_{1n;ij}) \\
& + (1 + \cos(2\phi_{ij}^u)) \Delta_x^u(\lambda_{ij}^c) \sum_{n=1}^{N_2} A_{2n} \sin(2\lambda_{ij}^u + P_{2n;ij}) \\
& + \sin(2\phi_{ij}^u) \Delta_x^u(\phi_{ij}^c) \sum_{n=1}^{N_2} A_{2n} \cos(2\lambda_{ij}^u + P_{2n;ij}) \\
& + \frac{3}{4} (3 \cos \phi_{ij}^u + \cos(3\phi_{ij}^u)) \Delta_x^u(\lambda_{ij}^c) \sum_{n=1}^{N_3} A_{3n} \sin(3\lambda_{ij}^u + P_{3n;ij}) \\
& \left. + \frac{3}{4} (\sin \phi_{ij}^u + \sin(3\phi_{ij}^u)) \Delta_x^u(\phi_{ij}^c) \sum_{n=1}^{N_3} A_{3n} \cos(3\lambda_{ij}^u + P_{3n;ij}) \right] \\
& \quad (12.239)
\end{aligned}$$

and

$$\begin{aligned}
F_{2;ij}^{t;v} = & -\frac{g_{ij}^v}{h_{2;ij}^v} \left[\frac{3}{2} \sin(2\phi_{ij}^v) \Delta_y^v(\phi_{ij}^c) \sum_{n=1}^{N_0} A_{0n} \cos(P_{0n;ij}) \right. \\
& + \sin(2\phi_{ij}^v) \Delta_y^v(\lambda_{ij}^c) \sum_{n=1}^{N_1} A_{1n} \sin(\lambda_{ij}^v + P_{1n;ij}) \\
& - 2 \cos(2\phi_{ij}^v) \Delta_y^v(\phi_{ij}^c) \sum_{n=1}^{N_1} A_{1n} \cos(\lambda_{ij}^v + P_{1n;ij}) \\
& + (1 + \cos(2\phi_{ij}^v)) \Delta_y^v(\lambda_{ij}^c) \sum_{n=1}^{N_2} A_{2n} \sin(2\lambda_{ij}^v + P_{2n;ij}) \\
& + \sin(2\phi_{ij}^v) \Delta_y^v(\phi_{ij}^c) \sum_{n=1}^{N_2} A_{2n} \cos(2\lambda_{ij}^v + P_{2n;ij}) \\
& \left. + \frac{3}{4} (\sin \phi_{ij}^v + \sin(3\phi_{ij}^v)) \Delta_y^v(\lambda_{ij}^c) \sum_{n=1}^{N_3} A_{3n} \sin(3\lambda_{ij}^v + P_{3n;ij}) \right. \\
& \left. + \frac{3}{4} (3 \cos \phi_{ij}^v + \cos(3\phi_{ij}^v)) \Delta_y^v(\phi_{ij}^c) \sum_{n=1}^{N_3} A_{3n} \cos(3\lambda_{ij}^v + P_{3n;ij}) \right]
\end{aligned}$$

$$\begin{aligned}
& + \frac{3}{4}(3 \cos \phi_{ij}^v + \cos(3\phi_{ij}^v)) \Delta_y^v(\lambda_{ij}^c) \sum_{n=1}^{N_3} A_{3n} \sin(3\lambda_{ij}^v + P_{3n;ij}) \\
& + \frac{3}{4}(\sin \phi_{ij}^v + \sin(3\phi_{ij}^v)) \Delta_y^v(\phi_{ij}^c) \sum_{n=1}^{N_3} A_{3n} \cos(3\lambda_{ij}^v + P_{3n;ij}) \Big] \\
\end{aligned} \tag{12.240}$$

12.3.15 Surface and bottom boundary conditions

12.3.15.1 surface boundary conditions

Application of the surface boundary conditions (5.226) and (5.224) gives

$$F_{13;ij,N_z+1} = 0, \quad F_{23;ij,N_z+1} = 0 \tag{12.241}$$

and

$$D_{13;ij,N_z+1} = \tau_{s1;ij}^u, \quad D_{23;ij,N_z+1} = \tau_{s2;ij}^v \tag{12.242}$$

for respectively the vertical advective and diffusive fluxes of momentum at the surface.

The components of the surface stress are calculated as function of meteorological variables. Different options are available and discussed in Section 5.7.

The following steps are taken

- The meteorological data are interpolated at the C-nodes of the model grid.
- The surface drag coefficient C_{ds} and the components of the surface stress are determined at the C-nodes.
- The stress components are interpolated at respectively the U- and V-node.

In the case that the surface drag and exchange are calculated using the Monin-Obukhov formulation, described in Section 5.7.3, the system of four equations (5.288), (5.293), (5.294) and (5.295) needs to be solved at each model grid point and time step using a time-consuming iteration procedure. The following simplifying procedure is taken in the program

- The equations generally depend on five meteorological variables: the magnitude of the surface wind W , air temperature T_a , sea surface temperature T_s , relative humidity RH and atmospheric pressure P_a . The following simplifications are made

- The atmospheric pressure enters the equations only indirectly to evaluate the surface humidity and is taken as constant in the equations, i.e. $P_a = P_{ref}$.
- Tests showed a larger dependence on the air minus sea temperature difference ΔT than on the individual values of T_a and T_s itself. Letting $T_s = T_{s;ref}$ one has $T_a = T_{s;ref} + \Delta T$.

The equations now only depend on three variables: W , ΔT , RH

- The equations are solved for C_{ds} , C_e , C_h on a “discretised” grid at the initial time

$$\begin{aligned} W &= W_{min} + l\delta W \quad \text{for} \quad l = 0, N_W; \quad N_W = \frac{W_{max} - W_{min}}{\delta W} \\ \Delta T &= \Delta T_{min} + m\delta(\Delta T) \quad \text{for} \quad m = 0, N_T; \quad N_T = \frac{\Delta T_{max} - \Delta T_{min}}{\delta(\Delta T)} \\ RH &= RH_{min} + n\delta RH \quad \text{for} \quad n = 0, N_R; \quad N_R = \frac{RH_{max} - RH_{min}}{\delta R} \end{aligned} \quad (12.243)$$

The following default values are taken

$$\begin{aligned} W_{min} &= 3, \quad W_{max} = 50, \quad \delta W = 0.25 \quad (\text{m/s}) \\ \Delta T_{min} &= -5, \quad \Delta T_{max} = 5, \quad \delta(\Delta T) = 1 \quad (\text{°C}) \\ RH_{min} &= 0.5, \quad RH_{max} = 1.0, \quad \delta R = 0.05 \end{aligned} \quad (12.244)$$

The lower limit $W_{min}=3$ m/s is taken since the equations diverge in the case of the free-convection limit $W \rightarrow 0$ and $\Delta T > 0$. The computed values are stored in 3-D arrays.

- The values of the drag and exchange coefficients are then obtained at a specific time by a tri-linear interpolation from the discretised values. Extrapolation is used if necessary.

12.3.15.2 bottom boundary conditions

The bottom boundary condition for vertical advection is the same as the one applied at surface (see equation (5.324)) so that

$$F_{13;ij1} = 0, \quad F_{23;ij1} = 0 \quad (12.245)$$

If the bottom stress is parameterised using the bottom values of the 3-D current, vertical diffusion is treated implicitly. The flux bottom boundary conditions (5.319) are then discretised as

$$D_{13;ij1} = \theta_v k_{b;ij}^{n;u} u_{ij1}^{n+1} + (1 - \theta_v) k_{b;ij}^{n;u} u_{ij1}^n \quad (12.246)$$

$$D_{23;ij1} = \theta_v k_{b;ij}^{n;v} v_{ij1}^{n+1} + (1 - \theta_v) k_{b;ij}^{n;v} v_{ij1}^n \quad (12.247)$$

where θ_v is the implicitity factor for vertical diffusion and the friction velocities are defined by

$$\begin{aligned} \text{no bottom stress (5.316)} &: k_{b;ij}^u = k_{b;ij}^v = 0 \\ \text{linear bottom stress (5.317)} &: k_{b;ij}^u = k_{b;ij}^v = k_{lin} \\ \text{3-D quadratic law (5.319)} &: k_{b;ij}^u = C_{db;ij}^u \left((u_{ij1}^n)^2 + (v_{ij1}^{n;u})^2 \right)^{1/2} \\ &: k_{b;ij}^v = C_{db;ij}^v \left((u_{ij1}^n)^2 + (v_{ij1}^n)^2 \right)^{1/2} \end{aligned} \quad (12.248)$$

If the bottom stress is expressed in terms of the depth mean current, the bottom flux is discretised explicitly

$$D_{13;ij1} = k_{b;ij}^{n;u} \bar{u}_{ij}^n \quad (12.249)$$

$$D_{23;ij1} = k_{b;ij}^{n;v} \bar{v}_{ij}^n \quad (12.250)$$

where the friction velocity is given by (12.248) in the absence of a bottom stress or a linear friction law and

$$\begin{aligned} k_{b;ij}^u &= C_{db;ij}^u \left((\bar{u}_{ij}^n)^2 + (\bar{v}_{ij}^{n;u})^2 \right)^{1/2} \\ k_{b;ij}^v &= C_{db;ij}^v \left((\bar{u}_{ij}^{n;v})^2 + (\bar{v}_{ij}^n)^2 \right)^{1/2} \end{aligned} \quad (12.251)$$

The bottom drag coefficient is discretised using (12.15)-(12.16) or specified externally.

12.3.16 Lateral boundary conditions for the 2-D mode

12.3.16.1 open boundary conditions for transports

Open boundary conditions for the 2-D mode are discussed in Section 5.10.1. The aim is to provide values of U at U- and of V at V-open boundaries. External data can be generally expressed as the sum of a non-harmonic and an harmonic part as given by (5.333). The expression is updated at each time step for the requested locations and variables (U , V or ζ) depending on the type of conditions, prior to the application of the boundary conditions itself.

The tidal constituents are selected by the user. In analogy with the astronomical tide (see Section 12.3.14), the space-independent astronomical

phases are calculated by using either the expressions in Section 5.5 with time converted to GMT if needed, or the linear approximation

$$P_l^{m+1} = V_l^{m+1} + u_l^{m+1} \simeq V_l^m + u_l^m + \omega_l \delta\tau \quad (12.252)$$

where ω_l are the tidal frequencies. Note that (12.252) is evaluated in double precision to avoid truncation errors for long integration periods.

The following notations are introduced

- \pm or \mp : The upper (lower) sign applies at a western/southern (eastern/northern) open boundary.
- $X_{i:i-1,j}$: The quantity X is evaluated at grid point (i,j) for a western and at $(i-1,j)$ for an eastern boundary.
- $Y_{i,j:j-1}$: The quantity Y is evaluated at grid point (i,j) for a southern and at $(i,j-1)$ for a northern boundary.
- U_{ij}^e , V_{ij}^e , ζ_{ij}^e denote externally specified values (harmonic or time series data).
- s_{ij} equals 1 if ζ_{ij}^e is defined at an exterior C-node and 2 if ζ_{ij}^e is defined at an open boundary U- or V-node.
- The gravity wave speed at the C-node nearest to the U- or V-open boundary location (i,j) is defined by

$$c_{ij}^u = \sqrt{g_{i:i-1,j}^c H_{i:i-1,j}^c}, \quad c_{ij}^v = \sqrt{g_{i,j:j-1}^c H_{i,j:j-1}^c} \quad (12.253)$$

- The following auxiliary parameters are defined at U- or V-nodes

$$\alpha_{ij}^u = \frac{\Delta\tau c_{ij}^u}{h_{1;i:i-1,j}^c}, \quad \alpha_{ij}^v = \frac{\Delta\tau c_{ij}^v}{h_{2;i,j:j-1}^c} \quad (12.254)$$

$$r_{ij}^u = \frac{h_{1;i:i-1,j}^c}{h_{1;ij}^u}, \quad r_{ij}^v = \frac{h_{2;i,j:j-1}^c}{h_{2;ij}^v} \quad (12.255)$$

The discretised versions of all available open boundary conditions are listed below using the same numbering system as in Section 5.10.1.

0. Clamped (see equation (5.337)).

$$U_{ij}^{m+1} = U_{ij}^m, \quad V_{ij}^{m+1} = V_{ij}^m \quad (12.256)$$

1. Zero slope (see equation (5.338)).

$$\begin{aligned} U_{ij}^{m+1} &= U_{ij}^m + \Delta\tau \left(f_{ij}^u (\theta_c V_{i:i-1,j}^{m+1;c} + (1 - \theta_c) V_{i:i-1,j}^{m;c}) \right. \\ &\quad \left. + H_{ij}^{m+1;u} F_{1;ij}^{t;m+1} + \tau_{s1;ij}^c - \tau_{b1;ij}^{n;u} \right) \end{aligned} \quad (12.257)$$

$$\begin{aligned} V_{ij}^{m+1} &= V_{ij}^m - \Delta\tau \left(f_{ij}^v (\theta_c U_{i,j:j-1}^{m+1;c} + (1 - \theta_c) U_{i,j:j-1}^{m;c}) \right. \\ &\quad \left. - H_{ij}^{m+1;v} F_{2;ij}^{t;m+1} - \tau_{s2;ij}^c + \tau_{b2;ij}^{n;v} \right) \end{aligned} \quad (12.258)$$

Note that the bottom stress components are evaluated at the old time t^n .

2. Zero volume flux (see equation (5.339)).

$$U_{ij}^{m+1} = U_{i+1:i-1,j}^{m+1}, \quad V_{ij}^{m+1} = V_{i,j+1:j-1}^{m+1} \quad (12.259)$$

3. Specified elevation (see (equation 5.340)).

$$\begin{aligned} U_{ij}^{m+1} &= U_{ij}^{L;m+1} \\ &= U_{ij}^{L;m} \mp s_{ij}^u \alpha_{ij}^u c_{ij}^u r_{ij}^u (\zeta_{i:i-1,j}^{m+1} - \zeta_{ij}^e) \\ &\quad + \Delta\tau \left(f_{ij}^u (\theta_c V_{i:i-1,j}^{m+1;c} + (1 - \theta_c) V_{i:i-1,j}^{m;c}) \right. \\ &\quad \left. + H_{ij}^{m+1;u} F_{1;ij}^{t;m+1} + \tau_{s1;ij}^c - \tau_{b1;ij}^{n;u} \right) \end{aligned} \quad (12.260)$$

$$\begin{aligned} V_{ij}^{m+1} &= V_{ij}^{L;m+1} \\ &= V_{ij}^{L;m} \mp s_{ij}^v \alpha_{ij}^v c_{ij}^v r_{ij}^v (\zeta_{i,j:j-1}^{m+1} - \zeta_{ij}^e) \\ &\quad - \Delta\tau \left(f_{ij}^v (\theta_c U_{i,j:j-1}^{m+1;c} + (1 - \theta_c) U_{i,j:j-1}^{m;c}) \right. \\ &\quad \left. - H_{ij}^{m+1;v} F_{2;ij}^{t;m+1} - \tau_{s2;ij}^c + \tau_{b2;ij}^{n;v} \right) \end{aligned} \quad (12.261)$$

4. Specified transport (see equation (5.341)).

$$U_{ij}^{m+1} = U_{ij}^e, \quad V_{ij}^{m+1} = V_{ij}^e \quad (12.262)$$

5. Radiation condition using shallow water speed (see equation (5.343)).

$$\begin{aligned} U_{ij}^{m+1} &= \frac{U_{ij}^m + \alpha_{ij}^u U_{i+1:i-1,j}^{m+1}}{1 + \alpha_{ij}^u} \\ V_{ij}^{m+1} &= \frac{V_{ij}^m + \alpha_{ij}^v V_{i,j+1:j-1}^{m+1}}{1 + \alpha_{ij}^v} \end{aligned} \quad (12.263)$$

6. Orlanski (1976) condition (see equation (5.344)).

$$U_{ij}^{m+1} = \left(1 - O_R(r_{1;ij}^u, r_{2;ij}^u, r_{3;ij}^u)\right)U_{ij}^m + O_R(r_{1;ij}^u, r_{2;ij}^u, r_{3;ij}^u)U_{i+1:i-1,j}^m \quad (12.264)$$

$$V_{ij}^{m+1} = \left(1 - O_R(r_{1;ij}^v, r_{2;ij}^v, r_{3;ij}^v)\right)V_{ij}^m + O_R(r_{1;ij}^v, r_{2;ij}^v, r_{3;ij}^v)V_{i,j+1:j-1}^m \quad (12.265)$$

The Orlanski weight function is defined by

$$\begin{aligned} O_R(r_1, r_2, r_3) &= \min\left(\max\left(\frac{r_1 - r_2}{r_3 - r_2}, 0\right), 1\right) && \text{if } r_2 \neq r_3 \\ O_R(r_1, r_2, r_3) &= 0 && \text{if } r_2 = r_3 \text{ and } r_1 \leq r_2 \\ O_R(r_1, r_2, r_3) &= 1 && \text{if } r_2 = r_3 \text{ and } r_1 > r_2 \end{aligned} \quad (12.266)$$

The arguments of the weight functions are defined by

$$r_{1;ij}^u = U_{i+1:i-1,j}^m, \quad r_{2;ij}^u = U_{i+1:i-1,j}^{m-1}, \quad r_{3;ij}^u = U_{i+2:i-2,j}^{m-1} \quad (12.267)$$

$$r_{1;ij}^v = V_{i,j+1:j-1}^m, \quad r_{2;ij}^v = V_{i,j+1:j-1}^{m-1}, \quad r_{3;ij}^v = V_{i,j+2:j-2}^{m-1} \quad (12.268)$$

7. Camerlengo & O'Brien (1980).

$$\begin{aligned} U_{ij}^{m+1} &= U_{i+1:i-1,j}^m && \text{if } U_{i+1:i-1,j}^m \geq U_{i+2:i-2,j}^{m-1} \\ U_{ij}^{m+1} &= U_{ij}^m && \text{otherwise} \end{aligned} \quad (12.269)$$

$$\begin{aligned} V_{ij}^{m+1} &= V_{i,j+1:j-1}^m && \text{if } V_{i,j+1:j-1}^m \geq V_{i,j+2:j-2}^{m-1} \\ V_{ij}^{m+1} &= V_{ij}^m && \text{otherwise} \end{aligned} \quad (12.270)$$

8. Flather (1976) with specified elevation and transport (see equation (5.346)).

$$\begin{aligned} U_{ij}^{m+1} &= U_{ij}^e \mp \frac{1}{2} s_{ij}^u c_{ij}^u (\zeta_{i:i-1,j}^{m+1} - \zeta_{ij}^e) \\ V_{ij}^{m+1} &= V_{ij}^e \mp \frac{1}{2} s_{ij}^v c_{ij}^v (\zeta_{i,j:j-1}^{m+1} - \zeta_{ij}^e) \end{aligned} \quad (12.271)$$

9. Flather (1976) with specified elevation (see equation (5.347)).

$$\begin{aligned} U_{ij}^{m+1} &= U_{ij}^{L;m+1} \mp \frac{1}{2} s_{ij}^u c_{ij}^u (\zeta_{i:i-1,j}^{m+1} - \zeta_{ij}^e) \\ V_{ij}^{m+1} &= V_{ij}^{L;m+1} \mp \frac{1}{2} s_{ij}^v c_{ij}^v (\zeta_{i,j:j-1}^{m+1} - \zeta_{ij}^e) \end{aligned} \quad (12.272)$$

where U^L, V^L are the local solutions obtained from (12.260) and (12.261).

10. Røed & Smedstad (1984) (see equations (5.348)–(5.349)).

The local solution for ζ is determined from

$$\zeta_{ij}^{L;m+1} = \zeta_{ij}^{L;m} - \frac{\Delta\tau(h_{1;i:i-1,j+1}^v V_{i:i-1,j+1}^m - h_{1;i:i-1,j}^v V_{i:i-1,j}^m)}{h_{1;i:i-1,j}^c h_{2;i:i-1,j}^c} \quad (12.273)$$

at U-nodes, and

$$\zeta_{ij}^{L;m+1} = \zeta_{ij}^{L;m} - \frac{\Delta\tau(h_{2;i+1,j:j-1}^u U_{i+1,j:j-1}^m - h_{2;i,j:j-1}^u U_{i,j:j-1}^m)}{h_{1;i,j:j-1}^c h_{2;i,j:j-1}^c} \quad (12.274)$$

at V-nodes.

The transports U and V are obtained using

$$U_{ij}^{m+1} = U_{ij}^{L;m+1} \mp c_{ij}^u (\zeta_{i:i-1,j}^{m+1} - \zeta_{ij}^{L;m+1}) \quad (12.275)$$

$$V_{ij}^{m+1} = V_{ij}^{L;m+1} \mp c_{ij}^v (\zeta_{i,j:j-1}^{m+1} - \zeta_{ij}^{L;m+1}) \quad (12.276)$$

where the local solutions $U_{ij}^{L;m+1}$, $U_{ij}^{L;m+1}$ are given by (12.260) and (12.261).

11. Characteristic method with specified elevation and transport.

Using the notations of Section 5.10.1 the transports are calculated as the average between the incoming (R_i) and outgoing (R_o) characteristic

$$\begin{aligned} U_{ij}^{m+1} &= \frac{1}{2}(R_{i;ij}^{m+1;u} + R_{o;ij}^{m+1;u}) \\ V_{ij}^{m+1} &= \frac{1}{2}(R_{i;ij}^{m+1;v} + R_{o;ij}^{m+1;v}) \end{aligned} \quad (12.277)$$

The incoming characteristic R_i is defined by equations (5.353) using prescribed values for transports and elevations

$$\begin{aligned} R_{i;ij}^{m+1;u} &= U_{ij}^e \pm \frac{1}{2}c_{ij}^u s_{ij}^u \left(\zeta_{ij}^e + (2 - s_{ij}^u) \zeta_{i:i-1,j}^{m+1} \right) \\ R_{i;ij}^{m+1;v} &= V_{ij}^e \pm \frac{1}{2}c_{ij}^v s_{ij}^v \left(\zeta_{ij}^e + (2 - s_{ij}^v) \zeta_{i,j:j-1}^{m+1} \right) \end{aligned} \quad (12.278)$$

The outgoing characteristic R_o is obtained by solving the discretised versions of equations (5.351)–(5.352):

$$\begin{aligned} (1 + \frac{3}{2}\alpha_{ij}^u r_{ij}^u) R_{o;ij}^{m+1;u} &= R_{o;ij}^{m;u} \\ &+ \alpha_{ij}^u r_{ij}^u (U_{i+1;i-1,j}^{m+1} + \frac{1}{2}R_{i;ij}^{m+1;u} \mp 2c_{ij}^u \zeta_{i:i-1,j}^{m+1}) \end{aligned}$$

$$\begin{aligned}
& + \frac{\alpha_{ij}^u}{h_{2;i:i-1,j}^c} \left(\pm (h_{1;i:i-1,j+1}^v V_{i:i-1,j+1}^m - h_{1;i:i-1,j}^v V_{i:i-1,j}^m) \right. \\
& + U_{i:i-1,j}^{m;c} (h_{2;i+1:i-1,j}^u - h_{2;ij}^u) \Big) \\
& + \Delta\tau \left(f_{ij}^u (\theta_c V_{i:i-1,j}^{m+1;c} + (1 - \theta_c) V_{i:i-1,j}^{m;c}) + H_{ij}^{m+1;u} F_{1;ij}^{t;m+1} + \tau_{s1;ij}^c - \tau_{b1;ij}^{n;u} \right) \\
\end{aligned} \tag{12.279}$$

and

$$\begin{aligned}
& (1 + \frac{3}{2} \alpha_{ij}^v r_{ij}^v) R_{o;ij}^{m+1;v} = R_{o;ij}^{m;v} \\
& + \alpha_{ij}^v r_{ij}^v (V_{i,j+1;j-1}^{m+1} + \frac{1}{2} R_{i;ij}^{m+1;v} \mp 2c_{ij}^v \zeta_{i,j:j-1}^{m+1}) \\
& + \frac{\alpha_{ij}^v}{h_{1;i,j:j-1}^c} \left(\pm (h_{2;i+1,j:j-1}^u U_{i+1,j:j-1}^m - h_{2;i,j:j-1}^u U_{i,j:j-1}^m) \right. \\
& + V_{i,j:j-1}^{m;c} (h_{1;i,j+1:j-1}^v - h_{1;ij}^v) \Big) \\
& - \Delta\tau \left(f_{ij}^v (\theta_c U_{i,j:j-1}^{m+1;c} + (1 - \theta_c) U_{i,j:j-1}^{m;c}) - H_{ij}^{m+1;v} F_{2;ij}^{t;m+1} - \tau_{s2;ij}^c + \tau_{b2;ij}^{n;v} \right) \\
\end{aligned} \tag{12.280}$$

Note that the propagation term is integrated fully implicitly and U_{ij}^{m+1} , V_{ij}^{m+1} have been eliminated in (12.279)–(12.280) by substituting for $R_i^{m+1;u}$ and $R_i^{m+1;v}$ from (12.278).

12. Characteristic method with specified elevation.

The method is as previous with U_{ij}^e , V_{ij}^e replaced by the local solutions $U_{ij}^{L;m+1}$, $V_{ij}^{L;m+1}$ from (12.260) and (12.261).

13. Characteristic method using zero normal gradient.

The method is the same as previous except that the incoming characteristics are obtained as solutions of the discretised versions of equations (5.355)–(5.356):

$$\begin{aligned}
R_{i;ij}^{m+1;u} & = R_{i;ij}^{m;u} \\
& - \frac{\alpha_{ij}^u}{h_{2;i:i-1,j}^c} \left(\pm (h_{1;i:i-1,j+1}^v V_{i:i-1,j+1}^m - h_{1;i:i-1,j}^v V_{i:i-1,j}^m) \right. \\
& + U_{i:i-1,j}^{c;m} (h_{2;i+1:i-1,j}^u - h_{2;ij}^u) \Big) \\
& + \Delta\tau \left(f_{ij}^u (\theta_c V_{i:i-1,j}^{m+1;c} + (1 - \theta_c) V_{i:i-1,j}^{m;c}) + H_{ij}^{m+1;u} F_{1;ij}^{t;m+1} + \tau_{s1;ij}^c - \tau_{b1;ij}^{n;u} \right) \\
\end{aligned}$$

(12.281)

and

$$\begin{aligned}
 R_{i;ij}^{m+1;v} &= R_{i;ij}^{m;v} \\
 &- \frac{\alpha_{ij}^v}{h_{1;i,j;j-1}^c} \left(\pm (h_{2;i+1,j;j-1}^u U_{i+1,j;j-1}^m - h_{2;i,j;j-1}^u U_{i,j;j-1}^m) \right. \\
 &\left. + V_{i,j;j-1}^{c;m} (h_{1;i,j+1;j-1}^v - h_{1;ij}^v) \right) \\
 &- \Delta\tau \left(f_{ij}^v (\theta_c U_{i,j;j-1}^{m+1;c} + (1 - \theta_c) U_{i,j;j-1}^{m;c}) - H_{ij}^{m+1;v} F_{2;ij}^{t;m+1} - \tau_{s2;ij}^c + \tau_{b2;ij}^{n;v} \right)
 \end{aligned} \tag{12.282}$$

14. Specified depth-mean current

$$U_{ij}^{m+1} = H_{ij}^u \bar{u}_{ij}^e, \quad V_{ij}^{m+1} = H_{ij}^v \bar{v}_{ij}^e \tag{12.283}$$

15. Specified discharge

$$U_{ij}^{m+1} = Q_{ij}^e / h_{2;ij}^u, \quad V_{ij}^{m+1} = Q_{ij}^e / h_{1;ij}^v \tag{12.284}$$

16. Distributed discharge

From (5.359) one obtains

$$\begin{aligned}
 U_{ij;l} &= \frac{H_{ij}^{1.5} C_{z;ij}}{\sum_{m=1}^{N_l} h_{2;ij;m} H_{ij;m}^{1.5} C_{z;ij;m}} Q_l \\
 V_{ij;l} &= \frac{H_{ij}^{1.5} C_{z;ij}}{\sum_{m=1}^{N_l} h_{1;ij;m} H_{ij;m}^{1.5} C_{z;ij;m}} Q_l
 \end{aligned} \tag{12.285}$$

where l is the index of the open boundary section where the discharge Q_l is imposed.

17. Imposed surface slope

The equations are similar to (12.260)–(12.261) for specified surface elevation, except that the slope is now externally specified

$$\begin{aligned}
 U_{ij}^{m+1} &= U_{ij}^{L;m+1} \\
 &= U_{ij}^{L;m} - \Delta\tau g_{ij}^u H_{ij}^u \left(\frac{1}{h_{ij;1}} \frac{\partial \zeta}{\partial \xi_1} \right)^e \\
 &\quad + \Delta\tau \left(f_{ij}^u (\theta_c V_{i:i-1,j}^{m+1;c} + (1 - \theta_c) V_{i:i-1,j}^{m;c}) \right)
 \end{aligned}$$

$$+ H_{ij}^{m+1;u} F_{1;ij}^{t;m+1} + \tau_{s1;ij}^c - \tau_{b1;ij}^{n;u} \Big) \quad (12.286)$$

$$\begin{aligned} V_{ij}^{m+1} &= V_{ij}^{L;m+1} \\ &= V_{ij}^{L;m} - \Delta\tau g_{ij}^v H_{ij}^v \left(\frac{1}{h_{ij;2}} \frac{\partial\zeta}{\partial\xi_2} \right)^e \\ &\quad - \Delta\tau \left(f_{ij}^v (\theta_c U_{i,j;j-1}^{m+1;c} + (1 - \theta_c) V_{i,j;j-1}^{m;c}) \right. \\ &\quad \left. - H_{ij}^{m+1;v} F_{2;ij}^{t;m+1} - \tau_{s2;ij}^c + \tau_{b2;ij}^{n;v} \right) \end{aligned} \quad (12.287)$$

12.3.16.2 open boundary conditions for 2-D advective and diffusive fluxes

Three schemes are available to evaluate the cross-stream advective fluxes in the U -equation at Y-open boundaries or in the V -equation at X-open boundaries

1. The first one uses a zero gradient condition

$$\bar{F}_{12;ij}^{uv} = \bar{F}_{12;i,j+1:j-1}^{uv} \quad \text{or} \quad \bar{F}_{21;ij}^{uv} = \bar{F}_{21;i+1:i-1,j}^{uv} \quad (12.288)$$

which is the same as before.

2. The flux is determined using the upwind scheme (where possible). This means that

$$\begin{aligned} \bar{F}_{12;ij}^{uv} &= \frac{1}{2} \bar{v}_{ij}^{uv} \left((1 + s_{ij}) U_{i,j-1:j} + (1 - s_{ij}) U_{i,j:j-1} \right) \quad \text{or} \\ \bar{F}_{21;ij}^{uv} &= \frac{1}{2} \bar{u}_{ij}^{uv} \left((1 + s_{ij}) V_{i-1:i,j} + (1 - s_{ij}) V_{i:i-1,j} \right) \end{aligned} \quad (12.289)$$

where $s_{ij} = 1$ in case of an inflow condition and either

- $(i,j-1:j)$ is a U-open boundary or $(i-1:i,j)$ is a V-open boundary
- $(i-1,j)$ is a closed (land or coastal) V-node or $(i,j-1)$ is a closed (land or coastal) U-node
- (i,j) is a closed V-node or (i,j) is a closed U-node.

In all other cases, $s_{ij} = -1$.

3. A tangential open boundary condition is imposed. As in the previous case the upwind scheme is used, but an external value of the transport is specified at an external U- or V- node for the cross-stream fluxes at respectively Y- or X-nodes. This means that

$$\begin{aligned}\bar{F}_{12;ij}^{uv} &= \frac{1}{2}\bar{v}_{ij}^{uv}\left((1 \pm s_{ij})U_{i,j-1:j}^e + (1 \mp s_{ij})U_{i,j:j-1}\right) \quad \text{or} \\ \bar{F}_{21;ij}^{uv} &= \frac{1}{2}\bar{u}_{ij}^{uv}\left((1 \pm s_{ij})V_{i-1:i,j}^e + (1 \mp s_{ij})V_{i:i-1,j}\right)\end{aligned}\quad (12.290)$$

where s_{ij} is the sign of the advecting current (\bar{v}_{ij}^{uv} or \bar{u}_{ij}^{uv}) at the corner node and the upper (lower) sign applies at a western/southern (eastern/northern) open boundary.

The cross-stream diffusive fluxes in the U -equation are evaluated as follows

- If either $(i,j-1:j)$ is a U-open boundary, or $(i-1,j)$ is a closed (land or coastal) V-node, or (i,j) is a closed V-node, the flux is calculated by equation (12.196) for an internal node.
- Otherwise, if $(i,j-1:j)$ is an interior U-node, a zero gradient condition is used

$$\bar{D}_{12;ij}^{uv} = \bar{D}_{12;i,j+1:j-1}^{uv} \quad (12.291)$$

- Otherwise, the flux is set to zero, i.e.

$$\bar{D}_{12;ij}^{uv} = 0 \quad (12.292)$$

Likewise, at the fluxes in the V -equation are given by

- If either $(i-1:i,j)$ is a V-open boundary, or $(i,j-1)$ is a closed (land or coastal) U-node, or (i,j) is a closed U-node, the flux is calculated by equation (12.198) for an internal node.
- Otherwise, if $(i-1:i,j)$ is an interior V-node, a zero gradient condition is used

$$\bar{D}_{21;ij}^{uv} = \bar{D}_{21;i,j+1:j-1}^{uv} \quad (12.293)$$

- Otherwise, the flux is set to zero, i.e.

$$\bar{D}_{21;ij}^{uv} = 0 \quad (12.294)$$

An optional relaxation scheme has been implemented which reduces the impact of advection within a user-defined distance from the open boundaries. In that case, the advective terms are multiplied by the relaxation factor

$$\alpha_{or} = \min(d/d_{max}, 1) \quad (12.295)$$

where d is the distance to the nearest open boundary. Experiments showed that, with an appropriate choice of the maximum relaxation distance d_{max} , instabilities, due to inaccuracies at the open boundaries, are prevented to propagate into the domain. The scheme has shown to be useful, in particular, to reduce instabilities, observed near ragged open boundaries.

12.3.16.3 boundary conditions at closed lateral boundaries

Following (5.383)–(5.384) one has

$$U_{ij} = 0, \quad V_{ij} = 0 \quad (12.296)$$

at coastal boundaries.

Likewise all fluxes for the cross-advection and diffusive terms are set to zero at closed Y- or X-node boundaries:

$$\bar{F}_{12;ij}^{uv} = 0.0, \quad \bar{D}_{12;ij}^{uv} = 0.0, \quad \bar{F}_{21;ij}^{uv} = 0.0, \quad \bar{D}_{21;ij}^{uv} = 0.0 \quad (12.297)$$

where a Y- or X-node grid point is called “closed” if one of the neighbouring V- or U-node points in the X- or Y-direction is a closed velocity node.

12.3.17 Lateral boundary conditions for the 3-D currents

12.3.17.1 open boundary conditions for horizontal 3-D currents

Open boundary conditions for the 3-D mode are discussed in Section 5.10.2.1. The aim is to provide values of u at U- and of v at V-open boundaries for each vertical level. The depth-mean parts of the currents are already determined by the 2-D open boundary conditions which means than only the baroclinic parts δu and δv need to be specified.

The discretised versions of all open available open boundary conditions are listed below using the same numbering system as in Section 5.10.2.1.

0. Zero gradient condition (see equation (5.365)).

$$\delta u_{ijk}^{n+1} = \frac{h_{2;i+1:i-1,j}^u h_{3;i+1:i-1,jk}^{n+1;u}}{h_{2;ij}^u h_{3;ijk}^{n+1;u}} \delta u_{i+1:i-1,jk}^{n+1;u} \quad (12.298)$$

$$\delta v_{ijk}^{n+1} = \frac{h_{1;i,j+1:j-1}^v h_{3;i,j+1:j-1,k}^{n+1;v}}{h_{1;ij}^v h_{3;ijk}^{n+1;v}} \delta v_{i,j+1:j-1,k}^{n+1;v} \quad (12.299)$$

This is the default condition.

1. Specified external profile (see equation (5.366)).

$$\delta u_{ijk}^{n+1} = \delta u_{ijk}^e \quad (12.300)$$

$$\delta v_{ijk}^{n+1} = \delta v_{ijk}^e \quad (12.301)$$

$$(12.302)$$

2. Second order gradient condition (see equation (5.367))

$$\begin{aligned} \delta u_i &= \frac{h_{2;i+1:i-1}^u}{h_{2;i}^u} \frac{h_{3;i+1:i-1}^u}{h_{3;i}^u} \left(1 + \frac{h_{1;i:i-1}^c}{h_{1;i+1:i-2}^c} \frac{h_{2;i:i-1}^c}{h_{2;i+1:i-2}^c} \right) \delta u_{i+1:i-1} \\ &- \frac{h_{2;i+2:i-2}^u}{h_{2;i}^u} \frac{h_{3;i+2:i-2}^u}{h_{3;i}^u} \frac{h_{1;i:i-1}^c}{h_{1;i+1:i-2}^c} \frac{h_{2;i:i-1}^c}{h_{2;i+1:i-2}^c} \delta u_{i+2:i-2} \end{aligned} \quad (12.303)$$

$$\begin{aligned} \delta v_i &= \frac{h_{1;j+1:j-1}^v}{h_{1;j}^v} \frac{h_{3;j+1:j-1}^v}{h_{3;j}^v} \left(1 + \frac{h_{2;j:j-1}^c}{h_{2;j+1:j-2}^c} \frac{h_{1;j:j-1}^c}{h_{1;j+1:j-2}^c} \right) \delta v_{j+1:j-1} \\ &- \frac{h_{1;j+2:j-2}^v}{h_{1;j}^v} \frac{h_{3;j+2:j-2}^v}{h_{3;j}^v} \frac{h_{2;j:j-1}^c}{h_{2;j+1:j-2}^c} \frac{h_{1;j:j-1}^c}{h_{1;j+1:j-2}^c} \delta v_{j+2:j-2} \end{aligned} \quad (12.304)$$

$$(12.305)$$

3. Local solution

$$\frac{h_{3;ik}^{n+1;u} \delta u_{ik}^{n+1} - h_{3;ik}^{n;u} \delta u_{ik}^n}{h_{3;ik}^{n+1;u}} = f \left(\theta_c v_{ik}^{n+1;c} - (1 - \theta_c) v_{ik}^{n;c} \right) \quad (12.306)$$

$$+ F_{1;i+1:i-1,k}^{b;n} - \frac{\bar{F}_{1;i+1:i-1,k}^{b;n}}{H_{i+1:i-1}^{n+1;u}} + \frac{D_{13;ij,k+1}^{uw} - D_{13;ik}^{uw}}{h_{3;i}^u} + \frac{\tau_{b1;i} - \tau_{s1;i}}{H_i^{n+1;u}}$$

$$\frac{h_{3;jk}^{n+1;v} \delta v_{jk}^{n+1;c} - h_{3;jk}^{n;v} \delta v_{jk}^{n;c}}{h_{3;jk}^{n+1;v}} = -f \left(\theta_c u_{jk}^{n+1;c} - (1 - \theta_c) u_{jk}^{n;c} \right) \quad (12.307)$$

$$+ F_{2;j+1:j-1,k}^{b;n} - \frac{\bar{F}_{2;j+1:j-1,k}^{b;n}}{H_{j+1:j-1}^{n+1;v}} + \frac{D_{23;j,k+1}^{vw} - D_{23;jk}^{vw}}{h_{3;jk}^v} + \frac{\tau_{b2;j} - \tau_{s2;j}}{H_j^{n+1;v}}$$

The diffusive fluxes are obtained from (12.207), (12.209) using the condition (12.242) at the surface and the bottom stress formulations given in Section 12.3.15.2.

4. Radiation condition using the baroclinic wave speed (see equation (5.370)).

$$\delta u_{ijk}^{n+1} = (1 - w_{ijk}^u) \delta u_{ijk}^n + w_{ijk}^u \delta u_{i+1:i-1,jk}^n \quad (12.308)$$

$$\delta v_{ijk}^{n+1} = (1 - w_{ijk}^v) \delta v_{ijk}^n + w_{ijk}^v \delta v_{i,j+1:j-1,k}^n \quad (12.309)$$

The weight factors are given by

$$\begin{aligned} w_{ijk}^u &= \pm \frac{R \Delta t}{h_{1;i:i-1,j}^c} \sqrt{g_{i:i-1,j}^c H_{i:i-1,j}^{n+1;c}} \\ w_{ijk}^v &= \pm \frac{R \Delta t}{h_{2;i,j:j-1}^c} \sqrt{g_{i,j:j-1}^c H_{i,j:j-1}^{n+1;c}} \end{aligned} \quad (12.310)$$

where R is the prescribed ratio of the baroclinic to surface gravity wave speed. Default value is 0.03.

5. Orlanski condition (see equation (5.371)).

In analogy with the 2-D case one has

$$\delta u_{ijk}^{n+1} = \left(1 - O_R(r_{1;ijk}^u, r_{2;ijk}^u, r_{3;ijk}^u) \right) \delta u_{ijk}^n + O_R(r_{1;ijk}^u, r_{2;ijk}^u, r_{3;ijk}^u) \delta u_{i+1:i-1,jk}^n \quad (12.311)$$

$$\delta v_{ijk}^{n+1} = \left(1 - O_R(r_{1;ijk}^v, r_{2;ijk}^v, r_{3;ijk}^v) \right) \delta v_{ijk}^n + O_R(r_{1;ijk}^v, r_{2;ijk}^v, r_{3;ijk}^v) \delta v_{i,j+1:j-1,k}^n \quad (12.312)$$

where the Orlanski function O_R is defined by (12.266) and

$$r_{1;ijk}^u = \delta u_{i+1:i-1,jk}^n, \quad r_{2;ijk}^u = \delta u_{i+1:i-1,jk}^{n-1}, \quad r_{3;ijk}^u = \delta u_{i+2:i-2,jk}^{n-1} \quad (12.313)$$

$$r_{1;ijk}^v = \delta v_{i,j+1:j-1,k}^n, \quad r_{2;ijk}^v = \delta v_{i,j+1:j-1,k}^{n-1}, \quad r_{3;ijk}^v = \delta v_{i,j+2:j-2,k}^{n-1} \quad (12.314)$$

6. Discharge condition (see equation (5.373))

$$\begin{aligned} \delta u_{ijk} &= \left(\frac{q_{ijk}^e}{h_{3;i;j;k}^u} - \frac{Q_{ij}^e}{H_{ij}^u} \right) \frac{1}{h_{2;ij}^u} \\ \delta v_{ijk} &= \left(\frac{q_{ijk}^e}{h_{3;i;j;k}^v} - \frac{Q_{ij}^e}{H_{ij}^v} \right) \frac{1}{h_{1;ij}^v} \end{aligned} \quad (12.315)$$

where q^e denotes the specified discharge profile. No discharge occurs at blocked cells where $q_k^e = 0$. The total discharge is given by $Q = \sum_{k=1}^{N_z} q_k^e$ and must be supplied as open boundary condition for the 2-D mode either in the form given by (12.284) or (12.285).

Once the baroclinic and mean components are known, the full 3-D currents are determined by adding the two components

$$u_{ijk}^{n+1} = U_{ij}^{n+1}/H_{i+1:i-1,j}^{n+1;c} + \delta u_{ijk}^{n+1}, \quad v_{ijk}^{n+1} = V_{ij}^{n+1}/H_{i,j+1:j-1}^{n+1;c} + \delta v_{ijk}^{n+1} \quad (12.316)$$

12.3.17.2 open boundary conditions for the advective and diffusive fluxes of 3-D currents

The formulations are identical to the one given in Section 12.3.16.2, except that the fluxes now include an additional k -index.

12.3.17.3 boundary conditions for the 3-D mode at closed lateral boundaries

The formulations are identical to the one given in Section 12.3.16.3, except that the fluxes now include an additional k -index. They are given for completeness.

Following (5.383)–(5.384) one has

$$u_{ijk} = 0, \quad v_{ijk} = 0 \quad (12.317)$$

at coastal boundaries.

Likewise all fluxes for the cross-advective and diffusive terms are set to zero at closed Y- or X-node boundaries:

$$F_{12;ijk}^{uv} = 0.0, \quad D_{12;ijk}^{uv} = 0.0, \quad F_{21;ijk}^{uv} = 0.0, \quad D_{21;ijk}^{uv} = 0.0 \quad (12.318)$$

12.3.18 Solution of the discretised equations for momentum

In the case of a 3-D current the discretised transport equations reduce to a system of linear equations of the form

$$\begin{aligned} A_{ijk} X_{ij,k-1}^{new} + B_{ijk} X_{ijk}^{new} + C_{ijk} X_{ij,k+1}^{new} &= D_{ijk}(X^{old}) \\ A_{ij,N_z} X_{ij,N_z-1}^{new} + B_{ijN_z} X_{ijN_z}^{new} &= D_{ijN_z}(X^{old}) \end{aligned} \quad (12.319)$$

where X^{old} and X^{new} are the values of the values of the current (u or v) at the “old” and “new” time step. Equations (12.319) form a tridiagonal matrix system in the vertical, which has to be solved at each horizontal grid point (i,j) .

Omitting the i and j indices for simplicity, the elements A_k , B_k and C_k can be written as the sum of different components, each representing particular term(s) in the corresponding momentum equation. Explicit expressions are given below for the update of u at the predictor step without operator splitting, as given by equation (12.4) or (12.28) so that $X^{old} = u^n$ and $X^{new} = u^p$. They are easily extended to the case for v -equation.

When operator splitting is used, four of the six steps are explicit integrations in which case the solution is straightforward. The two steps (12.55), (12.56) involving implicit terms are treated in a similar way.

The discretised 2-D equations for transports are written as

$$B_{ij}X_{ij}^{new} = D_{ij}X_{ij}^{old} \quad (12.320)$$

The composition of the B - and D -matrices is readily obtained from the discretisation formulae in the preceding sections and is not given here. The solution of (12.320) is straightforward.

12.3.18.1 composition of the tridiagonal matrix

1. Time derivative.

The contribution of the time derivative is given by

$$A_k^t = 0, \quad B_k^t = 1, \quad C_k^t = 0, \quad D_k^t = u_k^n \quad (12.321)$$

where $1 \leq k \leq N_z$.

2. Vertical advection.

The vertical advection term is split up into two contributions arising from the fluxes below and above a k -level. The former are given by

$$\begin{aligned} A_k^{a-} &= -\theta_a c_k^- (\alpha_k + f_k) \\ B_k^{a-} &= -\theta_a c_k^- (\beta_k - f_k) \\ C_k^{a-} &= 0 \\ D_k^{a-} &= (1 - \theta_a) c_k^- \left((\alpha_k + f_k) u_{k-1}^n + (\beta_k - f_k) u_k^n \right) \end{aligned} \quad (12.322)$$

where $2 \leq k \leq N_z$,

$$c_k^- = \frac{\Delta t \omega_k^{uw}}{2h_{3;k}^u}, \quad f_k = \left(1 - \Omega(r_k^{uw}) \right) s_k \quad (12.323)$$

and α_{ijk} , β_{ijk} , s_{ijk} , r_{ijk}^{uw} are defined by (12.169) and (12.170).

The terms arising from the flux above the k -level, are

$$\begin{aligned} A_k^{a+} &= 0 \\ B_k^{a+} &= \theta_a c_k^+ (\alpha_{k+1} + f_{k+1}) \\ C_k^{a+} &= \theta_a c_k^+ (\beta_{k+1} - f_{k+1}) \\ D_k^{a+} &= -(1 - \theta_a) c_k^+ \left((\alpha_{k+1} + f_{k+1}) u_k^n + (\beta_{k+1} - f_{k+1}) u_{k+1}^n \right) \end{aligned} \quad (12.324)$$

where $1 \leq k \leq N_z - 1$ and

$$c_k^+ = \frac{\Delta t \omega_{k+1}^{uw}}{2h_{3;k}^u} \quad (12.325)$$

3. Vertical diffusion.

As for vertical advection the fluxes below and above a k -level are taken separately. The former are given by

$$\begin{aligned} A_k^{d-} &= -\theta_v \frac{\Delta t \nu_{T;k}^{uw}}{h_{3;k}^u h_{3;k}^{uw}} \\ B_k^{d-} &= \theta_v \frac{\Delta t \nu_{T;k}^{uw}}{h_{3;k}^u h_{3;k}^{uw}} \\ C_k^{d-} &= 0 \\ D_k^{d-} &= -(1 - \theta_v) \frac{\Delta t \nu_{T;k}^{uw}}{h_{3;k}^u h_{3;k}^{uw}} (u_k^n - u_{k-1}^n) \end{aligned} \quad (12.326)$$

where $2 \leq k \leq N_z$.

The terms taken from the flux above the k -level, are

$$\begin{aligned} A_k^{d+} &= 0 \\ B_k^{d+} &= \theta_v \frac{\Delta t \nu_{T;k+1}^{uw}}{h_{3;k}^u h_{3;k+1}^{uw}} \\ C_k^{d+} &= -\theta_v \frac{\Delta t \nu_{T;k+1}^{uw}}{h_{3;k}^u h_{3;k+1}^{uw}} \\ D_k^{d+} &= (1 - \theta_v) \frac{\Delta t \nu_{T;k+1}^{uw}}{h_{3;k}^u h_{3;k+1}^{uw}} (u_{k+1}^n - u_k^n) \end{aligned} \quad (12.327)$$

where $1 \leq k \leq N_z - 1$.

4. Other explicit terms.

All other terms are explicit. Their contributions can be written as

$$\begin{aligned} A_k^e &= B_k^e = C_k^e = 0 \\ D_k^e &= \Delta t \left(\mathcal{O}_{1;k} - \tilde{\mathcal{A}}_{h1}(u)_k^n - \tilde{\mathcal{A}}_{h2}(u)_k^n + \mathcal{D}_{mh1}(\tau_{11})_k^{n;u} + \mathcal{D}_{mh2}(\tau_{12})_k^{n;u} \right) \end{aligned} \quad (12.328)$$

with $\mathcal{O}_{1;k}$ defined by (12.60).

5. Surface boundary conditions.

The contributions arising from the surface boundary conditions (12.241) for advection and (12.242) for diffusion are added to the highest level:

$$\begin{aligned} A_{Nz}^{a+} &= B_{Nz}^{a+} = C_{Nz}^{a+} = D_{Nz}^{a+} = 0 \\ A_{Nz}^{d+} &= B_{Nz}^{d+} = C_{Nz}^{d+} = 0 \\ D_{Nz}^{d+} &= \frac{\Delta t \tau_{s1}^u}{h_{3;Nz}^u} \end{aligned} \quad (12.329)$$

6. Bottom boundary conditions.

The contributions arising from the bottom boundary conditions (12.245) for advection are added to the lowest level:

$$A_1^{a-} = B_1^{a-} = C_1^{a-} = D_1^{a-} = 0 \quad (12.330)$$

The bottom contributions for vertical diffusion depends on whether the bottom stress is expressed as function of the bottom current (equation (12.246))

$$A_1^{d-} = C_1^{d-} = 0, \quad B_1^{d-} = \theta_v \frac{\Delta t k_b^u}{h_{3;1}^u}, \quad D_1^{d-} = -(1 - \theta_v) \frac{\Delta t k_b^u u_1^n}{h_{3;1}^u} \quad (12.331)$$

or the depth mean current (equation (12.249))

$$A_1^{d-} = B_1^{d-} = C_1^{d-} = 0, \quad D_1^{d-} = -\frac{\Delta t k_b^u \bar{u}^n}{h_{3;1}^u} \quad (12.332)$$

where the friction velocity is calculated using (12.248) or (12.251).

12.3.18.2 solution of tridiagonal systems

Tridiagonal matrix systems of the form (12.319) are solved using the algorithm, described in [Press *et al.* \(1992\)](#):

$$\begin{aligned}
 \beta_1 &= B_1, X_1^* = D_1/\beta_1 \\
 \gamma_k &= C_{k-1}/\beta_{k-1}, \beta_k = B_k - A_k\gamma_k, X_k^* = (D_k - A_kX_{k-1}^*)/\beta_k \\
 &\quad \text{for } k = 2, \dots, N_z \\
 X_{N_z}^{new} &= X_{N_z}^* \\
 X_k^{new} &= X_k^* - \gamma_{k+1}X_{k+1}^{new} \quad \text{for } k = N_z - 1, \dots, 1
 \end{aligned} \tag{12.333}$$

12.3.19 Elliptic equation for the free surface correction

When the implicit method for the surface slope term in the momentum equations is taken, an elliptic equation is obtained for the free surface correction ζ' . The discretised form of this equation is written in the form (12.34). The matrices A to G are evaluated in two phases:

1. The transports are first taken at interior wet points only and set to zero at solid and open boundaries.
2. The explicit and implicit terms arising by applying the open boundary condition are added.

12.3.19.1 interior terms

Defining

$$p_{ij}^u = \Delta t \mu_{ij}^u g_{ij}^u H_{ij}^{n+1, it; u} \frac{h_{2;ij}^u}{h_{1;ij}^u}, \quad p_{ij}^v = \Delta t \mu_{ij}^v g_{ij}^v H_{ij}^{n+1, it; v} \frac{h_{1;ij}^v}{h_{2;ij}^v} \tag{12.334}$$

where μ equals 1 at wet interior nodes and 0 at solid or open boundary velocity nodes, the coefficients of the elliptic matrix equation (12.34) become

$$\begin{aligned}
 A_{ij} &= -p_{ij}^u, \quad B_{ij} = -p_{ij}^v, \quad D_{ij} = -p_{i,j+1}^v, \quad E_{ij} = -p_{i+1,j}^u \\
 C_{ij} &= \frac{h_{1;ij}^c h_{2;ij}^c}{\Delta t} + p_{i+1,j}^u + p_{ij}^u + p_{i,j+1}^v + p_{ij}^v \\
 F_{ij} &= \frac{h_{1;ij}^c h_{2;ij}^c}{\Delta t} (\zeta_{ij}^n - \zeta^{n+1, it}) - h_{2;i+1,j}^u \mu_{i+1,j}^u U_{i+1,j}^p + h_{2;ij}^u \mu_{ij}^u U_{ij}^p \\
 &\quad - h_{1;i,j+1}^v \mu_{i,j+1}^v V_{i,j+1}^p + h_{1;ij}^v \mu_{ij}^v V_{ij}^p
 \end{aligned} \tag{12.335}$$

12.3.19.2 open boundary terms

From the discretised forms of the open boundary conditions additional terms are added to the coefficient matrices. They are described below for each type of open boundary condition using the notations introduced in Section 12.3.16.1. The following definitions are made in addition

$$\begin{aligned} m_{ij}^u &= \Delta t g_{i+1:i-1,j}^u H_{i+1:i-1,j}^{n+1, it;u} \frac{h_{2;ij}^u}{h_{1;i+1:i-1,j}^u} \\ m_{ij}^v &= \Delta t g_{i,j+1:j-1}^v H_{i,j+1:j-1}^{n+1, it;v} \frac{h_{1;ij}^v}{h_{2;i,j+1:j-1}^v} \end{aligned} \quad (12.336)$$

$$\beta_{ij}^u = \frac{1}{1 + 1.5\alpha_{ij}^u r_{ij}^u}, \quad \beta_{ij}^v = \frac{1}{1 + 1.5\alpha_{ij}^v r_{ij}^v} \quad (12.337)$$

$$\begin{aligned} O_{ij}^u &= \Delta t (f_{ij}^u V_{i:i-1}^c + H_{ij}^{n+1, it;u} F_{1;ij}^{t;n+1} + \tau_{s1;ij}^c - \tau_{b1;ij}^{n;u}) \\ O_{ij}^v &= \Delta t (-f_{ij}^v U_{i,j:j-1}^c + H_{ij}^{n+1, it;v} F_{2;ij}^{t;n+1} + \tau_{s2;ij}^c - \tau_{b2;ij}^{n;v}) \end{aligned} \quad (12.338)$$

0. Clamped

$$\begin{aligned} F_{i:i-1,j} &= F_{i:i-1,j} \pm h_{2;ij}^u U_{ij}^n \\ F_{i,j:j-1} &= F_{i,j:j-1} \pm h_{1;ij}^v V_{ij}^n \end{aligned} \quad (12.339)$$

1. Zero slope

$$\begin{aligned} F_{i:i-1,j} &= F_{i:i-1,j} \pm h_{2;ij}^u (U_{ij}^n + O_{ij}^u) \\ F_{i,j:j-1} &= F_{i,j:j-1} \pm h_{1;ij}^v (V_{ij}^n + O_{ij}^v) \end{aligned} \quad (12.340)$$

2. Zero volume flux

$$\begin{aligned} F_{i:i-1,j} &= F_{i:i-1,j} \pm h_{2;ij}^u U_{i+1:i-1,j}^p \\ F_{i,j:j-1} &= F_{i,j:j-1} \pm h_{1;ij}^v V_{i,j+1:j-1}^p \end{aligned} \quad (12.341)$$

$$\begin{aligned} C_{i:i-1,j} &= C_{i:i-1,j} - h_{2;ij}^u m_{ij}^u \\ C_{i,j:j-1} &= C_{i,j:j-1} - h_{1;ij}^v m_{ij}^v \end{aligned} \quad (12.342)$$

$$\begin{aligned} \text{West} &: E_{ij} = E_{ij} + h_{2;ij}^u m_{ij}^u \\ \text{East} &: A_{i-1,j} = A_{i-1,j} + h_{2;ij}^u m_{ij}^u \\ \text{South} &: D_{ij} = D_{ij} + h_{1;ij}^v m_{ij}^v \\ \text{North} &: B_{i,j-1} = B_{i,j-1} + h_{1;ij}^v m_{ij}^v \end{aligned} \quad (12.343)$$

3. Specified elevation

$$\begin{aligned} U_{ij}^{L;n+1, it} &= U_{ij}^{L;n} \mp s_{ij}^u \alpha_{ij}^u c_{ij}^u r_{ij}^u (\zeta_{i:i-1,j}^{n+1, it} - \zeta_{ij}^e) + O_{ij}^u \\ V_{ij}^{L;n+1, it} &= V_{ij}^{L;n} \mp s_{ij}^v \alpha_{ij}^v c_{ij}^v r_{ij}^v (\zeta_{i,j:j-1}^{n+1, it} - \zeta_{ij}^e) + O_{ij}^v \end{aligned} \quad (12.344)$$

$$\begin{aligned} F_{i:i-1,j} &= F_{i:i-1,j} \pm h_{2;ij}^u U_{ij}^{L;n+1, it} \\ F_{i,j:j-1} &= F_{i,j:j-1} \pm h_{1;ij}^v V_{ij}^{L;n+1, it} \end{aligned} \quad (12.345)$$

$$\begin{aligned} C_{i:i-1,j} &= C_{i:i-1,j} + h_{2;ij}^u s_{ij}^u \alpha_{ij}^u c_{ij}^u r_{ij}^u \\ C_{i,j:j-1} &= C_{i,j:j-1} + h_{1;ij}^v s_{ij}^v \alpha_{ij}^v c_{ij}^v r_{ij}^v \end{aligned} \quad (12.346)$$

4. Specified transport

$$\begin{aligned} F_{i:i-1,j} &= F_{i:i-1,j} \pm h_{2;ij}^u U_{ij}^e \\ F_{i,j:j-1} &= F_{i,j:j-1} \pm h_{1;ij}^v V_{ij}^e \end{aligned} \quad (12.347)$$

5. Radiation condition using shallow water speed

$$\begin{aligned} F_{i:i-1,j} &= F_{i:i-1,j} \pm h_{2;ij}^u \frac{U_{ij}^n + \alpha_{ij}^u U_{i+1:i-1,j}^p}{1 + \alpha_{ij}^u} \\ F_{i,j:j-1} &= F_{i,j:j-1} \pm h_{1;ij}^v \frac{V_{ij}^n + \alpha_{ij}^v V_{i,j+1:j-1}^p}{1 + \alpha_{ij}^v} \end{aligned} \quad (12.348)$$

$$\begin{aligned} C_{i:i-1,j} &= C_{i:i-1,j} - \frac{h_{2;ij}^u m_{ij}^u \alpha_{ij}^u}{1 + \alpha_{ij}^u} \\ C_{i,j:j-1} &= C_{i,j:j-1} - \frac{h_{1;ij}^v m_{ij}^v \alpha_{ij}^v}{1 + \alpha_{ij}^v} \end{aligned} \quad (12.349)$$

$$\begin{aligned} \text{West} &: E_{ij} = E_{ij} + \frac{h_{2;ij}^u m_{ij}^u \alpha_{ij}^u}{1 + \alpha_{ij}^u} \\ \text{East} &: A_{i-1,j} = A_{i-1,j} + \frac{h_{2;ij}^u m_{ij}^u \alpha_{ij}^u}{1 + \alpha_{ij}^u} \\ \text{South} &: D_{ij} = D_{ij} + \frac{h_{1;ij}^v m_{ij}^v \alpha_{ij}^v}{1 + \alpha_{ij}^v} \\ \text{North} &: B_{i,j-1} = B_{i,j-1} + \frac{h_{1;ij}^v m_{ij}^v \alpha_{ij}^v}{1 + \alpha_{ij}^v} \end{aligned} \quad (12.350)$$

6. Orlanski (1976) condition

$$\begin{aligned} F_{i:i-1,j} &= F_{i:i-1,j} \pm h_{2;ij}^u \left(1 - O_R \right) U_{ij}^n + O_R U_{i+1:i-1,j}^n \\ F_{i,j:j-1} &= F_{i,j:j-1} \pm h_{1;ij}^v \left(1 - O_R \right) V_{ij}^n + O_R V_{i,j+1:j-1}^n \end{aligned} \quad (12.351)$$

where the Orlanski weight function is defined by (12.266)–(12.268) at U- and V-nodes.

7. Camerlengo & O'Brien (1980)

$$\begin{aligned} F_{i:i-1,j} &= F_{i:i-1,j} \pm h_{2;ij}^u U_{i+1:i-1,j}^n & \text{if } U_{i+1:i-1,j}^n \geq U_{i+2:i-2,j}^{n-1} \\ F_{i:i-1,j} &= F_{i:i-1,j} \pm h_{2;ij}^u U_{ij}^n & \text{otherwise} \end{aligned} \quad (12.352)$$

$$\begin{aligned} F_{i,j:j-1} &= F_{i,j:j-1} \pm h_{1;ij}^v V_{i,j+1:j-1}^n & \text{if } V_{i,j+1:j-1}^n \geq V_{i,j+2:j-2}^{n-1} \\ F_{i,j:j-1} &= F_{i,j:j-1} \pm h_{1;ij}^v V_{ij}^n & \text{otherwise} \end{aligned} \quad (12.353)$$

8. Flather (1976) with specified elevation and transport

$$\begin{aligned} F_{i:i-1,j} &= F_{i:i-1,j} \pm h_{2;ij}^u \left(U_{ij}^e \mp \frac{1}{2} s_{ij}^u c_{ij}^u (\zeta_{i:i-1,j}^{n+1, it} - \zeta_{ij}^e) \right) \\ F_{i,j:j-1} &= F_{i,j:j-1} \pm h_{1;ij}^v \left(V_{ij}^e \mp \frac{1}{2} s_{ij}^v c_{ij}^v (\zeta_{i,j:j-1}^{n+1, it} - \zeta_{ij}^e) \right) \end{aligned} \quad (12.354)$$

$$\begin{aligned} C_{i:i-1,j} &= C_{i:i-1,j} + \frac{1}{2} h_{2;ij}^u s_{ij}^u c_{ij}^u \\ C_{i,j:j-1} &= C_{i,j:j-1} + \frac{1}{2} h_{1;ij}^v s_{ij}^v c_{ij}^v \end{aligned} \quad (12.355)$$

9. Flather (1976) with specified elevation

$$\begin{aligned} F_{i:i-1,j} &= F_{i:i-1,j} \pm h_{2;ij}^u \left(U_{ij}^{L;n+1, it} \mp \frac{1}{2} s_{ij}^u c_{ij}^u (\zeta_{i:i-1,j}^{n+1, it} - \zeta_{ij}^e) \right) \\ F_{i,j:j-1} &= F_{i,j:j-1} \pm h_{1;ij}^v \left(V_{ij}^{L;n+1, it} \mp \frac{1}{2} s_{ij}^v c_{ij}^v (\zeta_{i,j:j-1}^{n+1, it} - \zeta_{ij}^e) \right) \end{aligned} \quad (12.356)$$

where the local solutions $(U^{L;n+1, it}, V^{L;n+1, it})$ are determined by (12.344).

$$\begin{aligned} C_{i:i-1,j} &= C_{i:i-1,j} + h_{2;ij}^u s_{ij}^u c_{ij}^u \left(\frac{1}{2} + \alpha_{ij}^u r_{ij}^u \right) \\ C_{i,j:j-1} &= C_{i,j:j-1} + h_{1;ij}^v s_{ij}^v c_{ij}^v \left(\frac{1}{2} + \alpha_{ij}^v r_{ij}^v \right) \end{aligned} \quad (12.357)$$

10. Røed & Smedstad (1984)

$$\begin{aligned}\zeta_{ij}^{L;n+1,ut} &= \zeta_{ij}^{L;n} - \frac{\Delta t(h_{1;i:i-1,j+1}^v V_{i:i-1,j+1}^n - h_{1;i:i-1,j}^v V_{i:i-1,j}^n)}{h_{1;i:i-1,j}^c h_{2;i:i-1,j}^c} \\ \zeta_{ij}^{L;n+1,vt} &= \zeta_{ij}^{L;n} - \frac{\Delta t(h_{2;i+1,j:j-1}^u U_{i+1,j:j-1}^n - h_{2;i,j:j-1}^u U_{i,j:j-1}^n)}{h_{1;i,j:j-1}^c h_{2;i,j:j-1}^c}\end{aligned}\quad (12.358)$$

$$\begin{aligned}F_{i:i-1,j} &= F_{i:i-1,j} \pm h_{2;ij}^u \left(U_{ij}^{L;n+1,ut} - s_{ij}^u c_{ij}^u (\zeta_{i:i-1,j}^{n+1,ut} - \zeta_{ij}^{L;n+1,ut}) \right) \\ F_{i,j:j-1} &= F_{i,j:j-1} \pm h_{1;ij}^v \left(V_{ij}^{L;n+1,vt} - s_{ij}^v c_{ij}^v (\zeta_{i,j:j-1}^{n+1,vt} - \zeta_{ij}^{L;n+1,vt}) \right)\end{aligned}\quad (12.359)$$

$$\begin{aligned}C_{i:i-1,j} &= C_{i:i-1,j} + h_{2;ij}^u c_{ij}^u (1 + s_{ij}^u \alpha_{ij}^u r_{ij}^u) \\ C_{i,j:j-1} &= C_{i,j:j-1} + h_{1;ij}^v c_{ij}^v (1 + s_{ij}^v \alpha_{ij}^v r_{ij}^v)\end{aligned}\quad (12.360)$$

11. Characteristic method with specified elevation and transport

$$\begin{aligned}R_{i;ij}^{n+1,ut} &= U_{ij}^e \pm \frac{1}{2} s_{ij}^u c_{ij}^u \left(\zeta_{ij}^e + (2 - s_{ij}^u) \zeta_{i:i-1,j}^{n+1,ut} \right) \\ R_{i;ij}^{n+1,vt} &= V_{ij}^e \pm \frac{1}{2} s_{ij}^v c_{ij}^v \left(\zeta_{ij}^e + (2 - s_{ij}^v) \zeta_{i,j:j-1}^{n+1,vt} \right)\end{aligned}\quad (12.361)$$

$$\begin{aligned}(1 + \frac{3}{2} \alpha_{ij}^u r_{ij}^u) R_{o;ij}^{n+1,ut} &= R_{o;ij}^{n;u} \\ &+ \alpha_{ij}^u r_{ij}^u (U_{i+1:i-1,j}^p + \frac{1}{2} R_{i;ij}^{n+1,ut} \mp 2 c_{ij}^u \zeta_{i:i-1,j}^{n+1,ut}) \\ &+ \frac{\alpha_{ij}^u}{h_{2;i:i-1,j}^c} \left(\pm (h_{1;i:i-1,j+1}^v V_{i:i-1,j+1}^n - h_{1;i:i-1,j}^v V_{i:i-1,j}^n) \right. \\ &\left. + U_{i:i-1,j}^{n;c} (h_{2;i+1:i-1,j}^u - h_{2;ij}^u) \right) + O_{ij}^u\end{aligned}\quad (12.362)$$

and

$$\begin{aligned}(1 + \frac{3}{2} \alpha_{ij}^v r_{ij}^v) R_{o;ij}^{n+1,vt} &= R_{o;ij}^{n;v} \\ &+ \alpha_{ij}^v r_{ij}^v (V_{i,j+1:j-1}^p + \frac{1}{2} R_{i;ij}^{n+1,vt} \mp 2 c_{ij}^v \zeta_{i,j:j-1}^{n+1,vt}) \\ &+ \frac{\alpha_{ij}^v}{h_{1;i,j:j-1}^c} \left(\pm (h_{2;i+1,j:j-1}^u U_{i+1,j:j-1}^n - h_{2;i,j:j-1}^u U_{i,j:j-1}^n) \right)\end{aligned}$$

$$+ V_{i,j:j-1}^{n;c} (h_{1;i,j+1:j-1}^v - h_{1;ij}^v) \Big) + O_{ij}^v \quad (12.363)$$

$$\begin{aligned} F_{i:i-1,j} &= F_{i:i-1,j} \pm \frac{1}{2} h_{2;ij}^u \left(R_{i;ij}^{n+1,it;u} + R_{o;ij}^{n+1,it;u} \right) \\ F_{i,j:j-1} &= F_{i,j:j-1} \pm \frac{1}{2} h_{1;ij}^v \left(R_{i;ij}^{n+1,it;v} + R_{o;ij}^{n+1,it;v} \right) \end{aligned} \quad (12.364)$$

$$\begin{aligned} C_{i:i-1,j} &= C_{i:i-1,j} - \frac{1}{4} h_{2;ij}^u s_{ij}^u c_{ij}^u (2 - s_{ij}^u) (1 + \frac{1}{2} \alpha_{ij}^u r_{ij}^u \beta_{ij}^u) \\ &\quad + h_{2;ij}^u \alpha_{ij}^u c_{ij}^u r_{ij}^u \beta_{ij}^u - \frac{1}{2} \alpha_{ij}^u r_{ij}^u \beta_{ij}^u m_{ij}^u \end{aligned} \quad (12.365)$$

$$\begin{aligned} C_{i,j:j-1} &= C_{i,j:j-1} - \frac{1}{4} h_{1;ij}^v s_{ij}^v c_{ij}^v (2 - s_{ij}^v) (1 + \frac{1}{2} \alpha_{ij}^v r_{ij}^v \beta_{ij}^v) \\ &\quad + h_{1;ij}^v \alpha_{ij}^v c_{ij}^v r_{ij}^v \beta_{ij}^v - \frac{1}{2} \alpha_{ij}^v r_{ij}^v \beta_{ij}^v m_{ij}^v \end{aligned} \quad (12.366)$$

$$\begin{aligned} \text{West} &: E_{ij} = E_{ij} + \frac{1}{2} h_{2;ij}^u \alpha_{ij}^u r_{ij}^u \beta_{ij}^u m_{ij}^u \\ \text{East} &: A_{i-1,j} = A_{i-1,j} + \frac{1}{2} h_{2;ij}^u \alpha_{ij}^u r_{ij}^u \beta_{ij}^u m_{ij}^u \\ \text{South} &: D_{ij} = D_{ij} + \frac{1}{2} h_{1;ij}^v \alpha_{ij}^v r_{ij}^v \beta_{ij}^v m_{ij}^v \\ \text{North} &: B_{i,j-1} = B_{i,j-1} + \frac{1}{2} h_{1;ij}^v \alpha_{ij}^v r_{ij}^v \beta_{ij}^v m_{ij}^v \end{aligned} \quad (12.367)$$

12. Characteristic method with specified elevation

The discretisations are the same as in the previous case with (U_{ij}^e, V_{ij}^e) replaced by $(U_{ij}^{L;n+1,it}, V_{ij}^{L;n+1,it})$.

13. Characteristic method using zero normal gradient

The discretisations are the same as in the previous case with $(R_{i;ij}^{n+1,it;u}, R_{i;ij}^{n+1,it;v})$ defined by

$$\begin{aligned} R_{i;ij}^{n+1,it;u} &= R_{i;ij}^{n;u} \\ &\quad - \frac{\alpha_{ij}^u}{h_{2;i:i-1,j}^c} \left(\pm (h_{1;i:i-1,j+1}^v V_{i:i-1,j+1}^n - h_{1;i:i-1,j}^v V_{i:i-1,j}^n) \right. \\ &\quad \left. + U_{i:i-1,j}^{c;m} (h_{2;i+1:i-1,j}^u - h_{2;ij}^u) \right) + O_{ij}^u \end{aligned} \quad (12.368)$$

and

$$\begin{aligned}
 R_{i;ij}^{n+1, it;v} &= R_{i;ij}^{n;v} \\
 &- \frac{\alpha_{ij}^v}{h_{1;i,j;j-1}^c} \left(\pm (h_{2;i+1,j;j-1}^u U_{i+1,j;j-1}^n - h_{2;i,j;j-1}^u U_{i,j;j-1}^n) \right. \\
 &\quad \left. + V_{i,j;j-1}^{c;m} (h_{1;i,j+1;j-1}^v - h_{1;ij}^v) \right) + O_{ij}^v
 \end{aligned} \tag{12.369}$$

12.4 Scalar transport equations

12.4.1 General aspects of discretisation

General features of the discretisation are

- With exception of turbulence variables (see Section 12.5) scalar quantities are located at C-nodes.
- Horizontal advection and diffusion terms are discretised explicitly in time.
- In analogy with the momentum equations vertical advection is taken semi-implicitly while vertical diffusion is treated fully implicitly. The equations for vertical advection and diffusion are presented here in a more general form covering both the explicit, implicit and semi-implicit cases.
- As recommended by [Ruddick \(1995\)](#), the vertical spacing h_3 is eliminated from the time derivative (except in the absence of advection) by adding corrector terms to the right hand side of the transport equation.
- Source terms are discretised explicitly. Contrary to the momentum and turbulence transport equations the sink terms are evaluated explicitly. This has no impact on temperature and salinity, but has been introduced for future implementation of biological concentrations where conservation plays an important role.
- When the momentum equations are solved using mode-splitting, the advecting current (u_f, v_f) used for horizontal advection is composed of the baroclinic current at the “predictor” step plus a filtered depth-independent component obtained by averaging over the more rapidly varying 2-D mode (see equations (12.23)–(12.24)). Otherwise, $u_f = u^{n+1}$, $v_f = v^{n+1}$.

- The transport equation is integrated in time with or without the operator splitting method depending on the type of advection scheme. Note that the program allows to use of different advection schemes in the momentum and scalar transport modules.

In analogy with momentum the time discretisation of a scalar transport equation depends on the type of advection scheme selected in the program. Several schemes are available. The type is selected with the model switch `iopt_adv_scal` which, in analogy with the switch `iopt_adv_3d` for momentum, has the following meaning

- 0: horizontal and vertical advection disabled
- 1: upwind scheme for horizontal and vertical advection
- 2: Lax-Wendroff scheme for horizontal, central scheme for vertical advection⁷
- 3: TVD (Total Variation Diminishing) scheme using the superbee limiter as a weighting function between the upwind scheme and either the Lax-Wendroff scheme in the horizontal or the central scheme in the vertical
- 4: as the previous case now using the monotonic limiter.

12.4.2 Alternative formulation of the transport equation

The general form of a scalar transport equation is given by (5.38). Before discussing its discretisation, the following modifications are made

1. The advective velocities u, v are replaced by the “filtered” currents u_f, v_f given as the sum of the “filtered” depth-mean current and the baroclinic part of the “predicted” current so that the numerical time-integration guarantees the conservation of the scalar quantity (Deleersnijder, 1993).
2. Making use of the continuity equation (5.18), the time derivative term is written as

$$\begin{aligned} \frac{1}{h_3} \frac{\partial}{\partial t} (h_3 \psi) &= \frac{\partial \psi}{\partial t} + \frac{\psi}{h_3} \frac{\partial h_3}{\partial t} \\ &= \frac{\partial \psi}{\partial t} - \mathcal{C}_{s1}^f(\psi) - \mathcal{C}_{s2}^f(\psi) - \mathcal{C}_{s3}(\psi) \end{aligned} \quad (12.370)$$

⁷The “pure” Lax-Wendroff scheme has only been implemented for illustrative purposes and should be avoided in realistic simulations.

where the “corrector” terms are defined by

$$\mathcal{C}_{s1}^f(\psi) = \frac{\psi}{h_1 h_2 h_3} \frac{\partial}{\partial \xi_1} (h_2 h_3 u_f) \quad (12.371)$$

$$\mathcal{C}_{s2}^f(\psi) = \frac{\psi}{h_1 h_2 h_3} \frac{\partial}{\partial \xi_2} (h_1 h_3 v_f) \quad (12.372)$$

$$\mathcal{C}_{s3}(\psi) = \frac{\psi}{h_3} \frac{\partial \omega}{\partial s} \quad (12.373)$$

3. For reasons of conservation the source and sink are both discretised explicitly. The operation is not without risk since the method may produce negative concentrations (Burchard *et al.*, 2003). To simplify the notations the following operator is defined

$$\mathcal{T}(\psi) = \mathcal{P}(\psi) - \mathcal{S}(\psi) \quad (12.374)$$

An alternative method is the Patankar scheme, discussed in Section 12.5 below, which is monotone but does not guarantee conservation.

The new form of the scalar transport equation then becomes

$$\begin{aligned} \frac{\partial \psi}{\partial t} + \mathcal{A}_{h1}^f(\psi) + \mathcal{A}_{h2}^f(\psi) + \mathcal{A}_v(\psi) - \mathcal{C}_{s1}^f(\psi) - \mathcal{C}_{s2}^f(\psi) - \mathcal{C}_{s3}(\psi) \\ = \mathcal{T}(\psi) + \mathcal{D}_{sv}(\psi) + \mathcal{D}_{sh1}(\psi) + \mathcal{D}_{sh2}(\psi) \end{aligned} \quad (12.375)$$

where \mathcal{A}_{hi}^f are the horizontal advective operators using the filtered currents

$$\mathcal{A}_{h1}^f(\psi) = \frac{1}{h_1 h_2 h_3} \frac{\partial}{\partial \xi_1} (h_2 h_3 u_f \psi) \quad (12.376)$$

$$\mathcal{A}_{h2}^f(\psi) = \frac{1}{h_1 h_2 h_3} \frac{\partial}{\partial \xi_2} (h_1 h_3 v_f \psi) \quad (12.377)$$

$$(12.378)$$

12.4.3 Time discretisation

Three cases can be distinguished for the time integration. They are discussed in the subsections below.

12.4.3.1 integration without advection

In the absence of physical advection (`iopt_adv_scal=0`) the transport equation is integrated in time using

$$\frac{h_3^{n+1} \psi^{n+1} - h_3^n \psi^n}{h_3^{n+1} \Delta t} = \theta_v \mathcal{D}_{sv}(\psi^{n+1}) + (1 - \theta_v) \mathcal{D}_{sv}(\psi^n) + \mathcal{T}(\psi^n) + \mathcal{D}_{sh1}(\psi^n) + \mathcal{D}_{sh2}(\psi^n) \quad (12.379)$$

12.4.3.2 integration with advection but without operator splitting

If `iopt_adv_scal`=1 or 2, the transport equation (12.375) is integrated in time using

$$\begin{aligned} \frac{\psi^{n+1} - \psi^n}{\Delta t} = & -\mathcal{A}_{h1}^f(\psi^n) + \mathcal{C}_{s1}^f(\psi^n) - \mathcal{A}_{h2}^f(\psi^n) + \mathcal{C}_{s2}^f(\psi^n) \\ & - \theta_a \mathcal{A}_v(\psi^{n+1}) - (1 - \theta_a) \mathcal{A}_v(\psi^n) + \mathcal{C}_{s3}(\psi^n) + \theta_v \mathcal{D}_{sv}(\psi^{n+1}) \\ & + (1 - \theta_v) \mathcal{D}_{sv}(\psi^n) + \mathcal{T}(\psi^n) + \mathcal{D}_{sh1}(\psi^n) + \mathcal{D}_{sh2}(\psi^n) \end{aligned} \quad (12.380)$$

12.4.3.3 integration with operator splitting

If `iopt_adv_scal`=3, integration is performed along the following steps:

- Part A

$$\frac{\psi_A^{n+1/3} - \psi^n}{\Delta t} = -\mathcal{A}_{h1}^f(\psi^n) + \mathcal{C}_{s1}^f(\psi^n) + \mathcal{D}_{sh1}(\psi^n) \quad (12.381)$$

$$\frac{\psi_A^{n+2/3} - \psi_A^{n+1/3}}{\Delta t} = -\mathcal{A}_{h2}^f(\psi_A^{n+1/3}) + \mathcal{C}_{s2}^f(\psi^n) + \mathcal{D}_{sh2}(\psi_A^{n+1/3}) \quad (12.382)$$

$$\begin{aligned} \frac{\psi_A^{n+1} - \psi_A^{n+2/3}}{\Delta t} = & -\theta_a \mathcal{A}_v(\psi_A^{n+1}) - (1 - \theta_a) \mathcal{A}_v(\psi_A^{n+2/3}) + \mathcal{C}_{s3}(\psi^n) \\ & + \theta_v \mathcal{D}_{sv}(\psi_A^{n+1}) + (1 - \theta_v) \mathcal{D}_{sv}(\psi_A^{n+2/3}) + \mathcal{T}(\psi^n) \end{aligned} \quad (12.383)$$

- Part B

$$\begin{aligned} \frac{\psi_B^{n+1/3} - \psi^n}{\Delta t} = & -\theta_a \mathcal{A}_v(\psi_B^{n+1/3}) - (1 - \theta_a) \mathcal{A}_v(\psi^n) + \mathcal{C}_{s3}(\psi^n) \\ & + \theta_v \mathcal{D}_v(\psi_B^{n+1/3}) + (1 - \theta_v) \mathcal{D}_{sv}(\psi^n) + \mathcal{T}(\psi^n) \end{aligned} \quad (12.384)$$

$$\frac{\psi_B^{n+2/3} - \psi_B^{n+1/3}}{\Delta t} = -\mathcal{A}_{h2}^f(\psi_B^{n+1/3}) + \mathcal{C}_{s2}^f(\psi^n) + \mathcal{D}_{sh2}(\psi_B^{n+1/3}) \quad (12.385)$$

$$\frac{\psi_B^{n+1} - \psi_B^{n+2/3}}{\Delta t} = -\mathcal{A}_{h1}(\psi_B^{n+2/3}) + \mathcal{C}_{s1}^f(\psi^n) + \mathcal{D}_{sh1}(\psi_B^{n+2/3}) \quad (12.386)$$

- Updated value

$$\psi^{n+1} = \frac{1}{2}(\psi_A^{n+1} + \psi_B^{n+1}) \quad (12.387)$$

For the reasons discussed in Section 12.3.3.1 vertical advection is discretised semi-implicitly and vertical diffusion implicitly. The default values, taken for the implicitity factors, are then given by $\theta_a = 0.501$, $\theta_v = 1$. The use of the TVD scheme with the operator splitting method increases the CPU time but has the capacity to preserve horizontal and vertical gradients in frontal areas. The user therefore needs to make a balance between CPU time and accuracy when selecting an appropriate scheme.

12.4.4 Discretisation of advection

The advective terms in the scalar transport equations are written as the divergence of the fluxes F_1 , F_2 , F_3 defined in Table 12.3:

$$\mathcal{A}_{h1}^f(\psi) = \frac{1}{h_1 h_2 h_3} \frac{\partial}{\partial \xi_1} (h_2 h_3 u_f \psi) = \frac{1}{h_1 h_2 h_3} \frac{\partial}{\partial \xi_1} (h_2 h_3 F_1) \quad (12.388)$$

$$\mathcal{A}_{h2}^f(\psi) = \frac{1}{h_1 h_2 h_3} \frac{\partial}{\partial \xi_2} (h_1 h_3 v_f \psi) = \frac{1}{h_1 h_2 h_3} \frac{\partial}{\partial \xi_2} (h_1 h_3 F_2) \quad (12.389)$$

$$\mathcal{A}_v(\psi) = \frac{1}{h_3} \frac{\partial}{\partial s} (\omega \psi) = \frac{1}{h_3} \frac{\partial F_3}{\partial s} \quad (12.390)$$

12.4.4.1 advection in the X-direction

The advective term in the X-direction is obtained by differencing the flux F_1^u at the C-node

$$\mathcal{A}_{h1}^f(\psi)_{ijk}^c = \frac{h_{2;i+1,j}^u h_{3;i+1,jk}^u F_{1;i+1,jk}^u - h_{2;ij}^u h_{3;ijk}^u F_{1;ijk}^u}{h_{1;ij}^c h_{2;ij}^c h_{3;ijk}^c} \quad (12.391)$$

The flux is calculated from

$$F_{1;ij}^u = (1 - \Omega(r_{ijk}^u)) F_{up;ijk}^u + \Omega(r_{ijk}^u) F_{lw;ijk}^u \quad (12.392)$$

where $F_{up;ijk}^u$ and $F_{lw;ijk}^u$ are the upwind and Lax-Wendroff fluxes at the U-node:

$$F_{up;ijk}^u = \frac{1}{2} u_{f;ijk} \left((\alpha_{ij} + s_{ijk}) \psi_{i-1,jk} + (\beta_{ij} - s_{ijk}) \psi_{ijk} \right) \quad (12.393)$$

$$F_{lw;ijk}^u = \frac{1}{2} u_{f;ijk} \left((\alpha_{ij} + c_{ijk}) \psi_{i-1,jk} + (\beta_{ij} - c_{ijk}) \psi_{ijk} \right) \quad (12.394)$$

where s_{ijk} and c_{ijk} are the sign and CFL number of the advecting current

$$s_{ijk} = \text{Sign}(u_{f;ijk}), \quad c_{ijk} = \frac{u_{f;ijk} \Delta t}{h_{1;ij}^u} \quad (12.395)$$

$$\alpha_{ij} = \frac{h_{1;ij}^c}{h_{1;ij}^u}, \quad \beta_{ij} = \frac{h_{1;i-1,j}^c}{h_{1;ij}^u} \quad (12.396)$$

The form of the weighting function is given by (12.47)–(12.50), depending on the type of advection scheme, selected by the switch `iopt_adv_scal`. The argument r of the weight function is defined by

$$\begin{aligned} r_{ijk}^u &= \frac{(\alpha_{ij} + s_{ijk})\Delta F_{i-1,jk}^u + (\beta_{ij} - s_{ijk})\Delta F_{i+1,jk}^u}{2\Delta F_{ijk}^u} \\ \Delta F_{ijk}^u &= F_{lw;ijk}^u - F_{up;ijk}^u \end{aligned} \quad (12.397)$$

12.4.4.2 advection in the Y-direction

The advective term in the Y-direction is obtained by differencing the flux F_1^v at the C-node

$$\mathcal{A}_{h2}^f(\psi)_{ijk}^c = \frac{h_{1;i,j+1}^v h_{3;i,j+1,k}^v F_{2;i,j+1,k}^v - h_{1;ij}^v h_{3;ijk}^v F_{2;ijk}^v}{h_{1;ij}^c h_{2;ij}^c h_{3;ijk}^c} \quad (12.398)$$

The flux is calculated from

$$F_{1;ij}^v = \left(1 - \Omega(r_{ijk}^v)\right) F_{up;ijk}^v + \Omega(r_{ijk}^v) F_{lw;ijk}^v \quad (12.399)$$

where $F_{up;ijk}^v$ and $F_{lw;ijk}^v$ are the upwind and Lax-Wendroff fluxes at the V-node:

$$F_{up;ijk}^v = \frac{1}{2} v_{f;ijk} \left((\alpha_{ijk} + s_{ijk}) \psi_{i,j-1,k} + (\beta_{ijk} - s_{ijk}) \psi_{ijk} \right) \quad (12.400)$$

$$F_{lw;ijk}^v = \frac{1}{2} v_{f;ijk} \left((\alpha_{ijk} + c_{ijk}) \psi_{i,j-1,k} + (\beta_{ijk} - c_{ijk}) \psi_{ijk} \right) \quad (12.401)$$

where s_{ijk} and c_{ijk} are the sign and CFL number of the advecting current

$$s_{ijk} = \text{Sign}(v_{f;ijk}), \quad c_{ijk} = \frac{v_{f;ijk} \Delta t}{h_{2;ij}^v} \quad (12.402)$$

$$\alpha_{ij} = \frac{h_{2;ij}^c}{h_{2;ij}^v}, \quad \beta_{ij} = \frac{h_{2;i,j-1}^c}{h_{2;ij}^v} \quad (12.403)$$

The form of the weighting function is given by (12.47)–(12.50), depending on the type of advection scheme, selected by the switch `iopt_adv_scal`. The argument r of the weight function is defined by

$$\begin{aligned} r_{ijk}^v &= \frac{(\alpha_{ij} + s_{ijk})\Delta F_{i,j-1,k}^v + (\beta_{ij} - s_{ijk})\Delta F_{i,j+1,k}^v}{2\Delta F_{ijk}^v} \\ \Delta F_{ijk}^v &= F_{lw;ijk}^v - F_{up;ijk}^v \end{aligned} \quad (12.404)$$

12.4.4.3 advection in the vertical direction

The vertical advective term is obtained by differencing the flux F_3^w at the C-node

$$\mathcal{A}_v(\psi)_{ijk}^c = \frac{F_{3;ijk,k+1}^w - F_{3;ijk}^w}{h_{3;ijk}^c} \quad (12.405)$$

The flux is calculated from

$$F_{3;ijk}^w = \left(1 - \Omega(r_{ijk}^w)\right)F_{up;ijk}^w + \Omega(r_{ijk}^w)F_{ce;ijk}^w \quad (12.406)$$

where $F_{up;ijk}^w$ and $F_{ce;ijk}^w$ are the upwind and central fluxes at the W-node:

$$F_{up;ijk}^w = \frac{1}{2}\omega_{ijk}^w \left((\alpha_{ijk} + s_{ijk})\psi_{ij,k-1} + (\beta_{ijk} - s_{ijk})\psi_{ijk} \right) \quad (12.407)$$

$$F_{ce;ijk}^w = \frac{1}{2}\omega_{ijk}^w \left(\alpha_{ijk}\psi_{ij,k-1} + \beta_{ijk}\psi_{ijk} \right) \quad (12.408)$$

where

$$s_{ijk} = \text{Sign}(\omega_{ijk}^w), \quad \alpha_{ijk} = \frac{h_{3;ijk}^c}{h_{3;ijk}^w}, \quad \beta_{ijk} = \frac{h_{3;ijk,k-1}^c}{h_{3;ijk}^w} \quad (12.409)$$

The form of the weighting function is given by (12.47)–(12.50), depending on the type of advection scheme, selected by the switch `iopt_adv_scal`. The argument r of the weight function is defined by

$$\begin{aligned} r_{ijk}^w &= \frac{(\alpha_{ijk} + s_{ijk})\Delta F_{ij,k-1}^w + (\beta_{ijk} - s_{ijk})\Delta F_{ij,k+1}^w}{2\Delta F_{ij,k}^w} \\ \Delta F_{ij,k}^w &= F_{ce;ijk}^w - F_{up;ijk}^w \end{aligned} \quad (12.410)$$

12.4.4.4 corrector terms

The corrector terms, defined by (12.371)–(12.373), are discretised using

$$\mathcal{C}_{s1}^f(\psi)^c = \psi_{ijk} \frac{h_{2;i+1,j}^u h_{3;i+1,jk}^u u_{f;i+1,jk} - h_{2;ij}^u h_{3;ijk}^u u_{f;ijk}}{h_{1;ij}^c h_{2;ij}^c h_{3;ijk}^c} \quad (12.411)$$

$$\mathcal{C}_{s2}^f(\psi)^c = \psi_{ijk} \frac{h_{1;i,j+1}^v h_{3;i,j+1,k}^v v_{f;i,j+1,k} - h_{1;ij}^v h_{3;ijk}^v v_{f;ijk}}{h_{1;ij}^c h_{2;ijk}^c h_{3;ijk}^c} \quad (12.412)$$

$$\mathcal{C}_{s3}(\psi)^c = \psi_{ijk} \frac{\omega_{ij,k+1} - \omega_{ijk}}{h_{3;ijk}^c} \quad (12.413)$$

12.4.5 Discretisation of diffusion

The diffusive terms in the scalar transport equations are written as the divergence of the fluxes D_1 , D_2 , D_3 defined in Table 12.3:

$$\mathcal{D}_{sh1}(\psi) = \frac{1}{h_1 h_2 h_3} \frac{\partial}{\partial \xi_1} \left(\lambda_H \frac{h_2 h_3}{h_1} \frac{\partial \psi}{\partial \xi_1} \right) = \frac{1}{h_1 h_2 h_3} \frac{\partial}{\partial \xi_1} (h_2 h_3 D_1) \quad (12.414)$$

$$\mathcal{D}_{sh2}(\psi) = \frac{1}{h_1 h_2 h_3} \frac{\partial}{\partial \xi_2} \left(\lambda_H \frac{h_1 h_3}{h_2} \frac{\partial \psi}{\partial \xi_2} \right) = \frac{1}{h_1 h_2 h_3} \frac{\partial}{\partial \xi_2} (h_1 h_3 D_2) \quad (12.415)$$

$$\mathcal{D}_{sv}(\psi) = \frac{1}{h_3} \frac{\partial}{\partial s} \left(\frac{\lambda_T^\psi}{h_3} \frac{\partial \psi}{\partial s} \right) = \frac{1}{h_3} \frac{\partial D_3}{\partial s} \quad (12.416)$$

12.4.5.1 diffusion in the X-direction

The diffusion term in the X-direction is obtained by differencing the flux D_1^u at the C-node

$$\mathcal{D}_{sh1}(\psi)_{ijk}^c = \frac{h_{2;i+1,j}^u h_{3;i+1,jk}^u D_{1;i+1,jk}^u - h_{2;ij}^u h_{3;ijk}^u D_{1;ijk}^u}{h_{1;ij}^c h_{2;ij}^c h_{3;ijk}^c} \quad (12.417)$$

The flux is given by

$$D_{1;ijk}^u = \frac{\lambda_H^u (\psi_{ijk} - \psi_{i-1,jk})}{h_{1;ij}^u} \quad (12.418)$$

12.4.5.2 diffusion in the Y-direction

The diffusion term in the Y-direction is obtained by differencing the flux D_2^v at the C-node

$$\mathcal{D}_{sh2}(\psi)_{ijk}^c = \frac{h_{1;i,j+1}^v h_{3;i,j+1,k}^v D_{2;i,j+1,k}^v - h_{1;ij}^v h_{3;ijk}^v D_{2;ijk}^v}{h_{1;ij}^c h_{2;ij}^c h_{3;ijk}^c} \quad (12.419)$$

The flux is given by

$$D_{2;ijk}^v = \frac{\lambda_H^v (\psi_{ijk} - \psi_{i,j-1,k})}{h_{2;ij}^v} \quad (12.420)$$

12.4.5.3 diffusion in the vertical direction

The vertical diffusion term is obtained by differencing the flux D_3^w at the C-node

$$\mathcal{D}_{sv}(\psi_{ijk})^c = \frac{D_{3;ij,k+1}^w - D_{3;ijk}^w}{h_{3;ijk}^c} \quad (12.421)$$

The flux is given by

$$D_{3;ijk}^w = \frac{\lambda_T^{\psi;w} (\psi_{ijk} - \psi_{ij,k-1})}{h_{3;ijk}^w} \quad (12.422)$$

12.4.6 Diffusion coefficients for scalars

12.4.6.1 horizontal diffusion coefficients

The discretised values of the horizontal diffusion coefficient at the U- and V-nodes are obtained by applying (5.44) and interpolating $D_{T;ijk}^c$ and $D_{S;ijk}^{uv}$, given by (12.210)–(12.211), to the U- and V-nodes

$$\lambda_{H;ijk}^u = C_s h_{1;ij}^u h_{2;ij}^u \sqrt{\left(D_{T;ijk}^u\right)^2 + \left(D_{S;ijk}^u\right)^2} \quad (12.423)$$

$$\lambda_{H;ijk}^v = C_s h_{1;ij}^v h_{2;ij}^v \sqrt{\left(D_{T;ijk}^v\right)^2 + \left(D_{S;ijk}^v\right)^2} \quad (12.424)$$

12.4.6.2 vertical diffusion coefficient

The vertical diffusion coefficient for scalars λ_T is obtained from one of the available turbulence schemes, described in Section 5.3. Values are first stored at the W-nodes and interpolated afterwards at the U- and V-nodes for the calculation of the vertical diffusion fluxes in the momentum equations. The evaluation of λ_T only involves algebraic expressions so that the discretisation procedure is straightforward.

For further comments see Section 12.3.12.2.

12.4.7 Boundary conditions

12.4.7.1 surface boundary conditions

The program allows four type of boundary conditions

1. Neumann condition with a prescribed (downward) surface flux $F_{s;ij}^{\psi;w}$

$$D_{3;ijN_z}^w = F_{s;ij}^{\psi;w} \quad (12.425)$$

2. Neumann condition using a surface transfer velocity

$$D_{3;ijN_z}^w = C_{s;ij}^\psi \left(\psi_{s;ij}^w - (1 - \theta_v) \psi_{ijN_z} - \theta_v \psi_{ijN_z}^{n+1} \right) \quad (12.426)$$

where $C_{s;ij}^\psi$ is the transfer velocity and $\psi_{s;ij}^w$ a prescribed external value.

3. Dirichlet condition with a prescribed external value $\psi_{s;ij}^c$ at the first C-node below the surface

$$\psi_{ijN_z}^{n+1} = \psi_{s;ij}^c \quad (12.427)$$

4. Dirichlet condition with a prescribed external value $\psi_{s;ij}^w$ at the surface itself. In that case the value at the first node below the surface is determined by interpolation

$$\psi_{ijN_z}^{n+1} = \frac{2h_{3;ijN_z}^w \psi_{s;ij}^w + h_{3;ijN_z}^c \psi_{ij,n_z-1}^{n+1}}{2h_{3;ijN_z}^w + h_{3;ijN_z}^c} \quad (12.428)$$

which can be more conveniently rewritten in “tridiagonal” form

$$-\frac{h_{3;ijN_z}^c}{2h_{3;ijN_z}^w + h_{3;ijN_z}^c} \psi_{ij,N_z-1}^{n+1} + \psi_{ijN_z}^{n+1} = \frac{2h_{3;ijN_z}^w}{2h_{3;ijN_z}^w + h_{3;ijN_z}^c} \psi_{s;ij}^w \quad (12.429)$$

Note that the second Neumann condition is semi-implicit whereas both Dirichlet conditions use a fully implicit formulation.

12.4.7.2 bottom boundary conditions

The bottom boundary conditions are similar to the ones at the surface.

1. Neumann condition with a prescribed (upward) bottom flux $F_{b;ij}^{\psi;w}$

$$D_{3;ij1}^w = F_{b;ij}^{\psi;w} \quad (12.430)$$

2. Neumann condition using a bottom transfer velocity

$$D_{3;ij1}^w = C_{b;ij}^{\psi} \left((1 - \theta_v) \psi_{ij1} + \theta_v \psi_{ij1}^{n+1} - \psi_{b;ij}^w \right) \quad (12.431)$$

where $C_{b;ij}^{\psi}$ is the transfer velocity and $\psi_{b;ij}^w$ a prescribed external value.

3. Dirichlet condition with a prescribed external value $\psi_{b;ij}^c$ at the first C-node above the bottom

$$\psi_{ij1}^{n+1} = \psi_{b;ij}^c \quad (12.432)$$

4. Dirichlet condition with a prescribed external value $\psi_{b;ij}^w$ at the bottom itself. In that case the value at the first node above the sea bed is determined by interpolation

$$\psi_{ij1}^{n+1} = \frac{2h_{3;ij2}^w \psi_{b;ij}^w + h_{3;ij1}^c \psi_{ij2}^{n+1}}{2h_{3;ij2}^w + h_{3;ij1}^c} \quad (12.433)$$

which can be more conveniently rewritten in “tridiagonal” form

$$\psi_{ij1}^{n+1} - \frac{h_{3;ij1}^c}{2h_{3;ij2}^w + h_{3;ij1}^c} \psi_{ij2}^{n+1} = \frac{2h_{3;ij2}^w}{2h_{3;ij2}^w + h_{3;ij1}^c} \psi_{b;ij}^w \quad (12.434)$$

Note that the second Neumann condition is semi-implicit whereas both Dirichlet conditions use a fully implicit formulation.

12.4.7.3 lateral boundary conditions

At the open boundaries the flux normal to the boundary is determined by the upwind scheme. This means that

$$F_{1;ijk}^u = \frac{1}{2}u_{f;ijk} \left((1 \pm s_{ijk})\psi_{ijk}^e + (1 \mp s_{ijk})\psi_{i:i-1,jk} \right) \quad (12.435)$$

at U-boundaries and

$$F_{2;ijk}^v = \frac{1}{2}v_{f;ijk} \left((1 \pm s_{ijk})\psi_{ijk}^e + (1 \mp s_{ijk})\psi_{i,j:j-1,k} \right) \quad (12.436)$$

at V-boundaries, where the upper (lower) sign applies at western/southern (eastern/northern) boundaries, the flow sign s_{ijk} is defined by (12.395) or (12.402) and ψ_{ijk}^e denotes an external profile of ψ at one-half grid distance outside the open boundary. The open boundary problem then consists in determining the external profile ψ^e . The following four methods are available

1. Zero gradient condition.

$$\psi_{ijk}^e = \psi_{i:i-1,jk} \quad \text{or} \quad F_{1;ijk}^u = u_{f;ijk}\psi_{i:i-1,jk} \quad (12.437)$$

$$\psi_{ijk}^e = \psi_{i,j:j-1,k} \quad \text{or} \quad F_{2;ijk}^v = v_{f;ijk}\psi_{i,j:j-1,k} \quad (12.438)$$

2. The external profile is prescribed.
3. Radiation condition using the baroclinic wave speed.

$$\psi_{ijk}^{e;u;n+1} = (1 - w_{ijk}^u)\psi_{ijk}^{e;u;n} + w_{ijk}^u\psi_{i:i-1,jk}^n \quad (12.439)$$

$$\psi_{ijk}^{e;v;n+1} = (1 - w_{ijk}^v)\psi_{ijk}^{e;v;n} + w_{ijk}^v\psi_{i,j:j-1,k}^n \quad (12.440)$$

The weight factors are given by

$$\begin{aligned} w_{ijk}^u &= \pm \frac{R\Delta t}{h_{1;ij}^u} \sqrt{g_{ij}^u H_{i:i-1,j}^{n+1;c}} \\ w_{ijk}^v &= \pm \frac{R\Delta t}{h_{2;ij}^v} \sqrt{g_{ij}^v H_{i,j:j-1,k}^{n+1;c}} \end{aligned} \quad (12.441)$$

where R is the prescribed ratio of the baroclinic to surface gravity wave speed. Default value is 0.03.

4. Orlanski condition (see equation (5.371)).

$$\psi_{ijk}^{e;u;n+1} = \left(1 - O_R(r_{1;ijk}^u, r_{2;ijk}^u, r_{3;ijk}^u) \right) \psi_{ijk}^{e;u;n}$$

$$+ O_R(r_{1;ijk}^u, r_{2;ijk}^u, r_{3;ijk}^u) \psi_{i:i-1,jk}^n \quad (12.442)$$

$$\begin{aligned} \psi_{ijk}^{e;v;n+1} = & \left(1 - O_R(r_{1;ijk}^v, r_{2;ijk}^v, r_{3;ijk}^v) \right) \psi_{ijk}^{e;v;n} \\ & + O_R(r_{1;ijk}^v, r_{2;ijk}^v, r_{3;ijk}^v) \psi_{i,j:j-1,k}^n \end{aligned} \quad (12.443)$$

where the Orlanski function O_R is defined by (12.266) and

$$r_{1;ijk}^u = \psi_{i:i-1,jk}^n, \quad r_{2;ijk}^u = \psi_{i:i-1,jk}^{n-1}, \quad r_{3;ijk}^u = \psi_{i+1:i-2,jk}^{n-1} \quad (12.444)$$

$$r_{1;ijk}^v = \psi_{i,j:j-1,k}^n, \quad r_{2;ijk}^v = \psi_{i,j:j-1,k}^{n-1}, \quad r_{3;ijk}^v = \psi_{i,j+1:j-2,k}^{n-1} \quad (12.445)$$

5. Thacher-Harleman condition

In case of a Thacher-Harleman type of open boundary condition (see Section 5.10.3.1 the external profile is specified as follows

$$\begin{aligned} \psi_{ijk} = & \psi_{ijk}^{out} + \frac{1}{2} \left(\psi_{ijk}^e - \psi_{ijk}^{out} \right) \max \left[\cos \left(\pi \left(1.0 - \frac{t_{ij}^{out}}{T_{ret}} \right) \right) + 1, 0 \right], \quad 0 \leq t_{ij}^{out} \leq T_{ret;k} \\ \psi_{ijk}(t) = & \psi_{ijk}^e \quad t_{ij}^{out} > T_{ret;k} \end{aligned} \quad (12.446)$$

where t_{ij}^{out} is the time since the last outflow, ψ_{ijk}^{out} the value of ψ_{ijk} at the last outflow and $T_{ret;k}$ the pre-defined returned time.

Advective fluxes normal to a closed (coastal) open boundary are set to zero.

12.4.8 Solution of the discretised equations for scalars

As for momentum, the discretised equations can be written in tridiagonal form, as given by (12.319). Expressions for the matrix components are given below for the case that no operator splitting is used. They are easily extended to the case with operator splitting.

When a Dirichlet boundary condition is used at the surface (bottom), the surface (bottom) value of ψ is determined by the boundary condition itself. This means that k below varies between k_{min} and k_{max} where k_{min} equals 1 for a Neumann (flux) condition and 2 for a Dirichlet condition at the bottom. Likewise, k_{max} equals N_z for a Neumann and $N_z - 1$ for a Dirichlet condition at the surface.

For simplicity, the i and j indices are omitted.

1. Time derivative.

The contribution of the time derivative is given by

$$A_k^t = 0, \quad B_k^t = 1, \quad C_k^t = 0, \quad D_k^t = \psi_k^n \quad (12.447)$$

where $k_{min} \leq k \leq k_{max}$.

2. Vertical advection.

The vertical advection term is split up into two contributions arising from the fluxes below and above a k -level. The former are given by

$$\begin{aligned} A_k^{a-} &= -\theta_a c_k^- (\alpha_k + f_k) \\ B_k^{a-} &= -\theta_a c_k^- (\beta_k - f_k) \\ C_k^{a-} &= 0 \\ D_k^{a-} &= (1 - \theta_a) c_k^- \left((\alpha_k + f_k) \psi_{k-1}^n + (\beta_k - f_k) \psi_k^n \right) \end{aligned} \quad (12.448)$$

where $2 \leq k \leq k_{max}$,

$$c_k^- = \frac{\Delta t \omega_k^w}{2h_{3;k}^c}, \quad f_k = \left(1 - \Omega(r_k^w) \right) s_k \quad (12.449)$$

and α_{ijk} , β_{ijk} , s_{ijk} , r_{ijk}^w are defined by (12.409) and (12.410).

The terms arising from the flux above the k -level, are

$$\begin{aligned} A_k^{a+} &= 0 \\ B_k^{a+} &= \theta_a c_k^+ (\alpha_{k+1} + f_{k+1}) \\ C_k^{a+} &= \theta_a c_k^+ (\beta_{k+1} - f_{k+1}) \\ D_k^{a+} &= -(1 - \theta_a) c_k^+ \left((\alpha_{k+1} + f_{k+1}) \psi_k^n + (\beta_{k+1} - f_{k+1}) \psi_{k+1}^n \right) \end{aligned} \quad (12.450)$$

where $k_{min} \leq k \leq N_z - 1$ and

$$c_k^+ = \frac{\Delta t \omega_{k+1}^w}{2h_{3;k}^c} \quad (12.451)$$

3. Vertical diffusion.

As for vertical advection the fluxes below and above a k -level are taken separately. The former are given by

$$A_k^{d-} = -\theta_v \frac{\Delta t \lambda_{T;k}^{\psi;w}}{h_{3;k}^c h_{3;k}^w}$$

$$\begin{aligned}
B_k^{d-} &= \theta_v \frac{\Delta t \lambda_{T;k}^{\psi;w}}{h_{3;k}^c h_{3;k}^w} \\
C_k^{d-} &= 0 \\
D_k^{d-} &= -(1 - \theta_v) \frac{\Delta t \lambda_{T;k}^{\psi;w}}{h_{3;k}^c h_{3;k}^w} (\psi_k^n - \psi_{k-1}^n)
\end{aligned} \tag{12.452}$$

where $2 \leq k \leq k_{max}$.

The terms taken from the flux above the k -level, are

$$\begin{aligned}
A_k^{d+} &= 0 \\
B_k^{d+} &= \theta_v \frac{\Delta t \lambda_{T;k+1}^{\psi;w}}{h_{3;k}^c h_{3;k+1}^w} \\
C_k^{d+} &= -\theta_v \frac{\Delta t \lambda_{T;k+1}^{\psi;w}}{h_{3;k}^c h_{3;k+1}^w} \\
D_k^{d+} &= (1 - \theta_v) \frac{\Delta t \lambda_{T;k+1}^{\psi;w}}{h_{3;k}^c h_{3;k+1}^w} (\psi_{k+1}^n - \psi_k^n)
\end{aligned} \tag{12.453}$$

where $k_{min} \leq k \leq N_z - 1$.

4. Other explicit terms.

All other terms are explicit. Their contributions can be written as

$$\begin{aligned}
A_k^e &= B_k^e = C_k^e = 0 \\
D_k^e &= \Delta t \left(\mathcal{T}_k^n - \tilde{\mathcal{A}}_{h1}^f(\psi)_k^n - \tilde{\mathcal{A}}_{h2}^f(\psi)_k^n + \mathcal{C}_{s1}^f(\psi)_k^n \right. \\
&\quad \left. + \mathcal{C}_{s2}^f(\psi)_k^n + \mathcal{C}_{s3}^f(\psi)_k^n + \mathcal{D}_{sh1}^f(\psi)_k^n + \mathcal{D}_{sh2}^f(\psi)_k^n \right)
\end{aligned} \tag{12.454}$$

where $k_{min} \leq k \leq k_{max}$.

5. Surface boundary conditions.

Contributions from the surface boundary conditions depends on the type of condition as described in Section 12.4.7.1.

- Neumann condition with a prescribed surface flux $F_s^{\psi;w}$.

$$A_{N_z}^s = B_{N_z}^s = C_{N_z}^s = 0, \quad D_{N_z}^s = \frac{\Delta t F_s^{\psi;w}}{h_{3;N_z}^c} \tag{12.455}$$

- Neumann condition using a surface transfer velocity.

$$A_{N_z}^s = C_{N_z}^s = 0, \quad B_{N_z}^s = \theta_v \frac{\Delta t C_s^\psi}{h_{3;N_z}^c}, \quad D_{N_z}^s = \frac{\Delta t C_s^\psi}{h_{3;N_z}^c} \left(\psi_s^w - (1 - \theta_v) \psi_{N_z}^n \right) \quad (12.456)$$

- Dirichlet condition with a prescribed external value ψ_s^c at the first node below the surface.

$$A_{N_z}^s = C_{N_z}^s = 0, \quad B_{N_z}^s = 1, \quad D_{N_z}^s = \psi_s^c \quad (12.457)$$

- Dirichlet condition with a prescribed external value ψ_s^w at the surface itself.

$$\begin{aligned} A_{N_z}^s &= -\frac{h_{3;N_z}^c}{2h_{3;N_z}^w + h_{3;N_z}^c} \\ B_{N_z}^s &= 1, \quad C_{N_z}^s = 0 \\ D_{N_z}^s &= \frac{2h_{3;N_z}^w \psi_s^w}{2h_{3;N_z}^w + h_{3;N_z}^c} \end{aligned} \quad (12.458)$$

6. Bottom boundary conditions.

Contributions from the bottom boundary conditions depends on the type of condition as described in Section 12.4.7.2.

- Neumann condition with a prescribed bottom flux $F_b^{\psi;w}$.

$$A_1^b = B_1^b = C_1^b = 0, \quad D_1^b = \frac{\Delta t F_b^{\psi;w}}{h_{3;1}^c} \quad (12.459)$$

- Neumann condition using a bottom transfer velocity.

$$A_1^b = C_1^b = 0, \quad B_1^b = \theta_v \frac{\Delta t C_b^\psi}{h_{3;1}^c}, \quad D_1^b = \frac{\Delta t C_b^\psi}{h_{3;1}^c} \left(\psi_b^w - (1 - \theta_v) \psi_1^n \right) \quad (12.460)$$

- Dirichlet condition with a prescribed external value ψ_b^c at the first node above the sea bed.

$$A_1^b = C_1^b = 0, \quad B_1^b = 1, \quad D_1^b = \psi_b^c \quad (12.461)$$

- Dirichlet condition with a prescribed external value ψ_b^w at the bottom itself.

$$\begin{aligned} A_1^b &= 0, \quad B_1^b = 1 \\ C_1^b &= -\frac{h_{3;1}^c}{2h_{3;2}^w + h_{3;1}^c} \\ D_1^b &= \frac{2h_{3;2}^w \psi_b^w}{2h_{3;2}^w + h_{3;1}^c} \end{aligned} \quad (12.462)$$

12.5 Turbulence transport equations

This section deals with the numerical solution of the transport equations (5.165) for turbulence energy k , (5.166) for the dissipation rate ε and (5.170) for the turbulent energy times mixing length kl . The discretisation algorithms are highly similar to the ones used for scalar quantities. Main differences are

- Turbulence variables are determined at W-nodes.
- Production terms are taken explicitly at the old time step t^n , whereas the sink terms are discretised in time using the “quasi-implicit” approach, proposed by [Patankar \(1980\)](#), or

$$\mathcal{P}(\psi) = \mathcal{P}(\psi^n), \quad \mathcal{S}(\psi) = \mathcal{P}(\psi^n) \frac{\psi^{n+1}}{\psi^n} \quad (12.463)$$

- Since the turbulent equations are solved before the momentum equations, the horizontal advective terms are discretised using the non-filtered current (u, v) .
- The discretisation algorithms for advection and diffusion are the same as for C-node scalar quantities, except that all quantities are displaced in the vertical. This means that 3-D variables (fluxes, advective velocities, diffusion coefficients, . . .), previously evaluated at (C,U,V,W,UW, VW)-nodes are now taken at respectively (W,UW,VW,C,U,V)-nodes.

The turbulence transport equations are generally written as

$$\begin{aligned} \frac{\partial \psi}{\partial t} + \mathcal{A}_{h1}(\psi) + \mathcal{A}_{h2}(\psi) + \mathcal{A}_v(\psi) - \mathcal{C}_{s1}(\psi) - \mathcal{C}_{s2}(\psi) - \mathcal{C}_{s3}(\psi) \\ = \mathcal{P}(\psi) - \mathcal{S}(\psi) + \mathcal{D}_{sv}(\psi) + \mathcal{D}_{sh1}(\psi) + \mathcal{D}_{sh2}(\psi) \end{aligned} \quad (12.464)$$

where the corrector and horizontal advective operators \mathcal{C}_{si} and \mathcal{A}_{hi} are given by (12.371)–(12.373) and (12.376)–(12.377) with (u_f, v_f) replaced by (u, v) .

Despite the similarities with the previous section, the discretisation methods are described in detail below, to avoid any confusion.

12.5.1 Time discretisation

Three cases can be distinguished for the time integration. They are discussed in the subsections below.

12.5.1.1 integration without advection

In the absence of physical advection (`iopt_adv_scal=0`) the transport equation is integrated in time using

$$\begin{aligned} \frac{h_3^{n+1;w} \psi^{n+1;w} - h_3^{n;w} \psi^{n;w}}{h_3^{n+1;w} \Delta t} &= \theta_v \mathcal{D}_{sv}(\psi^{n+1;w}) + (1 - \theta_v) \mathcal{D}_{sv}(\psi^{n;w}) + \mathcal{P}(\psi^{n;w}) \\ &\quad - \mathcal{S}(\psi^{n;w}) \frac{\psi^{n+1;w}}{\psi^{n;w}} + \mathcal{D}_{sh1}(\psi^{n;w}) + \mathcal{D}_{sh2}(\psi^{n;w}) \end{aligned} \quad (12.465)$$

12.5.1.2 integration with advection but without operator splitting

If `iopt_adv_turb=1` or `2`, the transport equation (12.464) is integrated in time using

$$\begin{aligned} \frac{\psi^{n+1;w} - \psi^{n;w}}{\Delta t} &= -\mathcal{A}_{h1}(\psi^{n;w}) + \mathcal{C}_{s1}(\psi^{n;w}) - \mathcal{A}_{h2}(\psi^{n;w}) + \mathcal{C}_{s2}(\psi^{n;w}) \\ &\quad - \theta_a \mathcal{A}_v(\psi^{n+1;w}) - (1 - \theta_a) \mathcal{A}_v(\psi^{n;w}) + \mathcal{C}_{s3}(\psi^{n;w}) \\ &\quad + \theta_v \mathcal{D}_{sv}(\psi^{n+1;w}) + (1 - \theta_v) \mathcal{D}_{sv}(\psi^{n;w}) + \mathcal{P}(\psi^{n;w}) \\ &\quad - \mathcal{S}(\psi^{n;w}) \frac{\psi^{n+1;w}}{\psi^{n;w}} + \mathcal{D}_{sh1}(\psi^{n;w}) + \mathcal{D}_{sh2}(\psi^{n;w}) \end{aligned} \quad (12.466)$$

12.5.1.3 integration with operator splitting

If `iopt_adv_turb=3`, integration is performed along the following steps:

- Part A

$$\frac{\psi_A^{n+1/3;w} - \psi^{n;w}}{\Delta t} = -\mathcal{A}_{h1}(\psi^{n;w}) + \mathcal{C}_{s1}(\psi^{n;w}) + \mathcal{D}_{sh1}(\psi^{n;w}) \quad (12.467)$$

$$\begin{aligned} \frac{\psi_A^{n+2/3;w} - \psi_A^{n+1/3;w}}{\Delta t} &= -\mathcal{A}_{h2}(\psi_A^{n+1/3;w}) + \mathcal{C}_{s2}(\psi^{n;w}) \\ &\quad + \mathcal{D}_{sh2}(\psi_A^{n+1/3;w}) \end{aligned} \quad (12.468)$$

$$\begin{aligned} \frac{\psi_A^{n+1;w} - \psi_A^{n+2/3;w}}{\Delta t} &= -\theta_a \mathcal{A}_v(\psi_A^{n+1;w}) - (1 - \theta_a) \mathcal{A}_v(\psi_A^{n+2/3;w}) \\ &\quad + \mathcal{C}_{s3}(\psi^{n;w}) + \theta_v \mathcal{D}_{sv}(\psi_A^{n+1;w}) \\ &\quad + (1 - \theta_v) \mathcal{D}_{sv}(\psi_A^{n+2/3;w}) + \mathcal{P}(\psi^{n;w}) \\ &\quad - \mathcal{S}(\psi^{n+2/3;w}) \frac{\psi^{n+1;w}}{\psi^{n+2/3;w}} \end{aligned} \quad (12.469)$$

- Part B

$$\begin{aligned} \frac{\psi_B^{n+1/3;w} - \psi^{n;w}}{\Delta t} &= -\theta_a \mathcal{A}_v(\psi_B^{n+1/3;w}) - (1 - \theta_a) \mathcal{A}_v(\psi^{n;w}) \\ &\quad + \mathcal{C}_{s3}(\psi^{n;w}) + \theta_v \mathcal{D}_{sv}(\psi_B^{n+1/3;w}) \\ &\quad + (1 - \theta_v) \mathcal{D}_{sv}(\psi^{n;w}) + \mathcal{P}(\psi^{n;w}) \\ &\quad - \mathcal{S}(\psi^{n;w}) \frac{\psi^{n+1/3;w}}{\psi^{n;w}} \end{aligned} \quad (12.470)$$

$$\frac{\psi_B^{n+2/3;w} - \psi_B^{n+1/3;w}}{\Delta t} = -\mathcal{A}_{h2}(\psi_B^{n+1/3;w}) + \mathcal{C}_{s2}(\psi^{n;w}) + \mathcal{D}_{sh2}(\psi_B^{n+1/3;w}) \quad (12.471)$$

$$\frac{\psi_B^{n+1;w} - \psi_B^{n+2/3;w}}{\Delta t} = -\mathcal{A}_{h1}(\psi_B^{n+2/3;w}) + \mathcal{C}_{s1}(\psi^{n;w}) + \mathcal{D}_{sh1}(\psi_B^{n+2/3;w}) \quad (12.472)$$

- Updated value

$$\psi^{n+1;w} = \frac{1}{2}(\psi_A^{n+1;w} + \psi_B^{n+1;w}) \quad (12.473)$$

As before, the implicit factors are given by $\theta_a = 0.501$, $\theta_v = 1$.

12.5.2 Discretisation of advection

12.5.2.1 advection in the X-direction

The advective term in the X-direction is obtained by differencing the flux F_1^{uw} at the W-node

$$\mathcal{A}_{h1}(\psi)_{ijk}^w = \frac{h_{2;i+1,j}^u h_{3;i+1,jk}^{uw} F_{1;i+1,jk}^{uw} - h_{2;ij}^u h_{3;ijk}^{uw} F_{1;ijk}^{uw}}{h_{1;ij}^c h_{2;ij}^c h_{3;ijk}^w} \quad (12.474)$$

The flux is calculated from

$$F_{1;ij}^{uw} = \left(1 - \Omega(r_{ijk}^{uw})\right) F_{up;ijk}^{uw} + \Omega(r_{ijk}^{uw}) F_{lw;ijk}^{uw} \quad (12.475)$$

where $F_{up;ijk}^{uw}$ and $F_{lw;ijk}^{uw}$ are the upwind and Lax-Wendroff fluxes at the UW-node:

$$F_{up;ijk}^{uw} = \frac{1}{2} u_{ijk}^{uw} \left((\alpha_{ij} + s_{ijk}) \psi_{i-1,jk}^w + (\beta_{ij} - s_{ijk}) \psi_{ijk}^w \right) \quad (12.476)$$

$$F_{lw;ijk}^{uw} = \frac{1}{2} u_{ijk}^{uw} \left((\alpha_{ij} + c_{ijk}) \psi_{i-1,jk}^w + (\beta_{ij} - c_{ijk}) \psi_{ijk}^w \right) \quad (12.477)$$

where s_{ijk} and c_{ijk} are the sign and CFL number of the advecting current

$$s_{ijk} = \text{Sign}(u_{ijk}^{uw}), \quad c_{ijk} = \frac{u_{ijk}^{uw} \Delta t}{h_{1;ij}^u} \quad (12.478)$$

$$\alpha_{ij} = \frac{h_{1;ij}^c}{h_{1;ij}^u}, \quad \beta_{ij} = \frac{h_{1;i-1,j}^c}{h_{1;ij}^u} \quad (12.479)$$

The form of the weighting function is given by (12.47)–(12.50), depending on the type of advection scheme, selected by the switch `iopt_adv_turb`. The argument r of the weight function is defined by

$$\begin{aligned} r_{ijk}^u &= \frac{(\alpha_{ij} + s_{ijk}) \Delta F_{i-1,jk}^{uw} + (\beta_{ij} - s_{ijk}) \Delta F_{i+1,jk}^{uw}}{2 \Delta F_{ijk}^{uw}} \\ \Delta F_{ijk}^{uw} &= F_{lw;ijk}^{uw} - F_{up;ijk}^{uw} \end{aligned} \quad (12.480)$$

12.5.2.2 advection in the Y-direction

The advective term in the Y-direction is obtained by differencing the flux F_1^{vw} at the W-node

$$\mathcal{A}_{h2}(\psi)_{ijk}^w = \frac{h_{1;i,j+1}^v h_{3;i,j+1,k}^{vw} F_{2;i,j+1,k}^{vw} - h_{1;ij}^v h_{3;ijk}^{vw} F_{2;ijk}^{vw}}{h_{1;ij}^c h_{2;ij}^c h_{3;ijk}^w} \quad (12.481)$$

The flux is calculated from

$$F_{1;ij}^{vw} = \left(1 - \Omega(r_{ijk}^{vw})\right) F_{up;ijk}^{vw} + \Omega(r_{ijk}^{vw}) F_{lw;ijk}^{vw} \quad (12.482)$$

where $F_{up;ijk}^{vw}$ and $F_{lw;ijk}^{vw}$ are the upwind and Lax-Wendroff fluxes at the VW-node:

$$F_{up;ijk}^{vw} = \frac{1}{2} v_{ijk}^{vw} \left((\alpha_{ij} + s_{ijk}) \psi_{i,j-1,k}^w + (\beta_{ij} - s_{ijk}) \psi_{ijk}^w \right) \quad (12.483)$$

$$F_{lw;ijk}^{vw} = \frac{1}{2} v_{ijk}^{vw} \left((\alpha_{ij} + c_{ijk}) \psi_{i,j-1,k}^w + (\beta_{ij} - c_{ijk}) \psi_{ijk}^w \right) \quad (12.484)$$

where s_{ijk} and c_{ijk} are the sign and CFL number of the advecting current

$$s_{ijk} = \text{Sign}(v_{ijk}^{vw}), \quad c_{ijk} = \frac{v_{ijk}^{vw} \Delta t}{h_{2;ij}^v} \quad (12.485)$$

$$\alpha_{ij} = \frac{h_{2;ij}^c}{h_{2;ij}^v}, \quad \beta_{ij} = \frac{h_{2;i,j-1}^c}{h_{2;ij}^v} \quad (12.486)$$

The form of the weighting function is given by (12.47)–(12.50), depending on the type of advection scheme, selected by the switch `iopt_adv_turb`. The argument r of the weight function is defined by

$$\begin{aligned} r_{ijk}^{vw} &= \frac{(\alpha_{ij} + s_{ijk})\Delta F_{i,j-1,k}^{vw} + (\beta_{ij} - s_{ijk})\Delta F_{i,j+1,k}^{vw}}{2\Delta F_{ijk}^{vw}} \\ \Delta F_{ijk}^{vw} &= F_{lw;ijk}^{vw} - F_{up;ijk}^{vw} \end{aligned} \quad (12.487)$$

12.5.2.3 advection in the vertical direction

The vertical advective term is obtained by differencing the flux F_3^c at the W-node

$$\mathcal{A}_v(\psi)_{ijk}^w = \frac{F_{3;ijk}^c - F_{3;ij,k-1}^c}{h_{3;ijk}^w} \quad (12.488)$$

The flux is calculated from

$$F_{3;ijk}^c = (1 - \Omega(r_{ijk}^c)) F_{up;ijk}^c + \Omega(r_{ijk}^c) F_{ce;ijk}^c \quad (12.489)$$

where $F_{up;ijk}^c$ and $F_{ce;ijk}^c$ are the upwind and central fluxes at the C-node:

$$F_{up;ijk}^c = \frac{1}{2} \omega_{ijk}^c \left((1 + s_{ijk}) \psi_{ijk}^w + (1 - s_{ijk}) \psi_{ij,k+1}^w \right) \quad (12.490)$$

$$F_{ce;ijk}^c = \frac{1}{2} \omega_{ijk}^c (\psi_{ij,k}^w + \psi_{ij,k+1}^w) \quad (12.491)$$

where

$$s_{ijk} = \text{Sign}(\omega_{ijk}^c) \quad (12.492)$$

The form of the weighting function is given by (12.47)–(12.50), depending on the type of advection scheme, selected by the switch `iopt_adv_scal`. The argument r of the weight function is defined by

$$\begin{aligned} r_{ijk}^w &= \frac{(1 + s_{ijk})\Delta F_{ij,k-1}^c + (1 - s_{ijk})\Delta F_{ij,k+1}^c}{2\Delta F_{ijk}^c} \\ \Delta F_{ijk}^c &= F_{ce;ijk}^c - F_{up;ijk}^c \end{aligned} \quad (12.493)$$

12.5.2.4 corrector terms

The corrector terms are discretised using

$$\mathcal{C}_{s1}(\psi)^w = \psi_{ijk}^w \frac{h_{2;i+1,j}^u h_{3;i+1,jk}^{uw} u_{i+1,jk}^{uw} - h_{2;ij}^u h_{3;ijk}^{uw} u_{ijk}^{uw}}{h_{1;ij}^c h_{2;ij}^c h_{3;ijk}^w} \quad (12.494)$$

$$\mathcal{C}_{s2}(\psi)^w = \psi_{ijk}^w \frac{h_{1;i,j+1}^v h_{3;i,j+1,k}^{vw} v_{i,j+1,k}^{vw} - h_{1;ij}^v h_{3;ijk}^{vw} v_{ijk}^{vw}}{h_{1;ij}^c h_{2;ij}^c h_{3;ijk}^w} \quad (12.495)$$

$$\mathcal{C}_{s3}(\psi)^w = \psi_{ijk}^w \frac{\omega_{ij,k+1}^c - \omega_{ijk}^c}{h_{3;ijk}^w} \quad (12.496)$$

12.5.3 Discretisation of diffusion

12.5.3.1 diffusion in the X-direction

The diffusion term in the X-direction is obtained by differencing the flux D_1^{uw} at the W-node

$$\mathcal{D}_{sh1}(\psi)_{ijk}^w = \frac{h_{2;i+1,j}^u h_{3;i+1,jk}^{uw} D_{1;i+1,jk}^{uw} - h_{2;ij}^u h_{3;ijk}^{uw} D_{1;ijk}^{uw}}{h_{1;ij}^c h_{2;ij}^c h_{3;ijk}^w} \quad (12.497)$$

The flux is given by

$$D_{1;ijk}^{uw} = \frac{\lambda_H^{uw} (\psi_{ijk}^w - \psi_{i-1,jk}^w)}{h_{1;ij}^u} \quad (12.498)$$

12.5.3.2 diffusion in the Y-direction

The diffusion term in the Y-direction is obtained by differencing the flux D_2^w at the W-node

$$\mathcal{D}_{sh2}(\psi)_{ijk}^w = \frac{h_{1;i,j+1}^v h_{3;i,j+1,k}^{vw} D_{2;i,j+1,k}^{vw} - h_{1;ij}^v h_{3;ijk}^{vw} D_{2;ijk}^{vw}}{h_{1;ij}^c h_{2;ij}^c h_{3;ijk}^w} \quad (12.499)$$

The flux is given by

$$D_{2;ijk}^{vw} = \frac{\lambda_H^{vw} (\psi_{ijk}^w - \psi_{i,j-1,k}^w)}{h_{2;ij}^v} \quad (12.500)$$

12.5.3.3 diffusion in the vertical direction

The vertical diffusion term is obtained by differencing the flux D_3^c at the W-node

$$\mathcal{D}_{sv}(\psi_{ijk})^w = \frac{D_{3;ijk}^c - D_{3;ij,k-1}^c}{h_{3;ijk}^w} \quad (12.501)$$

The flux is given by

$$D_{3;ijk}^c = \frac{\lambda_T^{\psi;c} (\psi_{ij,k+1}^w - \psi_{ijk}^w)}{h_{3;ijk}^c} \quad (12.502)$$

12.5.4 Diffusion coefficients for turbulence variables

12.5.4.1 horizontal diffusion coefficients

The horizontal turbulent diffusion coefficients are the same as the one used for scalar transport. They are obtained by vertical interpolation of λ_H^u and λ_H^v , given by (12.423)–(12.424) to respectively the UW and VW-nodes.

12.5.4.2 vertical diffusion coefficients

The vertical turbulent diffusion coefficients, used in the ε -equation (5.166) and kl -equation (5.170), are proportional to ν_k which is the one used in the k -equation (5.165). Different formulations for the parameterisation of ν_k are available and discussed in Section 5.3.3.3. No specific discretisation procedures are required since all expressions are purely algebraic.

The following remarks are to be given

- Values are first obtained at the W-nodes and then interpolated at the C-nodes.
- No value is calculated at the surface and the bottom.
- Since ν_k is calculated prior to the solution of the turbulent transport equations, its value is obtained using values of k , ε and l at the old time t^n .

12.5.5 Production and sink terms

All turbulence transport equations contain, besides the diffusion terms, three terms on their right hand side. The first is the shear production term, the second is the buoyancy term which is a production or sink term if $N^2 < 0$ or $N^2 > 0$ and the third is a dissipation (sink) term. Defining

$$N^2 = \max(N^2, 0) + \min(N^2, 0) = N_+^2 - N_-^2 \quad (12.503)$$

one has

$$\begin{aligned} \mathcal{P}(k) &= \nu_T M^2 + \lambda_T N_-^2 \\ \mathcal{P}(\varepsilon) &= c_{1\varepsilon} \frac{\epsilon}{k} \left(\nu_T M^2 + c_{3\varepsilon} \lambda_T N_-^2 \right) \\ \mathcal{P}(kl) &= \frac{1}{2} l \left(E_1 \nu_T M^2 + E_3 \lambda_T N_-^2 \right) \end{aligned} \quad (12.504)$$

and

$$\begin{aligned} \mathcal{S}(k) &= \lambda_T N_+^2 + \varepsilon \\ \mathcal{S}(\varepsilon) &= c_{1\varepsilon} c_{3\varepsilon} \lambda_T N_+^2 + c_{2\varepsilon} \frac{\varepsilon^2}{k} \\ \mathcal{S}(kl) &= \frac{1}{2} \left(l E_3 \lambda_T N_+^2 + \epsilon_0 k^{3/2} \tilde{W} \right) \end{aligned} \quad (12.505)$$

The discretisation of M^2 , N_\pm^2 , ν_T and λ_T is discussed in Section 12.3.12.2. Production terms are taken explicit in time using values of all quantities at time t^n . Sink terms are discretised quasi-implicitly using (12.463):

$$\begin{aligned}\mathcal{S}(k) &= (\lambda_T N_+^2 + \varepsilon^n) \frac{k^{n+1}}{k^n} \\ \mathcal{S}(\varepsilon) &= c_{1\varepsilon} c_{3\varepsilon} \lambda_T N_+^2 \frac{\varepsilon^{n+1}}{\varepsilon^n} + c_{2\varepsilon} \frac{\varepsilon^n \varepsilon^{n+1}}{k^n} \\ \mathcal{S}(kl) &= \frac{1}{2} \left(E_3 \frac{\lambda_T N_+^2}{k^n} + \epsilon_0 \frac{(kl)^{n+1} \sqrt{k^n} \tilde{W}}{l^n} \right)\end{aligned}\quad (12.506)$$

12.5.6 Boundary conditions

12.5.6.1 surface boundary conditions

In analogy with the scalar case two Neumann and two Dirichlet type of surface boundary conditions are available in the program.

1. Neumann condition with a prescribed flux $F_{s;N_z+1}^{\psi;w}$ at the surface

$$D_{3;ij,N_z+1}^w = F_{s;ij,N_z+1}^{\psi;w} \quad (12.507)$$

The flux at the first C-node below the surface is then determined by interpolating the surface value and the calculated flux at the second C-node below the surface

$$\begin{aligned}D_{3;ij,N_z}^c &= \frac{2h_{3;ij,N_z}^w F_{s;ij,N_z+1}^{\psi;w} + h_{3;ij,N_z}^c D_{3;ij,N_z-1}^c}{2h_{3;ij,N_z}^w + h_{3;ij,N_z}^c} \\ &= \frac{2h_{3;ij,N_z}^w F_{s;ij,N_z+1}^{\psi;w}}{2h_{3;ij,N_z}^w + h_{3;ij,N_z}^c} + \frac{h_{3;ij,N_z}^c \lambda_{T;ij,N_z-1}^{\psi;c} (\psi_{ij,N_z}^w - \psi_{ij,N_z-1}^w)}{h_{3;ij,N_z-1}^c (2h_{3;ij,N_z}^w + h_{3;ij,N_z}^c)}\end{aligned}\quad (12.508)$$

with

$$\psi_{ij,N_z}^w - \psi_{ij,N_z-1}^w = \theta_v (\psi_{ij,N_z}^{n+1;w} - \psi_{ij,N_z-1}^{n+1;w}) + (1 - \theta_v) (\psi_{ij,N_z}^{n;w} - \psi_{ij,N_z-1}^{n;w}) \quad (12.509)$$

2. Neumann using a prescribed flux $F_{s;ij,N_z}^{\psi;c}$ at the first C-node below the surface

$$D_{3;ij,N_z}^c = F_{s;ij,N_z}^{\psi;c} \quad (12.510)$$

3. Dirichlet condition with a prescribed value $\psi_{s;ij,N_z+1}^w$ at the surface

$$\psi_{ij,N_z+1} = \psi_{s;ij,N_z+1}^w \quad (12.511)$$

Table 12.4: Discretisation schemes for each of the available surface and bottom boundary conditions for turbulent variables.

variable	equation	discretisation scheme
k	(5.245)	Dirichlet at the surface
ε	(5.245)	Dirichlet at the first W-node below the surface
l	(5.245)	Dirichlet at the first W-node below the surface
k	(5.247)	prescribed flux at the surface
ε	(5.248)	prescribed flux at the first C-node below the surface
k	(5.330)	Dirichlet at the bottom
ε	(5.330)	Dirichlet at the first W-node above the sea bed
l	(5.330)	Dirichlet at the first W-node above the sea bed
k	(5.331)	prescribed flux at the bottom
ε	(5.332)	prescribed flux at the first C-node above the sea bed

4. Dirichlet condition with a prescribed value $\psi_{s;ijN_z}^w$ at the first W-node below the surface

$$\psi_{ijN_z} = \psi_{s;ijN_z}^w \quad (12.512)$$

Several formulations of surface boundary conditions have been introduced in Section 5.6.5. The discretisation scheme for each formulation is indicated in Table 12.4.

It is remarked that all turbulent diffusion coefficients are calculated using those values of k , ε and l which are located within the water column and not at the surface itself so that (12.510) and (12.512) can be considered as realistic conditions.

12.5.6.2 bottom boundary conditions

In analogy with the scalar case two Neumann and three Dirichlet type of bottom boundary conditions are available in the program.

1. Neumann condition with a prescribed flux $F_{b;ij1}^{\psi;w}$ at the bottom

$$D_{3;ij1}^w = F_{b;ij1}^{\psi;w} \quad (12.513)$$

The flux at the first C-node above the sea bed is then determined by interpolating the bottom value and the calculated flux at the second C-node above the sea bed

$$D_{3;ij1}^c = \frac{2h_{3;ij2}^w F_{b;ij}^{\psi;w} + h_{3;ij1}^c D_{3;ij2}^c}{2h_{3;ij2}^w + h_{3;ij1}^c}$$

$$= \frac{2h_{3;ij2}^w F_{b;ij}^{\psi;w}}{2h_{3;ij2}^w + h_{3;ij1}^c} + \frac{h_{3;ij1}^c \lambda_{T;ij2}^{\psi;c} (\psi_{ij3}^w - \psi_{ij2}^w)}{h_{3;ij2}^c (2h_{3;ij2}^w + h_{3;ij1}^c)} \quad (12.514)$$

with

$$\psi_{ij3}^w - \psi_{ij2}^w = \theta_v (\psi_{ij3}^{n+1;w} - \psi_{ij2}^{n+1;w}) + (1 - \theta_v) (\psi_{ij3}^{n;w} - \psi_{ij2}^{n;w}) \quad (12.515)$$

2. Neumann using a prescribed flux $F_{b;ij1}^{\psi;c}$ at the first C-node above the bottom

$$D_{3;ij1}^c = F_{b;ij1}^{\psi;c} \quad (12.516)$$

3. Dirichlet condition with a prescribed value $\psi_{b;ij1}^w$ at the bottom

$$\psi_{ij1} = \psi_{b;ij1}^w \quad (12.517)$$

4. Dirichlet condition with a prescribed value $\psi_{b;ij2}^w$ at the first W-node above the bottom

$$\psi_{ij2} = \psi_{b;ij2}^w \quad (12.518)$$

Several formulations of bottom boundary conditions have been introduced in Section 5.9.4. The discretisation scheme for each formulation is indicated in Table 12.4.

12.5.6.3 lateral boundary conditions

The fluxes normal to an open boundary are calculated using the upwind scheme. Applying the zero gradient condition one obtains

$$\psi_{ijk}^{e;w} = \psi_{i:i-1,jk}^w \quad \text{or} \quad F_{1;ijk}^{uw} = u_{ijk}^{uw} \psi_{i:i-1,jk}^w \quad (12.519)$$

$$\psi_{ijk}^{e;w} = \psi_{i,j:j-1,k}^w \quad \text{or} \quad F_{2;ijk}^{vw} = v_{ijk}^{vw} \psi_{i,j:j-1,k}^w \quad (12.520)$$

Advective fluxes normal to a closed (coastal) open boundary are set to zero.

12.5.7 Solution of the discretised equations for turbulent transport variables

As for momentum, the discretised equations can be written in the tridiagonal form (12.319). Expressions for the matrix components are given below for the case that no operator splitting is used. They are easily extended to the case with operator splitting.

When a Neumann boundary condition is taken, no calculation is performed at the surface or bottom itself. In case of a Dirichlet condition, the surface (bottom) value of ψ is determined by the boundary condition itself. This means that the vertical index k varies between k_{min} and k_{max} . The lower limit k_{min} equals 3 for a Dirichlet condition at the first W-node above the bottom and 2 otherwise. Likewise k_{max} equals $N_z - 1$ for a Dirichlet condition at the first W-node below the surface and N_z otherwise.

For simplicity, the i and j indices are omitted.

1. Time derivative.

The contribution of the time derivative is given by

$$A_k^t = 0, \quad B_k^t = 1, \quad C_k^t = 0, \quad D_k^t = \psi_k^{n;w} \quad (12.521)$$

where $k_{min} \leq k \leq k_{max}$.

2. Vertical advection.

The vertical advection term is split up into two contributions arising from the fluxes below and above a k -level. The former are given by

$$\begin{aligned} A_k^{a-} &= -\theta_a c_k^- (1 + f_{k-1}) \\ B_k^{a-} &= -\theta_a c_k^- (1 - f_{k-1}) \\ C_k^{a-} &= 0 \\ D_k^{a-} &= (1 - \theta_a) c_k^- \left((1 + f_{k-1}) \psi_{k-1}^{n;w} + (1 - f_{k-1}) \psi_k^{n;w} \right) \end{aligned} \quad (12.522)$$

where $k_{min} \leq k \leq k_{max}$,

$$c_k^- = \frac{\Delta t \omega_{k-1}^c}{2h_{3;k}^w}, \quad f_k = \left(1 - \Omega(r_k^c) \right) s_k \quad (12.523)$$

and s_{ijk} , r_{ijk}^w are defined by (12.492) and (12.493).

The terms arising from the flux above the k -level, are

$$\begin{aligned} A_k^{a+} &= 0 \\ B_k^{a+} &= \theta_a c_k^+ (1 + f_k) \\ C_k^{a+} &= \theta_a c_k^+ (1 - f_k) \\ D_k^{a+} &= -(1 - \theta_a) c_k^+ \left((1 + f_k) \psi_k^{n;w} + (1 - f_k) \psi_{k+1}^{n;w} \right) \end{aligned} \quad (12.524)$$

where $k_{min} \leq k \leq k_{max}$ and

$$c_k^+ = \frac{\Delta t \omega_k^c}{2h_{3;k}^w} \quad (12.525)$$

3. Vertical diffusion.

As for vertical advection the fluxes below and above a k -level are taken separately. The former are given by

$$\begin{aligned} A_k^{d-} &= -\theta_v \frac{\Delta t \lambda_{T;k-1}^{\psi;c}}{h_{3;k-1}^c h_{3;k}^w} \\ B_k^{d-} &= \theta_v \frac{\Delta t \lambda_{T;k-1}^{\psi;c}}{h_{3;k-1}^c h_{3;k}^w} \\ C_k^{d-} &= 0 \\ D_k^{d-} &= -(1 - \theta_v) \frac{\Delta t \lambda_{T;k-1}^{\psi;c}}{h_{3;k-1}^c h_{3;k}^w} (\psi_k^{n;w} - \psi_{k-1}^{n;w}) \end{aligned} \quad (12.526)$$

where $k_{lo} \leq k \leq k_{max}$ and k_{lo} equals 2 for a Dirichlet condition at the bottom and 3 otherwise.

The terms taken from the flux above the k -level, are

$$\begin{aligned} A_k^{d+} &= 0 \\ B_k^{d+} &= \theta_v \frac{\Delta t \lambda_{T;k}^{\psi;c}}{h_{3;k}^c h_{3;k}^w} \\ C_k^{d+} &= -\theta_v \frac{\Delta t \lambda_{T;k}^{\psi;c}}{h_{3;k}^c h_{3;k}^w} \\ D_k^{d+} &= (1 - \theta_v) \frac{\Delta t \lambda_{T;k}^{\psi;c}}{h_{3;k}^c h_{3;k}^w} (\psi_{k+1}^{n;w} - \psi_k^{n;w}) \end{aligned} \quad (12.527)$$

where $k_{min} \leq k \leq k_{up}$ and k_{up} equals N_z for a Dirichlet condition at the surface and N_z-1 otherwise.

4. Sink terms.

$$A_k^S = C_k^S = D_k^S = 0, \quad B_k^S = \frac{\mathcal{S}(\psi^{n;w})_k}{\psi_k^{n;w}} \quad (12.528)$$

where $k_{min} \leq k \leq k_{max}$.

5. Other explicit terms.

All other terms are explicit. Their contributions can be written as

$$\begin{aligned} A_k^e &= B_k^e = C_k^e = 0 \\ D_k^e &= \Delta t \left(\mathcal{P}_k^n - \tilde{\mathcal{A}}_{h1}(\psi^{n;w})_k^w - \tilde{\mathcal{A}}_{h2}(\psi^{n;w})_k^w + \mathcal{C}_{s1}(\psi^{n;w})_k^w \right) \end{aligned}$$

$$+ \mathcal{C}_{s2}(\psi^{n;w})_k^w + \mathcal{C}_{s3}(\psi^{n;w})_k^w + \mathcal{D}_{sh1}(\psi^{n;w})_k^w + \mathcal{D}_{sh2}(\psi^{n;w})_k^w \Big) \quad (12.529)$$

where $k_{min} \leq k \leq k_{max}$.

6. Surface boundary conditions.

Contributions from the surface boundary conditions depends on the type of condition as described in Section 12.5.6.1.

- Neumann condition with a prescribed surface flux $F_{s;N_z+1}^{\psi;w}$.

$$\begin{aligned} A_{N_z}^s &= \frac{\theta_v \Delta t h_{3;N_z}^c \lambda_{T;N_z-1}^{\psi;c}}{h_{3;N_z}^w (2h_{3;N_z}^w + h_{3;N_z}^c) h_{3;N_z-1}^c} \\ B_{N_z}^s &= -\frac{\theta_v \Delta t h_{3;N_z}^c \lambda_{T;N_z-1}^{\psi;c}}{h_{3;N_z}^w (2h_{3;N_z}^w + h_{3;N_z}^c) h_{3;N_z-1}^c} \\ C_{N_z}^s &= 0 \\ D_{N_z}^s &= \frac{\Delta t}{2h_{3;N_z}^w + h_{3;N_z}^c} \left(2F_{s;N_z+1}^{\psi;w} \right. \\ &\quad \left. + (1 - \theta_v) \frac{h_{3;N_z}^c \lambda_{T;N_z-1}^{\psi;c} (\psi_{N_z}^{n;w} - \psi_{N_z-1}^{n;w})}{h_{3;N_z}^w h_{3;N_z-1}^c} \right) \end{aligned} \quad (12.530)$$

- Neumann using a prescribed flux $F_{s;N_z}^{\psi;c}$ at the first C-node below the surface

$$A_{N_z}^s = B_{N_z}^s = C_{N_z}^s = 0, \quad D_{N_z}^s = \frac{\Delta t F_{s;N_z}^{\psi;c}}{h_{3;N_z}^w} \quad (12.531)$$

- Dirichlet condition with a prescribed value $\psi_{s;N_z+1}^w$ at the surface

$$A_{N_z+1}^s = C_{N_z+1}^s = 0, \quad B_{N_z+1}^s = 1, \quad D_{N_z+1}^s = \psi_{s;N_z+1}^w \quad (12.532)$$

- Dirichlet condition with a prescribed value $\psi_{s;N_z}^w$ at the first W-node below the surface

$$A_{N_z}^s = C_{N_z}^s = 0, \quad B_{N_z}^s = 1, \quad D_{N_z}^s = \psi_{s;N_z}^w \quad (12.533)$$

7. Bottom boundary conditions.

Contributions from the bottom boundary conditions depends on the type of condition as described in Section 12.5.6.2.

- Neumann condition with a prescribed bottom flux $F_{b;1}^{\psi;w}$ at the bottom

$$\begin{aligned}
 A_2^b &= 0 \\
 B_2^b &= -\frac{\theta_v \Delta t h_{3;1}^c \lambda_{T;2}^{\psi;c}}{h_{3;2}^w (2h_{3;2}^w + h_{3;1}^c) h_{3;2}^c} \\
 C_2^b &= \frac{\theta_v \Delta t h_{3;1}^c \lambda_{T;2}^{\psi;c}}{h_{3;2}^w (2h_{3;2}^w + h_{3;1}^c) h_{3;2}^c} \\
 D_2^b &= -\frac{\Delta t}{2h_{3;2}^w + h_{3;1}^c} \left(2F_{b;1}^{\psi;w} + (1 - \theta_v) \frac{h_{3;1}^c \lambda_{T;2}^{\psi;c} (\psi_3^{n;w} - \psi_2^{n;w})}{h_{3;2}^w h_{3;2}^c} \right)
 \end{aligned} \tag{12.534}$$

- Neumann using a prescribed flux $F_{b;1}^{\psi;c}$ at the first C-node above the bottom

$$A_2^b = B_2^b = C_2^s = 0, \quad D_2^b = -\frac{\Delta t F_{b;1}^{\psi;c}}{h_{3;2}^w} \tag{12.535}$$

- Dirichlet condition with a prescribed value $\psi_{b;1}^w$ at the bottom

$$A_1^b = C_1^b = 0, \quad B_1^b = 1, \quad D_1^b = \psi_{b;1}^w \tag{12.536}$$

- Dirichlet condition with a prescribed value $\psi_{b;2}^w$ at the first W-node above the bottom

$$A_2^b = C_2^b = 0, \quad B_2^b = 1, \quad D_2^b = \psi_{b;2}^w \tag{12.537}$$

12.6 Discretisations on reduced grids

12.6.1 Discretised 1-D mode equations

1. To make the code compatible with the 3-D case which uses an Arakawa C-grid, the model grid on which the equations are discretised, does not reduce to a single point but consists of 3 rows and columns (i.e. `nc=nr=3`) of which the last column and the last row consist of dummy land points (see Figure 4.7). This produces a computational overhead since the same calculation is performed at each of the four wet C-nodes and the two internal U and V velocity nodes.
2. Momentum equations

- The 1-D versions (5.68)– $\tilde{\text{A}}\S 5.69$ are integrated in time without operator and mode splitting using the formulations given in Sections 12.3.1.1 and 12.3.1.3. Firstly, “predicted” values are calculated

$$\frac{\tilde{u}^p - u^n}{\Delta t} = fv^n + \theta_v \mathcal{D}_{mv}(\tilde{u}^p) + (1 - \theta_v) \mathcal{D}_{mv}(u^n) - g \frac{\partial \zeta^{n+1}}{\partial x} + F_1^{t;n+1} \quad (12.538)$$

$$\frac{\tilde{v}^p - v^n}{\Delta t} = -fu^n + \theta_v \mathcal{D}_{mv}(\tilde{v}^p) + (1 - \theta_v) \mathcal{D}_{mv}(v^n) - g \frac{\partial \zeta^{n+1}}{\partial y} + F_2^{t;n+1} \quad (12.539)$$

An implicit correction is added for the Coriolis force giving (u^p, v^p) by equations (12.7). “Corrected” values are obtained by

$$u^{n+1} = \frac{h_{3;k}^n}{h_{3;k}^{n+1}} u^p, \quad v^{n+1} = \frac{h_{3;k}^n}{h_{3;k}^{n+1}} v^p \quad (12.540)$$

where the surface slopes and the elevations used to calculate the vertical grid spacing $h_{3;k}^{n+1} = (h + \zeta^{n+1})\Delta\sigma_k$ are prescribed externally by expressions of the form (5.70).

- The vertical diffusion terms and coefficients are discretised using the formulations given in Sections 12.3.11 and 12.3.12.2.
- Once the currents are updated, the depth-mean currents \bar{u} and \bar{v} are evaluated (for user output only).

3. Scalar equations

- The transport equation for a scalar ψ is integrated in time without operator splitting using the discretisation (12.379).
 - The vertical diffusion term and coefficient are discretised as described in Sections 12.4.5.3 and 12.4.6.2.
 - Neumann and Dirichlet conditions can be applied as discussed in Sections 12.4.7.1–12.4.7.2.
4. The turbulence equations are solved as in the 3-D case without advection and operator splitting.

12.6.2 Discretised depth-integrated equations

1. Momentum equations

- The surface elevation and depth-integrated currents are updated by solving the 2-D momentum equations using the same discretisation procedures given in Sections 12.3 without the depth-integrated baroclinic terms but with the same barotropic time step $\Delta\tau$.
- To make the code compatible with the 3-D case all “3-D” currents are set to their depth-mean value

$$\begin{aligned} u_f &= \bar{u} = U/H = u \\ v_f &= \bar{v} = V/H = v \end{aligned} \quad (12.541)$$

2. Scalar equations

- Scalar transport is discretised in exactly the same way as in the 3-D case with the larger 3-D time step Δt .
- The vertical diffusion term is retained and discretised using (12.422) except that the upper and lower fluxes are located at the respectively the surface and the bottom and therefore obtained by the surface and bottom boundary conditions which must obviously be of the Neumann type.

3. No turbulence transport equations need to be solved.

12.7 Solution procedure

The general solution procedure can be summarised as follows

1. Initial time $t = 0$.
 - 1.1 Obtain initial conditions for $U, V, \zeta, u, v, \omega, T, S, k, l$ or ε , and quantities which are updated in time at open boundaries.
 - 1.2 Initialise ρ, β_T, β_S from the equation of state.
 - 1.3 Evaluate the astronomical force at the initial time.
 - 1.4 Initialise meteorological data.
 - 1.5 Initialise open boundary data for the 2-D mode, 3-D mode, temperature, salinity.
 - 1.6 Initialise surface and bottom stress.
2. Predictor step at $t = t^p = t^n + \Delta\tau$ with $n=0, \dots, N_{tot} - 1$.
 - 2.1 Update meteorological data (if needed).

- 2.2 Update ρ, β_T, β_S from the equation of state.
- 2.3 Update all vertical diffusion coefficients. In case of a RANS model, k, l or ε are first updated at time t^{n+1} .
- 2.4 Evaluate the components of the baroclinic pressure gradient.
- 2.5 Evaluate the horizontal diffusion coefficients at different nodes.
- 2.6 Obtain u^p, v^p by solving the 3-D momentum equations.
3. Barotropic time steps $t = t^n + m\Delta\tau$ with $m=1, \dots, M_t$.
 - 3.1 Update meteorological data (if needed).
 - 3.2 Solve 2-D continuity equation for ζ .
 - 3.3 Update open boundary data for the 2-D mode (if needed).
 - 3.4 Update astronomical force.
 - 3.5 Update U, V by solving the 2-D momentum equations.
 - 3.6 Update the time-averaged transports U_f, V_f .
4. Corrector step at $t = t^{n+1} = t^n + \Delta t$ with $n=1, \dots, N_{tot}$.
 - 4.1 3-D mode
 - 4.1.1 Update open boundary data (if needed).
 - 4.1.2 Apply open boundary conditions.
 - 4.1.3 Apply filter correction to obtain u^{n+1}, v^{n+1} .
 - 4.1.4 Evaluate filtered currents u_f, v_f .
 - 4.1.5 Solve 3-D baroclinic continuity equation for ω .
 - 4.1.6 Update physical vertical current.
 - 4.1.7 Update bottom and surface stress (if needed).
 - 4.2 Update temperature at time t^{n+1} .
 - 4.2.1 Update open boundary data (if needed).
 - 4.2.2 Apply open boundary conditions.
 - 4.2.3 Evaluate solar irradiance.
 - 4.2.4 Evaluate surface (non-solar) heat fluxes.
 - 4.2.5 Solve temperature equation.
 - 4.3 Update salinity at time t^{n+1} .
 - 4.3.1 Update open boundary data (if needed).
 - 4.3.2 Apply open boundary conditions.

4.3.3 Solve salinity equation.

Note that

- Some of the previous steps are only conditionally performed, depending on the setting of model switches. For example, the temperature equation is only updated when `iopt_temp=2`, the astronomical tidal force is only included if `iopt_astro_tide=1`,
- Update of surface or open boundary forcing data depends on the settings of the `tlms` attribute, discussed in Section ?? of the User Manual.

Part III

Description of the model code

Chapter 13

Program conventions and techniques

The following items are discussed in this chapter.

- The COHERENS V2.0 code is written in standard FORTRAN 90 format. To improve portability and transparancy a number of programming conventions, further denoted as the “COHERENS conventions”, have been adopted. They are described in Section 13.1. These rules are of importance for developers who are working on new developments and have the intention to make the new code available to the COHERENS community.
- Implementation of specific FORTRAN 90 features such as allocatable arrays, derived types, modules and generic routines, are discussed in Sections 13.1.3–13.1.6.
- The format for internal documentation is explained in Section 13.1.7.
- Basic aspects of the model code such as the principle of key ids, time formats, data flags and the units of program variables are discussed in Section 13.2.

13.1 Implementation of FORTRAN 90

A main difference between versions 1 and 2 of COHERENS is that the code in the old version is written in FORTRAN 77 and the latter in FORTRAN 90. This section discusses the programming conventions based upon specific features of FORTRAN 90. Users, who have no experience with FORTRAN 90 but are familiar with the FORTRAN 77 standard may consult the many books,

course notes and other publications available from commercial publishers or via the internet.

13.1.1 COHERENS programming conventions

1. All source code is written in “free format”. This implies the following:

- The column position is irrelevant. The adopted rule is that all program lines start at column 1, with exceptions of statements within control constructs such as IF blocks, DO loops and SELECT CASE constructs, which are intended by one TAB position to the right. Lines within a nested construct are intended with respect to the previous one. For clarity no indentation is applied if the DO/ENDDO statement of a nested loop is located just below/above the DO/ENDDO of the parent loop. This is illustrated with the following example.

```

idesc_360: DO idesc=1,MaxIOTypes
ifil_360: DO ifil=1,MaxIOFiles
    SELECT CASE (idesc)
        CASE (io_mppmod,io_modgrd,io_metgrd,io_sstgrd,io_biogrd,&
            & io_nstgrd,io_biospc,io_rlxobc,io_nstspc)
            modfiles(idesc,ifil,:)%nocoords = 0
        CASE (io_1uvsur,io_2uvobc,io_3uvobc,io_salobc,io_tmppobc,&
            & io_bioobc)
            IF (ifil.EQ.1) THEN
                modfiles(idesc,ifil,:)%nocoords = 0
            ELSE
                modfiles(idesc,ifil,:)%nocoords = 1
            ENDIF
        CASE (io_inicon,io_2uvnst,io_3uvnst,io_salnst,io_tmppnst,&
            & io_bionst,io_metsur,io_ssstsur,io_biosur)
            modfiles(idesc,ifil,:)%nocoords = 1
    END SELECT
ENDDO ifil_360
ENDDO idesc_360

```

Example 13.1: indentation of control structures and continuation lines.

- Although the free format allows line lengths upto 132 characters, a length of 80 characters is taken for clarity since this is the maximum length on standard X-windows on UNIX/LINUX machines.

Statements longer than this limit are continued on the next line by appending a ‘&’ character at the end of the line and at the start of the (possibly) intended line, as shown in Example 13.1.

- Comments lines start with a ‘!’ in the first column. Although allowed by the FORTRAN 90 standard, the common practice is, for reason of clarity, not to use comments after the first column (i.e. in the middle of a line).
 - Short statements may be written on one line by inserting a ‘;’ before each next statement.
 - Statements labels always start at the first column.
2. Names of variables, named constants, program units (subroutines, functions, modules) and dummy arguments can contain letters, digits and underscores. The first character must be a letter. The maximum length of a name is given by the program parameter `lenname = 31`, which is also the FORTRAN 90 standard. The underscore character ‘_’ is used in the code for names composed of different words, e.g. `density_equation`, `land_mask`, or for key ids (see Section ??) below.
 3. Although names in FORTRAN 90 are case insensitive, the convention is adopted to write all FORTRAN 90 specific names in uppercase, and all names, defined in the code in lowercase letters. A mixed case is used for some systems parameters (see e.g. `syspars.f90`).
 4. The program uses explicit typing. This means that the type of each variable, parameter or function result needs to declared explicitly in the declaration part of a program unit. This part must be preceded by the line:

`IMPLICIT NONE`

5. The indices of an array defined on the model grid, as defined in Section 15.1.1, are denoted by `i,j,k` for respectively the X-, Y- and Z-direction.
6. In FORTRAN 77 the values of a model grid array can only be accessed element-wise within an assignment statement. The FORTRAN 90 standard allows to use assignments on whole arrays or array sections. The rule is that the expression on the right is either a scalar or an array with the same shape as the one on the left. In the first case the value of the scalar is assigned to all elements in the array on the left. In

the second one, the values of the array expression on the right are assigned element-wise to the corresponding elements of the array variable on the left. The convention, adopted in COHERENS, is to use array assignments where possible, e.g.

```
REAL, DIMENSION(ncloc,nrloc,nz) :: array3dc
...
array3dc = 0.0
```

This procedure is primarily used for the initialisation of variables. However, in most parts of the program distinction has to be made between grid points located on land and sea points. The method consists in maintaining a vertical ("k") loop, whereas the ("i,j") loops in the horizontal direction are replaced by array assignments within a **WHERE** block, as in the following example:

```
k_434: DO k=1,nz
  WHERE (maskatc_int)
    psic_A(1:ncloc,1:nrloc,k) = tridcfc(:,:,k,4)
  END WHERE
ENDDO k_434
```

Example 13.2: array assignment and land masks.

The array **maskatc_int** is **.TRUE.** or **.FALSE.** for grid points located at sea or on land and has the same shape as the array assignment(s) within the **WHERE** block. In this way calculations are restricted to sea points only.

7. Scalar and array variables must be initialised. More precisely, each scalar or array element must have a value assigned to it, before it can appear within an expression on the right of an assignment statement. The standard initialisation value for variables which have no useful default value is 0.0 for **REAL**, 0 for **INTEGER**, **.FALSE.** for **LOGICAL** and " (empty string) for **CHARACTER** variables. In this way a model grid array is always defined at all grid points. Values at sea points are usually re-defined within the program whereas values in land areas remain equal to the initial (zero or **.FALSE.**) value.
8. As explained in Section 17.1, there are four types of program units: main program, external routines, module routines and modules where the main variables used in the program are declared. The following rules are applied in the code:

- Dummy arguments of an external or module routine need to be declared with the **INTENT** attribute (although this is not required by the **FORTRAN 90** standard). The **INTENT** attribute can have the values **IN**, **OUT** and **INOUT**.
- Optional arguments are only allowed in module routine declarations and not in external routines. This avoids the programming of explicit interfaces. Optional arguments are positioned after all non-optional arguments.
- Argument association in a routine call is positional for non-optional arguments, whereas keyword association is used for (eventually) optional arguments.

```
CALL cf90_inquire_variable(iunit,ivar,name=f90_name(ivars),&
                           & dimids=iddims(1:ndims))
```

where **name** and **dimids** are the names of dummy optional arguments.

- Module routines are declared in a sub-program with a **USE** statement of the form given in Section 13.1.5.
- The last lines of a **SUBROUTINE** or **FUNCTION** consist of a **RETURN** statement followed either by an empty line, one or more specification or error code lines, followed by a **END proc_name** statement where **proc_name** is the name of the routine.
- Since all variables (scalars and arrays) are to be initialised, they must be defined with a value before they can be used as an actual argument with the **INTENT(IN)** or **INTENT(INOUT)** attribute. This is of importance, since, for example, passing an undefined value as argument in a read or **MPI** call may produce unexpected side effects.

9. The following **FORTRAN 77** features are not allowed or not recommended in the **COHERENS** programming convention:

- **COMMON** and **INCLUDE**¹ are excluded
- **GOTO** statements are allowed only in exceptional cases (e.g. for error coding).

¹**INCLUDE mpif.h** is the only allowed exception.

13.1.2 Data types

Table 13.1 lists the data types implemented in COHERENS V2.0. The COMPLEX, INTEGER(KIND=8) and (derived) TYPE do not exist or are non-standard in FORTRAN 77. The following comments are to be given:

- For the LOGICAL, INTEGER, REAL, COMPLEX types without a KIND specifier, a default KIND value is taken, as given in the last column. These defaults are standard on most (commercial and free software) compilers. If this is not the case, they can usually be enforced through compiler options.
- The INTEGER (KIND=8) type is not supported by all compilers (such as gfortran). In that case, the KIND value is replaced 4. This poses no problem for most program applications, since the type is only used to designate the number of seconds since a given date (see Section 13.2.1 for a further discussion).
- Each type has a corresponding id, represented by an INTEGER parameter, given in the second column of Table 13.1. The id is e.g. used in the program to identify the type of variable in a routine call. For example, the last argument in the routine call

```
CALL error_alloc('depmeanatc',2,(/ncloc+2,nrloc+2/),real_type)
```

informs the routine `error_alloc` that the variable 'depmeanatc' is of type REAL.

- The KIND value is represented by a named constant, given in column 3, whose value is given in the fourth column of Table 13.1.
- DERIVED TYPES are a new feature of FORTRAN 90 further discussed in Section 13.1.4.
- COMPLEX variables are currently only used for fast Fourier analysis (file `fft_library.f90`).

Type declaration statements have the following general FORTRAN 90 syntax:

```
type [, att,...] :: var_name
```

where

type : the data type of the variable

Table 13.1: Model data types.

FORTRAN type	COHERENS type	KIND parameter	(assumed) size in bytes
CHARACTER	char_type	kndchar	1
LOGICAL	log_type	kndlog	4
INTEGER	int_type	kndint	4
INTEGER (KIND=8)	longint_type	kndilong	4 or 8 (non-standard)
REAL	real_type	kndreal	4
REAL (KIND=8)	long_type	kndllong	8
COMPLEX	complx_type	kndcmplx	8
DERIVED	—	—	—

att : one or more attribute(s) of the variable, which can take the following forms

SAVE: the value of the variable is saved after the routines is exited

INTENT: used to declare dummy arguments only. Takes one of the forms: **INTENT(IN)** if the actual argument is defined in the calling routine and not re-defined in the called routine, **INTENT(OUT)** of the variable is assumed to be undefined in the calling routine and becomes defined within the called routine, **INTENT(INOUT)** if the actual argument is defined in the calling routine and can be re-defined in the called routine. Note that the **INTENT** attribute is always given in declaration statements of dummy arguments (although this is not required by the FORTRAN 90 standard).

DIMENSION: used to define the shape of an array variable. The array shape is added in parentheses.

ALLOCATABLE: to declare an allocatable array (see Section 13.1.3)

PARAMETER: a named constant whose value cannot be changed by the program

var_name name of the variable

It is remarked that

- **CHARACTER** variables have an additional attribute **LEN=*len*** where *len* is the length of the character string.
- Variables, sharing the same attributes are (preferentially) declared on the same line.

- Declaration and definition of DERIVED TYPE variables is discussed in Section 13.1.4. To improve transparency all DERIVED TYPE definitions are made in the common file *datatypes.f90*.

For example

```
CHARACTER (LEN=12) :: ctype
CHARACTER (LEN=lenname), DIMENSION(MaxProgLevels) :: procname
INTEGER, PARAMETER :: lentime = 21
INTEGER, SAVE :: iunit
REAL :: xtemp, ytemp
REAL, DIMENSION(ncloc,nrloc) :: array2dc1
REAL, SAVE, ALLOCATABLE, DIMENSION(:, :) :: array2dc2, array2dc3
```

Example 13.3: examples of type declarations.

13.1.3 Allocatable arrays

Allocatable arrays are arrays declared with a rank but without shape. After been allocated with the **ALLOCATE** statement, allocated arrays can be deallocated with a **DEALLOCATE** statement at any time within the program.

Advantages are:

- An efficient programming of **ALLOCATE** and **DEALLOCATE** statements offers the possibility for significant memory savings and a more efficient use of internal memory. For example, the **COHERENS V1** program, written in **FORTRAN 77**, required that all memory allocations are known at compilation, implying the consumption of unused memory.
- Array bounds can (re)assigned with non-constant values.
- Allocatable arrays can be **SAVED**, even before the “**ALLOCATABLE**” statement is executed (see Example 13.3)
- Contrary to automatic arrays which are usually stored on the machine’s stack memory, allocated arrays use internal memory. This offers a clear advantage on machines with a limited stack size.
- In parallel mode, different bounds can be defined for the same array on different process domains.

Disadvantages are:

- Arrays, that have not been allocated yet, cannot be passed as arguments to a routine call.
- Errors are difficult to debug. For example, it may occur that values are assigned to an array which is allocated with a zero size. This causes a memory fault which is often difficult to detect, since it usually causes a crash at a different location within the program.
- Although this has not been observed from the performance tests performed with COHERENS V2.0, an intensive use of allocation/deallocation may decrease the performance of the program.

A source code example of array allocation is given below.

```
REAL, ALLOCATABLE, DIMENSION(:, :) :: array2d
...
11 = ...; 12 = ...; u2 = ...
ALLOCATE (array2d(11:12,u2),STAT=ierr)
CALL error_alloc('array2d',2,(/12-11+1,u2/),real_type)
...
IF (ALLOCATED(array2d)) DEALLOCATE (array2d)
```

Example 13.4: example of an allocate statement in COHERENS.

The `error_alloc` routine is called by the program after each `ALLOCATE` statement to check whether an allocation error occurred.

Array allocation is applied in the model as follows.

- “Global arrays”, i.e. arrays which are accessible to all program units such as temperature, currents, ..., are (almost) always declared with the `ALLOCATE` and `SAVE` attribute². Note that in case of a parallel mode, the shapes of model grid arrays depend on the size of the sub-domain.
- Local arrays are only accessible to a program unit (`SUBROUTINE` or `FUNCTION`). They are declared either as allocatable or as automatic arrays depending on whether the CPP option `-DMPI` is set (see Section 3.2). Consider the following example

²For technical reasons a few arrays are declared with constant array dimensions.

```

SUBROUTINE ....
!---declare
#ifdef ALLOC
    REAL, SAVE, ALLOCATABLE, DIMENSION(:,:,:,:) :: source
#else
    REAL, DIMENSION(ncloc,nrloc,nz) :: source
#endif /*ALLOC*/
...
!---allocate
#ifdef ALLOC
    ALLOCATE (source(ncloc,nrloc,nz),STAT=errstat)
    CALL error_alloc('source',3,(/ncloc,nrloc,nz/),real_type)
#endif /*ALLOC*/
.....
!---deallocate
#ifdef ALLOC
    DEALLOCATE (source)
#endif /*ALLOC*/
END SUBROUTINE

```

Example 13.5: allocation/deallocation of local arrays.

If the ALLOC option is set, the local array `source` is declared as ALLOCATABLE, allocated at the beginning of the routine and deallocated before exiting the routine. Otherwise, it is declared as an automatic array. The first case has to be taken if there is no sufficient memory available, the second if the allocation/deallocation procedures have a negative impact on CPU time. The choice is obviously machine-dependent.

13.1.4 Derived types

DERIVED TYPE variables can be considered as aggregated structures composed of “atomic” FORTRAN data types (INTEGER, REAL, LOGICAL, CHARACTER). The aim, in COHERENS, is to store all possible information about a specific item (e.g. a file or variable) in a structured format.

Before a DERIVED TYPE can be declared, its TYPE needs to be defined. The example below, taken from the source code, shows the definition of a derived type variable for storing the attributes of a program variable.

```

TYPE :: VariableAttrs
    CHARACTER (LEN=lenname) :: f90_name

```

```

CHARACTER (LEN=lendesc) :: long_name, vector_name
CHARACTER (LEN=lenunit) :: units
CHARACTER (LEN=lennode) :: node
INTEGER :: data_type, ivarid, nrank
INTEGER, DIMENSION(4) :: shape
END TYPE VariableAtts

```

Example 13.6: `TYPE` definition for storing variable attributes.

A variable of this type can be defined as

```
TYPE (VariableAtts) :: varatts
```

The components of `varatt` are accessed using the FORTRAN 90 syntax

- `varatts%f90_name`: a character string with the FORTRAN name of the variable
- `varatts%long_name`: a string of length `lendesc` describing the variable
- `varatts%data_type`: the data type of the variable as given in the second column of Table 13.1
- `varatts%nrank`: rank of the variable (0 for a scalar, 1 for a vector, ...)
- ...

The DERIVED `TYPE` `FileParams` is used to store all attributes of a file

```

TYPE :: FileParams
  LOGICAL :: defined, info, opened, time_regular
  CHARACTER (LEN=1) :: form, status
  CHARACTER (LEN=leniofile) :: filename, pathname
  CHARACTER (LEN=lendesc) :: filedesc
  INTEGER :: endfile, header_type, iostat, iunit, lenrec, maxrecs, &
             & nocords, nodim, novars, timeid, timerec, tskips, &
             & varid, zetaid
  INTEGER, DIMENSION(3) :: tlims
END TYPE FileParams

```

Example 13.7: `TYPE` definition for storing file attributes.

A useful property of DERIVED TYPEs is that they can be declared with the same attributes as any other data type. This means that they can be declared as scalars, arrays with a given shape, allocatable arrays with a given rank and with the `SAVE` or `INTENT` attribute. The following example illustrates how the attributes of the variables within a certain data file are defined first and then written to the file.

```

!---declare
INTEGER :: numvars
TYPE (FileParams), DIMENSION(MaxIOTypes,MaxIOFiles,2) :: modfiles
TYPE (FileParams) :: filepars
TYPE (VariableAtts), ALLOCATABLE, DIMENSION(:) :: varatts

!---define file attributes
CALL set_modfiles_atts(io_modgrd,1,2)
filepars = modfiles(io_modgrd,1,2)
numvars = filepars%novars

!---open file
CALL open_filepars(filepars)

!---define variable attributes
ALLOCATE (varatts(numvars),STAT=errstat)
CALL error_alloc_struct('varatts',1,(/numvars/), 'VariableAtts')
CALL set_modvars_atts(io_modgrd,1,2,varatts,numvars)

!---write variable attributes
CALL write_atts_mod(filepars,varatts,numvars)
...
!---deallocate
DEALLOCATE (varatts)

```

Example 13.8: defining and writing variable attributes to a data file.

File formats will be discussed in Chapter 14.

The next example describes how the relative coordinates of a 2-D external grid are obtained and stored. These type of coordinates are used to perform interpolation of model data to each point of the data grid and are further discussed in Sections 15.3 and 15.4.2. The following TYPEs are defined

```

!---attributes of surface grids
TYPE :: GridParams

```

```

    INTEGER :: nhtype, n1dat, n2dat
    REAL :: delxdat, delydat, x0dat, y0dat
END TYPE GridParams
!---horizontal relative coordinates
TYPE :: HRelativeCoords
    INTEGER :: icoord, jcoord
    REAL :: xcoord, ycoord
END TYPE HRelativeCoords

```

Example 13.9: DERIVED TYPE definitions for interpolation to surface grids.

Assume that the external grid is rectangular with uniform grid spacings in either direction. The grid can then be completely defined using the attributes stored in a variable of type `GridParams`. In particular the attributes `n1dat` and `n2dat` are the dimensions of the data grid in the X- and Y-direction. The relative coordinates of each data point are stored in a 2-D array of type `HRelativeCoords` using the following procedure

```

!---declare
TYPE (GridParams) :: surfacegrid
TYPE (HRelativeCoords), SAVE, ALLOCATABLE, DIMENSION(:,:) :: &
    & gridcoords
!---define the external data grid
surfacegrid%n1dat = ...; surfacegrid%n2dat = ...
...
!---allocate
n1dat = surfacegrid%n1dat; n2dat = surfacegrid%n2dat
ALLOCATE (gridcoords(n1dat,n2dat),STAT=errstat)

!---evaluate and store the relative coordinates of the data grid
idat_110: DO idat=1,n1dat
jdat_110: DO jdat=1,n2dat
    gridcoords(idat,jdat)%icoord = ...; gridcoords(idat,jdat)%jcoord = ...
    gridcoords(idat,jdat)%xcoord = ...; gridcoords(idat,jdat)%ycoord = ...
ENDDO idat_110
ENDDO jdat_110

```

Example 13.10: storing the relative coordinates of an external data grid.

13.1.5 Modules

Modules are program units which can be used within a FORTRAN 90 program in two ways. Firstly, variables which need to be accessible in different

program routines can be declared within a module. These types are further denoted as “declaration modules”. In COHERENS V1, all variables with a global scope were stored in a **COMMON** block located in a *.inc* file and are made accessible with a **INCLUDE** statement. In COHERENS V2.0 there are no **COMMON** blocks any more, since the declarations are made within a module and become accessible to a program unit by putting the appropriate **USE** statement.

As an example, the example below shows the declarations given in the module **currents** for all arrays related to currents

```
MODULE currents

IMPLICIT NONE

REAL, ALLOCATABLE, DIMENSION(:, :) :: p2dbcgradatu, udevint, udfvel, udvel, &
                                      & udvel_old, umped, umvel
REAL, ALLOCATABLE, DIMENSION(:, :) :: p2dbcgradatv, vdevint, vdfvel, vdvel, &
                                      & vdvel_old, vmpred, vmvel
REAL, ALLOCATABLE, DIMENSION(:, :, :) :: p3dbcgradatu, ufvel, uvel, uvel_old
REAL, ALLOCATABLE, DIMENSION(:, :, :) :: p3dbcgradatv, vfvel, vvel, vvel_old
REAL, ALLOCATABLE, DIMENSION(:, :, :) :: wvel, wphys

SAVE

END MODULE currents
```

Example 13.11: contents of the **currents** module.

If one or more of these arrays are used in a program unit, a **USE** statement must appear in the declaration part:

```
USE currents
```

Other types of application of the **FORTRAN 90** module concept are the so-called “module routines”. These routines have the same form and purpose as the usual external **SUBROUTINE** and **FUNCTION** subprograms, except that:

- Module routines accept optional arguments, keyword arguments, array valued function results and can be used to construct generic interfaces without the need to program explicit interfaces.
- Contrary to external sub-programs, module routines require that a proper **USE** statement must be given in the calling sub-program.

Module routines are mainly used in **COHERENS** to construct so-called “libraries”, i.e. an ensemble of routines with a general common purpose usually implemented through generic interfaces. They are quasi-independent of the main source code. For example, all specific MPI and netCDF routine calls are located in the files *MPI_comms.F90* and *cf90_routines.F90*. In this way, only one of these files has to be re-programmed when a newer version of the MPI or netCDF is implemented in the future. A list of module routine files is given in Table 17.2.

13.1.6 Generic procedures

Generic procedures group a series of procedures with similar functionality together under a common name. This generalises the **FORTRAN 77** concept of **INTRINSIC** routines. The generic name is defined via an **INTERFACE** statement block. This is illustrated by the following example of the `read_vars` generic routine

```
MODULE inout_routines
  ...
INTERFACE read_vars
  MODULE PROCEDURE read_vars_int_0d, read_vars_int_1d, &
    & read_vars_int_2d, read_vars_int_3d, &
    & read_vars_int_4d, read_vars_real_0d, &
    & read_vars_real_1d, read_vars_real_2d, &
    & read_vars_real_3d, read_vars_real_4d
END INTERFACE
```

Example 13.12: definition of a generic routine through an **INTERCACE** block statement.

The generic routine `read_vars` is called when data have to be read from an external data file in **COHERENS** standard format. The appropriate routine is selected by the program from the list in the **MODULE PROCEDURE** statement, depending on the type and rank of the argument which returns the data to be read. In this way the input data argument can be of type **INTEGER** or **REAL** and represent scalars or arrays of rank 1 to 4. The **FORTRAN** code of the specific routines with the same generic name needs to be located in the same file (*inout_routines.f90*) where the **INTERFACE** statement is made. Note, that although each specific routine must have the same number of non-optional arguments, the number and type of optional arguments may differ.

If a call to a generic or non-generic module routine is made, the **USE** statement must be given in the declaration part of the calling routine.

```
USE module_name, ONLY: routine_name
```

Example 13.13: syntax of a USE statement for module routines.

where *module_name* is the name of the module and *routine_name* the name of the (non)-generic routine. Consider the following example in the file *Grid_Arrays.F90*:

```
SUBROUTINE read_grid
  ...
  USE inout_routines, ONLY: close_filepars, open_filepars, &
                           & read_glbatts_mod, read_varatts_mod, &
                           & read_vars
  ...
  CALL read_vars(gsigcoord,filepars,varid,varatts)
  ...
END SUBROUTINE read_grid
```

Example 13.14: example of a USE statement for module routines.

since *gsigcoord* is a REAL array of rank 3, the actually called routine is *read_vars_real_3d*.

13.1.7 Internal documentation and structured layout of the code

The COHERENS programming conventions include rules for internal documentation and code layout. These rules and some of the conventions discussed in the sections above are illustrated in Example 13.15 which shows the complete code of routine *Zdif_at_C* which calculates the vertical diffusion term in a scalar transport equation. A line number is added for illustrative purposes in the first four columns at each line in the example. This means that the actual code line (in this example only) starts in column 5. Note that the numbers in the discussion below refer to specific lines.

Five parts can be distinguished: a header with the routine declaration and comments, declaration part, initialisation, “main” code lines and finalisation.

header lines 1–19 with the following information:

- name of the routine and a short description
- name(s) of the author(s)
- a more detailed description (if needed) under the **Description** header

- reference to a publication or section of the user manual
- the name(s) of the sub-programs calling the routine
- the name(s) of the routines called in this routines (the routines `log_timer_in` and `log_timer_out` are called in most routines and may be omitted here)

declaration part This consists of the following segments

- lines 20–29: `USE` statements. The `ONLY` attribute is given for module routines only.
- lines 30–32 with the `IMPLICIT NONE` statement
- lines 33–44: declaration of the arguments with the `INTENT` attribute (preceded by the `*Arguments*` comment line)
- lines 45–74: comment lines with a description of the arguments in a three column format (name, type, purpose), the unit of the variable is given at the end of the line
- lines 75–88: declaration of all local variables. Except for a few exceptions in the program, local arrays are declared within a `#ifdef` block, either as allocatable or automatic arrays depending on the status of the `-DALLOC` compiler option.
- lines 89–96: internal documentation of the most meaningful local variables using the same three column format
- lines 97–98: two blank lines to make a clear separation between the header and the program code itself

initialisation

- lines 99–101: write information to the log-file
- lines 102–121: allocate local arrays in case `-DALLOC` is defined. Note that each `ALLOCATE` statement is checked for errors.
- lines 122–161: initialise parameters and arrays

main code

- lines 162–279: calculation of the vertical diffusion term in a scalar transport equation (including boundary conditions)

finalisation

- lines 280–289: deallocation of local arrays if `-DALLOC` is defined.
- line 290: write information to the log-file

- lines 291–292: two blank lines
- line 293: the **RETURN** statement is not required by the **FORTRAN 90** standard, but has been implemented in the **COHERENS** programming convention
- line 294: blank line
- line 295: **END** statement followed by the name of the routine (the latter is not required in **FORTRAN 90** but included in the convention for clarity)

sectioning

The actual code, i.e. excluding the declaration part, is divided into numbered sections, subsections, subsubsections. **DO** loop blocks within one of these sections has a label composed of the name of the iteration counter followed by **_**, followed by the section number. An extra number is attached is there are more than one **DO** loops within a section, subsection, For example the **k**-loop which starts on line 218, is the second one in subsection 3.3 and has the label **k_332**.

```
1  :SUBROUTINE Zdif_at_C(psic,tridcfc,vdifcoefatw,novars,ibcsur,ibcbot,nbcs,&
2  :                               & nbcb,bcsur,bcbot,kbounds)
3  :*****!
4  :!
5  :!*Zdif_at_C* Vertical diffusion term for a quantity at C-nodes
6  :!
7  :! Author - Patrick Luyten
8  :!
9  :! Description -
10 :!
11 :! Reference -
12 :!
13 :! Calling program - transport_at_C_3d, transport_at_C_4d1,
14 :!                      transport_at_C_4d2
15 :!
16 :! Module calls - error_alloc
17 :!
18 :*****!
19 :!
20 :USE depths
21 :USE grid
22 :USE gridpars
23 :USE iopars
24 :USE physpars
25 :USE switches
26 :USE syspars
27 :USE timepars
28 :USE error_routines, ONLY: error_alloc
29 :USE time_routines, ONLY: log_timer_in, log_timer_out
30 :
31 :IMPLICIT NONE
32 :
33 :!
34:!* Arguments
35 :!
36 :INTEGER, INTENT(IN) :: ibcbot, ibcsur, nbcb, nbcs, novars
37 :INTEGER, INTENT(IN), DIMENSION(2) :: kbounds
38 :REAL, INTENT(IN), DIMENSION(1-nhalo:ncloc+nhalo,1-nhalo:nrloc+nhalo,&
39 :                               & nz,novars) :: psic
40 :REAL, INTENT(INOUT), DIMENSION(ncloc,nrloc,nz,4,novars) :: tridcfc
41 :REAL, INTENT(IN), DIMENSION(ncloc,nrloc,nz+1) :: vdifcoefatw
```



```

83 :    REAL, SAVE, ALLOCATABLE, DIMENSION(:,:,:,:) :: array3d, diffflux
84 :#else
85 :    REAL, DIMENSION(ncloc,nrloc) :: array2dc1, array2dc2, array2dc3
86 :    REAL, DIMENSION(ncloc,nrloc,2:nz) :: array3d
87 :    REAL, DIMENSION(ncloc,nrloc,nz+1) :: diffflux
88 :#endif /*ALLOC*/
89 :
90 :!
91 :! Name      Type  Purpose
92 :!-----
93 :!*diffflux*  REAL  Diffusive flux (times factor) at W-nodes      [m^2/psic]
94 :!
95 :!-----
96 :!
97 :!
98 :!
99 :procname(pglev+1) = 'Zdif_at_C'
100:CALL log_timer_in(npcc)
101:
102:!
103:!1. Allocate arrays
104:!-----
105:!
106:!
107:#ifdef ALLOC
108:    ALLOCATE (array2dc1(ncloc,nrloc),STAT=errstat)
109:    CALL error_alloc('array2dc1',2,(/ncloc,nrloc/),real_type)
110:    ALLOCATE (array2dc2(ncloc,nrloc),STAT=errstat)
111:    CALL error_alloc('array2dc2',2,(/ncloc,nrloc/),real_type)
112:    ALLOCATE (array2dc3(ncloc,nrloc),STAT=errstat)
113:    CALL error_alloc('array2dc3',2,(/ncloc,nrloc/),real_type)
114:    ALLOCATE (array3d(ncloc,nrloc,2:nz),STAT=errstat)
115:    CALL error_alloc('array3d',3,(/ncloc,nrloc,nz-1/),real_type)
116:    IF (iopt_vdif_impl.NE.2) THEN
117:        ALLOCATE (diffflux(ncloc,nrloc,nz+1),STAT=errstat)
118:        CALL error_alloc('diffflux',3,(/ncloc,nrloc,nz+1/),real_type)
119:    ENDIF
120:#endif /*ALLOC*/
121:
122:!
123:!2. Initialise

```

```

124:!-----
125:!
126:!---fluxes
127:IF (iopt_vdif_impl.NE.2) THEN
128:  WHERE (maskatc_int)
129:    diffflux(:,:,1) = 0.0
130:    diffflux(:,:,nz+1) = 0.0
131:  END WHERE
132:ENDIF
133:
134:!---work space arrays
135:WHERE (maskatc_int)
136:  array2dc1 = deptotatc(1:ncloc,1:nrloc)**2
137:END WHERE
138:k_201: DO k=2,nz
139:  WHERE (maskatc_int)
140:    array3d(:,:,k) = vdifcoefatw(:,:,k)&
141:                           & /(array2dc1*delsatw(1:ncloc,1:nrloc,k))
142:  END WHERE
143:ENDDO k_201
144:
145:IF (ibcsur.LE.2) THEN
146:  WHERE (maskatc_int)
147:    array2dc2 = deptotatc(1:ncloc,1:nrloc)*delsatc(1:ncloc,1:nrloc,nz)
148:  END WHERE
149:ENDIF
150:
151:IF (ibcbot.LE.2) THEN
152:  WHERE (maskatc_int)
153:    array2dc3 = deptotatc(1:ncloc,1:nrloc)*delsatc(1:ncloc,1:nrloc,1)
154:  END WHERE
155:ENDIF
156:
157:!---time factors
158:xexp = delt3d*(1.0-theta_vdif)
159:ximp = delt3d*theta_vdif
160:theta_vdif1 = 1.0-theta_vdif
161:
162:!
163:!3. Diffusion terms
164:!-----

```

```
165:!
166:
167:ivar_300: DO ivar=1,novars
168:
169:!
170:!3.1 Explicit fluxes
171:!-----
172:!
173:
174:  IF (iopt_vdif_impl.NE.2) THEN
175:    k_310: DO k=2,nz
176:      WHERE (maskatc_int)
177:        diffflux(:,:,:,k) = xexp*array3d(:,:,:,k)*&
178:                      & (psic(1:ncloc,1:nrloc,k,ivar)-&
179:                      & psic(1:ncloc,1:nrloc,k-1,ivar))
180:      END WHERE
181:    ENDDO k_310
182:  ENDIF
183:
184:!
185:!3.2 Explicit terms
186:!-----
187:!
188:
189:  IF (iopt_vdif_impl.NE.2) THEN
190:    k_320: DO k=kbounds(1),kbounds(2)
191:      WHERE (maskatc_int)
192:        tridcfc(:,:,:,k,4,ivar) = tridcfc(:,:,:,k,4,ivar) + &
193:                                  & (diffflux(:,:,:,k+1)-diffflux(:,:,:,k))&
194:                                  & /delsatc(1:ncloc,1:nrloc,k)
195:      END WHERE
196:    ENDDO k_320
197:  ENDIF
198:
199:!
200:!3.3 Implicit terms
201:!-----
202:!
203:
204:  IF (iopt_vdif_impl.NE.0) THEN
205:
```

```

206:!      ---lower flux
207:      kmax = MERGE(nz,nz-1,ibcsur.LE.2)
208:      k_331: DO k=2,kmax
209:          WHERE (maskatc_int)
210:              array2dc1 = ximp*array3d(:,:,:,k)/delsatc(1:ncloc,1:nrloc,k)
211:              tridcfc(:,:,:,k,1,ivar) = tridcfc(:,:,:,k,1,ivar) - array2dc1
212:              tridcfc(:,:,:,k,2,ivar) = tridcfc(:,:,:,k,2,ivar) + array2dc1
213:          END WHERE
214:      ENDDO k_331
215:
216:!      ---upper flux
217:      kmin = MERGE(1,2,ibcbot.LE.2)
218:      k_332: DO k=kmin,nz-1
219:          WHERE (maskatc_int)
220:              array2dc1 = ximp*array3d(:,:,:,k+1)&
221:                          & /delsatc(1:ncloc,1:nrloc,k)
222:              tridcfc(:,:,:,k,2,ivar) = tridcfc(:,:,:,k,2,ivar) + array2dc1
223:              tridcfc(:,:,:,k,3,ivar) = tridcfc(:,:,:,k,3,ivar) - array2dc1
224:          END WHERE
225:      ENDDO k_332
226:
227:      ENDIF
228:
229:!
230:!3.4 Boundary conditions
231:!-----
232:!
233:!3.4.1 Surface
234:!-----
235:!
236:!      ---Neumann (prescribed flux)
237:      IF (ibcsur.EQ.1) THEN
238:          WHERE (maskatc_int)
239:              tridcfc(:,:,:,nz,4,ivar) = tridcfc(:,:,:,nz,4,ivar) + &
240:                                      & delt3d*bcsur(:,:,:,1,ivar)/array2dc2
241:          END WHERE
242:
243:!      ---Neumann (using transfer velocity)
244:      ELSEIF (ibcsur.EQ.2) THEN
245:          WHERE (maskatc_int)
246:              tridcfc(:,:,:,nz,2,ivar) = tridcfc(:,:,:,nz,2,ivar) + &

```

```

247:                               & ximp*bcsur(:,:,2,ivar)/array2dc2
248:     tridcfc(:,:,nz,4,ivar) = tridcfc(:,:,nz,4,ivar) - &
249:                               & delt3d*bcsur(:,:,2,ivar)*&
250:                               & (theta_vdif1*psic(1:ncloc,1:nrloc,nz,ivar)-&
251:                               & bcsur(:,:,1,ivar))/array2dc2
252:     END WHERE
253:   ENDIF
254:
255:!
256:!3.4.2 Bottom
257:!-----
258:!
259:! ---Neumann (prescribed flux)
260:  IF (ibcbot.EQ.1) THEN
261:    WHERE (maskatc_int)
262:      tridcfc(:,:,1,4,ivar) = tridcfc(:,:,1,4,ivar) - &
263:                               & delt3d*bcbot(:,:,1,ivar)/array2dc3
264:    END WHERE
265:
266:! ---Neumann (using transfer velocity)
267:  ELSEIF (ibcbot.EQ.2) THEN
268:    WHERE (maskatc_int)
269:      tridcfc(:,:,1,2,ivar) = tridcfc(:,:,1,2,ivar) + &
270:                               & ximp*bcbot(:,:,2,ivar)/array2dc3
271:      tridcfc(:,:,1,4,ivar) = tridcfc(:,:,1,4,ivar) - &
272:                               & delt3d*bcbot(:,:,2,ivar)*&
273:                               & (theta_vdif1*psic(1:ncloc,1:nrloc,1,ivar)-&
274:                               & bcbot(:,:,1,ivar))/array2dc3
275:    END WHERE
276:  ENDIF
277:
278:ENDDO ivar_300
279:
280:!
281:!4. Deallocate arrays
282:!-----
283:!
284:!
285:#ifdef ALLOC
286:  DEALLOCATE (array2dc1,array2dc2,array2dc3,array3d)
287:  IF (iopt_vdif_impl.NE.2) DEALLOCATE (diffflux)

```

```

288:#endif /*ALLOC*/
289:
290:CALL log_timer_out(npcc,itm_vdif)
291:
292:
293:RETURN
294:
295:END SUBROUTINE Zdif_at_C

```

Example 13.15: example layout and internal documentation of a COHERENS routine.

Declaration modules have a similar layout and internal documentation, except that the code only consists of the **MODULE *module_name*** declaration on the first, type declarations and the **END MODULE *module_name*** statement on the last line. This is illustrated in Example 13.16.

```

MODULE density
!*****
!
! *density* Density arrays
!
! Author - Patrick Luyten
!
! Version - @COHERENSdensity.f90  V2.0
!
! Description -
!
!*****
!
IMPLICIT NONE

REAL, ALLOCATABLE, DIMENSION(:,:,:,:) :: beta_sal, beta_temp, dens, sal, temp

SAVE

!
! Name      Type  Purpose
!-----
!*beta_sal* REAL  Salinity expansion coefficient      [1/PSU]
!*beta_temp*REAL  Temperature expansion coefficient   [1/deg C]

```

```

!*dens*      REAL  Mass density          [kg/m^3]
!*sal*       REAL  Salinity            [PSU]
!*temp*      REAL  Temperature         [deg C]
!
!*****
END MODULE density

```

Example 13.16: example layout and internal documentation of a COHERENS declaration module.

13.2 Specific program features

13.2.1 Date and time formats

Time can be represented in the program in four different formats. The first two are absolute calendar date and times. The next two are relative times with respect to a reference date.

1. A string format in the form of a string of `lentime` (23) characters: ‘yyyy/mm/dd;hh:mm:ss:mmm’ where `yyyy` = year, `mm` = month, `dd` = day in month, `hh` = hour in day, `mm` = minutes, `ss` = seconds, `mmm` = milliseconds.

- Examples

```

CHARACTER (LEN=lentime) :: CDateTime, CEndDateTime, &
                           & CStartDatetime

```

where the first variable represents the current date (updated at each 2-D time step), the next two respectively the end and start date of the simulation, defined by the user, e.g.

```

CStartDatetime = '2009/06/15;05:09:00:000'
CEndDatetime = '2009/07/01;15:45:06:000'

```

- These time formats are part of the model setup and are used to calculate the solar altitude for evaluation of surface solar irradiance and as date/time stamp in all time series input/output.
- Precision is 1 millisecond.
- The separators need to be at the correct positions, their values are unimportant.

2. A vector **INTEGER** format in the form of a vector with 7 elements: year, month, day, hour, minutes, seconds, milliseconds

- Examples

```
INTEGER, DIMENSION(7) :: IDatetTime, IEndDatetTime, &
& IStartDatetTime
```

which have the same meaning as above.

- The format is only used to perform internal date/time calculations.
- Precision is 1 millisecond.

3. A scalar **INTEGER** format

```
INTEGER (KIND=kndilong) :: nosecsrun
```

representing the number of seconds since the start of the simulation.

- Used internally to compare the date/time in a data file with the current one in the simulation.
- Lower precision is 1 second, upper precision \sim 68 years if **longint** = 4 or (practically) unlimited otherwise (**longint** = 8).

4. A scalar **INTEGER** (“index”) format

```
INTEGER :: nt
```

which equals the current time “index” defined as the number of (2-D) time steps since the start of the simulation.

- Used both internally as externally by the user to set output times.
- Precision is the number of seconds within one time step. This variable is defined in single precision.

The following additional time parameters are used

REAL :: delt2d	User defined 2-D time step in seconds. Although defined as a REAL variable, to prevent rounding errors in the calculation of the calendar date and time its precision has been restricted to 1 millisecond for time steps smaller than 1000 seconds and to 1 second otherwise.
----------------	---

REAL :: time_zone	User defined time zone, i.e. difference of local time with respect to GMT in hours. Difference is positive (negative) eastwards (westwards) from Greenwich. Default is 0. The program assumes that the start, end and current date and time within the program are given in local time. Important to note is that the date and time, obtained from external data files, must be given with respect to the same time zone as the one used in the program.
INTEGER :: ic3d	User defined number of 2-D time steps within one 3-D time step.

13.2.2 Data flags

Data flags are commonly used in observational data sets for representing invalid data. They have been introduced in the COHERENS code to represent undefined values. The following variables are defined in the program for the flagging of model variables

REAL :: real_fill	Large negative number used for flagging of REAL model variables.
REAL:: real_min	Large negative number used to determine whether a real variable is flagged. More specifically, the variable X is taken as flagged if $X \leq \text{real_min}$. Obviously, $\text{real_fill} < \text{real_min}$.
INTEGER :: int_fill	Large negative number used for flagging of INTEGER model variables.
LOGICAL :: log_fill	Large negative number used for flagging of LOGICAL model variables.

Data flags are used (e.g.) for the following purposes:

- If a data value in a vertical open boundary profile has been flagged, a zero gradient condition is applied at that particular point, and the data value is no longer considered as an external value.
- Flagged values in a SST forcing file are automatically replaced by the modelled temperature at the highest (near surface) level.
- Flags are used to disable interpolation at external locations.
- Flagging of some user-defined model parameters has been implemented as a practical utility in the program. It informs the program that the

parameter has some specific value unknown by the user, but known to the program.

Although most model grid arrays are not defined on land areas, no flags are, for practical reasons, applied in this case. This behaviour may be changed in future versions where land values can be flagged with the `netCDF _FillValue` attribute.

Chapter 14

Model input and output

14.1 Classification of model files

The files, which can be created by the model fall to four different categories. Exception is the file `defruns` (see Section 18.3).

1. Monitoring files

- log file
 - Writes “tracing” information during the simulation.
 - Informs about progress of the run upto a user-defined level. A zero level means that the log file is not created. Note that this is the default setting.
 - For parallel runs a log file can be created by each process.
 - The utility is useful for debugging the model code or for tracing errors in model setup.
- error file
 - Writes error messages.
 - If an error is detected by an error checking routine, an error message is written. In most cases the program aborts immediately afterwards while in other cases several checks are made before the program aborts.
 - For parallel runs an error file is created by each process.
- warning file
 - Writes “warning” messages about suspicious values of setup parameters or variables.
 - The program will not abort.

- timer report

This is a file with information about the total execution time and the percentages of time spent by different model “compartments” (e.g. advection, 2-D mode, I/O, parallel communications, ...).

2. Central input file (CIF)

- This is a file with a complete list of parameter values used for the setup of the application. This includes all the parameters which can be defined by the user in the routines

`usrdef_init_params`, `usrdef_mod_params`, `usrdef_tsr_params`,
`usrdef_avr_params`, `usrdef_anal_freqs`, `usrdef_anal_params`

For further details see Chapters ?? and [27](#).

- The file is created on request by the user and can be used as input to the program in a subsequent run.
- If the CIF is used for input, the above routines are no longer called by the program.

3. Forcing files

- Those files include data arrays used for model setup, e.g. model grid data, type of open boundary conditions, initial conditions, forcing data (open boundaries and meteorological). The files with open boundary data for nested sub-grids also fall within this category.
- All files are used as input to the model, except the open boundary data files for nested sub-grids which are defined as output files.
- All input files may be given in any user-defined format, but can optionally be converted to a COHERENS standard format. The output files are always in COHERENS standard format. The forcing file can be made “virtual”, if the forcing data are directly defined by the user without using an external data file.

4. User output files

- These are output data files created by the program on request by the user.
- Spatial and temporal resolution and type of output data are selected by the user.
- The files are always in COHERENS standard format.

- Output specifications are defined in the user-defined routines `usrdef_tsr_params` for time series, `usrdef_avr_params` for time averaged and `usrdef_anal_params`, `usrdef_anal_freqs` for harmonic data output.

14.2 Default file names

Each file has a default name which can be reset by the user. A default file name (in FORTRAN string format) is defined as:

`title//'. '//filedesc//filenum//form//pid`

for monitoring files, central input file(s) and forcing files, and

`title//'_//filenum//'. '//freqnum//filedesc//dim//form`

for output files, where `title` is a simulation-specific title, `filedesc` the file descriptor (representative for the type of the data in the file), `filenum` the file number in case there is more than one file with the same descriptor, `form` the format of the file, `pid` the process id number, `dim` the dimension of output data and `freqnum` the frequency number. Values of these sub-strings are discussed below.

14.2.1 title

Possible values are

- `runttitle` Simulation title defined in `defruns`. This is the value used for monitoring files and CIF(s)(Categories 1 and 2).
- `inttitle` User-defined parameter used as prefix name for forcing files (Category 3). Default value is `runttitle`.
- `outtitle` User-defined parameter used as prefix name for user output files (Category 4). Default value is `runttitle`.

14.2.2 pid

Process id number, used only for log and error files in parallel mode, empty otherwise.

14.2.3 form

The file format can take the following values:

- ‘A’ ASCII format. This format is portable and readable but unsuitable for large data sets because of its great comsumption of disk space and its use of sequential storage.
- ‘U’ Unformatted binary format. Advantage is a more efficient use of disk space. Disadvantages are that the file is (mainly) non-portable, not directly readable and uses sequential storage.
- ‘N’ NetCDF format (recommended). This format is portable, directly readable (with the `netCDF ncdump` utility). Data can be accessed directly by specifying the appropriate record number. Only (possible) disadvantage is that the `netCDF` library needs to be compiled first.
- ‘I’ This does not represent a specific data format but indicates that the file is an ASCII information file containing the metadata of a forcing or output data file.

Monitoring and central input files are always in ‘A’-format. Forcing and output files may be in any of the first three formats.

14.2.4 filedesc

The string `filedesc`, further denoted as the “file descriptor”, is a character string taking one of the values below.

1. Monitoring files

`inilog` log file with tracing information during the initialisation phase of the simulation
`runlog` log file with tracing information during the main (time loop) phase of the simulation
`errlog` error file
`warlog` warning file
`timing` timer report file

2. CIF file. Only one file is currently available.

`cifmod` parameters for model setup

3. Forcing files. The string `filedesc` refers to the contents of the file.

mppmod	arrays defining the domain decomposition (parallel mode only)
inicon	initial conditions
fincon	final conditions
modgrd	model grid, bathymetry and location of open boundaries
metgrd	surface meteorological grid
sstgrd	sea surface temperature (SST) grid
wavgrd	surface wave grid
nstgrd	locations of open boundary locations of nested sub-grids. A file number needs to be supplied for each sub-grid.
sedspc	specifiers (i.e. particle attributes) for the sediment module
1uvsur	1-D surface forcing
2uvobc	2-D mode normal open boundary forcing
2xyobc	2-D mode tangential open boundary forcing
3uvobc	3-D mode normal open boundary forcing (baroclinic currents)
3xyobc	3-D mode tangential open boundary forcing (baroclinic currents)
salobc	salinity open boundary forcing
tmpobc	temperature open boundary forcing
sedobc	sediment open boundary forcing
rlxobc	definitions of relaxation zones
nstspc	specifiers for sub-grid nesting
2uvnst	2-D normal open boundary data for the nested sub-grids. A file number is supplied for each sub-grid.
2xynst	2-D tangential open boundary data for the nested sub-grids. A file number is supplied for each sub-grid.
3uvnst	baroclinic current normal open boundary data for the nested sub-grids. A file number is supplied for each sub-grid.
3xynst	baroclinic current tangential open boundary data for the nested sub-grids. A file number is supplied for each sub-grid.
salnst	salinity open boundary data for the nested sub-grids. A file number needs is supplied for each sub-grid.
tmpnst	temperature open boundary data for the nested sub-grids. A file number is supplied for each sub-grid.
sednst	sediment open boundary data for the nested sub-grids. A file number is supplied for each sub-grid.

metsur	meteorological surface data
sstsur	SST surface data
wavsur	surface wave data
drycel	dry cell locations
thndam	thin dam locations
weibar	weirs/barries locations and parameters
disspc	discharge specifiers
disloc	discharge locations
disvol	volume discharges
discur	momentum discharges
dissal	salinity discharge
distmp	temperature discharges

4. Output files.

tsout	time series output
avrgd	time averaged output
resid	output of residuals
amplt	output of harmonic amplitudes
phase	output of harmonic phases
ellip	output of tidal ellipse parameters

14.2.5 filenum

1. If the file descriptor equals **1uvsur**, **2uvobc**, **3uvobc**, **salobc**, **tmpobc** or **sedobc**, the contents of the file depends on the value of **filenum**:
 - **filenum=1**: forcing specifiers (e.g. type of open boundary conditions)
 - **filenum>1**: forcing data itself. This allows to spread the data over several files. For example, one file may contain open sea and another river open boundary data. Each file can have its own temporal resolution.
2. In case the file descriptor equals **2uvnst**, **3uvnst**, **salnst**, **tmpnst** or **sednst**, the file contains nested data and **filenum** equals the number of the associated sub-grid (between 1 and **nonestsets**).

3. In all other cases (including surface forcing data) the file number is not used.

The maximum allowed file number is given by the system parameter `MaxIOFiles`, defined in `syspars.f90`.

14.2.6 freqnum

Frequency number only used for output of amplitudes, phases and elliptic parameters. Empty otherwise.

14.2.7 dim

The file dimension is only used for files of Category 4:

0 : 0-D output data grid

2 : 2-D output data grid

3 : 3-D output data grid

G: file containing the output grid coordinates but not the data themselves

14.3 Formats of monitoring files

14.3.1 Log files

Log files contain the following information:

- Some general information (e.g. current date, number of time steps, value of 2-D CFL limit, ...).
- Each time a file is opened within the program, a message is written with the name, unit number and format of the file. A similar message is printed when a file is closed.
- A line starting with a number followed by ‘:’ and the name of a routine means that the program entered this routine. The number denotes the program routine level. Main program is at level 1, a routine called by the main program has level 2, a routine called by another routine at level n is at level n+1,
- The maximum number of levels traced by the log file, is defined by the user. Note that large log files may be written if this maximum equals

or exceeds a value larger than 5. A value of 3 is recommended for normal runs, a value of 7 if the log file is used for debugging. When the program returns to the previous level (at the end of the called routine), a line is written with the same level number followed by ‘:R’.

- Two separate log files can be written by the program: the *inilog* file which traces information during the initialisation phase and is closed when the program enters the time loop and the *runlog* file which is active during the time loop only. The two program phases are discussed in Section 17.2.
- If the maximum level for tracing is set to zero (default), no log file is written.
- The following parameters can be defined by the user:
 - The tracing level of the *inilog* and *runlog* files. In parallel mode, different values can be selected for different sub-domains. Note that, by default, all levels are set to zero so that no log file is created.
 - The names of the log files. In parallel mode, all *inilog* or *runlog* files have the same standard name (which can be redefined by the user) appended by a suffix with the process id number.
 - A number of time steps can be defined after which the *runlog* file is overwritten. Default is the total number of time steps (i.e. information is written at all time steps and the file is never overwritten).
 - The writing of an exit statement of the form ‘**num:R**’, where ‘**num**’ is the program level in the “log”-file on exit of a routine call if **.TRUE.** (which is also the default).

For details see Section ??.

An example is given below.

```
Open file fredyA.runlogA on unit 1 type OUT (A)
2:update_time
3:add_secs_to_date_int
3:R
3:convert_date_to_char
3:R
3:day_number_int
```

```

3:R
2003/01/01;00:00:30,000
2:R
2:equation_of_state
2:R
2:baroclinic_gradient
3:Zcoord_arr
3:R
3:Carr_at_W: sal
3:R
...
2:R
2:hydrodynamic_equations
3:current_pred
3:R
3:current_2d
umax = 0.1765635E-02 (28,30)
vmax = 0.1765635E-02 (30,28)
3:R
...
Close file on unit 8 (A)
2:R
2:simulation_end
3:rng_finalize
3:R
3:timer_report
Open file fredyA.timingA on unit 4 type OUT (A)
3:R
3:deallocate_mod_arrays
3:R
2:End of simulation: fredyA
Close file fredyA.warlogA on unit 3 (A)
Close file fredyA.errlogA on unit 2 (A)
Close file fredyA.runlogA on unit 1 (A)

```

Example 14.1: Part of the *runlog* file written by running test case ***fredyA***.
Tracer level is 3.

The tracing information in the log files is implemented within the code by defining the name of the routine and calling the routine `log_timer_in` on entry and `log_timer_out` just before exiting the routine, e.g.

```

procname(pglev+1) = 'open_boundary_arrays'
CALL log_timer_in()
...
CALL log_timer_out()

```

The first line is the name of the routine which has been entered and will be written to the log file. Routine `log_timer_in` sets the current program level by increasing the counter `pglev` by 1 and writes the entry message. The call to `log_timer_out` decreases `pglev` by 1. Note that if the first call is programmed in the code, the second one must be inserted as well just before the `RETURN` statement.

14.3.2 Error files

Error checking routines have been implemented in the model code. Errors are always considered as fatal which means that the program aborts. The program exit is usually executed after a series of checks. In this way several error messages can be written to the `errlog` file. The file is created in the beginning of the program. If no errors are found, the file is deleted at the end of the simulation. Error checking is controlled by the following parameters which can be set by the user:

- Level of error checking
 - 0: Error checking is disabled and no file is created. This is the default.
 - 1: Error checking is performed during initialisation only.
 - 2: Error checking is enabled throughout the whole program (initialisation, time loop and finalisation). This level is very useful for the detection of read errors (e.g. end of file conditions) during an input operation, but should be selected only to check the program for the first time, since it may affect the CPU performance.

In parallel mode, different levels can be taken for different sub-domains.

- The name of the error file. In parallel mode, all `errlog` files have the same standard name (which can be redefined by the user) appended by a suffix with the process id number.
- The maximum number of error messages. Default is the system parameter `MaxErrMesgs`. The reason for limiting the number of messages, is to avoid that an unnecessary amount of error messages is written, since the program performs checks on model arrays as well.

Table 14.1: Key ids for error coding and associated error messages.

key_id	message
ierrno_fopen	Not possible to open file
ierrno_fclose	Unable to close file
ierrno_read	Read error
ierrno_write	Write error
ierrno_fend	End of file condition
ierrno_input	Wrong input values
ierrno_inival	Invalid initial values for model parameters or arrays
ierrno_runval	Invalid values for variables at run time
ierrno_alloc	Not possible to allocate arrays
ierrno_arg	Missing or invalid argument in a routine call
ierrno_comms	Communication error
ierrno_MPI	Error in a MPI call
ierrno_CDF	Error in a netCDF call

The format of an *errlog* file is illustrated with the example below. Line numbers are added for illustration purposes only.

```

1:Unable to open non-existing file: rhonegrid.dat
2:A total of 1 errors occurred in open_filepars
3:Error type 1 : Not possible to open file
4:PROGRAM TERMINATED ABNORMALLY

```

Example 14.2: Contents of an *errlog* file.

The first line gives a description of the error. When more than one error is found, a message line is written for each error. The second line gives the name of the routine where the error is found. The third line writes a message code describing the general type of the error(s). Each message code is presented in the program by a key id of the form `ierrno_*`. A list of available key ids is given in Table 14.1. The last line in example 14.2 is standard for all *errlog* files.

The error code is programmed as follows:

- The number of detected errors is given by the parameter `nerrs`. Its initial value is zero.
- A series of routines are implemented for checking setup variables. For example, the model parameters, defined in `usrdef_mod_params`, are checked in `check_mod_params`. These routines

are located in *check_model.f90* and fully described in Section ???. Error checking is also performed when a file is opened or closed, metadata are read from a forcing file, or if a READ or an end-of-file condition occurs.

- Within these “checking” routines, calls are made to one or more routines, defined in *error_routines.F90*. For example, `error_limits_var` to verify whether a switch has allowed values between certain limits. If the routine detects an error, the error message is written and `nerrs` is increased. These routines in *error_routines.f90* are fully described in Section ???.
- The routine `error_abort` is called with a typical error code, in the form of a key id. If `nerrs>0`, lines 2–4 of the previous example are written with an error message which corresponds to the given key id (see Table 14.1), and the program aborts immediately afterwards. In parallel mode, the error message will only be written by the processes where error(s) were detected.

For example

```
CALL error_lbound_var(nc,'nc',0,.FALSE.)
...
CALL error_limits_var(iopt_grid_htype,'iopt_grid_htype',1,3)
...
CALL error_lbound_var_date(CEndDateTime,'CEndDateTime',CStartTime,&
                           & .FALSE.)
...
CALL error_limits_var(dlat_ref,'dlat_ref',-90.0,90.0)
...
CALL error_abort('check_mod_params',ierrno_inival)
```

Example 14.3: Excerpts of routine `check_mod_params` illustrating the use of error coding.

The first four calls are error checking routines. The first tests whether `nc` is positive, the second whether the switch `iopt_grid_htype` has a value between 1 and 3, the third whether the end date is later than the start date, the fourth whether the reference latitude is between -90° and 90° . If a test turns `.FALSE.`, an error message is written. The routine `error_abort` is called with the key id `ierrno_inival`, representative for invalid setup parameters.

14.3.3 Warning file

Besides error messages which are always fatal, the program may also write warning messages which do not cause termination of the program. The utility can be useful for debugging. The aim is to provide information for the user about suspicious selection of model parameters (usually switches) or arrays or to inform the user that certain setup parameters (defined by the user or default) have been reset. For example, if a 2-D simulation is selected with `iopt_grid_nodim=2` and the vertical resolution parameter `nz` is set to a value larger than 1, a warning message is issued that this parameter is reset to 1. The following control parameters can be (re)set by the user:

- The warning utility is switched on by default, but can be disabled by the user.
- The name of the warning file. In parallel mode, only one file is permitted, written by the master process.

```
WARNING: value of integer parameter iopt_mode_2D is set from 1 to 0
WARNING: value of integer parameter iopt_dens_grad is set from 1 to 0
WARNING: value of integer parameter iopt_bstres_drag is set from 3 to 0
WARNING: value of integer parameter nprocsx is set from 0 to 1
WARNING: value of integer parameter nprocsy is set from 0 to 1
```

Example 14.4: Contents of the warlog file produced by running test case `pycnoA`.

14.3.4 Timer report file

A timer report is a file which contains information about the total execution time and the percentages of time spent by different model “compartments”. Each compartment has an associated time key id, listed in Table 14.2. The following control parameters can be defined by the user:

- The type of information contained in the report.
 - 0: No timer report is written. This is the default.
 - 1: Only the total execution time is written.
 - 2: Time information (in percentage of total time) is written for all “timers”. In case of a parallel simulation, the percentages are given for the process with the largest amount of time, the lowest amount of time, as an average over all processes and for the master process.

Table 14.2: Timer key ids and their meaning.

key_id	description
itm_hydro	hydrodynamics
itm_1dmode	water column mode calculations
itm_2dmode	2-D mode calculations
itm_3dmode	3-D mode calculations
itm_dens	total of density (including temperature and salinity) calculations
itm_temp	temperature
itm_sal	salinity
itm_init	initialisation procedures
itm_trans	transport routines
itm_adv	advection routines
itm_hdif	horizontal diffusion
itm_vdif	vertical diffusion (including turbulence modules)
itm_phgrad	baroclinic pressure gradient
itm_input	input operations
itm_output	output operations
itm_inout	total of input and output operations
itm_com_coll	collect communication calls
itm_com_comb	combine communication calls
itm_com_copy	copy communication calls
itm_com_dist	distribute communication calls
itm_com_exch	exchange communication calls
itm_com_util	utility communication calls
itm_coms	total of parallel communications
itm_MPI	total of MPI calls
itm_CDF	netCDF calls
itm_arrint	interpolation of model grid arrays
itm_user	usrdef routine calls
itm_nest	nesting procedures
itm_libs	internal library routine calls
itm_astro	astronomical tide
itm_bconds	boundary conditions
itm_meteo	meteorological routines
itm_structs	structures and discharges
itm_wait	wait calls
itm_sed	sediment model
itm_bio	biological model

- 3: The same as the previous case, but the time percentages are now given for each individual process in addition. In the serial case, behaviour is as for case 2.
- The name of the timer report file. In parallel mode, only one file is permitted, written by the master process.
 - The unit of time for writing the total execution can be written in seconds, minutes, hours or days. Default is seconds.

The utility is useful for testing the CPU efficiency of a parallel decomposition. An example of such test is given in Figure 16.1. Two examples are given below using the same test case *plumeC*. The first shows the contents of the timer report obtained from a serial run:

```
plume1C: 392s.822
Hydrodynamics      : 39.152
2D mode            : 12.125
3D mode            : 27.939
Density             : 26.150
Salinity            : 22.338
Initialisation     : 0.025
Transport           : 50.284
Advection           : 25.721
Horizontal diffusion: 13.541
Vertical diffusion  : 18.450
Baroclinic pressure : 3.118
Input                : 0.001
Output               : 1.762
Input/output          : 1.763
Array interpolation : 12.917
User calls           : 14.979
Library calls         : 4.284
Boundary conditions : 0.660
```

Example 14.5: Timer report for test case *plumeC* on a serial machine.

The second example is for the same test case now obtained on a parallel machine with four processors and full timer information. For each timer process there are now two lines. The first one gives statistical information (maximum, minimum, mean and master). The second gives the times for each individual processor. The report now contains additional information

about parallel communication calls, given by the compartments ‘Combine comms’, ..., ‘MPI calls’. Note that the numbers given in ‘Parallel comms’ are the sum of the corresponding ones for the combine, copy, exchange and utility operations. This information is not given in the serial case since the times related to parallel communications are, obviously zero, and zero times are not printed in the table.

```

plume1C: 169s.783
Hydrodynamics      : 44.064 44.009 44.037 44.044
                     44.044 44.064 44.009 44.039
2D mode            : 18.971 18.889 18.939 19.172
                     19.172 18.958 18.889 18.971
3D mode            : 25.648 25.584 25.620 25.568
                     25.568 25.648 25.584 25.627
Density            : 27.339 27.210 27.285 27.176
                     27.176 27.339 27.210 27.306
Salinity           : 23.711 23.659 23.677 23.677
                     23.677 23.711 23.659 23.660
Initialisation     : 0.094 0.088 0.092 0.088
                     0.088 0.088 0.094 0.094
Transport          : 45.924 45.189 45.574 45.046
                     45.046 45.189 45.610 45.924
Advection          : 28.831 28.505 28.639 28.648
                     28.648 28.505 28.582 28.831
Horizontal diffusion: 12.218 11.796 12.077 12.074
                     12.074 11.796 12.218 12.216
Vertical diffusion : 10.446 10.359 10.412 10.189
                     10.189 10.359 10.446 10.431
Baroclinic pressure : 2.963 2.891 2.917 2.845
                     2.845 2.898 2.891 2.963
Input               : 0.024 0.024 0.024 0.071
                     0.071 0.024 0.024 0.024
Output              : 0.000 0.000 0.000 2.951
                     2.951 0.000 0.000 0.000
Input/output        : 0.024 0.024 0.024 3.022
                     3.022 0.024 0.024 0.024
Combine comms      : 3.351 3.239 3.290 0.353
                     0.353 3.239 3.280 3.351
Copy comms         : 0.035 0.029 0.031 0.000
                     0.000 0.035 0.029 0.029
Exchange comms     : 12.032 10.301 10.995 11.444

```

```

11.444 12.032 10.652 10.301
Utility comms      : 0.524  0.424  0.465  0.530
                  0.530  0.424  0.448  0.524
Parallel comms    : 15.731 14.206 14.782 12.328
                  12.328 15.731 14.409 14.206
MPI calls         : 12.998 11.485 12.016  9.271
                  9.271 12.998 11.565 11.485
Array interpolation : 12.404 11.920 12.165 11.939
                  11.939 11.920 12.171 12.404
User calls        : 9.842  9.694  9.776 10.525
                  10.525 9.694  9.792  9.842
Library calls     : 2.544  2.385  2.463  2.645
                  2.645  2.544  2.461  2.385
Boundary conditions : 0.536  0.477  0.514  0.459
                  0.459  0.477  0.530  0.536

```

Example 14.6: Timer report for test case *plumeC* on a parallel machine with four processors.

The following example shows how timing is implemented in the program

```

CALL log_timer_in(npcc)
...
CALL log_timer_out(npcc,itm_adv)

```

A timer vector array is created at the start of the program and initialised to zero. The routine `log_timer_in` is called in the beginning of the routine and stores the current clock count value of the processor clock in the optional argument `npcc`. The value is obtained by calling the FORTRAN 90 intrinsic routine `SYSTEM_CLOCK`. The call to `log_timer_out` is made at the last line of the subprogram. The routine calls `SYSTEM_CLOCK` again and subtracts the new clock count from the previous one. This gives the time spent in the routine, measured in clock counts. The result is added to the value stored in the corresponding element of the “timer” vector array. The array index is given by the timer key id `itm_adv`. At the end of the program the array values are converted to seconds, divided by the total execution time and multiplied by 100. This gives the computation times associated with different timer key ids in percentage.

14.4 Central input file

14.4.1 Syntax of a CIF

As shown in the example 14.7 below, each data line in the CIF has the following syntax

```
varname = value 1, value 2, ..., value n
```

where **varname** is the FORTRAN name of a model parameter and **value 1** to **value n** are the input values of the parameter, separated by the data separator ','. The file is read line-wise. The data strings **value 1**, **value 2**, ... are converted to the appropriate (numeric, logical, character) data format associated with the variable FORTRAN variable **varname**. The following rules apply

- If a comment character ‘!’ appears in the string, all characters in the string, starting from this character are ignored. However, the comment character can only appear at the first position of the data line (in which case the entire line is ignored) or after the last character of the last data string.
- If **varname** corresponds to a scalar, it is obvious that only one value needs to be given and there is no data separator. In case of a vector, the number of data can be lower than the size of the vector in which case the non-defined values are set to their defaults. However if a vector has a specified “physical” size, all expected data must be given. Examples are the arrays **index_abc** (physical size given by **nconabc**) or **ntrestart** (physical size given by **nrestarts**).
- If the model parameter represents a multi-dimensional array (of rank **m**), the first **m-1** data strings represent the vector index for the first **m-1** dimensions, the subsequent the values for each array index of the last dimension. As before, the number of values does not need to be equal to the size of the last dimension, unless a “physical” size is expected.
- If the variable is a derived type scalar variable, the data strings represent the components in the order given by the **TYPE** definition in **datatype.f90**. Derived type arrays are initialised element-wise, i.e. a separate line for each array element. The first data string(s) are the array indices of the first, ..., last array dimension.
- The first array index for the variable **modfiles** (see Section 19.4) is not given by a numeric value but by its file descriptor in string format, e.g.

the string `modgrid` corresponds to the key id `io_modgrd` whose numeric value is set by the program to 3.

- If a data string contains only blanks or equals the null string, the value of the corresponding model parameter is undefined, in which case its default value is retained. When the CIF is written by the program, all variables (even defaults) are defined in the data strings.
- No error occurs if a model scalar or array parameter does not appear on any input line in which case the default value is retained.
- The characters in the string `varname` are case insensitive. If the CIF is written by the program, the names are always given in upper case characters.
- When a CIF is written by the program, all setup parameters are included in the file. The values are either the default settings or the re-defined values from a call to the appropriate `usrdef_` routine or the ones reset by the program after a call to a `reset_` routine. Only exception to this rule is the parameter `cold_start` which is always written as `.FALSE.` and can only be changed by editing the CIF manually.

14.4.2 CIF blocks

A CIF file is composed of six blocks which must be given in a specific order. Each block corresponds to a `usrdef_` routine (given in parentheses below) where the parameters could be defined in absence of the CIF.

- 1: monitoring parameters (`usrdef_init_params`)
- 2: general model setup parameters (`usrdef_mod_params`)
- 3: parameters for the setup of time series output (`usrdef_out_params`)
- 4: parameters for the setup of time averaged output (`usrdef_avr_params`)
- 5: definitions for making harmonic analyses (`usrdef_anal_freqs`)
- 6: parameters for harmonic output (`usrdef_anal_params`)

The following rules apply for CIF blocks

- A CIF block is terminated by a line whose first character is the block separator '#' (the rest of the line is ignored).
- A block may be empty but the separator lines must always be there. This means that the file must contain 6 lines (including the last one) starting with a '#'. An empty block is represented by two consecutive separator lines.

- Empty blocks are written by the program in the following cases
 - block 3: no time series output (`iopt_out_tsers=0`)
 - block 4: no time averaged output (`iop_out_avrgd=0`)
 - blocks 5 and 6: no harmonic output (`iop_out_anal=0`)
- On the other hand, the above blocks may be non-empty even when the appropriate switch is zero. In that case the input lines are read by the program, but no assignment is made.

CIF special characters

The CIF utility uses the following special characters

- ‘,’ separates the data strings on an input line
- ‘=’ separates the string `varname` from the data strings. Must be on all input lines except those starting with a ‘!’ or ‘#’ character
- ‘!’ indicates the start of a comment. All characters on the input line at and beyond this character are ignored.
- ‘#’ block separator. Must always be the first character on a separator line.

These special characters cannot be used in the string `varname` or in a data string representing a string variable. For this reason the ‘,’ character, used in previous versions as separator between seconds and milliseconds in a date/-time string is now replaced by a ‘:’.

14.4.3 Order of definitions

Each scalar or array parameter must be defined within its specific block. However, the order of definition within a block is, in principle, irrelevant. However, if the number of data on an input line depends on a “physical size” dimension parameter defined by another model parameter, this size parameter must appear on a previous data line.

```
COLD_START = F
LEVPROCS_INI = 3
LEVPROCS_RUN = 3
INILOG_FILE = plume1A.inilogA
RUNLOG_FILE = plume1A.runlogA
RUNLOG_COUNT = 8640
MAX_ERRORS = 50
```

```
LEVPROCS_ERR = 1
ERRLOG_FILE = plume1A.errlogA
WARNING = T
WARLOG_FILE = plume1A.warlogA
LEVTIMER = 3
TIMING_FILE = plume1A.timingA
TIMER_FORMAT = 1
#
IOPT_ADV_SCAL = 3
IOPT_ADV_TURB = 0
IOPT_ADV_TVD = 1
IOPT_ADV_2D = 3
IOPT_ADV_3D = 3
IOPT_ARRINT_HREG = 0
IOPT_ARRINT_VREG = 0
IOPT_ASTRO_ANAL = 0
IOPT_ASTRO_PARS = 0
IOPT_ASTRO_TIDE = 0
...
NC = 121
NR = 41
NZ = 20
NOSBU = 80
NOSBV = 120
NRVBU = 0
NRVBV = 1
NONESTSETS = 0
NORLXZONES = 0
NPROCS = 1
NPROCSX = 1
NPROCSY = 1
IDMASTER = 0
CSTARTDATETIME = 2003/01/03;00:00:00:000
CENDDATETIME = 2003/01/06;00:00:00:000
DELT2D = 30.
IC3D = 10
ICNODAL = 0
TIME_ZONE = 0.
NTOBCRLX = 0
ATMPRES_REF = 101325.
BDRAGCOEF_CST = 0.
```

```

BDRAGLIN = 0.

...
NCONOBC = 1
INDEX_OBC = 46
NCONASTRO = 0
ALPHA_BLACK = 0.2
ALPHA_MA = 10.
ALPHA_PP = 5.
BETA_MA = 3.33
BETA_XING = 2.

...
NORESTARTS = 1
NTRESTART = 8640
INTITLE = plume1A
OUTTITLE = plumeA
MAXWAITSECS = 3600
NOWAITSECS = 0
NRECUNIT = 4
NOSETSTSR = 4
NOSTATSTSR = 0
NOVARSTSR = 9
NOSETSavr = 0
NOSTATSAVR = 0
NOVARSAVR = 0
NOSETSANAL = 1
NOFREQSANAL = 1
NOSTATSANAL = 0
NOVARSANAL = 7
MODFILES = inicon,1,1,U,R,plumeA.phsicsU,0,0,0,0,F,F,
MODFILES = modgrd,1,1,A,R,plumeA.modgrdA,0,0,0,0,F,F,
MODFILES = 2uvobc,1,1,U,R,plume1A.2uvobc1U,0,0,0,0,F,F,
MODFILES = 3uvobc,1,1,A,R,plume1A.3uvobc1A,0,0,0,0,F,F,
MODFILES = salobc,1,1,A,R,plume1A.salobc1A,0,0,0,0,F,F,
MODFILES = 2uvobc,2,1,U,R,plume1A.2uvobc2U,0,0,1,0,F,F,
MODFILES = 3uvobc,2,1,A,R,plume1A.3uvobc2A,0,0,1,0,F,F,
MODFILES = salobc,2,1,A,R,plume1A.salobc2A,0,0,1,0,F,F,
SURFACEGRIDS = 1,1,0,0,1000.,1000.,0.,0.

#
TSRVARS = 1,0,0,0,0,0.,C,width,Plume width,km,
TSRVARS = 2,0,0,0,0,0.,C,hfront,Plume length,km,
TSRVARS = 3,92,2,0,0,0.,C,umvel,X-component of depth-mean current,m/s,

```

```

        Depth-mean current
TSRVARS = 4,101,2,0,0,0.,C,vmvel,Y-component of depth-mean current,m/s,
        Depth-mean current
TSRVARS = 5,81,2,0,0,0.,C,zeta,Surface elevation,m,
TSRVARS = 6,93,3,0,0,0.,C,uvel,X-component of current,m/s,Current
TSRVARS = 7,102,3,0,0,0.,C,vvel,Y-component of current,m/s,Current
TSRVARS = 8,106,3,0,0,0.,C,wphys,Physical vertical velocity,m/s,
        Physical current
TSRVARS = 9,111,3,0,0,0.,C,sal,Salinity,PSU,
IVARSTSR = 1,6,7,8,9
IVARSTSR = 2,6,7,8,9
IVARSTSR = 3,6,7,8,9
IVARSTSR = 4,1,2,3,4,5
TSR3D = 1,T,U,plumeA_1.tsout3U,T,,2
TSR3D = 2,T,U,plumeA_2.tsout3U,T,,2
TSR3D = 3,T,U,plumeA_3.tsout3U,T,,2
TSR0D = 4,T,A,plumeA_4.tsout0A,T,,2
TSR2D = 4,T,U,plumeA_4.tsout2U,T,,2
TSRGPARS = 1,T,F,F,F,2003/01/03;00:00:00:000,3,0,0,1,120,1,1,40,1,20,20,1,0,
           8640,360
TSRGPARS = 2,T,F,F,F,2003/01/03;00:00:00:000,3,0,0,30,30,1,1,40,1,1,20,1,0,
           8640,360
TSRGPARS = 3,T,F,F,F,2003/01/03;00:00:00:000,3,0,0,1,120,1,5,5,1,1,20,1,0,
           8640,360
TSRGPARS = 4,T,F,F,F,2003/01/03;00:00:00:000,2,0,0,30,30,1,1,1,1,1,1,1,0,
           8640,12
#
#
INDEX_ANAL = 46
NOFREQSHARM = 1
IFREQSHARM = 1,1
#
ANALVARS = 1,92,2,0,0,0.,C,umvel,X-component of depth-mean,current,m/s,
        Depth-mean current
ANALVARS = 2,101,2,0,0,0.,C,vmvel,Y-component of depth-mean,current,m/s,
        Depth-mean current
ANALVARS = 3,81,2,0,0,0.,C,zeta,Surface elevation,m,
ANALVARS = 4,93,3,0,0,0.,C,uvel,X-component of current,m/s,Current
ANALVARS = 5,102,3,0,0,0.,C,vvel,Y-component of current,m/s,Current
ANALVARS = 6,106,3,0,0,0.,C,wphys,Physical vertical velocity,m/s,
        Physical current

```

```

ANALVARS = 7,111,3,0,0,0.,C,sal,Salinity,PSU,
IVARSANAL = 1,1,2,3,4,5,6,7
IVARSELL = 1,1,10
IVECELL2D = 1,1,2
IVECELL3D = 1,1,2
RES2D = 1,T,A,plumeA_1.resid2A,T,,2
RES3D = 1,T,A,plumeA_1.resid3A,T,,2
AMP2D = 1,1,T,A,plumeA_1.1amplt2A,T,,2
PHA2D = 1,1,T,A,plumeA_1.1phase2A,T,,2
ELL2D = 1,1,T,A,plumeA_1.1ellip2A,T,,2
ELL3D = 1,1,T,A,plumeA_1.1ellip3A,T,,2
ANALGPARS = 1,T,F,F,F,2003/01/03:06:00:00:000,3,0,0,1,120,1,1,40,1,1,20,1,0,
8640,1440
#

```

Example 14.7: (Parts) of the CIF produced for test case *plumeA*.

14.5 Forcing files

14.5.1 General aspects

The forcing files, used in the program code, can be divided into two categories:

1. Files, which have no time dependence, are called “initialisation” files and may contain the following information:
 - definitions of a domain decomposition
 - model grid and bathymetry
 - definitions of 2-D external grids
 - locations of the open boundary points of a nested sub-grid
 - specifiers for 2-D and 3-D open boundary conditions or 1-D water column forcing
 - specifiers for nesting
 - definition of relaxation zones.
2. Time series files provide forcing data at one or more specific times, given as a sequence of time record(s).

- initial conditions¹
- open boundary data
- 1-D water column forcing data
- 2-D and 3-D open boundary data written for nested sub-grids
- data defined on a 2-D external (meteorological, SST) grid.

The standard structure of forcing files is composed of

- a metadata (header) section with global information about the file (called “global attributes” in **netCDF** language) and information about the data variables within the file (“variable attributes” in **netCDF** language)
- a data section with the values of the data. The time coordinate (if present) is considered as an additional data variable.

The general structure of the file is then²

```
[Names and values of dimensions]
Global attributes
...
[Attributes of the time coordinate]
Attributes of variable 1
...
Attributes of variable 2
...
Attributes of variable n
[First data time]
Values of variables 1
...
Values of variables n
[Second data time
...]
```

Example 14.8: General layout of a forcing file.

The time coordinate and data time(s) need, obviously, only to be present in case of time series files. Note that the data times must be stored in

¹Initial conditions can be given at one or more specific times.

²The lines in [] are not always present. For example, metadata and data for the time coordinate variable are only needed in case of a forcing file containing time series.

chronological order, but may be given at non-regular time intervals. The detailed formats of forcing files are discussed in the subsections below.

As mentioned in Section 13.1.4, the properties (or attributes) of data files are stored into the derived type variable `FileParams` (see Example 13.7 for a full list of file attributes). The attributes of the forcing files are stored into the 3-D derived type array `modfiles`:

```
TYPE (FileParams), DIMENSION(MaxIOTypes,MaxIOFiles,2) :: modfiles
```

where

`MaxIOTypes` is the maximum number of file descriptors

`MaxIOFiles` is the maximum allowed number of files per file descriptor.

An element of the array `modfiles` can be generically written as
`modfiles(iddesc,ifil,iotype)` where

`iddesc` denotes the file descriptor key id. The program name of the key id has the form `io_filedesc` where `filedesc` is one of the descriptor strings given in Section 14.2.4. For example, `io_2uvobc` is the key id for all forcing files related to 2-D open boundary conditions.

`ifil` is the file number. In some cases, this number is always 1. For example, the input data for defining the model grid are stored in one file with key id `io_modgrd`. On the other hand, if a main grid contains several nested sub-grids, a data file has to be written for each requested parameter and each sub-grid. The parameter is represented by its key id, while the file number denotes the number of the sub-grid. A similar approach is followed for open boundary data. For example, the temperature profiles at open boundaries can be obtained from several data files having the same key id `io_tmppobc`.

`iotype` equals 1 for an input and 2 for an output file. All forcing files used by the model are input files, except for the files written for sub-grid nesting. However, the program provides the possibility to re-write the data obtained from an input file to a separate output file in COHERENS format. This is further discussed in Section 19.4.

In the current implementation, the global attributes of the files are defined by components of the derived type variable `modfiles` and are given below. A (*) at the end of the description means that the attribute can be defined by the user.

status* Status of the file

- ‘0’ : The file is not activated.
- ‘N’ : The file is activated but its contents are defined by the user in a `usrdef_` routine. The user needs to decide whether the data are obtained from some external file or that the file only exists virtually and the data are defined without making a file connection. The option is only available for input files.
- ‘R’ : The file is activated and its contents are read from a data file in COHERENS standard format. The option is only allowed for input data (`ioype`=1).
- ‘W’ : The data are written in COHERENS standard format. This is always the case for nesting data. The option can be used to re-write data, previously obtained in a user format. The file can then be used in a subsequent simulation with the ‘R’ status. Since the file is created for output, the `ioype` index must be 2.

<code>form</code> *	Format of the data file.
‘A’	ASCII
‘U’	unformatted binary
‘N’	<code>netCDF</code>
<code>filename</code> *	Name of the file. If the file status is ‘R’ or ‘W’ and the name of the file is not defined by the user, a default name is given (see Section 14.2). If the status is ‘N’, the name is either defined by the user or unknown. In the latter case the file name is set to ‘N’ (unknown) which may mean that the data are not obtained from a file.
<code>info</code> *	An info (‘I’) file is produced with the metadata information only.
<code>end_type</code> *	Switch to decide what action needs to be taken when an end of file condition occurs during a read. See Section ?? for details.
	0: The program aborts with an error message
	1: The program continues, no further attempt will be made to read data.
	2: The program continues, a next attempt to read the data will be made after <code>nowaitsecs</code> seconds.
<code>tlims(1:3)</code> *	Start/end/step time indices (i.e. times measured in units of the 2-D time step <code>delt2d</code>). The data will be updated after <code>ABS(tlims(3))</code>

$\times \text{delt2d}$ seconds. If $\text{tlims}(3) > 0$, time interpolation will be performed. If $\text{tlims}(3) < 0$, the data values are set to the one obtained from the latest data record earlier than the current time.

iunit File unit number. This parameter is set internally and cannot be reset by the user.

iostat The I/O status of the file.

-1: An error occurred when the program attempted to open the file.

0 : The file is not open.

1 : The file is open and the file pointer is located at the start or before the end of the file.

2 : The file pointer is located at the end of the file (i.e. an end of file condition will occur on a next read).

3 : An end of file condition did occur.

nocoords Number of coordinate arrays within the file, equal to 0 for an initialisation file and 1 for a time series file.

novars Number of (non-coordinate) data variables within the file.

filedesc A string with a description of the file.

timeid netCDF variable id of the time coordinate.

The variable attributes are stored into a temporary derived type array

```
TYPE (VariableAtts), DIMENSION(nocoords+novars) :: varatts
```

where **nocoords** and **novars** are the global attributes stored in the **modfiles** array. The following attributes are defined:

f90_name A string with the FORTRAN name of the variable as used in the program. Maximum length is set by the system parameter **lenname**.

long_name A string with a description of the variable. Maximum length is set by the system parameter **lendesc**.

vector_name If the variable represents a vector component, a string with a description of the vector. Its value must be the same for all components of the same vector. Maximum length is set by the system parameter **lendesc**.

units	A string describing the variable's unit. The string has a format recognised by UNIDATA's Udunits package (UDUNITS, 1997) which can be considered as an internationally recognised standard. Maximum length is set by the system parameter <code>lenunit</code> .
data_type	The type of variable as given in the second column of Table 13.1 (e.g. <code>real_type</code> for a <code>REAL</code> variable).
ivarid	The variable's key id.
nrank	If the variable is an array, the rank of the array. Otherwise, the variable is a scalar and the rank is zero.
shape	If the variable is an array, a vector with the array size(s) in each dimension. In case of a scalar the shape vector has one element with the value 1.

14.5.2 Data contents of forcing files

Table [14.3](#) provides a general listing of the data contents for each type of forcing file. Note that the exact number of data variables depends on how the model has been set up. A detailed discussion is given in the Part [IV](#).

The integer parameters used in the table for array dimensioning have the following meaning:

nprocs	number of processes in case the program is applied in parallel mode
ncloc	X-dimension of the local grid
nrloc	Y-dimension of the local grid
nc	X-dimension of the global (computational) grid
nr	Y-dimension of the global (computational) grid
nz	number of vertical layers
nhalo	halo size (=2)
nf	number of sediment fractions
nconobc	number of tidal constituents at open boundaries
nconastro	number of astronomical constituents used for the tidal force
nobu	number of open boundary points at U-nodes
nobv	number of open boundary points at V-nodes
nodat	number of data (e.g. discharge locations)
numdis	number of discharge locations

numdry	number of dry cells
numthinu	number of thin dams at U-nodes
numthinv	number of thin dams at V-nodes
numwbaru	number of weirs/barriers at U-nodes
numwbarv	number of weirs/barriers at V-nodes
n1dat	X-dimension of an external 2-D data grid
n2dat	Y-dimension of an external 2-D data grid
nonestsets	number of nested sub-grids
nhdat	number of sub-grid open boundary locations in the horizontal used for nesting of a sub-grid
nzdat	number of vertical levels at sub-grid open boundary locations used for nesting of a sub-grid
nonodes	number of “nodal” grids used for interpolation in nesting procedures (see Section ??)
novars	number of data variables (meteorological, wave, discharge) variables in a data file
numprofs	number of vertical profiles in a 3-D open boundary forcing file
nofiles	number of data files (plus one) for open boundary or 1-D surface forcing
norlxzones	number of zones for application of the relaxation scheme near open boundaries

Table 14.3: Data contents for each type of input forcing file. In the last column ‘R’, ‘I’, ‘C’, ‘D’ denote respectively real, integer, character and derived type data.

key id	file number	variable	shape	type
io_mppmod	1	nc1procs	nprocs	I
		nc2procs	nprocs	I
		nr1procs	nprocs	I
		nr2procs	nprocs	I
io_inicon	1	udvel	(1-nhalo:ncloc+nhalo,1-nhalo:nrloc+nhalo)	R
		vdvel	(1-nhalo:ncloc+nhalo,1-nhalo:nrloc+nhalo)	R
		zeta	(0:ncloc+1,0:nrloc+1)	R
		uvel	(1-nhalo:ncloc+nhalo,1-nhalo:nrloc+nhalo,nz)	R
		vvel	(1-nhalo:ncloc+nhalo,1-nhalo:nrloc+nhalo,nz)	R
		wvel	(0:ncloc,0:nrloc,nz+1)	R

(Continued)

Table 14.3: *Continued*

	temp	(1-nhalo:ncloc+nhalo,1-nhalo:nrloc+nhalo,nz)	R
	sal	(1-nhalo:ncloc+nhalo,1-nhalo:nrloc+nhalo,nz)	R
	tke	(1-nhalo:ncloc+nhalo, 1-nhalo:nrloc+nhalo,nz+1)	R
	zlmix	(1-nhalo:ncloc+nhalo, 1-nhalo:nrloc+nhalo,nz+1)	R
	dissip	(1-nhalo:ncloc+nhalo, 1-nhalo:nrloc+nhalo,nz+1)	R
	bdragcoefatc	(0:ncloc,0:nrloc)	R
	zroughatc	(0:ncloc,0:nrloc)	R
	fnode_obic	nconobc	R
	phase_obic	nconobc	R
	fnode_astro	nconastro	R
	phase_astro	nconastro	R
	wbarelossu	numwbaru	R
	wbarelossv	numwbarv	R
	obcsalatu	(nobu,nz,0:2)	R
	obcsalatv	(nobv,nz,0:2)	R
	obctmpatu	(nobu,nz,0:2)	R
	obctmpatv	(nobv,nz,0:2)	R
	obc2uvatu	(nobu,2)	R
	obc2uvatv	(nobv,2)	R
	obc3uvatu	(nobu,nz,2)	R
	obc3uvatv	(nobv,nz,2)	R
io_inicon 2	cvol	(1-nhalo:ncloc+nhalo,1- nhalo:nrloc+nhalo,nz,nf)	R
	zroughatc_sed	(0:ncloc,0:nrloc)	R
	bed_fraction	(0:ncloc,0:nrloc,nf)	R
	obcsedatu	(nobu,nz,0:2,nf)	R
	obcsedatv	(nobv,nz,0:2,nf)	R
io_modgrd 1	gxcoordglb	(nc,nr)	R
	gycoordglb	(nc,nr)	R
	gscoordglb	(nc-1,nr-1,nz+1)	R
	depmeanglb	(nc-1,nr-1)	R
	jobu	nobu	I
	jobv	nobv	I
	jobv	nobv	I

(Continued)

Table 14.3: *Continued*

io_metgrd	1	xcoord	(n1dat,n2dat)	R
		ycoord	(n1dat,n2dat)	R
	or	surfgridglb	(nc,nr)	D
io_sstgrd	1	xcoord	(n1dat,n2dat)	R
		ycoord	(n1dat,n2dat)	R
	or	surfgridglb	(nc,nr)	D
io_wavgrd	1	xcoord	(n1dat,n2dat)	R
		ycoord	(n1dat,n2dat)	R
	or	surfgridglb	(nc,nr)	D
io_nstgrd	1:nonestsets	xcoord	nhdat	R
		ycoord	nhdat	R
	or	hnests	(nhdat,nonodes)	D
		zcoord	(nhdat,nzdat)	R
io_sedspc	1	dp	nf	R
		rhos	nf	R
		tau_cr_cst	nf	R
		ws_cst	nf	R
io_1uvsur	1	gxslope_amp	nconobc	R
		gxslope_ph	nconobc	R
		gyslope_amp	nconobc	R
		gyslope_ph	nconobc	R
		zeta_amp	nconobc	R
		zeta_ph	nconobc	R
	2:nofiles	ciodate	time	C
		data1d	novars	R
io_2uvobc	1	ityp2dobu	nobu	I
		iloczobu	nobu	I
		ityp2dobv	nobv	I
		iloczobv	nobv	I
		no2dobuv	2:nofiles	I
		iobc2dtype	2:nofiles	I
		index2dobuv	(nobu+nobv,2:nofiles)	I
		udatobu_amp	(nobu,nconobc)	R
		zdatobu_amp	(nobu,nconobc)	R
		udatobu_ph	(nobu,nconobc)	R
		zdatobu_ph	(nobu,nconobc)	R
		vdatobv_amp	(nobv,nconobc)	R
		zdatobv_amp	(nobv,nconobc)	R

(Continued)

Table 14.3: *Continued*

		vdatobv_pha	(nobv,nconobc)	R
		zdatobv_pha	(nobv,nconobc)	R
		iqsecobu	(nqsecobu,2)	I
		jqsecobv	(nqsecobv,2)	I
	2:nofiles	ciodateime	—	C
		data2d	(nodat,novars)	R
io_2xyobc	1	ityp2dobx	nobx	I
		ityp2doby	noby	I
		no2dobxy	2:nofiles	I
		index2dobxy	(nobx+noby,2:nofiles)	I
		vdatobx_amp	(nobx,nconobc)	R
		vdatobx_pha	(nobx,nconobc)	R
		udatoby_amp	(noby,nconobc)	R
		udatoby_pha	(noby,nconobc)	R
	2:nofiles	ciodateime	—	C
		data2d	(nodat,novars)	R
io_3uvobc	1	itypobu	nobu	I
		iprofobu	nobu	I
		itypobv	nobv	I
		iprofobv	nobv	I
		noprofsd	2:nofiles	I
		indexprof	(nobu+nobv,2:nofiles)	I
		iprofrlx	norlxzones	I
	2:nofiles	ciodateime	—	C
		psiprofdat	(numprofs,nz)	R
io_3xyobc	1	itypobx	nobx	I
		iprofobx	nobx	I
		itypoby	noby	I
		iprofoby	noby	I
		noprofsd	2:nofiles	I
		indexprof	(nobx+noby,2:nofiles)	I
	2:nofiles	ciodateime	—	C
		psiprofdat	(numprofs,nz)	R
io_salobc	1	itypobu	nobu	I
		iprofobu	nobu	I
		itypobv	nobv	I
		iprofobv	nobv	I
		noprofsd	2:nofiles	I

(Continued)

Table 14.3: *Continued*

		indexprof	(nobu+nobv,2:nofiles)	I
		iprofrlx	norlxzones	I
	2:nofiles	ciodatetime	–	C
		psiprofdat	(numprofs,nz)	R
io_tmppobc	1	itypobu	nobu	I
		iprofobu	nobu	I
		itypobv	nobv	I
		iprofobv	nobv	I
		noprofsd	2:nofiles	I
		indexprof	(nobu+nobv,2:nofiles)	I
		iprofrlx	norlxzones	I
	2:nofiles	ciodatetime	–	C
		psiprofdat	(numprofs,nz)	R
io_sedobc	1	itypobu	nobu	I
		iprofobu	nobu,nf	I
		itypobv	nobv	I
		iprofobv	nobv,nf	I
		noprofsd	2:nofiles	I
		indexprof	(nf*(nobu+nobv),2:nofiles)	I
		indexvar	(nf*(nobu+nobv),2:nofiles)	I
		iprofrlx	norlxzones	I
	2:nofiles	ciodatetime	–	C
		psiprofdat	(numprofs,nz)	R
io_rlxobc	1	inodesrlx	2	I
		idirrlx	norlxzones	I
		ityprlx	norlxzones	I
		iposrlx	norlxzones	I
		jposrlx	norlxzones	I
		ncrlx	norlxzones	I
		nrrlx	norlxzones	I
io_nstspc	1	nestcoords	nonestsets	I
		nohnstglbc	nonestsets	I
		nohnstglbu	nonestsets	I
		nohnstglbv	nonestsets	I
		novnst	nonestsets	I
		inst2dtype	nonestsets	I
io_metsur	1	ciodatetime	–	C
		surdata	(n1dat,n2dat,novars)	R

(Continued)

Table 14.3: *Continued*

io_sstsur	1	ciodatetime	—	C
		surdata	(n1dat,n2dat,1)	R
io_wavsur	1	ciodatetime	—	C
		surdata	(n1dat,n2dat,novars)	R
io_drycel	1	idry	numdry	I
		jdry	numdry	I
io_thndam	1	ithinu	numthinu	I
		jthinu	numthinu	I
		ithinv	numthinv	I
		jthinv	numthinv	I
io_weibar	1	iwbaru	numwbaru	I
		jwbaru	numwbaru	I
		oricoefu	numwbaru	R
		oriheightu	numwbaru	R
		orisillu	numwbaru	R
		wbarcoefu	numwbaru	R
		wbarcrestu	numwbaru	R
		wbarmodlu	numwbaru	R
		iwbarv	numwbarv	I
		jwbarv	numwbarv	I
		oricoefv	numwbarv	R
		oriheightv	numwbarv	R
		orisillv	numwbarv	R
		wbarcoefv	numwbarv	R
		wbarcrestv	numwbarv	R
		wbarmodlv	numwbarv	R
io_disspc	1	kdistype	numdis	I
		mdistype	numdis	I
io_disvol	2:nofiles	ciodatetime	—	C
		disdata	(nodat,novars)	R
io_discur	2:nofiles	ciodatetime	—	C
		disdata	(nodat,novars)	R
io_dissal	2:nofiles	ciodatetime	—	C
		disdata	(nodat,novars)	R
io_distmp	2:nofiles	ciodatetime	—	C
		disdata	(nodat,novars)	R

14.5.3 Standard format of forcing files

14.5.3.1 ASCII files

The ASCII format is illustrated with two examples. They are taken from a case study for the North Sea (not discussed in the current version of the manual). Examples 14.9 and 14.10 show the contents of the files *nsp89.2uvobc1A* and *nsp89.metsurA*. The line numbers have been added in the header section for illustrative purposes only and do not appear in the actual file.

```

1 : 1
2 :V2.0
3 :2010/06/17;09:22:34
4 :Type of open boundary conditions for 2-D mode
5 : 0
6 : 17
7 : 620 3 1 29
8 :ityp2dobu
9 :Type of 2-D open boundary condition at U-nodes
10:- 11: 607 3 1 29
12:iloczobu
13:Position of elevation points with respect to U-open boundaries
14:- 15: 616 3 1 29
16:itypintobu
17:Switch to allow momentum advection near U-open boundaries
18:- 19: 621 3 1 111
20:ityp2dobv
21:Type of 2-D open boundary condition at V-nodes
22:- 23: 608 3 1 111
24:iloczobv
25:Position of elevation points with respect to V-open boundaries
26:- 27: 617 3 1 111
28:no2dobuv
29:Number of input data per input file
30:- 31: 612 3 1 2
32:iobc2dtype

```

```
33:Type 2-D open boundary data input
34:-
35:      611      3      2      140      2
36:index2dobuv
37:Map of data points to open boundary locations
38:-
39:      637      5      2      29      9
40:udatobu_amp
41:Amplitude of X-component of depth-integrated current at U-open
  boundaries
42:m^2/s
43:      648      5      2      29      9
44:zdatobu_amp
45:Amplitude of surface elevation at U-open boundaries
46:m
47:      638      5      2      29      9
48:udatobu_ph
49:Phase of X-component of depth-integrated current at U-open boundaries
50:radian
51:      649      5      2      29      9
52:zdatobu_ph
53:Phase of surface elevation at U-open boundaries
54:m
55:      640      5      2      111      9
56:vdatobv_amp
57:Amplitude of Y-component of depth-integrated current at V-open
  boundaries
58:m^2/s
59:      651      5      2      111      9
60:zdatobv_amp
61:Amplitude of surface elevation at V-open boundaries
62:m
63:      641      5      2      111      9
64:vdatobv_ph
65:Phase of Y-component of depth-integrated current at V-open boundaries
66:radian
67:      652      5      2      111      9
68:zdatobv_ph
69:Phase of surface elevation at V-open boundaries
70:m
ityp2dobu
```

```

          11      11 ...
iloczobu
          1      1 ...
ityp2dobv
          11     11 ...
iloczobv
          1      1 ...
no2dobuv
          127     13
iobc2dtype
          1      3
index2dobuv
          1      2 ...
udatobu_amp
  0.4421976  0.5311887 ...
zdatobu_amp
  0.3082999E-01  0.3055999E-01 ...
ud2obu_ph
  4.054574   4.223073 ...
zdatobu_ph
  5.249485   5.269729 ...
vdatobv_amp
  0.5446934  0.7153571 ...
zdatobv_amp
  0.4053500E-01  0.3966500E-01 ...
vdatobv_ph
  2.541209   2.624729 ...
zdatobv_ph
  0.2956865  0.3112219 ...

```

Example 14.9: Contents of the file *nsp89.2uvobc1A* in standard ASCII COHERENS format.

- Lines 1–6 give the values of the attributes:

<code>header_type</code>	the type of header which (in the current version) is always 1 for a forcing file
<code>coherens_version</code>	the current program version number, which is the same for all forcing files
<code>creation_date</code>	the exact date and time when the file was created
<code>filedesc</code>	a description of the file

nocoords	the number of coordinate variables (as defined in modfiles)
novars	the number of (non-coordinate) variables (as defined in modfiles)

- In case of a time series file, the next four lines list the attributes of the time coordinate (see Example 14.10).
- The remaining lines 7–70 show the attributes of the data variables.
- The attributes of each variable are given on four lines
 - line 1: the attributes **ivard**, **data_type**, **nrank**, **shape**, giving $3+nrank$ integer parameters on the input line
 - line 2: the **f90_name** attribute
 - line 3: the **long_name** attribute
 - line 4: the **units** attribute
- The total number of header lines is then given by $6+4*(\text{nocoords}+\text{novars})$.
- The data section gives the values of the variables in the order in which they have been defined in the header section.
 - In case of an initialisation file, the name of the variable is written on one line, followed by its values.
 - In case of a time series file, the value of the time coordinate is written first using the date/time string format (see Section 13.2.1), the data values are written as in the previous case except that the data values are preceded by an empty line (for illustration this line is presented in the example by the variable's name in []). The line next to the values of the last variable is the date/time of the next time record,

The contents of an ASCII forcing file are to be read sequentially, i.e. line by line.

- The lines in the header either contain a character string or integer parameters which makes it easier to read them using the character ('A') format (strings) or free (*) format (integers). The header information does not provide additional information for the program, since the attributes are already known internally, but are used for error checking only. For users who like to read the data from some external program,

the metadata provides useful information (number, rank and shape of the data variables).

- The data values (except the date/time string in time series files) are either of type **INTEGER** or **REAL** and are read in **FORTRAN** array order, e.g.

```
READ (iunit, IntegerFormat) itypintobu
READ (iunit, RealFormat) uwindatc
```

where the string format specifications **IntegerFormat** and **RealFormat** are system parameters, defined in *syspars.f90*. Values are

```
IntegerFormat='(50I11)'; RealFormat='(50G16.7)'
```

It is advised not to change these values (except for the repeat specification) since they allow to represent the data with the highest possible precision.

```
1 : 1
2 :V2.0
3 :2010/06/17;09:22:34
4 :Meteo input surface data
5 : 1
6 : 7
7 : 952 1 2 23 -1
8 :time
9 :Time
10:date/time
11: 410 5 2 50 28
12:uwindatc
13:X-component of surface wind
14:m/s
15: 411 5 2 50 28
16:vwindatc
17:Y-component of surface wind
18:m/s
19: 402 5 2 50 28
20:atmpres
21:Atmospheric pressure
22:N/m^2
```

```
23:      401      5      2      50      28
24:airtemp
25:Air temperature
26:degC
27:      407      5      2      50      28
28:relhum
29:Relative humidity
30:-
31:      403      5      2      50      28
32:cloud_cover
33:Cloud cover
34:-
35:      404      5      2      50      28
36:evapminprec
37:Evaporation minus precipitation rate
38:kg/m^2/s
1989/01/01;00:00:00,000
[uwindatc]
  0.8044688E-06  -0.2852140 ...
[vwindatc]
  9.202050        8.167472 ...
[atmpres]
  102672.9        102768.9 ...
[airtemp]
  13.85400        13.92300 ...
[relhum]
  0.9958699        0.9328000 ...
[cloud_cover]
  0.6250000        0.6250000 ...
[evapminprec]
  0.7811863E-05  0.7811863E-05 ...
1989/01/01;03:00:00,000
[uwindatc]
  -0.3340597      -0.5992587 ...
[vwindatc]
  9.566232        8.569798 ...
[atmpres]
  102548.8        102661.5 ...
[airtemp]
  13.95900        13.99600 ...
[relhum]
```

```

0.7983300      0.8664200 ...
[cloud_cover]
0.6250000      0.6250000 ...
[evapminprec]
0.7811863E-05  0.7811863E-05 ...
1989/01/01;06:00:00,000
...

```

Example 14.10: Contents of the file *nsp89.metsurA* in standard ASCII COHERENS format.

14.5.3.2 unformatted binary files

When the unformatted binary or ASCII format is selected, the same metadata and data are written in the forcing file. Differences are that in case of a binary format:

- String attributes, numerical attributes and data values are written in binary format. The form depends on the internal binary presentation of each (character, integer, real) data type. Common types are known as ‘NATIVE’, ‘LITTLE_ENDIAN’, ‘BIG_ENDIAN’, ‘IBM’.
- The file can be read only on machines which use the same binary presentation³.
- Reading of the file or conversion to a readable format can only be performed using some external (post-processing) program.
- The data are written sequentially as in the ASCII case but without a format specification.
- Data values are no longer preceded by a blank line.
- Prime advantage of this format is that each numerical value (defined in single precision) only uses (in most cases) only 4 bytes of disk space, reducing the required storage space by a factor 3 to 4 compared to the ASCII format.

The program provides an utility (using the `info` file attribute) to view the metadata contents of a binary file without additional programming tools. If the `info` attribute is set to `.TRUE.`, the program will create an additional “info” file with all metadata information in ASCII. An example is given below for the rhone test case.

³Some compilers provide a `CONVERT` specifier in a `FORTRAN OPEN` call allowing to make a conversion between two different binary formats.

```

header type:                      1
coherens version:                 V2.0
creation date:                   2010/06/18;16:49:53
file description:                 Model grid
nocoords:                         0
novars:                            6
      32 5 1          31
gsigcoord
Sigma coordinates on uniform grid
-
      105 5 2         108          50
depmeanglb
Global mean water depth at C-nodes
m
      43 3 1          65
iobu
Global X-index of U-open boundaries
-
      51 3 1          65
jobu
Global Y-index of U-open boundaries
-
      45 3 1          110
iobv
Global X-index of V-open boundaries
-
      53 3 1          110
jobv
Global Y-index of V-open boundaries
-

```

Example 14.11: Contents of *rhoneA.modgrdI* info file, giving all metadata information included in the rhone grid file.

14.5.3.3 netCDF files

The netCDF format is binary and non-sequential. The data are organised in records which can be read by specifying the appropriate record number as argument to a netCDF routine call. Moreover, the contents of the file, or part of the contents (metadata only, values of one or more data variables) can easily be converted to an ASCII format, which is the so-called CDL (network

Common Data form Language) format. For a full description of CDL, see [Russ *et al.* \(2004\)](#).

- Data and metadata are stored, in a platform-independent way, as (non-sequential) internal records.
- Reading and writing is performed by `netCDF` routine calls described in the `netCDF` manual ([Pincus & Rew, 2008](#)). Specific routines are available in the program to write metadata and data into a standard COHERENS format, similar to the one used in the ASCII and unformatted binary formats.
- An alias (starting with `cf_90`) is defined in the code for each `netCDF` routine call (starting with `NF90_`). The aim is a more efficient implementation of future `netCDF` versions.
- Although the file is in a compressed binary format, the contents or part of the contents of a `netCDF` file can be converted to ASCII form with the `ncdump` utility:

```
ncdump file.cdf
ncdump -h file.cdf
ncdump -v var file.cdf
```

The first form rewrites the file `file.cdf` to standard output in ASCII format. The second rewrites the metadata section only, the third form rewrites the metadata section and the values of the variable `var`.

Example 14.12 shows the metadata file in CDL format of the initial condition file for the North Sea case study. Differences with the ASCII a binary formats are:

- Variable dimensions must be defined with a given name.
- The `f90_name`, `nrank` and `shape` attributes are no longer explicitly defined, but provided implicitly when a `netCDF` variable is defined with the `NF_90_def_var` library call.
- Global attributes are the same as before except for the `cdfversion` attribute giving the current `netCDF` version used by the model and the `timerec` attribute giving the number of time records in the file.

Reading and writing of netCDF metadata and data is performed using the routines of the netCDF library. A detailed description is found in the netCDF FORTRAN 90 manual (Pincus and Rew, 2008).

```
netcdf nsp89
dimensions:
tlendim = 23 ;
T = UNLIMITED ; // (2 currently)
X002 = 141 ;
Y002 = 128 ;
...
Z020 = 2 ;
variables:
char time(T, tlendim) ;
time:ivarid = 952 ;
time:long_name = "Time" ;
time:units = "date/time" ;
float udvel(T, Y002, X002) ;
udvel:ivarid = 157 ;
udvel:long_name = "X-component_of_depth-integrated_current" ;
udvel:units = "m^2/s" ;
float vdvel(T, Y003, X003) ;
vdvel:ivarid = 166 ;
vdvel:long_name = "Y-component_of_depth-integrated_current" ;
vdvel:units = "m^2/s" ;
float zeta(T, Y004, X004) ;
zeta:ivarid = 113 ;
zeta:long_name = "Surface_elevation" ;
zeta:units = "m" ;
float uvel(T, Z005, Y005, X005) ;
uvel:ivarid = 162 ;
uvel:long_name = "X-component_of_current" ;
uvel:units = "m/s" ;
float vvel(T, Z006, Y006, X006) ;
vvel:ivarid = 171 ;
vvel:long_name = "Y-component_of_current" ;
vvel:units = "m/s" ;
float wvel(T, Z007, Y007, X007) ;
wvel:ivarid = 176 ;
wvel:long_name = "Transformed_vertical_velocity" ;
wvel:units = "m/s" ;
```

```
float temp(T, Z008, Y008, X008) ;
temp:ivarid = 205 ;
temp:long_name = "Temperature" ;
temp:units = "degC" ;
float sal(T, Z009, Y009, X009) ;
sal:ivarid = 204 ;
sal:long_name = "Salinity" ;
sal:units = "PSU" ;
float tke(T, Z010, Y010, X010) ;
tke:ivarid = 304 ;
tke:long_name = "Turbulent_kinetic_energy" ;
tke:units = "J/kg" ;
float fnode_ocb(T, X011) ;
fnode_ocb:ivarid = 353 ;
fnode_ocb:long_name = "Nodal_factors_of_tidal_constituents_at_open_boundaries" ;
fnode_ocb:units = "-" ;
float phase_ocb(T, X012) ;
phase_ocb:ivarid = 360 ;
phase_ocb:long_name = "Astronomical_phases_at_open_boundaries" ;
phase_ocb:units = "radian" ;
float obcsalatu(T, Z013, Y013, X013) ;
obcsalatu:ivarid = 625 ;
obcsalatu:long_name = "Storage_array_for_salinity_at_U-open_boundaries" ;
obcsalatu:units = "PSU" ;
float obcsalatv(T, Z014, Y014, X014) ;
obcsalatv:ivarid = 626 ;
obcsalatv:long_name = "Storage_array_for_salinity_at_V-open_boundaries" ;
obcsalatv:units = "PSU" ;
float obctmpatu(T, Z015, Y015, X015) ;
obctmpatu:ivarid = 627 ;
obctmpatu:long_name = "Storage_array_for_temperature_at_U-open_boundaries" ;
obctmpatu:units = "degC" ;
float obctmpatv(T, Z016, Y016, X016) ;
obctmpatv:ivarid = 628 ;
obctmpatv:long_name = "Storage_array_for_temperature_at_V-open_boundaries" ;
obctmpatv:units = "degC" ;
float obc2uvatu(T, Y017, X017) ;
obc2uvatu:ivarid = 629 ;
obc2uvatu:long_name = "Storage_array_for_2-D_mode_at_U-open_boundaries" ;
obc2uvatu:units = "-" ;
float obc2uvatv(T, Y018, X018) ;
```

```

obc2uvatv:ivarid = 630 ;
obc2uvatv:long_name = "Storage_array_for_2-D_mode_at_V-open_boundaries" ;
obc2uvatv:units = "-" ;
float obc3uvatu(T, Z019, Y019, X019) ;
obc3uvatu:ivarid = 631 ;
obc3uvatu:long_name = "Storage_array_for_3-D_mode_at_U-open_boundaries" ;
obc3uvatu:units = "m/s" ;
float obc3uvatv(T, Z020, Y020, X020) ;
obc3uvatv:ivarid = 632 ;
obc3uvatv:long_name = "Storage_array_for_3-D_mode_at_V-open_boundaries" ;
obc3uvatv:units = "m/s" ;

// global attributes:
:header_type = 1 ;
:coherens_version = "V2.0" ;
:creation_date = "2010/06/21;17:49:36" ;
:filedesc = "Physical initial conditions" ;
:cdfversion = "3.6.2" of" ;
:nocoords = 1 ;
:novars = 19 ;
:timerec = 2 ;
data:

time =
"1989/01/01;00:00:00,000",
"1989/12/25;00:00:00,000" ;
udvel =
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -0.08447912, 0.3973033, 0, ...
vdvel =
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
zeta =
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1.607709, 1.618565, 1.631894, 0, ...
uvel =
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -0.006357468, 0.006396886, 0, ...
vvel =
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
wvel =
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
temp =
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 10.34, 10.34, 10.34, 0, 0, 0, ...
sal =

```

```

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 35.4, 35.4, 35.4, 0, 0, 0, 0, 35.33, ...
tke =
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1.114186e-05, 5.660635e-06, ...
fnode_abc =
1.165938, 1.165991, 1.10667, 0.9655771, 0.9624547, 0.9655771, ...
phase_abc =
5.17733, 2.496852, 0.2368603, 5.49324, 5.436536, 2.75393, 3.223492, ...
obcsalatu =
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
obcsalatv =
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
obctmpatu =
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
obctmpatv =
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
obc2uvatu =
-20.18008, -20.35524, -18.70897, -17.60509, -17.31932, -16.37582, ...
obc2uvatv =
-0.6839569, 2.442981, 1.570055, 0.996551, 1.981653, 2.819804, ...
obc3uvatu =
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
obc3uvatv =
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...

```

Example 14.12: Metadata of the initial condition file for the North Sea test case in netCDF CDL format.

14.6 User output files

14.6.1 General aspects

The procedures for defining user output are discussed at length in Chapter 27. Only a summary, needed to understand the formatting of the data file discussed in the next subsection, will be given here.

The following forms of user output can be defined

- Time series: output of model data at regular time intervals.
- Time averaged: output of model data averaged over a specific period and repeated at regular time intervals.

- Harmonic: output of harmonically analysed model data over a specific period and repeated at regular time intervals. Output data may consist of residuals, amplitudes, phases and elliptic parameters.

User output is organised by defining so-called “output sets”. Each set is characterised by a specific spatial and time resolution and a number of selected variables. Each output set (except harmonic output) may contain 3 files at most:

- 0-D file: “globally” evaluated data (e.g. domain averaged temperature) or data without a spatial dimension (e.g. width of a river plume)
- 2-D file: 2-D (horizontal) data without a vertical dimension (e.g. water levels)
- 3-D file: 3-D data having both horizontal and vertical dimensions

For each file a derived type variable is defined of `TYPE(FileParams)`. The following attributes can be defined by the user:

<code>defined</code>	Activates or disactivates the file if set to <code>.TRUE.</code> or <code>.FALSE.</code> respectively.
<code>form</code>	Format of the output file 'A': ASCII 'U': unformatted binary 'N': netCDF
<code>info</code>	An info ('I') file is produced with the metadata information if <code>.TRUE.</code>
<code>header_type</code>	No header (metadata) will be written if set to zero. The option is not available for netCDF files.

The attributes `iunit`, `iostat`, `nocoords`, `novars` and `timeid` are defined internally and have the same meaning as for forcing files (see Section 14.5.1).

The user is free to choose the output grid which may be the full model grid, a subsection of the model grid or a completely irregular grid as is the case for data at station locations. In case of a 2-D or 3-D grid, the horizontal grid may be taken as one point so that the 2-D data reduce to one data value and the 3-D data to one vertical profile per variable and time step.

The attributes of the output grid are stored in a derived type variable of `TYPE(OutGridParams)`. The definitions of its attributes are given in Section ??, but repeated here for clarity.

gridded	If .TRUE. (default), the output data are defined on a sub-grid of the model grid (or the whole model grid). If .FALSE., the data are taken at a number of irregularly spaced locations (“station” data) defined by the user.
grid_file	If .TRUE., the coordinates of the output grid will be written on a separate output file. Otherwise, they are written within the data file itself (default).
land_mask	A land mask will be applied if .TRUE. and gridded =.TRUE.. This means that the gridded data will be stored as a vector excluding land points. Advantage may be a significant reduction in disk space. Disadvantage is that the data need to be restored on the original output grid by a postprocessing program. Default is .FALSE.
time_grid	If .TRUE., the data grid is taken as time-dependent (since the vertical positions in a σ -grid depend on time). Surface elevations will be written as an additional coordinate variable at each time step. Default is .FALSE., in which case the vertical positions are referred to the mean water level. This option is only available for time series output.
time_format	Format of the time coordinate.
	0: date/time in string format (default value)
	1: seconds
	2: minutes
	3: hours
	4: days
	5: months
	6: years
	7: date in years
	Cases 1-6 are numerical formats. Cases 0 and 7 are absolute times, while the others are times relative to the reference date/-time refdate .
refdate	Reference date/time for calculating relative times. If not given, refdate equals the first output date/time, rounded to the nearest minute, hour, ... depending on the value of time_format .
tlims	Start/end/step time indices for data output. This means that output will be written at intervals of delt2d*tlims(3) seconds.

nodim	Dimension of the output grid (0/2/3). For example, the dimension must be set to 3 to enable 3-D output.
nostats	Number of data stations in case of non-gridded (station) data.
xlims	Start/end/step X-index in case of gridded data. This defines the output sub-grid in the X-direction. (Option not available for 0-D or station output).
ylims	Start/end/step Y-index in case of gridded data. This defines the output sub-grid in the Y-direction. (Option not available for station or 0-D output).
zlims	Start/end/step Z-index in case of gridded data. This defines the output sub-grid in the Z-direction and applies for gridded and non-gridded output. (Option only available for 3-D output).

Attributes of variables and coordinate arrays of the output grid are stored in a derived type variable of **TYPE(VariableAtts)** (see Section 14.5.1). The program makes distinction between two types of variables:

- Standard variables whose attributes are known to the program. In that case the variable's key id is supplied by the user and only a limited number of definitions need to be made. A number of operators (minimum, maximum or mean value, value at a given vertical level or depth) can be applied to the output data. The output values are automatically generated by the program.
- User-defined variables in which case all attributes and output values are to be defined by the user.

The procedures for defining time series output can be summarised as follows.

1. Define the three general parameters

nosetsts number of sets
novarsts total number of output variables
nostatsts total number of output stations

2. Define the attributes of all variables used for time series output. Attributes can be automatically generated if the **ivard** attribute is defined by the user.

3. For each set

- Select which files are to be written (0-D, 2-D, 3-D) using the `define` attribute.
- Define the file attributes `form`, `filename`, `info`, `header_type`. Defaults are available.
- Select the variables for each output file.
- Define the attributes of the output grid. Defaults are available.

The procedures for time averaged output are the same, except that an averaging period needs to be defined instead of an output time interval.

For harmonic output additional parameters need to be defined. Assume then that N frequencies are selected within one set. The following output files may be defined within that set:

- 0-D: 1 residual, N amplitude and N phase files
- 2-D: 1 residual, N amplitude, N phase and N elliptic (tidal ellipse parameters) files
- 3-D: 1 residual, N amplitude, N phase and N elliptic files

giving a maximum of $3+8N$ files per set.

In addition to the the procedures for time series output, harmonic output requires to:

1. Define the total number of frequencies `nofreqsanal`.
2. Define all harmonic frequencies.
3. For each set
 - select which of the $3+8N$ files are written
 - select output frequencies
 - select elliptic parameters (optional)
 - select the period for analysis.

14.6.2 Structure of user output files

The general structure is similar to the one used for forcing files, except that the file additionally contains the coordinates of the output grid.

```

[Names and values of dimensions]
Global attributes
...
Attributes of spatial coordinate 1
...
Attributes of spatial coordinate 2
...
Attributes of time coordinate
...
Attributes of variable 1
...
Attributes of variable 2
...
Attributes of variable n
...
Values of spatial coordinate 1
...
Values of spatial coordinate 2
...
First output time
[Surface elevation coordinate]
Values of variables 1
...
Values of variables n
Second output time
...

```

Example 14.13: General layout of a user output file.

In principle, the number of coordinate variables should be equal to or lower than four (three spatial and one time variable). However, the model uses a σ -grid which is fixed in time in the transformed coordinate system, but not in physical space due to the up and down movements of the water column. In most graphical applications these changes are negligible and the total water depth is approximated by its mean value. However, time-varying grids can be taken into account in the current version of COHERENS. The output grid can then be defined through the following six coordinates (where the spatial dimension is given in parentheses):

xout X-coordinate in m or fractional degrees longitude (2-D)
yout Y-coordinate in m or fractional degrees latitude (2-D)

zout Z-coordinate in m using mean water depths, i.e. between $-h$ and 0 (3-D)
depout mean water depth in m (2-D)
zetout surface elevation in m (2-D)
time output time (0-D). Unit is specified by the `time_format` attribute.

It is clear that

- In the case of a 0-D output only the time coordinate is included.
- In the case of a 2-D output only `xout`, `yout`, `depout` and `time` are included.
- Elevation data are only included if the user selects a time varying 3-D output grid using the `time_grid` attribute.

The first four arrays have no associated time dimension. The last two are stored within the file in chronological order at regular time intervals. In the ASCII and unformatted binary format, storage is sequential. In the netCDF format, data are stored with increasing record numbers at subsequent times.

14.6.3 Format of files with user-defined output

14.6.3.1 ASCII files

The ASCII format is illustrated with examples taken from the plume test case. The first shows the contents of the output file *plumeA_2.out3A*. The line numbers have been added in the header section for illustrative purposes only and do not appear in the actual file.

```

1 :          2
2 :V2.0
3 :2010/06/23;12:19:42
4 :Time series data: plume1A
5 :          3
6 :F
7 : T F F
8 :          0          1          1
9 :          1          40         20          0          0          25
10: 10800.00
11:2003/01/03;00:00:00,000
12:2003/01/06;00:00:00,000

```

```
13:2003/01/03;00:00:00,000
14:          0
15:          5
16:          5
17:          4
18:    957      5      2      1      40
19:xout
20:X-coordinate
21: m
22:    962      5      2      1      40
23:yout
24:Y-coordinate
25: m
26:    968      5      3      1      40      20
27:zout
28:Z-coordinate
29: m
30:    951      5      2      1      40
31:depout
32:Mean water depth
33: m
34:    952      5      2      23      25
35:time
36:Time
37: date/time
38:    162      5      4      1      40      20      25
39:uvel
40:X-component of current
41:m/s
42:Current
43:    171      5      4      1      40      20      25
44:vvel
45:Y-component of current
46:m/s
47:Current
48:    175      5      4      1      40      20      25
49:wphys
50:Physical vertical velocity
51:m/s
52:Physical current
53:    204      5      4      1      40      20      25
```

```

54:sal
55:Salinity
56:PSU
57:
[xout]
  29500.00      29500.00      ...
[yout]
  500.0000      1500.000      ...
[zout]
  -19.50000     -19.50000     ...
[depout]
  20.00000      20.00000      ...
2003/01/03;00:00:00,000
[uvel]
  -0.2228398    -0.2281678    ...
[vvel]
  -0.8393249E-02 -0.1428325E-01 ...
[wphys]
  0.000000      0.000000      ...
[sal]
  33.00000      33.00000      ...
2003/01/03;03:00:00,000
...

```

Example 14.14: Contents of the output data file *plumeA_2.tsout3A* from the plume test case.

- Line 1: the `header_type` attribute. If set to 0, no further header information will be given (except this line). Otherwise, its value is 2.
- Line 2: the `coherens_version` attribute with the currently used COHERENS version.
- Line 3: the `creation_date` giving the exact date and time when the file was created.
- Line 4: the `filedesc` attribute with a description of the simulation.
- Line 5: the `nodim` attribute giving the spatial dimension of the output grid.
- Line 6: the `grid_file` attribute. If `.TRUE.` (`.FALSE.`), grid data and metadata are (are not) written to a separate data file.

- Line 7: the `gridded`, `land_mask` and `time_grid` attributes.
- Line 8: values of the switches of `iopt_grid_sph`, `iopt_grid_htype` and `iopt_grid_vtype` which define the type of grid (see Section ??).
- Line 9: the attributes `ncout`, `nrou`, `nzout`, which define the size of the output grid (gridded case), `nowetout`, giving the number of sea points in the output grid (if the `land_mask` attribute is `.TRUE.`, zero otherwise), `nostats` (the number of stations in the non-gridded case) and `nstepout` (number of time records).
- Line 10: the output time step in seconds (`deltout` attribute).
- Line 11: date/time of first output (`startdate` attribute).
- Line 12: date/time of last output (`enddate` attribute).
- Line 13: reference date/time (`refdate` attribute). If the time coordinate has a numerical format, the time is given as the time elapsed from this date.
- Line 14: the `time_format` attribute.
- Line 15: the `nocoords` attribute giving the number of grid coordinates.
- Line 16: the `timeid` attribute giving the variable id of the time coordinate. This attribute is only used for reading the time coordinate in a `netCDF` file.
- In case of a time varying grid (only), the `zetaid` attribute giving the variable id of the `zetout` coordinate, used for reading the surface elevation coordinate in a `netCDF` file.
- Line 17: the `novars` attribute giving the number of data variables.
- In case of station data, the labels and names of the stations are written on subsequent lines with one line per station.
- Lines: 18-37: attributes of each coordinate variable using four lines per variable.
 - line 1: the attributes `ivarid`, `data_type`, `nrank`, `shape`, represented by `3+nrank` integer parameters
 - line 2: the `f90_name` attribute
 - line 3: the `long_name` attribute

- line 4: the `units` attribute
- Lines 38–57: attributes of each data variable using five lines per variable
 - line 1: the attributes `ivarid`, `data_type`, `nrank`, `shape`, represented by `3+nrank` integer parameters
 - line 2: the `f90_name` attribute
 - line 3: the `long_name` attribute
 - line 4: the `units` attribute
 - line 5: the `vector_name` attribute
- The total number of header lines in the current example is then given by `17+4*nocoords+5*novars`.
 - If `nostats>0`, `nostats` lines have to be added.
 - If `time_grid` is `.TRUE.`, there is 1 additional line for the `zetaid` attribute. In that case `nocoords` is also increased by 1.
 - If `grid_file` is `.TRUE.`, lines 7–16 and 18–37 are moved to the header of the grid file. The total number of header lines is then decreased by `10+4*nocoords`.
- The remaining lines in the example form the data section, which is composed of two parts:
 - The first part lists (if the `time_grid` attribute is `.TRUE.`) the values of the time-independent coordinate arrays, preceded by an empty line and written in the same order as defined in the header.
 - The second lists (if the `time_grid` attribute is `.TRUE.`) the values of the time-dependent coordinate arrays (and data variables in the following order:
 - * `time` coordinate
 - * `zetout` variable (if `time_grid` is `.TRUE.` and preceded by an empty line)
 - * values of each data variable in the same order as defined in the header. Each variable list is preceded by an empty line.
 - * For clarity, the name of the variable is substituted in `[]` within the example.
 - The same procedure is followed for the next time records.

The contents of an ASCII forcing file are to be read sequentially, i.e. line by line.

- The parameters listed on each line all have the same data type (except for the station parameters) which makes it easy to read them using the character ('A') format (strings) or in free (*) format (integer, real, logical data).
- The data values (except the date/time string if the time coordinate is given in a non-numeric format) are all of type **REAL**, read in **FORTRAN** array order. Format specifications are the same as for forcing files (see Section 14.5.3.1), i.e. '(A)' for the date/time string and **RealFormat** for the other variables.

If **grid_file** is set to **.TRUE.**, all coordinate information and data are moved to a separate grid file (i.e. lines 7–16, 18–37 and the values of the coordinate arrays).

The next example is a 0-D output file from the test case **fredyA**.

```

2
V2.0
2010/06/24;14:07:32
Time series data: fredyA
0
F
T F F
0      1      1
0      0      0      0      0      865
600.0000
2003/01/01;00:00:00,000
2003/01/07;00:00:00,000
2003/01/01;00:00:00,000
0
1
1
8
952      5      2      23      865
time
Time
date/time
0      5      1      60
ekin

```

Kinetic energy
GJ

0 5 1 60

epot

Available potential energy
GJ

0 5 1 60

theta

Energy ratio

0 5 1 60

enstr

Enstrophy
 m^3/s^2

0 5 1 60

a1pt

A1%

10^8 m^2

0 5 1 60

salmin

Minimum salinity

PSU

0 5 1 60

salmax

Maximum salinity

PSU

0 5 1 60

smean

Mean salinity deviation

PSU

2003/01/01;00:00:00,000

0.000000	9.042427	0.000000	0.000000
----------	----------	----------	----------

```

0.2500000      33.75000      34.85000      -0.3296069E-02
2003/01/01;00:10:00,000

0.2259467E-01    8.989937      0.2513329E-02    0.1228950E-04
0.3700000      33.75000      34.85001      -0.3295073E-02
...
2003/01/07;00:00:00,000

0.6530415      7.097948      0.9200427E-01    0.2413782
3.370000      34.27942      34.85036      -0.3307598E-02

```

Example 14.15: Contents of the output data file *freddyA_2.tsout0A* from the ***freddyA*** test case.

- Since the data have no spatial dimension in a 0-D file, the `time` variable is the only coordinate.
- Contrary to the general case, 0-D data are not stored individually but as a vector of size `novars`. They are read using

```

REAL, DIMENSION(novars) :: out0ddat

READ (iunit,'(A)') ciodate
READ (iunit,RealFormat) out0ddat
ekin = out0ddat(1); ...; smean = out0ddat(novars)

```

- The `ivarid` attribute is zero for all data variables. This means that the `f90_name`, `long_name` and `units` attributes are non-standard and defined by the user.

14.6.3.2 unformatted binary files

When the unformatted binary format is selected for a forcing file, the same metadata and data are written as in the case of an ASCII format. The differences are the same as described in Section 14.5.3.2 for forcing files.

14.6.3.3 netcdf files

The description of the netCDF format is similar to the one given in Section 14.5.3.3 and will not be repeated here. Two examples are given below. The first one lists the contents of the file *plumeA_1.ellip3N*, with harmonic output from the test case ***plumeA***.

```
netcdf plumeA_1
dimensions:
  xdim = 120 ;
  ydim = 40 ;
  zdim = 20 ;
  tlendim = 23 ;
  tdim = UNLIMITED ; // (6 currently)
variables:
  float xout(ydim, xdim) ;
  xout:ivarid = 957 ;
  xout:long_name = "X-coordinate" ;
  xout:units = "m" ;
  float yout(ydim, xdim) ;
  yout:ivarid = 962 ;
  yout:long_name = "Y-coordinate" ;
  yout:units = "m" ;
  float zout(zdim, ydim, xdim) ;
  zout:ivarid = 968 ;
  zout:long_name = "Z-coordinate" ;
  zout:units = "m" ;
  float depout(ydim, xdim) ;
  depout:ivarid = 951 ;
  depout:long_name = "Mean_water_depth" ;
  depout:units = "m" ;
  char time(tdim, tlendim) ;
  time:ivarid = 952 ;
  time:long_name = "Time" ;
  time:units = "date/time" ;
  float ellmin3d(tdim, zdim, ydim, xdim) ;
  ellmin3d:ivarid = 0 ;
  ellmin3d:long_name = "M2-Ellipticity" ;
  ellmin3d:units = "-" ;
  ellmin3d:vector_name = "_" ;

// global attributes:
:header_type = 2 ;
:coherens_version = "V2.0" ;
:creation_date = "2010/06/23;12:19:42" ;
:filedesc = "Elliptic parameters" ;
:cdfversion = "3.6.2" of" ;
:iopt_CDF_fill = 0 ;
```

```
:grid_dimension = 3 ;
:grid_file = 0 ;
:gridded = 1 ;
:land_mask = 0 ;
:time_grid = 0 ;
:grid_type = 0, 1, 1 ;
:dimensions = 120, 40, 20, 0, 0, 6 ;
:time_step = 43200.f ;
:startdate = "2003/01/03;06:00:00,000" ;
:enddate = "2003/01/05;18:00:00,000" ;
:refdate = "2003/01/03;06:00:00,000" ;
:time_format = 0 ;
:nocoords = 5 ;
:timeid = 5 ;
:novars = 1 ;
:timerec = 6 ;
data:

xout =
 500, 1500, ...

yout =
 500, 500, ...

zout =
 -19.5, -19.5, ...

depout =
 20, 20, ...

time =
 "2003/01/03;06:00:00,000",
 "2003/01/03;18:00:00,000",
 "2003/01/04;06:00:00,000",
 "2003/01/04;18:00:00,000",
 "2003/01/05;06:00:00,000",
 "2003/01/05;18:00:00,000" ;

ellmin3d =
 0.03929285, 0.03556633, ...
 -0.1459797, -0.1480648, -0.1499712, -0.1511258, -0.1511276, -0.1443668 ;
```

Example 14.16: Contents of the output data file *plumeA_1.ellip3N* from the **plumeA** test case.

The attributes are similar to the ones listed in the ASCII format. Some are defined with a different name (e.g. the `dimensions` attribute which combines the values of the previous parameters `ncout`, `nrou`, `nzout`, `nowetout`, `nostats`, `nstepout` into one vector). Other attributes are defined implicitly such as `f90_name`, `nrank` and `shape`. The `timerec` giving the current number of (time) records and `cdffversion` with the current version of netCDF, do not exist in the ASCII and unformatted binary formats.

The second example is the same as Example 14.17 now given in CDL format.

```
netcdf freddyA_2
dimensions:
  tlendim = 23 ;
  tdim = UNLIMITED ; // (865 currently)
variables:
  char time(tdim, tlendim) ;
  time:ivardid = 952 ;
  time:long_name = "Time" ;
  time:units = "date/time" ;
  float ekin(tdim) ;
  ekin:ivardid = 0 ;
  ekin:long_name = "Kinetic_energy" ;
  ekin:units = "GJ" ;
  ekin:vector_name = "_" ;
  float epot(tdim) ;
  epot:ivardid = 0 ;
  epot:long_name = "Available_potential_energy" ;
  epot:units = "GJ" ;
  epot:vector_name = "_" ;
  float theta(tdim) ;
  theta:ivardid = 0 ;
  theta:long_name = "Energy_ratio" ;
  theta:units = "_" ;
  theta:vector_name = "_" ;
  float enstr(tdim) ;
  enstr:ivardid = 0 ;
  enstr:long_name = "Enstrophy" ;
```

```
enstr:units = "m^3/s^2" ;
enstr:vector_name = "_" ;
float a1pt(tdim) ;
a1pt:ivarid = 0 ;
a1pt:long_name = "A1%" ;
a1pt:units = "10^8_m^2" ;
a1pt:vector_name = "_" ;
float salmin(tdim) ;
salmin:ivarid = 0 ;
salmin:long_name = "Minimum_salinity" ;
salmin:units = "PSU" ;
salmin:vector_name = "_" ;
float salmax(tdim) ;
salmax:ivarid = 0 ;
salmax:long_name = "Maximum_salinity" ;
salmax:units = "PSU" ;
salmax:vector_name = "_" ;
float smean(tdim) ;
smean:ivarid = 0 ;
smean:long_name = "Mean_salinity_deviation" ;
smean:units = "PSU" ;
smean:vector_name = "_" ;

// global attributes:
:header_type = 2 ;
:coherens_version = "V2.0" ;
:creation_date = "2010/06/24;09:55:22" ;
:filedesc = "Time series data: fredyA" ;
:cdfversion = "3.6.2" of" ;
:iopt_CDF_fill = 0 ;
:grid_dimension = 0 ;
:grid_file = 0 ;
:gridded = 1 ;
:land_mask = 0 ;
:time_grid = 0 ;
:grid_type = 0, 1, 1 ;
:dimensions = 0, 0, 0, 0, 0, 865 ;
:time_step = 600.f ;
:startdate = "2003/01/01;00:00:00,000" ;
:enddate = "2003/01/07;00:00:00,000" ;
:refdate = "2003/01/01;00:00:00,000" ;
```

```

:time_format = 0 ;
:nocoords = 1 ;
:timeid = 1 ;
:novars = 8 ;
:timerec = 865 ;
data:

time =
"2003/01/01;00:00:00,000",
...
"2003/01/07;00:00:00,000" ;

ekin = 0, 0.02259467, 0.08382063, ...

epot = 9.042427, 8.989937, ...

theta = 0, 0.002513329, ...

enstr = 0, 1.22895e-05, ...

a1pt = 0.25, 0.37, ...

salmin = 33.75, 33.75, ...

salmax = 34.85, 34.85001, ...

smean = -0.003296069, -0.003295073, ...

```

Example 14.17: Contents of *freddyA_2.tsout0N* output data file from the ***freddyA*** test case in CDL format.

Contrary to the ASCII case, a separate **netCDF** call has to be made to read each of the 0-D output data, i.e. the data are not stored in vector form but as separate records in the **netCDF** file.

Chapter 15

Model grid and spatial interpolation

15.1 Model grid arrays

15.1.1 Array shapes

The layout of the model grid and the grid indexing system have been presented in Section 4.2.1. The shape of a model array defined on the model grid (further denoted as “model grid arrays”) depends on its nodal location. If the model code would have been designed for serial (non-parallel) applications only, the shape of a model grid array could be obtained from Table 4.1. For example, the shapes of a C-node, U-node or W-node array would be (nc, nr, nz) , (nc, nr, nz) or $(nc, nr, nz+1)$, where it is recalled that nc , nr , nz are the (computational) grid dimensions in the X-, Y- and Z-direction.

Since the model can be set up either in serial or in parallel mode, two additional complexities arise:

1. The parallel code is based upon non-shared memory between the processes. This means that model grid arrays are only defined locally. The local array shape then depends on the dimensions of the local process domains. The local dimensions in the X- and Y-direction are given by the process-dependent parameters `ncloc` and `nrloc`, whereas the vertical dimension is still given by `nz`.
2. The solution of the discretised equations on a parallel grid requires that the domain on which an array is defined, must extend into the neighbouring domains. This extended area, called the halo area, is defined by adding extra rows and columns. The maximum size of the

halo in each of the four directions (West/East/South/North) is given by the parameter `nhalo` = 2. The actual size is array-dependent.

Examples of array shapes declarations are illustrated below.

```
REAL, DIMENSION(1-nhalo:ncloc+nhalo,1-nhalo:nrloc+nhalo,nz) :: sal
REAL, DIMENSION(0:ncloc+1,0:nrloc+1) :: zeta
REAL, DIMENSION(0:ncloc,0:nrloc) :: atmpres
REAL, DIMENSION(ncloc,nrloc,nz+1) :: vdifcoefscal
```

Example 15.1: examples of model grid array bounds.

The first array has a full size halo of two rows and columns in each direction, the second a halo of one row and column in each direction, the third a one column halo attached on its western boundary, whereas the last one is defined without halo.

In Section 4.2.1 it was explained that the last array column at the eastern boundary and the last row at the northern boundary of the computational grid consist of dummy land points. Important to note that this is mostly not the case for local array definitions, i.e. the horizontal grid points with index `(ncloc,j)` and `i,nrloc` are physical points unless one of the edges of the sub-domain extends into one of these two dummy boundary areas.

Local arrays usually have no global equivalent in the program. However, in case that a global array has to be defined, then it must be declared, for consistency, with the same halo sizes as its local equivalent. For example, the arrays

```
gxcoord(0:ncloc+1,0:nrloc+1), gxcoordgbl(0:nc+1,0:nr+1)
```

are the local and global representations of the same array.

A more detailed account of parallel grids is presented in Chapter 16.

15.1.2 Parameters and arrays related to the model grid

15.1.2.1 definition of the model grid

The following parameters and arrays are required for the definition of the model grid:

1. The number of grid cells in the X-, Y- and vertical direction: parameters `nc`, `nr`, `nz`.
2. Horizontal grid

2.1 Rectangular uniform. The following attributes of the derived variable `surfacegrids(igrd_model,1)` are needed:

`x0dat` reference coordinate x_r or longitude λ_r
`y0dat` reference coordinate y_r or latitude ϕ_r
`delxdat` uniform grid spacing Δx or $\Delta \lambda$ in the X-direction
`delydat` uniform grid spacing Δy or $\Delta \phi$ in the Y-direction
`rotated` .TRUE in case a rotated grid is selected
`gridangle` grid rotation angle α in case of rotated grids
`y0rot` reference latitude ϕ'_r in case of a rotated spherical grid

2.2 Rectangular non-uniform.

- The following attributes of the derived variable `surfacegrids(igrd_model,1)` are required:

`x0dat` reference coordinate x_r or longitude λ_r
`y0dat` reference coordinate y_r or latitude ϕ_r
`rotated` .TRUE in case a rotated grid is selected
`gridangle` grid rotation angle α in case of rotated grids
`y0rot` reference latitude ϕ'_r in case of a rotated spherical grid

- The grid spacings are stored in the arrays `gdelxglb(1:nc)` and `gdelyglb(1:nr)` for respectively the X- and Y-direction.

2.3 Fully curvilinear.

- The coordinates are obtained with respect to a reference location whose position is determined by attributes of the derived variable `surfacegrids(igrd_model,1)`

`x0dat` reference x-coordinate x_r or longitude λ_r
`y0dat` reference y-coordinate y_r or latitude ϕ_r

- The geographical locations of the cell corners (solid circles in Figure 4.7) with respect to the reference point are stored firstly in the arrays `gxcoordglb(1:nc,1:nr)` and `gycoordglb(1:nc,1:nr)`, after which the reference coordinates are added to the arrays. The reason for this procedure is to avoid rounding errors for fine-resolution grids.

3. Vertical grid

3.1 Vertically uniform grid. This is the default case. The vertical grid, i.e. σ -coordinates of the cells corners are stored in the 3-D array `gscoord(1:nc-1,1:nr-1,1:nz+1)` by the program.

3.2 Vertically non-uniform, horizontally uniform.

- If the switch `iopt_grid_vtype_transf` is not set, the vertical array `gsigcoord(1:nz+1)` is supplied and stored by the program in `gscoord(1:nc-1,1:nr-1,1:nz+1)` at each horizontal location. Note that `gscoord` must be equal to 0 at the lowest and 1 at the highest cell.
- If `iopt_grid_vtype_transf` is set to 11, 12 or 13, the vertical grid is obtained by the program using the transformation (4.23), (4.24) or (4.26).

3.3 Vertically and horizontally non-uniform.

- If the switch `iopt_grid_vtype_transf` is not set, the full 3-D array `gscoord(1:nc-1,1:nr-1,1:nz+1)` must be given at each horizontal and vertical position. Note that the array must be equal to 0 at the lowest and 1 at the highest cell.
- If `iopt_grid_vtype_transf` is set to 21, the vertical grid is obtained by the program using the [Song & Haidvogel \(1994\)](#) transformation given by equations (4.33) and (4.35).

A list of the grid parameters and arrays defining a model grid in COHERENS is given in Table 15.1.

- Local (global) means that the parameter or array is defined on the local sub-domain (global computational grid). In case of a serial application, local and global definitions coincide (e.g. `ncloc=nc`)
- The bounds of the global array are obtained from the local one by replacing `(nc,nr)` with `(ncloc,nrloc)`.

15.1.2.2 definition of the open boundaries

The following information has to be given by the user

- the number of open sea (`nosbu,nosbv`) and river (`nrvbu,nrvbv`) boundaries at U- and V-nodes
- the location of the open boundaries at U- and V-nodes: arrays `iobu(nobu)`, `jobu(nobu)`, `iobv(nobv)`, `jobv(nobv)` where `nobu`, `nobv` are the total (open sea and river) boundaries at U- and V-nodes

Table 15.1: Model grid parameters and arrays.

Local name	Global name	Local array bounds	Purpose
ncloc	nc	—	number of grid cells in the X-direction
nrloc	nr	—	number of grid cells in the Y-direction
nz	nz	—	number of grid cells in the Z-direction
gxcoord	gxcoordglb	(0:ncloc+1,0:nrloc+1)	X-coordinates of the cell corners (UV-nodes) in meters or degrees longitude (between -180^0 and 180^0)
gycoord	gycoordglb	(0:ncloc+1,0:nrloc+1)	Y-coordinates of the cell corners (UV-nodes) in meters or degrees latitude (between -90^0 and 90^0)
gscoord	gscoordglb	(0:ncloc+1,0:nrloc+1, nz+1)	σ -level coordinate at the W-nodes.

The following remarks apply:

- At a U- or V-node open boundary one of the two adjacent cells must be wet (sea) while the other one must be either a land cell or located outside the physical domain. This allows to define linear as well as “ragged” (stair-case) open boundaries.
- Open sea boundaries at U-nodes must be stored in `iobu(1:nosbu)`, `jobu(1:nosbu)`, river boundaries in `iobu(nosbu+1:nosbu+nrvbu)`, `jobu(nosbu+1:nosbu+nrvbu)`. A similar procedure is taken at V-nodes.
- Taking account of the previous restriction, the order of storage is irrelevant. However, the ordering is of importance for the definition of the open boundary conditions (see Chapter 21) since it introduces an indexing system, i.e. `ii` is (e.g.) the index of the U-open boundary located at (`iobu(ii)`, `jobu(ii)`).

Besides U- and V-open boundaries the program also uses internally the concept of X- and Y-open boundaries which are defined as follows:

- A corner node is a X-open boundary if one of the adjacent U-nodes is an open boundary and the other one a land/coastal or open sea boundary.

Table 15.2: Open boundary parameters and arrays.

Local name	Global name	Local array bounds	Purpose
<code>nosbuloc</code>	<code>nosbu</code>	—	number of West/East open sea boundaries at U-nodes
<code>nrvbuloc</code>	<code>nrvbu</code>	—	number of West/East river open boundaries at U-nodes
<code>nobuloc</code>	<code>nobu</code>	—	total number of West/East open boundaries at U-nodes (<code>nobuloc</code> = <code>nosbuloc+nrvbuloc</code>)
<code>nosbvloc</code>	<code>nosbv</code>	—	number of South/North open sea boundaries at V-nodes
<code>nrvbvloc</code>	<code>nrvbv</code>	—	number of South/North river open boundaries at V-nodes
<code>nobvloc</code>	<code>nobv</code>	—	total number of South/North open boundaries at V-nodes (<code>nobvloc</code> = <code>nosbvloc+nrvbvloc</code>)
<code>nobxloc</code>	<code>nobx</code>	—	number of X-open boundaries at corner nodes
<code>nobyloc</code>	<code>noby</code>	—	number of Y-open boundaries at corner nodes
<code>iobuloc</code>	<code>iobu</code>	<code>nobuloc</code>	X-indices of the West/East open boundaries at U-nodes
<code>jobuloc</code>	<code>jobu</code>	<code>nobuloc</code>	Y-indices of the West/East open boundaries at U-nodes
<code>iobvloc</code>	<code>iobv</code>	<code>nobvloc</code>	X-indices of the South/North open boundaries at V-nodes
<code>jobvloc</code>	<code>jobv</code>	<code>nobvloc</code>	Y-indices of the South/North open boundaries at V-nodes
<code>iobxloc</code>	<code>iobx</code>	<code>nobxloc</code>	X-indices of the open boundaries at X-nodes
<code>jobxloc</code>	<code>jobx</code>	<code>nobxloc</code>	Y-indices of the open boundaries at X-nodes
<code>iobyloc</code>	<code>ioby</code>	<code>nobyloc</code>	X-indices of the open boundaries at Y-nodes
<code>jobyloc</code>	<code>joby</code>	<code>nobyloc</code>	Y-indices of the open boundaries at Y-nodes

- A corner node is a Y-open boundary if one of the adjacent V-nodes is an open boundary and the other one a land/coastal or open sea boundary.
- Nodes located at the corners of the physical domain are excluded.

The indices of the X- and Y-open boundaries are defined within the program. The indexing scheme is illustrated in Figure 15.1. For X-node open boundaries, the point with index 1 is located below or above the U-open boundary 1, the point with index 2 above the U-point with index 1 or below the U-point with index 2, and so on. The scheme is similar for Y-node boundaries.

A list of all grid parameters and arrays is given in Table 15.2.

- Local (global) means that the parameter or array is defined on the local sub-domain (global computational grid). In case of a serial application, local and global definitions coincide (e.g. `nosbuloc=nosbu`).
- The bounds of the global array are obtained from the local one by replacing the local dimension by the global one (e.g. `nobuloc` by `nobu`).

15.1.2.3 grid spacings

Grid spacings in the horizontal direction are calculated by the standard formulae

$$\begin{aligned}\Delta X_{ij}^v &= h_{1;ij}^v = \sqrt{(x_{i+1,j} - x_{ij})^2 + (y_{i+1,j} - y_{ij})^2} \\ \Delta Y_{ij}^u &= h_{2;ij}^u = \sqrt{(x_{i,j+1} - x_{ij})^2 + (y_{i,j+1} - y_{ij})^2}\end{aligned}\quad (15.1)$$

or

$$\begin{aligned}\Delta X_{ij}^v &= h_{1;ij}^v = R \sqrt{(\lambda_{i+1,j} - \lambda_{ij})^2 + \cos \frac{1}{2}(\phi_{ij} + \phi_{i+1,j})(\phi_{i+1,j} - \phi_{ij})^2} \\ \Delta Y_{ij}^u &= h_{2;ij}^u = R \sqrt{(\lambda_{i,j+1} - \lambda_{ij})^2 + \cos \frac{1}{2}(\phi_{ij} + \phi_{i,j+1})(\phi_{i,j+1} - \phi_{ij})^2}\end{aligned}\quad (15.2)$$

depending on whether Cartesian or spherical coordinates are used. Similar expressions are used to evaluate the grid spacings at other nodes. Grid spacings in the vertical are calculated using $h_3 = H\Delta\sigma$ where H is the total water depth and $\Delta\sigma$ the σ -difference between two neighbouring levels.

The following grid spacing arrays are defined:

- grid spacing h_1 in meters

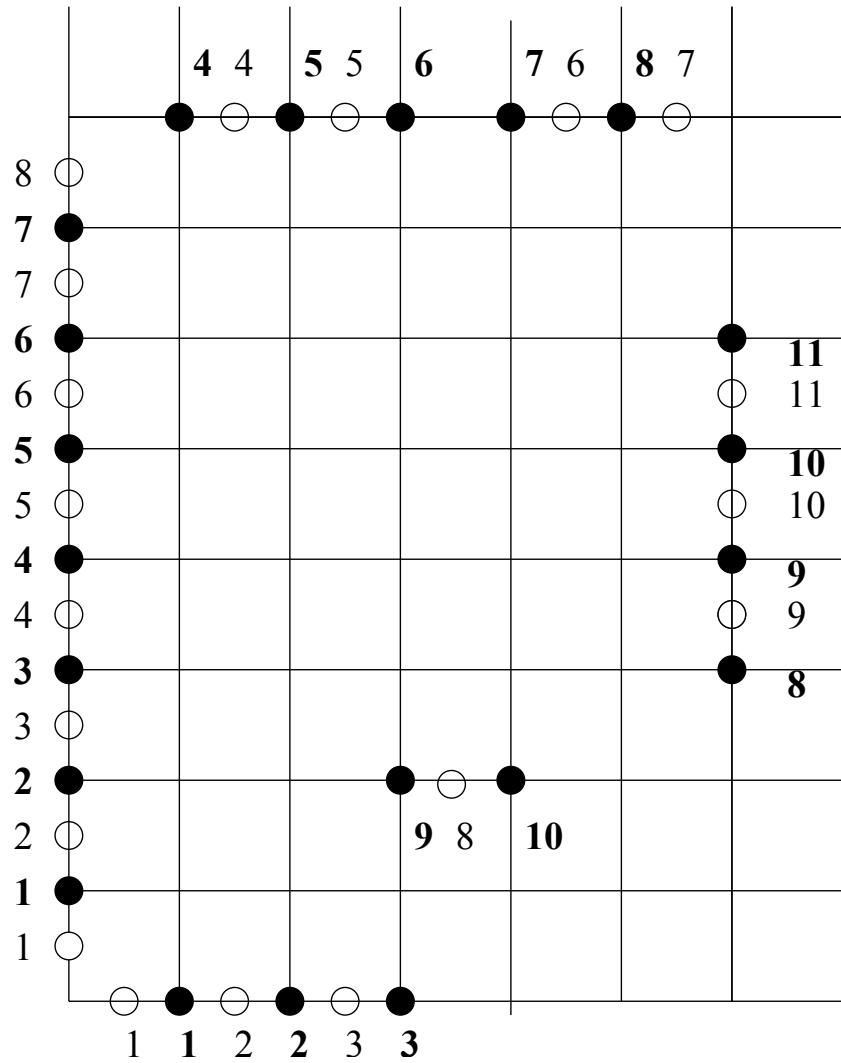


Figure 15.1: Illustration of the indexing for X- and Y-node open boundaries. U- and V-node open boundaries are indicated by empty circles, X- and Y-node open boundaries by solid circles. The corresponding indices are given by numbers in normal font (U- and V-points) or in bold (X- and Y-points).

`delxatc, delxatu, delxatv, delxatuv`

- grid spacing h_2 in meters

`delyatc, delyatu, delyatv, delyatuv`

- σ -grid spacing $\Delta\sigma = h_3/H$

`delsatc, delsatu, delsatv, delsatw, delsatuw, delsatvw`

The last letter(s) in the variable name denote(s) the grid node where the array is defined.

15.1.2.4 pointer arrays

Pointer arrays denote the status of a grid node (dry, wet, open boundary). A specific array is defined for each horizontal nodal type

`nodeatc` status at C-nodes

- 0: inactive cell which can either by a permanently land cell or a sea cell temporarily set to dry by a mask criterium in case a flooding scheme is applied (see Section ??)
- 1: active (wet) sea cell

`nodeatu` Pointers at U-node cell faces

- 0: dry (land) cell face
- 1: coastal boundary
- 2: interior wet U-node
- 3: open sea boundary
- 4: river open boundary

`nodeatv` Pointers at V-node cell faces

- 0: dry (land) cell face
- 1: coastal boundary
- 2: interior wet V-node
- 3: open sea boundary
- 4: river open boundary

`nodeatuv` Pointer at corner (UV) nodes

- 0: at least two surrounding U-nodes or at least two surrounding V-nodes are dry
- 1: interior wet node, i.e. at most one surrounding U-node and at most one surrounding V-node is dry and none of the four surrounding velocity nodes are open boundaries
- 2: X-node open boundary, in which case at least one of the surrounding U-nodes is an open boundary while the other one is either a closed node or open boundary, but the node is not a Y-node open boundary
- 3: Y-node open boundary, in which case at least one of the surrounding V-nodes is an open boundary while the other one is either a closed node or open boundary, but the node is not an X-node open boundary
- 4: the node is both a X- and a Y-node open boundary

nodeatuw Pointer at UW-node cell faces

- 0: dry (land) cell face or bottom cell (1) or surface cell ($\text{nz}+1$)
- 1: coastal boundary
- 2: interior wet UW-node
- 3: open sea boundary
- 4: river open boundary

nodeatvw Pointer at VW-node cell faces

- 0: dry (land) cell face or bottom cell (1) or surface cell ($\text{nz}+1$)
- 1: coastal boundary
- 2: interior wet VW-node
- 3: open sea boundary
- 4: river open boundary

- The values of the pointer arrays are obtained from the bathymetry (zero water depths at land, positive depths at sea) and the positions of the open boundaries as given by the user.
- In the current implementation the arrays are fixed in time. Dynamic masks are foreseen in future versions.
- Local bounds in the horizontal are given by $(1-\text{nhalo}:\text{ncloc}+\text{nhalo}, 1-\text{nhalo}:\text{nrloc}+\text{nhalo})$.
- All arrays, except **nodeatc**, have a vertical dimension of **nz** at U-, V- or UV-nodes and **nz+1** at UW- or VW-nodes. The vertical dimension has been added for the future implementation of structures.

- The bottom and surface values of `nodeatuw`, `nodeatvw` are, by definition, always equal to 1.
- In the current implementation all arrays must be vertically uniform (with exception of the previous restriction for UW- and VW-nodes).

15.2 Interpolation of model arrays at a different node

Since the model uses an Arakawa C-grid, arrays often have to be interpolated from one grid node to another one. The simplest way is to consider uniform interpolation between neighbouring points. The following issues need to be taken into consideration:

- Uniform averaging is no longer applicable (or at least recommended) in case of a curvilinear grid.
- Model grid arrays are undefined at land points. Land mask can be used to eliminate these points from the interpolation.

These points will be addressed in the discussion below.

15.2.1 Interpolation without land flags

The general formula for interpolation a quantity X defined at node “a” to node “b” is

$$X^b(x, y, z) = \frac{\sum_{n=1}^N w_n X^a(x_n, y_n, z_n)}{\sum_{n=1}^N w_n} \quad (15.3)$$

where w_n are grid-dependent weight factors. In case of a uniform interpolation all $w_n = 1/N$ so that

$$X^b = \frac{1}{N} \sum_{n=1}^N X^a(x_n, y_n, z_n) \quad (15.4)$$

The parameter N depends on the number of grid dimensions involved in the interpolation:

- $N=2$: interpolation along a coordinate line (1 direction). In case of non-uniform interpolation the weights factors have dimensions of lengths.
- $N=4$: interpolation within a coordinate surface (2 directions). In case of non-uniform interpolation the weight factors have dimensions of areas.

- N=8: 3-D interpolation (3 directions). In case of non-uniform interpolation the weight factors have dimensions of volumes.

General rules are

- Interpolations within a grid cell are always uniform.
- By default, interpolation involving values at neighbouring cells are uniform. This can be overruled with the switch `iopt_array_interp` or by calling the interpolation routines with the optional `hregular` and `vregular` argument set to `.FALSE.`. The second option always takes precedence over the first.
- The interpolation routines in the program provide an option to exclude land points or open sea boundaries from the interpolation.

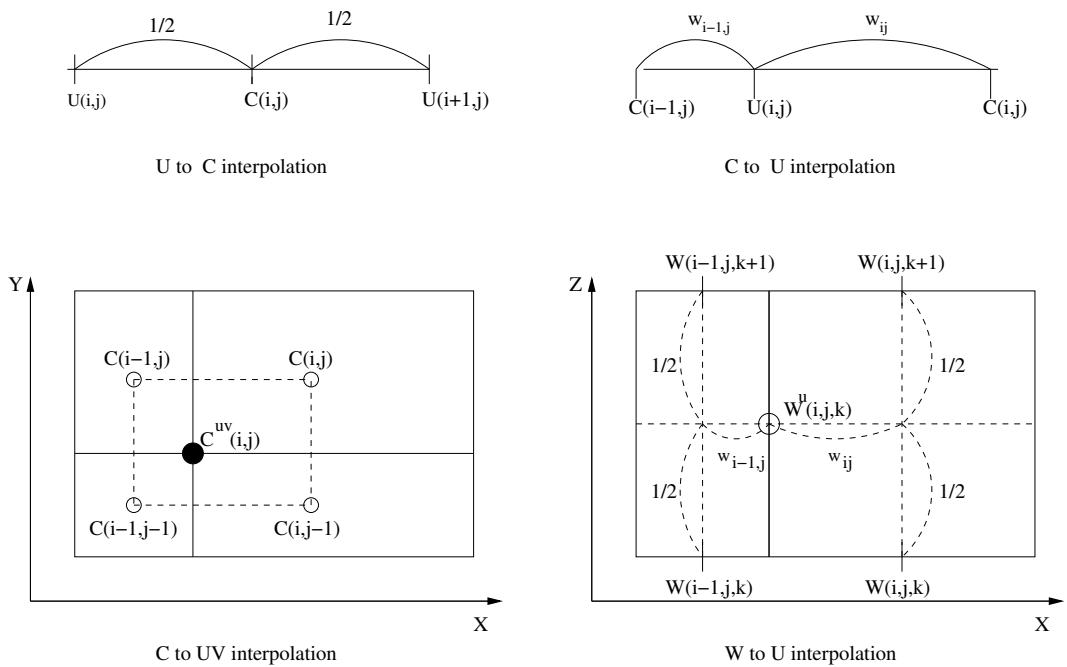


Figure 15.2: Interpolation of model grid arrays at a different node.

Examples

1. interpolation from U- to C-node:

$$X^c(i, j) = \frac{1}{2}(X^u(i, j) + X^u(i + 1, j)) \quad (15.5)$$

2. (non-uniform) interpolation from C- to U-node:

$$\begin{aligned} X^u(i, j) &= \frac{w(i, j)X^c(i-1, j) + w(i-1, j)X^c(i, j)}{w(i-1, j) + w(i, j)} \\ w(i, j) &= \frac{1}{2}h_1^c(i, j) \end{aligned} \quad (15.6)$$

3. (non-uniform) interpolation from C- to UV-node:

$$\begin{aligned} X^{uv}(i, j) &= \left(w(i, j)X^c(i-1, j-1) + w(i-1, j)X^c(i, j-1) \right. \\ &\quad \left. + w(i, j-1)X^c(i-1, j) + w(i-1, j-1)X^c(i, j) \right) \\ &\quad / \left(w(i, j) + w(i-1, j) + w(i, j-1) + w(i-1, j-1) \right) \\ w(i, j) &= \frac{1}{4}h_1^c(i, j)h_2^c(i, j) \end{aligned} \quad (15.7)$$

4. (non-uniform) interpolation from W- to U- node:

$$\begin{aligned} X^u(i, j, k) &= \frac{1}{2} \left(w(i, j)X^w(i-1, j, k) + \right. \\ &\quad w(i-1, j)X^w(i, j, k) + w(i, j)X^w(i-1, j, k+1) \\ &\quad \left. + w(i-1, j)X^w(i, j, k+1) \right) / \left(w(i, j) + w(i-1, j) \right) \\ w(i, j) &= \frac{1}{2}h_1^c(i, j) \end{aligned} \quad (15.8)$$

What type of interpolation needs to be taken ?

- uniform Cartesian grids: uniform (allways)
- uniform spherical: uniform (recommended)
- non-uniform rectangular and curvilinear: user decision (depending on the variation of grid size between neighbouring cells)

15.2.2 Interpolation with land flags

Land flags can be used to eliminate land areas or temporarily inactive cells (in case a mask function is applied as described in Section ??) from the interpolation. The general interpolation formula with land flags becomes

$$X^b(x, y, z) = \frac{\sum_{n=1}^N s_n w_n X^a(x_n, y_n, z_n)}{\sum_{n=1}^N s_n w_n} \quad (15.9)$$

where s_n equals 0 for flagged and 1 for non-flagged grid points. The program foresees the following options

- 0: No flags ($s_n=1$ everywhere).
- 1: Land cells and cell faces at or near coastal boundaries (except open sea boundaries) are flagged.
- 2: Land cells and cell faces at or near coastal boundaries (including open boundaries) are flagged.
- 3: Only cell faces at open sea boundaries are flagged (allowing the inclusion of coastal boundaries in the interpolation).

As an example the formula for array interpolation from C- to UV-nodes becomes

$$\begin{aligned}
 X^{uv}(i, j) &= \left(s(i-1, j-1)w(i, j)X^c(i-1, j-1) \right. \\
 &\quad + s(i, j-1)w(i-1, j)X^c(i, j-1) \\
 &\quad + s(i-1, j)w(i, j-1)X^c(i-1, j) \\
 &\quad \left. + s(i, j)w(i-1, j-1)X^c(i, j) \right) / \\
 &\quad \left(s(i-1, j-1)w(i, j) + s(i, j-1)w(i-1, j) \right. \\
 &\quad \left. + s(i-1, j)w(i, j-1) + s(i, j)w(i-1, j-1) \right) \\
 w(i, j) &= \frac{1}{4}h_1^c(i, j)h_2^c(i, j) \tag{15.10}
 \end{aligned}$$

Note that the flags can be applied at the source node “a” (as given in the equation above) as well as at the destination node “b”.

15.3 Curvilinear, index and relative coordinates

Before proceeding to discuss the interpolation towards or from an external data grid, some more detailed discussion is required how the model’s curvilinear coordinates are related to the grix index system introduced in Section 4.2.1.

Consider the grid layout as shown in Figure 15.3. The index (i,j) of a UV-nodal grid point can be interpreted as the curvilinear coordinates (ξ_1 , ξ_2) with respect to axes in the X- and Y-direction and origin at corner index

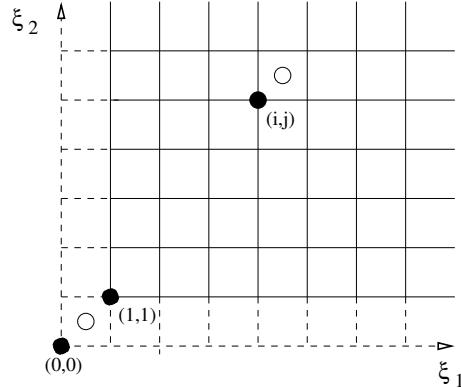


Figure 15.3: Curvilinear versus index coordinates.

(0,0). Within the same frame of reference, the corresponding curvilinear coordinates of the C-point with index (i,j) are $(i+1/2, j+1/2)$. For U- and V-nodal points this becomes $(i,j+1/2)$ and $(i+1/2,j)$.

Instead of taking the origin at the corner point with index (0,0), the origin can be moved to the C-point with index (0,0). In that case, the curvilinear coordinates of a corner point at (i,j) are $(i-1/2, j-1/2)$, while the curvilinear coordinates of a C-grid point are the same as its indices. For U- and V-nodal points this becomes $(i-1/2, j)$ and $(i, j-1/2)$. Similarly, if the origin is taken at the U-node index point (0,0), the curvilinear coordinates of a C-point, U-point, V-point and corner point are respectively $(i+1/2, j)$, (i, j) , $(i+1/2, j-1/2)$ and $(i, j-1/2)$. Finally, taking the origin at the point with V-node index (0,0), the curvilinear coordinates becomes $(i, j+1/2)$ for a C-node, $(i-1/2, j+1/2)$ for a U-node, (i, j) for a V-node and $(i-1/2, j)$ for a corner point.

Relative coordinates are a convenient way to perform interpolation from one to an other. Let (ξ_1, ξ_2) be the (normalised) curvilinear coordinates of an arbitrary point, the corresponding relative coordinates are then (i, j, x, y) where (i, j) are the integer and (x, y) the decimal parts of (ξ_1, ξ_2) . From the previous discussion, the integer coordinates are taken with respect to a certain node (C, U, V, UV). For example, if a corner index system is taken, the relative coordinates of a C-point with index (i,j) with respect to this corner grid are $(i, j, 1/2, 1/2)$.

15.4 Interpolation of a 2-D external data grid at the model grid

15.4.1 General description of the procedure

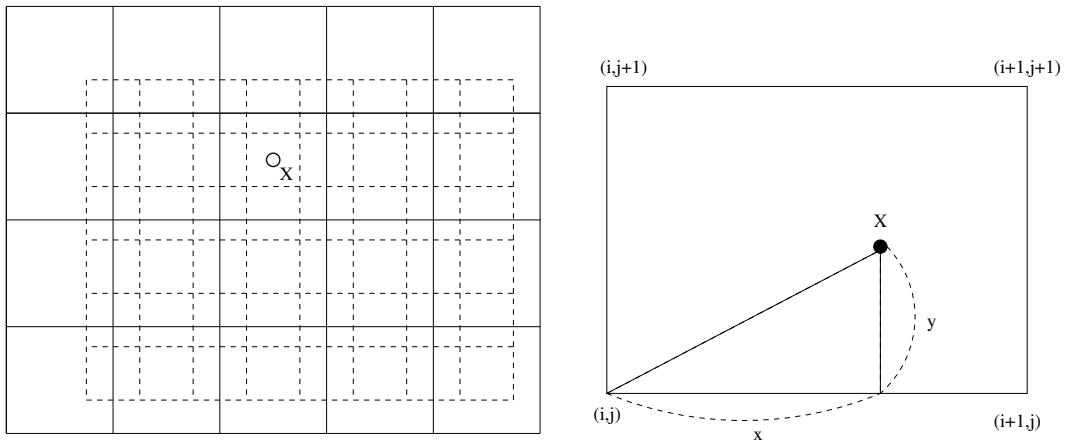


Figure 15.4: Interpolation of external data to the model grid. The solid (dashed) lines represent the external (model) grid.

Assume that the model grid is embedded within an external 2-D data grid as shown in Figure 15.4. The relative coordinates of the C-grid point X with respect to the data grid at its cell corners are (i,j,x,y) where (x,y) are expressed as normalised coordinates (i.e. between 0 and 1). The interpolated value of data 'd' at X is given by

$$d(X) = (1-x)(1-y)d(i,j) + x(1-y)d(i+1,j) + (1-x)yd(i,j+1) + xyd(i+1,j+1) \quad (15.11)$$

where i, j are the indices defined on the data grid.

The following remarks apply

- The model points are principally taken at C-nodes, although the method can be applied to other nodes as well.
- If the external grid is rectangular (uniform or non-uniform), the relative coordinates can be calculated by the program. No algorithm is currently implemented for the determination of the relative coordinates in case of a curvilinear or unstructured data grid.

- The weight factors in (15.11) only depend on the locations of the external grid and do not take account of possible land points, which should be excluded from the interpolation. The following options are available:
 1. All land points are taken into account.
 2. Interpolation is only performed if at least one of the surrounding four points is located at sea.
 3. Interpolation is only performed if all the four surrounding points are at sea.

If the data grid represents meteorological data, no distinction is made between land and sea points.

15.4.2 Implementation

The method can currently be applied for interpolation of surface meteorological data or (satellite) surface temperature data. Future extensions are planned for surface wave data.

As a first step the type of the 2-D external data grid is defined via the following derived type definition:

```
TYPE :: GridParams
  INTEGER :: nhtype, n1dat, n2dat
  REAL :: delxdat, delydat, x0dat, y0dat
END TYPE GridParams
```

where

nhtype Type of the surface data grid.

- 0: single grid point
- 1: uniform rectangular grid
- 2: non-uniform rectangular grid
- 3: non-rectangular (curvilinear or non-structured) grid
- 4: the same as the model grid

n1dat number of grid cells in the X-direction

n2dat number of grid cells in the Y-direction

delxdat grid spacings in the X-direction (m or fractional degrees longitude)
when **nhtype**=1

delydat grid spacings in the Y-direction (m or fractional degrees latitude) when **nhtype**=1
x0dat X-coordinate (Cartesian or longitude) of the lower left corner when **nhtype**=1
y0dat Y-coordinate (Cartesian or latitude) of the lower left corner when **nhtype**=1

Note that **nhtype** has to be supplied always if the value is different from its default one (0). The **n1dat**, **n2dat** attributes have to be given if **nhtype**>0.

All surface grid attributes are stored into the array **surfacegrids**

```
TYPE (GridParams), DIMENSION(MaxGridTypes,2) :: surfacegrids
```

where **MaxGridTypes** is the maximum number of external grids. A key id of the form **igrd_*** is used as index for the first dimension. The following values are currently implemented

- **igrd_meteo**: surface meteorological grid
- **igrd_sst**: sea surface temperature (SST) grid
- **igrd_waves**: surface wave grid
- **igrd_model**: model grid

The last id seems to imply that the model grid itself is considered as an external data grid which is obviously not the case. It has only been implemented as an utility in case the user wants to define a rectangular uniform model grid (see Section ??).

The second index can currently only take the value of 1 which means interpolation to the model grid. The value 2 is intended for interpolation of model data to the data grid which is currently not implemented.

The next step consists in determining the relative coordinates of all model grid points located at sea. Use is made of the derived type definition

```
TYPE :: HRelativeCoords
  INTEGER :: icoord, jcoord
  REAL :: xcoord, ycoord
END TYPE HRelativeCoords
```

For example, in case of surface meteorological data, all relative coordinates are stored in the array

```
TYPE (HRelativeCoords), DIMENSION(ncloc,nrloc) :: meteogrid
```

The relative coordinates itself can be determined by one of the following procedures depending on the value of the **nhtype** attribute

- 0 : The external grid reduces to one data point. No interpolation is required.
- 1 : The external data grid is rectangular and uniform. The relative coordinates are determined by the program.
- 2 : The external data grid is rectangular and non-uniform. The geographical (Cartesian or spherical) coordinates of the data grid are supplied by the user. The relative coordinates are calculated by the program.
- 3 : The external data grid is curvilinear or unstructured. No algorithm is provided by the program. The relative coordinates have to be supplied by the user.
- 4 : The data grid coincides with the model grid. No interpolation is required.

The user procedure for defining the input of 2-D external forcing data, is then the following:

1. The following parameters are defined by the user in **usrdef_mod_params**:
 - Set the appropriate switch which is **iopt_meteo** for meteo input or **iopt_temp_sbc** for SST input.
 - Define the attributes of the external grid(s) in **surfacegrids**.
 - The attributes of the following files have or may be defined:
 - modfiles(io_metgrd,1,1)**: grid locations of the meteorological grid in case **nhtype>1**
 - modfiles(io_sstgrd,1,1)** : grid locations of the SST grid in case **nhtype>1**
 - modfiles(io_wavgrd,1,1)**: grid locations of the surface wave grid in case **nhtype>1**
 - modfiles(io_metsur,1,1)**: meteorological data file
 - modfiles(io_sstsur,1,1)** : SST data file
 - modfiles(io_wavsur,1,1)**: wave data file
2. If **nhtype>0**, define the locations of the external grid. Two options are available depending on the value of **nhtype**:
 - The grid locations are obtained in absolute (geographical) coordinates by calling the user defined routines **usrdef_surface_absgrd**. The program converts the absolute coordinates into relative ones.

- The grid locations are obtained in relative coordinates by calling the user defined routines `usrdef_surface_relgrd`.
3. Define data input by calling the user-defined routines `usrdef_surface_data`.

Note that a `usrdef_*` routine is not called if the corresponding `status` attribute of the associated file is set to 'R' in which case a corresponding `read_*` routine is called where the data are read from a file in standard COHERENS format.

Detailed descriptions of the procedures are given in Chapter ?? and Section 22.2.

15.5 Interpolation of model data at external locations

15.5.1 General description of the procedure

The procedure is similar to the one discussed in the previous section, except that the roles of the external grid and model grid are interchanged. This type of interpolation is currently only implemented for nesting procedures, but may be used in future versions for the development of one- or two-way coupling with meteorological and surface wave models.

Advantage now is that the external data points no longer need to be located on some external (structured or unstructured) "data" grid. The only information needed by the program is the location, i.e. the relative coordinates, of the external locations with respect to the model grid.

However, there are additional complexities which need to be taken into account:

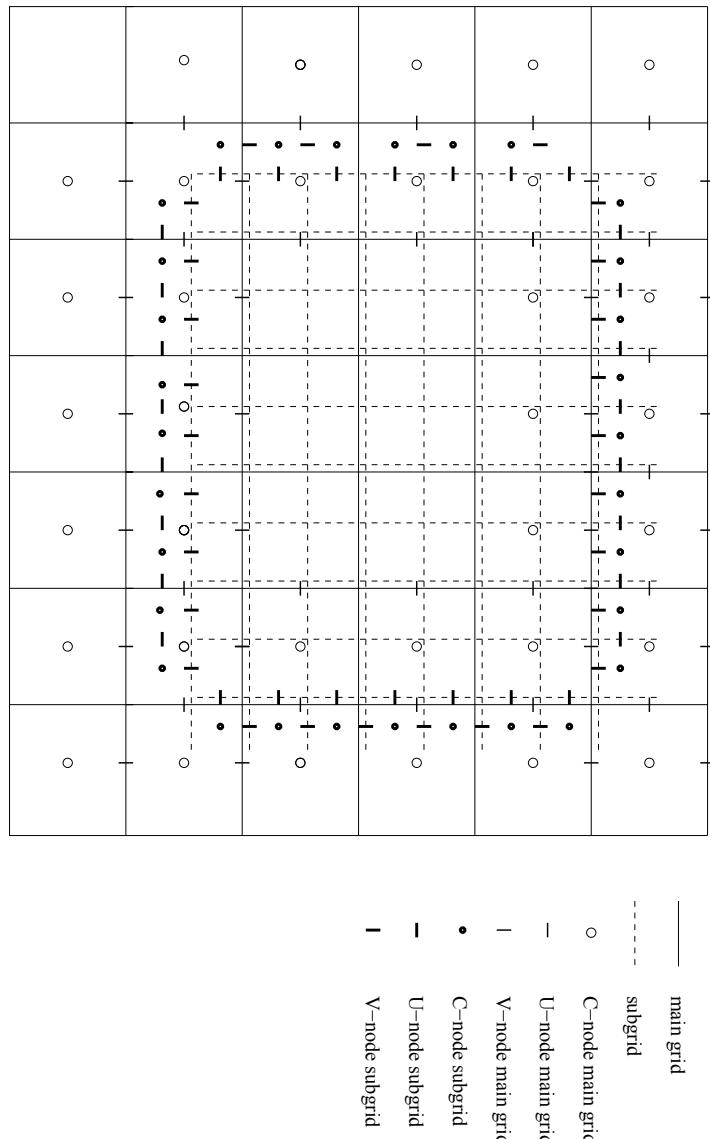
- Since the model uses a staggered C-grid, currents and scalar quantities are calculated at different locations (nodes). This means that the relative coordinates need to be defined in general with respect to different curvilinear coordinate origins ($C(0,0), U(0,0), V(0,0)$), whereby the type of origin depends on the type of variable to be interpolated ($C(0,0)$ for C-node quantities and $U(0,0), V(0,0)$ for vector quantities).
- Model data located on land should be eliminated from the interpolation.
- Interpolation of 3-D quantities requires that the relative coordinates must contain a vertical dimension as well.

- The program allows to define multiple nested sub-grids within the same main grid.

A horizontal layout of the main and sub-grid with the positions of all points used in the interpolation on the two grids is displayed in Figure 15.5. Interpolation can be performed on the 2-D variables U , V , ζ and the 3-D variables δu , δv , T , S . The number and type of data depends on the type of open boundary conditions used by the sub-grid and the dimensions (2-D or 3-D) of the two grids. The only restriction, imposed by the program, is that, although the model permits to use elevation data either at external C-nodes (the solid circles in Figure 15.5 or at the velocity nodes on the open boundaries itself (shown by the horizontal and vertical thick line segments in the figure), only the latter form is allowed when the open boundary data are derived from interpolation. The following five kinds of interpolation then have to be considered in general.

1. Interpolation of U-node model values on the main grid to U-node open boundary points on the sub-grid. This is denoted as U-to-U interpolation and used to obtain “normal” (i.e. normal to the open boundary) boundary conditions at the sub-grid for U and δu).
2. Interpolation of V-node model values on the main grid to V-node open boundary points on the sub-grid. This is denoted as V-to-V interpolation and used to obtain “normal” boundary conditions at the sub-grid for V and δv)
3. Interpolation of C-node model values on the main grid to U-node open boundary points on the sub-grid. This is denoted as C-to-U interpolation and used to obtain open boundary conditions for ζ at the sub-grid open boundaries.
4. Interpolation of C-node model values on the main grid to V-node open boundary points on the sub-grid. This is denoted as C-to-V interpolation and used to obtain open boundary conditions for ζ at the sub-grid open boundaries.
5. Interpolation of C-node model values on the main grid to C-node open boundary points. This is denoted as C-to-C interpolation and used to obtain open boundary conditions scalar quantities (e.g. T and S) at the sub-grid open boundaries.
6. Interpolation of V-node model values on the main grid to V-node points used to obtain “tangential” open boundary conditions for the

Figure 15.5: Illustration of the nesting procedure in the horizontal plane. Solid lines mark the main (coarser) grid, dashed lines the (finer) sub-grids. The empty circles and (light colour) line segments mark respectively the center and velocity node points on the main grid used in the interpolation, the filled circles and (thick) line segments the points on the sub-grid to which interpolation needs to be performed.



cross-stream advective flux at X-node open boundaries on the sub-grid. These points are located outside the sub-grid and one half-grid distance away from the X-node boundaries (see Figure 15.5). To avoid confusion with the previous definitions, these points will be denoted as X-node points and the type of interpolation as V-to-X interpolation.

7. Interpolation of U-node model values on the main grid to U-node points used to obtain “tangential” open boundary conditions for the cross-stream advective flux at Y-node open boundaries on the sub-grid (see Section 12.3.16.2). These points are located outside the sub-grid and one half-grid distance away from the Y-node boundaries (see Figure 15.5). To avoid confusion with the previous definitions, these points will be denoted as Y-node points and the type of interpolation as U-to-Y interpolation.

For each type of interpolation relative coordinates have to be defined with different nodal type and origin.

In case of a 3-D quantity (δu , δv , T , S), the C-to-C, U-to-U, V-to-V, U-to-Y and V-to-X interpolation must extend to the vertical direction as well. This is achieved by adding the vertical relative coordinates k , z to the four horizontal ones (i , j , x , y) where k is the vertical index level of the W-, UW- or VW-node point just below the data point at the which interpolation has to be performed and z is (here) the normalised vertical distance (between 0 and 1) to the W-, UW-, VW-node level. The definition is illustrated in Figure 15.6.

The following remarks are to be given

- Interpolation does not take account of variations in water depth, i.e. the vertical position at which interpolation is to performed, is assumed to be independent in time.
- Vertical interpolation is performed prior to horizontal interpolation. This means that the model data at the four surrounding main grid points are firstly interpolated vertically at the same level as the destination point. If a depth point is below or above the z -level of the lowest or highest grid level, the interpolated value is set to the model value at the lowest or highest level. The profile at the sub-grid location is next obtained by horizontal interpolation of the four profiles using (15.11).

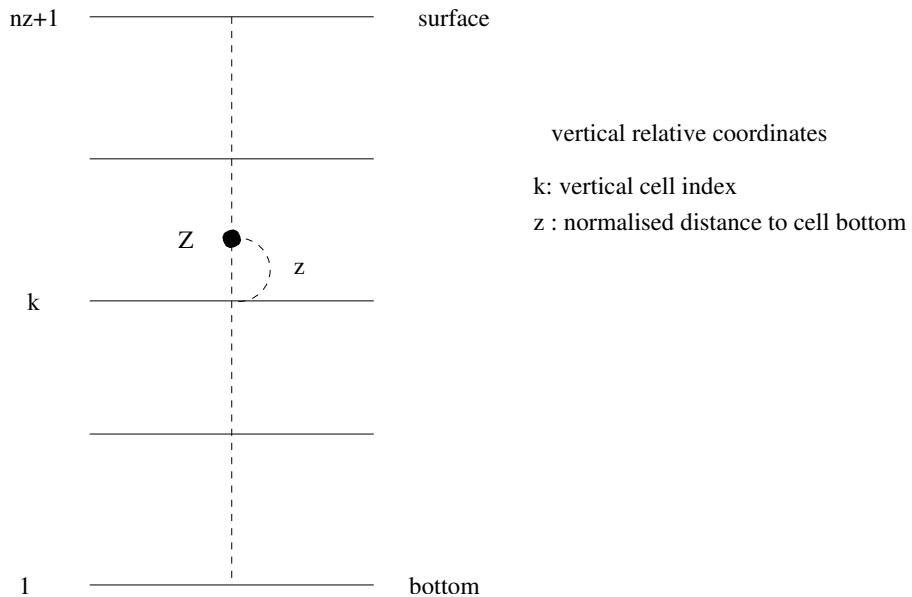


Figure 15.6: Definition of vertical relative coordinates.

15.5.2 Implementation

The program provides the following derive type definition for storing vertical relative coordinates

```
TYPE :: VRelativeCoords
  INTEGER :: kcoord
  REAL :: zcoord
END TYPE VRelativeCoords
```

Details of the code implementation are more complex than the ones in Section 15.4.2. To simplify the discussion below, it is assumed that the program is set up in serial mode.

1. The following parameters are defined by the user in `usrdef_mod_params`:
 - The program switch `iopt_nests` is set to 1.
 - The parameter `nonestsets` is set to the number of nested sub-grids.
 - The attributes of the following files have or may be defined:
 - `modfiles(io_nstspc,1,1)` parameters to be defined in `usrdef_nstgrd_spec`
 - `modfiles(io_nstgrd,1:nonestsets,1)` locations of the sub-grid open boundary points in absolute or relative coordinates for each sub-grid

modfiles(io_2uvnst,1:nonestsets,2) output data files with interpolated values (U-to-U, V-to-V, C-to-U, C-to-V) of U , V , ζ for each sub-grid
 modfiles(io_3uvnst,1:nonestsets,2) output data files with interpolated values (U-to-U, V-to-V) of δu , δv
 modfiles(io_2xynst,1:nonestsets,2) output data files with interpolated values (V-to-X, U-to-Y) of U , V for each sub-grid
 modfiles(io_3xynst,1:nonestsets,2) output data files with interpolated values (V-to-X, U-to-Y) of δu , δv
 modfiles(io_salnst,1:nonestsets,2) output data files with interpolated (C-to-U, C-to-V) values of S
 modfiles(io_tmppnst,1:nonestsets,2) output data files with interpolated (C-to-U, C-to-V) values of T
 modfiles(io_sednst,1:nonestsets,2) output data files with interpolated (C-to-U, C-to-V) values of sediments

Note that output will be written only when the `status` attribute of the respective output data file is set to 'W'.

2. The following arrays are defined in `usrdef_nstgrd_spec`

```

INTEGER, DIMENSION(nonestsets) :: nohnstglbc, nohnstglbu, &
                                 & nohnstglbv, nohnstglbx, nohnstglby
INTEGER, DIMENSION(nonestsets) :: novnst, inst2dtype, nosednst
INTEGER, DIMENSION(maxsedvars,nonestsets) :: instsed
  
```

The first five arrays represent, for each sub-grid, the number of points on the sub-grid at respectively exterior C-nodes, U- and V-nodes at velocity points on the open boundaries, V- and U- at velocity points outside the grid (see Figure 15.5). The sixth gives the number of vertical levels for each sub-grid and `inst2dtype` selects the type of data for 2-D nesting. The arrays `nosednst`, `instsed` inform the program how many and which fractions are used for the nesting of sediment concentrations. For details see Sections 21.3.1 and 23.1.7.

3. The locations of the open boundary locations are determined for each sub-grid. The relative coordinates for each type of interpolation are stored in the following derived type arrays

```

TYPE (HRelativeCoords), DIMENSION(numhnstc) :: hnstctoc
TYPE (HRelativeCoords), DIMENSION(numhnstu) :: hnstutou, hnstctou
TYPE (HRelativeCoords), DIMENSION(numhnstv) :: hnstvtov, hnstctov
TYPE (HRelativeCoords), DIMENSION(numhnstx) :: hnstvtoy
TYPE (HRelativeCoords), DIMENSION(numhnsty) :: hnstutox
TYPE (VRelativeCoords), DIMENSION(2,2,numhnstc,numvnst) :: vnstctoc
TYPE (VRelativeCoords), DIMENSION(2,2,numhnstu,numvnst) :: vnstutou
TYPE (VRelativeCoords), DIMENSION(2,2,numhnstv,numvnst) :: vnstvtov
TYPE (VRelativeCoords), DIMENSION(2,2,numhnstx,numvnst) :: vnstvtox
TYPE (VRelativeCoords), DIMENSION(2,2,numhnsty,numvnst) :: vnstutoy

```

where

numhnstc Total (over all sub-grids) number of C-node sub-grid points
numhnstu Total (over all sub-grids) number of U-node sub-grid points
numhnstv Total (over all sub-grids) number of V-node sub-grid points
numhnstx Total (over all sub-grids) number of X-node sub-grid points
numhnsty Total (over all sub-grids) number of Y-node sub-grid points

These locations are to be defined in the user. Two options are available:

- The user supplies, within `usrdef_nstgrd_abs`, the horizontal positions in absolute (geographical) and the vertical positions, taken as the positive distance from the mean sea level. The relative coordinate arrays `hnstctoc`, ..., `vnstutoy` are determined by the program.
- The user supplies, within `usrdef_nstgrd_rel`, the horizontal positions in relative coordinates and the vertical positions, taken as the positive distance from the mean sea level. The horizontal coordinates are stored in `hnstctoc`, `hnstctou`, `hnstctov`, `hnstutou`, `hnstvtov`, `hnstvtox`, `hnstutoy`, the vertical relative coordinates are calculated by the program and stored in `vnstctoc`, `vnstutou`, `vnstvtov`, `vnstvtox`, `vnstutoy`.

4. The following “index mapping” arrays are defined by the program

```

INTEGER, DIMENSION(numhnstc,noprocs,nonestsets) :: indexnstc
INTEGER, DIMENSION(numhnstu+numhnstv,noprocs,nonestsets) :: indexnstuv
INTEGER, DIMENSION(numhnstx+numhnsty,noprocs,nonestsets) :: indexnstxy

```

where

noprocs number of processes (1 in serial mode)
indexnstc projects the local index of a C-node data point onto a “global” index over all sub-domains
indexnstuv projects the local index of a U-node or V-node data point onto a “global” index over all sub-domains
indexnstxy projects the local index of a “X”-node or “Y”-node data point onto a “global” index over all sub-domains

5. Model data are interpolated and written to the appropriate output file. Time resolution is determined by the **tlms** file attribute.

Note that a **usrdef_*** routine is not called if the corresponding **status** attribute of the associated file is set to ‘R’ in which case a corresponding **read_*** routine is called where the data are read from a file in standard **COHERENS** format.

Detailed descriptions of the procedures are given in Chapter ?? and Section 21.3.

Chapter 16

Aspects of parallelisation

16.1 Basic principles

16.1.1 Implementation of MPI

With the implementation of MPI the work load is divided among a given number N_p of processes. MPI uses non-shared memory, i.e. each process does not share its own memory with the other processes. The global domain is divided in N_p horizontal sub-domains (no decomposition in the vertical). To solve the discretised equations in the horizontal, communications need to be set up between neighbouring domains. Communications are handled by the routines of the MPI (Message Passing Interface) library ([MPI, 1995](#)). Current implementation is Version 1.1.

Advantages are:

- The program runs faster.
- Internal memory is reduced when increasing N_p .
- Land areas can be removed from the global domain by taking a sufficiently large number of processes.

Disadvantages are:

- Increasing N_p decreases the work load per process but increases the number of communications and % of time spent in communications. A maximum efficiency will be attained when $N_p = N_{max}$ depending on the application. The effect is illustrated in Figure 16.1.

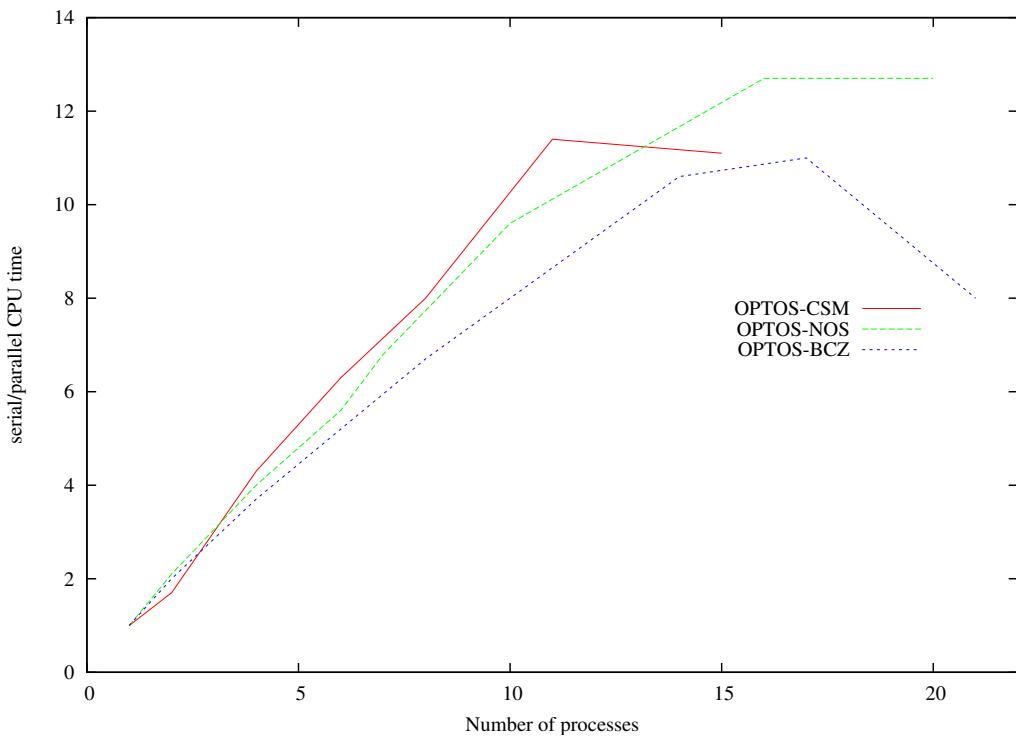


Figure 16.1: CPU efficiency, defined as the CPU time for a serial run divided by the one obtained for the parallel run, as function of the number of processes for the three *optos* test cases.

- Parallelisation is less efficient for 2-D (horizontally averaged) simulations and the mode-splitting scheme (vertical profiles reduce to one point, small time steps).
- Parallel code produces a slight overhead for serial applications.

MPI routine calls are only found in the file *comms_MPI.F90*¹. Each MPI routine call (starting with **MPI**_) has a corresponding alias (starting with **comms**_). This allows a more efficient implementation of future MPI versions.

¹Sole exception is the **MPI_abort** call in *error_routines.F90*. Since MPI is based upon FORTRAN 77, MPI parameters are declared and defined in an “include” file instead of a module file. Instead of a **USE** a **INCLUDE mpif.h** statement has to be made inside the code, contrary to the programming rules stated in Section 13.1.1. This will be removed in future MPI implementations.

16.1.2 Principles of the parallel code

Variables (parameters and arrays) on a parallel grid can either be:

global defined on each sub-domain with the same value

local either defined on each sub-domain but with different values or defined on one or more (but not all) sub-domains

A specific aspect of parallelisation is that different effects are or may be produced by the model code on different domains:

- A variable may be defined on one and non-defined on another domain.
- Arrays can be either allocated, deallocated or undefined on different domains.
- Local arrays usually have different shapes (or may be of size 0) on different domains.
- An IF statements may evaluate as .TRUE. on one or .FALSE. on another sub-domain if the expression contains local variables.

The basic rule for an efficient parallel code is that the work load is (as much as possible) equally spread among the processes. An important (but inevitable) exception is that output operations (except for log and error files) are restricted to one process, called the “master” process. On the other hand, all processes are allowed to read from the same file, avoiding to copy the input data from one particular “reader” process.

16.2 Domain decomposition

16.2.1 Definition

Process domains are arranged as a 2-D “parallel” grid with dimensions `nprocsx` and `nprocsy`. Domains, solely composed of land points, can be excluded from the grid so that $nprocsx \times nprocsy \geq nprocs$ where `nprocs` is the number of “effective” processes. Each domain has a process id number, between 0 and `nprocs-1`, which is assigned by MPI, and two domain grid indices (i,j) where $1 \leq i \leq nprocsx$ and $1 \leq j \leq nprocsy$. Dummy (land) domains are defined with a NULL process id (`MPI_proc_null`). Work load is as much as evenly partitioned between the sub-domains (since the sub-domains are not of the same size). Exception is the master process which is the only one with write access. Reading is performed by all processes (except defined otherwise by the user).

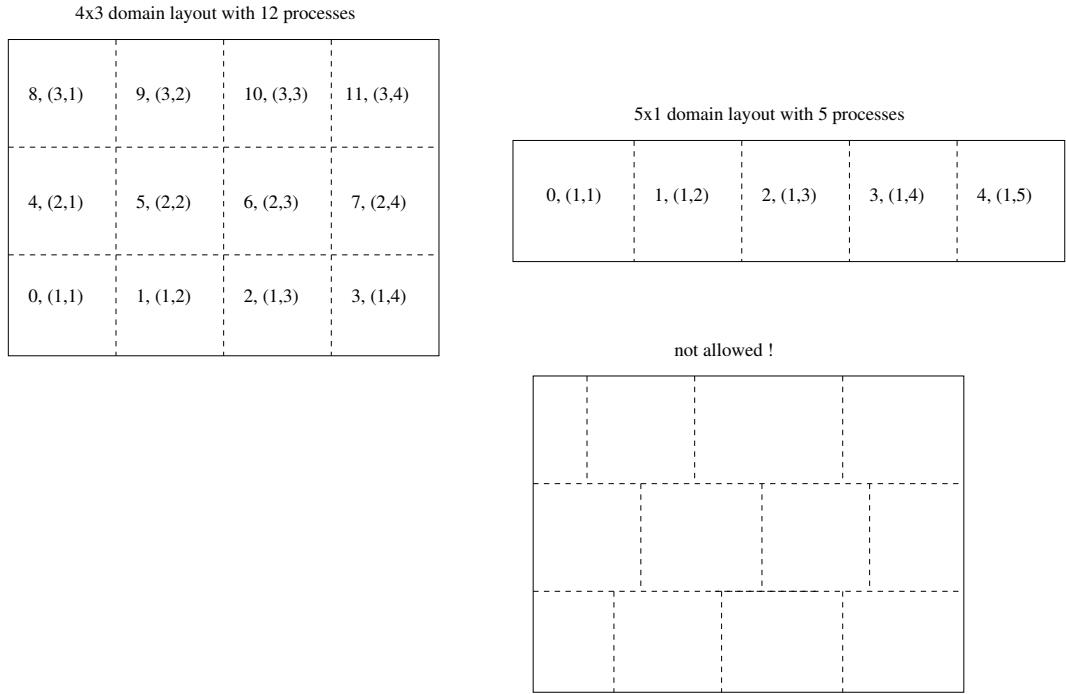


Figure 16.2: Examples of allowed and non-allowed domain decompositions. The first number is the process id, the two numbers in parentheses denote the domain indices.

Some examples of simple domain decompositions are illustrated in Figure 16.2. Figure 16.3 shows an example of a decomposition where part of the land areas have been removed.

16.2.2 Local grid indexing system

The grid indexing system for a local sub-domain, shown in Figure 16.4, is practically the same as the one taken on the global grid (see Figure 4.7). Main differences are:

- The local grid dimensions are now `ncloc` and `nrloc`, instead of `nc` and `nr`.
- The most eastern column and most northern row are no longer composed of dummy land points (except when they are located at the edges of the global computational domain).

The local grid dimensions and indices are related to the global ones through the following definitions:

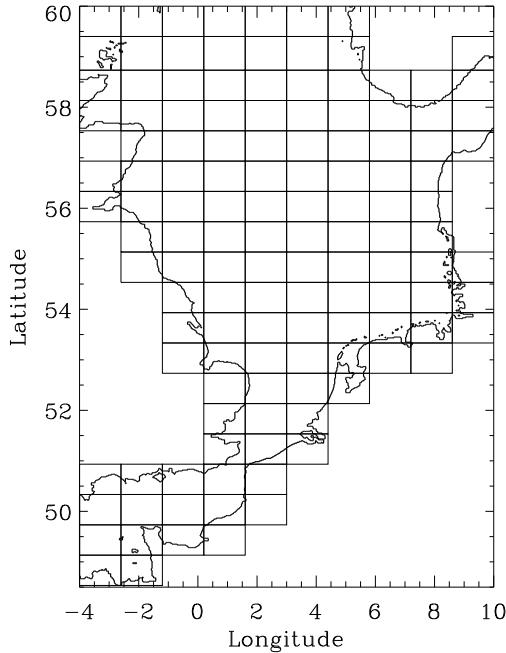


Figure 16.3: Domain decomposition for the North Sea area with 128 processes based on a 10×19 domain grid.

<code>ncloc</code>	X-dimension of the local domain (local variant of <code>nc</code>)
<code>nrloc</code>	Y-dimension of the local domain (local variant of <code>nr</code>)
<code>nc1loc</code>	global X-index of the leftmost column (local)
<code>nc2loc</code>	global X-index of the rightmost column (local)
<code>nr1loc</code>	global Y-index of the lowest row (local)
<code>nr2loc</code>	global Y-index of the highest row (local)
<code>nc1procs(nprocs)</code>	array with the values of <code>nc1loc</code> for each domain (global)
<code>nc2procs(nprocs)</code>	array with the values of <code>nc2loc</code> for each domain (global)
<code>nr1procs(nprocs)</code>	array with the values of <code>nr1loc</code> for each domain (global)
<code>nr2procs(nprocs)</code>	array with the values of <code>nr2loc</code> for each domain (global)

Global and local indices (at any node) are then related by

```
iglb = iloc + nc1loc - 1
jglb = jloc + nr1loc - 1
```

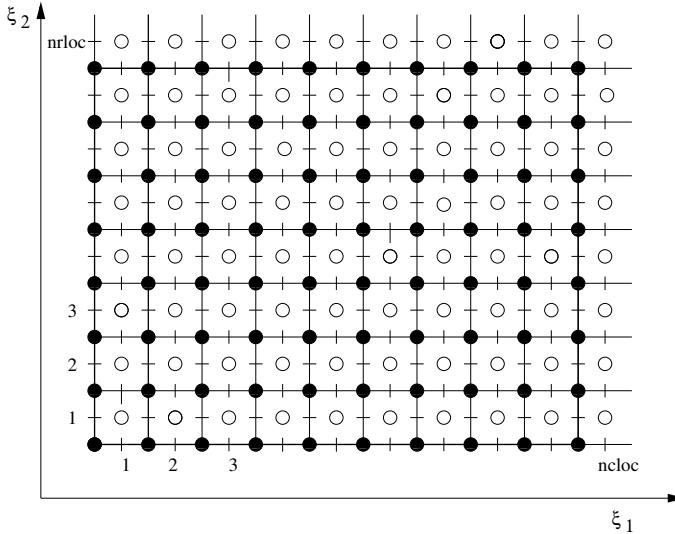


Figure 16.4: Horizontal layout of the computational grid on a local sub-domain. Different nodes are indicated by the same symbols as in Figure 4.7.

Domain decomposition is uniquely defined by these last 4 “process” arrays. Two options are available, depending on the value of the switch `iopt_MPI_partit`:

- 1 Simple decomposition: the user specifies the values of two of the three parameters `nprocs`, `nprocsx`, `nprocsy`. The arrays are defined internally (see Section ?? for further details).
- 2 The arrays are defined externally by the user.

Other (global or local) parameters used in the model for parallel application are:

<code>parallel_set</code>	user-defined parameter to switch on/off the parallel mode (global)
<code>idloc</code>	local process id (local)
<code>iprocloc</code>	local process number = <code>idloc+1</code> (local)
<code>idmaster</code>	Process id of the master process (global). Its value can be changed by the user. Default is 0.
<code>idprocs(nprocs)</code>	vector of local process ids (global)
<code>master</code>	.TRUE. if <code>idloc</code> equals <code>idmaster</code> , .FALSE. otherwise (local)
<code>shared_read</code>	user-defined parameter enabling reading by all processes if .TRUE. (global)

comm_work	MPI communicator composed of all “working” processes (global)
iddomain(0:nprocsx+1,0:nprocsy+1)	process id as function of domain grid indices (global)

16.3 Halos

Numerical discretisations in the horizontal and horizontal averaging may require the availability of values of a model grid array located at grid points within a neighbouring sub-domain. These values are calculated, not inside the domain itself, but within its neighbours. They are obtained by establishing MPI communications between the domain and its neighbours. Each of its neighbours sends an internal section of the array which is received and stored by the domain in one of the local array’s halo sections (see Figure 16.5 and Figure 16.6).

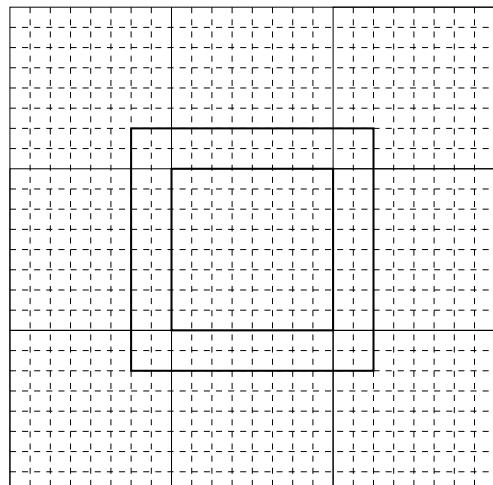


Figure 16.5: Illustration of an halo for an array defined on a local sub-domain. The halo of the sub-domain, located in the inner rectangle, is situated between the inner and outer rectangles.

A halo is created by adding extra columns and rows to the model array. Most arrays have halo sizes which are all equal to the parameter `nhalo = 2`, e.g.

```
uvel(1-nhalo:ncloc+nhalo,1-nhalo:nrloc+nhalo,nz)
```

Other arrays may have smaller halos or asymmetric halos, e.g.

(i-1,j+1) NW	(i,j+1) N	(i+1,j+1) NE
(i-1,j) W	(i,j)	(i+1,j) E
(i-1,j-1) SW	(i,j-1) S	(i+1,j-1) SE

Figure 16.6: Partitioning of a sub-domain halo. Domain indices are given in parentheses.

```
gxcoord(0:ncloc+1,0:nrloc+1), atmpres(0:ncloc,0:nrloc)
```

The West/East/South/North sizes are usually given by a 4-element vector. The halo sizes of the 3 arrays above are respectively

```
(nhalo,nhalo,nhalo,nhalo), (1,1,1,1), (1,0,1,0)
```

Halo sizes of a model array are the same on all sub-domains. The corresponding global arrays (i.e. defined over the whole domain) are declared with the same halo sizes as their local counterparts. Halos of sub-domains which are located at the edge of the (full) computational domain, extend to the outside of the computational domain. These outside points are taken as dummy land points belonging to a dummy outside sub-domain with NULL process id.

16.4 Communications

16.4.1 Send and receive in MPI

A MPI communication consist of a send operation on one process (say A) and a receive operation on another process (say B). COHERENS uses only so-called “blocking” send and receive operations. The implementation in COHERENS is based upon the following communication modes:

1. Synchronous send.

- A sends a message with the data to B.
- B receives the message and the data, sends a message back to A and completes next.
- A receives the message from B and completes.

This is the most robust mode.

2. Bufffered send.

- A sends a message to B, sends the data to a buffer, and completes next.
- B receives the message, retrieves the data from the buffer and completes next.

This mode is useful for transfering a large amount of data.

3. Standard send.

- The operation is performed either synchronously or in buffered mode. The choice is made internally by MPI .
- Less robust than synchronous mode but usually more efficient.

Only the standard and synchronous mode are implemented in COHERENS.

1. Syntax of a standard send

```
SUBROUTINE MPI_send(buf, count, datatype, dest, tag, comm, ierror)
<type>, INTENT(IN), DIMENSION(*) :: buf initial address of the send
                                         buffer
INTEGER, INTENT(IN) :: count      number of elements in the send buffer
INTEGER, INTENT(IN) :: datatype   type of data in the send buffer (e.g.
                                         MPI_REAL for real data)
INTEGER, INTENT(IN) :: dest       rank (process id) of the destination
                                         process
INTEGER, INTENT(IN) :: tag        message tag
INTEGER, INTENT(IN) :: comm       communicator (usually comm_work)
INTEGER, INTENT(OUT) :: ierror    returned error code.
```

2. Syntax of a synchronous send

```
SUBROUTINE MPI_ssенд(buf, count, datatype, dest, tag, comm, ierror)
```

where the arguments have the same meaning as before.

3. Syntax of a receive operation

```
SUBROUTINE MPI_recv(buf, count, datatype, source, tag, comm, &
& status, ierror)
```

`<type>, INTENT(OUT), DIMENSION(*) :: buf` initial address of the receive buffer

`INTEGER, INTENT(IN) :: count` number of elements in the receive buffer

`INTEGER, INTENT(IN) :: datatype` type of the data in the receive buffer
(e.g. `MPI_INT` for integer data)

`INTEGER, INTENT(IN) :: source` rank (process id) of the source process

`INTEGER, INTENT(IN) :: tag` message tag

`INTEGER, INTENT(IN) :: comm` communicator (usually `comm_work`)

`INTEGER, INTENT(OUT), STATUS(MPI_STATUS_SIZE) :: status` return status

`INTEGER, INTENT(OUT) :: ierror` returned error code.

In the COHERENS code, send and receive operations are performed using the `comms_send_*` and `comms_recv_*` routines defined in `comms_MPI.F90`.

16.4.2 Sort of communications

The following sort of communications are used in the program: copy, distribute, combine, combine all, exchange, collect.

1. Copy operations.

- Copies (the same) data from a root (usually the master) process to all other processes.
- The operation involves $N_p - 1$ sends from the root and 1 receive at each other process.
- This is called a “one-to-all” operation and therefore asymmetric.
- Used to copy data read by the root process from a data file (not needed if `shared_read = .TRUE.`).

2. Distribute operations.

- Copies (i.e. distributes) the local parts of a global model array from the root process to all other sub-domains. Each local section may or may not contain the array's local halo parts.
- The operation involves $N_p - 1$ sends from the root and 1 receive at each other process.
- This is called a “one-to-all” operation and therefore asymmetric.
- Used to set up initial conditions on each local domain or to distribute surface data in case the surface data grid coincides with the model grid. Distribute operations are redundant if `shared_read` = `.TRUE.`.

3. Combine operations.

- Copies (i.e. combines) a local array from each sub-domain to a corresponding section of a global array on the root process. Halos are not included.
- The operation involves $N_p - 1$ receives at the root and 1 send from each other process.
- This is called a “all-to-one” operation and therefore asymmetric.
- Various versions are implemented:
 - Combination of local “full” model arrays into a global “full” array.
 - Combination of local subsections of model arrays into a global array.
 - Combination of local irregular (station) data into a global array.
 - Combination of local open boundary arrays into a global boundary array.
- Used for the construction of global arrays, obtained from local model arrays, local regular or irregular sub-arrays. The global arrays are mostly intended for output, the calculation of global array sums, array maxima and minima.

4. Combine-all operations.

- The same as the combine operation except that the global data are made available to all processes.

- The operation involves N_p-1 sends and N_p-1 receives on each process.
- This is called a “all-to-all” operation and therefore symmetric. Note that work load is (about) the same for each sub-domain so that the CPU time for a combine-all is about the same as for a corresponding combine operation.

5. Exchange operations.

- Send sub-sections of local model arrays to a corresponding section within the halo of each neighbouring domain. Receives data from each neighbouring domain and stores these data in one of its own halo sections.
- Since each sub-domain has 8 neighbours (including dummy domains outside the computational domain), the operation requires in general 8 sends and 8 receives on each sub-domain. However, this number can be reduced by specifying the halo sections for which an exchange is needed or when component(s) of the halo size vector is (are) zero.
- The operation is symmetric.
- Exchange operations are an essential part of the parallel code and are mainly for implementation of numerical algorithms for horizontal advection and diffusion, and for horizontal averaging.

6. Collect operations.

- Stores all local arrays into one global array with an extra dimension of size `nprocs`.
- The operation involves N_p-1 sends and N_p-1 receives on each domain.
- This is a “all-to-all” operation and therefore symmetric.
- The operation is not frequently used.

Remarks:

- The root process is by default the master process.
- The communication routines are programmed using the `MPI_send` and `MPI_recv` routines either in standard or synchronous mode.
- There are options to use MPI collective calls instead for some operations: `MPI_bcast` for copy and `MPI_allgather` for collect operations.

- For exchange operations there is an alternative option to use the `MPI_sendrecv` utility routine which combine a send and a receive operation into one call.

16.4.3 Implementation

The main difficulty in programming MPI communications is to prevent so-called dead locks, i.e. a send/receive operation cannot terminate because one of the processes is engaged in another communication which cannot terminate as well and so on. The problem does not arise for the asymmetric “one-to-all” and “all-to-one” operations since each process is engaged in either a send or a receive operation but not both. The problem is more severe for “all-to-all” and exchange operations. Implementation of these two type of operations is discussed below.

16.4.3.1 all-to-all operations

Firstly, an order of operations is defined for each process via a $2N_p \times N_p$ array. The elements are either -1, 0, 1, 2. Taking the example of $N_p = 4$, the array is defined as follows (with easy extension for a general value of N_p):

$$\begin{pmatrix} 0 & 2 & 2 & 2 \\ 1 & 0 & 2 & 2 \\ 1 & 1 & 0 & 2 \\ 1 & 1 & 1 & 0 \\ -1 & 1 & 1 & 1 \\ 2 & -1 & 1 & 1 \\ 2 & 2 & -1 & 1 \\ 2 & 2 & 2 & -1 \end{pmatrix}$$

where each column (except the first) is obtained by shifting the previous one downwards by 1 position. The first column represents the type and order of the communications performed by process 0, the second by process 1 and so on. The numbers present the kind of the operations to be performed:

- 1: Dummy operation. Nothing is done.
- 0: The process “communicates” with itself, the local data are stored directly into the global array.
- 1: Send operation.
- 2: Receive operation.

Secondly, the source or destination of each communication needs to be defined. The process ids are taken from the following vector (of size $2N_p$): $(0, 1, 2, 3, 0, 1, 2, 3)^T$. Replacing the one and twos in the previous array by respectively S and R, and inserting the process ids into each column of the previous array, all communications can be symbolically presented by the matrix

$$\begin{pmatrix} 0 & R0 & R0 & R0 \\ S1 & 0 & R1 & R1 \\ S2 & S2 & 0 & R2 \\ S3 & S3 & S3 & 0 \\ -1 & S0 & S0 & S0 \\ R1 & -1 & S1 & S1 \\ R2 & R2 & -1 & S2 \\ R3 & R3 & R3 & -1 \end{pmatrix}$$

where the first column applies to process 0, the second column to process 1 and so on. The numbers 0 and -1 have the same meaning as before, Si means send to process i and Ri receive from process i . For example, process 2 (third column) performs the following operations : receive from process 0, receive from 1, internal storage, send to 3, send to 0, send to 1, nothing, receive from 3. It can easily be shown that the order of communications, defined in this way, can never produce a dead lock, since for each send/receive the destination/source process will always be available to receive/send the data.

16.4.3.2 exchange operations

Consider as example a 4×4 decomposition which is arranged in a chessboard pattern as shown in Figure 16.7.

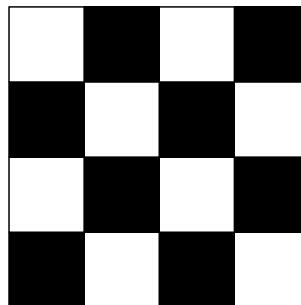


Figure 16.7: Communication pattern for exchange operations.

Each domain has to communicate with its 8 neighbours. Dummy domains (with process id `MPI_proc_null`) are added outside the computational domain

(not shown). Each communication has a specified direction (W, E, S, N, SW, NE, NW, SE, as shown in Figure 16.6). The 8 sends and receives are organised in 16 steps. The first four deal with W/E communications.

- 1: Black sends W, white receives E.
- 2: Black receives E, white sends W.
- 3: Black sends E, white receives W.
- 4: Black receives W, white sends E.
- 5-8: The same now for S/N directions.
- 9-12: The same now for SW/NE directions.
- 13-16: The same now for SE/NW directions.

The actual number of communications depends on the sizes of the halo. For example, an array with halo size (0,2,2,2) has no western halo so that steps 3, 4, 11, 12, 13, 14 become unnecessary. There is also an option foreseen in the exchange call to skip all corner communications (steps 9 to 16).

16.4.3.3 program routines for communications

The following generic communication routines have been implemented in the code. For a discussion of the **FORTRAN** syntax see Section ??.

<code>collect_vars</code>	collect operations
<code>combine_mod</code>	combine operations on full model grid arrays
<code>combine_obc</code>	combine operations on open boundary arrays
<code>combine_stats</code>	combine operations on irregular (stations) arrays
<code>combine_submod</code>	combine operations on sub-sections of full model grid arrays
<code>copy_chars</code>	copy operation on character data
<code>copy_vars</code>	copy operation on numerical data
<code>distribute_mod</code>	distribute operation on model grid arrays
<code>exchange_mod</code>	exchange operation on model grid arrays.

16.5 Local versus global array indexing

Since the parallel decomposition uses non-shared memory, arrays exist only on a local basis. In some cases, one needs to store all these local components into some global array. The question is how to relate a local array element with the corresponding element in the global array. The answer is so-called index mapping, which maps local array indices into the corresponding global ones.

For arrays defined on the model grid, this mapping has a simple form and follows from the definition of the domain decomposition itself. Assume that $(\text{iloc}, \text{jloc})$ are the local and $(\text{iglb}, \text{jglb})$ its corresponding global indices. They are related by

$$\begin{aligned}\text{iglb} &= \text{iloc} + \text{nc1loc} - 1 \\ \text{jglb} &= \text{jloc} + \text{nr1loc} - 1\end{aligned}$$

The solution is less obvious for arrays not indexed by positions on the model grid. Consider the example shown in Figure 16.8. The domain is decomposed in 4 sub-domains.

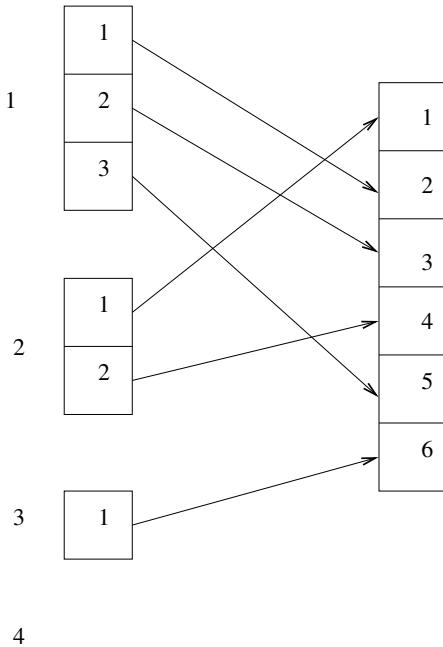


Figure 16.8: Example of index mapping between global and local arrays.

There are 6 data points in total of which 3 are located in the domain with process number 1, 2 in domain 2, 1 in domain 3 and none in domain

4. The 3 elements of domain 1 map into elements (2,3,5) of the global array, the 2 elements of domain 2 into (1,4), the element from domain 3 into global element 6. No mapping is obviously needed for domain 4. The mapping can be programmed as follows

```

INTEGER :: iproc, maxstats, nostatsglb, nostatsloc
INTEGER, DIMENSION(nprocs) :: nostatsprocs
INTEGER, DIMENSION(:, :) :: lstatsprocs

nostatsglb = 6
nostatsprocs = (/3,2,1,0/)
iproc_110: DO iproc = 1,nprocs
  IF (idloc.EQ.idprocs(iproc)) THEN
    nostatsloc = nostatsprocs(iproc)
  ENDIF
ENDDO iproc_110
maxstats = MAXVAL(nostatsprocs)
ALLOCATE (lstatsprocs(nprocs,maxstats))
lstatsprocs(1,1:3) = (/2,3,5/)
lstatsprocs(2,1:2) = (/1,4/)
lstatsprocs(3,1) = 6

```

where `nostatsloc`, `nostatsglb` are the local and global number of data points, `nostatsprocs` an array with the number of data points per process and `lstatsprocs` the array defining the index mapping. If, for example, the data points represent stations used for output, the local data must be combined first into the global array using the index mapping array. The situation is more complex in practice since the number of points per domain are unknown initially. Usually, these points have corresponding geographical positions which allows to determine their distribution over the different sub-domains.

The second example is related to the indexing of open boundary locations. The following definitions are made

<code>nobu</code>	number of global U-open boundary points (global)
<code>nobuloc</code>	number of local U-open boundary points (local)
<code>westobu(nobu)</code>	.TRUE. (.FALSE.) for U-open boundary points at western (eastern) boundaries (global)
<code>iobuloc(nobuloc)</code>	local X-index of U-open boundary points on the local grid (local)
<code>jobuloc(nobuloc)</code>	local Y-index of U-open boundary points on the local grid (local)

<code>indexobu(nobuloc)</code>	global indexes of local U-boundary points (local)
<code>indexobuprocs(nobu,nprocs)</code>	global indexes of local U-boundary points per domain (global)

The parameter `nobuloc` and the last 3 arrays are defined in routine `open_boundary_arrays` (file *Grid_Arrays.F90*). Consider the following code

```
iiloc_110: DO iiloc=1,nobuloc
    i = iobuloc(iiloc); j = jobuloc(iiloc)
    ii = indexobu(iiloc)
    IF (westobu(ii)) THEN
        ....
    ENDIF
ENDDO iiloc_110
```

The `IF` statement determines whether a local open boundary points is located at a western or eastern boundary.

Index mapping is used in the program for local versus global indexing of:

- open boundary arrays
- output arrays of model data on an irregular grid
- the positions of the open boundary points belonging to nested sub-grids

Chapter 17

Structure of the model code

17.1 Source code files

The main code files, located in the **source** directory, can be classified into distinct groups, using their file name.

- Files with suffix *.f90* are FORTRAN source code files, files with *.F90* are FORTRAN files containing C-language code (`#ifdef`) statements.
- Files whose name start with a lower case character and include no underscore character, contain “declaration modules” with declarations of variables and arrays as described in Section 13.1.5.
- Files whose name start with a lower case character and include an underscore character, contain module routines, as described in Section 13.1.5. The routines are often generic and of a general nature.
- Files whose name start with an upper case character (except *Usrdef_** files) are external subprograms with “actual” code.
- Files whose name start with *Usrdef_*, contain routines for user setup. (The ones in the **source** directory are empty and intended for proper compilation only, user-defined versions are defined elsewhere).

Table 17.1: List of declaration module files.

file name	contents
<i>currents.f90</i>	(2-D and 3-D) current arrays
<i>datatype.f90</i>	definitions of all derived types, used in the program
<i>density.f90</i>	density arrays
<i>depths.f90</i>	water depth and surface elevations arrays
<i>diffusion.f90</i>	horizontal and vertical diffusion coefficient arrays
<i>fluxes.f90</i>	arrays of surface and bottom fluxes, drag and exchange coefficients, roughness lengths
<i>grid.f90</i>	model grid arrays
<i>gridpars.f90</i>	model grid and related parameters
<i>iopars.f90</i>	parameters and arrays for all kind of I/O (including user-defined)
<i>meteo.f90</i>	surface meteorological arrays
<i>modids.f90</i>	key id definitions of physical model variables
<i>nestgrids.f90</i>	parameters and arrays for sub-grid nesting applications
<i>obconds.f90</i>	arrays for 2-D and 3-D open boundary conditions, arrays for 1-D surface forcing
<i>optics.f90</i>	optical arrays (including irradiance)
<i>paralpars.f90</i>	parameters and arrays needed for parallel applications
<i>physpars.f90</i>	physical model parameters
<i>relaxation.f90</i>	arrays for applying relaxation conditions
<i>sedarrays.f90</i>	sediment model arrays
<i>sedids.f90</i>	key id definitions of sediment model variables
<i>sedpars.f90</i>	sediment model parameters
<i>sedswitches.f90</i>	sediment model switches
<i>structures.f90</i>	parameters and arrays for the structure and discharge units
<i>switches.f90</i>	physical model switches
<i>syspars.f90</i>	“system” parameter constants
<i>tide.f90</i>	parameters and arrays for tidal applications
<i>timepars.f90</i>	date and time parameters
<i>turbpars.f90</i>	turbulence model constants
<i>turbulence.f90</i>	turbulence model arrays

Table 17.2: List of module routine files.

file name	contents
<i>array_interp.f90</i>	routines for interpolation on the model grid
<i>cf90_routines.F90</i>	library of netCDF routine calls
<i>check_model.f90</i>	routines for checking of user-defined parameters and arrays in the physical model
<i>check_sediments.f90</i>	routines for checking of user-defined parameters and arrays in the sediment model
<i>cif_routines.f90</i>	utility routines used for reading and writing a CIF
<i>comms_MPI.F90</i>	library of MPI routine calls
<i>datatype_init.f90</i>	initialisation of derived type scalar and array variables
<i>default_model.f90</i>	default settings for the physical model
<i>default_sediments.f90</i>	default settings for the sediment model
<i>diagnostic_routines.F90</i>	utility routines calculating terms in the energy equation, total energy, potential energy, enstrophy and vorticity
<i>error_routines.F90</i>	routines performing error checking
<i>fft_library.f90</i>	routines for performing fast Fourier transforms
<i>grid_interp.f90</i>	routines for performing interpolation from and to external grids and locations
<i>grid_routines. f90</i>	utility routines performed on the model grid
<i>inout_paral.f90</i>	routines for preparing input/output in parallel mode
<i>inout_routines.f90</i>	routines for performing input/output in standard format
<i>math_library.f90</i>	library of diverse mathematical routines (e.g. root finder)
<i>model_output.f90</i>	routines for defining standard output data in the physical model
<i>modvars_routines.f90</i>	attributes of variables and files in the physical model
<i>nla_library.F90</i>	linear algebra library
<i>paral_comms.f90</i>	parallel communication library
<i>paral_utilities.f90</i>	utility routines for parallel applications
<i>reset_model.F90</i>	reset setup parameters and arrays in the physical model defined by the user
<i>reset_sediments.f90</i>	reset setup parameters and arrays in the sediment model defined by the user
<i>rng_library.f90</i>	random generator library
<i>sediment_output.f90</i>	routines for defining standard output data in the sediment model
<i>sedvars_routines.f90</i>	attributes of variables and files in the sediment model
<i>time_routines.f90</i>	(calendar) date and time utility routines
<i>turbulence_routines.F90</i>	routines used by the turbulence subprogram
<i>utility_routines.f90</i>	various utility routines

Table 17.3: List of files with external procedures.

file name	contents
<i>Advection_Terms.F90</i>	advective terms in the transport equations
<i>Allocate_Sediment_Arrays.f90</i>	allocate/deallocate variables with a global scope for the sediment model
<i>Allocate_Sediment_Arrays.f90</i>	allocate/deallocate variables with a global scope for the sediment model
<i>Bottom_Fluxes.f90</i>	bottom drag coefficient and shear stress
<i>Coherens_Program.f90</i>	COHERENS main program
<i>Corrector_Terms.F90</i>	corrector terms in the transport equations
<i>Density_Equations.F90</i>	salinity and temperature equations, optical module, equation of state, baroclinic pressure gradient
<i>Diffusion_Coefficients.F90</i>	horizontal and vertical diffusion coefficients
<i>Diffusion_Terms.F90</i>	diffusion terms in the transport equations
<i>Grid_Arrays.F90</i>	model grid parameters and arrays (grid spacings, pointer arrays, open boundary locations, water depths)
<i>Harmonic_Analysis.f90</i>	harmonic analysis
<i>Hydrodynamic_Equations.F90</i>	hydrodynamic equations (currents, 2-D mode, elevations)
<i>Model_Finalisation.f90</i>	finalise physical model
<i>Model_Initialisation.F90</i>	initialise physical model
<i>Model_Parameters.f90</i>	read/write a CIF for the physical model
<i>Nested_Grids.F90</i>	sub-grid nesting
<i>Open_Boundary_Conditions.f90</i>	apply open boundary conditions
<i>Open_Boundary_Data_2D.f90</i>	define 2-D open boundary conditions and update data
<i>Open_Boundary_Data_Prof.f90</i>	define 3-D open boundary conditions and update data
<i>Parallel_Initialisation.f90</i>	initialise parallel mode (parameters, domain decomposition, ...)
<i>Relaxation_Zones.f90</i>	define and apply relaxation conditions
<i>Sediment_Bottom_Fluxes.F90</i>	near bed boundary conditions, (skin) shear stress and roughness length, and critical shear stress in the sediment model
<i>Sediment_Density_Equations.F90</i>	sediment contributions in the calculations within the physical model involving density
<i>Sediment_Equations.F90</i>	COHERENS sediment module (main part)

(Continued)

Table 17.3: *Continued*

<i>Sediment_Finalisation.f90</i>	finalise sediment model
<i>Sediment_Initialisation.F90</i>	initialise sediment model
<i>Sediment_Parameters.f90</i>	read/write a CIF for the sediment model
<i>Structures_Model.f90</i>	structure (dry cells, thin dams, weirs, barriers) and discharge model units
<i>Surface_Boundary_Data_1D.f90</i>	define and apply surface forcing conditions (slope and elevation) for the water column mode
<i>Surface_Data.f90</i>	update 2-D external forcing data
<i>Surface_Fluxes.F90</i>	surface fluxes
<i>Surface_Grids.f90</i>	define 2-D external grids
<i>Tidal_Forcing.F90</i>	astronomical argument, nodal factors, tidal force
<i>Time_Averages.f90</i>	time-averaged output
<i>Time_Series.f90</i>	time series output
<i>Transport_Equations.F90</i>	solve transport equations
<i>Turbulence_Equations.F90</i>	turbulence models

Table 17.4: List of user-defined routine files.

file name	contents
<i>Usrdef_Harmonic_Analysis.f90</i>	parameters and data for harmonic analysis and output
<i>Usrdef_Model.f90</i>	“basic” model setup (model parameters, bathymetry, domain decomposition, initial conditions, open boundary conditions)
<i>Usrdef_Nested_Grids.f90</i>	setup of sub-grids for nesting
<i>Usrdef_Output.f90</i>	output completely specified by the user
<i>Usrdef_Sediment.f90</i>	setup of the sediment model
<i>Usrdef_Surface_Data.f90</i>	definition of external 2-D grids and update of 2-D external data
<i>Usrdef_Time_Averages.f90</i>	parameters and data for time-averaged output
<i>Usrdef_Time_Series.f90</i>	parameters and arrays for time series output

17.2 Structure diagrams

17.2.1 General structure

The general structure of the program is given in Figure 17.1. The program

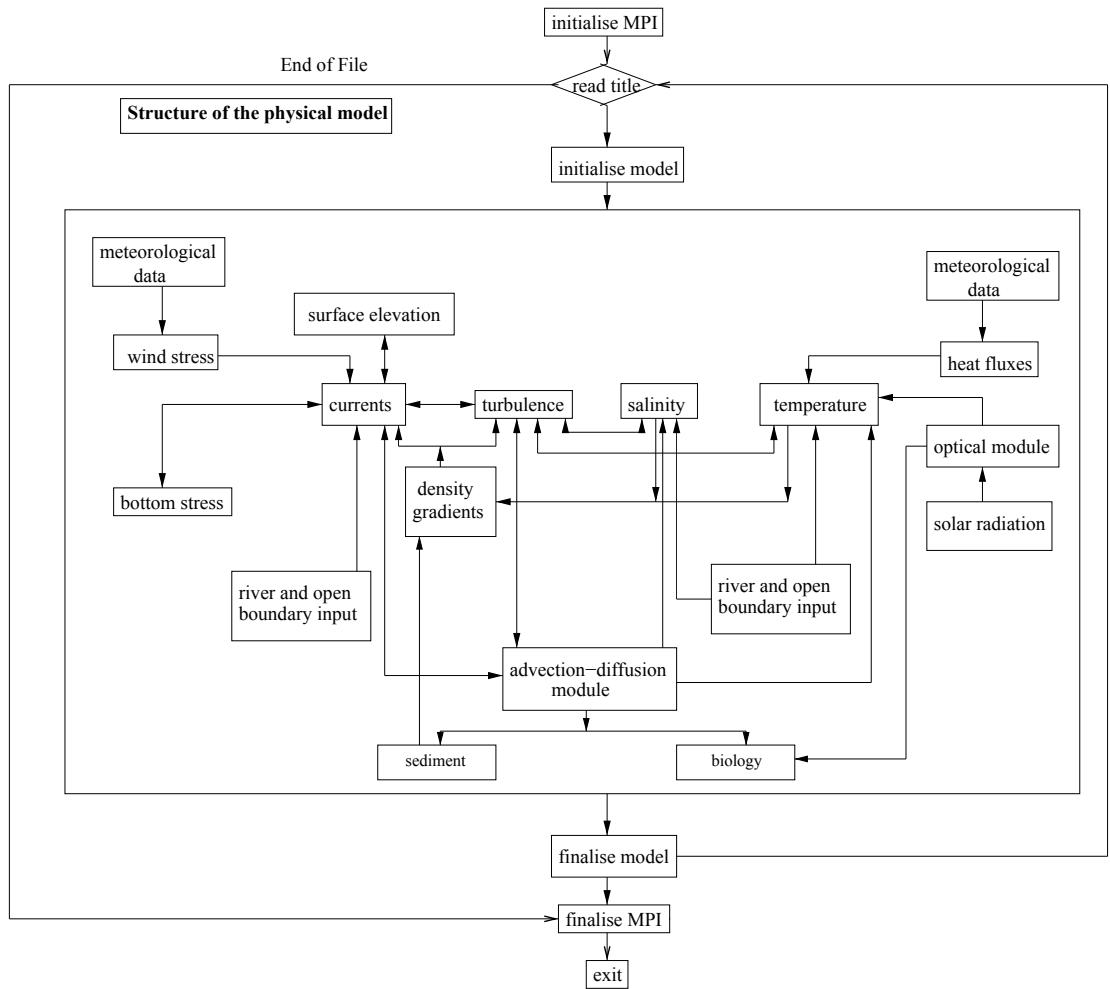


Figure 17.1: General structure of COHERENS.

contains two major loops.

- The first (outer) loop is contained within the large (semi-)rectangle. Each cycle corresponds to a new simulation, initiated by reading a next input line from the file `defruns`. This is further discussed in Section 18.3.
 - The second (inner) loop within the smaller rectangle denotes the time-stepping.

Each simulation is composed of three parts: initialisation, time-stepping (where the actual calculations are performed) and finalisation. Details are given below. Initialisation and finalisation of MPI are the only procedures outside the outer loop. This means that, although multiple simulations can be performed within one run, they must all be conducted either in serial or in parallel mode.

The advection-diffusion module forms the central “core” part of the time-loop section. The module is coded in a generic way and solves the advective-diffusive transport equations of any scalar variable (temperature, salinity, sediment concentrations, biological state variables, ...), 3-D and 2-D currents, and turbulence tranport equations. The inputs for the routine are prepared in separate routines for each variable (temperature, salinity, 3-D currents, 2-D currents, sediments, turbulent energy, ...) and are composed of source terms, open boundary and surface forcing data. The boundary data are obtained via general routine calls. For example, there is one program module (discussed in Section 17.2.4 below) dealing with reading and updating of open boundary profile data for any 3-D quantity.

17.2.2 Initialisation procedures

The initialisation procedures are schematically presented in Figure 17.2. Except for user output, all initialisation routines are called from `initialise_model`.

The first task to be performed by the program is the definition of all parameters and arrays needed to setup the application. This is organised in different sections.

1. Model parameters:

- model switches
- date and time parameters (start and end date, time steps)
- model grid (dimensions, resolution, number of open sea and river boundaries)
- various parameters like number and type of tidal constituents, number of nested sub-grids, ...
- parameters for setting up the model in parallel mode
- physical model parameters
- numerical model parameters
- parameters for the turbulence sub-module(s)
- attributes of external 2-D grids

- attributes of the forcing files
- parameters to define the type and form of monitoring
- parameters and switches for the sediment transport module

For details see Chapter ??.

2. Model grid and bathymetry:

- coordinates of the model grid
- bathymetry
- location of open boundaries

For details see Sections 20.1.

3. Domain decomposition (parallel mode, see Section 20.2). Once the model grid and domain decomposition have been defined, memory is allocated to all model grid arrays and a series of additional arrays (grid spacings, pointer arrays, ...) are defined.
4. Initial conditions (see Section 5.11 for the physics and 23.1.5 for the sediments).
5. Definition of the areas for application of the open boundary relaxation scheme (Section ??).
6. Locations of nested sub-grid(s) (see Section 21.3).
7. Positions of external 2-D data grids (Section 22.2).
8. Type and form of open boundary conditions for the 2-D mode (Section 21.1.1), baroclinic currents and all 3-D scalars (Section 21.2.1).
9. Initialise surface and bottom fluxes.
10. Parameters for setting up user defined output (see Chapter 27).

Open boundary and surface forcing data are usually given as time series. If the first data time coincides with the initial time, the data file is opened and the first and (eventually second) time records are read from the file during the initialisation phase of the program. This has the further advantage that error checking can be performed (existence of the file, data formats, ...) before the program enters the time loop.

Model parameters and switches for the different mofel compartments (physics, sediments) can be defined either in a `usrdef_` routine or by reading from a Central Input File (CIF).

Each section of the initialisation contains (or may contain) the following sub-tasks:

1. Defaults values are given to several model parameters and arrays. In many cases, these defaults are meaningfull and should be maintained. In other case, they are not meaningfull and must be replaced by the user. The advantage of such procedure is a more efficient error checking.
2. Values are (re)-defined. Two methods are available:
 - The definitions are programmed by the user in a `usrdef_` routine. Options are foreseen in the program to write these definitions to an external file in COHERENS standard format.
 - All values are obtained as input from an external file in COHERENS standard format.
3. Depending on the definitions given by the user, parameters are reset from their default values.
4. The setup of the model (definitions of model parameters and arrays) are checked for errors. If errors are detected, appropriate messages are written to an `errlog` file and the program aborts.

17.2.3 Time loop

Figure 17.3 shows a diagram of the time loop. The order of routine calls is in line with the solution procedure described in Section 12.7.

The routines where the 2-D mode, 3-D current, temperature, salinity and sediment transport equations are solved, are schematically presented in Figures 17.4–17.8. Each routines is composed of an initialisation section, a main part where the variable(s) is (are) updated and a finalisation section.

initialisation Actions performed during the initialisation phase (time $t=0$): allocation of local arrays, definition of open boundary conditions, opening of data files and reading of first time records.

- The initialisation of the 2-D mode is actually perfomed in routines `update_2dcbc_data` called from `current_2d` and `define_2dcbc_spec` called from `update_2dcbc_data`.
- Open boundary conditions for 3-D currents are defined in `current_cor`.

- If the temperature equation is forced with SST, the SST grid and data are defined and initialised in `temperature_equation`.

main section

- update open boundary data
- apply open boundary conditions
- apply surface and bottom boundary conditions (3-D variables only)
- calculate source terms
- solve the transport equations by calling the appropriate transport routine
- exchange array sections with neighbouring sub-domains (parallel mode only)
- write interpolated data for nested sub-grids (if requested)

finalisation

Deallocate arrays at the current or final time step.

The following remarks are to be given:

- Surface elevations are updated in `current_2d` before the depth-integrated transports.
- Open boundary conditions are applied in `current_2d` after solving the 2-D depth-integrated momentum equations.
- The 3-D current calculations are split over two routines called at different (predictor and corrector) time steps: surface and bottom boundary conditions are applied, source terms calculated and transport equations solved in `current_pred`; open boundary conditions and corrector step are applied, vertical currents calculated and nested output written in `current_corr`.
- Besides routines for solving the transport equations for all sediment fractions, the sediment model provides separate routines for update of the bed or total load transport.
- Meteorological forcing data are defined by a separate call to `meteo_input` from the main program.

The update of a 2-D or 3-D quantity by an advection-diffusion type equation is performed in one of the `transport_at_*` routines, which integrates the model equations in time. Exceptions are surface elevation and vertical current which are obtained from the 2-D and 3-D continuity equations. The procedures closely follow the numerical descriptions given in Chapter 12 so that no diagrams need to be given here.

17.2.4 Open boundary and surface forcing data input

The procedure for applying open boundary conditions for the 2-D mode is summarised in Figure 17.9:

1. The routine `update_2dabc_data` is called from `current_2d`:
 - At the initial time the routine `define_2d_abc_spec` is called where
 - A series of arrays to be specified by the user, are allocated.
 - The arrays are defined by calling either the user-defined routine `usrdef_2dabc_spec` or as input from a standard COHERENS file by calling `read_2dabc_spec`. If requested, the arrays are written to a standard file by calling `write_2dabc_spec`.
 - Error checking is performed.
 - If there are external data files, it is checked first for each data file, whether the data are still up to date, which means that the last date and time for which data are available is later than the current one. If this is not the case (such as at the initial time), `define_2d_abc_data` is called where:
 - New data are obtained by calling either the user-defined routine `usrdef_2dabc_data` or as input from a standard COHERENS file by calling `read_2dabc_data`. If requested, the arrays are written to a standard file by calling `write_2dabc_data`.
 - If an end of file condition occurs, further action is determined by the `endfile` attribute. This is further discussed in Section ??.
 - The new data (if any), representing the ψ_0^e term in equation (5.333), are stored in the appropriate open boundary arrays and interpolated in time (if requested).
 - Harmonic tidal expansions are evaluated. If needed (which is usually the case at the initial time), astronomical arguments and nodal factors are calculated by calling `astro_params`. The harmonic terms are added to the data values.
2. The open boundary conditions are applied by calling `open_boundary_conds_2d`.

User-defined setup for 2-D open boundary conditions is further discussed in Section ??.

The procedure for applying open boundary conditions for the 3-D mode is given in Figure 17.10. The code is written in a generic form so that the routines can be used for any 3-D quantity (currents, temperature, . . .).

1. At the initial time the routine `define_profobc_spec` is called from the “main” routine (`current_corr`, `temperature_equation`, ...):
 - A series of arrays to be specified by the user, are allocated.
 - The arrays are defined by calling either the user-defined routine `usrdef_profobc_spec` or as input from a standard COHERENS file by calling `read_profobc_spec`. If requested, the arrays are written to a standard file by calling `write_profobc_spec`.
 - Error checking is performed.
2. The routine `update_profobc_data` is called where:
 - It is checked first for each data file, whether the data are still up to date, which means that the last date and time for which data are available is later than the current one. If this is not the case (such as at the initial time), `define_profobc_data` is called where:
 - New data are obtained by calling either the user-defined routine `usrdef_profobc_data` or as input from a standard COHERENS file by calling `read_profobc_data`. If requested, the arrays are written to a standard file by calling `write_profobc_data`.
 - If an end of file condition occurs, further action is determined by the `endfile` attribute. This is further discussed in Section ??.
 - The new data (if any) are stored in the appropriate open boundary profile arrays and interpolated in time (if requested).
 - If any of the interpolating values has a flagged value, the interpolated open boundary profile data value will be flagged as well. A flagged value at a certain vertical level within a vertical profile means that a zero gradient condition will be applied at that particular level.
3. The open boundary conditions are applied by calling `open_boundary_conds_3d` for baroclinic currents or `open_boundary_conds_prof` for scalars.

User-defined setup for baroclinic open boundary conditions is further discussed in Section ??.

The application of 2-D external data requires firstly the definition of the data grid, which is implemented as follows (no diagram shown):

1. A derived type “grid” array is created by allocation in the “main” routine (`meteo_input` for the meteo, `temperature_equation` for the SST

grid or `wave_input` for surface waves), for storing the relative coordinates of the data grid with respect to the model grid (see Section 13.1.4).

2. The grid is defined by calling `define_surface_input_grid`. Definition depends on the value `nhtype` grid attribute (see Section 15.4.2):
 - 0: No grid needs to be defined
 - 1: The grid is uniform rectangular and is defined by calling `construct_regular_grid`.
 - 2: The grid is non-uniform rectangular. Coordinate arrays are obtained by calling either the user-defined routine `usrdef_surface_absgrd` or as input from a standard COHERENS file by calling `read_surface_absgrd`. If requested, the coordinates are written to a standard file by calling `write_surface_absgrd`. The relative coordinates are obtained by calling `model_to_data_coords`.
 - 3: The grid is non-uniform and non-rectangular. The relative coordinate arrays are directly obtained by calling either the user-defined routine `usrdef_surface_relgrd` or as input from a standard COHERENS file by calling `read_surface_relgrd`. If requested, the coordinates are written to a standard file by calling `write_surface_relgrd`.
 - 4: The grid coincides with the model grid and does not need to be defined here.

Setup of 2-D data grids is discussed in Sections 22.2.1–22.2.2.

The procedure for the input of forcing data from a 2-D external data grid is shown in Figure 17.11. The code is written in a generic form so that the routines can be used for meteorological, SST, wave, ... data. The routine `update_surface_data` is called from the “main” routine (`meteo_input` for the meteo or `temperature_equation` for the SST grid):

1. It is checked first for each data file, whether the data are still up to date, which means that the last date and time for which data are available is later than the current one. If this is not the case (such as at the initial time), `define_surface_data` is called where:
 - New data are obtained by calling either the user-defined routine `usrdef_surface_data` or as input from a standard COHERENS file by calling `read_surface_data`. If requested, the arrays are written to a standard file by calling `write_surface_data`.
 - If an end of file condition occurs, further action is determined by the `endfile` attribute. This is further discussed in Section ??.

2. The data are interpolated in time.
3. If any of the interpolating values has a flagged value, the interpolated open boundary profile data value will be flagged as well. In case of SST data, the flagged value is replaced by the modelled temperature at the highest level. There is currently no procedure foreseen for flagged meteorological data values.
4. The data are interpolated in space by calling `intpol_data_to_model_2d` if $0 < \text{nhtype} < 4$.

17.2.5 Finalisation procedures

After termination of the time loop the simulation is finalised as follows:

- All files, which are still open with exception of monitoring files, are closed.
- A timer report is written on request.
- Model arrays, which are still allocated, are deallocated.
- All monitoring files are closed.
- A new input line is read from the `defruns` file. If there is an end of file condition, the program checks whether MPI was switched on at the start, finalises MPI if needed and terminates. Otherwise, a new simulation starts and all model parameters and arrays are reset to their default values as part of the re-initialisation procedures discussed in Section 17.2.2.

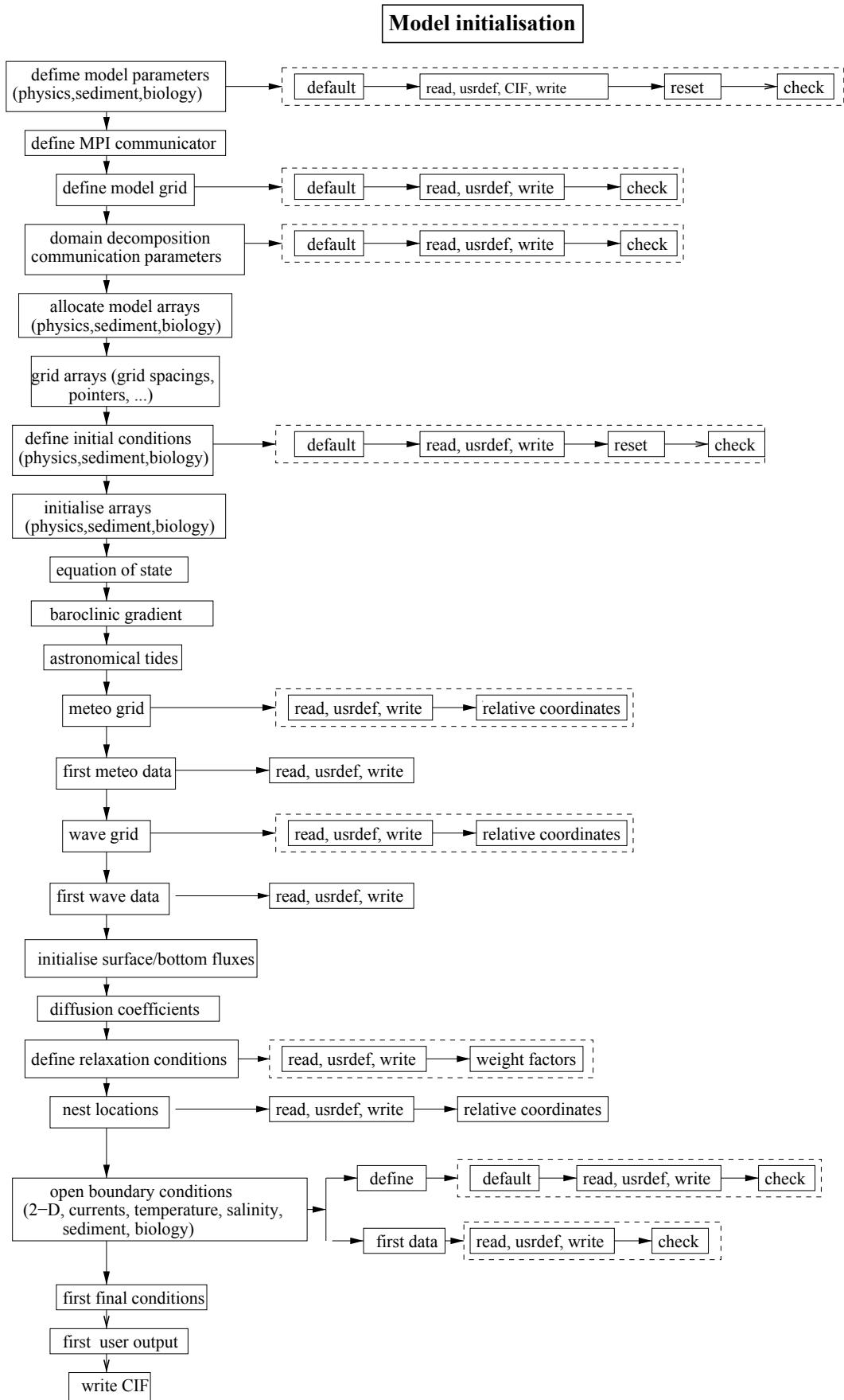


Figure 17.2: Schematic diagram of all initialisation procedures.

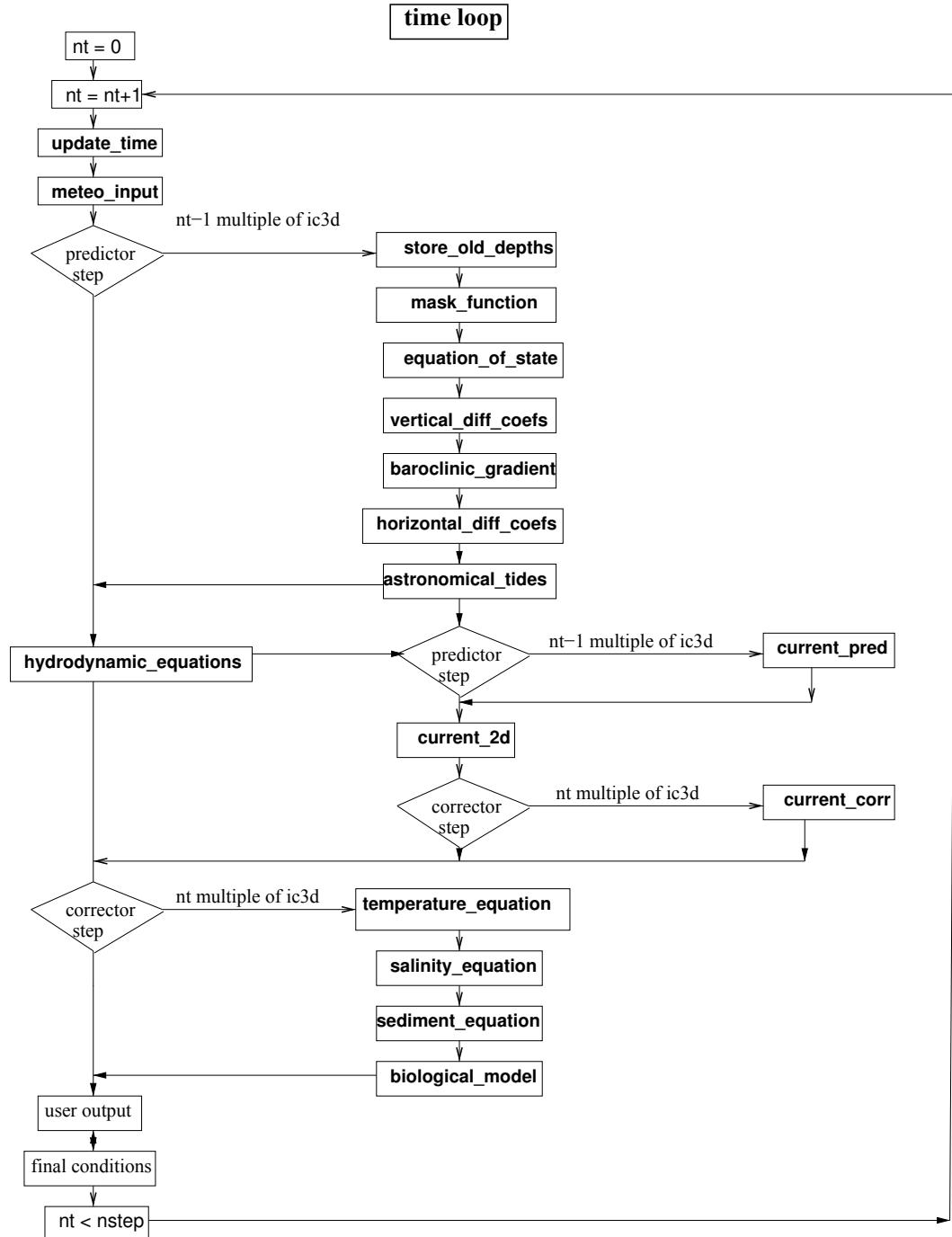


Figure 17.3: Structure diagram of the time loop.

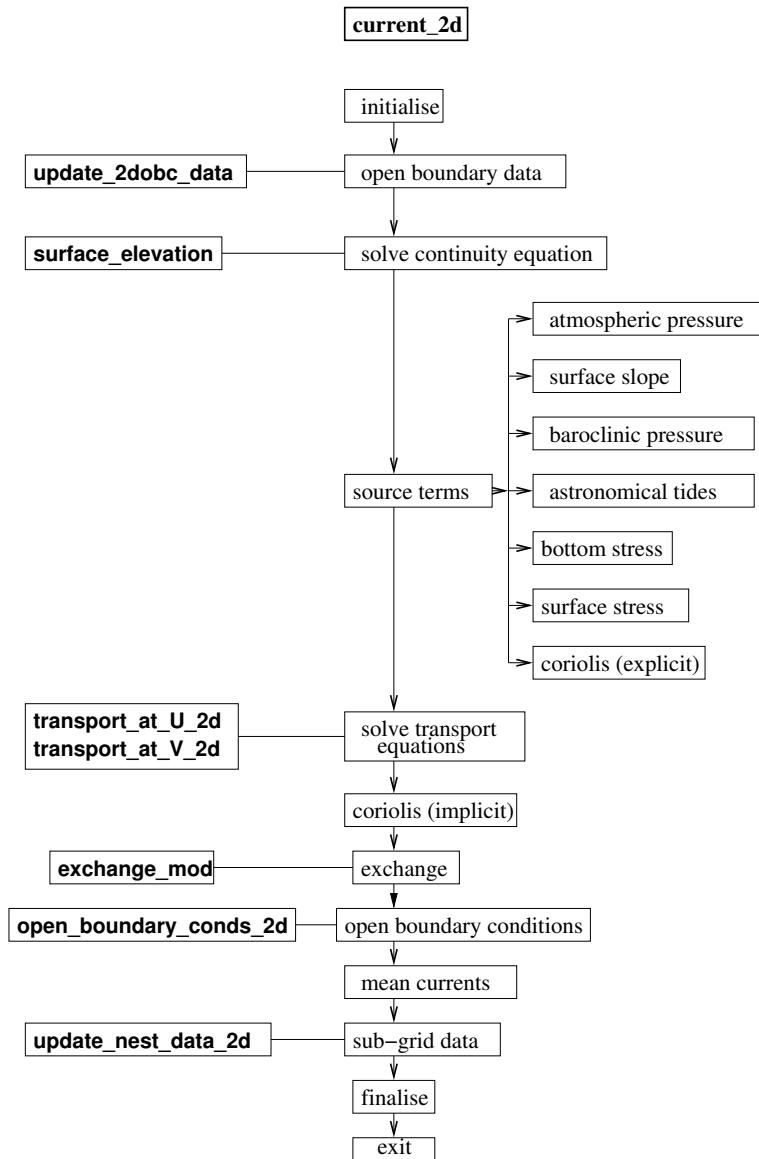


Figure 17.4: Diagram of routine `current_2d` which solves the 2-D mode equations.

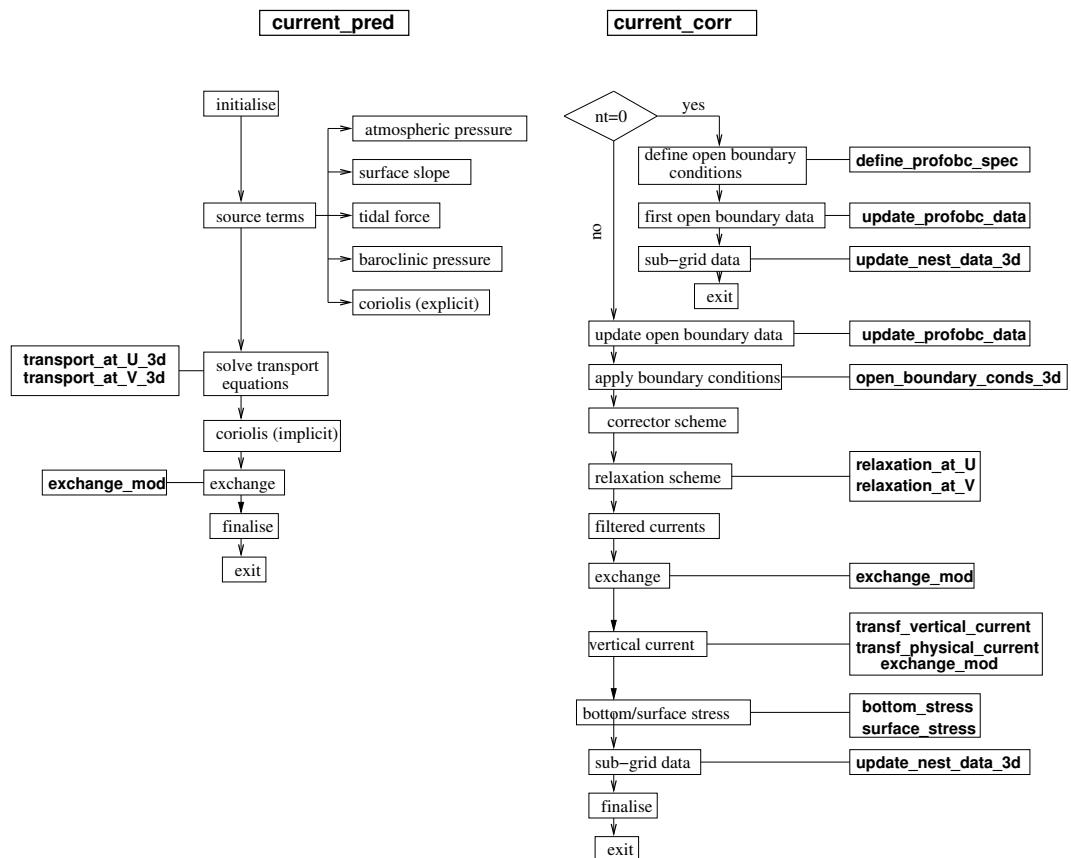


Figure 17.5: Diagrams of the routines `current_pred` and `current_corr` which solve the 3-D momentum equation at the predictor and corrector step.

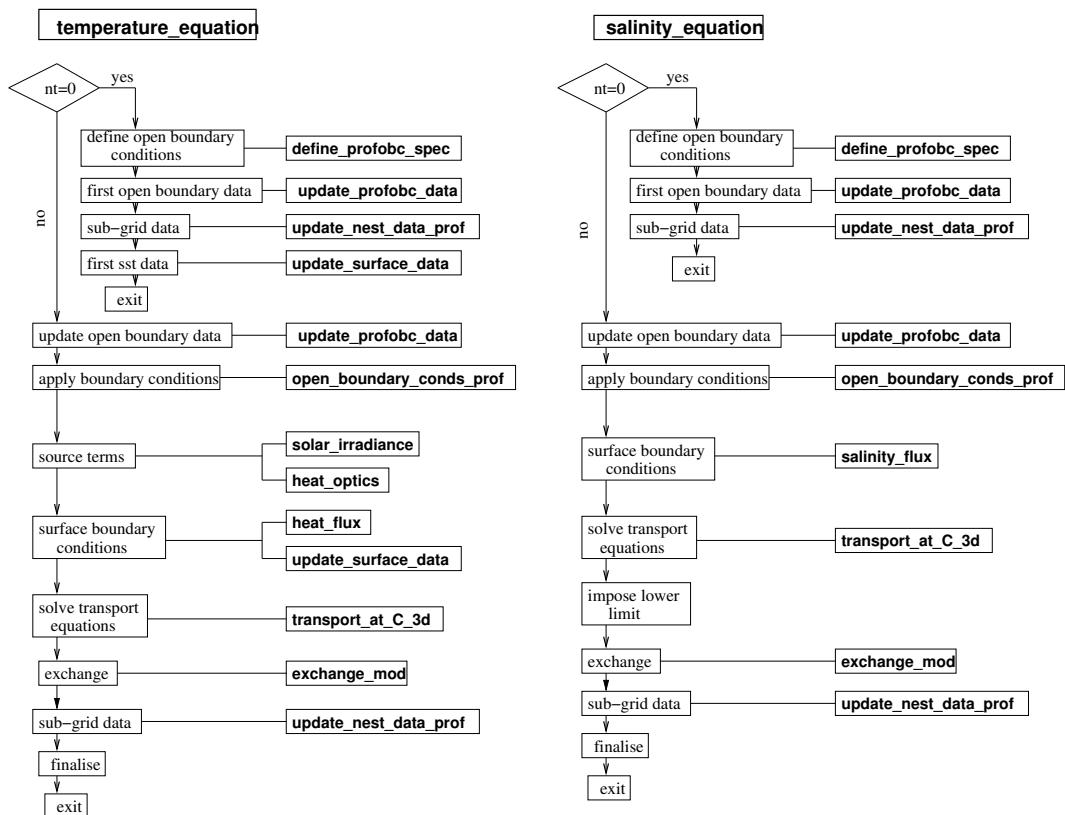


Figure 17.6: Diagrams of the routines `temperature_equation` and `salinity_equation` which solve the temperature and salinity equations.

Sediment equations

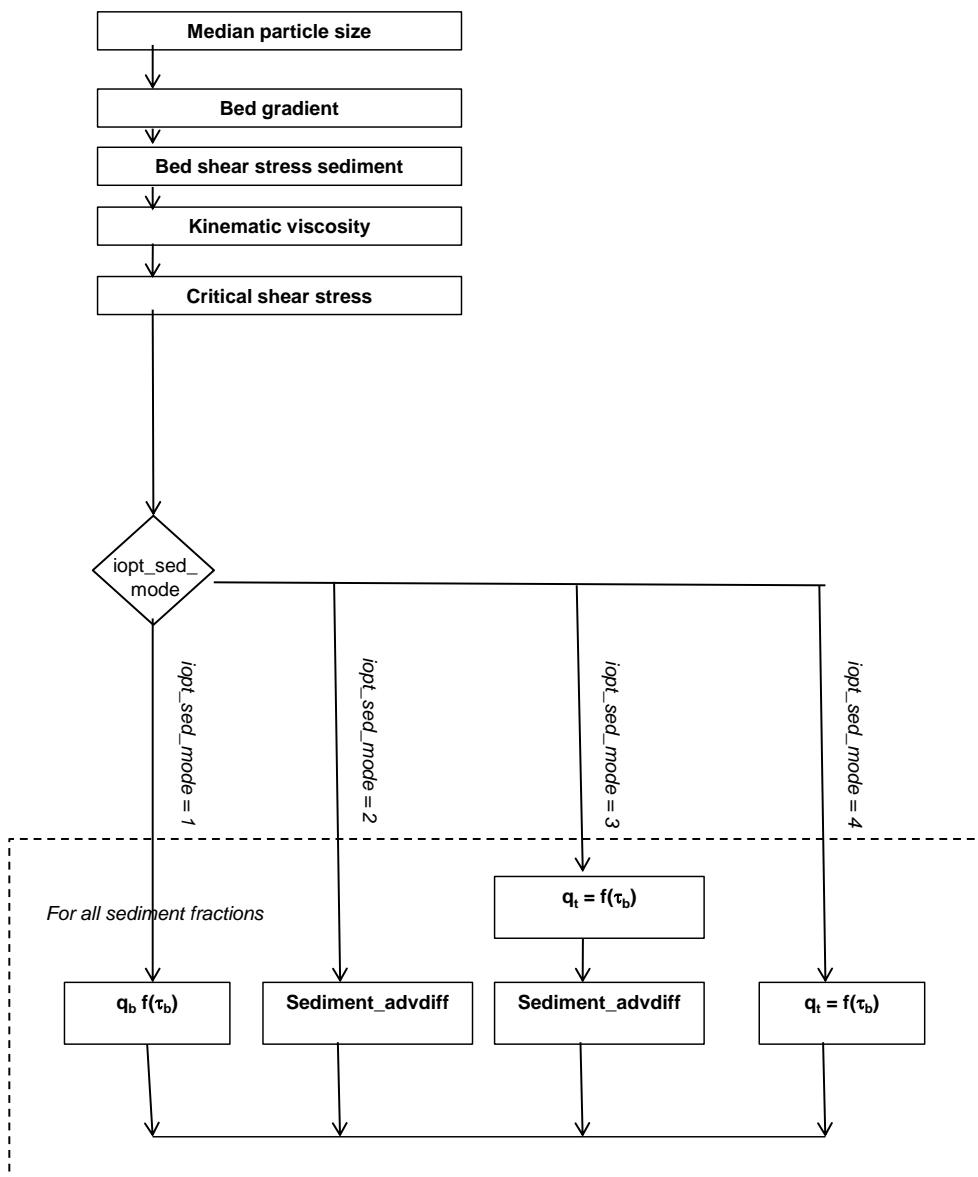


Figure 17.7: Diagram of the routine `sediment_equation` which is the “main” routine of the sediment transport model.

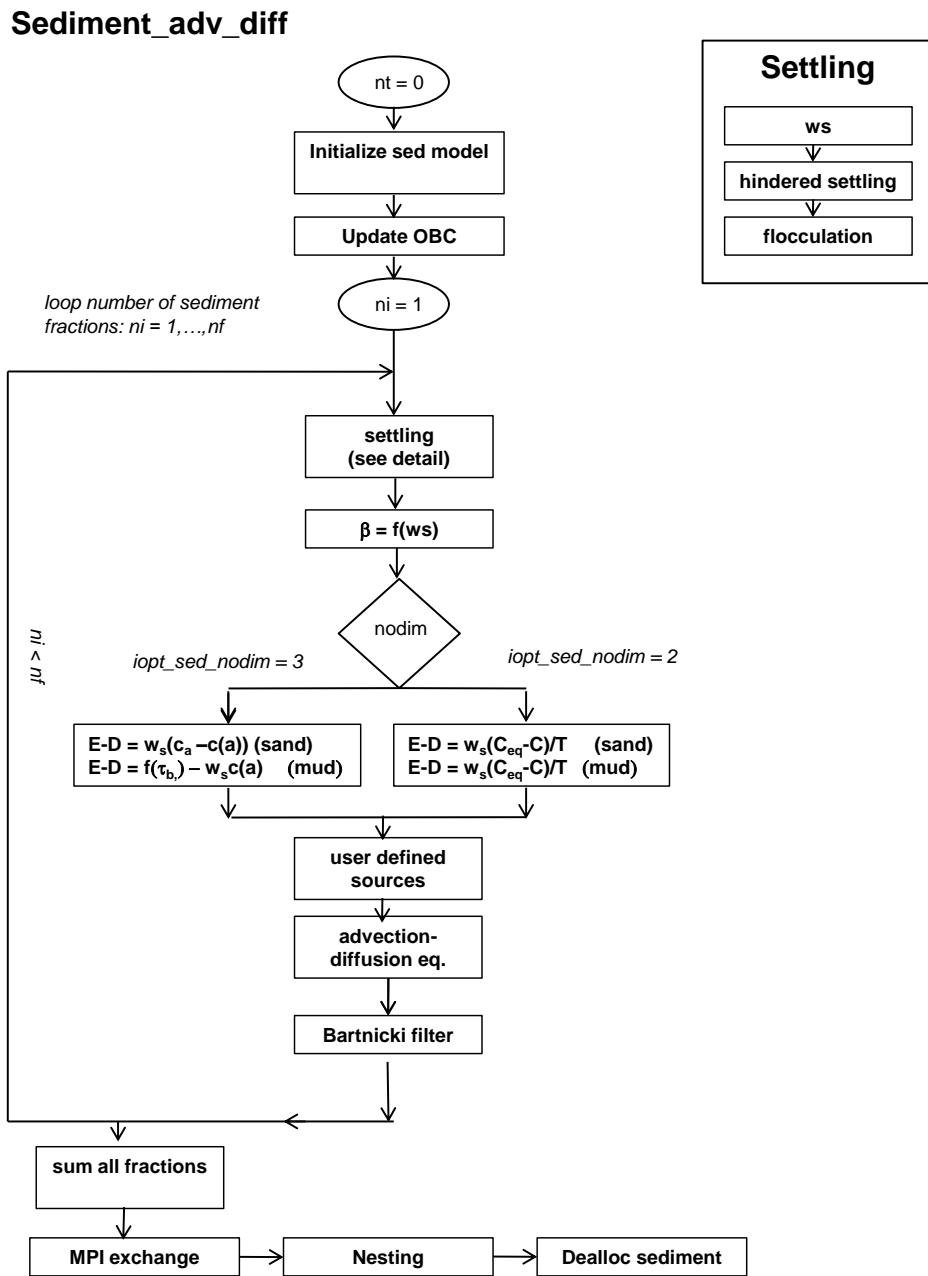


Figure 17.8: Diagram of the routine `sediment_advdiffequation` which solves the transport equations for sediments.

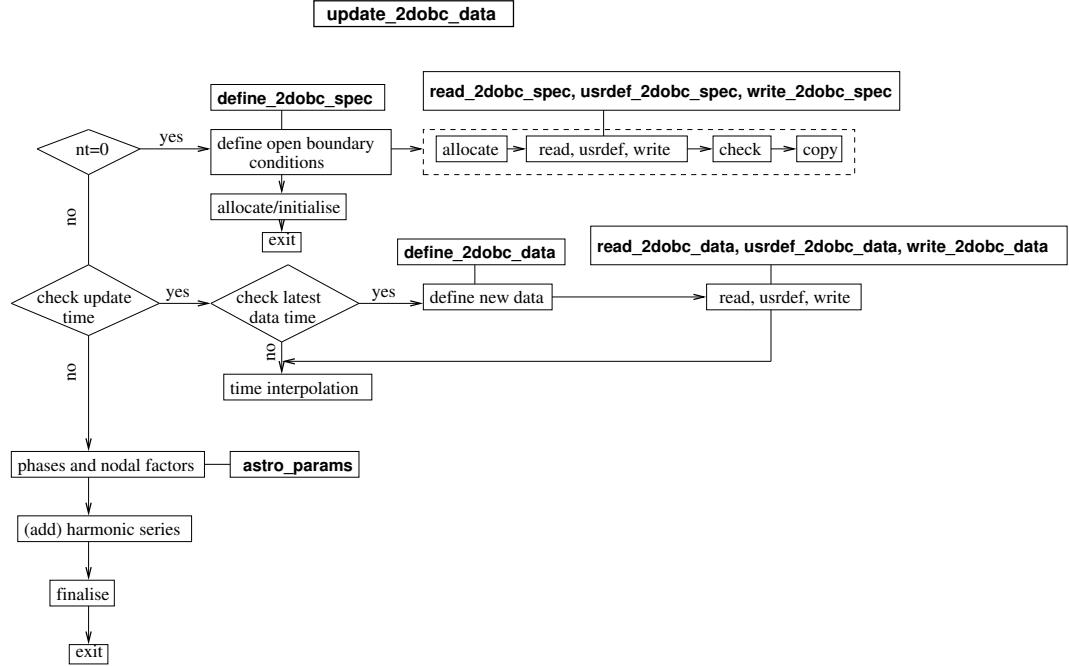


Figure 17.9: Diagrams of the routines used for defining and updating 2-D open boundary conditions and data.

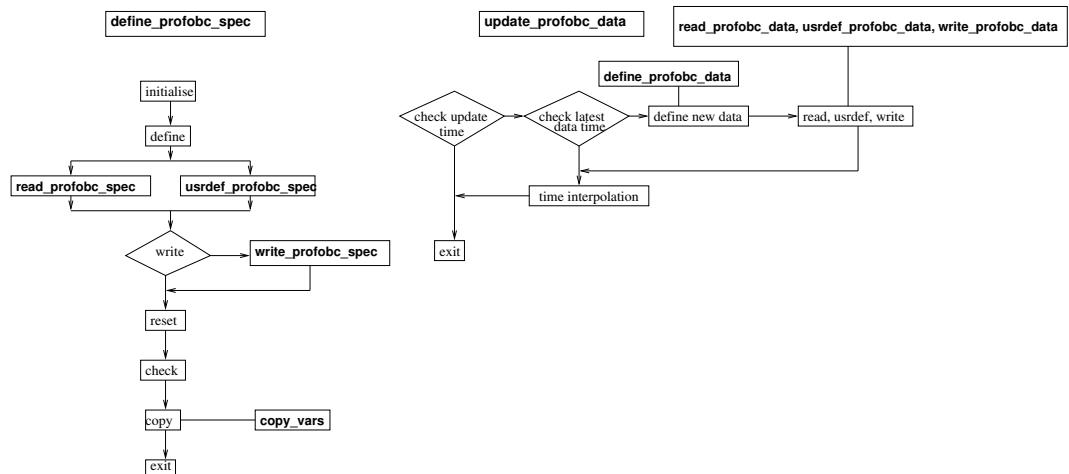


Figure 17.10: Diagrams of the routines used for defining and updating 3-D open boundary conditions and data.

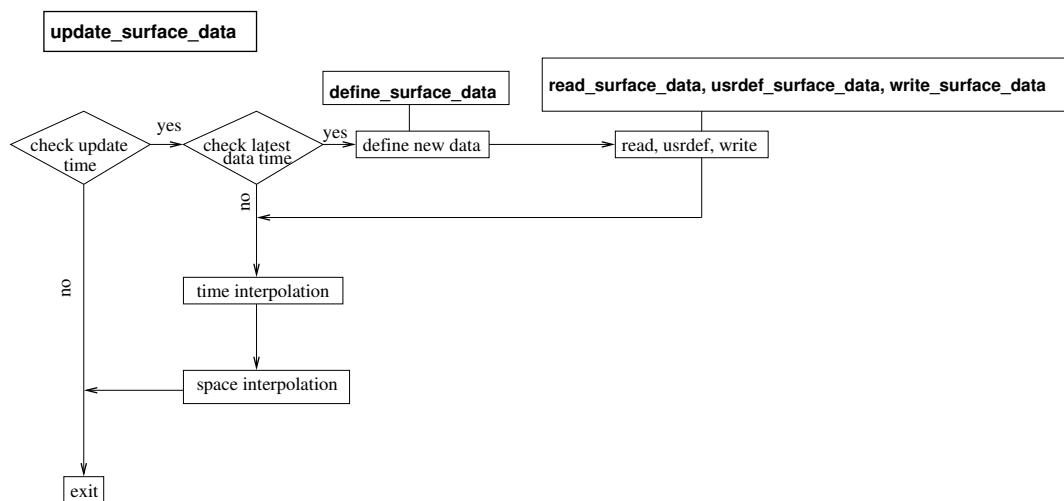


Figure 17.11: Diagram of the routines used for defining and updating data from an external 2-D grid.

Part IV

User manual

Chapter 18

Introduction

18.1 Setup methods

The setup of the model consists of three parts.

1. In the first one a series of model setup parameters are defined by the user:
 - general settings such as number of grid cells, time parameters (time step, start/end date, time counters, monitoring parameters, parallel setup, ...)
 - switches and model parameters for the different compartments (hydrodynamics, sediments, biology, tracers)
 - attributes of forcing files, surface grids and output results files.
2. In the second part the inputs of forcing data are prepared, either in the form of “standard” forcing files (which is the standard procedure for realistic setups) or through FORTRAN programming. The following types of forcing data are needed, depending on the type of simulation:
 - model grid and bathymetry
 - initial conditions (hydrodynamics, sediments, biology, tracers)
 - type of open boundary conditions
 - time series of open boundary data for different model variables
 - time series of surface forcing data (meteorological, waves, ...)
 - discharge data for one or more model variables.

Three methods are available for making a model setup:

1. With the `usrdef` method a series of **FORTRAN** routines are created by the user. More details are given in the next section.
2. Model settings parameters can alternatively be obtained by reading in a so-called “Central Input File” or **CIF**. The file can be created by the user without **FORTRAN** programming. This type of setup is discussed in Chapter 28.
3. Instead of using **FORTRAN** code the user can create forcing files by preprocessing using flexible standard formats. This is further discussed in Chapters 29–30.

The user can decide to make a mix of the three methods since each of the `usrdef` routines (with a few exceptions) can be replaced either by a **CIF** or by a standard forcing file.

For users with no experience in **FORTRAN 90** programming it is recommended to use the **CIF** and standard forcing methods for which no advanced experience is needed. Advantage of the `usrdef` method is that it allows to define the model setup in a more flexible way (e.g. obtaining forcing data in a non-standard format or by direct programming, …). Examples of **FORTRAN** coding can be found in the **setups** sub-directories where setups are defined for 47 test cases.

A complete description of all model setup parameters and arrays, which can possibly be used for model setup, is given in chapters 19–27 below. Most of them have a default value which means that if the user doesn’t define the variable, the default is taken. For most realistic applications, the majority of these defaults can be maintained and do not need to be changed. On the other hand, some parameters without default need to be defined (time step, number of grid cells, …). More details are given in the chapters below.

Before starting the simulation a (small) file `defruns` needs to be created in the directory where the simulation will be made. This is further discussed below in Section 18.3.

18.2 `usrdef` routines

A complete list of all `usrdef` files and routines is given in Table 18.1. Note that these files need to be installed in the user directory where the simulation will take place.

Table 18.1: Overview of all *Usrdef_* files and *usrdef_* routines in the program.

file	routine	C/F/U ¹	purpose
<i>Usrdef_Model.f90</i>	<i>usrdef_init_params</i>	U/C	setup and formats of monitoring files
	<i>usrdef_mod_params</i>	U/C	model parameters and attributes of forcing data
	<i>usrdef_grid</i>	U/F	model grid and bathymetry
	<i>usrdef_partition</i>	U/F	domain decomposition
	<i>usrdef_physics</i>	U/F	physical initial conditions
	<i>usrdef_1dsur_spec</i>	U/F	surface forcing conditions (elevations and/or surface slope) for 1-D (water column) applications
	<i>usrdef_2dcbc_spec</i>	U/F	open boundary conditions for the 2-D mode
	<i>usrdef_profcbc_spec</i>	U/F	open boundary conditions for scalar variables (baroclinic currents, temperature, salinity, sediments, biology)
	<i>usrdef_1dsur_data</i>	U/F	input of surface forcing data for 1-D (water column) applications
<i>Usrdef_Surface_Data.f90</i>	<i>usrdef_surface_absgrd</i>	U/F	input of 2-D open boundary data
	<i>usrdef_profcbc_data</i>	U/F	input of open boundary data for (baroclinic) currents and scalar variables

(Continued)

Table 18.1: Continued

	usrdef_surface_relgrd	U/F	definition of surface data grids in relative coordinates
	usrdef_surface_data	U/F	input of (2-D) surface forcing data
<i>Usrdef_Nested_Grids.f90</i>	usrdef_nstgrd_spec	U/C	parameters needed for nesting (number of sub-grid open boundary points, ...)
	usrdef_nstgrd	U/F	locations of the sub-grid open boundaries
<i>Usrdef_Sediment.f90</i>	usrdef_sed_params	U/C	parameters for the sediment model
	usrdef_morph_params	U/C	parameters for the morphological model
	usrdef_sedics	U/F	initial conditions for the sediment model
	usrdef_morphics	U/F	initial conditions for the morphological model
	usrdef_sed_spec	U/C/F	sediment particle properties
<i>Usrdef_Biology.f90</i>	usrdef_bio_params	U/C	parameters for the biological model
	usrdef_bioics	U/F	initial conditions for the biological model
<i>Usrdef_Particle.f90</i>	usrdef_part_params	C	parameters for the particle module
	usrdef_partics	U/F	initial conditions for the particle module
	usrdef_part_output	U	user-formatted output for the particle module
	usrdef_part_spec	U/C/F	specifiers for particle clouds
	usrdef_pout_params	U/C	parameters for output particle trajectories

(Continued)

Table 18.1: Continued

	<code>usrdef_parcl_data</code> <code>usrdef_particle_grid</code> <code>usrdef_particle_phys_data</code>	U/F U/F U/F	release time and location of particle clouds model grid arrays in case the particle module is run in off-line mode hydrodynamic data in case the particle module is run in off-line mode
<i>Usrdef_Structures.f90</i>	<code>usrdef_dry_cells</code> <code>usrdef_thin_dams</code> <code>usrdef_weirs</code> <code>usrdef_dischr_spec</code> <code>usrdef_dischr_data</code>	U/F U/F U/F U/C U/F	locations of dry cells locations of thin dams locations of and setup parameters for weirs and barriers specifiers for the discharge module discharge data
<i>Usrdef_Time_Series.f90</i>	<code>usrdef_tsr_params</code> <code>usrdef_tsr0d_vals</code> <code>usrdef_tsr2d_vals</code> <code>usrdef_tsr3d_vals</code>	U/C U U U	definition of metadata and output grid for time series output definition of 0-D time series (non-standard) output data definition of 2-D time series (non-standard) output data definition of 3-D time series (non-standard) output data
<i>Usrdef_Time_Averages.f90</i>	<code>usrdef_avr_params</code> <code>usrdef_avr0d_vals</code>	U/C U	definition of metadata and output grid for time averaged output definition of 0-D time averaged (non-standard) output data

(Continued)

Table 18.1: Continued

	usrdef_avr2d_vals	U	definition of 2-D time averaged (non-standard) output data
	usrdef_avr3d_vals	U	definition of 3-D time averaged (non-standard) output data
<i>Usrdef_Harmonic_Analysis.f90</i>	usrdef_anal_freqs	U/C	definition of frequencies and formats for harmonic analysis
	usrdef_anal_params	U/C	definition of metadata and output grid for harmonic output
	usrdef_anal0d_vals	U	definition of 0-D (non-standard) data for harmonic analysis
	usrdef_anal2d_vals	U	definition of 2-D (non-standard) data for harmonic analysis
	usrdef_anal3d_vals	U	definition of 3-D (non-standard) data for harmonic analysis
<i>Usrdef_Output.f90</i>	usrdef_output	U	user-formatted output (except particle module)

18.3 Preparing a simulation

Before starting the simulation a (small) file *defruns* needs to be created in the directory where the simulation will be made. The program will open this file at the start of the simulation(s) which is read line-wise. Each line contains, for a specific run, the values of three parameters separated by a ','. The general syntax is

`runtitle, status, filename`

¹Indicates which type of setup can be used: **usrdef** (U), **CIF** (C) or standard forcing file (F).

where

runtile The title of the simulation.

status Status of the CIF file

'0' The CIF utility is switched off (both for reading and writing).
Default.

'R' Model setup parameters are read from a CIF.

'W' Model setup parameters (including defaults) are written to a CIF.

filename Name of the CIF file. If not given, the default name **TRIM(runtile) // ".cifmod"** is taken. This parameter will be ignored if **status** equals '0'.

The **status** and **filename** parameters are stored in the derived type variable **ciffile** defined by

```
TYPE :: FileParams
  CHARACTER (LEN=1) :: status
  CHARACTER (LEN=leniofile) :: filename
  ...
END TYPE FileParams
TYPE (FileParams) :: ciffile
```

Defaults are taken (except for the **runtile** parameter which must always be given) when the value is an empty string, one or several blanks. All other blanks on the input line are ignored.

Consider the following example

```
conesA,,  
conesB,R,  
conesC,W,myciffile
```

The first line initiates the run **conesA** without CIF, the second one reads the setup from the file **conesB.cifmod**, the third writes the CIF data to the file **myciffile**.

Lines in **defruns** can be commented if the first character is a '!'. In that case the line is ignored.

The procedure can be used to combine multiple simulations within one run:

1. The program opens the file at the start.
2. The first line is read.

3. A first simulation is started with the given title.
4. At the end of the simulation, a next line is read with a new title and a next simulation starts up.
5. When there are no more lines to be read, the file is closed and the program terminates.

Once the setup is completed and the *defruns* file has been created, the program needs to be compiled using the procedures discussed in Chapter 3.2.

18.4 Setup conventions

The model setup, presented in the next chapters, uses certain conventions which are described below.

18.4.1 File formats

The format of the forcing files used in the setup and the output files with model results can take three forms:

- ‘A’ ASCII format. This format is portable and directly readable but unsuitable for large data sets because of its great comsumption of disk space and its use of sequential storage.
- ‘U’ Unformatted binary format. Advantage is a more efficient use of disk space. Disadvantages are that the file is (mainly) non-portable, not directly readable and uses sequential storage. The format is only maintained for compliance with previous versions and may be considered as obsolete in future COHERENS versions.
- ‘N’ netCDF format (recommended). This format is portable, binary (taking less disk space) and directly readable using the netCDF `ncdump` utility. Data can be accessed directly by specifying the appropriate record number.

18.4.2 Time parameters

During the simulation the current time is represented in both an absolute and a relative format. The first one is the calendar date and time `CDateTime` which is a string of 23 characters of the form ‘yyyy/mm/dd;hh:mm:ss:mmm’ where yyyy = year, mm = month, dd = day, hh = hour, mm = minutes, ss = seconds, mmm = milliseconds. The second one is the parameter `nosecsrun`

representing the time in seconds (rounded below to the nearest integer value) since the start of the simulation. Note that the second format does not take account of milliseconds.

When a calendar date and time is defined in the setup the milliseconds part 'mmm' must always be zero so that only the first 19 characters in the string need to be defined. This is the case for e.g. the setup parameters `CStartTime` and `CEndDateTime` representing respectively the start and end date of the simulation (see Section 19.3.3), the time coordinate `ciodatetime` used in forcing files,

For example

```
CStartTime = '2009/06/15;05:09:00:000'
CEndDateTime = '2009/07/01;15:45:00:000'
ciodatetime = '2020/05/10;02:15:10:000'
```

Different time steps are or may be used in the simulation. The smallest one is the “base” time step `timestep` which must be defined by the user in seconds (see Section 19.3.3). Its precision is restricted to 1 millisecond. All other time steps are taken as integer multiples of the base time step. For each time step a time counter is defined representing the ratio of the time step with respect to the base time step.

Other time parameters which are defined inside the code, but useful for the setup are

```
INTEGER :: nt, nsteps
```

where

nt Current time index in the simulation equal to the number of base time steps since the start of the run. Time indices are used in the program to determine the time when output will be written (Chapter 27) or to make updates of time-dependent forcing data (Section 19.4).

nstep Number of base time steps between the start and end of the simulation.

nosesrun Number of seconds since the start of the simulation. The parameter is used to compare the date and time in a forcing file with the current one in the simulation. Lowest precision is 1 second (milliseconds are not taken into account), upper precision \sim 68 years if `nosesrun` is defined in single precision or unlimited if defined in double precision precision¹.

¹Some compilers (like ifort) allow the use of double precision integers, others (like gfortran) do not allow it.

18.4.3 Key ids

Key ids are named integer constants which refer to a specific item, such as a variable, file class, tidal constituent, error code, The name of a key id is composed by the name of its general “class” to which the item belongs, followed by a ‘_’ and its specific key name. Some examples are given below. Key ids are a useful tool for making up a model setup.

1. Variable key ids have the common class name `iarr`. Their specific key names are the same as their **FORTRAN** name. For example, `iarr_sal` is the key id of the program variable ‘`sal`’ which is salinity. Variable key ids are used in the model setup to define the variables for user-defined output (see Chapter 27 for details).
2. Key ids referring to tidal constituents belong to the class `icon_`. The specific name is given by the constituent’s traditional name, e.g. `icon_M2` for the M_2 tide. Tidal key ids are a practical tool for defining the tidal constituents applied in open boundary conditions or for selecting the constituents for the astronomical tidal force.
3. The class name for model forcing files is ‘`io_`’. The specific name refers to the type of forcing. For example, `io_inicon` is the key id referring to a initial condition file (see Section 19.4 for details).
4. Key ids for biological state variables belong to the class `isv_` for 3-D variables in the water column and `isvs_` for 2-D variables in the sediment.

A list of available key classes is displayed in Table 18.2.

18.4.4 Data types

Each data type in the model has a specific type name and **KIND** parameter which are defined in the model by named constants (see `syspars.f90`). The name of these parameters are listed in Table 18.3.

18.4.5 Named constants and array dimensions

In the chapters below named constants are used in the declaration of setup arrays and data strings. These parameters can only be modified by changing their values in the source files `syspars.f90`. A listing and their current values are given in Table 18.4.

²The **KIND** parameter is set to 4 if the compiler doesn’t support long integers (such as `gfortran`) and to 8 otherwise.

Table 18.2: List of available key classes

Class	Purpose	Defined in	Examples
iarr_	model variables	<i>modids.f90</i> , <i>sedids.f90</i> , <i>bioids.f90</i> , <i>partids.f90</i>	<i>iarr_temp</i> (temperature)
io_	forcing file	<i>iopars.f90</i>	<i>io_modgrd</i> (model grid file)
icon_	tidal	<i>tide.f90</i>	<i>icon_M2</i> (M_2 constituent)
ics_	constituents		
	initial conditions	<i>iopars.f90</i>	<i>ics_phys</i> (physical initial conditions)
isv_	3-D biological state variables	<i>biopars.f90</i>	<i>isv_si</i> (silicon consuming plankton)
isvs_	2-D biological state variables	<i>biopars.f90</i>	<i>isvs_oms</i> (sediment organic matter)
igrd_	surface grids	<i>iopars.f90</i>	<i>igrd_meteo</i> (meteo data grid)
icif_	CIF block	<i>iopars.f0</i>	<i>icif_sed</i> (sediment model setup parameters)
itm_	timers for timing report	<i>iopars.f90</i>	<i>itm_hydro</i> (hydrodynamics)
ierrno_	error codes	<i>iopars.f90</i>	<i>ierrno_alloc</i> (allocation error)

Table 18.3: List of type names and KIND parameters.

Data type name	Type	KIND name	KIND value
CHARACTER	<i>char_type</i>	<i>kndchar</i>	'A'
LOGICAL	<i>log_type</i>	<i>kndlog</i>	.TRUE.
INTEGER	<i>int_type</i>	<i>kndint</i>	4
LONG INTEGER	<i>ilong_type</i>	<i>kndilong</i>	8 or 4 ²
REAL	<i>real_type</i>	<i>kndreal</i>	8
DOUBLE PRECISION REAL	<i>rlong_type</i>	<i>kndrlong</i>	8
COMPLEX	<i>kndcmplx</i>	<i>kndcmplx</i>	4

18.4.6 Data flags

Data flags are commonly used in observational data sets for representing invalid data. They have been introduced in the COHERENS code as “undefined” values. The following variables are defined in the program for the flagging of model variables

REAL :: real_fill	Flag for simple precision real variables.
REAL(KIND=knrlong) :: double_fill	Flag for double precision real variables.
REAL :: float_fill	Equals <code>real_fill</code> or <code>float_fill</code> depending on whether COHERENS is compiled in simple precision or double precision mode.
INTEGER :: int_fill	Flag for integer variables.

Data flags are used (e.g.) for the following purposes:

- If a data value in a vertical open boundary data profile has been flagged, a zero gradient condition is applied at that particular point, and the data value is no longer considered as an external value.
- Missing data or data at dry locations in a surface forcing data file are flagged. Procedures for flagged values are discussed in Chapter 22.
- When a variable is horizontally interpolated from four neighbouring points, the interpolating points at dry locations are marked with a flag to exclude them from the interpolation procedure.
- Flags are used to mark dry cells in output files.
- Flagging of some user-defined model parameters has been implemented as a practical utility in the program. It informs the program that the parameter has some specific value unknown by the user, but known to the program.

18.4.7 Variable units

The units of most model variables are based on the ‘kg-meter-sec-PSU’ system³. Table 18.5 represents the units of the principle variables used in the program. Note that the unit used inside the code, may be different from the one used for output.

³Other specific units are used for biological variables.

18.4.8 Dimensions of setup arrays

A number of setup parameters are used for defining the shapes of model setup and forcing arrays. These parameters are discussed in the next chapters. A list of dimension names and their purpose is given in the Table 18.6 below.

18.5 Selecting method for model setup

The question arises how the user can select which of the three setup methods will be used.

- The **CIF** is selected when the following conditions are satisfied.
 - The **CIF** is activated by setting its status to 'R' in the `defruns` file.
 - As explained in Section 28.1, the **CIF** is composed of a series of blocks. Each block has a name given on the first data line of the block which is the name of a `usrdef` routine without the prefix `usrdef_`. The **CIF** method will be used if the corresponding block name is found in the **CIF**. Otherwise the `usrdef` method is taken. The **CIF** method is further discussed in Chapter 28.
- The choice between a standard forcing input file or a `usrdef` routine is determined by the `status` attribute of the 3-D derived type setup array `modfiles`. Each type of forcing has an associated key id `iddesc` of class `io_*` and a file number `ifil` (since in some cases the input data can be spread over multiple files). If a **CIF** option is available for a forcing type (see Table 18.1), the setup data are obtained from the **CIF** when the conditions above are satisfied. In all other cases, the `usrdef` or standard forcing file method is selected when `modfiles(iddesc,ifil,1)%status` equals 'N' or 'R'. For more detailed information about `modfiles` and forcing key ids see Section 19.4.

18.6 Some recommendations for programming

The model setup with the `usrdef` method requires that FORTRAN 90 program code is inserted in one or more `Usrdef_` files. Below are a few recommendations and hints.

declaration of variables All variables used within a routine must be declared. For local variables this is easily done within the routine. Although not required explicitly, the `COHERENS` code is entirely pro-

grammed without “implicit typing rules”. This means that the local variable declaration part should start with the line

```
IMPLICIT NONE
```

Most variables within a `usrdef_` routine have a non-local scope and are already declared in a **MODULE** file. They are made accessible in the `usrdef_` routine via a

```
USE X
```

statement at the head of the routine where `X` is the name of the module. The name of a module always appears on the first line of the a module file. These are the files in one of the source directories whose names start with a lower case character and contain no “`_`” (underscore) character. The file names have a close relation with their contents (like `currents.f90` which contains the modules with all declarations concerning currents). Module routines used within a `usrdef_` routine are declared with a

```
USE X, ONLY: Y
```

statement where `X` is the name of the module and `Y` the name of the module routine. Module routine file names start with a lower case letter and have an “`_`” character in their name. The routine name can be found within the routine. The following is an example of **USE** statements

```
USE currents
USE inout_routines, ONLY: close_file, open_file
```

file operations Important to know is that files cannot be opened or closed with the standard FORTRAN **OPEN** and **CLOSE** statements. There are two alternatives, based upon routines declared in `inout_routines.f90`:

1. Use `open_file` instead of **OPEN** and `close_file` instead of **CLOSE**. The routines are declared as follows.

```
SUBROUTINE open_file(iounit,filename,ioctype,ioform)
CHARACTER (LEN=lenform), INTENT(IN) :: ioform
CHARACTER (LEN=*), INTENT(IN) :: filename, ioctype
INTEGER, INTENT(INOUT) :: iounit
```

where

iounit File unit which is determined by the program and not by the user (as is the case for the **OPEN** statement).

filename The name of the input or output file.

iotype File access which may have the values

'IN' : Read access.

'OUT' : Write access.

'INOUT': Both read and write access.

ioform Format of the inout/output file.

'A': ASCII

'U': Unformatted binary

'N': netCDF format

Note that the file unit is attributed by the program and not by the user to avoid that the same unit is used for different files.

```
SUBROUTINE close_file(iounit,ioform,filename,fildel)
LOGICAL, INTENT(IN), OPTIONAL :: fildel
CHARACTER (LEN=lenform), INTENT(IN) :: ioform
CHARACTER (LEN=*), INTENT(IN), OPTIONAL :: filename
INTEGER, INTENT(INOUT) :: iounit
```

where

iounit File unit.

ioform Format of the inout/output file.

'A': ASCII

'U': Unformatted binary

'N': netCDF format

filename The name of the input or output file.

fildel The file is deleted if PRESENT and .TRUE.

Once the file is closed, the file unit can be re-used for opening of a new file.

2. The second option is to open and close a file using the routines **open_filepars** and **close_filepars**.

```
SUBROUTINE open_filepars(filepars,iotype)
CHARACTER (LEN=*), INTENT(IN), OPTIONAL :: iotype
TYPE(FileParams), INTENT(INOUT) :: filepars
```

where

filepars A derived type variable containing the attributes of the file (name, format, unit, status, ...).

iotype File access which may have the values

'IN' : Read access.

'OUT' : Write access.

'INOUT': Both read and write access.

As in the previous case, the file unit is attributed by the program. For more information about the derived type **FileParams** see Sections 19.4 and 27.2.2.

time series data A special procedure needs to be followed within a **usrdef_** routine which reads time series data (i.e. the routines having **ciodate** as an argument).

1. On first call, the file is opened and the **iostat** file attribute is reset from 0 to 1. No data are read.
2. A second call is made, immediately after the first one (i.e. at the same model time step) to read the date/time and first series of data. These calls are repeated until the date/time of the last input is later than the actual model time.
3. Two time records are saved in memory. The first is the latest record before or at the initial time, the second is the earliest record later than the initial time. The two records are used to perform interpolation in time at run times between the two records.
4. When during program execution, the internal date in the model equals or becomes later than the one obtained from the last input, the **usrdef_** routine is called again, until the date/time in the file becomes later than the actual model time.
5. If an end of file condition occurs or the user knows that this will occur on a next read, the user should set the **iostat** number to 2. The program will no longer call the routine. Further action (decided by the program) now depends on the value of the **endfile** attribute (see Section 19.4).
6. At the end of the simulation the program automatically closes all open files whose attributes are stored in an element of the array **modfiles** using **close_filepars**. Otherwise, the user has to close the file using **close_file** when the last data record has been read.

Table 18.4: List of constants used in the declaration of model setup arrays and strings.

Parameter	Purpose	Current value
<code>lencifline</code>	Maximum length of a data line in a CIF	2000
<code>lendesc</code>	Maximum length of a long name variable attribute	120
<code>lenform</code>	Length of a form file attribute	1
<code>lenformat</code>	Maximum length of a FORTRAN format specification	30
<code>leniofile</code>	Maximum length of a file name	120
<code>lenname</code>	Maximum length of a f90_name variable attribute	35
<code>lentime</code>	Length of a calendar date and time string	23
<code>lentitle</code>	Maximum length of the simulation title	20
<code>lenunit</code>	Maximum length of a units variable attribute	60
<code>MaxAstroTides</code>	Maximum number of constituents for the astronomical force	56
<code>MaxCifVars</code>	Maximum number of data values on a CIF input line	500
<code>MaxConstituents</code>	Maximum number of tidal constituents at open boundaries	77
<code>MaxErrMesgs</code>	Maximum allowed number of error messages	50
<code>MaxGridTypes</code>	Maximum number of surface grid types	5
<code>MaxIOfiles</code>	Maximum number of forcing files for each forcing type	33
<code>MaxIOTypes</code>	Maximum number of model forcing types	51
<code>MaxMGLevels</code>	Maximum allowed number of grid levels (including the main grid) when the multi-grid scheme is activated	
<code>MaxProgLevels</code>	Maximum number of sub-program levels	20
<code>MaxRestarts</code>	Maximum number of times in the simulation when final conditions are written	10

Table 18.5: Units of principal variables.

variable type	unit
length	m
time	s
mass	kg
currents	m/s
transports	m ² /s
temperature	°C
salinity	PSU (=10 ⁻³ kg/kg)
angle	radians
density	kg/m ³
horizontal coordinates	m or fractional degrees
vertical coordinates	m or dimensionless
pressure	Pa
diffusion coefficients	m ² /s
frequencies	radians/s
stress	m ² /s ² (internally) or Pa (output)
heat flux	W/m ²
salinity flux	PSU m/s
turbulent energy	m ² /s ² (or J/kg)
turbulent dissipation	m ² /s ³ (or W/kg)
sediment concentrations	m ³ /m ³ (internally) or kg/m ³ (output)
bed/total load	m ² /s (internally) or kg/m/s (output)

Table 18.6: Dimension names used in the model setup for forcing arrays.

Parameter	Purpose
nb	number a sediment bed layers
nc	number of grid cells in the X-direction on the (global) computational grid
ncloc	number of grid cells in the X-direction on the local grid
nconastro	number of tidal constituents used for astronomical forcing
nconobc	number of tidal constituents used for open boundary forcing
nf	number of sediment fractions used in the multi-fraction sediment model
nobu	number of open boundaries at U-nodes
nobv	number of open boundaries at V-nodes
nobx	number of open boundaries at X-nodes
noby	number of open boundaries at Y-nodes
noclouds	number of clouds in the particle module
nolabels	number of particle labels in the particle module
nopart	number of particles used in the particle module
nonestsets	number of nested sub-grids
nprocs	number of parallel processes (1 in the serial case)
nqsecobu	number of sections along the U-boundaries where a distributed discharge condition is specified
nqsecobv	number of sections along the V-boundaries where a distributed discharge condition is specified
nr	number of grid cells in the Y-direction on the (global) computational grid
nrloc	number of grid cells in the Y-direction on the local grid
numwbaru	number of weirs or barriers at U-nodes
numwbarv	number of weirs or barriers at V-nodes
numdis	number of discharge locations
numdry	number of dry cells
numthinu	number of thin dams at U-nodes
numthinv	number of thin dams at V-nodes
nz	number of vertical grid levels

Chapter 19

General and physical model parameters

The parameters, discussed in this chapter are defined in the following routines of *Usrdef_Model.f90* :

- `usrdef_init_params`: setup of monitoring parameters (Section 19.1)
- `usrdef_mod_params`: switches, model parameters and attributes of forcing files and surface grids (Sections 19.4-19.5).

19.1 Parameters for monitoring

This section describes the parameters used to set up the monitoring and a few other general parameters. They are defined in `usrdef_init_params` or in the **CIF** block `init_params` if it exists and the **CIF** is activated.

```
SUBROUTINE usrdef_init_params
```

```
  USE iopars
  USE paralpars
  USE switches
  USE syspars
```

```
  IMPLICIT NONE
```

```
END SUBROUTINE usrdef_init_params
```

The following parameters and arrays may be defined here

```

LOGICAL :: cold_start, sedlog, warning
CHARACTER (LEN=leniofile) :: errlog_file, inilog_file, monlog_file, &
                           & runlog_file, timing_file, sedlog_file, &
                           & warlog_file
INTEGER :: iopt_part_model, iopt_waves_model, levtimer, maxerrors, &
            & monlog, nprocscoh, nprocswav, runlog_count, timer_format
INTEGER, DIMENSION(npworld) :: levprocs_err, levprocs_ini, &
                           & levprocs_run

```

19.1.1 Cold start

cold_start If `.TRUE.`, the program executes only the initialisation and finalisation procedures, but does not enter the time loop. The option is useful for debugging. Default is `.FALSE.`.

19.1.2 Log files

- levprocs_ini** Determines the level of tracing of the *inilog* file for each process. Different levels can be defined for different processes. If zero, no *inilog* file will be written. The *inilog* file only contains information about model initialisation and is closed as soon as the program enters the time loop. The size (`npworld`) of the vector array equals either the number of processes defined within `MPI_COMM_WORLD` in the parallel case or 1 in the serial case. Default is zero.
- levprocs_run** Determines the level of tracing of the *runlog* file for each process. Different levels can be defined for different processes. If zero, no *runlog* file will be written. The *runlog* file traces program execution during the time loop. The size (`npworld`) of the vector array equals either the number of processes defined within the `MPI_COMM_WORLD` communicator in the parallel case or 1 in the serial case. Default is zero.
- inilog_file** Name of the *inilog* file. Default is `TRIM(runtile)//'.inilog'`. In parallel mode, the file name is appended with the process id number.
- runlog_file** Name of the *runlog* file. Default is `TRIM(runtile)//'.runlog'`. In parallel mode, the file name is appended with the process id number.
- runlog_count** Sets the number of time steps after which the *runlog* file is updated (overwritten) during the run. Default is the total number

of model time steps (i.e. information is written at all time steps and the file is never overwritten).

19.1.3 Error files

maxerrors Maximum allowed number of error messages within the *errlog* file. Default is **MaxErrMsgs** defined in *syspars.f90*.

levprocs_err Determines the level of error checking for each process.

- 0: Error checking is disabled (for a particular process) and no file is created.
- 1: Error checking is enabled during the initialisation phase only.
- 2: Error checking is enabled for the whole simulation. Default.

errlog_file Name of the *errlog* file. Default is **TRIM(runttitle) // '.errlog'**. In parallel mode, the name is appended with the process id number.

19.1.4 Warning file

warning Disables/enables the writing of a *warning* file. Default is **.TRUE.**.

warlog_file Name of the *warning* file. Default is **TRIM(runttitle) // '.warlog'**.

19.1.5 Timer file

levtimer Determines the type of information in the timer report.

- 0: No timer report is written.
- 1: Writes the total execution time only. Default.
- 2: Writes time information (in % of the total time) for all “timers”. In case of a parallel run, the information is written as follows: time on the master process, mean, minimum and maximum time over all processes.
- 3: The same as previous, but in case of a parallel run, the information is additionally written for each individual process. In the serial case, behaviour is the same as for case 2.

timing_file Name of the timer report file. Default is **TRIM(runttitle) // '.timing'**.

timer_format Format for writing the total execution time in the timer report.

- 1: Seconds. Default.

- 2: Minutes.
- 3: Hours.
- 4: Days.

19.1.6 Monitoring files

monlog	Determines the level of monitoring for the iteration procedure used in the multi-grid scheme.
0:	No monitoring report is produced. Default.
1:	Outer loop only
2:	Outer and inner loop (without smoothing iterations)
3:	Outer and inner loop (including smoothing iterations)
sedlog	Determines whether a monitoring file is written for the sediment module. Default is .TRUE..
monlog_file	Name of the monitoring file for the multi-grid scheme if monlog is non-zero. Default is <code>TRIM(runtile) //'monlog'</code> .
sedlog_file	Name of the monitoring file for the sediment module if monlog is non-zero. Default is <code>TRIM(runtile) //'sedlog'</code> .

19.1.7 General switches

iopt_part_model	Selects the mode for particle tracking.
0:	Particle module is disabled. Default.
1:	Particle module and the hydrodynamic part of COHERENS run in serial mode.
2:	Particle module runs on-line, while COHERENS runs in parallel mode and sends hydrodynamic data to the particle mode using MPI communication calls.
3:	Particle module runs off-line in forward mode and reads hydrodynamic data either from a previous COHERENS run or from a non-COHERENS external data file.
4:	Particle module runs off-line in backward mode and reads hydrodynamic data either from a previous COHERENS run or from a non-COHERENS external data file.
iopt_waves_model	Selects whether COHERENS is coupled with an external (SWAN) wave model.

- 0: No coupling. Default.
- 1: COHERENS is coupled with the SWAN model.

19.1.8 Number of processes for each sub-model

nprocscoh Number of processes used by COHERENS. Default is **npworld**. This parameter must always be defined if **iopt_waves_model**=1 or **iop_part_model**=2.

nprocswav Number of processes used by the SWAN model when **iopt_waves_model**=1.

Note that the sum of the process numbers for all sub-models must be equal to the number of processes used in the batch job (PBS, slurm, ...) by which COHERENS is launched, i.e.

$$\text{nprocscoh} + \text{nprocswav} + \text{nprocspart} = \text{npworld}$$

where **nprocspart** equals 1 if **iop_part_model** equals 2 and 0 otherwise.

19.2 Model switches

In this section a list of the switches and parameters is presented for the model in general and for the physical part. They are all defined in the routine **usrdef_mod_params** or in the **CIF** block **mod_params** if it exists and the **CIF** is activated.

FORTRAN template

SUBROUTINE **usrdef_mod_params**

```
USE gridpars
USE iopars
USE nestgrids
USE obconds
USE paralpars
USE physpars
USE structures
USE switches
USE tide
USE timepars
USE turbpars
```

IMPLICIT NONE

```

procname(pglev+1) = 'usrdef_mod_params'
CALL log_timer_in()
...
CALL log_timer_out()

END SUBROUTINE usrdef_mod_params

```

The aim of the model switches (FORTRAN name starting with `iop_`) is to make selections between different options (type of the model, activation or deactivation of a specific module or specific utilities, options for selecting different kind of processes or numerical schemes, ...). Only general switches or switches related to the physical part can be defined here. Switches for other modules such as sediments, flocculation, morphology, biology and tracers are defined in other routines, described in Sections 23.1.1.1, 23.1.4, 23.1.2.1, 23.2.1, 24.1.1 and 25.1.1.

Note that all switches are of type `INTEGER`.

19.2.1 Model grid

<code>iop_grid_htype</code>	Type of horizontal grid.
	1: Uniform rectangular grid. Default.
	2: Non-uniform rectangular grid.
	3: Curvilinear grid.
<code>iop_grid_nodim</code>	Grid dimension.
	1: 1-dimensional grid (water column model).
	2: 2-dimensional grid (depth-averaged model without vertical structure).
	3: 3-dimensional grid. Default.
<code>iop_grid_sph</code>	Type of coordinates.
	0: Cartesian coordinates.
	1: Spherical coordinates. Default.
<code>iop_grid_vtype</code>	Type of vertical grid.
	1: Uniform σ -grid. Default.
	2: Horizontally uniform and vertically non-uniform σ -grid.

3: Horizontally and vertically non-uniform σ -grid.

iopt_grid_vtype_transf Type of vertical grid transformation.

0 : No vertical grid transformation. Default.

20: User defined when **iopt_grid_vtype**=2.

21: Log-transformation (4.23) at the bottom following [Davies & Jones \(1991\)](#) if **iopt_grid_vtype**=2.

22: Log-transformation (4.24) at the surface following [Davies & Jones \(1991\)](#) if **iopt_grid_vtype**=2.

23: [Burchard & Bolding \(2002\)](#) transformation with enhanced resolution near the bottom and/or the surface as defined by (4.26) if **iopt_grid_vtype**=2.

30: User defined when **iopt_grid_vtype**=3.

31: [Song & Haidvogel \(1994\)](#) transformation given by (4.33) and (4.35) if **iopt_grid_vtype**=3.

32: [Burchard & Bolding \(2002\)](#) spatially non-uniform transformation with enhanced resolution near the bottom and/or the surface as defined by (4.38)–(4.39) if **iopt_grid_vtype**=3.

19.2.2 Interpolation

iopt_arrint_depths Selects type of interpolation for water depths at velocity nodes.

1: Linear interpolation. Default (recommended).

2: Using the minimum value from the two surrounding C-nodes (not recommended).

iopt_arrint_hreg Disables/enables the use of non-uniform weight factors for horizontal interpolation of arrays defined on the model grid.

0: Disabled. Default.

1: Enabled.

iopt_arrint_vreg Disables/enables the use of non-uniform weight factors for vertical interpolation of arrays defined on the model grid.

0: Disabled. Default.

1: Enabled.

iopt_arrint_3D Selects the dimension of masks or weight factors used in some array interpolations.

0: 2-D masks or weights. Default (recommended).

1: 3-D masks or weights.

It is recommended to reset the switches `iopt_arrint_hreg` and `iopt_arrint_vreg` to 1 only in the case of model grids with highly irregular grid spacings and to keep the default value for `iopt_arrint_3D`.

19.2.3 Hydrodynamics

iopt_curr Type of model for current and surface elevations.

0: Currents and elevations are set to their default (zero) values and are not updated.

1: Currents and elevations are initialised but not updated in time.

2: Currents and elevations are initialised and updated in time. Default.

iopt_curr_wfall When enabled, a correction term given by (5.39) is added to the horizontal current components in case of a scalar quantity representing particulate matter having a fall/rising velocity.

0: Disabled. Default.

1: Enabled.

iopt_hydro_impl Disables/enables the implicit scheme for the barotropic mode.

0: The momentum and continuity equations are solved with the explicit (mode-splitting) scheme. Only recommended for high resolution grids when the time step limitations for the 3-D and 2-D mode, as given by equations (12.3) and (12.1) are comparable. Default.

1: The momentum and continuity equations are solved using the implicit algorithm.

19.2.4 Density

<code>iopt_dens</code>	Evaluation of the density and expansion coefficients.
0:	Uniform density, zero expansion coefficients. Default.
1:	Density is calculated from the linear equation of state (5.67), expansion coefficients are uniform. Retained for compatibility with previous versions. Should be used only for idealised tests.
2:	From the McDougall <i>et al.</i> (2003) general equation of state (5.62)–(5.66) without pressure effects.
3:	From the McDougall <i>et al.</i> (2003) general equation of state (5.62)–(5.66) with pressure effects included. Recommended when temperature and/or salinity are included in the simulations.
<code>iopt_dens_convect</code>	Disables/enables the vertical convective adjustment scheme.
0:	Disabled. Default.
1:	Enabled only when <code>iopt_dens > 1</code> .
<code>iopt_dens_grad</code>	Selects the numerical algorithm for discretisation of the baroclinic pressure gradient.
0:	The baroclinic pressure gradient is set to zero.
1:	Traditional σ -coordinate (second order) method. Default.
2:	z -level method. Not recommended.
3:	Method of Shchepetkin & McWilliams (2003) which is the most accurate one in case of significant bottom slopes, but at the expense of a larger computing time.
<code>iopt_sal</code>	Salinity update.
0:	Uniform (space and time) salinity field. Default.
1:	Salinity field initialised but not updated in time.
2:	Salinity field initialised and updated in time.
<code>iopt_temp</code>	Temperature update.
0:	Uniform (space and time) temperature field. Default.
1:	Temperature field initialised but not updated in time.
2:	Temperature field initialised and updated in time.

<code>iop_temp_optic</code>	Disables/enables the optical module.
	0: All solar radiation is assumed to be absorbed at the surface, i.e. the water column is considered as opaque. Not recommended (only used for idealised experiments).
	1: Solar radiation is absorbed within the water column using specified values for the attenuation depths. Default (recommended).
<code>iop_temp_sbc</code>	Type of surface boundary condition for temperature.
	1: Neumann condition using the model's surface heat flux formulations. Default (recommended).
	2: Dirichlet using prescribed surface temperatures taken at the first grid point below the surface.
	3: Dirichlet using prescribed surface temperature taken at the surface itself.

Remarks

- The switch `iop_dens` should be non-zero if either `iop_sal` or `iop_temp` is activated with a non-zero value.
- The values 2 and 3 for `iop_temp_sbc` can be used when the user wants to perform simulations with observed sea surface temperatures. However, there is no guarantee that this produces better results compared to the default case.

19.2.5 Sediment and tracers modules

<code>iop_biology</code>	Disables/enables the activation of the biological module.
	0: Disabled. Default.
	1: Enabled.
<code>iop_sed</code>	Selects the type of sediment module.
	0: Disabled. Default.
	1: Using the multi-fraction sediment transport module without flocculation.
	2: Using the flocculation module.
<code>iop_morph</code>	Disables/enables the morphological module.

- 0: Disabled. Default.
- 1: Enabled provided `iop_sed=1`.

`iop_part_write` Disables/enables the writing of physical data from a COHERENS run (without the particle module) to an output file which can be used for a subsequent off-line run with the particle module.

- 0: Disabled. Default.
- 1: Enabled.

Remarks

These switches must be defined in `usrdef_mod_params` (or the **CIF**) and not in the `usrdef` routines (or **CIF**) where the parameters specific to sediment, morphology (Chapter 23), biology (Chapter 24) and particle tracing (Chapter 25) modules are defined.

19.2.6 Advection

`iop_adv_scal` Type of scheme for the advection of scalar quantities.

- 0: Advection disabled.
- 1: Upwind scheme.
- 2: Lax-Wendroff (explicit) in the horizontal, central (semi-implicit) in the vertical. Not recommended.
- 3: TVD scheme. Default.

`iop_adv_turb` Type of scheme for the advection of turbulence quantities.

- 0: Advection disabled. Default. Highly recommended.
- 1: Upwind scheme.
- 2: Lax-Wendroff (explicit) in the horizontal, central (semi-implicit) in the vertical. Not recommended.
- 3: TVD scheme.

`iop_adv_tvd` Type of limiting function for the TVD scheme.

- 1: Superbee limiter. Default.
- 2: Monotone limiter.

<code>iop_adv_2D</code>	Type of scheme for the advection of 2-D depth-integrated currents. Only used in case the mode-splitting scheme is used (<code>iop_hydro_impl=0</code>) or for 2-D applications (<code>iop_grid_nodim=2</code>).
0:	Advection disabled.
1:	Upwind scheme. Default.
2:	Lax-Wendroff (explicit) in the horizontal. Not recommended.
3:	TVD scheme.
<code>iop_adv_3D</code>	Type of scheme for the advection of 3-D currents.
0:	Advection disabled
1:	Upwind scheme. Default.
2:	Lax-Wendroff (explicit) in the horizontal, central (semi-implicit) in the vertical. Not recommended.
3:	TVD scheme
<code>iop_scal_depos</code>	Type of discretisation for the vertical advective deposition flux at the sea bed. Only used for particulate matter (<code>iop_sed=1</code>).
0:	Deposition flux is set to zero.
1:	First order scheme. Default.
2:	Second order scheme using extrapolation.

Remarks

- The Lax-Wendroff/central scheme is non-monotone and should not be selected (except for numerical case studies such as the test case `cones`).
- The TVD scheme has the ability to retain sharp gradients, but consumes more CPU time compared to the upwind scheme. However, in the presence of fronts, the TVD scheme must be used to avoid excessive (numerical) diffusion.
- TVD is the recommended scheme for 3-D scalars, since it avoids excessive diffusion by the (faster) upwind scheme. The TVD scheme is recommended for currents in the case when 3-D currents occur with a high horizontal shear, as occurring in e.g. frontal zones.
- Advection of turbulence is considered of less importance than the production and dissipation terms in the k , $k-\varepsilon$ and $k-l$ transport equations. It is recommended not to change the zero default value of `iop_adv_turb`.

19.2.7 Horizontal diffusion

`iopthdif_coef` Type of formulation for the horizontal diffusion coefficients.

- 0: Horizontal diffusion is disabled. Default.
- 1: Spatially uniform.
- 2: Smagorinsky formulation (5.43) for momentum and (5.44) for scalars.

`iopthdif_lim` Selects type of limiting factor for scalar horizontal diffusion when `iopthdif_scal`=3.

- 0: Disabled.
- 1: NEMO limiting condition. Default.
- 2: [Gerdes *et al.* \(1991\)](#).

`iopthdif_scal` Type of horizontal diffusion schemes for scalars transport equations.

- 0: Disabled. Default.
- 1: Laplacian diffusion.
- 2: Diffusion along geopotential surfaces.
- 3: Isoneutral diffusion along isopycnal surfaces.

`iopthdif_skew` Disables/enables the addition of the [Griffies \(1998\)](#) skew diffusive flux to the diffusion matrix for eddy-induced transport.

- 0: Disabled. Default.
- 1: Enabled.

`iopthdif_turb` Disables/enables horizontal diffusion in the turbulence transport equations.

- 0: Disabled. Default (recommended).
- 1: Enabled.

`iopthdif_2D` Disables/enables horizontal diffusion in the 2-D current equations.

- 0: Disabled. Default.
- 1: Enabled.

`iopthdif_3D` Disables/enables horizontal diffusion in the 3-D current equations.

0: Disabled. Default.

1: Enabled.

iopc_kinvisc Formulation for kinematic viscosity.

0: User-defined uniform value **kinvisc_cst**. Default.

1: [ITTC \(1978\)](#) relation ([7.13](#)).

Remarks

- If horizontal diffusion is enabled, the Smagorinsky formulation, taken from LES modelling, is a more robust scheme compared to a constant diffusion coefficient. A uniform coefficient could be used for reducing unwanted numerical instabilities which may e.g. occur near open boundaries.
- Isopycnal diffusion is computationally more expensive compared to Laplacian or geopotential diffusion. The scheme is currently used for deep sea applications. Its effects on coastal sea applications are less known.
- Horizontal diffusion of turbulence variables is only introduced for historical reasons and compatibility with COHERENS V1, but is not recommended.
- The switches **iopc_hdif_2D**, **iopc_hdif_3D**, **iopc_hdif_scal**, **iopc_hdif_turb**, **iopc_hdif_lim** and **iopc_hdif_skew** are set to zero by the program if **iopc_hdif**=0.

19.2.8 Vertical diffusion

iopc_vdif_coef Selects the (general) type of vertical diffusion scheme.

0: Vertical diffusion disabled.

1: Uniform diffusion coefficient.

2: Algebraic formulation as described in Section [5.3.2.2](#).

3: Second-order turbulence closure as described in Section [5.3.3](#). Default.

iopc_vdif_rot Disables/enables the addition of a vertical rotated diffusion term to the K_{33} -term in the diffusion tensor given by ([5.194](#)) when **iopc_hdif_scal**=2,3.

0: Disabled when **iopc_hdif_scal**=1. Should only be used for numerical experiments.

1: Enabled. Default (recommended) when **iopc_hdif_scal**=2,3.

19.2.9 Turbulence schemes

<code>iopt_turb_alg</code>	Type of algebraic scheme if <code>iopt_vdif_coef</code> = 2.
	1: Pacanowski-Philander formulation (5.93)–(5.96). Default.
	2: Munk-Anderson formulation (5.97)–(5.101).
	3: Flow dependent formulation as described in Section 5.3.2.2 with α given by (5.109).
	4: Flow dependent formulation as described in Section 5.3.2.2 with α given by (5.110).
	5: Flow dependent formulation as described in Section 5.3.2.2 with α given by (5.111).
	6: Parabolic profile (5.115).
<code>iopt_turb_dis_bbc</code>	Type of bottom boundary condition for the dissipation rate ε .
	1: Neumann condition (5.332).
	2: Dirichlet condition (5.330). Default.
<code>iopt_turb_dis_sbc</code>	Type of surface boundary condition for the dissipation rate ε .
	1: Neumann condition (5.248).
	2: Dirichlet condition (5.245). Default.
<code>iopt_turb_iwlim</code>	Type of background mixing scheme as described in Section 5.3.3.6.
	0: Using uniform background coefficients. Default.
	1: Using limiting conditions for turbulence parameters.
	2: The Large <i>et al.</i> (1994) scheme given by (5.188)–(5.189).
<code>iopt_turb_kinvisc</code>	Selects type of background mixing.
	0: User-defined constant value <code>vdifmom_cst</code> . Default.
	1: Kinematic viscosity as selected by <code>iopt_kinvisc</code> .
<code>iopt_turb_lmix</code>	Mixing length formulation as described in Section 5.3.3.5.
	1: Parabolic law (5.174).
	2: “Modified” parabolic law (5.175).

	3: “Xing” formulation (5.176).
	4: “Blackadar” asymptotic formulation (5.177). Default.
<code>iopt_turb_ntrans</code>	Number of transport equations as described in Section 5.3.3.4.
	0: Zero-equation model (equilibrium or Mellor-Yamada level 2 method) with a mixing length selected by <code>iopt_turb_lmix</code> .
	1: Turbulence energy equation with a mixing length selected by <code>iopt_turb_lmix</code> . Default.
	2: $k-\varepsilon$ of $k-kl$ equation depending on the value of <code>iopt_turb_param</code> .
<code>iopt_turb_param</code>	Selects type of second turbulence variable.
	1: Mixing length l ($k-l$ scheme).
	2: Dissipation rate ε ($k-\varepsilon$ scheme). Default.
<code>iopt_turb_stab_form</code>	Selects type of stability function.
	1: Constant value (5.159).
	2: Munk-Anderson form (5.160).
	3: From RANS model as explained in Section 5.3.3.3. Default.
<code>iopt_turb_stab_lev</code>	Selects level for stability functions if <code>iopt_turb_stab_form</code> =3.
	1: Quasi-equilibrium method (Section 5.3.3.3). Default.
	2: Non-equilibrium method (Section 5.3.3.3).
<code>iopt_turb_stab_mod</code>	Selects type of closure (RANS) model.
	1: MY82-model (Mellor & Yamada, 1982).
	2: KC94-model (Kantha & Clayson, 1994).
	3: BB95-model (Burchard & Baumert, 1995).
	4: HR82-model (Hossain & Rodi, 1982). Default.
	5: CA01-model (Canuto <i>et al.</i>, 2001).
	6: CA02-model (Canuto <i>et al.</i>, 2001).
<code>iopt_turb_stab_tke</code>	Formulation for the turbulent diffusion coefficient ν_k (or stability coefficient S_k) of turbulent energy.
	1: Constant value for S_k as given by equation (5.161).

	2: S_k is taken as proportional to momentum stability function S_u as given by (5.162). Default.
	3: Using the formulation of Daly & Harlow (1970) as given by (5.146) or (5.152) depending on the value of <code>iopt_turb_stab_lev</code> .
<code>iopt_turb_tke_bcc</code>	Type of bottom boundary condition for turbulence energy. 1: Neumann condition (5.331). 2: Dirichlet condition (5.330). Default.
<code>iopt_turb_tke_sbc</code>	Type of surface boundary condition for turbulence energy. 1: Neumann condition (5.247). Default. 2: Dirichlet condition (5.245). Default.

19.2.10 Bottom boundary conditions

<code>iopt_bstres_drag</code>	Formulation for the bottom drag coefficient C_{db} . 0: Not used. 1: Spatially uniform value. 2: User-defined spatially non-uniform value. 3: Using a spatially uniform roughness length. Default. 4: User-defined spatially non-uniform roughness length. 5: As sum of grain roughness, wave ripple roughness and bedload roughness (see equation (7.17)). This option is only available when the sediment transport module is activated (<code>iopt_sed=1</code>).
<code>iopt_bstres_form</code>	Type of formulation for the bottom stress. 0: Bottom stress is set to zero. 1: Linear bottom stress law (5.317) or (5.318). Only used for idealistic test cases. 2: Quadratic bottom stress (5.319) or (5.318). Default (recommended).
<code>iopt_bstres_nodim</code>	Type of currents used in the bottom stress formulation.

2: Depth-mean currents. Used in case of a 2-D simulation.

3: 3-D current taken at the bottom grid cell. Default in case of a 3-D simulation (recommended).

iopt_bstres_waves_bfric Selects the type of wave-current interaction scheme for the bottom stress. Only allowed if **iopt_waves**=1.

0: Disabled. Default.

1: Soulsby & Clarke (2005) model.

2: Malarkey & Davies (2012) model.

3: Grant & Madsen (1979, 1986) model.

4: Christofferson & Jonsson (1985) model.

19.2.11 Surface meteo

iopt_meteo Disables/enables meteorological input.

0: Disabled. In that case all surface fluxes for T, S and all switches below are set to zero. Default.

1: Enabled.

iopt_meteo_data Type of meteo input.

1: Surface winds, atmospheric pressure, air temperature, relative humidity, cloud cover. Default.

2: Surface wind stress, atmospheric pressure, non-solar and solar heat flux.

iopt_meteo_heat Disables/enables the input of data used for the surface heat flux.

0: Disabled. Default if **iopt_temp**=0.

1: Enabled. Default unless **iopt_temp**=0.

iopt_meteo_pres Disables/enables the input of atmospheric pressure.

0: Disabled.

1: Enabled. Default unless in the case of a 1-D application or when the surface data are restricted to one location.

iopt_meteo_precip Determines the type of input for precipitation and evaporation.

- 0: No input. Default.
 - 1: Precipitation rate only. Evaporation rate is obtained from (5.229).
 - 2: Evaporation minus precipitation rate.
- iopt_meteo_stres** Disables/enables the input of surface winds or surface stress components.
- 0: Disabled.
 - 1: Enabled. Default.

Remarks

Note that all meteorological surface forcing is disabled if `iop_meteo=0`. This means that all surface fluxes are automatically set to zero and the input of any meteorological data is disabled.

19.2.12 Surface fluxes

- iopt_sflux_pars** Formulation for the (neutral) surface drag and heat exchange coefficients.
- 1: Generic case given by (5.249). Default.
 - 2: [Large & Pond \(1981\)](#).
 - 3: [Smith & Banke \(1975\)](#).
 - 4: [Geernaert *et al.* \(1986\)](#).
 - 5: [Kondo \(1975\)](#).
 - 6: [Wu \(1980\)](#).
 - 7: Using Charnock's relation as given by (5.255).
 - 8: North Sea formulation as given by (5.256).
 - 9: [Large & Yeager \(2004\)](#).
- iopt_sflux_precip** Selects how the precipitation minus evaporation rates are used provided that `iopt_meteo_precip>0`.
- 0: Disabled. Default.
 - 1: As surface flux for salinity.
 - 2: As source or sink term in the 2-D continuity equation.
 - 3: As surface flux for salinity and source or sink term in the 2-D continuity equation.

`iopt_sflux_qlong` Selects the formulation for the upwards surface long-wave radiation.

- 1: [Brunt \(1932\)](#). Default.
- 2: [Clark *et al.* \(1974\)](#).
- 3: [Bignami *et al.* \(1995\)](#).

`iopt_sflux_qshort` Selects the formulation for the downward surface short-wave radiative flux.

- 1: [Rosati & Miyakoda \(1988\)](#). Default.
- 2: [Zillman \(1972\)](#).
- 3: [Shine \(1984\)](#).

`iopt_sflux_strat` Disables/enables the dependence of surface drag and exchange coefficients on atmospheric stratification effects using Monin-Obukhov similarity theory (see Section 5.7.3).

- 0: Disabled. Default.
- 1: Enabled.

19.2.13 Open boundary conditions

`iopt_obic_advrlx` Disables/enables the relaxation scheme for horizontal momentum advection.

- 0: Relaxation scheme disabled. Default.
- 1: Relaxation scheme enabled. In that case the parameter `distrlx_obic`, representing the parameter d_{max} in (12.295) must be defined by the user.

`iopt_obic_bio` (General) type of open boundary conditions for biological state variables.

- 0: Default conditions at all open boundaries. Default.
- 1: A non-default condition is at least applied at one open boundary location.

`iopt_obic_invbar` Disables/enables the inverse barometric effect at open boundaries as given by (5.335).

- 0: Disabled. Default.
- 1: Enabled.

iopt_abc_sal	(General) type of open boundary conditions for salinity. 0: Default conditions at all open boundaries. Default. 1: A non-default condition is at least applied at one open boundary location.
iopt_abc_sed	(General) type of open boundary conditions for sediments. 0: Default conditions at all open boundaries. Default. 1: A non-default condition is at least applied at one open boundary location.
iopt_abc_temp	(General) type of open boundary conditions for temperature. 0: Default conditions at all open boundaries. Default. 1: A non-default condition is at least applied at one open boundary location.
iopt_abc_th	Disables/enables the use of the Thatcher & Harleman (1972) open boundary condition. 0: Disabled. Default. 1: Enabled.
iopt_abc_2D	(General) type of open boundary conditions for the 2-D mode. 0: Default conditions at all open boundaries. Default. 1: A non-default condition is at least applied at one open boundary location.
iopt_abc_2D_tang	(General type) of tangential open boundary conditions for the 2-D mode. 0: Default conditions at all open boundaries. Default. 1: A non-default condition is at least applied at one open boundary location.
iopt_abc_3D	(General) type of open boundary conditions for 3-D currents. 0: Default conditions at all open boundaries. Default. 1: A non-default condition is at least applied at one open boundary location.

iopt.obc_3D_tang (Gnral) type of tangential open boundary conditions for the 3-D mode.

- 0: Default conditions at all open boundaries. Default.
- 1: A non-default condition is at least applied at one open boundary location.

Remarks

Note that, if the appropriate switch is not set, the open boundary conditions for that specific variable automatically reduce to their defaults (see Section 5.10) and the input of open boundary data is disabled.

19.2.14 Nesting

iopt_nests Disables/enables the writing of open boundary data for nested sub-grids.

- 0: Disabled. Default.
- 1: Enabled. In that case the parameter **nonestsets** must be defined by the user.

19.2.15 1-D applications

iopt_sbc_1D Selects type of surface forcing data in case of a 1-D water column application when **iopt_sur_1D**=0.

- 0: No surface forcing applied. Default.
- 1: Surface slopes and elevation.
- 2: Surface elevation only.
- 3: Surface slopes only.

iopt_sur_1D Disables/enables surface forcing (surface slopes and elevations) in case of a 1-D (**iopt_grid_nodim**=1) water column application.

- 0: Disabled. Default.
- 1: Enabled.

19.2.16 Tides

iopt_astro_pars Type of formulation for evaluating the astronomical phases $V_{qn}(t)$.

- 0: Not used.
- 1: Using the linear format (5.334) with user-defined initial phases. Should be used for special cases only, but is not recommended in general.
- 2: Using (5.210) with t_{ref} either defined initially with no further updates or updated at a time interval of days given by the parameter `noastrodays` selected by the user. Default.

`iopt_astro_tide` Disables/enables the inclusion of the astronomical tidal force in the momentum equations. This requires that a spherical grid is used (`iopt_grid_sph=1`).

- 0: Disabled. Default.
- 1: Enabled.

19.2.17 Surface waves

<code>iopt_waves</code>	Disables/enables surface wave effects.
	<ul style="list-style-type: none"> 0: Disabled. Default. 1: Enabled.
<code>iopt_waves_couple</code>	Determines the type of coupling between COHERENS and SWAN.
	<ul style="list-style-type: none"> 0: Disabled. Default. 1: One-way coupling with COHERENS receiving data from SWAN. 2: One-way coupling with COHERENS sending data to SWAN. 3: Two-way coupling.
<code>iopt_waves_curr</code>	Disables/enables the inclusion of the Stokes drift in the momentum equations.
	<ul style="list-style-type: none"> 0: Disabled. Default. 1: Enabled.
<code>iopt_waves_dissip</code>	Disables/enables the inclusion of the wave dissipation terms in the momentum equations.
	<ul style="list-style-type: none"> 0: Disabled. Default. 1: Enabled.

<code>iopt_waves_extrapol</code>	Disables/enables extrapolation if the model or wave data are located at grid points outside the model or data domain.
	0: Disabled.
	1: Enabled. Default.
<code>iopt_waves_form</code>	Selects how the wave parameters are obtained.
	1: Using significant wave height, peak wave number and wave direction. Default.
	2: As random waves, obtained from the wave model by integration over the wave spectrum.
<code>iopt_waves_pres</code>	Disables/enables the inclusion of the static sea level term in the momentum equations.
	0: Disabled.
	1: Enabled. Default.

19.2.18 Drying/wetting scheme

<code>iopt fld</code>	Selects the type of drying/wetting scheme.
	0: Drying/wetting disabled. Default.
	1: Drying/wetting enabled.
<code>iopt fld alpha</code>	Type of formulation for the α -factor used in the drying/wetting algorithm for momentum advection.
	0: No reduction applied ($\alpha = 1$).
	1: Using the power law given by (11.29a)–(11.29c). Default.
	2: Using the asymptotic law (11.30).
<code>iopt fld crit</code>	Disables/enables the use of a mask criterion.
	0: Disabled.
	1: Enabled. Default.

19.2.19 Structures

<code>iopt_dischr</code>	Disables/enables the discharge module.
	0: Disabled. Default.

1: Enabled.

iopt_dischr_land Selects the procedure for discharges in case that the discharge is located on a land cell.

- 1: The discharge is ignored at flagged land points. Default.
- 2: Discharge location is moved to the nearest wet point on the model grid.

iopt_drycel Disables/enables the dry cell module.

- 0: Disabled. Default.
- 1: Enabled.

iopt_thndam Disables/enables the thin dam module.

- 0: Disabled. Default.
- 1: Enabled.

iopt_weibar Disables/enables the weirs/barriers module.

- 0: Disabled. Default.
- 1: Enabled.

19.2.20 Numerical switches

multi-grid scheme

iopt_mg_cycle Type of cycle used in the multi-grid procedure.

- 1: V-cycle. Default.
- 2: W-cycle.

iopt_mg_prolong Type of multi-grid prolongation operator.

- 1: Injection.
- 2: Bilinear interpolation. Default.

iopt_mg_smoothen Type of smoother for the multi-grid scheme.

- 1: Jacobi. Default.
- 2: Gauss-Seidel with Red-Black ordering.

other numerical switches

`iopf_cor_impl` Time-integration of the Coriolis terms.

- 0: Explicit.
- 1: Semi-implicit. Default.
- 2: Implicit.

`iopf_vadv_impl` Time-integration of vertical advective terms.

- 0: Explicit.
- 1: Semi-implicit. Default.
- 2: Implicit.

`iopf_vdif_impl` Time-integration of vertical diffusion terms.

- 0: Explicit.
- 1: Semi-implicit.
- 2: Implicit. Default.

Remarks

It is advised (except for expert users) not to change the defaults of these switches since they have an impact on the accuracy and stability of the numerical schemes.

19.2.21 MPI mode

`iopf_MPI_abort` 0: If an error is detected in a MPI routine, an error message will be written, but the program will not abort immediately. Default.

- 1: If an error is detected in a MPI routine, an error message will be written and the program will abort immediately afterwards.

`iopf_MPI_comm_all` Communication type for “all to all” operations.

- 1: Blocking, standard send.
- 2: Blocking, synchronous send. Default.

`iopf_MPI_comm_coll` Disables/enables the use of MPI collective calls.

- 0: Disabled. Default.
- 1: Enabled.

iopt_MPI_comm_exch Communication type for “exchange” operations.

- 1: Blocking, standard send.
- 2: Blocking, synchronous send. Default.

iopt_MPI_comm_gath Communication type for “all to one” gather (combine) operations.

- 1: Blocking, standard send.
- 2: Blocking, synchronous send. Default.

iopt_MPI_comm_scat Communication type for “one to all” scatter (distribute and copy) operations.

- 1: Blocking, standard send.
- 2: Blocking, synchronous send. Default.

iopt_MPI_partit Selects the method for domain decomposition.

- 1: “Simple” partition based on the values of **nprocsx** and **nprocsy**. Default.
- 2: Decomposition defined by the user in routine **usrdef_partition** or obtained from an external data file in standard format.

iopt_MPI_sync Disables/enables synchronisation calls at the end of a series of blocking or non-blocking operations.

- 0: Disabled. Default.
- 1: Enabled.

Remarks

- Except for the switches **iopt_MPI_abort** intended for debugging and **iopt_MPI_partit** for externally defined domain decompositions, the other MPI switches are useful only for testing different kind of options for communications by the developers and expert users.
- Synchronisation of communication calls may lower the CPU performance.

19.2.22 User output

`iopt_out_anal` Disables/enables harmonic output.

- 0: Disabled. Default.
- 1: Enabled.

`iopt_out_avrgd` Disables/enables time averaged output.

- 0: Disabled. Default.
- 1: Enabled.

`iopt_out_tsers` Disables/enables time series output.

- 0: Disabled.
- 1: Enabled. Default.

19.2.23 NetCDF

`iopt_CDF_abort` 0: If an error is detected in a `netCDF` routine, an error message will be written, but the program will not abort immediately. Default.

- 1: If an error is detected in a `netCDF` routine, an error message will be written and the program will abort immediately afterwards.

`iopt_CDF_fill` Disables/enables the use of fill values.

- 0: Disabled. Default.
- 1: Enabled.

`iopt_CDF_format` Selects the type of `netCDF` file format.

- 1: Classic format. Default.
- 2: 64-bit offset format.

`iopt_CDF_shared` Disables/enables sharing of data between writing and reading processes in the same `netCDF` file.

- 0: Disabled. Default.
- 1: Enabled.

`iopt_CDF_sync` Disables/enables synchronisation of the disk copy of a `netCDF` user output file with in-memory buffers by calling `NF90_SYNC` before each read and after each write. Advantage is that data loss is minimised in case of abnormal termination. Disadvantage is a larger execution time.

0: Disabled. Default.

1: Enabled.

iopt_CDF_tlim Sets the type of time dimension in user output **netCDF** files.

1: Fixed value. Data loss is prevented in case of abnormal termination. Default.

2: Unlimited time dimension. Advantage is that additional time records can be appended after completion of the run. Disadvantage is that data loss is not prevented in case of abnormal termination (unless **iopt_CDF_sync** is set).

Remarks

The default of **iopt_CDF_format** limits the size of the **netCDF** file, but has the advantage that it is portable for all versions of **netCDF**. The 64-bit offset format no longer poses an upper limit for file size, provided that a fixed time dimension is selected with **iopt_CDF_tlim=1** and is recommended for **netCDF** version 4.0 and later.

The different **netCDF** file formats and other utilities are discussed in the **netCDF** User Manual.

19.2.24 Various

iopt_rng_seed Defines the type of initial seed for all random generators.

- 1: The same default seed is used for all generators (the same sequence of random numbers is produced for all simulations). Default.
- 2: The initial seeds are generated depending on the current date and time (different runs produce different series of random numbers).

19.2.25 Internal switches

A number of switches are defined internally and cannot be (re)-defined by the user.

iopt_CDF Disables/enables the use of the **netCDF** library. This switch is defined internally with the **-DCDF** compiler option.

<code>iopt_MCT</code>	Disables/enables the use of the MCT library. This switch is defined internally with the <code>-DMCT</code> compiler option.
<code>iopt_MPI</code>	Disables/enables parallel processing with MPI. This switch is defined internally with the <code>-DMPI</code> compiler option. Is set to zero in case of serial applications (<code>npworld=1</code>).
<code>iopt_mode_2D</code>	Type of 2-D mode calculations.
<code>iopt_mode_3D</code>	Type of 3-D mode calculations.

19.3 Model parameters

All parameters in this section are defined in `usrdef_mod_params` or in the CIF file block `mod_params`.

19.3.1 Process numbers for the domain decomposition

`INTEGER :: nprocsx, nprocsy`

The parameters below are used to set up a simple domain decomposition.

`nprocsx` X-dimension of the decomposed domain. Default is 0.

`nprocsy` Y-dimension of the decomposed domain. Default is 0.

The process numbers `nprocsx` and `nprocsy` are needed by the program for making a “simple” domain decomposition when the switch `iopt_MPI_partit = 1`. In the alternative case, i.e. when `iopt_MPI_partit=2`, these parameters are determined by the program and don’t need to be defined.

The following cases are allowed

- `nprocsx` is non-zero and a divisor of `nprocscoh` whereas `nprocsy` is initially set to zero. In that case

`nprocsy=nprocscoh/nprocsx`

- `nprocsy` is non-zero and a divisor of `nprocscoh` whereas `nprocsx` is initially set to zero. In that case

`nprocsx=nprocscoh/nprocsy`

- Both `nprocsx` and `nprocsy` are zero (default case). In that case `nprocsx` and `nprocsy` are defined internally as divisors of `nprocscoh` such that $ABS(nprocsx-nprocsy)$ is minimal and

$$nprocsx*nprocsy = nprocscoh$$

To make an optimum domain decomposition it is recommended to select a value for `nprocscoh` which is not a prime number and has a sufficient number of divisors. Selecting a non-zero value for `nprocsx` or `nprocsy` can be useful for special cases such as channel flows where one of the coordinate axes is aligned with the along-channel (say X-) direction. A proper choice then is to take `nprocsx=nprocscoh` so that `nprocsy=1`.

19.3.2 Grid parameters

```
INTEGER :: nc, nosbu, nosbv, nr, nrvbu, nrvbv, nz
REAL :: b_SH, depmean_cst, depmean_flag, dlat_ref, dlon_ref,&
       & dl_BB, du_BB, hcrit_BB, hcrit_SH, sigstar_DJ, sig0_DJ, theta_SH
```

integer parameters

- nc** Number of grid cells of the computational domain in the X-direction, including a “dummy” extra column at the eastern edge. This means that the actual physical size of the model grid in the Y-direction equals `nc-1`. Must always be defined. No default.
- nr** Number of grid cells in the Y-direction of the computational domain, including a “dummy” extra row at the northern edge. This means that the actual physical size of the model grid in the X-direction equals `nr-1`. Must always be defined. No default.
- nosbu** Number of open sea boundaries at West/East U-nodes. Must be defined unless its value is zero (default).
- nosbv** Number of open sea boundaries at South/North V-nodes. Must be defined unless its value is zero (default).
- nrvbu** Number of river open boundaries at West/East U-nodes. Must be defined unless its value is zero (default).
- nrvbv** Number of river open boundaries at South/North V-nodes. Must be defined unless its value is zero (default).
- nz** Number of grid cells in the vertical direction. Must always be defined except for 2-D applications.

Remarks

- **nc** and **nr** must be positive and are automatically (re)set to 3 for water column applications (**iopt_grid_nodim**=1).
- **nz** must be positive and is automatically (re)set to 1 for 2-D applications (**iopt_grid_nodim**=2).

19.3.2.1 Bathymetry and geographical location

- depmean_cst** If its value is different from the flagged value **float_fill**, it is used to set up a uniform bathymetry [m]. Otherwise, the bathymetry is defined by the user (see Section 20.1). Default is **float_fill**.
- depmean_flag** Flag used to identify (permanent) land cells in the bathymetry. Default is **float_fill**.
- dlat_ref** Reference latitude to be used for the Coriolis frequency in the case of a Cartesian grid [degrees latitude, positive North]. Default is 0.0.
- dlon_ref** Reference longitude to be used for solar irradiance in the case of a Cartesian grid [degrees longitude, positive East]. Default is 0.0.

19.3.2.2 Grid transformation

- b_SH** Parameter b used in the s -coordinate formulation given by (4.35). Default is 0.1.
- dl_BB** Parameter d_l in the [Burchard & Bolding \(2002\)](#) vertical grid transformation (4.26). Default is 1.5.
- du_BB** Parameter d_u in the [Burchard & Bolding \(2002\)](#) vertical grid transformation (4.26). Default is 1.5.
- hcrit_BB** Critical water depth h_{crit} used in the s -coordinate formulation given by (4.39) [m]. Default is 200.0.
- hcrit_SH** Critical water depth h_{crit} used in the s -coordinate formulation given by (4.35) [m]. Default is 200.0.
- sigstar_DJ** Parameter σ_* in the [Davies & Jones \(1991\)](#) vertical grid transformations (4.23) and (4.24). Default is 0.0.
- sig0_DJ** Parameter σ_0 in the [Davies & Jones \(1991\)](#) vertical grid transformations (4.23) and (4.24). Default is 0.1.

theta_SH Parameter θ used in the s -coordinate formulation given by (4.35). Default is 8.0.

19.3.3 Date and time parameters

CHARACTER (LEN=lentime) :: CEndDateTime, CStartTime	
INTEGER :: iccvt, ic3d, maxwaitseconds, norestarts, nowaitsecs	
INTEGER, DIMENSION(norestarts) :: ntrestart	
REAL :: timestep, time_zone	
CStartTime	Start date of the simulation given as a 23-character string (YYYY/MM/DD;HH:MM:SS:mmm). If the last 4 characters are omitted they will be set to ':000'. Must always be defined. No default.
CEndDateTime	End date of the simulation given as a 23-character string (YYYY/MM/DD;HH:MM:SS:mmm). If the last 4 characters are omitted they are set to ':000'. Must always be defined. No default.
iccvt	Time counter (number of base time steps) used in the convective adjustment scheme. Must be defined if <code>iopt_dens_convect=1</code> . No default.
ic3d	Number of base time steps within one 3-D time step. Must always be defined in case the mode-splitting scheme is used. If <code>iopt_hydro_impl=1</code> or <code>iopt_grid_nodim=1</code> or 2, <code>ic3d</code> is always (re)set to 1. Default is 1.
maxwaitsecs	Maximum allowed time spent in wait calls (see Section 19.4.1) [s]. Default is 3600.
norestarts	The number of times during the simulation that initial condition data are written for (eventual) restart runs. Value is limited by the system parameter <code>MaxRestarts</code> defined in <code>syspars.f90</code> . No default.
nowaitsecs	Number of seconds to wait between two read attempts (see Section 19.4.1) [s]. No default.
ntrestart	Time step indices indicating the time during the simulation when restart data are to be written. If the user sets a value to the flag <code>int_fill</code> , it will automatically be replaced by the total number of base time steps <code>nstep</code> . No default.
timestep	Smallest (“base”) time step used in the model. Equal to the 2-D or the 3-D time step depending whether the mode-splitting scheme or the implicit scheme is used [s]. All

	other time steps in COHERENS are defined as multiples of timestep . Must always be defined. No default.
time_zone	Time zone, i.e. the difference of the local time with respect to GMT [hours]. Difference is positive (negative) eastwards (westwards) from Greenwich. Default is 0.0, which means that the date and time are given in GMT.

Remarks

- If **timestep** is lower than 1000 seconds, its precision is 1 millisecond and decimal numbers from the fourth position after the decimal point will be discarded. If **timestep** is larger than 1000 seconds, its precision is 1 second and its decimal part is ignored. If **iopt_hydro_impl**=0, the base time step is limited by the CFL condition (12.1) for surface gravity waves. The value of the limiting time step is written to the “log” file. Note that there is also a time limit for the 3-D time step (see equations (12.3) and (12.2)). Contrary to the 2-D limit, the 3-D CFL limit depends on the currents and is therefore difficult to determine *a priori*.
- The parameter **time_zone** is of type **REAL** and must be between -12.0 and 12.0 and is used to reset the start and end dates within the code to GMT where necessary. The time zone is needed for the calculation of solar radiance and the astronomical Greenwich argument where the local time must be converted to GMT (if **time_zone** is non-zero).
- It is clear that **ic3d** only needs to be defined for 3-D applications and with mode splitting (**iopt_grid_nodim**=3, **iop_hydro_impl**=0). Note that the 3-D time step is limited by the constraints (12.2), (12.3).

19.3.4 Parameters for diffusion

```
REAL :: hdifmom_cst, hdifscal_cst, kinvisc_cst, rho_crit_iso, &
& skewdiff_cst, slopemax_iso, smag_coef_mom, smag_coef_scal, &
& vdifmom_cst, vdifscal_cst
```

hdifmom_cst	Spatially uniform coefficient ν_H for horizontal diffusion of currents when iop_hdif_coef =1 [m ² /s]. No default.
hdifscal_cst	Spatially uniform coefficient λ_H for horizontal diffusion of scalars when iop_hdif_coef =1 [m ² /s]. No default.
kinvisc_cst	Spatially uniform kinematic viscosity coefficient when iop_kinvisc =0 [m ² /s]. Default is 10 ⁻⁶ .

<code>rho_crit_iso</code>	Critical density ρ_{crit} used to determine the surface mixed layer depth and isopycnal slopes in Section 5.4.6. Used when <code>iopt_hdif_scal=3</code> . Default is 0.01.
<code>skewdiff_cst</code>	Parameter κ_{GM} used in the Griffies (1998) scheme for the skew-diffusive flux (see Section 5.4.5) [m^2/s]. Used when <code>iopt_hdif_skew=1</code> . Default is 500.0.
<code>sloemax_iso</code>	Upper limit for the magnitude of the slope vector components S_i as used in the rotated diffusion tensor formulation (see Section 5.4.1). Used when <code>iopt_hdif_scal=3</code> . Default is 0.01.
<code>smag_coef_mom</code>	Coefficient C_m used in the Smagorinsky expression (5.43) for the scalar horizontal diffusion coefficient. Default is 0.1
<code>smag_coef_scal</code>	Coefficient C_s used in the Smagorinsky expression (5.44) for the scalar horizontal diffusion coefficient. Default is 0.1.
<code>vdifmom_cst</code>	Spatially uniform coefficient ν_T for vertical diffusion of currents when <code>iopt_vdif_coef=1</code> or as background value if <code>iopt_turb_iwlim=0</code> [m^2/s]. Default is 10^{-6} .
<code>vdifscal_cst</code>	Spatially uniform coefficient λ_T^ψ for vertical diffusion of scalars when <code>iopt_vdif_coef=1</code> or as background value if <code>iopt_turb_iwlim=0</code> [m^2/s]. Default is 10^{-6} .

19.3.5 Tidal parameters

```
INTEGER :: noastrodays, nconastro, nconobc
INTEGER, DIMENSION(nconastro) :: index_astro
INTEGER, DIMENSION(nconobc) :: index_obc
REAL :: dlon_ref_tides_anal, dlon_ref_tides_obc
```

<code>dlon_ref_tides_anal</code>	If <code>iopt_astro_pars=2</code> , harmonically analysed phases are taken with respect to the astronomical phases at the reference longitude λ_r^a (see Section 5.12.1) [decimal degrees, positive East]. Default is zero, meaning that the reference longitude is taken at Greenwich.
<code>dlon_ref_tides_obc</code>	If <code>iopt_astro_pars=2</code> , phases at open boundaries are assumed to be taken with respect to the astronomical phases at the reference longitude λ_r^o (see Section 5.10.1) [degrees longitude, positive East]. Default is zero, meaning that the reference longitude is taken at Greenwich.

index_astro	Key ids of the tidal constituents used for the astronomical tidal forcing. Their FORTRAN names are icon_ followed by the name of the constituent, e.g. icon_M2 for the M_2 tide. A full listing of all possible constituents is given in <i>tide.f90</i> . Values are between 1 and MaxAstroTides . Must be defined if iopt_astro.tide=1 . No default.
index_obc	Key ids of the tidal constituents used for the tidal forcing at open boundaries. Their FORTRAN name is icon_ followed by the name of the constituent, e.g. icon_M2 for the M_2 tide. A full listing of all possible constituents is given in <i>tide.f90</i> . Values are between 1 and MaxConstituents . Must be defined if nconobc>0 . No default.
nconastro	Number of tidal constituents used for astronomical forcing if iopt_astro.tide=1 . No default.
noastrodays	Time step (in days) for updating the reference time t_{ref} in equation (5.210) for the astronomical phases. If zero, t_{ref} is only updated at the initial time. Default is 0.
nconobc	Number of tidal constituents used for the open boundary forcing if iopt_grid_nodim>1 (see (5.333)) or for the surface forcing if iopt_grid_nodim=1 (see (5.70)).

19.3.6 Open boundary conditions and nesting

```
INTEGER :: nonestsets, nqsecobu, nqsecobv, ntobcrlx
REAL :: distrlx_obc
REAL, DIMENSION(nz) :: return_time
```

distrlx_obc	Maximum distance d_{max} used to calculate the α_{or} factor in (12.295) if the relaxation scheme for momentum advection is enabled (iopt_abc_advrlx=1) [m]. No default.
nonestsets	Number of nested sub-grids used when iopt_nests=1 . No default.
nqsecobu	Number of sections along U-boundaries where a distributed discharge condition is specified (i.e. when ityp2dobu equals 16).
nqsecobv	Number of sections along V-boundaries where a distributed discharge condition is specified (i.e. ityp2dobv equals 16).
ntobcrlx	The relaxation period T_r , measured in units of the base time step timestep, used to define the time-relaxation factor $\alpha_r(t)$ in (5.336). If zero, T_r is zero and relaxation is disabled. For details see Section 5.10.1. Default is zero.

return_time Vertical profile of the return times T_{ret} as used in (5.378) when a Thatcher-Harleman open boundary condition is applied [s]. No default.

19.3.7 Bottom boundary conditions

REAL :: bdragcoef_cst, bdraglin, zbtoz0lim, zrough_cst

bdragcoef_cst Constant bottom drag coefficient C_{db} when **iopt_bstres_drag=1**.
No default.

bdraglin Bottom friction velocity k_{lin} used in the linear bottom friction law if **iopt_bstres_form=1** [m/s]. No default.

zbtoz0lim Value of the limiting ratio ξ_{min} for z_b/z_0 (see Section 5.9.2). Default is 2.0.

zrough_cst Constant bottom roughness length z_0 when **iopt_bstres_drag=3** [m]. No default.

19.3.8 Surface boundary conditions

REAL :: ccharno, ces_scst, ces_ucst, chs_scst, chs_ucst, &
& zref_temp, zref_wind
REAL, DIMENSION(4) :: cdspars

ccharno Charnock's constant a_{ch} used in Charnock's relation (5.255)). Default is 0.014.

cdspars The parameters (c_1, c_2, c_3, c_4) used in the generic formulation (5.249) for the surface drag coefficient. Defaults are (0.0,0.00113,0.0,0.0).

ces_scst Constant value C_e^s used in some formulations for the exchange coefficient C_e in case of a stable stratification. Default is 0.00113.

ces_ucst Constant value C_e^u used in some formulations for the exchange coefficient C_e in case of an unstable stratification. Default is 0.00113.

chs_scst Constant value C_h^s used in some formulations for the exchange coefficient C_h in case of a stable stratification. Default is 0.00113.

chs_ucst Constant value C_h^u used in some formulations for the exchange coefficient C_h in case of an unstable stratification. Default is 0.00113.

zref_temp Height z_r^t above the sea surface where air temperature and relative humidity are provided by the meteo data [m]. Used only when **iopt_sflux_strat=1** [m]. Default is 10.0.

zref_wind Height z_r^w above the sea surface where wind velocities are provided by the meteo data [m]. Used only when `iopt_sflux_strat=1` [m]. Default is 10.0.

19.3.9 Reference values for physical variables

```
REAL :: atmpres_ref, gacc_ref, rho_air, sal_ref, specheat, &
& temp_min, temp_ref
```

atmpres_ref Reference value for the atmospheric pressure P_{ref} as used in the formulation (5.335) for the inverse barometer effect (`iopt_obc_invbar=1`) [Pa]. Default is 101325.0.

gacc_ref If its value is different from the flagged value `float_fill`, the acceleration of gravity, taken as horizontally uniform. Otherwise, g is evaluated as function of latitude using (5.16) [m/s²]. Default is `float_fill`.

rho_air Air mass density ρ_a [kg/m³]. Default is 1.2.

sal_ref Reference salinity S_{ref} used as the default value for the salinity field if `iopt_sal=0`, in the linear equation of state (5.67), as default initial condition or to calculate the reference density ρ_0 from the equation of state [PSU]. Default is 33.0.

specheat Specific heat of seawater c_p at constant pressure [J/kg/degC]. Default is 3987.5.

temp_min Minimum value used for temperature. If set to the flagged value `float_fill`, the minimum is taken as the freezing point of sea water instead (see equation (5.7)) [°C]. Default is 0.0.

temp_ref Reference temperature T_{ref} used as default value for the temperature field if `iopt_temp=0`, in the linear equation of state (5.67), as default initial condition or to calculate the reference density ρ_0 from the equation of state [°C]. Default is 12.0.

19.3.10 Optical parameters

```
REAL :: optattcoef1_cst, optattcoef2_cst, opt_frac
```

optattcoef1_cst Inverse optical attenuation depth (λ_1^{-1}) for the absorption of infrared solar radiation as used in (5.17)[m⁻¹]. Default is 10.0.

- optattcoef2_cst** Inverse optical attenuation depth (λ_2^{-1}) for the absorption of short-wave solar radiation as used in (5.17) [m^{-1}]. Default is 0.067.
- opt_frac** Long-wave fraction R of surface solar radiance as used in (5.17) [-]. Default is 0.54.

19.3.11 Parameters for the structure module

```
INTEGER :: numdis, numdry, numthinu, numthinv, numwbaru, numwbarv
REAL :: wbarrlxu, wbarrlxv
```

- numdis** Number of discharge locations. Used when `iopt_dischr=1`. No default.
- numdry** Number of dry cells. Used when `iopt_drycel=1`. No default.
- numthinu** Number of thin dams at U-nodes. Used when `iopt_thndam=1`. No default.
- numthinv** Number of thin dams at V-nodes. Used when `iopt_thndam=1`. No default.
- numwbaru** Number of weirs/barriers at U-nodes. Used when `iopt_weibar=1`. No default.
- numwbarv** Number of weirs/barriers at V-nodes. Used when `iopt_weibar=1`. No default.
- wbarrlxu** Time relaxation coefficient θ_{wrl} at U-node weirs/barriers. Used in (11.15) when `iopt_weibar=1`. Default is 1.0.
- wbarrlxv** Time relaxation coefficient θ_{wrl} at V-node weirs/barriers. Used in (11.15) when `iopt_weibar=1`. Default is 1.0.

19.3.12 Parameters for the drying-wetting scheme

```
REAL :: dcrit_fld, dmin_fld, dthd_fld, ndwexp
```

- dcrit_fld** Critical water depth d_{crit} used in the drying/wetting algorithm [m]. Default is 0.1.
- dmin_fld** Minimum water depth d_{min} used in the drying/wetting algorithm [m]. Default is 0.02.
- dthd_fld** Threshold water depth d_{th} used in the mask criteria for drying and flooding [m]. Default is 0.1.
- ndwexp** Exponent n_{wd} used in the formulation given by (11.29b) for the α -factor. Default is 1.0.

19.3.13 Parameters used in numerical schemes

multi-grid scheme

```
INTEGER :: maxitsimp, nomgiterations, nomglevels, nopostsweeps, &
           & nopresweeps, nosmoothsteps
REAL :: dzetaresid_conv, mg_tol, ur_mg, ur_smooth, theta_sur
```

dzetaresid_conv Allowed maximum (over the whole domain) residual of surface elevation after the last iteration in the outer loop of the multi-grid scheme [m]. Default is 10^{-5} .

maxitsimp Maximum allowed number of outer iterations for the implicit scheme. Default is 1.

mg_tol Relative tolerance used by the multi-grid scheme for solving the linear system. Default is 10^{-7} .

nomgiterations Maximum allowed number of iterations in the multigrid procedure. Default is 100.

nomglevels Number of sub-grid levels (including the main grid). Default is 1 (i.e. no sub-grids).

nopostsweeps Number of post-relaxation iterations. Default is 2.

nopresweeps Number of pre-relaxation iterations. Default is 2.

nosmoothsteps Number of linear solver (smoother) iterations at the coarsest level. Default is 2.

theta_sur Implicit factor used for the surface slope in the multi-grid formulation. Default is 0.5. Recommended.

ur_mg Underrelaxation factor for multi-grid updates. Default is 1.0.

ur_smooth Underrelaxation factor for the smoother. Default is 0.8.

other numerical parameters

```
REAL :: cgravratio, theta_cor, theta_vadv, theta_vdif
```

cgravratio Ratio of internal to external wave speed used in the open boundary condition (5.370) for the baroclinic current. Default is 0.03.

theta_cor Implicit factor θ_c for the Coriolis term [between 0.0 and 1.0]. Default is 0.5 (recommended).

theta_vadv Implicit factor θ_a for vertical advection [between 0.0 and 1.0]. Default is 0.501 (recommended).

theta_vdif Implicit factor θ_d for vertical diffusion [between 0.0 and 1.0]. Default is 1.0 (recommended).

19.3.14 Physical model constants

```
REAL :: celtokelv, kboltz, Rearth, stefbolz
```

The parameters below should not be changed by the user

celtokelv	Difference between temperature in ^0K and ^0C . Value is 273.15.
kboltz	Boltzmann's constant [$\text{m}^2\text{kg s}^{-2} \text{K}^{-1}$]. Value is 1.38065×10^{-23} .
Rearth	Mean Earth radius assuming that the Earth has a spherical shape with the same volume as the actual Earth [m]. Value is 6371000.
stefbolz	Stefan-Boltzmann constant [$\text{Wm}^{-2} \text{K}^{-4}$]. Value is 5.67×10^{-8} .

19.3.15 Turbulence model parameters

general parameters

```
REAL :: ckar, dissipmin, riccrit_iw, tkemin, vdifmom_iw,
vdifscal_iw, vdifshear_iw, zlmixmin
```

ckar	von Karman constant. Default is 0.4. Should not be changed.
dissipmin	Numerical lower limit ε_{min} for ε [W/kg]. Default is 10^{-12} .
riccrit_iw	Critical Richardson number Ri_0 in the Large et al. (1994) background mixing scheme (5.188). Default is 0.7.
tkemin	Numerical lower limit k_{min} for k [J/kg]. Default is 10^{-14} .
vdifmom_iw	Internal wave breaking diffusion coefficient ν_{T0} for momentum in the Large et al. (1994) background mixing scheme (5.188) [m^2/s]. Default is 10^{-4} .
vdifscal_iw	Internal wave breaking diffusion coefficient λ_{T0} for scalars in the Large et al. (1994) background mixing scheme (5.188) [m^2/s]. Default is 5×10^{-5} .
vdifshear_iw	Maximum mixing due to unresolved vertical shear ν_0^s in the Large et al. (1994) background mixing scheme (5.188) [m^2/s]. Default is 0.005.
zlmixmin	Numerical lower limit l_{min} for l [m]. Default is 1.7×10^{-10} .

Remarks

It is recommended to keep the default numerical bounds for the turbulence variables (see Section 5.3.3.7). This applies also for the other defaults which are obtained from literature.

algebraic schemes

The following parameters are used in case that an algebraic scheme is selected for turbulence (iopt_vdif_coef=2). Defaults are obtained from literature and should not be changed.

```
REAL :: alpha_Black, alpha_ma, alpha_pp, beta_ma, beta_Xing, cnu_ad, &
& delta1_ad, delta2_ad, expmom_ma, expmom_pp, expscal_ma, &
& k1_ad, k2_ad, lambda_ad, omega1_ad, r1_ad, r2_ad, vbmom_pp, &
& vbscal_pp, vmaxmom_ma, vmaxscal_ma, vmax_pp, v0dif_ma, &
& v0dif_pp, zrough_bot, zrough_sur
```

alpha_Black	Constant α_1 in the Blackadar (1962) mixing length formulation (5.178). Default is 0.2.
alpha_ma	Parameter α_m in the Munk & Anderson (1948) scheme (5.97)–(5.100). Default is 10.0.
alpha_pp	Parameter α_p in the Pacanowski & Philander (1981) scheme (5.93)–x(5.95). Default is 5.0.
beta_ma	Parameter β_m in the Munk & Anderson (1948) scheme (5.97)–(5.100). Default is 3.33.
beta_Xing	Attenuation factor β_1 in the Xing & Davies (1996) mixing length formulation (5.176). Default is 2.0.
cnu_ad	Parameter C_ν in equation (5.112). Default is 2.0.
delta1_ad	Parameter δ_1 in equation (5.104). Default is 0.0.
delta2_ad	Parameter δ_2 in equation (5.104). Default is 0.0.
expmom_ma	Parameter n_1 in the Munk & Anderson (1948) scheme (5.97)–(5.100). Default is 0.5.
expmom_pp	Parameter n_p in the Pacanowski & Philander (1981) scheme (5.93)–(5.95). Default is 2.0.
expscal_ma	Parameter n_2 in the Munk & Anderson (1948) scheme (5.97)–(5.100). Default is 1.5.
k1_ad	Parameter K_1 in equations (5.109) and (5.111). Default is 0.0025.
k2_ad	Parameter K_2 in equation (5.110). Default is 2×10^{-5} .
lambda_ad	Parameter λ_* in equation (5.107) [m]. Default is 0.0.
omega1_ad	Parameter ω_1 in equation (5.112) [s^{-1}]. Default is 0.0001.
r1_ad	Parameter r_1 in equation (5.104). Default is 1.0.

r2_ad	Parameter r_2 in equation (5.104). Default is 1.0.
vbmom_pp	Parameter ν_{bp} in the Pacanowski & Philander (1981) scheme (5.93)–(5.95) [m ² /s]. Default is 10^{-4} .
vbscal_pp	Parameter λ_{bp} in the Pacanowski & Philander (1981) scheme (5.93)–(5.95) [m ² /s]. Default is 10^{-5} .
vmaxmom_ma	Parameter ν_{max} in the Munk & Anderson (1948) scheme (5.97)–(5.100). Default is 3.0.
vmaxscal_ma	Parameter λ_{max} in the Munk & Anderson (1948) scheme (5.97)–(5.100). Default is 4.0.
vmax_pp	Parameter ν_{max} in the Pacanowski & Philander (1981) scheme (5.93)–(5.95). Default is 3.0.
v0dif_ma	Parameter ν_{0m} in the Munk & Anderson (1948) scheme (5.97)–(5.100) [m ² /s]. Default is 0.06.
v0dif_pp	Parameter ν_{0p} in the Pacanowski & Philander (1981) scheme (5.93)–(5.95) [m ² /s]. Default is 0.01.
zrough_bot	Bottom roughness length z_{0b} in the mixing length formulation (5.173) [m]. Default is 0.0.
zrough_sur	Surface roughness length z_{0s} in the mixing length formulation (5.173) [m]. Default is 0.0.

The following parameters are used in case that a second order turbulence closure scheme is selected (`iopt_vdif_coef=3`). Defaults are obtained from literature and should not be changed.

```
REAL :: c1_eps, c2_eps, c31_eps, c32_eps, c_sk, e1_my, e2_my,
       e3_my, sigma_k, skeps, sq_my, tkelim, wfltk
```

c1_eps	Constant $c_{1\varepsilon}$ in the shear production term of the ε -equation (5.166). Default is 1.44.
c2_eps	Constant $c_{2\varepsilon}$ in the dissipation term of the ε -equation (5.166). Default is 1.92.
c31_eps	Constant $c_{3\varepsilon}$ in the buoyancy sink term of the ε -equation (5.166) in case of stable stratification ($N^2 > 0$). Default is 0.2.
c32_eps	Constant $c_{3\varepsilon}$ in the buoyancy source term of the ε -equation (5.166) in case of unstable stratification ($N^2 < 0$). Default is 1.0.
c_sk	Daly-Harlow parameter c_{sk} in (5.138). Default is 0.15.
e1_my	Constant E_1 in the shear production term of the kl -equation (5.170). Default is 1.8.

- e2_my** Constant E_2 in the wall proximity term (5.171) of the kl -equation (5.170). Default is 1.33.
- e3_my** Constant E_3 in the buoyancy source/sink term of the kl -equation (5.170). Default is 1.0.
- sigma_k** Parameter σ_k used to define S_k in (5.162). Default is 1.0.
- skeps** Neutral value S_{k0} of the stability coefficient S_k in the k - ε model (see equation (5.161)). Default is 0.09.
- sq_my** Parameter S_q used to determine S_{k0} in the Mellor-Yamada model (see equation (5.163)). Default is 0.2.
- tkelim** Background limit k_{lim} for k (see equation (5.187)) [J/kg]. Default is 10^{-6} .
- wfltke** Surface wave factor c_w used in the surface flux condition (5.247) for turbulent energy. Default is 0.0.

19.3.16 Parameters for user-defined output

```
INTEGER :: nofreqsanal, nosetsanal, nosetsavr, nosetstsr, &
           & nostatsanal, nostatsavr, nostatstsr, novarsanal, &
           & novarsavr, novarstsr
CHARACTER (LEN=lentitle) :: intitle, outtitle
CHARACTER (LEN=lendesc) :: institution_CF, references_CF
```

A few general parameters need to be specified for user-defined output in `usrdef_mod_params` or in the CIF if `cifile%status` equals '0' or 'R'. For more details about the meaning of the parameters below, see chapter 27.

- intitle** Title (maximum 20 characters) used to create names of model forcing files. Default is `runtitle`.
- nofreqsanal** Number of harmonic frequencies used for harmonic analysis if `iopt_out_anal=1`. No default.
- nosetsanal** Number of harmonic file sets if `iopt_out_anal=1`. No default.
- nosetsavr** Number of time averaged file sets if `iopt_out_avrgd=1`. No default.
- nosetstsr** Number of time series file sets if `iopt_out_tsers=1`. No default.
- nostatsanal** Number of harmonic output stations if `iopt_out_anal=1`. No default.
- nostatsavr** Number of time averaged output stations if `iopt_out_avrgd=1`. No default.

<code>nostatssr</code>	Number of time series output stations if <code>iopt_out_tsers=1</code> . No default.
<code>novarsanal</code>	Number of harmonic variables if <code>iopt_out_anal=1</code> . No default.
<code>novarsavr</code>	Number of time averaged variables if <code>iopt_out_avrgd=1</code> . No default.
<code>novarstsr</code>	Number of time series variables if <code>iopt_out_tsers=1</code> . No default.
<code>outtitle</code>	Title (maximum 20 characters) used to create names of user output files. Default is <code>runtile</code> .
<code>institution_CF</code>	Name of the institute where the simulation was performed. Used as a global attribute following the CF-conventions in all standard COHERENS files (see Section 27.6.2.1). Default is
	<code>RBINS - Operational Directorate Natural Environment</code>
<code>references_CF</code>	A global CF attribute where the user can add reference information (name of project, publications, ...). Default is none.

19.3.17 Formats used for ASCII output

```
LOGICAL :: freefmt
CHARACTER (Len=lenformat) :: characterfmt, doublefmt, integerfmt, &
                            & logicalfmt, realfmt
```

The parameters define the data formats used by COHERENS when output is written to an ASCII output file.

<code>freefmt</code>	All ASCII output is in free format if <code>.TRUE.</code> . Default.
<code>characterfmt</code>	ASCII format for character data if <code>freefmt=.FALSE..</code> Default is <code>CharacterFormat</code> defined in <code>syspars.f90</code> .
<code>doublefmt</code>	ASCII format for double precision data if <code>freefmt=.FALSE..</code> Default is <code>DoubleFormat</code> defined in <code>syspars.f90</code> .
<code>integerfmt</code>	ASCII format for integer data if <code>freefmt=.FALSE..</code> Default is <code>IntegerFormat</code> defined in <code>syspars.f90</code> .
<code>logicalfmt</code>	ASCII format for logical data if <code>freefmt=.FALSE..</code> Default is <code>LogicalFormat</code> defined in <code>syspars.f90</code> .
<code>realfmt</code>	ASCII format for single precision real data if <code>freefmt=.FALSE..</code> Default is <code>RealFormat</code> defined in <code>syspars.f90</code> .

19.4 Attributes of forcing files

All parameters in this section are defined in `usrdef.mod_params` or in the CIF block `mod_params`.

Model forcing requires the definition of parameters for providing the input data. They can be directly defined within a `usrdef_*` routine or by reading them for some external file. A series of file attributes, defined below, needs to be set by the user to inform the program how the data are accessed for each type of forcing. These attributes are stored in the 3-D derived type array `modfiles` defined by (omitting attributes which are only used internally and must not be defined by the user):

```

TYPE :: FileParams
  LOGICAL :: header
  CHARACTER (LEN=1) :: form, status
  CHARACTER (LEN=leniofile) :: filename
  INTEGER :: endfile
  INTEGER, DIMENSION(3) :: tlims
END TYPE FileParams
TYPE (FileParams), DIMENSION(MaxIOTypes,MaxIOFiles,2) :: &
  & modfiles

```

An element of the array `modfiles` can be generically represented as `modfiles(iddesc,ifil,iotype)` where `iddesc` is the “file descriptor” of the forcing file, `ifil` the “file number” and `iotype` equals 1 for input and 2 for output.

1. The first index `iddesc` is the forcing key id whose FORTRAN name is given by `io_` followed by 6 characters denoting the type of forcing, e.g. `io_inicon` for initial conditions. A full listing of all key ids and their purpose is listed in Table 19.1.
2. Forcing parameters and data for a specified forcing can be obtained from more than one file. The number of the file is given by the second index `ifil`. The maximum number of files for a given forcing id is given by the parameter `MaxIOFiles` defined in `syspars.f90`.
3. The third index `iotype` equals 1 for an input or 2 for an output forcing data file. The meaning is as follows. Almost all forcing data (except nesting and final conditions files) are input data, i.e. represented by an element of `modfiles` with `iotype=1`. Assume that the input data are provided in a user-defined non-standard format (i.e. with the `status` set to 'N', see below). By defining a corresponding output file with `iotype=2` one has the possibility to re-write the input data now in a

COHERENS standard format. When a first run is made using the non-standard format, the newly created forcing file can be used as input for a subsequent run. In that case the user only needs to change the **status** attribute from 'N' to 'R' (see below). Exceptions are

- Initial conditions (forcing key id **io_inicon**) files are always input files (**iotype**=1) and cannot be written in standard format to a file with the same key id.
- Final conditions (forcing id **io_fincon**) files are always output files (**iotype**=2) written in standard format which can be used to define initial conditions for a restart of the model at a different (including the initial) time.
- Nesting data are always written to an output (**iotype**=2) file.

Table 19.1: Listing of all forcing types, the **usrdef_** routine where they are defined if the status attribute equals 'N' and their purpose.

iddesc	ifil	iotype	routine	purpose
io_mppmod	1	1,2	usrdef_partition	domain decomposition
io_inicon	ics_phys	1	usrdef_physics	initial conditions for the hydrodynamics
	ics_sed	1	usrdef_sedics	initial conditions for the sediments
	ics_morp	1	usrdef_morphics	initial conditions for the morphology
	ics_bio	1	usrdef_bioics	initial conditions for the biological module
	ics_part	1	usrdef_partics	initial conditions for the particle module
io_fincon	ics_phys	2		restart conditions for the hydrodynamics
	ics_sed	2		restart conditions for the sediments
	ics_morph	2		restart conditions for the morphology
	ics_bio	2		restart conditions for the biological module
	ics_part	2		restart conditions for the particle module

(Continued)

Table 19.1: Continued

iddesc	ifil	iotype	routine	purpose
io_modgrd	1	1,2	usrdef_grid	model grid, bathymetry, open boundary locations
io_metabs	1	1,2	usrdef_surface_abs_grid	external meteo grid in absolute coordinates
io_sstabs	1	1,2	usrdef_surface_abs_grid	external SST grid in absolute coordinates
io_wavabs	1	1,2	usrdef_surface_abs_grid	external surface wave grid in absolute coordinates
io_parabs	1	1,2	usrdef_surface_abs_grid	external surface (PAR) grid in absolute coordinates used in the biological module
io_nstgrd	1, ... nonestsets	1,2	usrdef_nstgrid	open boundary locations of nested subgrid(s)
io_metrel	1	1,2	usrdef_surface_rel_grid	external meteo grid in relative coordinates
io_sstrel	1	1,2	usrdef_surface_rel_grid	external SST grid in relative coordinates
io_wavrel	1	1,2	usrdef_surface_rel_grid	external surface wave grid in relative coordinates
io_parrel	1	1,2	usrdef_surface_rel_grid	external surface (PAR) grid in relative coordinates used in the biological model
io_sedspc	1	1,2	usrdef_sed_spec	attributes of sediment particles per fraction
io_parspc	1	1,2	usrdef_part_spec	attributes of tracer particles and cloud definitions

(Continued)

Table 19.1: Continued

iddesc	ifil	itype	routine	purpose
io_luvsur	1	1,2	usrdef_1dsur_spec	type of surface boundary condition and (optionally) tidal amplitudes and phases for sea elevation and/or surface slopes, in case of 1-D applications
	2	1,2	usrdef_1dsur_data	time series of surface data for surface elevations and/or surface slopes in non-harmonic form (used only for 1-D applications)
io_2uvobc	1	1,2	usrdef_2dcbc_spec	type of 2-D normal boundary conditions and data specifications for the barotropic mode and (optionally) harmonic components
	2,...	1,2	usrdef_2dcbc_data	time series of surface elevations and/or depth-integrated currents used at normal open boundaries
io_2xyobc	1	1,2	usrdef_2dcbc_spec	type of 2-D tangential boundary conditions and data specifications for the barotropic mode and (optionally) harmonic components
	2,...	1,2	usrdef_2dcbc_data	time series of depth-mean currents used at tangential open boundaries

(Continued)

Table 19.1: Continued

iddesc	ifil	itype	routine	purpose
io_3uvobc	1	1,2	usrdef_profobc_spec	type of boundary conditions normal to the open boundary and data specifications for baroclinic currents
	2,...	1,2	usrdef_profobc_data	time series of baroclinic currents used at normal open boundaries
io_3xyobc	1	1,2	usrdef_profobc_spec	type of boundary conditions tangential to the open boundary and data file specifications for horizontal currents
	2,...	1,2	usrdef_profobc_data	time series of horizontal currents used at tangential open boundaries
io_salobc	1	1,2	usrdef_profobc_spec	type of boundary conditions and data specifications for salinity
	2,...	1,2	usrdef_profobc_data	time series of salinity profiles used at normal open boundaries
io_tmlobc	1	1,2	usrdef_profobc_spec	type of boundary conditions and data specifications for temperature
	2,...	1,2	usrdef_profobc_data	time series of temperature profiles used at normal open boundaries
io_sedobc	1	1,2	usrdef_profobc_spec	type of boundary conditions and data specifications for sediment concentrations

(Continued)

Table 19.1: Continued

iddesc	ifil	iotype	routine	purpose
	2,...	1,2	<code>usrdef_profobc_data</code>	time series of sediment profiles used at normal open boundaries
<code>io_bioobc</code>	1	1,2	<code>usrdef_profobc_spec</code>	type of boundary conditions and data file specifications for biological concentrations
	2,...	1,2	<code>usrdef_profobc_data</code>	time series of profiles of biological concentrations used at normal open boundaries
<code>io_nstspc</code>	1	1,2	<code>usrdef_nstgrd_spec</code>	definitions of parameters and arrays used to define sub-grids for nesting (number of output locations, type of conditions and variables)
<code>io_2uvnst</code>	1,..., nonestsets	2		output file(s) containing the (normal) open boundary data (depth-integrated currents and/or surface elevations) for each nested sub-grid
<code>io_2xynst</code>	1,..., nonestsets	2		output file(s) containing the (tangential) open boundary data (depth-mean currents) for each nested sub-grid
<code>io_3uvnst</code>	1,..., nonestsets	2		output file(s) containing the (normal) open boundary baroclinic current data for each nested sub-grid

(Continued)

Table 19.1: Continued

iddesc	ifil	iotype	routine	purpose
io_3xynst	1,..., nonestsets	2		output file(s) containing the (tangential) open boundary current data for each nested sub-grid
io_salnst	1,..., nonestsets	2		output file(s) containing the open boundary salinity data for each nested sub-grid
io_tmppnst	1,..., nonestsets	2		output file(s) containing the open boundary temperature data for each nested sub-grid
io_sednst	1,..., nonestsets	2		output file(s) containing the open boundary sediment concentrations data for each nested sub-grid
io_bionst	1,..., nonestsets	2		output file(s) containing the open boundary biological concentrations data for each nested sub-grid
io_metsur	1	1,2	usrdef_surface_data	time series of meteorological surface data as defined on the meteorological grid
io_ssstsur	1	1,2	usrdef_surface_data	time series of sea surface temperature data as defined on the external SST grid
io_wavsur	1	1,2	usrdef_surface_data	time series of wave data as defined on the external wave grid

(Continued)

Table 19.1: Continued

iddesc	ifil	itype	routine	purpose
io_biosur	1	1,2	usrdef_surface_data	time series of biological (PAR) data as defined on the external biological grid
io_drycel	1	1,2	usrdef_dry_cells	dry cell locations (iopt_drycel=1)
io_thndam	1	1,2	usrdef_thin_dams	thin dams locations (iopt_thndam=1)
io_weibar	1	1,2	usrdef_weirs	locations of weirs/bars and parameter arrays used in the weirs/barriers module (iopt_weibar=1)
io_disspc	1	1,2	usrdef_dischr_spec	type of vertical location used in the setup of the discharge module (iopt_dischr=1)
io_disloc	1	1,2	usrdef_dischr_data	time series of the discharge locations (iopt_dischr=1)
io_disvol	1	1,2	usrdef_dischr_data	time series of the discharged water volumes (iopt_dischr=1)
io_discur	1	1,2	usrdef_dischr_data	time series of the area and direction of the discharges (iopt_dischr=1)
io_dissal	1	1,2	usrdef_dischr_data	time series of the salinity content of the discharged volume of water (iopt_dischr=1)
io_dissed	1	1,2	usrdef_dischr_data	time series of the sediment content of the discharged volume of water (iopt_dischr=1)

(Continued)

Table 19.1: Continued

iddesc	ifil	iotype	routine	purpose
io_distmp	1	1,2	usrdef_dischr_data	time series of the temperature of the discharged volume of water (iopt_dischr=1)
io_parcld	1,...,	1,2	usrdef_parcld_data	date/times and locations of particle cloud releases
io_pargrd	1	1,2	usrdef_particle_grid	setup of model grid for off-line runs with the particle module
io_parphs	1	1,2	usrdef_particle_phys_data	time series of physical data used by the particle module in off-line mode

19.4.1 Forcing attributes for input data (**iotype=1**)

status Status of the data file.

‘0’: Disabled. This is the default.

‘N’: User-defined in which case a corresponding `usrdef_` routine is called.

‘R’: The data are obtained from a file in standard format.

form File format.

‘A’: ASCII (portable, sequential).

‘U’: Unformatted binary (non-portable, sequential).

‘N’: netCDF format (portable, non-sequential). Default.

header .TRUE. if the forcing file contains a header section in ASCII, unformatted binary or netCDF format with metadata information (see Chapter 29 for details).

filename File name. If not defined, a default name is used (see below).

tlims Start/end/step time indices, given as multiples of the (smallest) time step `timestep`. They should only be provided in the case that the forcing data are given as time series. These parameters are not directly used for reading the data, but to make updates using

time-interpolation after `tlims(3)×timestep` seconds. Assume that the input data in the forcing file are provided at intervals of 3 hours. If `timestep` equals 5 min and `tlims(3)=3`, the data will be interpolated in time at 15 minutes intervals. No time interpolation is performed if `tlims(3)<0`. Further details are given below. Defaults are (0,0,1) meaning that the data are provided at the initial time only and taken as time-independent.

- endfile** Switch to decide what action needs to be taken when an end of file conditions occurs.
- 0: The program aborts with an error message. Default.
 - 1: The program continues, no further attempt will be made to read the data.
 - 2: The program continues, a next attempt to read the data will be made after `nowaitsecs` seconds.

Remarks

- Important to note is that the `status` attribute equals '0' by default which means that the setup parameters and arrays associated with the descriptor `iddesc` will not be defined in the corresponding `usrdef_` routine and not be read from a file in standard format.
- If `status` is set to 'N', it is up to the user to write the code for the corresponding `usrdef_` routine. In that case, the notion of a "forcing" file may become irrelevant if the user decides to define the required data without any input file. On the other hand, the forcing data could also be obtained from more than one input file. In both cases, the `filename` and `endfile` attributes are no longer useful. This means that the notion of a "forcing file" can be used in a real sense, i.e. as an existing file, or in a virtual sense where the data are defined by the user using FORTRAN code.
- If the name of the data file is not provided, its default name is either either given by

```
TRIM(intitle) //'.' //TRIM(namedesc) //'.' //TRIM(suffix)
```

or

```
TRIM(intitle) //'.' //TRIM(namedesc) //cfil //'.' //TRIM(suffix)
```

where `namedesc` is the name of the forcing key id in string format, e.g. ‘`metsur`’ for the forcing key `io_metsur`, `suffix` is ‘`txt`’ for ASCII, ‘`bin`’ for unformatted binary and ‘`nc`’ for `netCDF` files. The forcing file number `cfil` is added to the file name if `ifil` is allowed to take values larger than 1 (see Table 19.1).

- The meaning of `tlims` is illustrated as follows in case of e.g. meteorological forcing data. These data are used to evaluate the surface fluxes of momentum, heat and salinity and for the atmospheric pressure gradient in the momentum equations. All these quantities will be updated from time `tlims(1)` upto time `tlims(2)` at time intervals given by `tlims(3)`. The data are read into the program with a date/time stamp which is saved. If `tlims(3)>0`, which should be shorter than the time interval between two input dates, the meteo data are first linearly interpolated in time between their values at the most recent date, earlier (or equal) than the current program time, and the earliest date later than the current time. Since these dates are stored in memory, the program knows automatically when new data are needed. If `tlims(3)<0`, the method is the same but without time interpolation, i.e. the data at the current program time are set to their values at the most recent date earlier than or equal to the program time. Although it is not absolutely necessary, it is recommended that `tlims(3)` is smaller than or equal to the time interval between two consecutive inputs. Note that if an element of the vector `tlims` is set to the undefined value `int_fill`, this value will be automatically replaced by `nstep` which is the total number of base time steps in the simulation, meaning that the last update takes place at the end of the run.
- If `endfile` equals 2 and an “end of file condition” occurs during a read, the program waits for `nowaitsecs` seconds before make a next attempt. The total waiting time is given by `maxwaitsecs` after which the program aborts with an error message. The procedure is intended for making simulations in interactive mode. For example, assume that a main grid writes the open boundary data for a nested sub-grid. If the main and sub-grid simulations are launched together and the former runs slower than the latter, the latter grid will wait for input from the former.

19.4.2 Forcing attributes for output data (`itype=2`)

`status` Status of the data file.

‘0’ : Disabled. This is the default.

	'W': The data are written to a file in COHERENS standard format.
form	File format. 'A': ASCII (portable, sequential). 'U': Unformatted binary (non-portable,sequential). 'N': netCDF format (portable, non-sequential).
filename	File name. If not defined, a default name is used which is the same as the one for the input case (see above).
tlims	Start/end/step time indices for writing the output. Only needed in case nested output is written.

Note that the **header** attribute for output forcing files is always **.TRUE.**.

It is obvious that the same forcing file cannot be activated both in input and output mode.

The **fortran** example below illustrates the procedures for forcing files.

```
!
!1. Attributes for input forcing
!-----
!
!--model grid
modfiles(io_modgrd,1,1)%status = 'R'
modfiles(io_modgrd,1,1)%form = 'N'
modfiles(io_modgrd,1,1)%filename = 'bathymetry.dat'

!--initial conditions
modfiles(io_inicon,ics_phys,1)%status = 'N'
modfiles(io_inicon,1,1)%form = 'N'
modfiles(io_modgrd,1,1)%filename = 'north_sea_init_conds.nc'

!--open boundary conditions
modfiles(io_2uvobc,1,1)%status = 'N'
modfiles(io_2uvobc,1,1)%form = 'A'
modfiles(io_2uvobc,1,1)%filename = 'tidal_harmonics'
modfiles(io_2uvobc,2:3,1)%status = 'N'
modfiles(io_2uvobc,2:3,1)%form = 'A'
modfiles(io_2uvobc,2,1)%filename = 'open_sea_boundary_data'
modfiles(io_2uvobc,3,1)%filename = 'river_open_boundary_data'
modfiles(io_2uvobc,2,1)%tlims = (/0,int_fill,30/)
```

```

modfiles(io_2uvobc,3,1)%tlims = (/0,int_fill,30/)

!
!2. Attributes for output forcing
!-----
!

!--initial conditions
norestarts = 1
ntrestart(1) = int_fill
modfiles(io_fincon,ics_phys,2)%status = 'W'
modfiles(io_fincon,ics_phys,2)%form = 'N'

!--open boundary conditions
modfiles(io_2uvobc,1:3,2)%status = 'W'
modfiles(io_2uvobc,1:3,2)%form = 'N'

```

The following forcing files are defined

- A model grid input file with a user-defined name in standard **netCDF** format.
- User-defined initial condition file in **netCDF** format using the default file name. The data are obtained by calling the routine **usrdef_physics**.
- Type of open boundary conditions are defined **usrdef_2dcbc_spec**. Tidal harmonics are obtained from an external ASCII file in non-standard format and with a user-defined name.
- Two files, in ASCII format with a user-defined name, contain time series of 2-D open boundary data (e.g. one for open sea and one for river boundaries). The data are provided in **usrdef_2dcbc_data** and interpolated in time at an interval of 30 time steps.
- A final conditions file is written at the last time step.
- All data and parameters defined in **usrdef_2dcbc** (eventually including tidal harmonics) are written to a COHERENS standard file in **netCDF** format with the default file name.
- The open boundary time series are re-written to a file in COHERENS standard **netCDF** format using the default file name.

19.5 Parameters for surface data grids

All parameters in this section are defined in `usrdef_mod_params` or in the CIF block `mod_params`.

Surface data grids are external grids where (e.g. meteorological) data are defined for the surface forcing. The parameters characterising a surface grid are stored into the derived type 2-D array `surfacegrids`, which is declared as follows

```
TYPE :: GridParams
  LOGICAL :: rotated
  INTEGER :: datflag, nhtype, n1dat, n2dat
  REAL :: delxdat, delydat, gridangle, x0dat, y0dat, y0rot
END TYPE GridParams
TYPE (GridParams), DIMENSION(MaxGridTypes,2) :: surfacegrids
```

An element of the array `surfacegrids` can be generically represented as `surfacegrids(igrd,ifil)` where `igrd` is a key id, called the “grid descriptor” and `ifil` the “file number”. The file number can take the value of 1 for an external grid intended for input and 2 for an external grid intended for output. The output case is intended for future applications and currently not implemented. This means that `ifil` must always be 1.

All parameters of this section are defined in `usrdef_mod_params` or in the CIF block `mod_params`.

19.5.1 Grid descriptors

The grid descriptor can take the following values:

<code>igrd_model</code>	model grid
<code>igrd_meteo</code>	meteorological external grid
<code>igrd_sst</code>	sea surface temperature external grid
<code>igrd_waves</code>	surface wave external grid
<code>igrd_par</code>	external grid with PAR (Photosynthetically Available Radiation) remote sensing data used in the biological module

19.5.2 Attributes of the horizontal model grid

Identifying the model grid as an external grid seems rather strange at first sight. The intention is to provide a series of attributes which can be used by COHERENS to construct the horizontal grid in a flexible way. The following attributes can be defined.

1. Uniform rectangular grids

delxdat (Uniform) grid spacing in the X-direction [m or degrees longitude].

delydat (Uniform) grid spacing in the Y-direction [m or degrees latitude].

x0dat X-coordinate of the lower left (South-West) corner node of the model grid [m or degrees longitude].

y0dat Y-coordinate of the lower left (South-West) corner node of the model grid [m or degrees latitude].

2. Rotated (uniform or non-uniform) rectangular grids

rotated Must be set to `.TRUE.` in case of a rotated grid. Default is `.FALSE..`

gridangle Grid rotation angle α (see Section 4.1.3) [decimal degrees]. Must be between 0 and 180^0 .

y0rot The reference latitude ϕ'_r from with the longitude of the displaced North Pole is determined (see Section 4.1.3) [decimal degrees]. Must between -90^0 and 90^0 .

3. Curvilinear grid

No attributes have to be defined. The model grid is defined in `usrdef_grid` or from a COHERENS standard file.

19.5.3 Attributes of an external data grid

In the case of an external (meteorological, sea surface temperature, surface wave, PAR) grid, the following attributes must or may be defined

nhtype Type of the surface data grid.

0: single grid point

1: uniform rectangular grid

2: non-uniform rectangular grid

3: non-rectangular (curvilinear or non-structured)

4: the same as the model grid

n1dat X-dimension of the external grid.

n2dat Y-dimension of the external grid.

delxdat Uniform grid spacing in the X-direction [m or degrees longitude].

delydat Uniform grid spacing in the Y-direction [m or degrees latitude].

x0dat X-coordinate of the lower left corner [m or degrees longitude].

y0dat Y-coordinate of the lower left corner [m or degrees latitude]

datflag Selects the type of interpolation when the surface data contain flagged values (land points or invalid data)¹

0: Data do not contain flagged values. Default for meteo grid.

1: Data may contain flagged values, interpolation is performed if all surrounding data values are not flagged.

2: Data may contain flagged values, interpolation is performed if at least one surrounding data value is not flagged. Default for surface wave and SST grid.

Remarks

In the current version of COHERENS rotated grids are only allowed for the model grid and not for external grids. Extensions to external grids could be envisaged in future releases (e.g. meteo grids with a displaced North Pole).

¹For more information about the use of data flags see Section 18.4.6.

Chapter 20

Model grid and initial conditions

This chapter explains the setup of model grid, bathymetry and physical initial conditions, which are defined in the following *Usrdef_Model.f90* routines:

- `usrdef_grid`: model grid and bathymetry
- `usrdef_partition`: domain decomposition
- `usrdef_physics`: initial conditions

20.1 Model grid and bathymetry

The routine `usrdef_grid` is called when

```
modfiles(io_modgrd,1,1)%status = 'N'  
FORTRAN template  
SUBROUTINE usrdef_grid  
IMPLICIT NONE  
  
USE depths  
USE grid  
USE gridpars  
USE iopars  
USE time_routines, ONLY: log_timer_in, log_timer_out  
  
procname(pglev+1) = 'usrdef_grid'
```

```

CALL log_timer_in()
...
CALL log_timer_out()

END SUBROUTINE usrdef_grid

```

The aim of this routine is to define the model grid, bathymetry and open boundary locations. The following arrays may or must be defined here.

```

! used when
REAL, DIMENSION(nc) :: gdelxglb           ! iopt_grid_htype=2
REAL, DIMENSION(nr) :: gdelyglb           ! iopt_grid_htype=2
REAL, DIMENSION(nc, nr) :: gxcoordglb, gycoordglb ! iopt_grid_htype=3
REAL, DIMENSION(nz+1) :: gsigcoordatw      ! iopt_grid_vtype=2,
                                              ! iopt_grid_vtype_transf=20
REAL, DIMENSION(nc-1, nr-1, nz+1) :: gscoordglb ! iopt_grid_vtype=3,
                                              ! iopt_grid_vtype_transf=30
REAL, DIMENSION(nc-1, nr-1) :: depmeanglb
REAL, DIMENSION(nobu) :: iobu, jobu
REAL, DIMENSION(nobv) :: iobv, jobv

```

1. Horizontal model grid

gdelxglb Grid spacings in the X-direction at C-nodes in case a non-rectangular grid is selected [m or degrees longitude].

gdelyglb Grid spacings in the Y-direction at C-nodes in case a non-rectangular grid is selected [m or degrees latitude].

gxcoordglb X-coordinates of the model grid at corner nodes in case a curvilinear grid is selected [m or degrees longitude, positive East].

gycoordglb Y-coordinates of the model grid at corner nodes in case a curvilinear grid is selected [m or degrees latitude, positive North].

2. Vertical model grid

gsigcoordatw Sigma coordinates at W-nodes in case a horizontally uniform, but vertically non-uniform grid is selected. If `iopt_grid_vtype_transf` equals 21, 22 or 23, the array is not needed here since the vertical grid is constructed by the program using one of the vertical coordinates transformations described in Section 4.1.4.1.

gscoordglb Sigma grid at W-nodes in case a user-defined horizontally and vertically non-uniform grid is selected. If **iop_grid_vtype_transf** equals 31 or 32, the array is not needed here since the vertical grid is constructed by the program using one of the coordinates transformations described in Section 4.1.4.2.

3. Bathymetry

depmeanglb Mean water depths at C-nodes [m]. If not supplied, a uniform bathymetry is assumed with water depths equal to the parameter **depmean_cst**. Permanent land cells are marked with the flag **depmean_flag**. If the drying/wetting scheme is activated, negative values are used for grid locations above the mean sea level (e.g. tidal flats).

4. Open boundary locations

jobu (Global) X-index of the West/East open boundary points at U-nodes.

jobu (Global) Y-index of the West/East open boundary points at U-nodes.

jobv (Global) X-index of the South/North open boundary points at V-nodes.

jobv (Global) Y-index of the South/North open boundary points at V-nodes.

Remarks

- The parameters **nobu** and **nobv** represent the total number of open boundaries at respectively U- and V-nodes and are internally defined as the sum of open sea and river boundaries, i.e.

```
nobu = nosbu+nrvbu; nobv = nosbv + nrvbv
```

- Coordinate units in the horizontal are meters or fractional degrees depending on whether **iop_grid_sph** equals 0 or 1.
- In the case of a uniform rectangular grid (**iop_grid_htype=1**), the horizontal grid is defined in **usrdef_mod_params** using the parameters defined in Section 19.5.2.

- In case of a uniform vertical σ -grid (`iopt_grid_vtype=1`), the σ -levels are uniformly distributed over the vertical. This is automatically performed by the program.
- The code requires that open sea boundaries must be stored in the first section of the arrays (e.g. `iobu(1:nosbu)`), the river boundaries in the next section (e.g. `iobu(nosbu+1:nobu)`).
- A good practice is to order the indexing of the open boundary locations in the following way:
 - at U-nodes: first the locations on western boundaries from South to North, then the locations on eastern boundaries from South to North and finally the river boundaries.
 - at V-nodes: first the locations on southern boundaries from West to East, then the locations on northern boundaries from West to East and finally the river boundaries.
- Water depths are measured downwards. This means, that in case that the inundation scheme is applied, land topography is represented by negative values of `depmeanglb`. Values equal to the data flag `depmean_flag` are considered as permanent land cells and excluded from the calculation. In the case that no inundation is applied (`iopt_fld = 0`), it is advised to set `depmean_flag` to zero (which is also the default value), all values of the bathymetry at locations above sea level to zero and (eventually) impose a minimum water depth at sea to prevent accidental drying which may cause a crash of the program.

20.2 Domain decomposition

The routine `usrdef_partition` is called when

```
iop_MPI_partit = 2
modfiles(io_mppmod,1,1)%status = 'N'
```

and the user wants to use a non-regular domain decomposition.

```

FORTRAN template
SUBROUTINE usrdef_partition

USE iopars
```

```

USE multigrid
USE paralpars

IMPLICIT NONE

procname(pglev+1) = 'usrdef_partition'
CALL log_timer_in()
...
CALL log_timer_out()

END SUBROUTINE usrdef_partition

```

The domain decomposition is stored in the derived type vector array `mgvars`, defined by (only including the attributes used in this routine):

```

TYPE :: MultiGridVars
INTEGER, POINTER, DIMENSION(nprocs) :: nc1procs, nc2procs, &
                                         & nr1procs, nr2procs
END TYPE MultiGridVars
TYPE (MultiGridVars), DIMENSION(0:nomgelevels-1) :: mgvars

```

where `nprocs` is the number of parallel processes used in the simulation (1 in the serial case) and `nomgelevels` the number grid levels used in the multigrid scheme (1 if `iopt_hydro_impl=0`). The attributes are defined as follows:

`mgvars%nc1procs` Global X-index of the most western C-node cells of the process domain.
`mgvars%nc2procs` Global X-index of the most eastern C-node cells of the process domain.
`mgvars%nr1procs` Global Y-index of the most southern C-node cells of the process domain.
`mgvars%nr2procs` Global Y-index of the most northern C-node cells of the process domain.

The arrays are needed when the program is set up in parallel mode `nprocs>1` and `iopt_MPI_partit=2`. It must be remarked here that it is not straightforward to set up a non-regular domain decomposition. In particular, when `nomgelevels>1`, care must be taken that the domain boundaries of the coarser (higher level) sub-grids are aligned with domain boundaries of the finer (lower level) grids. In view of this constraint, the procedure for constructing domain decompositions at all grid levels is not evident. This routine is therefore seldom used in practice. It is therefore recommended instead to

define the decomposition using the `gridproc` utility program, which provides the decomposition arrays in standard format. For details see Section ??.

20.3 Initial physical conditions

The routine `usrdef_physics` is called when

```
modfiles(io_inicon,ics_phys,1)%status = 'N'
```

This is the routine where the initial conditions are defined for the hydrodynamic module. The routine is called by *all* processes if `modfiles(io_physics,1,1)%status='N'`. The following arrays can be defined here

```

! used when
REAL, DIMENSION(ncloc,nrloc) :: udvel, vdvel, zeta ! iopt_grid_nodim=2 or
! iopt_hydro_impl=0
! iopt_grid_nodim=2,3
REAL, DIMENSION(ncloc,nrloc) :: zeta
REAL, DIMENSION(ncloc,nrloc,nz) :: uvel, vvel
REAL, DIMENSION(ncloc,nrloc,nz+1) :: wvel ! iopt_grid_nodim=1 or 3
REAL, DIMENSION(ncloc,nrloc,nz) :: sal ! iopt_grid_nodim=3
REAL, DIMENSION(ncloc,nrloc,nz) :: temp ! iopt_sal>0
REAL, DIMENSION(ncloc,nrloc,nz+1) :: tke ! iopt_temp>0
REAL, DIMENSION(ncloc,nrloc,nz+1) :: zlmix ! iopt_vdif_coef=3,
! iopt_turb_ntrans=1,2
! iopt_vdif_coef=3,
! iopt_turb_ntrans=2,
! iopt_turb_param=1
REAL, DIMENSION(ncloc,nrloc,nz+1) :: dissip ! iopt_vdif_coef=3,
! iopt_turb_ntrans=2,
! iopt_turb_param=2
REAL, DIMENSION(ncloc,nrloc) :: bdragcoefatc ! iopt_bstres_drag=2
REAL, DIMENSION(ncloc,nrloc) :: zroughatc ! iopt_bstres_drag=4,5
REAL, DIMENSION(nconobc) :: phase_abc ! iopt_astro_pars=1
```

The following remarks should be given here:

- The array dimensions `ncloc` and `nrloc` indicate that these arrays are defined on the local grid in case the model is run in parallel mode. This in contrast with the forcing file method (see Section 30.3) where they are defined on the global grid. The reason is that model arrays are only defined on the local grid in the parallel case and not on the global grid to save internal memory. The only exceptions are the arrays for initialising the model grid discussed in the previous section. The

example below shows how a model grid array (like temperature) can be read firstly from an external data file and secondly distributed on the local grid.

- In almost all applications only a small fraction of the above arrays should be defined over here by the user since their default values given by the program are mostly sufficient.

The example code below shows different ways for reading initial conditions (restricted here to temperature). FORTRAN example

```
SUBROUTINE usrdef_physics
```

```
! example code to show how initial data can be read on the global
! grid and distributed on the local grid afterwards
! two alternative methods are given to open/close the data file:
! (1) using the modfiles attributes and calling open_filepars,
!     close_filepars
! (2) without these attributes and calling open_file, close_file
! both ASCII and netCDF case are considered
! two alternative methods are given for distributing the global arrays
! on the local grid
! (1) using the distribute_mod internal routine
! (2) using array sections of the global array
```

```
IMPLICIT NONE
```

```
!---modules needed within the example
USE datatypes
USE density
USE gridpars
USE iopars
USE switches
USE syspars
USE cf90_routines, ONLY: cf90_inquire, cf90_inquire_variable, cf90_get_var
USE inout_routines, ONLY: close_file, close_filepars, open_file, &
    & open_filepars
USE paral_comms, ONLY: distribute_mod
USE time_routines, ONLY: log_timer_in, log_timer_out

!---modules needed for all initial conditions
USE currents
```

```

USE density
USE depths
USE fluxes
USE obconds
USE tide
USE turbulence

!
!*Local variables
!
INTEGER :: iunit, ivar, novars
INTEGER :: read_method = ?, dist_method = ?
CHARACTER (LEN=lenform) :: form
CHARACTER (LEN=lenname) :: varname
REAL, DIMENSION(nc, nr, nz) :: tempdat
TYPE (FileParams) :: filepars

procname(pglev+1) = 'usrdef_physics'
CALL log_timer_in()

!
!1. Open data file
!-----
!

filepars = modfiles(io_inicon, ics_physics, 1)
form = filepars%form
!---standard method
IF (read_method.EQ.1) THEN
  CALL open_filepars(filepars)
  iunit = filepars%iunit
!---alternative method
ELSEIF (read_method.EQ.2) THEN
  CALL open_file(iunit, 'temperature.dat', 'IN', form)
ENDIF

!
!2. Initialise
!-----
!

```

```

tempdat = 0.0

!

!3. Read data on the 'physical' global domain
!-----
!

!---ASCII format
IF (form.EQ.'A') THEN
  READ(iunit,*) tempdat(1:nc-1,nr-1,:)
!---netCDF format
ELSEIF (form.EQ.'N')
  CALL cf90_inquire(iunit,nvariables=novars)
  ivar_210: DO ivar=1,novars
    CALL cf90_inquire_variable(iunit,ivar,varname)
    SELECT CASE (TRIM(varname))
      CASE ('temp')
        CALL cf90_get_var(iunit,ivar,tempdat(1:nc-1,nr-1,:),1,&
                         & start=(/1,1,1),count=(/nc-1,nr-1,nz/))
      END SELECT
    ENDDO ivar_210
  ENDIF

!

!4. Distribute from global to local
!-----
!

!---serial case
IF (iopt_MPI.EQ.0) THEN
  temp = tempdat
!---parallel case
ELSEIF(iopt_MPI.EQ.1) THEN
  IF (dist_method.EQ.1) THEN
    CALL distribute_mod(tempdat,temp(1:ncloc,1:nrloc,:),(/1,1,1/),&
                        & (/0,0,0/),0,0.0)
  ELSEIF (dist_method.EQ.2) THEN
    temp(1:ncloc,1:nrloc,:) = tempdat(nc1loc:nc2loc,nr1loc:nr2loc,:)
  ENDIF
ENDIF

```

```

!
!5. Close data file
!-----
!
!--standard method
IF (read_method.EQ.1) THEN
  CALL close_filepars(filepars,form)
!--alternative method
ELSEIF (read_method.EQ.2) THEN
  CALL close_file(iunit,form)
ENDIF

CALL log_timer_out()

RETURN

END SUBROUTINE usrdef_physics

```

Below is the full list of the arrays used for initialisation. They are either defined by the user or defined through their default values.

2-D mode

- udvel** Depth-integrated current in the X-direction [m^2/s]. Defined only for 2-D applications or for 3-D applications when the mode-splitting scheme is activated.
- vdvel** Depth-integrated current in the Y-direction [m^2/s]. Defined only for 2-D applications or for 3-D applications when the mode-splitting scheme is activated.
- zeta** Surface elevation [m]. Defined for 2-D or 3-D applications.

3-D currents

- uvel** Current u in the X-direction [m/s]. Defined for 1-D and 3-D applications.
- vvel** Current v in the Y-direction [m/s]. Defined for 1-D and 3-D applications.
- wvel** Transformed vertical current ω [m/s]. Defined only for 3-D applications.

Except for special applications or experimental case studies, these forcings should be obtained from a final condition file. Details of the procedure are given in Section 30.3.1.

density arrays

temp Temperature T [^0C]. Define only when `iopt_temp>0`. Negative (freezing) values are not allowed in the current implementation.

sal Salinity S [PSU].

turbulence arrays

Turbulence arrays can be defined if a RANS model (see Section 5.3.3) is selected (`iopt_vdif_coef=3`).

tke Turbulent kinetic energy k [J/kg].

zlmix Mixing length l [m].

dissip Dissipation of turbulent energy ε [W/kg].

These forcing data should not be procuded by the user (except for very exceptional case studies) but either by a final conditions file from a previous or by taking the default values given in Section 5.11 which are mostly sufficient.

arrays for bottom stres

bdragcoefatc Bottom drag coefficient C_{db} .

zroughatc Bottom roughness length z_0 [m].

tidal arrays

phase_obc Astronomical phases of the tidal constituents at open boundaries [rad]. Zero values by default.

Should only be used for idealised case studies.

defaults If a variable is not defined, the following defaults are assumed:

- temperature: uniform value given by `temp_ref`, defined in `usrdef_mod_params`
- salinity: uniform value given by `sal_ref`, defined in `usrdef_mod_params`
- turbulence: see Section 5.11

Zero default values are used for all other forcing arrays.

Finally, it is noted that initial conditions for all relevant arrays can be written during the simulation at intermediate and/or the final time step to a “final” condition file. This file can then be re-used to start up a new simulation (see Section 30.3).

Chapter 21

Open boundary conditions and nesting

This chapter deals with the setup of the open boundary conditions for the 2-D and 3-D mode, and the procedures for nesting. The following routines are discussed:

- `usrdef_2dcbc_spec`: specifies the type of conditions for 2-D variables and defines specification arrays needed to read the input data
- `usrdef_2dcbc_data`: defines the input of open boundary data for the 2-D mode
- `usrdef_profcbc_spec`: specifies the type of conditions for the 3-D mode and defines specification arrays needed to read the input data
- `usrdef_profcbc_data`: defines the input of open boundary data for 3-D currents and scalars
- `usrdef_nstgrd_spec`: general definitions for nesting (number of sub-grid data points and type of data)
- `usrdef_nstgrd`: geographical positions of the sub-grid (nested) data points

The first four routines are defined in *Usrdef_Model.f90*, the last two in *Usrdef_Nested_Grids.f90* (see Table 18.1).

21.1 2-D open boundary conditions

21.1.1 Open boundary specifications for the 2-D mode

Arrays for setting up normal and/or tangential open boundary conditions in the 2-D case are defined in the routine `usrdef_2dcbc_spec`. The routine is called for normal open boundary conditions if

```
iopt_abc_2D = 1
modfiles(io_2uvobc,1,1)%status = 'N'
```

and for tangential open conditions if

```
iopt_abc_2D_tang = 1
modfiles(io_2xyobc,1,1)%status = 'N'
```

Note that the file index for all open boundary specifiers is 1. The open boundary data are defined in files whose attributes are stored in `modfiles(io_2uvobc,ifil,1)` or `modfiles(io_2xyobc,ifil,1)` where `ifil` takes values of 2 upto `nfiles`. The number of associated data files is therefore given by `nfiles-1`. The parameter `nfiles` is defined by

```
nfiles = COUNT(modfiles(iddesc,:,:,1)%status.NE.'0')
```

where `iddesc` equals `io_2uvobc` in the normal and `io_2xyobc` in the tangential case.

FORTRAN template

```
SUBROUTINE usrdef_2dcbc_spec(iddesc)

INTEGER, INTENT(IN) :: iddesc

USE gridpars
USE iopars
USE obconds
USE tide
USE time_routines, ONLY: log_timer_in, log_timer_out

IMPLICIT NONE

procname(pglev+1) = 'usrdef_2dcbc_spec'
CALL log_timer_in()
...
```

```

CALL log_timer_out()

RETURN

END SUBROUTINE usrdef_2dcbc_spec

```

where

iddesc The file descriptor key id which may take the values

- io_2uvobc normal open boundary conditions
- io_2xyobc tangential open boundary conditions

External (specified) values for U , V , ζ or $\partial\zeta/\partial x_i$ are written in the general form (5.333). The first part ψ_0^e must be defined in `usrdef_2dcbc_data`, usually as time series input from a data file. The amplitudes A_n and phases φ_n for the harmonic part are time-independent and must be defined in `usrdef_2dcbc_spec`.

21.1.1.1 Type of 2-D open boundary conditions conditions

The following arrays are defined over here

```

! used when
INTEGER, DIMENSION(nobu) :: ityp2dobu, iloczobu ! iopt_2D=1
INTEGER, DIMENSION(nobv) :: ityp2dobv, iloczobv ! iopt_2D=1
INTEGER, DIMENSION(nobx) :: ityp2dobx           ! iopt_2D_tang=1
INTEGER, DIMENSION(noby) :: ityp2doby           ! iopt_2D_tang=1
INTEGER, DIMENSION(nqsecobu,2) :: iqsecobu     ! iopt_2D=1
INTEGER, DIMENSION(nqsecobv,2) :: iqsecobv     ! iopt_2D=1

```

1. Normal open boundary conditions

ityp2dobu Type of open boundary condition at U-nodes. See Section 5.10.1 for details.

- 0 : Clamped. Default.
- 1 : Zero slope.
- 2 : Zero volume flux.
- 3 : Specified elevation and local solution for transport.
- 4 : Specified transport.
- 5 : Radiation condition using shallow water speed.

6	: Orlanski (1976) condition.
7	: Camerlengo & O'Brien (1980) .
8	: Flather (1976) with specified elevation and transport.
9	: Flather (1976) with specified elevation.
10	: Røed & Smedstad (1984) .
11	: Characteristic method with specified elevation and transport.
12	: Characteristic method with specified elevation.
13	: Characteristic method using a zero normal gradient condition.
14	: Specified depth mean current.
15	: Specified discharge.
16	: Specified discharge along open boundary sections.
17	: Specified surface slope.
ityp2dobv	Type of open boundary condition at V-nodes. Meaning is the same as above.
iloczobu	If the elevation has to be specified at the open boundary, the array selects the position of the specified elevation with respect to the open boundary.
0	: Not required. Default.
1	: At the open boundary U-node. Default.
2	: At the “nearest” C-node outside the domain.
iloczobv	As previous now for V-node open boundary points.
iqsecobu	Start and end index of the U-open boundary sections where a discharge condition is specified (ityp2dobuv =16).
iqsecobv	Start and end index of the V-open boundary sections where a discharge condition is specified (ityp2dobv =16).

2. Tangential open boundary conditions

ityp2dobx	Type of 2-D tangential open boundary condition for the evaluation of the cross-stream advection term in the V -momentum equation at X-nodes. See Section 12.3.16.2 for details.
0	: Zero gradient condition. Default.
1	: Using external values for U .

2: Upwind scheme.

ityp2doby Type of 2-D tangential open boundary condition for the evaluation of the cross-stream advecting term in the U -momentum equation at Y-nodes. See Section 12.3.16.2 for details.

0: Zero gradient condition. Default.

1: Using external values for V .

2: Upwind scheme.

Note that 2-D tangential open boundary conditions can only be used in case of a 2-D grid (`iopt_grid_nodim=2`) or when the time integration scheme is set to explicit (`iopt_hydro_impl=0`).

21.1.1.2 Data file specifications

```
INTEGER, DIMENSION(2:nofiles) :: iobc2dtype, no2dobuv, no2dobxy
INTEGER, DIMENSION(nobu+nobv,2:nofiles) :: index2dobuv
INTEGER, DIMENSION(nobx+noby,2:nofiles) :: index2dobxy
```

The arrays are needed to obtain, from associated datafiles, the non-harmonic part of ζ , U , V or the components of the surface slope. The type of variable depends on the selected type of open boundary condition. Obviously, the routine is only called when `nofiles>1`.

1. Normal open boundary conditions

no2dobuv Number of data locations within each data file.

iobc2dtype Identifies the variables within the data file. No default.

1: Depth-integrated currents and elevations.

2: Elevations only.

3: Depth-integrated currents only.

index2dobuv Each data file contains a sub-set of open boundary data points. The element `index2dobuv(idat,ifil)` maps, for file `ifil`, the local data index `idat` into a corresponding global open boundary index (between `1:nobu` for U- and `nobu+1:nobu+nobv` for V-open boundaries). For file `ifil`, the actual size of the first dimension of `index2dobuv`, used for the setup, equals `no2dobc(ifil)`.

The procedure is illustrated in Figure 21.1. The filled circles represent open boundary points. The data are spread over 4 data files. The first number in parentheses denotes the number of the data file (between 2 and 6), the second number the open boundary index ranging from 1 to `nobu` at U-nodes and `nobu+1` to `nobu+nobv` at V-nodes. In the example, `nobu=11` and `nobv=8`. Each file contains data for the following points:

- `ifil=2`: data at (U-)o.b. points 1 to 8 on the western boundary
- `ifil=3`: data at (U-)o.b. points 9 to 11 on the eastern boundary
- `ifil=4`: data at (V-)o.b. points 12 to 14 on the southern boundary
- `ifil=5`: data at (V-)o.b. points 15 to 18 on the northern boundary
- `ifil=6`: data at (V-)o.b. point 19 at a river boundary

The definitions in FORTRAN code are:

```
nofiles = 6
no2dobuv = (/8,3,3,4,1/)
index2dobuv(1:8,2) = (/1,2,3,4,5,6,7,8/)
index2dobuv(1:3,3) = (/9,10,11/)
index2dobuv(1:3,4) = (/12,13,14/)
index2dobuv(1:4,5) = (/15,16,17,18/)
index2dobuv(1,6) = 19
```

If `iobc2dtype(ifil)=1`, two data values (one for the depth-integrated current and one for the surface elevation) need to be provided at each location. Otherwise, only one data is defined (either depth-integrated current or elevation).

2. Tangential open boundary conditions

`no2dobxy` Number of data locations within each data file.

`index2dobxy` Each data file contains a sub-set of open boundary data points. The element `index2dobxy(idat,ifil)` maps, for file `ifil`, the local data point `idat` into a corresponding global open boundary index (between 1:`nobx` for X- and `nobx+1:nobx+noby` for Y-open boundaries). For file `ifil`, the actual size of the first dimension, used for the setup, equals `no2dobxy(ifil)`.

The procedure is similar to the one discussed above for normal boundaries.

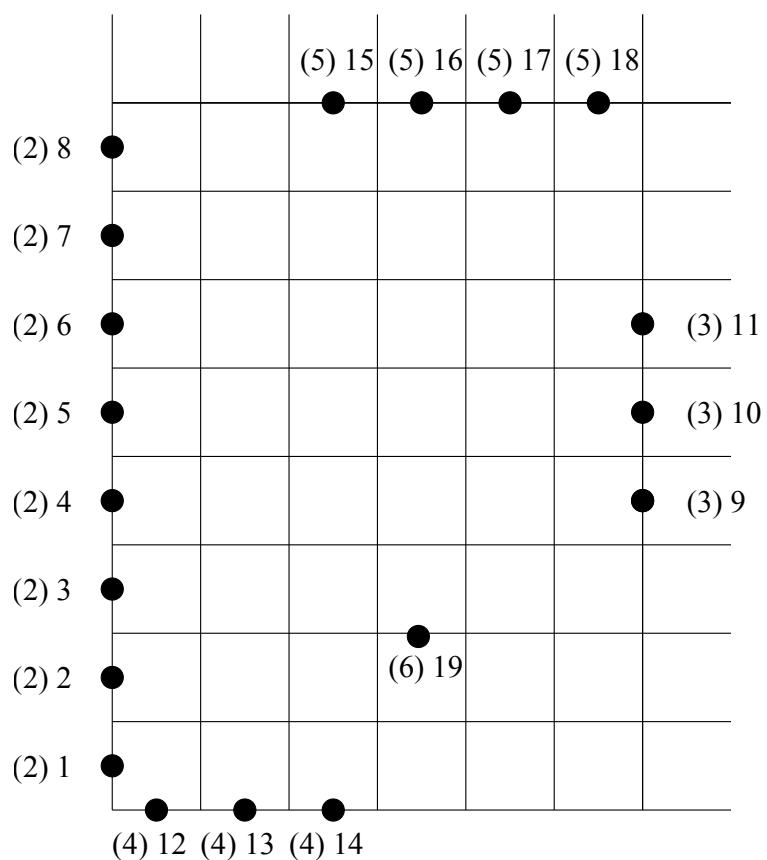


Figure 21.1: Example showing how to define the arrays `no2dobuv` and `index2dobuv`.

21.1.1.3 Amplitudes and phases

```

REAL, DIMENSION(nobu,nconobc) :: udatobu_amp, udatobu_pha, &
                                  & zdatobu_amp, zdatobu_pha
REAL, DIMENSION(nobv,nconobc) :: vdatobv_amp, vdatobv_pha, &
                                  & zdatobv_amp, zdatobv_pha
REAL, DIMENSION(noby,nconobc) :: udatoby_amp, udatoby_pha
REAL, DIMENSION(nobx,nconobc) :: vdatobx_amp, vdatobx_pha

```

where `nconobc` denotes the number of tidal constituents used at open boundaries.

The arrays are used to determine the harmonic part of the external values of ζ , U , V or the surface slope. The type of data depends on the selected type of open boundary condition.

1. Normal open boundary conditions

- `udatobu_amp` Amplitudes of the depth-integrated current U at U-open boundaries [m^2/s].
- `vdatobv_amp` Amplitudes of the depth-integrated current V at V-open boundaries [m^2/s].
- `zdatobu_amp` Amplitudes of the surface elevation ζ or surface slope $\partial\zeta/\partial x_1$ at U-open boundaries [m] or [-].
- `zdatobv_amp` Amplitudes of the surface elevation ζ or surface slope $\partial\zeta/\partial x_2$ at V-open boundaries [m] or [-].
- `udatobu_pha` Phases of the depth-integrated current U at U-open boundaries [rad].
- `vdatobv_pha` Phases of the depth-integrated current V at V-open boundaries [rad].
- `zdatobu_pha` Phases of the surface elevation ζ or surface slope $\partial\zeta/\partial x_1$ at U-open boundaries [rad].
- `zdatobv_pha` Phases of the surface elevation ζ or surface slope $\partial\zeta/\partial x_2$ at V-open boundaries [rad].

2. Tangential open boundary conditions

- `vdatobx_amp` Amplitudes of the depth-integrated V -current at exterior V-nodes adjacent to X-node open boundaries [m/s].
- `udatoby_amp` Amplitudes of the depth-integrated U -current at exterior U-nodes adjacent to Y-node open boundaries [m/s].

- vdatobx_ph** Phases of the depth-integrated V -current at exterior V -nodes adjacent to X -node open boundaries [rad].
- udatoby_ph** Phases of the depth-integrated U -current at exterior U -nodes adjacent to Y -node open boundaries [rad].

Remarks

- By default, the program uses zero values for amplitudes and phases. In that case, the program will (obviously) not make an harmonic expansion of harmonic constituents, even when the tidal frequencies `index.obc` are defined in `usrdef_mod_params`.
- Phases must be given in radians and not in degrees. Conversion from degrees to radians can be made by multiplying the phases with the parameter `degtorad` which is equal to $\pi/180$.
- As stated above, the harmonic amplitudes and phases in the tangential case can only be defined if `iopc_2D_tang=1` and either `iopc_grid_nodim=2` or `iopc_hydro_impl=0`.

21.1.2 Open boundary data for the 2-D mode

The data for 2-D mode open boundary data are defined in `usrdef_2dabc_data` which is called for data at normal open boundaries if

```
iopc_2D=1
modfiles(io_2uvobc,ifil,1)%status='N'
```

or at tangential open boundaries if

```
iopc_2D_tang=1
modfiles(io_2xyobc,ifil,1)%status='N'
```

where `ifil` is the file index of the data file.

FORTRAN template

```
SUBROUTINE usrdef_2dabc_data(iddesc,ifil,ciodatetime,data2d,nodat,novars)
CHARACTER (LEN=lentime), INTENT(INOUT) :: ciodatetime
INTEGER, INTENT(IN) :: iddesc, ifil, nodat, novars
REAL, INTENT(INOUT), DIMENSION(nodat,novars) :: data2d

USE syspars
USE time_routines, ONLY: log_timer_in, log_timer_out
```

```
IMPLICIT NONE
```

```
procname(pglev+1) = 'usrdef_2dcbc_data'
CALL log_timer_in()
...
CALL log_timer_out()
```

```
RETURN
```

```
END SUBROUTINE usrdef_2dcbc_data
```

The arguments of type **INTENT(IN)** have the following meaning

iddesc The file descriptor key id which may take the values

io_2uvcbc in case of normal open boundary data.

io_2xycbc in case of tangential open boundary data.

ifil File number index of the data file (>1).

nodat The number of data points given by **no2dobuv(ifil)** or **no2dobxy(ifil)**.

novars In the normal case, the number of data variables depends on the value of **iobc2dtype(ifil)**.

1: Equals 2 since both depth integrated current and surface elevation data are required.

2: Equals 1 since only surface elevation data are required.

3: Equals 1 since only depth integrated current data are required.

In the tangential case **novars=1** .

The arguments of type **INTENT(INOUT)** need to be defined here. They have the following meaning:

ciodatetime Date/time of the input data in string format¹.

data2d Values of the open boundary data.

A example is given below showing different ways for obtaining the data.
FORTRAN example

¹If the parameter **time_zone** is defined with a non-zero value, the time of the input data must be given in local time.

```
SUBROUTINE usrdef_2dobjc_data(iddesc,ifil,ciodatetime,data2d,nodat,novars)
```

```
! Example code to show how (2-D) open boundary data are obtained.
! Three alternative methods are given
! (1) reading the data using the modfiles attributes and calling
!     open_filepars
! (2) reading without these attributes and calling open_file
! (3) defined in the routine without external data file.
! Method 1 is preferred compared to method 2 since all files
! defined through the modfiles array are automatically
! closed at the end of the run.
! It is assumed that the data file is in ASCII format.
```

```
USE iopars
USE syspars
USE inout_routines, ONLY: open_file, open_filepars
USE time_routines, ONLY: log_timer_in, log_timer_out
```

```
IMPLICIT NONE
```

```
!---arguments
INTEGER, INTENT(IN) :: iddesc, ifil, nodat, novars
CHARACTER (LEN=lentime), INTENT(INOUT) :: ciodatetime
REAL, INTENT(INOUT), DIMENSION(nodat,novars) :: data2d
```

```
!---local variables
INTEGER :: iunit, method = 1
TYPE (FileParams) :: filepars
```

```
procname(pglev+1) = 'usrdef_2dobjc_data'
CALL log_timer_in()
```

```
!
!1. Open data file on first call
!-----
!
```

```
filepars = modfiles(iddesc,ifil,1)
IF (filepars%iostat.EQ.0) THEN
! ---standard method
IF (method.EQ.1) THEN
```

```

        CALL open_filepars(filepars)
        iunit = filepars%iunit
!  ---alternative method
ELSEIF (method.EQ.2) THEN
    CALL open_file(iunit,'obc.dat','IN',form)
    modfiles(iddesc,ifil,1)%iunit = iunit
    modfiles(iddesc,ifil,1)%iostat = 1
    GOTO 1000
!  ---without data file
ELSEIF (method.EQ.3) THEN
    modfiles(iddesc,ifil,1)%iostat = 1
    GOTO 1000
ENDIF
ENDIF

!
!2. Read data
!-----
!

IF (method.LT.3) THEN
    iunit = modfiles(iddesc,ifil,1)
    READ (iunit,'(A)') ciodatetime
    READ (iunit,*,END=99) data2d
ELSE
    ciodatetime = ...
    data2d = ...
ENDIF

CALL log_timer_out()

RETURN

100 modfiles(iddesc,ifil,1)%iostat = 3

RETURN

END SUBROUTINE usrdef_2dabc_data

```

The following remarks should be given

- The routine is called two times at the initial time. Firstly to open (even-

tually) the data file and to set the iostat attribute to 1 (which is obligatory and automatically done if the file is opened with `open_filepars`) and secondly to read the first series of data.

- It is useful to reset the iostat attribute to 3, when an end of file condition occur. In that case, the program will automatically issue an error message and abort.

21.1.3 2-D open boundary conditions: code example

The code below shows an example of a realistic setup of 2-D open boundary conditions. There are three steps. Firstly the attributes of the appropriate forcing files are defined in `usrdef_mod_params`.

```
SUBROUTINE usrdef_mod_params

USE iopars
USE inout_routines, ONLY: close_file, open_file
USE time_routines, ONLY: log_timer_in, log_timer_out

IMPLICIT NONE

!---local variables
INTEGER :: ifil, iunit
CHARACTER(LEN=40) :: filename

procname(pglev+1) = 'usrdef_mod_params'
CALL log_timer_in()

...
!

!1. Attributes of forcing files
!-----
!
!1.1 status
!-----
!

modfiles(io_2uvobc,1,1)%status = 'N'
modfiles(io_2uvobc,2,1)%status = 'R'
modfiles(io_2uvobc,3:5,1)%status = 'N'
```

```

!
!1.2 File format
!-----
!

modfiles(io_2uvobc,2,1)%form = 'N'
modfiles(io_2uvobc,3:5,1)%form = 'A'

!

!1.3 File names
!-----
!

!--open sea
modfiles(io_2uvobc,2,1)%filename = 'elevation_nst.nc'

!--rivers
CALL open_file(iunit,'riverfilenames.dat','IN','A')
ifil_110: DO ifil=3,5
    READ (iunit,*) filename
    modfiles(io_2uvobc,ifil,1)%filename = TRIM(filename)
ENDDO ifil_110
CALL close_file(iunit,'A')

!

!1.4 Times for update
!-----
!

modfiles(io_2uvobc,2,1)%tlims = (/0,int_fill,5/)
ifil_140: DO ifil=3,5
    modfiles(io_2uvobc,ifil,1)%tlims = (/0,int_fill,30/)
ENDDO ifil_140

...

CALL log_timer_out()

END SUBROUTINE usrdef_mod_params

```

- The first forcing data file (ifil=2), contains the elevation data at all

open sea boundaries. They are obtained (in this setup) through nesting from a larger model grid. The data are therefore accessible in standard format. Update of the data is made at each 5th time step.

- The next three files contain discharge data at three locations. Update is made after each 30 time steps.

Next, the type of open boundary conditions and the specifications for reading the data are defined. It is assumed that all river boundaries are at U-nodes.

```

SUBROUTINE usrdef_2dobjc_spec(iddesc)

USE gridpars
USE iopars
USE obconds
USE time_routines, ONLY: log_timer_in, log_timer_out

IMPLICIT NONE

!---arguments
INTEGER, INTENT(IN) :: iddesc

!---local variables
INTEGER :: l

procname(pglev+1) = 'usrdef_2dobjc_spec'
CALL log_timer_in()

!
!1. Type of o.b.c
!-----
!
!---open sea
ityp2dobj(1:nosbu) = 12
ityp2dobj(1:nosbv) = 12

!---river
ityp2dobj(nosbu+1:nosbu+4) = 12
ityp2dobj(nosbu+5:nosbu+12) = 15

```

```

!
!2. Data file specifiers
!-----
!
!--open sea boundaries
no2dobuv(2) = nosbu+nosbv
iobc2dtype(2) = 2
index2dobuv(1:nosbu,2) =  (/1,l=1,nosbu/)
index2dobuv(nosbu+1:nosbu+nosbv,2) =  (/1,l=nobu+1,nobu+nosbv/)

!--rivers
!--number of data per time step
no2dobuv(3) = 4; no2dobuv(4) = 1; no2dobuv(5) = 7
!--type of input data
iobc2dtype(3) = 2; iobc2dtype(4) = 3; iobc2dtype(5) = 3
!--index mapping array
index2dobuv(1:4,3) = (/1,l=nosbu+1,nosbu+4/)
index2dobuv(1,4) = nosbu+5
index2dobuv(1:7,5) =(/1,l=nosbu+6,nosbu+12/)

CALL log_timer_out()

RETURN

END SUBROUTINE usrdef_2dcbc_spec

```

- A Riemann open boundary condition with elevation only is applied at all open sea boundaries and the first four river boundaries. A discharge condition is applied at the other river locations.
- The open sea boundary files and the first four river locations contain elevation data. These four locations actually cover one river whose width consists of four grid cells since the same input file is used (see above). The other data files contain the river discharges rates.
- The index mapping array indicates that a separate value is provided at each open boundary location.

The third step consists in defining data input for the rivers only, since the open sea boundary input is automatically provided by the model when the **status** attribute of the input file has been set to 'R'.

```

SUBROUTINE usrdef_2dobjc_data(iddesc,ifil,ciodatetime,data2d,nodat,novars)

USE iopars
USE syspars
USE inout_routines, ONLY: open_filepars
USE time_routines, ONLY: log_timer_in, log_timer_out, convert_date

IMPLICIT NONE

!---arguments
CHARACTER (LEN=lentime), INTENT(INOUT) :: ciodatetime
INTEGER, INTENT(IN) :: iddesc, ifil, nodat, novars
REAL, INTENT(INOUT), DIMENSION(nodat,novars) :: data2d

!---local variables
INTEGER :: iunit
INTEGER, DIMENSION(7) :: intdate
REAL :: gridspan, qdis, zetobc

procname(pglev+1) = 'usrdef_2dobjc_data'

CALL log_timer_in()

IF (modfiles(io_2uvobc,ifil,1)%iostat.EQ.0) THEN
  CALL open_filepars(modfiles(io_2uvobc,ifil,1),'IN')
  iunit = modfiles(io_2uvobc,ifil,1)%iunit
  READ (iunit,'(5/)')
  GOTO 1000
ENDIF

iunit = modfiles(io_2uvobc,ifil,1)%iunit
intdate(7) = 0

!---first river
IF (ifil.EQ.3) THEN
  READ (iunit,*) intdate(1:6), zetobc
  data2d(:,1) = zetobc
!---other rivers
ELSE
  READ(iunit,*) intdate(1:6), qdis

```

```

gridspan = SIZE(data2d(:,1))
data2d(:,1) = qdis/gridspan
ENDIF

!---convert date/time to string format
ciodate = convert_date(intdate)

1000 CALL log_timer_out()

RETURN

END SUBROUTINE usrdef_2dcbc_data

```

- The data files are opened on first call.
- The elevation data for the first river are read.
- The discharge data are read and divided by the number of grid cells spanning the width of the river at its outlet.
- Since the data files are opened with `open_filepars`, they will be automatically closed by the program at the end of the simulation.

21.2 3-D open boundary conditions

21.2.1 Open boundary specifications for the 3-D mode

Arrays for setting up open boundary conditions in the 3-D case are defined in `usrdef_profobc_spec`. The routine is used by the program for 3-D baroclinic currents (used either for normal as for tangential conditions) and all 3-D scalar quantities for which a transport equations needs to be solved (currently T , S , sediments, biological state variables). No conditions are to be defined for turbulence variables, which are solved with the default zero gradient condition at the open boundaries.

The routine is called when

```
modfiles(iddesc,1,1)%status = 'N'
```

with `iddesc` defined below and when the appropriate switch is set (see below).

FORTRAN template

```

SUBROUTINE usrdef_profobc_spec(iddesc,itypobux,itypobvy,iprofobux,&
                               & iprofobvy,noprofsd,indexprof,nofiles,&
                               & nobux,nobvy,novars)
INTEGER, INTENT(IN) :: iddesc, nobux, nobvy, nofiles, novars
INTEGER, INTENT(INOUT), DIMENSION(2:nofiles,novars) :: noprofdsd
INTEGER, INTENT(INOUT), DIMENSION(nobux,novars) :: iprofobux, ityobux
INTEGER, INTENT(INOUT), DIMENSION(nobvy,novars) :: iprofobvy, itypobvy
INTEGER, INTENT(INOUT), DIMENSION(nobux+nobvy,2:nofiles,novars) :: &
                               & indexprof

```

USE time_routines, ONLY: log_timer_in, log_timer_out

IMPLICIT NONE

```

procname(pglev+1) = 'usrdef_profobc_spec'
CALL log_timer_in()
...
CALL log_timer_out()

```

RETURN

END SUBROUTINE usrdef_profobc_spec

where the **INTENT(IN)** arguments have the following meaning:

iddesc The file descriptor key id of the 3-D quantity which may take the following values.

io_3uvobc Baroclinic horizontal currents at normal open boundaries if **iopt.obc_3D=1**.

io_3xyobc Horizontal currents at tangential open boundaries if **iopt.obc_3D_tang=1**.

io_salobc Salinity if **iopt.obc_sal=1**.

io_tmppobc Temperature if **iopt.obc_temp=1**.

io_sedobc Sediment fractions if **iopt.obc_sed=1**.

io_bioobc Biological state variables if **iopt.obc_bio=1**.

nobux Equal to **nobx** if **iddesc = io_3xyobc** or **nobu** otherwise.

nobvy Equal to **noby** if **iddesc = io_3xyobc** or **nobv** otherwise.

nofiles The number of data files plus 1 (the file index for data files ranges from 2 to **nofiles**) defined by

```
nofiles = COUNT(modfiles(iddesc,:,:,1)%status.NE.'0')
```

novars The number of variables for which open boundary conditions are defined. For currents, temperature and salinity its value is 1. **novars** equals the number of sediment fractions **nf** for sediments, for the circulation model and the number of 3-D state variables **nobiowars3d** in the biological case.

21.2.2 Type of 3-D open boundary conditions

itypobuy Selects the type of open boundary condition used at U-nodes (normal case) or Y-nodes (tangential case) at those locations where no profile number has been defined with a positive value (see below). The type of condition depends on the value of **iddesc**:

io_3uvobc 0: First order zero gradient condition. Default.

- 1: Second order zero gradient condition.
- 2: Local solution.
- 3: Radiation condition using internal wave speed.
- 4: Orlanski type of radiation condition.
- 5: Discharge condition.

io_3xyobc 0: First order zero gradient condition. Default.

- 1: Upwind scheme.

scalars 0: First order zero gradient condition. Default.

- 1: Radiation condition using the internal wave speed.
- 2: Orlanski condition.
- 3: Thatcher-Harleman condition.

itypobvx Selects type of open boundary condition at V-nodes (normal case) or X-nodes (tangential case) at those locations where no profile number has been defined with a positive value (see below). Definitions are the same as above for **itypobux**.

iprofobux Profile number used at U-node (normal case) or X-node (tangential case) open boundaries. In case of multi-variable data (**novars>1**), different profile numbers can be selected for different variables (sediment fractions or biological state variables). Negative values are not allowed. If a profile number is zero, the open boundary condition is defined by the value of **itypobux**. In the normal case, a zero

gradient condition is applied if the profile number is zero or when the profile contains flagged data, i.e. values equal to `float_fill`.

`iprofobvy` Profile number used at V-node (normal case) or Y-node (tangential case) open boundaries. In case of multi-variable data (`novars>1`), different profile numbers can be selected for different variables (sediment fractions or biological state variables). Negative values are not allowed. If a profile number is zero, the open boundary condition is defined by the value of `itypobvy`. In the normal case, a zero gradient condition is applied if the profile number is zero or when the profile contains flagged data, i.e. values equal to `float_fill`.

Remarks

- The same profile number can be used at different open boundary locations or for different variables. For example

$$\begin{aligned} \text{iprofobux(ii1,ivar1)} &= \text{iprofobvy(ii1,ivar1)} & \text{or} \\ \text{iprofobux(ii1,ivar1)} &= \text{iprofobux(ii2,ivar2)} & \text{or} \\ \text{iprofobvy(jj1,ivar1)} &= \text{iprofobuy(jj2,ivar2)} & \text{or} \\ \text{iprofobux(ii1,ivar1)} &= \text{iprofobvx(jj2,ivar2)} \end{aligned}$$

for any `ii1`, `ii2`, `jj1`, `jj2`, `ivar1`, `ivar2`.

- The data profiles itself are provided as time series in `usrdef_profobc_data` or by a standard forcing file.
- By default, `itypobux`, `itypobvy`, `iprofobux`, `iprofobvy` are set to zero, giving a zero gradient condition.
- In the sediment case (`iddesc=io_sedobc`), the arrays `itypobux`, `itypobvy`, `iprofobuyx` and `iprofobvy` only need to be defined for suspended fractions, i.e. `sedpart%iopt_suseq=1`.

21.2.3 Data file specifications

`noprofsd` Number of data profiles for each data file.

`indexprof` Each data file contains a sub-set of open boundary profiles. The element `indexprof(iprof,ifil,ivar)` maps, for file `ifil` and variable `ivar`, the local profile number `iprof` into a corresponding “global” index as defined by `iprofobux` and `iprofobvy`. The actual size of the first dimension for file `ifil` equals `noprofsd(ifil,ivar)`. If not defined and `nofiles=2`, the program sets `indexprof(1:noprofsd(ifil,ivar),ifil,ivar) = ((1,2,...,noprofsd(ifil,ivar)))/.`

Remarks

In case of sediment fractions, `indexprof` only needs to be defined for fractions in suspended mode.

The purpose of the array `indexprof` can be illustrated with the following FORTRAN code example (assuming `novars=1`)

```
! ---Vertical profile arrays read from the data files
REAL, DIMENSION(maxprofs,nz,2:nofiles) :: profdat
! ---Vertical profile arrays used at open boundaries
REAL, DIMENSION(noprofs,nz,1) :: obcdat
```

where `maxprofs` denotes the maximum number of profiles read from each data files and `noprofs` the total number of profiles (over all variables) used at open boundaries. The input profiles from `profdat` are then stored into `obcdat` using

```
ifil_110: DO ifil=2,nofiles
  IF (noprofsd(ifil,1).GT.0) THEN
    1_111: DO l=1,noprofsd(ifil,1)
      iprof = indexprof(l,ifil,1)
      obcdat(iprof,:,1) = profdata(l,:,ifil)
    ENDDO 1_111
  ENDIF
ENDDO ifil_110
```

Let `noprofsivar` = `MAX(MAXVAL(iprofobu(:,ivar)),MAXVAL(iprofobv(:,ivar)))` where `ivar` is the variable index between 1 and `novars`. The following constraints apply

- For each `iprof` between 1 and `noprofsivar`, there is at least one array element of `iprofobu` or `iprofobv` equal to `iprof`.
- The value of `indexprof(iprof,ifil,ivar)` must be between and 1 and `noprofsivar`.
- For each `ivar` between 1 and `novars` and `iprof` between 1 and `noprofsivar`, there must correspond one and only one data profile `iprofdat` for which `indexprof(iprofdat,ifil,ivar)=iprof`.

The procedure is illustrated in Figure 21.2. The filled circles represent open boundary points. The data are spread over 4 data files. The first number in parentheses denotes the number of the data file (between 2 and 5), the second one is the number of the profile applied at the open boundary location. In the example a zero gradient condition is applied at the eastern boundary. The data files contain the following profiles

- ifil=2: profiles 1 and 2
- ifil=3: profile 3
- ifil=4: profile 4
- ifil=5: profile 5

In FORTRAN code, the definitions become

```

nobu = 11; nobv = 8
itypobux = 0; itypobvy = 0
iprofobux = (/1,1,1,1,2,2,2,2,0,0,0/)
iprofobvy = (/3,3,3,4,4,4,4,5/)
nofiles = 5; noprofsd(2:5,1) = (/2,1,1,1/)
indexprof(1:2,2,1) = (/1,2/)
indexprof(1,3,1) = 3
indexprof(1,4,1) = 4
indexprof(1,5,1) = 5

```

21.2.4 Open boundary data for the 3-D mode

The data for 3-D mode open boundary conditions are defined in `usrdef_profobc_data` which is called if the appropriate switch (`iopt_abc_3D`, `iopt_abc_sal`, `iopt_abc_temp`, `iopt_abc_sed`, `iopt_abc_bio`) is set to 1 and

```
modfiles(iddesc,ifil,1)%status='N'
```

where `iddesc` if the file descriptor id and `ifil` the index of the data file.

FORTRAN template

```

SUBROUTINE usrdef_profobc_data(iddesc,ifil,ciodatetim,psiprofdat,&
                               & numprofs,nobcvars)
CHARACTER (LEN=lentime), INTENT(INOUT) :: ciodatetim
INTEGER, INTENT(IN) :: iddesc, ifil, nobcvars, numprofs
REAL, INTENT(INOUT), DIMENSION(numprofs,nz,nobcvars) :: psiprofdat

USE gridpars
USE syspars
USE time_routines, ONLY: log_timer_in, log_timer_out

IMPLICIT NONE

```

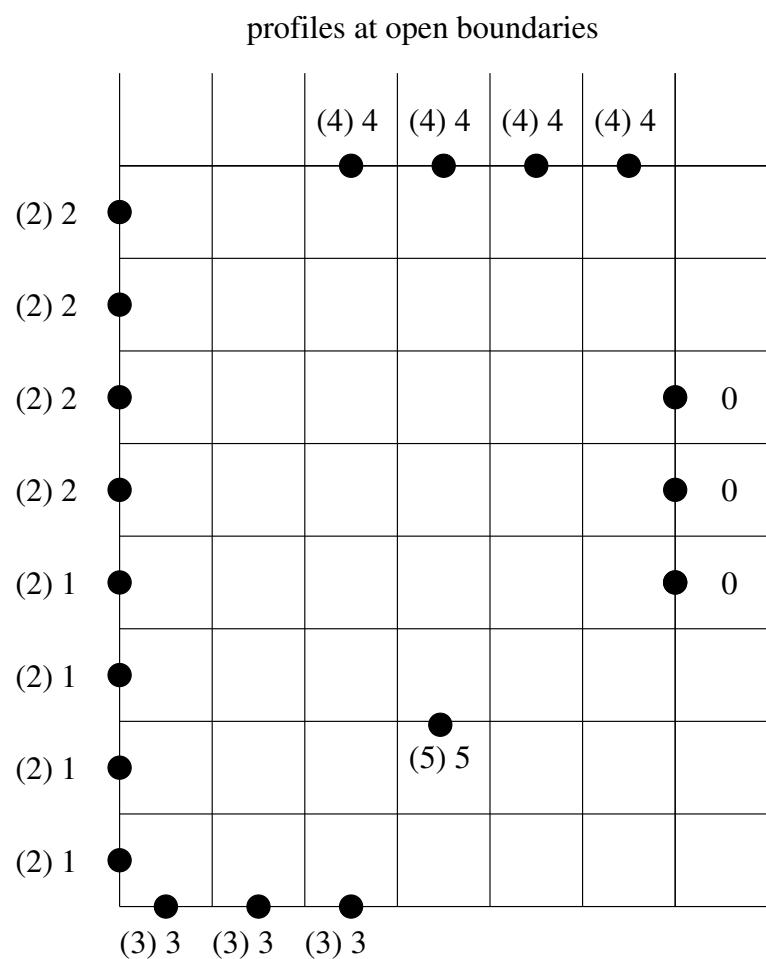


Figure 21.2: Example how to define the open boundary specifier arrays.

```

procname(pglev+1) = 'usrdef_profobc_data'
CALL log_timer_in()
...
CALL log_timer_out()

```

RETURN

END SUBROUTINE usrdef_profobc_data

The **INTENT(IN)** arguments have the following meaning:

iddesc The file descriptor key id of the 3-D quantity which may take the following values:

- io_3uvobc** horizontal baroclinic currents ($\delta u, \delta v$) at normal open boundaries
- io_3xyobc** horizontal currents (u, v) at tangential open boundaries
- io_salobc** salinity
- io_tmlobc** temperature
- io_sedobc** sediment fractions
- io_bioobc** biological state variables

ifil File number index of the data file. Must be greater than 1.

numprofs The number of input profiles equal to **MAXVAL(noprofsd(ifil,:))**.

nobcvars The number of variables for which open boundary conditions are defined. For currents, temperature and salinity its value is 1. For sediments and biology **nobcvars** equals respectively the number of sediment fractions **nf** or the number of state variables **nobiobcvars3d**.

The following **INTENT(OUT)** variables must be defined here

ciodatetime Date/time of the profile data in string format¹.

psiprofdat Values of the profile data.

Values in the data array which are equal to the flag value **float_fill** are considered as flagged. In that case the open boundary condition at that specific vertical location (only) is changed from an external data profile to a zero gradient condition.

In the user defined routines, it is necessary to define separate cases when boundary conditions for different variables (salinity, sediment, biology, 3D velocity profile) are defined.

21.2.5 3-D open boundary conditions: code example

The example code below shows how to set up 3-D open boundary conditions for a realistic case. As for the 2-D case there are three steps. In the first one the attributes of the forcing files are defined in `usrdef_mod_params`.

```
SUBROUTINE usrdef_mod_params
```

```
USE iopars
USE switches
USE syspars
USE time_routines, ONLY: log_timer_in, log_timer_out
```

```
IMPLICIT NONE
```

```
procname(pglev+1) = 'usrdef_mod_params'
CALL log_timer_in()
```

```
....
```

```
nosbu = ?; nosbv = ?
nrvbu = 8; nrvbv = 0
```

```
...
```

```
!---open boundary conditions (3-D current)
IF (iopt_abc_3D.EQ.1) THEN
    modfiles(io_3uvobc,1:2,1)%status = 'N'
    modfiles(io_3uvobc,2,1)%tlims = (/0,0,1/)
ENDIF
```

```
!---open boundary conditions (salinity)
IF (iopt_abc_sal.EQ.1) THEN
    modfiles(io_salobc,1,1)%status = 'N'
!  --open sea
    modfiles(io_salobc,2,1)%status = 'R'
    modfiles(io_salobc,2,1)%form = 'N'
    modfiles(io_salobc,2,1)%filename = 'Salinity_abc.nc'
    modfiles(io_salobc,2,1)%tlims = (/0,int_fill,10/)
!  --first river
    modfiles(io_salobc,3,1)%status = 'N'
    modfiles(io_salobc,3,1)%form = 'A'
```

```

    modfiles(io_salobc,3,1)%filename = 'Scheldt_Salinity.dat'
    modfiles(io_salobc,3,1)%tlims = (/0,int_fill,20/)
! --other rivers
    modfiles(io_salobc,4,1)%status = 'N'
    modfiles(io_salobc,4,1)%tlims = (/0,0,1/)

ENDIF

!---open boundary conditions (temperature)
IF (iopt_obi_temp.EQ.1) THEN
    modfiles(io_tmppobc,1,1)%status = 'N'
! --open sea
    modfiles(io_tmppobc,2,1)%status = 'R'
    modfiles(io_tmppobc,2,1)%form = 'N'
    modfiles(io_tmppobc,2,1)%filename = 'Temperature_obi.nc'
    modfiles(io_tmppobc,2,1)%tlims = (/0,int_fill,10/)
! --first river
    modfiles(io_tmppobc,3,1)%status = 'N'
    modfiles(io_tmppobc,3,1)%form = 'A'
    modfiles(io_tmppobc,3,1)%filename = 'Scheldt_Temperature.dat'
    modfiles(io_tmppobc,3,1)%tlims = (/0,int_fill,20/)
! --other rivers
    modfiles(io_tmppobc,4,1)%status = 'N'
    modfiles(io_tmppobc,4,1)%tlims = (/0,0,1/)

ENDIF

...
CALL log_timer_out()

RETURN

END SUBROUTINE usrdef_mod_params

```

- There are 8 river boundaries at U-nodes.
- One forcing file is defined for baroclinic currents. The data are uniform in time and obtained at the initial time.
- There are three forcing files for salinity and temperature. The first one (ifil=2) will read the data for open sea boundaries from a file in standard format. The data were written by a previous run on a coarser grid in which the current grid is nested. It is assumed in the example

that the second file (ifil=3) contains data for one river. The third one (ifil=4) then contains time-independent data for the remaining rivers.

The second step consists in defining the type of conditions and providing the instructions how to read in the data.

```

SUBROUTINE usrdef_profobc_spec(iddesc,itypobux,itypobvy,iprofobux,&
                               & iprofobvy,noprofsd,indexprof,nofiles,&
                               & nobux,nobvy,novars)

USE gridpars
USE iopars
USE time_routines, ONLY: log_timer_in, log_timer_out

IMPLICIT NONE

!---arguments
INTEGER, INTENT(IN) :: iddesc, nobux, nobvy, nofiles, novars
INTEGER, INTENT(INOUT), DIMENSION(2:nofiles,novars) :: noprofsd
INTEGER, INTENT(INOUT), DIMENSION(nobux,novars) :: iprofobux, itypobux
INTEGER, INTENT(INOUT), DIMENSION(nobvy,novars) :: iprofobvy, itypobvy
INTEGER, INTENT(INOUT), DIMENSION(nobux+nobvy,2:nofiles,novars) :: &
                               & indexprof

!---local variables
INTEGER :: l

procname(pglev+1) = 'usrdef_profobc_spec'
CALL log_timer_in()

IF (iddesc.EQ.io_salobc.OR.iddesc.EQ.io_tmlobc) THEN
    noprofsd(2,1) = nosbu + nosbv
    noprofsd(3:4,1) = 1
    iprofobux(1:nosbu,1) = (/ (l, l=1, nosbu) /)
    iprofobvy(1:nosbv,1) = (/ (l, l=nosbu+1, nosbu+nosbv) /)
    iprofobux(nosbu+1:nosbu+4,1) = nosbu+nosbv+1
    iprofobux(nosbu+5:nobu,1) = nosbu+nosbv+2
    indexprof(1:nosbu+nosbv,2,1) = (/ (l, l=1, nosbu+nosbv) /)
    indexprof(1,3,1) = nosbu+nosbv+1
    indexprof(1,4,1) = nosbu+nosbv+2
ELSEIF (iddesc.EQ.io_3uvobc) THEN

```

```

noprofsd(2,1) = 1
iprofobux(nosbu+1:nobu,1) = 1
indexprof(1,2,1) = 1
ENDIF

CALL log_timer_out()

RETURN

END SUBROUTINE usrdef_profobc_spec

```

Remarks

- Since the profile data at all open sea boundaries are obtained from nesting, a profile number is defined for each location, giving a total number of nosbu+nosbv profiles. They are stored in the data file using the procedures followed in COHERENS, i.e. U-nodes first, V-nodes next.
- Two profiles are defined for salinity and temperature at river boundaries. One profile is used for the first (Scheldt) river. The width of this river at its mouth covers four grid cells, which means that four river open boundary locations have to be defined. The same salinity and temperature profile is used at these four locations. The second profile applies for the remaining rivers (all located at U-nodes).
- A zero gradient condition is applied for the baroclinic current at open sea boundaries. The same (time-independent) current profile is used at all rivers.

The third step consists of defining data input for the rivers.

```

SUBROUTINE usrdef_profobc_data(iddesc,ifil,ciodatetim,psiprofdat,&
                               & numprofs,nobcvvars)

USE iopars
USE gridpars
USE syspars
USE timepars
USE inout_routines, ONLY: open_filepars
USE time_routines, ONLY: log_timer_in, log_timer_out, convert_date

IMPLICIT NONE

```

```

!---arguments
CHARACTER (LEN=lentime), INTENT(INOUT) :: ciodatetime
INTEGER, INTENT(IN) :: iddesc, ifil, nobcvars, numprofs
REAL, INTENT(INOUT), DIMENSION(numprofs,nz,nobcvars) :: psiprofdat

!--local variables
INTEGER, DIMENSION(7) :: intdate
INTEGER :: i, istat, iunit
REAL :: xdat

procname(pglev+1) = 'usrdef_profobc_data'
CALL log_timer_in()

!

!1. Open data files on first call
!-----
!

IF (modfiles(iddesc,ifil,1)%iostat.EQ.0) THEN
  SELECT CASE (iddesc)
    CASE (io_3uvobc)
      modfiles(iddesc,ifil,1)%iostat = 1
    CASE (io_salobc,io_tmlobc)
      IF (ifil.EQ.3) THEN
        CALL open_filepars(modfiles(iddesc,ifil,1),'IN')
        iunit = modfiles(iddesc,ifil,1)%iunit
        READ (iunit,'(5/)')
      ELSE
        modfiles(iddesc,ifil,1)%iostat = 1
      ENDIF
    END SELECT
    GOTO 1000
  ENDIF

!

!2. Read data
!-----
!

SELECT CASE (iddesc)

```

```

CASE (io_3uvobc)
  ciodatetime = CStartTime
  psiprofdat = 0.0
CASE (io_salobc)
!  ---first river
  IF (ifil.EQ.3) THEN
    intdate(6:7) = 0
    iunit = modfiles(iddesc,ifil,1)%iunit
    READ (iunit,*) intdate(1:5), xdat, istat
    psiprofdat = MERGE(xdat,float_fill,istat.GT.0)
    ciodatetime = convert_date(intdate)
!  ---other rivers
  ELSE
    ciodatetime = CStartTime
    psiprofdat = 0.0
  ENDIF
CASE (io_tmppobc)
!  ---first river
  IF (ifil.EQ.3) THEN
    intdate(6:7) = 0
    iunit = modfiles(iddesc,ifil,1)%iunit
    READ (iunit,*) intdate(1:5), xdat, istat
    psiprofdat = MERGE(xdat,float_fill,istat.GT.0)
    ciodatetime = convert_date(intdate)
!  ---other rivers
  ELSE
    ciodatetime = CStartTime
    psiprofdat = float_fill
  ENDIF
END SELECT

1000 CALL log_timer_out()

RETURN

END SUBROUTINE usrdef_profobc_data

```

- On first call the `status` attribute is set to 1, to inform the program that the file has been “opened” even when there is no forcing data file (case when `ifil=4`).
- The baroclinic profile is set to zero at all open boundary locations,

meaning that the discharge is uniform over the vertical. Important to note here is that a zero profile is to be preferred at river open boundaries compared to a zero gradient condition. This practice is recommended in general.

- The Scheldt data are read together with a status flag. If the flag indicates invalid data, a zero gradient condition will be used instead.
- For the other rivers a zero salinity profile and a zero gradient condition for temperature are taken at the inlet.
- The date and time are first read in vector format and then converted into string format using the `convert_date` routine.
- Note (again) that since the data files are opened with `open_filepars`, they will be automatically closed by the program at the end of the simulation.

21.3 Nesting

The objective of nesting is to interpolate values of specific model arrays to external data points, which constitute the open boundaries of one or more (destination) sub-grids nested within the main (source) grid. The interpolated data are written to output files in standard COHERENS format. A different file is used for each type of variable and each sub-grid. Writing of a file is enabled by setting `modfiles(iddesc,iset,2)%status='W'` in `usrdef_mod_params` where

`iddesc` the file descriptor which can take the following values:

`io_2uvnst` 2-D mode variables (transport and surface elevation)
`io_3uvnst` 3-D baroclinic current
`io_salnst` salinity
`io_tmprnst` temperature
`io_sednst` sediment fractions
`io_bionst` biological state variables

`iset` number of the nested sub-grid (between 1 and `nonestsets`)

The following five types of horizontal interpolation (depending on the type of variable) need to be considered:

- From main grid C-nodes to sub-grid C-nodes: 3-D scalars needed for advective fluxes of scalars at U and/or V open boundaries.
- From main grid C-nodes to sub-grid U- or V-nodes: elevations ζ needed when the open boundary conditions for transports require external elevation data at U- and/or V-nodes.
- From main grid U-nodes to sub-grid U-nodes: X-components of the transports U and baroclinic currents δu needed when the open boundary conditions at U-nodes require external values of transport data and/or external profiles of the baroclinic current.
- From main grid V-nodes to sub-grid V-nodes: Y-components of the transports V and baroclinic currents δv needed when the open boundary conditions at V-nodes require external values of transport data and/or external profiles of the baroclinic current.
- From main grid U-nodes to sub-grid V-nodes: X-components of transports U and currents u needed as (tangential) boundary condition for the cross-stream advective flux in the Y-component of the momentum equation. These sub-grid nodes will, for convenience, be denoted below as X-nodes.
- From main grid V-nodes to sub-grid U-nodes: Y-components of the transports V and currents v needed as (tangential) boundary condition for the cross-stream advective flux in the X-component of the momentum equation. These sub-grid nodes will, for convenience, be denoted below as Y-nodes.

In summary, the nesting procedure is defined as follows:

1. Set `iopt_nests` to 1 in `usrdef_mod_params`.
2. Define `nonestsets` as the number of nested sub-grids in `usrdef_mod_params`.
3. Define the number of data points in `usrdef_nstgrd_spec`.
4. Define the positions of the data points in `usrdef_nstgrd`.
5. Enable the interpolation and writing of specific model variables by setting `modfiles(iddesc,iset,2)%status='W'` in `usrdef_mod_params` with `iset` between 1 and `nonestsets`.

The source code of the three subroutines, mentioned above, must be provided in the file `Usrdef_Nested_Grids.f90`. They are described below.

21.3.1 Specifications of the sub-grid open boundaries

This section describes the parameters used for nesting. This includes the number of open boundary locations (horizontal as well as vertical) for each sub-grid and open boundary node, and the type of nesting. They are either defined in `usrdef_nstgrd_spec` if

```
modfiles(io_nstgrd_spc,1,1)%status = 'N'
```

or in the `CIF` block `nstgrd_spec` if it exists and the `CIF` is activated or from a standard forcing file if

```
modfiles(io_nst_spc,1,1)%status = 'R'
```

`FORTTRAN` template

```
SUBROUTINE usrdef_nstgrd_spec
```

```
USE iopars
```

```
USE nestgrids
```

```
USE time_routines, ONLY: log_timer_in, log_timer_out
```

```
IMPLICIT NONE
```

```
procname(pglev+1) = 'usrdef_nstgrd_spec'
```

```
CALL log_timer_in()
```

```
...
```

```
CALL log_timer_out()
```

```
RETURN
```

```
END SUBROUTINE usrdef_nstgrd_spec
```

The following arrays may be defined here

```
INTEGER, DIMENSION(nonestsets) :: inst2dtype, nohnstglbc, nohnstglbu, &
& nohnstglbv, nohnstglx, nohnstglby, novnst
```

where

`nohnstglbc` Number of C-node sub-grid open boundary points in the horizontal (used for open boundary conditions of scalars and baroclinic currents).

`nohnstglbu` Number of U-node sub-grid open boundary points in the horizontal (used for normal open boundary conditions).

nohnstglbv Number of V-node sub-grid open boundary points in the horizontal (used for normal open boundary conditions).
nohnstglbx Number of X-node sub-grid open boundary points in the horizontal (used for tangential open boundary conditions).
nohnstglby Number of Y-node sub-grid open boundary points in the horizontal (used for tangential open boundary conditions).
novnst Number of vertical grid cells of the sub-grid.
inst2dtype Selects the type of data for 2-D nesting.
 1: transports and elevations
 2: elevations
 3: transports

21.3.2 Locations of the sub-grid open boundaries

The positions of the sub-grid open boundary data locations with index *iset* in Cartesian or spherical coordinates (depending on the value of *iopt_grid_sph*) are obtained in **usrdef_nstgrd**. The routine is called when

```

  iopt_nests=1
  modfiles(io_nstgrd,iset,1)%status='N'

FORTRAN template

SUBROUTINE usrdef_nstgrd(iset,nhdat,nzdat,xcoord,ycoord,scoord,cnode)
CHARACTER (LEN=lennode), INTENT(IN) :: cnode
INTEGER, INTENT(IN) :: iset, nhdat, nzdat
REAL, INTENT(OUT), DIMENSION(nhdat) :: xcoord, ycoord
REAL, INTENT(OUT), DIMENSION(nhdat,nzdat) :: coord

USE iopars
USE syspars
USE time_routines, ONLY: log_timer_in, log_timer_out

IMPLICIT NONE

procname(pglev+1) = 'usrdef_nstgrd'
CALL log_timer_in()
...
CALL log_timer_out()

```

RETURN

END SUBROUTINE `usrdef_nstgrd`

where

- iset** The index number of the sub-grid.
- cnode** The node on the destination grid at which the interpolation is performed and which may take one of the values 'C', 'U', 'V', 'X', 'Y'.
- nhdat** The number of sub-grid points in the horizontal, equal to the value of either `nohnstglbc(iset)`, `nohnstglbu(iset)`, `nohnstglbv(iset)`, `nohnstglx(iset)` or `nohnstglby(iset)` depending on the value of **cnode**.
- nzdat** The number of data points in the vertical equal to `novnst(iset)`.

Provided that the conditions above are fulfilled, the routine is called for each sub-grid at most 5 times with respectively `cnode='C','U','V','X','Y'`.

The following coordinate arrays are to be defined:

- xcoord** X-coordinates of the sub-grid open boundary data points at the specific node [m or degrees longitude, positive East].
- ycoord** Y-coordinates of the sub-grid open boundary data points at the specific node [m or degrees latitude, positive North].
- scoord** σ -coordinates of the sub-grid locations when `nzdat>0`. If not provided, a spatially uniform σ -grid is taken by default.

Chapter 22

Surface forcing

This chapter discusses the setup of the surface forcing needed for applying surface boundary conditions. The following routines are described in this chapter:

- `usrdef_1dsur_spec`: specifies the setup of boundary forcing for water column applications (1-D mode).
- `usrdef_1dsur_data`: defines the input of surface forcing data for water column applications (1-D mode).
- `usrdef_surface_absgrd`: defines a surface data grid using absolute coordinates.
- `usrdef_surface_relgrd`: defines a surface data grid using relative coordinates.
- `usrdef_surface_data`: defines the input of surface data.

The first two routines are defined in *Usrdef_Model.f90* and the next three in *Usrdef_Surface_Data.f90* (see Table 18.1).

22.1 Water column surface forcing

The routines below are only used for 1-D applications (`iopt_grid_nodim=1`).

22.1.1 Surface forcing specifiers for the 1-D mode

Contrary to the 2-D and 3-D case, the surface elevation and the horizontal pressure gradient are in the case of a water column application (`iopt_grid_nodim=1`)

not calculated by the model itself but are considered as an external forcing at the surface instead. The subroutine `usrdef_1dsur_spec` describes the arrays for setting up this water column surface forcing.

The routine is called if

```

iopt_sur_1D=1
modfiles(io_luvsur,1,1)%status ='N'

FORTRAN template
SUBROUTINE usrdef_1dsur_spec

USE iopars
USE obconds
USE tide
USE time_routines, ONLY: log_timer_in, log_timer_out

IMPLICIT NONE

procname(pglev+1) = 'usrdef_1dsur_spec'
CALL log_timer_in()
...
CALL log_timer_out()

RETURN

END SUBROUTINE usrdef_1dsur_spec

```

The following parameters and arrays can be defined here:

```

REAL, DIMENSION(nconobc) :: gxslope_amp, gxslope_ph, gyslope_amp,
                           gyslope_ph, zeta_amp, zeta_ph

```

where `nconobc` is the number of tidal constituents.

As explained in Section 5.2.1, the surface and/or surface elevation can be written as the sum of a non-harmonic and an harmonic part, given by (5.70). The first one is defined in routine `usrdef_1dsur_data` described below. The arrays above determine the harmonic part

`gxslope_amp` Amplitudes of the X-component of the pressure gradient divided by ρ_0 [m^2/s].
`gxslope_ph` Phases of the X-component of the pressure gradient [rad].
`gyslope_amp` Amplitudes of the Y-component of the pressure gradient divided by ρ_0 [m^2/s].

gyslope_phg Phases of the Y-component of the pressure gradient [rad]
 zeta_amp Amplitudes of the surface elevation [m].
 zeta_phg Phases of the surface elevation [rad].

Remarks

- The pressure gradient is normalised by the reference density ρ_0 .
- The phases must be given in radians and not in degrees. To make the conversion from degrees to radians, one needs to multiply the phases with the system parameter degtorad which is equal to $\pi/180$.

22.1.2 Surface forcing data for the 1-D mode

The data for the water column forcing are defined in the routine `usrdef_1dsur_data`.

The routine is called if

```
iopt_sur_1D=1
modfiles(io_iuvsur,2,1)%status='N'
```

FORTRAN template

```
SUBROUTINE usrdef_1dsur_data(ciodatetime,data1d,novars)
CHARACTER (LEN=lentime), INTENT(INOUT) :: ciodatetime
INTEGER, INTENT(IN) :: novars
REAL, INTENT(INOUT), DIMENSION(novars) :: data1d
```

```
USE iopars
USE obconds
USE syspars
USE tide
USE time_routines, ONLY: log_timer_in, log_timer_out
```

```
IMPLICIT NONE
```

```
procname(pglev+1) = 'usrdef_1dsur_data'
CALL log_timer_in()
...
CALL log_timer_out()
```

```
RETURN
```

```
END SUBROUTINE usrdef_1dsur_data
```

where the arguments of type **INTENT(IN)** have the following meaning

novars The number of data variables depending on the value of the switch **iopt_sbc_1D**

- 1: three data values (X- and Y-component of the pressure gradient and elevation)
- 2: one data value (elevation)
- 3: two data values (X- and Y-component of the pressure gradient).

The arguments of type **INTENT(INOUT)** need to be defined here. They have the following meaning:

ciodate Date/time in string format¹.

data1d Forcing data. The type of contents depends on the value of the switch **iopt_sbc_1D**:

- 1: Surface slopes in X- and Y-direction (normalised by the reference density ρ_0 [m^2/s]). Surface elevation [m].
- 2: Surface elevation [m].
- 3: Surface slopes in X- and Y-direction (normalised by the reference density ρ_0 [m^2/s]).

22.2 2-D surface forcing

22.2.1 Surface grid in absolute coordinates

The subroutine **usrdef_surface_absgrd** defines a surface data grid in “absolute” (geographical) coordinates. The X- and Y-coordinates are Cartesian or spherical depending on the value of the switch **iopt_grid_sph**. The routine is called when

```
iopt_meteo = 1                      ! meteorological grid
iopt_temp_sbc = 2 or 3                ! SST grid
iopt_waves = 1                        ! wave grid
iopt_bio_par = 2                      ! PAR grid
```

and the **status** attribute of the corresponding forcing file (see below) is set to 'N'.

¹If the parameter **time_zone** is defined with a non-zero value, the time of the input data must be given in local time.

The grid is non-uniform rectangular or curvilinear, i.e. `surfacegrids(idgrd,ifil)%nhtype=2` or `3` where `idgrd` is the grid key id (see below).

FORTRAN template

```
SUBROUTINE usrdef_surface_absgrid(iddesc,ifil,n1dat,n2dat,xcoord,ycoord)
  INTEGER, INTENT(IN) :: iddesc, ifil, n1dat, n2dat
  REAL, INTENT(INOUT), DIMENSION(n1dat,n2dat) :: xcoord, ycoord

  USE iopars
  USE time_routines, ONLY: log_timer_in, log_timer_out

  IMPLICIT NONE

  procname(pglev+1) = 'usrdef_surface_absgrid'
  CALL log_timer_in()
  ...
  CALL log_timer_out()

  RETURN

END SUBROUTINE usrdef_surface_absgrid
```

The `INTENT(IN)` arguments have the following meaning:

`iddesc` The file descriptor of the corresponding data file. The key id in parentheses below is the associated grid key id (`idgrd`):

- `io_metgrd` surface meteo grid (`igrd_meteo`)
- `io_sstgrd` surface grid for sea surface temperature (`igrd_sst`)
- `io_wavgrd` surface wave ghrd (`igrd_waves`)
- `io_pargrd` PAR data grid (biological module) `igrd_par`).

`ifil` File index. In the current version its value is 1.

`n1dat` X-dimension of the data grid equal to `surfacegrids(idgrd,ifil)%n1dat`.

`n2dat` Y-dimension of the data grid equal to `surfacegrids(idgrd,ifil)%n2dat`.

The following arrays of `INTENT(OUT)` need to be defined here:

`xcoord` X-coordinates of the surface grid [m or degrees longitude, positive East].

`ycoord` Y-coordinates of the surface grid [m or degrees latitude, positive North].

Remarks

- The grid coordinates of the surface grid represent the locations where the input data are defined. The type of the grid can be non-uniform rectangular or curvilinear. In case of a uniform rectangular grid, the coordinates are defined with the parameters provided in `usrdef_mod_params` (see Section 19.5.2) and do not need to be defined.
- The routines `usrdef_surface_absgrd` and `usrdef_surfac_relgrd` cannot be called both within the same simulation. This is checked by the program.
- The coordinate units of the grid must be the same in the model and surface and is determined by the value of `iopt_grid_sph`.

22.2.2 Surface grid in relative coordinates

The subroutine `usrdef_surface_relgrd` defines a surface data grid in “relative” coordinates.

The routine is called when

```
iopt_meteo = 1                      ! meteorological grid
iopt_temp_sbc = 2 or 3                ! SST grid
iopt_waves = 1                        ! wave grid
iopt_bio_par = 2                      ! PAR grid
```

and the `status` attribute of the corresponding forcing file (see below) is set to 'N'.

FORTRAN template

```
SUBROUTINE usrdef_surface_relgrd(iddesc,ifil,surfgridlb,nx,ny)
INTEGER, INTENT(IN) :: iddesc, ifil, nx, ny
TYPE (HRelativeCoords), INTENT(INOUT), &
& DIMENSION(nx,ny,nonodes) :: surfgridlb
```

```
USE datatypes
USE iopars
USE time_routines, ONLY: log_timer_in, log_timer_out
```

```
IMPLICIT NONE
```

```
procname(pglev+1) = 'usrdef_surface_relgrd'
CALL log_timer_in()
...
```

```

CALL log_timer_out()

RETURN

END SUBROUTINE usrdef_surface_relgrd

```

The relative coordinates are stored in a DERIVED TYPE array of type HRelativeCoords

```

TYPE :: HRelativeCoords
  INTEGER :: icoord, jcoord
  REAL :: xcoord, ycoord
END TYPE HRelativeCoords

```

where `icoord`, `jcoord` are the indices, at the corner node, in the X- and Y-direction of the grid cell containing the data point and `xcoord`, `ycoord` the normalised Cartesian coordinates (between 0 and 1) with respect to axes along the cell faces and origin at the lower left corner. For further details see Chapter 15.

The `INTENT(IN)` arguments have the following meaning

`iddesc` The file descriptor of the corresponding data file. The key id in parentheses below is the associated grid key id (`idgrd`).

```

io_metgrd  surface meteo grid (igrd_meteo)
io_sstgrd  surface grid for SST (igrd_sst)
io_wavgrd  grid for surface waves (igrd_waves)
io_pargrd  PAR data grid for the biological module (igrd_par).

```

`ifil` File index. In the current version its value is 1.

`nx` Currently equal to the global X-dimension `nc` of the model grid.

`ny` Currently equal to the global Y-dimension `nr` of the model grid.

The relative coordinates, defined in this routine, must be stored in the array

`surfgridglb` Relative coordinates of the model C-node grid with respect to a data grid represented by `idgrd`.

Remarks

- The routines `usrdef_surface_absgrd` and `usrdef_surfac_relgrd` cannot be called both within the same simulation. This checked by the program.

- Preference should be given to absolute instead of relative coordinates, at least when the `usrdef` method is used. In the former case the relative coordinates are calculated inside the program. However, in the case of non-rectangular data grids, these calculations can consume a large amount of time. In that case, the relative coordinates can be obtained as follows
 1. Make a first setup run with `cold_start=.TRUE.` using absolute coordinates
 2. Set `modfiles(iddesc,1,2)%status='W'` where `iddesc` equals `io_metrel`, `io_sstrel`, `io_wavrel` or `io_parrel`.
 3. A standard forcing file with the relative coordinates is obtained which can be used for the actual run by setting `modfiles(iddesc,1,1)%status` to '`R`'.
- For more information see Section 30.4.2.

22.2.3 Surface forcing data

The surface forcing data are defined in `usrdef_surface_data`. The routine is called if

- `modfiles(iddesc,1,1)%status='N'` where `iddesc` is the file descriptor (see below).
- The switch `iopt_meteo` is set to 1 in the case of a meteorological grid.
- The switch `iopt_temp_sbc` equals 2 or 3 in the case of a SST (sea surface temperature) grid.
- The switch `iopt_waves` is set to 1 in the case of a surface wave grid.
- The switch `iopt_bio_par` is set to 2 in the case of a PAR grid.

FORTRAN template

```
SUBROUTINE usrdef_surface_data(iddesc,ifil,ciodatetim,&
                               & surdata,n1dat,n2dat,novars)
CHARACTER (LEN=lentime), INTENT(INOUT) :: ciodatetim
INTEGER, INTENT(IN) :: iddesc, ifil, novars, n1dat, n2dat
REAL, INTENT(INOUT), DIMENSION(n1dat,n2dat,novars) :: surdata

USE syspars
```

```
USE time_routines, ONLY: log_timer_in, log_timer_out
```

```
IMPLICIT NONE
```

```
procname(pglev+1) = 'usrdef_surface_data'  
CALL log_timer_in()  
...  
CALL log_timer_out()
```

```
RETURN
```

```
END SUBROUTINE usrdef_surface_data
```

The **INTENT(IN)** arguments have the following meaning

iddesc The file descriptor of the corresponding data file

- io_metsur** surface meteo data
- io_sstsur** surface SST data
- io_wavsur** surface wave data.
- io_parsur** PAR data.

ifil The file index. In the current version its value is 1.

n1dat X-dimension of the data grid equal to `surfacegrids(idgrd,ifil)%n1dat`
where `idgrd` is the associated grid key id `igrd_*`.

n2dat Y-dimension of the data grid equal to `surfacegrids(idgrd,ifil)%n2dat`.

novars Number of data variables. In case of meteorological data its value
ranges from 2 to 7, in case of SST or PAR data its value is 1, in case
of wave data its value equals 3 to 12.

The data, to be defined, are

ciodate Date/time in string format¹.

surdata Surface forcing data as defined on the surface data grid. The
last dimension represents the type of variable.

22.2.3.1 Meteorological forcing data

Meteorological surface data are defined when

```
iopt_meteo=1  
modfiles(io_metsur,1,1)%status = 'N'
```

The following meteorological data can be stored in `surdata`. The order of storage must be as given in the listing below. Only variables which are used (see below) can be obtained.

```

REAL, DIMENSION(n1dat,n2dat) :: uwindatc          ! used when
                                                ! iopt_meteo_data=1,
REAL, DIMENSION(n1dat,n2dat) :: vwindatc          ! iopt_meteo_stres=1
                                                ! iopt_meteo_data=1,
REAL, DIMENSION(n1dat,n2dat) :: usstresatc        ! iopt_meteo_stres=1
                                                ! iopt_meteo_data=2,
REAL, DIMENSION(n1dat,n2dat) :: vsstresatc        ! iopt_meteo_stres=1
                                                ! iopt_meteo_data=2,
REAL, DIMENSION(n1dat,n2dat) :: atmpres           ! iopt_meteo_pres=1,nhtype>0
REAL, DIMENSION(n1dat,n2dat) :: airtemp            ! iopt_meteo_data=1,
                                                ! iopt_meteo_heat=1
REAL, DIMENSION(n1dat,n2dat) :: relhum              ! iopt_meteo_data=1,
                                                ! iopt_meteo_heat=1
REAL, DIMENSION(n1dat,n2dat) :: cloud_cover        ! iopt_meteo_data=1,
                                                ! iopt_meteo_heat=1
REAL, DIMENSION(n1dat,n2dat) :: qnsol               ! iopt_meteo_data=2,
                                                ! iopt_meteo_heat=1
REAL, DIMENSION(n1dat,n2dat) :: qrad                ! iopt_meteo_data=2,
                                                ! iopt_meteo_heat=1
REAL, DIMENSION(n1dat,n2dat) :: evapminprec        ! iopt_meteo_precip=1
REAL, DIMENSION(n1dat,n2dat) :: precipitation        ! iopt_meteo_precip=2

```

where `n1dat`,`n2dat`,`nhtype` are the dimension and grid type attributes of the surface grid (see Section 19.5.3).

<code>uwindatc</code>	X-component of the surface wind [m/s].
<code>vwindatc</code>	Y-component of the surface wind [m/s].
<code>usstresatc</code>	X-component of the surface stress [m ² /s ²].
<code>vsstresatc</code>	Y-component of the surface stress [m ² /s ²].
<code>atmpres</code>	Atmospheric pressure [Pa].
<code>airtemp</code>	Air temperature [°C].

²Note that the dimensions of the forcing variables, defined here, are the non-interpolated ones on the data grid. The variables used inside the code have the same name but have dimensions (`ncnloc`,`nrloc`) obtained after interpolation and distribution (in the parallel case) on the local grid. To avoid confusion, different names should be used by the user inside the routine `usrdef_surface_data`.

Table 22.1: Key ids for surface metereorological data

Key id	Purpose	unit
imet_uwindatc	X-component of surface wind	m/s
imet_vwindatc	Y-component of surface wind	m/s
imet_usstresatc	X-component of surface stress (normalised by density)	m ² /s ²
imet_vsstresatc	Y-component of surface stress (normalised by density)	m ² /s ²
imet_atmpres	Atmospheric pressure	Pa
imet_airtemp	Air temperature	°C
imet_relhum	Relative humidity (between 0 and 1)	—
imet_cloud_cover	Cloud cover (between 0 and 1)	—
imet_qnonsol	Non-solar upward heat flux	W/m ²
imet_qrad	Surface solar downward radiance	W/m ²
imet_evapminprec	Evaporation minus precipitation rate	kg/m ² /s
imet_precipitation	Precipitation rate	kg/m ² /s

relhum Relative humidity (between 0 and 1).
cloud_cover Cloud cover (between 0 and 1).
qnonsol Non-solar upward surface heat flux [W/m²].
qrad Surface solar irradiance [W/m²].
evapminprec Evaporation minus precipitation rate [kg/m²/s].
precipitation Precipitation rate [kg/m²/s].

Remarks

Meteo data can be stored in the `surdata` array using their key id `imet_*` (e.g. `imet_uwindatc`). A listing of the key ids is given in Table 22.1.

22.2.3.2 Sea surface temperature data

Sea surface temperature data are defined when

```
iopt_temp_sbc=2,3
modfiles(io_ssts,1,1)%status = 'N'
```

Forcing data are

```
! used when
REAL, DIMENSION(n1dat,n2dat) :: sst ! iopt_temp_sbc=2,3
```

where

`sst` Sea surface temperature [°C].

22.2.3.3 Surface wave data

Surface wave data are defined when

```
iopt_waves=1
modfiles(io_wavsur,1,1)%status = 'N'
```

The following surface wave data can be stored in `surdata`. The order of storage must be as given in the listing below. Only variables which are used (see below) are stored ².

```
! used when
REAL, DIMENSION(n1dat,n2dat) :: waveheight
REAL, DIMENSION(n1dat,n2dat) :: waveperiod
REAL, DIMENSION(n1dat,n2dat) :: wavedir
REAL, DIMENSION(n1dat,n2dat) :: wavevel      ! iopt_waves_form=2
REAL, DIMENSION(n1dat,n2dat) :: waveexcurs   ! iopt_waves_form=2
REAL, DIMENSION(n1dat,n2dat) :: umstokesatc  ! iopt_waves_curr=1,
                                               ! iopt_waves_form=2
REAL, DIMENSION(n1dat,n2dat) :: vmstokesatc  ! iopt_waves_curr=1,
                                               ! iopt_waves_form=2
REAL, DIMENSION(n1dat,n2dat) :: wavepres      ! iopt_waves_pres=1
REAL, DIMENSION(n1dat,n2dat) :: umswdissipatc ! iopt_waves_dissip=1
REAL, DIMENSION(n1dat,n2dat) :: vmswdissipatc ! iopt_waves_dissip=1
REAL, DIMENSION(n1dat,n2dat) :: umbwdissipatc ! iopt_waves_dissip=1
REAL, DIMENSION(n1dat,n2dat) :: vmbwdissipatc ! iopt_waves_dissip=1
```

where `n1dat,n2dat` are the dimension and grid type attributes of the surface grid (see Section 19.5.3)².

<code>waveheight</code>	Significant wave height [m].
<code>waveperiod</code>	Peak wave period [s].
<code>wavedir</code>	Wave direction [rad].
<code>wavevel</code>	Near bed wave velocity [m/s].
<code>waveexcurs</code>	Near bed wave excursion amplitude [m].
<code>umstokesatc</code>	X-component of the depth-mean Stokes drift [m/s].
<code>vmstokesatc</code>	Y-component of the depth-mean Stokes drift [m/s].
<code>wavepres</code>	Wave induced pressure (normalised by density) [m^2/s^2].
<code>umswdissipatc</code>	X-component of the 2-D surface wave dissipation force [m^2/s^2].
<code>vmswdissipatc</code>	Y-component of the 2-D surface wave dissipation force [m^2/s^2].

Table 22.2: Key ids for surface wave data

Key id	Purpose	unit
iwav_waveheight	Significant wave height	m
iwav_waveperiod	Peak wave period	s
iwav_wavedir	Mean wave direction	rad
iwav_wavevel	Near-bottom wave orbital velocity	m/s
iwav_waveexcurs	Near-bottom wave excursion amplitude	m
iwav_umstokesatc	X-component of the depth-mean Stokes drift	m/s
iwav_vbstokesatc	Y-component of the depth-mean Stokes drift	m/s
iwav_wavepres	Wave induced pressure (normalised by density)	m^2/s^2
iwav_umswdissipatc	X-component of the surface wave dissipation force	m^2/s^2
iwav_vmswdissipatc	Y-component of the surface wave dissipation force	m^2/s^2
iwav_umbwdissipatc	X-component of the surface wave dissipation force	m^2/s^2
iwav_vmbwdissipatc	Y-component of the surface wave dissipation force	m^2/s^2

umbwdissipatc X-component of the 2-D bed wave dissipation force [m^2/s^2].

vmbwdissipatc Y-component of the 2-D bed wave dissipation force [m^2/s^2].

Remarks

Surface wave data can be stored in the `surdata` array using their key id `iwav_*` (e.g. `iwav_umstokesatc`). A listing of the key ids is given in Table 22.2.

22.2.3.4 PAR data

PAR data are defined when

```
iopt_bio_par = 2
modfiles(io_parsur,1,1)%status = 'N'
```

Forcing data are

```
! used when
REAL, DIMENSION(n1dat,n2dat) :: peakpar ! iopt_bio_par = 2
```

where

peakpar Photosynthetically available radiation [mole photons/ m^2/s]

Chapter 23

Sediment and morphology manual

This chapter describes the setup for the sediment transport, flocculation and morphology modules. The following `usrdef_` routines, located in `Usrdef_Sediment.f90`, are available for setup

- `usrdef_sed_params`: switches and model parameters for the multi-fraction sediment and flocculation modules (`iopt_sed>0`).
- `usrdef_sedics`: initial conditions for the multi-fraction sediment and flocculation modules (`iopt_sed>0`).
- `usrdef_sed_spec`: attributes of sediment particles in the multi-fraction sediment transport module for each fraction (`iopt_sed=1`).
- `usrdef_morph_params`: switches and model parameters for the morphological module (`iopt_morph=1`)
- `usrdef_morphics`: initial conditions for the morphological module (`iopt_morph=1`).

Note that these routines are only called when the appropriate switch is activated. The setup of both sediment modules is discussed in the first section below, the setup of the morphology in the second.

23.1 Sediment transport and flocculation model setup

In this section a list of the switches and parameters is presented for the sediment modules. They are either defined either in the routine `usrdef_sed_params`

or in the CIF block `sed_params` if it exists and the CIF is activated.

23.1.1 Switches and parameters for the multi-fraction sediment module

23.1.1.1 Switches for the multi-fraction model

Two types of switches are available. The first are fraction independent, the second are switches which depend on the fraction class. The latter are defined in `usrdef_sed_spec` (see Section 23.1.4). Besides the two “generic” switches `iopt_sed` (for activating the multi-fraction sediment module when set to 1) and `iopt_obic_sed` (enabling non-default open boundary conditions for sediments), the following switches are available

```
INTEGER :: iopt_sed_beta, iopt_sed_dens_grad, iopt_sed_filter, &
& iopt_sed_nodim, iopt_sed_obic_flux, iopt_sed_rough_btr, &
& iopt_sed_rough_rip, iopt_sed_rough_skin, iopt_sed_transp, &
& iopt_sed_vadv
```

<code>iopt_sed_beta</code>	The type of equation used for β_n , the ratio between the eddy viscosity and eddy diffusivity.
0: $\beta_n = 1$. Default.	
1: User defined value (<code>beta_sed_cst</code>). Default.	
2: Van Rijn (1984b) .	
<code>iopt_sed_dens_grad</code>	Disables/enables the inclusion of sediment stratification in the formulations for the buoyancy frequency and baroclinic pressure gradient.
0: Disabled. Default.	
1: Enabled.	
<code>iopt_sed_filter</code>	Disables/enables the application of the Bartnicki filter to prevent the occurrence of negative concentrations.
0: Disabled. Default.	
1: Enabled.	
<code>iopt_sed_nodim</code>	The grid dimension used for sediment transport.
2: Depth (2-D) averaged transport ¹	

¹Note that `iopt_sed_nodim` is (re)set to 2 if `iopt_grid_nodim` = 2.

	3: 3-D sediment transport. Default.
iopt_sed.obc_flux	Disables/enables the use of a zero gradient condition for bed and total load transport at open sea boundaries. 0: Disabled. Default. 1: Enabled.
iopt_sed.rough.btr	Selects type of model for the bedload roughness length. 0: Disabled. Default. 1: Grant & Madsen (1982) . 2: Wiberg & Rubin (1989) . 3: Nielsen (1992) .
iopt_sed.rough.rip	Selects type of wave ripple model. 0: Disabled. Default. 1: Wiberg & Harris (1994) . 2: Li <i>et al.</i> (1996) . 3: Soulsby & Whitehouse (2005) . 4: Goldstein <i>et al.</i> (2013) .
iopt_sed.rough.skin	Selects formulation for the skin bottom roughness length. 0: Same value as the form roughness. 1: User-defined spatially uniform value. 2: As function of the median grain size, given by (7.16). Default.
iopt_sed.transp	Method for solving transport equations in the sediment model. 1: Each sediment fraction is updated separately. Default. 2: All sediment fractions are simultaneously updated.
iopt_sed.vadv	Selects the type of vertical advection scheme for settling in case of a 1-D simulation. In the 3-D case, the advection scheme is selected by the general switch iopt_adv.scal. 0: Vertical settling disabled. 1: Upwind scheme. 2: Central scheme. 3: TVD scheme. Default.

23.1.1.2 Parameters for the multi-fraction model

The following parameters can be defined

```
INTEGER :: icsed, maxitbartnicki, nb, nf, nrquad_sed
REAL ::  alpha_VR, a_leussen, beta_sed_cst, beta_sed_max, &
& beta_sed_min, b_leussen, cgel, cmax, cmax_RichZaki, &
& coef_bed_grad, floc_VR_max, floc_VR_min, n_RichZaki, &
& parth_coef, parth_exp, wu_exp, zrough_grain, &
& zrough_min, zrough_rip, zrough_sed_cst
```

integer parameters

icsed	Time step counter (i.e. number of base time steps) for update of bed/total load transport and the morphology. Set to ic3d if suspended sediment transport is enabled for at least one fraction. Default is ic3d.
maxitbartnicki	Maximum number of iterations used by the bartnicki filter. Default is 100.
nb	Number of sediment bed layers. Set to 1 if morphology is disabled. Default is 1.
nf	Number of sediment classes. Default is 1.
nrquad_sed	Number of vertical locations used by the Gauss-Legendre numerical integration scheme for depth averaging. Default is 7.

real parameters

alpha_VR	Exponent α_{fl} in the flocculation equation (7.49) by Van Rijn (2007b) . Default is 2.19.
a_leussen	Coefficient a in the (7.47) flocculation equation by Van Leussen (1994) [s]. Default is 0.02.
beta_sed_cst	Constant value of the eddy diffusivity to viscosity ratio β_n if <code>iopt_sed_beta=1</code> . Default is 1.0.
beta_sed_max	Maximum value for the ratio of sediment diffusivity to eddy viscosity if <code>iopt_sed_beta=2</code> . Default is 1.5.
beta_sed_min	Minimum value for the ratio of sediment diffusivity to eddy viscosity if <code>iopt_sed_beta=2</code> . Default is 1.0.
b_leussen	Coefficient b in the (7.47) flocculation equation by Van Leussen (1994) [s^2]. Default is 0.0024.

cgel	Volumetric gelling concentration used for hindered settling of mud and flocculation [m^3/m^3]. Default is 0.075.
cmax	Maximum volumetric maximum concentration used for calculating the reference concentration in the Smith & McLean (1977) formula (7.95) and the Rouse profile (7.98) [m^3/m^3]. Default is 0.65.
cmax_Richzaki	Maximum value for the total concentration as used in the Richardson-Zaki formulation [m^3/m^3]. Default is 0.7.
coef_bed_grad	Coefficient β_s used in the bed slope formula (7.71) of Koch & Flokstra (1981) . Default is 1.3.
floc_VR_max	Maximum value for the flocculation factor f_{fl} in equation (7.49) by Van Rijn (2007b) . Default is 10.0.
floc_VR_min	Minimum value for the flocculation factor f_{fl} in equation (7.49) by Van Rijn (2007b) . Default is 1.0.
n_RichZaki	Exponent n_h in equation (7.45) for hindered settling by Richardson & Zaki (1954) . Default is 4.0.
parth_coef	Coefficient M_p in the formulation (7.89) for erosion of mud by Partheniades (1965) [$\text{kg}/\text{m}^2/\text{s}$]. Default is 2.0×10^{-4} .
parth_exp	Exponent n_p in the formulation (7.89) for erosion of mud by Partheniades (1965) . Default is 1.0.
wu_exp	Exponent m_h used to calculate the hiding factor (7.37) in the Wu et al. (2000) formulation. Default is 0.6.
zrough_grain	Proportionality factor a_{sg} used in the definition of the grain size roughness in (7.16). Default is 0.0833.
zrough_min	Minimum value for the skin roughness z^s [m]. Default is 10^{-5} .
zrough_rip	Parameter a_{sr} used to determine the wave ripple roughness in (7.27). Default is 0.923.
zrough_sed_cst	Uniform roughness length used for obtaining the (skin) bed stress if <code>iopt_sed_rough=1</code> [m]. No default.

23.1.2 Switches and parameters for the flocculation model

23.1.2.1 Switches for the flocculation model

Besides the two “generic” switches `iopt_sed` (for activating the flocculation module when set to 2) and `iopt_obic_sed` (enabling non-default open boundary conditions for sediments), the following switches are available

```
INTEGER :: iopt_floc_bbc_ref, iopt_floc_bstres_cr, iopt_floc_slope, &
& iopt_floc_ws, iopt_floc_ws_hindset, iopt_floc_ws_lim, &
& iopt_sed_beta, iopt_sed_dens_grad, &
& iopt_sed_rough_btr, iopt_sed_rough_rip, &
& iopt_sed_rough_skin, iopt_sed_transp, iopt_sed_vadv
```

`iopt_floc_bbc_ref` Selects the formulation for the reference concentration $c_{a,n}$.

1: [Smith & McLean \(1977\)](#). Default.

2: [Van Rijn \(1984a\)](#).

`iopt_floc_bstres_cr` Selects the formulation for the critical bottom shear stress.

1: User-defined value for each fraction. Default.

2: [Brownlie \(1981\)](#) as given by (7.31).

3: [Soulsby & Whitehouse \(1997\)](#) as given by (7.32).

4: Constant value for the critical Shield parameter from [Wu et al. \(2000\)](#).

`iopt_floc_slope` Selects the type of slope factor for the critical shear stress.

0: Disabled. Default.

1: Enabled using (7.39).

`iopt_floc_ws` Selects the formulation for the settling velocity.

0: Disabled.

1: Stokes formula. Default.

2: [Camenen \(2007\)](#).

`iopt_floc_ws_hindset` Formulation for hindered settling.

0: Hindered settling disabled. Default.

1: [Richardson & Zaki \(1954\)](#) equation (7.45).

`iopt_floc_ws_lim` Disables/enables the limitation of the settling velocity for shallow waters.

0: Disabled. Default.

1: Enabled.

<code>iopt_sed_beta</code>	The same as above.
<code>iopt_sed_dens_grad</code>	The same as above.
<code>iopt_sed_rough_btr</code>	The same as above.
<code>iopt_sed_rough_rip</code>	The same as above.
<code>iopt_sed_rough_skin</code>	The same as above.
<code>iopt_sed_transp</code>	The same as above.
<code>iopt_sed_vadv</code>	The same as above.

23.1.2.2 Parameters for the flocculation model

A number of parameters used in the multi-fraction module are also used in the flocculation module

```
REAL :: beta_sed_cst, beta_sed_max, beta_sed_min, cmax, cmax_RichZaki, &
& n_RichZaki, parth_coef, parth_exp, wu_exp, zrough_grain, &
& zrough_min, zrough_rip, zrough_sed_cst
```

Their meaning and defaults are described above.

The following setup parameters are additionally available for the flocculation module

```
REAL :: agg_alpha, brk_es, brk_ffy, brk_frac, brk_p, brk_q, &
& bstres_cr_cstP, dpP, floc_ncmax, floc_ncmin, nfrdim, rhosP, &
& wsfloclimfac
```

<code>agg_alpha</code>	Collision efficiency factor α . Default is 0.1.
<code>brk_es</code>	Efficiency factor for breaking E_b [$s^{1/2}/m$]. Default is 10^{-4} .
<code>brk_ffy</code>	Yield strength of flocs F_y [Pa]. Default is 10^{-10} .
<code>brk_frac</code>	Fraction f of flocculi generated by floc breakage. Default is 1.0.
<code>brk_p</code>	Empirical exponential factor p for breakage kinetics used in (7.125). Default is 1.0.
<code>brk_q</code>	Empirical exponential factor q for breakage kinetics used in (7.125). Default is 0.7.
<code>bstres_cr_cstP</code>	Uniform critical shear stress for flocculi [m^2/m^2]. No default but must be defined when <code>iop_floc_bstres_cr=1</code> .
<code>dpP</code>	Particle diameter of flocculi [m]. No default but must be defined.
<code>floc_ncmax</code>	Maximum value of N_c . Default is 100.0

floc_ncmin	Minimum value of N_c . Default is 2.0.
nfrdim	Fractal dimension of macroflocs n_F . Default is 2.0.
rhosP	Particle density of microflocs [kg/m ³]. Default is 2650.
wsflocclimfac	Limiting CFL number used for the settling velocity if iopt_floc_ws_lim = 1. Default is 10.

23.1.3 Forcing file parameters

Forcing attributes for the multi-fraction sediment and flocculation modules are defined in routine `usrdef_mod_params` or by the CIF. The following forcing “files” are used for sediments

- `modfiles(io_inicon,ics_sed,:)`: initial conditions for the sediments
- `modfiles(io_sedspc,1,:)`: attributes of sediment particle fractions (multi-fraction module only)
- `modfiles(io_seddbc,:,:)`: specifiers and open boundary data for sediments
- `modfiles(io_sednst,1:nonestsets,2)`: sediment open boundary data for nested sub-grids (one file per sub-grid).

23.1.4 Switches and parameters per fraction

A number of sediment particle attributes are defined for the multi-fraction sediment module in the routine `usrdef_sed_spec` if

```
modfiles(io_sedspc,1,1)%status = 'N'
```

or in the CIF block `sed_spec` if it exists and the CIF is activated, or from a standard forcing file if

```
modfiles(io_sedspc,1,1)%status = 'R'
```

FORTRAN template

```
SUBROUTINE usrdef_sed_spec
```

```
USE iopars
```

```
USE sedarrays
```

```
USE time_routines, ONLY: log_timer_in, log_timer_out
```

```
IMPLICIT NONE
```

```

procname(pglev+1) = 'usrdef_sed_spec'
CALL log_timer_in()
...
CALL log_timer_out()

RETURN

END SUBROUTINE usrdef_sed_spec

```

The setup switches and parameters are stored into the derived type vector array `sedpart` of size `nf`. They are defined as follows

```

TYPE :: SedPartAtts
  INTEGER :: iopt_bbc_eq, iopt_bbc_ref, iopt_bedeq, iopt_bstres_cr, &
             & iopt_hidexp, iopt_slope, iopt_suseq, iopt_toteq, iopt_type, &
             & iopt_ws, iopt_ws_floc, iopt_ws_hindset, iopt_ws_lim
  REAL :: bstres_cr_cst, dp, rhos, ws_cst, wslimfac
END type SedpartAtts
TYPE (SedPartAtts), ALLOCATABLE, DIMENSION(nf) :: sedpart

```

where

<code>iopt_bbc_eq</code>	Selects the formulation for the equilibrium sediment concentration.
1:	Rouse profile. Default.
2:	Using q_t/U determined with the equation of Engelund & Hansen (1967) and $\theta_{*,n} = \theta_n$.
3:	Using q_t/U determined with the equation of Engelund & Hansen (1967) and using the modified version of Chollet & Cunge (1979) for $\theta_{*,n}$.
4:	Using q_t/U determined with the equation of Ackers & White (1973) .
5:	Using q_t/U determined using (7.82) from Madsen & Grant (1976) .
<code>iopt_bbc_ref</code>	Formulation for the reference concentration $c_{a,n}$.
1:	Smith & McLean (1977) . Default.
2:	Van Rijn (1984a) .
<code>iopt_bedeq</code>	Type of bedload equation.

	0: Disabled. Default. 1: Meyer-Peter & Müller (1948) . 2: Engelund & Fredsøe (1976) . 3: Van Rijn (1984b) . 4: Wu <i>et al.</i> (2000) . 5: Soulsby (1997) . This equation includes explicit wave effects. 6: Van Rijn (1993) .
<code>iopt_bstres_cr</code>	Formulation for the critical bottom shear stress. 1: User-defined value for each fraction. Default. 2: Brownlie (1981) as given by (7.31). 3: Soulsby & Whitehouse (1997) as given by (7.32). 4: Constant value for the critical Shield parameter from Wu <i>et al.</i> (2000) .
<code>iopt_hidexp</code>	Type of model for hiding and exposure. 0: Disabled. Default. 1: Wu <i>et al.</i> (2000) as given by (7.37). 2: Ashida & Michiue (1972) as given by (7.38).
<code>iopt_slope</code>	Use of a slope factor for the critical shear stress and bed load transport. 0: Disabled. Default. 1: Enabled using (7.39).
<code>iopt_suseq</code>	Disables/enables suspended transport. 0: Disabled. Default. 1: Enabled.
<code>iopt_toteq</code>	Method for total load transport. 0: Disabled. Default. 1: Engelund & Hansen (1967) with $\theta_{*,n} = \theta_n$. 2: Engelund & Hansen (1967) using the modified version of Chollet & Cunge (1979) for $\theta_{*,n}$. 3: Ackers & White (1973) .

	4: Madsen & Grant (1976) .
	5: Wu <i>et al.</i> (2000) . Total load is calculated as the sum of suspended and bed load.
<code>iopt_type</code>	Type of sediment.
	1: Sand. Default.
	2: Mud.
<code>iopt_ws</code>	Type of method for the settling velocity.
	1: User-defined value for each fraction. Default.
	2: Camenen (2007) formulation (7.40). The coefficients A , B , m depend on the type of sediment selected with the <code>iopt_type</code> attribute.
	3: Stokes formula (7.41).
	4: Soulsby (1997) formula (7.42).
	5: Zhang & Xie (1993) equation (7.43).
<code>iopt_ws_floc</code>	Type of flocculation factor for the settling velocity
	0: Flocculation effect disabled. Default.
	1: Van Leussen (1994) equation (7.47).
	2: Van Rijn (2007b) equation (7.49).
<code>iopt_ws_hindset</code>	Formulation for hindered settling.
	0: Hindered settling disabled. Default.
	1: Richardson & Zaki (1954) equation (7.45).
	2: Winterwerp & van Kesteren (2004) formula (7.46).
<code>iopt_ws_lim</code>	Disables/enables the limitation of the settling velocity for shallow waters.
	0: Disabled. Default.
	1: Enabled.
<code>bstres_cr_cst</code>	Constant critical bed shear stress used when <code>iopt_bstres_cr=1</code> [m^2/s^2]. Default is 0.0.
<code>dp</code>	Particle diameter [m]. No default.
<code>rhos</code>	Particle density [kg/m^3]. Default is 2650.0.
<code>ws_cst</code>	Constant fall velocity used when <code>iopt_ws=1</code> .
<code>wslimfac</code>	Limiting CFL number used for the settling velocity if <code>iopt_ws_lim = 1</code> . Default is 10.

23.1.5 Initial conditions for the sediment module

The initial conditions for the sediment modules are defined in `usrdef_sedics` which is called when `iopt_sed=1,2` and

```

modfiles(io_inicon,ics_sed,1)%status='N'

FORTRAN template

SUBROUTINE usrdef_sedics

USE iopars
USE sedpars
USE sedarrays
USE time_routines, ONLY: log_timer_in, log_timer_out

procname(pglev+1) = 'usrdef_sedics'
CALL log_timer_in()

...
CALL log_timer_out()

RETURN

END SUBROUTINE usrdef_sedics

```

The procedures for reading and distributing (in the parallel case) data are the same as for the hydrodynamical initial conditions. For details see Section 20.3.

23.1.5.1 multi-fraction model

The following arrays can be initialised

```

! used when
REAL, DIMENSION(ncloc,nrloc,nz,nf) :: cvol      ! for fractions in
                                                ! suspended mode
REAL, DIMENSION(ncloc,nrloc,nb,nf) :: bed_fraction ! nf>1

```

The array dimensions `ncloc` and `nrloc` indicate that these arrays are defined on the local grid in case the model is run in parallel mode. This in contrast with the forcing file method where they are defined on the global grid. The

reason is that these model arrays are only defined on the local grid in the parallel case and not on the global grid. Note that the multi-fraction model for suspended transport is only enabled when `iopt_sed` is set to 1.

- `cvol` The volumetric sediment concentration for each sediment fraction when suspended load is enabled for at least one fraction [m^3/m^3].
- `bed_fraction` The amount of material in the sediment bed layers for each sediment fraction. Note that the sum over all fractions must be equal to 1 or zero. The latter case corresponds to an empty bed sediment layer.

Remarks

- Initial concentrations for sediment concentrations are difficult to define. It can therefore be usefull to split the simulation in two runs. In the first (“cold”) run the concentrations are set to their default values. A usefull procedure is to take zero (default) values at the start. After some time a quasi-equilibrium stage between erosion and deposition will gradually be established. At the end of the run the data are written to a final condition file (see Section 30.3.2) providing the initial data for the second run.
- By default, all bed fractions are equal and set to the uniform value $1/\text{nf}$.

23.1.5.2 Flocculation model

The flocculation model is enabled when `iopt_sed` is set to 2.

The following arrays can be initialised

```
REAL, DIMENSION(ncloc,nrloc,nz,3) :: cnump
```

`cnump` State variables in the flocculation model. The last index has the following meaning:

- 1: Number concentration of flocculi (N_p) [m^{-3}].
- 2: Number concentration of flocs (N_F) [m^{-3}].
- 3: Number concentration of flocculi bound in flocs (N_T) [m^{-3}].

Remarks

- As for the multi-fraction model the array dimensions `ncloc` and `nrloc` indicate that these arrays are defined on the local grid in case the model is run in parallel mode. The reason is that model arrays are only defined on the local grid in the parallel case and not on the global grid to save internal memory.
- It may be difficult to define initial concentrations. It can therefore be usefull to split the simulation in two runs. In the first (“cold”) run the concentrations are set to their default values. A usefull procedure is to take zero (default) values at the start. After some time a quasi-equilibrium stage will gradually be established. At the end of the run the data are written to a final condition file (see Section 30.3.3) providing the initial data for the second run.

23.1.6 Sediment open boundary conditions

Open boundary conditions for sediments are defined in the routine `usrdef_profobc_spec` which is called for sediments if

```
modfiles(io_sedobc,1,1)%status = 'N'
```

For details see Section 21.2.1. Note that `novars` argument in the routine equals `nf` in the multi-fraction and 3 in the flocculation module.

Open boundary data profiles of sediment concentrations in the multi-fraction model or the flocculation model are obtained in `usrdef_prof_obcdata` which is called for sediments if

```
modfiles(io_sedobc,ifil,1)%status='N'
```

where `ifil` is the data file index which may take values between 2 and `nfiles`. For details see Section 21.2.4.

Note that open boundary conditions can only be used in the multi-fraction model for sediment fractions in suspended mode (i.e. `sedpart(f)%iopt_suseq=1` for fraction `f`), but can be used for all three state variables in the flocculation model. If not defined by the user, a zero gradient condition is taken by default.

For more details see Section 21.2.3.

23.1.7 Sediment nesting

It is possible to export suspended sediment concentrations for nesting. In order to use sediment concentrations as open boundary data in a sub-grid model by nesting, it is necessary to set `modfiles(io_sednst,ifil,2)%status = 'W'`,

where `ifil` is the number of the nested sub-grid (between 1 and `nonestsets`). Procedures are the same as described in Section 21.3. Nesting is only performed for fractions in suspended mode in the case of the multi-fraction model and for all (3) number concentrations in the case of the flocculation model.

23.2 Morphological model setup

In this section a list of the switches and parameters is presented for the morphological module. They are defined in the routine `usrdef_morph_params` or in the CIF block `morph_params` if it exists and the CIF is activated.

23.2.1 Parameters for the morphological module

Besides the switch `iopt_morph`, defined in `usrdef_mod_params`, for activating the morphology, the following switches are available

```
INTEGER :: iopt_morph_accel, iopt_morph_active_layer, &
& iopt_morph_avalanching, iopt_morph_corr, &
& iopt_morph_fixed_layer, iopt_morph_hdiff, &
& iopt_morph_tidal_scheme, iopt_morph_time_init, &
& iopt_morph_time_int, iopt_morph_vert_fluxes
```

`iopt_morph_accel` Selects the method for morphological acceleration.

- 0: Disabled. Default.
- 1: Using a morphological acceleration factor.
- 2: Using the tidal averaging method.

`iopt_morph_active_layer` Selects the type of formulation for the active layer thickness.

- 0: Disabled.
- 1: Flat-bed conditions (Armanini, 1995). Default.
- 2: Using the average bedform height (Ribberink, 1987; Parker, 1991; Armanini, 1995; Blom, 2003).
- 3: Combination of sediment features and hydraulic conditions (Harris & Wiberg, 1997).

`iopt_morph_avalanching` Disables/enables avalanching.

- 0: Disabled. Default.

	1: Enabled.
<code>iopt_morph_corr</code>	Selects the type of action taken when more sediment is retrieved from the bed layer than available.
	0: The active or fixed bed layer depth is set to zero without preserving sediment mass. Default.
	1: The mass conservation algorithm is used.
<code>iopt_morph_fixed_layer</code>	Disables/enables a fixed (finite) sediment layer depth.
	0: Disabled. Default.
	1: Enabled.
<code>iopt_morph_hdiff</code>	Disables/enables the use of numerical diffusion in the bed elevation equation.
	0: Disabled. Default.
	1: Enabled.
<code>iopt_morph_tidal_scheme</code>	Selects the type of velocity correction if tidal averaging is enabled.
	0: No correction is applied.
	1: Continuity correction (8.10). Default.
	2: Friction correction (8.11).
	3: Combined continuity/friction correction (8.12).
<code>iopt_morph_time_int</code>	Selects the kind of time integration scheme for the bed elevation equation.
	1: First order Euler scheme. Default.
	2: Second order Adams-Bashforth scheme.
	3: Third order Adams-Bashforth scheme.
<code>iopt_morph_time_init</code>	Informs the program, in the case of a higher order time integration, whether the values of the bed elevation at previous times have been provided by the user at the initial time.
	0: Not provided. A first order time integration scheme is used at the first time step, in case <code>iopt_time_int=2,3</code> . A second order time integration scheme is used at the second time step, in case <code>iopt_time_int=3</code> . Default.

	1: Provided as initial conditions.
<code>iopt_morph_vert_fluxes</code>	Selects the kind of vertical exchange model
	0: Disabled. Default.
	1: Enabled using the exchange layer concept of Ribeirink (1987) .

23.2.2 Parameters for the morphological module

The following parameters can be defined for the morphological module

```
INTEGER :: accelstep, icaval, icsedbal, max_iterations_aval, &
           & max_iterations_corr, morph_steps, nstep_hydro, &
           & number_tidal_steps
REAL :: alpha_jameson, average_bedform_height, bed_porosity_cst, &
        & dyn_angle_dry, dyn_angle_wet, k1_harris, k2_harris, &
        & k2_jameson, k4_jameson, morph_factor, &
        & similarity_range, stat_angle_dry, stat_angle_wet
```

Integer parameters

<code>accelstep</code>	Number of morphological time steps for storing or retrieving hydrodynamical data used when <code>iopt_morph_accel</code> =2. No default.
<code>icaval</code>	Time step counter (i.e. number of base time steps) for update of avalanching in case that <code>iopt_morph_avalanching</code> =1. Must be a multiple of <code>ic3d</code> . No default.
<code>icsedbal</code>	Time step counter (i.e. number of base time steps) for checking conservation of mass during a time period provided by <code>icsedbal</code> . Must be larger than one or zero (in which no checking is made). Must be a multiple of <code>ic3d</code> . Default is zero.
<code>max_iterations_aval</code>	Maximum number of iterations allowed in the avalanching algorithm (<code>iopt_morph_avalanching</code> =1. Must be positive. Default is 500.
<code>max_iterations_corr</code>	Maximum number of iterations used in the bed elevation equation correction scheme to prevent negative concentrations when <code>iopt_morph_corr</code> =1. Default is 100.
<code>morph_factor</code>	Factor used for morphological acceleration.

<code>morph_steps</code>	Number of morphological cycles N_{mor} used in the tidal averaging scheme (<code>iopt_morph_accel</code> =2). No default.
<code>nstep_hydro</code>	The number of hydrodynamic time steps during the hydrodynamical cycle used when <code>morph_accel</code> =2. No default.
<code>number_tidal_steps</code>	The number of morphological time steps during the hydrodynamical and morphological cycles used when <code>iopt_morph_accel</code> =2. No default.

Real parameters

<code>alpha_jameson</code>	Factor α used for the Jameson <i>et al.</i> (1981) parameterisation (equation (8.5)) of the 4 th order artificial diffusion flux in the bed level equation. Default is 2.0.
<code>average_bedform_height</code>	Average bedform height z_{ba} used in the formulation for the active layer thickness when <code>iopt_morph_active_layer</code> =2 [m]. Default is 0.05.
<code>bed_porosity_cst</code>	Constant value of the bed porosity used in the current version of COHERENS. Default is 0.4.
<code>dyn_angle_dry</code>	Dynamic angle of repose S_{cr}^d after avalanching at dry cells [degrees]. Default is 25.
<code>dyn_angle_wet</code>	Dynamic angle of repose S_{cr}^d after avalanching at wet cells [degrees]. Default is 30.
<code>k1_harris</code>	Factor k_1 in the Harris & Wiberg (1997) formulation of the active layer thickness. Default is 0.7.
<code>k2_harris</code>	Factor k_2 in the Harris & Wiberg (1997) formulation of the active layer thickness. Default is 6.0.
<code>k2_jameson</code>	Factor k_2 used for the Jameson <i>et al.</i> (1981) parameterisation (equation (8.5)) of the 2 nd order artificial diffusion flux in the bed level equation. Default is 8.0.
<code>k4_jameson</code>	Factor k_4 used for the Jameson <i>et al.</i> (1981) parameterisation (equation (8.5)) of the 4 th order artificial diffusion flux in the bed level equation. Default is 0.03125.
<code>morph_factor</code>	Morphological factor used when <code>iopt_morph_accel</code> =1. Default is 1.0 (no morphological acceleration).

similarity_range	Factor S_R used in the similarity criterion (8.26). Default is 0.05.
stat_angle_dry	Dynamic angle of repose S_{cr}^s before avalanching at dry cells [degrees]. Default is 34.
stat_angle_wet	Dynamic angle of repose S_{cr}^d before avalanching at wet cells [degrees]. Default is 45.

23.2.3 monitoring info

When the `sedlog` parameter is set to `.TRUE.` in `usrdef_init_params`, monitoring information is written to the `sedlog` file.

1. If `icsedbal` is greater than zero and `iopt_morph_fixed_layer=1`, the following monitoring parameters are defined and written to the `sedlog` file:

<code>sediment_tot</code>	The total amount of sediment in the water column and the sediment bed layer [ton].
<code>sediment_vol</code>	The difference of <code>sediment_tot</code> over the time integration interval specified by <code>icsedbal</code> [ton].
<code>sediment_obc</code>	The net amount of sediment transported into or out the domain at open boundaries by bed, total load or suspended transport over the time integration interval [ton].
<code>sediment_err</code>	The mass conservation error given by <code>sediment_vol-sediment_obc</code> [ton].
<code>sediment_err_pt</code>	The relative error in percentage of the total sediment mass defined as $100 * \text{sediment_err} / \text{sediment_tot}$.
<code>sediment_err_acc</code>	The mass conservation error accumulated in time [ton].

The above parameters are also available for time series output.

2. When `iop_morph_corr=1`, the number of iterations used by the correction scheme is written to the `sedlog` file.

23.2.4 Initial conditions for the morphology

The initial conditions for the morphology module are defined in `usrdef_morphics` which is called when `iop_morph=1` and

```

    modfiles(io_inicon,ics_morph,1)%status='N'

FORTRAN template
SUBROUTINE usrdef_morphics

USE iopars
USE sedpars
USE morpharrays
USE time_routines, ONLY: log_timer_in, log_timer_out

procname(pglev+1) = 'usrdef_morphics'
CALL log_timer_in()

...
CALL log_timer_out()

RETURN

END SUBROUTINE usrdef_morphics

```

The procedures for reading and distributing (in the parallel case) data are the same as for the hydrodynamical initial conditions. For details see Section 20.3.

The following arrays can be initialised

```

! used when
REAL, DIMENSION(ncloc,nrloc,nb) :: bed_layer_thickness ! iopt_morph_fixed_lay
REAL, DIMENSION(ncloc,nrloc,nf) :: bed_update_acc

```

where

bed_layer_thickness Thickness of the sediment bed layers [m].

bed_update_acc The change in the bed level elevation accumulated over time [m].

Remarks

- By default, all arrays are initialised to zero.
- In case of a restart, all arrays above are initialised from a final condition file generated by COHERENS (depending on the restrictions given above). This forcing file contains additional arrays which cannot be defined by the user for an initial run. See Section 30.3.4 for details.

- In the case of a first start the bed layer thicknesses must be defined with non-zero values in case a fixed sediment layer depth is used.
- Note that no open boundary conditions or nesting procedures are required for the morphological module.

23.3 Sediment output

Output for sediment is generated by the standard output routines in COHERENS, i.e. time series (*Usrdef_Time_Series.f90*), time averaged (*Usrdef_Time_Averages.f90*), harmonic analysis (*Usrdef_Harmonic_Analysis.f90*) and user formatted output (*Usrdef_Output.f90*). Important to note is that output variables may have an extra attribute `isub` representing the fraction number in case the model variable has an extra last dimension of size `nf`. This attribute, when needed, has to be stored in the `tsrvars`, `avrvars`, `analvars` arrays. The output in COHERENS is described in detail within Chapter 27.

Chapter 24

Biology manual

This chapter describes the setup of the biology. The following `usrdef_` routines, located in `Usrdef_Biology.f90`, are available for setup

- `usrdef_bio_params`: switches and model parameters
- `usrdef_bioics`: initial conditions

Note that these routines are called when the biological module is activated, i.e. `iopt_biology=1`.

24.1 Switches and parameters for the biology

In this section the list of the switches and parameters available for the biology module is presented. They are defined either in the routine `usrdef_bio_params` or in the `CIF` block `bio_params` if it exists and the `CIF` is activated.

24.1.1 Switches

Firstly, it is noted that two “generic” switches, which are related to the biology, are already defined in `usrdef_mod_params`:

- `iopt_biology` for activating the biology module (Section 19.2.5)
- `iopt_obc_obc` for enabling non-default open boundary conditions for biological state variables (Section 19.2.13).

The following additional switches switches are available

```
INTEGER :: iopt_bio_benth, iopt_bio_par, iopt_bio_transp, &
& iopt_bio_vadv, iopt_bio_water_col
```

<code>iop_bio_benth</code>	Formulation for the benthic layer.
0:	Disabled. The sediment layer is not taken into account. In that case the 2-D state variables for the benthos are not used.
1:	Only the boundary fluxes are updated. The sediment state variables are not updated (only used for monitoring or debugging).
2:	The sediment state variables are updated. Default.
<code>iop_bio_par</code>	Method used to evaluate photosynthetically available radiation (PAR) at the surface.
1:	Proportional to solar radiation.
2:	As function of the (peak) PAR at noon, obtained from an external data file, and solar altitude. Default.
<code>iop_bio_transp</code>	Method for solving the advection-diffusion transport equations for the 3-D state variables in the water column.
0:	Each state variable is updated separately. Default.
1:	The state variables are simultaneously updated. This option needs less MPI communications, but consumes more internal memory.
<code>iop_bio_vadv</code>	Selects the type of vertical advection scheme for settling in case of a 1-D simulation. In the 3-D case, the advection scheme is selected by the general switch <code>iop_adv_scal</code> .
0:	Vertical settling disabled.
1:	Upwind scheme.
2:	Central scheme.
3:	TVD scheme. Default.
<code>iop_bio_water_col</code>	Disables/enables the evaluation of the water column source terms
0:	Disabled (only intended for debugging).
1:	Enabled. Default.

24.1.2 Parameters

The following parameters can be defined

```

!---biological model parameters
REAL :: alphaNSC, alphaSC, burialBSi, burialOMs, denitrifS, grazeff, &
& grazmax, grazprefNSC, grazprefOM, grazprefSC, KdsiSC, Kgraz, &
& Knh4NSC, Knh4SC, Kno3NSC, Kno3SC, Kpo4NSC, Kpo4SC, mortNSC, &
& mortSC, mortZP, nitrifS, nitrifW, parattchl, paratt0, partorad, &
& PboptNSC, PboptSC, regenBSi, regenNH4, regenNO3, regenPO4, &
& reminBSi, reminOM, reminOMs, Tfamp, uptprefNH4, uptprefNO3, &
& wsinkBSi, wsinkOM

REAL, DIMENSION(4) :: CtoChlpars

!---elemental ratios (mole of C, N, Si per mole P)
REAL :: erCinNSC, erCinOM, erCinOMs, erCinSC, erCinZ, erNinNSC, erNinOM, &
& erNinOMs, erNinSC, erNinZ, erPinNSC, erPinOM, erPinOMs, &
& erPinSC, erPinZ, erSiinSC

!---minimum thresholds
REAL, DIMENSION(nobiovars2d) :: biovarsmin2d
REAL, DIMENSION(nobiovars3d) :: biovarsmin3d
REAL :: chlmin

!---flags for open boundary data and nesting
INTEGER, DIMENSION(nobiovars3d,nonestsets) :: indexbio_nst

```

photosynthesis

- alphaNSC** Maximum light utilization coefficient α_{NSC} for NSC [$m^2s/(mole photons)$]. Default is 0.016.
- alphaSC** Maximum light utilization coefficient α_{SC} for SC [$m^2s/(mole photons)$]. Default is 0.018.
- PboptNSC** Optimal net primary production $P_{opt,NSC}^B$ at nutrient saturation for NSC [mg C/mg Chla/hour]. Default is 1.29.
- PboptSC** Optimal net primary production $P_{opt,SC}^B$ at nutrient saturation for SC [mg C/mg Chla/hour]. Default is 4.84.
- Tfamp** Amplitude of the temperature function (9.7) [degC^{-1}]. Default is 0.81.
- Tfexp** Factor in the exponent of the temperature function (9.7). Default is 0.0631.

half-saturation constants for nutrient uptake

- KdsiSC** Half saturation constant $K_{DSi,SC}$ for DSi uptake by SC [mol $DSi\ m^{-3}$]. Default is 3.9×10^{-3} .
KnH4NSC Half saturation constant $K_{NH_4,NSC}$ for nutrient NH_4 uptake by NSC [mol $N\ m^{-3}$]. Default is 0.6×10^{-3} .
KnH4SC Half saturation constant $K_{NH_4,SC}$ for nutrient NH_4 uptake by SC [mol $N\ m^{-3}$]. Default is 1.7×10^{-3} .
Kno3NSC Half saturation constant $K_{NO_3,NSC}$ for nutrient NO_3 uptake by NSC [mol $N\ m^{-3}$]. Default is 1.02×10^{-3} .
Kno3SC Half saturation constant $K_{NO_3,SC}$ for nutrient NO_3 uptake by SC [mol $N\ m^{-3}$]. Default is 0.4×10^{-3} .
Kpo4NSC Half saturation constant $K_{PO_4,NSC}$ for nutrient PO_4 uptake by NSC [mol $P\ m^{-3}$]. Default is 0.18×10^{-3} .
Kpo4SC Half saturation constant $K_{PO_4,SC}$ for nutrient PO_4 uptake by SC [mol $P\ m^{-3}$]. Default is 2.5×10^{-3} .
CtoChlpars Parameters in the formulation (9.9) for the carbon to chlorophyll ratio [-, -, $\text{degC}^{-1}, \text{m}^2\text{day}/(\text{mole photons})$]. Default is (0.003, 0.0154, 0.050, -0.059).

mortality

- mortNSC** Mortality rate m_{NSC} of NSC [hour $^{-1}$]. Default is 0.00625.
mortSC Mortality rate m_{SC} of SC [hour $^{-1}$]. Default is 0.000625.
mortZP Mortality rate m_Z of zooplankton Z . Default is 0.0014.

grazing

- grazeff** Efficiency of zooplankton grazing rate ϵ [hour $^{-1}$]. Default is 0.75.
grazmax Maximum zooplankton grazing rate g_{max} [h $^{-1}$]. Default is 0.0754.
grazprefNSC Grazing preference $pref_{NSC}$ of NSC by zooplankton. Default is 0.3.
grazprofOM Grazing preference $pref_{OM}$ of OM by zooplankton. Default is 0.3.
grazprefSC Grazing preference of $pref_{SC}$ of SC by zooplankton. Default is 0.4.
Kgraz Half saturation constant K_g for zooplankton grazing [mg C/m 3]. Default is 0.12.

(de)nitrification

- denitrifS** Denitrification in the sediment layer $denitrif_s$ [mg C/m³/hour]. Default is 1.2.
- nitrifS** Nitrification rate in the sediment $nitrif_s$ [mg C/m²/hour]. Default is 5.7×10^{-4} .
- nitrifW** Nitrification rate in the water column $nitrif$ [mg C/m³/hour]. Default is 5.5×10^{-6}

burial

- burialBSi** Burial rate b_{BSi} of BSi in the sediment [hour⁻¹]. Default is 0.1.
- burialOMs** Burial rate b_{SOM} of SOM in the sediment [hour⁻¹]. Default is 0.1.

remineralisation rate

- reminBSi** Remineralization rate λ_{BSi}^* of BSi to DSi in the water column [hour⁻¹]. Default is 0.01.
- reminOM** Remineralization rate λ_{OM}^* of OM in the water column [hour⁻¹]. Default is 0.03.
- reminOMs** Remineralization rate λ_s^* of SOM in the sediment layer [hour⁻¹]. Default is 0.00671.

uptake preferences

- uptprefNH4** Preferential uptake f_{NH_4} of NH_4 by phytoplankton. Default is 0.75.
- uptprefNO3** Preferential uptake f_{NO_3} of NO_3 by phytoplankton. Default is 0.25.

regeneration rates

- regenBSi** Regeneration rate r_{BSi} of BSi in the sediment to DSi in the water column [day⁻¹]. Default is 0.1.
- regenNH4** Regeneration rate r_{NH_4} of NH_4 from the sediment to the water column [day⁻¹]. Default is 0.1.
- regenNO3** Regeneration rate r_{NO_3} of NO_3 from the sediment to the water column [day⁻¹]. Default is 0.1.
- regenPO4** Regeneration rate r_{PO_4} of PO_4 from the sediment to the water column [day⁻¹]. Default is 0.1.

parameters for PAR calculation

- parattchl** PAR attenuation coefficient k_{E0} for pure water used in (9.11) [m^{-1}]. Default is 0.02.
- paratt0** Coefficient k_{chl} used in (9.11) [$m^2/(mg\ Chl)$]. Default is 0.35.
- partorad** Ratio R_{par} of the surface PAR to the solar irradiance at the surface as used in (9.13) [mole photons/J]. Default is 1.88.

sinking velocities

- wsinkBSi** Sinking velocity ws_{BSi} of biogenic silicon BSi [m/hour]. Default is 1.0.
- wsinkOM** Sinking velocity ws_{OM} of organic matter OM [m/hour]. Default is 0.1.

elemental ratios

- erCinNSC** Mole of C in non-silicon consuming plankton per mole P [mole C/mole P]. Default is 106.
- erCinOM** Mole of C in organic matter per mole P [mole C/mole P]. Default is 106.
- erCinOMs** Mole of C in sediment organic matter per mole P [gram C/gram P]. Default is 106.
- erCinSC** Mole of C in silicon consuming plankton per mole P [mole C/mole P]. Default is 106.
- erCinZ** Mole of C in zooplankton per mole P [mole C/mole P]. Default is 106.
- erNinNSC** Mole of N in non-silicon consuming plankton per mole [mole C/-mole P]. Default is 16.
- erNinOM** Mole of N in organic matter per mole P [mole N/mole P]. Default is 16.
- erNinOMs** Mole of N in sediment organic matter per mole P [mole N/mole P]. Default is 16.
- erNinSC** Mole of N in silicon consuming plankton per mole P [mole C/mole P]. Default is 16.
- erNinZ** Mole of N in zooplankton per mole P [mole C/mole P]. Default is 16.
- erPinNSC** Mole of P in non-silicon consuming plankton per mole. Default is 1.

erPinOM Mole of P in organic matter per mole N [mole P/mole N]. Default is 1.

erPinOMs Mole of P in sediment organic matter per mole N [mole P/mole N]. Default is 1.

erPinSC Mole of P in silicon consuming matter per mole P .Default is 1.

erPinZ Mole of P in zooplankton per mole P. Default is 1.

erSiinSC Mole of Si in silicon consuming plankton per mole P [mole N/mole P]. Default is 12.8.

other parameters

biovarsmin2d Lower limits for 2-D state variables. Default is 0.

biovarsmin3d Lower limits for 3-D state variables. Default is 0.

index_bio_nst Indices of the 3-D state variables used for nesting per nesting file. A zero means that no nesting is applied for that specific variable. By default are variables are nested, i.e.

```
indexbio_nst(:,iset)= (/(1,l=1,nobiovars3d)/)
```

Remarks

In the setup arrays above it is recommended to use the state variable key id in stead of the variable index number. A list of all state variable key ids is given in Table 24.1.

24.2 Initial conditions for the biology module

The initial conditions for the biology are defined in `usrdef_bioics` which is called when `iopt_biology=1` and

```
modfiles(io_inicon,ics_bio,1)%status='N'

FORTRAN template

SUBROUTINE usrdef_bioics

USE iopars
USE biopars
USE biovars
USE time_routines, ONLY: log_timer_in, log_timer_out
```

```

procname(pglev+1) = 'usrdef_bioics'
CALL log_timer_in()

...
CALL log_timer_out()

RETURN

END SUBROUTINE usrdef_bioics

```

The procedures for reading and distributing (in the parallel case) the data are the same as for the hydrodynamical initial conditions. For details see Section 20.3. The following arrays can be initialised

```

! used when
REAL, DIMENSION(ncloc,nrloc,nz,nobiovars3d) :: biovars3d
REAL, DIMENSION(ncloc,nrloc,nobiovars2d) :: biovars2d      ! iopt_bio_benth>0

```

The array dimensions `ncloc` and `nrloc` indicate that these arrays are defined on the local grid in case the model is run in parallel mode. This is in contrast with the forcing file method where they are defined on the global grid. The reason is that these model arrays are only defined on the local grid in the parallel case and not on the global grid.

`biovars3d` 3-D state variables for the water column. The last array index refers to the state variable key id (see Table 24.1).

`biovars2d` 2-D state variables for the sediment layer. The last array index refers to the state variable key id (see Table 24.1).

By default, all arrays are initialised to zero.

24.3 Biology open boundary conditions

Open boundary conditions are defined in the routine `usrdef_profobc_spec` which is called for biology if

```
modfiles(io_bioobc,1,1)%status = 'N'
```

For details see Section 21.2.1. Note that the `novars` argument in the routine equals `nobiovars3d`.

Open boundary data profiles of (3-D) biological concentrations are obtained in `usrdef_prof_obcdata` which is called for sediments if

```
modfiles(io_bioobc,ifil,1)%status='N'
```

where `ifil` is the data file index which may take values between 2 and `nfiles`. For details see Section 21.2.4.

If not defined by the user, a zero gradient condition is taken by default. For more details see Section 21.2.3.

24.4 Biological nesting

In order to use biological (3-D) state variables as open boundary data in a sub-grid model using nesting, one needs

- to set the switch `iopt_nests=1`
- to set `modfiles(io_bionst,ifil,2)%status = 'W'`, where `ifil` is the number of the nested sub-grid (between 1 and `nonestsets`)
- to define the concentrations which are nested by the setup array `index_bio_nst`.

These settings are defined in the routine `usrdef_bio_params` or in the CIF block `bio_params`.

Procedures are the same as described in Section 21.3.

Table 24.1: Key ids for biological state variables.

Key id	Value	Description	Unit
3-D water column state variables			
isv_sc	1	Silicon consuming phytoplankton	mol C/m ³
isv_nsc	2	No silicon consuming plankton	mol C/m ³
isv_zp	3	Zooplankton	mol C/m ³
isv_om	4	Organic matter	mol POC/m ³
isv_po4	5	Phosphate	mol PO ₄ /m ³
isv_no3	6	Nitrate	mol NO ₃ /m ³
isv_nh4	7	Ammonium	mol NH ₄ /m ³
isv_dsi	8	Dissolved silicon	mol DSi/m ³
isv_bsi	9	Biogenic silicon	mol BSi/m ³
2-D sediment state variables			
isvs_oms	1	Sediment organic matter	mol POC/m ²
isvs_po4s	2	Sediment phosphate	mol PO ₄ /m ²
isvs_no3s	3	Sediment nitrate	mol NO ₃ /m ²
isvs_nh4s	4	Sediment ammonium	mol NH ₄ /m ²
isvs_bsis	5	Sediment biogenic silicon	mol BSi/m ²

Chapter 25

Tracers and contaminant manual

25.1 Lagrangian transport

This section describes the setup for the Lagrangian particle module. The following `usrdef_` routines, defined in `Usrdef_Particle.f90`, are available for setup. They are needed only if the module is activated by setting `iopt_part_model>0`.

- `usrdef_part_params`: switches and model parameters
- `usrdef_partics`: initial conditions for particle distributions in case the particle cloud option is switched off (`iopt_particle_cloud=0`)
- `usrdef_part_output`: routine where output is defined in a non-standard format
- `usrdef_part_spec`: properties of particles with a given label and attributes of cloud distributions
- `usrdef_pout_params`: definitions for particle trajectory output
- `usrdef_parcl_data`: release times of particle clouds and locations of the cloud center
- `usrdef_particle_grid`: definition of an external (non-COHERENS) model grid if needed
- `usrdef_particle_phys_data`: time series of hydrodynamic data from an external (non-COHERENS) model.

25.1.1 Switches and model parameters

This section describes the parameters used in the particle tracing module. They are defined in `usrdef_part_params` or in the CIF block `part_params` if it exists and the CIF is activated.

FORTRAN template

```
SUBROUTINE usrdef_part_params

USE iopars
USE partpars
USE partswitches
USE time_routines, ONLY: log_timer_in, log_timer_out

procname(pglev+1) = 'usrdef_part_params'
CALL log_timer_in()
...
CALL log_timer_out()

END SUBROUTINE usrdef_part_params
```

25.1.1.1 switches

```
INTEGER :: iopt_part_model, iopt_part_write
INTEGER :: iopt_part_cloud, iopt_part_conc, iopt_part_dens, &
           & iopt_part_hadv, iopt_part_hdif, iopt_part_leeway, &
           & iopt_part_out, iopt_part_vadv, iopt_part_vdif, &
           & iopt_part_wind
```

Two switches related to the particle module were already defined in previous routines.

1. Switch `iopt_part_model` is defined in `usrdef_init_params` (see Section 19.1.7) and selects the mode for particle tracking
 - 0: Particle module is disabled. Default.
 - 1: Particle module runs on-line. COHERENS runs in serial mode.
 - 2: Particle module runs on-line. COHERENS runs in parallel mode.
 - 3: Particle module runs off-line in forward mode and reads data from a previous (non)-COHERENS run.
 - 4: Particle module runs off-line in backward mode and reads data from a previous (non)-COHERENS run.

2. Switch `iop_t_part_write` is defined in `usrdef_mod_params` (see Section 19.2) and disables/enables the writing of hydrodynamic data from a COHERENS run (without the particle module) to an output file which can be used for a subsequent off-line run with the particle module.

0: Disabled. Default.

1: Enabled.

3. Switches defined in `usrdef_part_params`.

`iop_t_part_cloud` Disables/enables the generation of cloud distributions.

0: Cloud distributions disabled. Default.

1: Cloud distributions enabled. Particle tracking in non-cloud format is prohibited.

`iop_t_part_conc` Converts particle distributions to concentrations over the model grid by counting the number of particles per grid cell and defines the type of output. The option is used for output only. Different output formats can be used.

0: Disabled. Default.

1: As a number concentration [number/m³].

2: As a volume concentration [m³/m³]. The particle diameter `diamconc` (see Section 25.1.4.1 below) must be provided by the user.

3: As a mass concentration [kg/m³]. The particle diameter `diamconc` and density `rhoconc` (see Section 25.1.4.1 below) must be provided by the user.

`iop_t_part_dens` Type of density field for calculating the buoyant velocity.

0: Buoyancy effects disabled. Default.

1: Using a reference value ρ_0 for the water density.

2: Using the density field obtained from the equation of state.

`iop_t_part_hadv` Type of time integration scheme for horizontal advection.

0: Disabled. Default in case of 1-D simulations.

	1: First order forward Euler.
	2: Fourth order Runge-Kutta. Default for 3-D simulations.
iopt_part_hdif	Disables/enables horizontal diffusion. 0: Disabled. Default. 1: Enabled.
iopt_part_leeway	Disables/enables the Leeway effect on surface drifting (see Section 10.1.1.1). 0: Disabled. Default. 1: Enabled.
iopt_part_out	Disables/enables trajectory output. 0: Disabled. 1: Enabled.
iopt_part_vadv	Disables/enables vertical advection. 0: Disabled. 1: Enabled. Default.
iopt_part_vdif	Disables/enables vertical diffusion. 0: Disabled. Default. 1: Using a uniform diffusion coefficient. 2: Using the COHERENS diffusion coefficient ν_T or another external (non-COHERENS) diffusion coefficient.
iopt_part_wind	Disables/enables drifting of floating particles by the wind. 0: Disabled. Default. 1: Enabled.

25.1.1.2 model parameters

Model parameters for the particle module are defined in `usrdef_part_params`.

```
INTEGER :: icpart, noclouds, nolabels, nopart, nosetspart, ptime_unit
REAL :: cdrift_fac, wdrift_angle1, wdrift_angle2, wdrift_slope, &
& wranhdif, wranvdif, xdifpart_cst, ydifpart_cst, zdifpart_cst
```

cdrift_fac	The drift factor α_c representing the fraction of the surface current contributing to the drift of surface floating particles. Default is 1.0.
icpart	When the particle module runs off-line, icpart defines the number of “base” time steps for update of the hydrodynamic data. Default is 1.
noclouds	Number of particle clouds used in the simulation. Multiple release times can be defined for each cloud (see Section 25.1.6 below). Default is 1 if iopt_part_cloud =1.
nolabels	The number of labels which can be attributed to a particle. Default is 1.
nopart	Number of particles used in the simulation. The user must take care that the total number of particles released in the form of cloud distribution(s) must exactly match this value. Default is 1.
nosetspart	Number of output trajectory files. Default is 1.
ptime_unit	Unit of time used for updating the particle’s age and trajectory output. Values are <ul style="list-style-type: none"> 0: Seconds and milliseconds. 1: Seconds. Default. 2: Minutes 3: Hours 4: Days
wdrift_angle1	The angle θ_{d1} used in (10.8) [degrees]. Default is 40°.
wdrift_angle2	The angle θ_{d2} used in (10.8) [degrees]. Default is 8°.
wdrift_slope	The fraction α_w of the surface wind contributing to the drift of surface floating particles. See equation (10.8). Default is 0.0315.
wranhdif	Factor r_h used in the definition (10.3a)–(10.3b) of the horizontal diffusive velocity. Default is 2.0.
wranvdif	Factor r_v used in the definition (10.3c) of the vertical diffusive velocity. Default is 2.0.
xdifpart_cst	Coefficient ν_{H_x} of horizontal diffusion in the X-direction if iopt_part_hdif =1 [m ² /s]. Default is 0.0.
ydifpart_cst	Coefficient ν_{H_y} of horizontal diffusion in the Y-direction if iopt_part_hdif =1 [m ² /s]. Default is 0.0.

zdifpart_cst Uniform coefficient ν_{Vz} for diffusion in the vertical if **iopc_part_vdif=1** [m^2/s]. Default is 0.0.

25.1.1.3 forcing files

A number key ids for forcing files can be defined here

io_inicon If the file number **ifil=ics_part**, the forcing file contains the initial conditions for the particle module if **iotype=1** or final conditions if **iotype=2**.

io_parspc Particle attributes depending on its label, parameters used for cloud distributions.

io_parcl Cloud releases.

io_pargrd Definitions of a non-COHERENS external particle model grid if the module runs off-line (**iopc_part_model=3,4**).

io_parphs Update of non-COHERENS external physical data if the module runs off-line (**iopc_part_model=3,4**).

Important to note here is that thec attributes of these forcing files must be defined in the routine **usrdef_mod_params** or in the **CIF** block **mod_params** if it exists and the **CIF** is activated.

25.1.2 Initial conditions

The routine **usrdef_partics** is called when

```

modfiles(io_inicon,ics_part,1)%status = 'N'

FORTRAN template

SUBROUTINE usrdef_partics

USE iopars
USE partpars
USE parttypes
USE partvars
USE syspars
USE time_routines, ONLY: log_timer_in, log_timer_out

IMPLICIT NONE

procname(pglev+1) = 'usrdef_partics'
```

```

CALL log_timer_in()
...
CALL log_timer_out()

END SUBROUTINE usrdef_partics

```

In this routine initial conditions are defined in the form of attributes assigned to each particle. They are stored in a derived type vector array **part** of size **nopart**. The array and **TYPE** are defined as follows (omitting the attributes not used for setup).

```

TYPE :: PartAtts
    ! used when
    INTEGER :: state
    INTEGER :: kdistype          ! iopt_grid.nodim=3
    INTEGER :: label
    REAL (KIND=kndrlong) :: xpos
    REAL (KIND=kndrlong) :: ypos
    REAL (KIND=kndrlong) :: zpos      ! iopt_grid.nodim=3
    CHARACTER (LEN=lentime) :: starttime
    REAL (KIND=kndrlong) :: age
END TYPE PartAtts
TYPE (PartAtts), DIMENSION(nopart) :: part

```

where

state Type of drifting. No default.

- 1: Surface floating particle.
- 2: Submerged particle.

kdistype Selects the type of initial location in the vertical for submerged particles. Default is 0.

- 0: At a specified *z*-level [m].
- 1: At a specified C-node vertical grid cell.
- 2: At a given distance from the sea bed [m].
- 3: At a given distance from the sea surface [m].

label Particle label (between 1 and **nolabels**). Default is 1.

xpos The initial X-location of the particle in double precision [m or degrees longitude]. Default is 0.0.

ypos	The initial Y-location of the particle in double precision [m or degrees latitude]. Default is 0.0.
zpos	The initial vertical location of the submerged particle in double precision. Value and unit depends on the value of kdistype attribute. Default is 0.0.
	0: z -coordinate between $-h$ and ζ [m].
	1: Vertical grid level between 1 and nz .
	2: Distance from the bottom between 0 and H [m].
	3: Distance from the surface between 0 and H [m].
starttime	Time of release of the particle given as a 23-character string (YYYY/MM/DD:HH:MM:SS:mmm). No default.
age	The particle age at the time of release in double precision. Unit of time is determined by the value of the parameter ptime_unit . Usually obtained from a previous run using a final condition file. Default is 0.0.

25.1.3 Output in user-defined format

FORTRAN template

```
SUBROUTINE usrdef_part_output
```

```
IMPLICIT NONE
```

```
USE iopars
```

```
USE time_routines, ONLY: log_timer_in, log_timer_out
```

```
procname(pglev+1) = 'usrdef_part_output'
```

```
CALL log_timer_in()
```

```
...
```

```
CALL log_timer_out()
```

```
END SUBROUTINE usrdef_part_output
```

The routine allows the user to define output in non-standard COHERENS format and is the analogue of the **usrdef_output** routine in the physical model (see Section 27.7). Since the routine is called from the particle module, parameters and arrays defined in other parts of COHERENS are not accessible here when the particle module runs in parallel or off-line mode (**iopt_part_model >1**).

25.1.4 Properties depending on the particle label and cloud attributes

Particle label and cloud attributes are defined in the routine `usrdef_part_spec` if

```
modfiles(io_part_spec,1,1)%status = 'N'
```

or in the CIF block `part_spec` if it exists and the CIF is activated, or from a standard forcing file if

```
modfiles(io_part_spec,1,1)%status = 'R'
```

FORTRAN template

```
SUBROUTINE usrdef_part_spec
```

```
IMPLICIT NONE
```

```
USE iopars
```

```
USE partpars
```

```
USE parttypes
```

```
USE partvars
```

```
USE time_routines, ONLY: log_timer_in, log_timer_out
```

```
procname(pglev+1) = 'usrdef_part_spec'
```

```
CALL log_timer_in()
```

```
...
```

```
CALL log_timer_out()
```

```
END SUBROUTINE usrdef_part_spec
```

The routine has two purposes:

1. to define particle attributes depending on the value of the `label` attribute
2. to define attributes used for creating cloud distributions.

25.1.4.1 Properties of particle labels

The properties are not defined as attributes in the `part` array but in the form of arrays with dimension `nolabels`.

```
REAL, DIMENSION(nolabels) :: diamconc, rhoconc
```

where

- diamconc** Particle diameter [m]. If no value is given, the switch `iopt_part_dens` must be zero and the switch `iopt_part_conc` not larger than 1. No default.
- rhoconc** Particle mass concentration [kg/m³]. If no value is given, the switch `iopt_part_dens` must be lower than 2 and the switch `iopt_part_conc` not larger than 2. No default.

25.1.4.2 cloud attributes

The attributes for cloud distributions are defined in a derived type vector array `cloud` of size `noclouds`. The array and `TYPE` are defined as follows.

```
TYPE :: PartCloud
  INTEGER :: kdistype, label, nopart, noreleases, state
  REAL :: length, orientation, thick, width
END TYPE PartCloud
TYPE (PartCloud), DIMENSION(noclouds) :: cloud
```

where

- kdistype** Selects the type of location of the cloud centre in the vertical if `state` equals 2. Default is 0.
- 0: At a specified *z*-level.
 - 1: At a specified C-node vertical grid level.
 - 2: At a given distance from the sea bed.
 - 3: At a given distance from the sea surface.
- label** Label of the particles released in each cloud. Default is 1.
- nopart** Number of particles released by each cloud during all releases. The sum over all clouds must be equal to the value of `nopart` defined in `usrdef_part_params`. No default.
- noreleases** Number of releases for each cloud. No default.
- state** State of the particles within each cloud.
- 1: Surface floating particles.
 - 2: Submerged particles.
- length** Semi-major axis L_{cl} of the ellipsoid as defined by (10.20) [m or degrees longitude]. No default.

orientation	Angle of the semi-major axis with respect to the coordinate X-axis [degrees]. Default is 0^0 .
thick	Thickness T_{cl} of the ellipsoid as defined by (10.20) [m]. No default.
width	Semi-minor axis W_{cl} of the ellipsoid as defined by (10.20) [m or degrees latitude]. No default.

25.1.5 Particle trajectory output

This section describes the parameters used for particle output trajectories. They are defined in `usrdef_pout_params` or in the `CIF` block `pout_params` if it exists and the `CIF` is activated.

`FORTRAN` template

```
SUBROUTINE usrdef_pout_params
```

```
IMPLICIT NONE
```

```
USE iopars
USE partpars
USE parttypes
USE partvars
USE syspars
USE time_routines, ONLY: log_timer_in, log_timer_out

procname(pglev+1) = 'usrdef_pout_params'
CALL log_timer_in()
...
CALL log_timer_out()

END SUBROUTINE usrdef_pout_params
```

The routine is called if `iopt_part_out=1`. The aim is to define here the settings for trajectory output. Information about the output file(s) is stored in the derived type vector array `part1d`.

```
TYPE :: FileParams
  LOGICAL :: defined
  INTEGER :: rtype
  CHARACTER (LEN=lenform) :: form
  CHARACTER (LEN=leniofile) :: filename
END TYPE FileParams
TYPE (Fileparams), DIMENSION(nosetspart) :: part1d
```

where

defined Trajectory output is enabled if set to `.TRUE.`. Default is `.FALSE.`

form Format of the output file

- 'A': ASCII text format. Default if the program is not compiled with the `netCDF` library.
- 'U': unformatted binary
- 'N': `netCDF` format. Default if the program is compiled with the `netCDF` library.

filename Name of the output file. Default is
`TRIM(outtitle)//'-'//TRIM(cset)//'.'//tspars1.'//TRIM(suffix)` where

- outtitle** : Output title as defined in `usrdef_mod_params` (`runtile` by default).
- cset** : File number between 1 and `nosetspart`.
- suffix** : File suffix equal to 'txt', 'bin' or 'nc' if the **form** attribute equals 'A', 'U' or 'N'.

rtype This attribute determines whether output data are written in single or double precision mode. Values are

- `real_type` : Single precision.
- `rlong_type`: Double precision.
- `float_type` : Single or double precision depending on whether `COHERENS` is compiled in single or double precision mode. Default.

The type of trajectory output is defined by the attributes of the derived type vector array `outppars`.

```
TYPE :: OutPartParams
  LOGICAL :: aging
  CHARACTER (LEN=lentime) :: refdate
  INTEGER :: ktype, label, nodim, ntype, time_format
  INTEGER, DIMENSION(3) :: tlims
END TYPE OutPartParams
TYPE (OutPartParams), DIMENSION(nosetspart) :: outppars
```

The attributes are defined by

aging The age of the particles is written to output if `.TRUE.`. Default is `.FALSE.`

refdate	Reference date and time, given as a 23-character string (YYYY/MM/DD:HH:MM:SS:mmm), used for setting the initial time for particle aging and particle trajectories. Default is the start date and time of the simulation.
ktype	Selects type of vertical coordinate output. 0: <i>z</i> -coordinate. Default. 1: Distance from the sea bed. 2: Distance from the sea surface.
label	Label of the particles selected for output. Must be between 0 and nolabels . A zero (default) mean that all particles are selected.
ntype	Type of particles selected for output. 0: All particles. Default. 1: Particles with a label selected by the label attribute. 2: Particles at cloud centers. Only used when iopt_part_cloud = 1. 3: Particles at cloud centers with a label selected by the label attribute. Only used when iopt_part_cloud = 1.
nodim	Selects type output coordinates. 0: Horizontal and vertical coordinates for all particles selected with ntype . Default. 2: Horizontal coordinates for all surface drifting particles selected with ntype . 3: Horizontal and vertical coordinates for all immersed particles selected with ntype .
tlims	Start/end/step time index ¹ . Output will be written between time step tlims (1) and tlims (2) at time step intervals of tlims (3) corresponding to time intervals of timestep * tlims (3) seconds. No default.
time_format	Format of the time coordinate in the output file 0: as a date/time 23-character string (YYYY/MM/DD:HH:MM:SS:mmm) 1: seconds. Default. 2: minutes

¹The time index is defined as the number of base time steps since the start of the run.

- 3: hours
- 4: days
- 5: weeks
- 6: months (one month equals 2629800 seconds)
- 7: years (one year equals 12 months)
- 8: year number in decimal format.

The names of the variables used in the output files are

- time** Time coordinate. Unit is specified by the **time_format** parameter.
- part_xpos** The particle X-coordinates [m or degrees longitude].
- part_ypos** The particle Y-coordinates [m or degrees latitude].
- part_zpos** The particle Z-coordinates as defined by the **ktype** attribute [m].
- part_age** The particle ages. Unit is specified by the **ptime_unit** parameter.

25.1.6 Particle cloud releases

The aim is to define the time and location of cloud releases. The routine is called when

```

modfiles(io_parcl,icld,1)%status = 'N'

FORTRAN template
SUBROUTINE usrdef_parcl_data(icld,irel,ciodatetime,disdata)

CHARACTER (LEN=lentime), INTENT(OUT) :: ciodatetime
INTEGER, INTENT(IN) :: icld, irel
DOUBLE PRECISION, INTENT(INOUT), DIMENSION(3) :: disdata

USE iopars
USE syspars
USE time_routines, ONLY: log_timer_in, log_timer_out

IMPLICIT NONE

procname(ngleve+1) = 'usrdef_parcl_data'
CALL log_timer_in()
...
CALL log_timer_out()

END SUBROUTINE usrdef_parcl_data

```

where

icld Cloud number between 1 and `noclouds`.
irel The release index which must be between 1 and `cloud(icld)%norelease`.
ciodatetime Release date and time corresponding to the release index irel given as a 23-character string (YYYY/MM/DD:HH:MM:SS:mmm).
disdata Coordinates of the cloud centre. The vector components represent
 1: X-coordinate in double precision [m or degrees longitude].
 2: Y-coordinate in double precision [m or degrees latitude].
 3: Z-coordinate. The format and unit depend on the value of `cloud(icld)%kdistype`. No value needs to be given if the cloud contains only surface floating particles (`cloud(icld)%state=1`).

The routine `usrdef_parcl_data` is called at the initial time. The locations of the cloud centres and the time of release are then obtained for each release and each cloud.

25.1.7 External model grid for off-line applications

The routine `usrdef_particle_grid` is called when `iopt_part_model=3,4` and

```

modfiles(io_pargrd,1,1)%status = 'N'

FORTRAN template

SUBROUTINE usrdef_particle_grid

IMPLICIT NONE

USE depths
USE grid
USE gridpars
USE iopars
USE phspars
USE switches
USE time_routines, ONLY: log_timer_in, log_timer_out

procname(pglev+1) = 'usrdef_particle_grid'
CALL log_timer_in()

```

```

...
CALL log_timer_out()

END SUBROUTINE usrdef_particle_grid

```

When the particle module runs in off-line mode, a model grid needs to be defined either from a previous COHERENS run or from a non-COHERENS grid. In the first case, the data are written from a previous run with COHERENS with the switch `iopt_part_write` set to 1. In the second case, the data are supplied by calling this routine. The only constraints for the external grid are that it must be, like COHERENS, rectilinear or curvilinear in the horizontal and must use σ -coordinates in the vertical.

The following arrays must or may be defined here.

```

! used when
REAL, DIMENSION(nc, nr) :: gxcoord
REAL, DIMENSION(nc, nr) :: gycoord
REAL, DIMENSION(nz+1) :: gsigcoordatw      ! iopt_grid_nodim=3,
                                                ! iopt_grid_vtype=1,2
REAL, DIMENSION(nc-1, nr-1, nz) :: gscoordatc ! iopt_grid_nodim=3,
                                                ! iopt_grid_vtype=3
REAL, DIMENSION(nc, nr-1, nz) :: gscoordatu  ! iopt_grid_nodim=3,
                                                ! iopt_grid_vtype=3
!
REAL, DIMENSION(nc-1, nr, nz) :: gscoordatv  ! iopt_grid_nodim=3,
                                                ! iopt_grid_vtype=3
REAL, DIMENSION(nc-1, nr-1, nz+1) :: gscoordatw ! iopt_grid_nodim=3,
                                                ! iopt_grid_vtype=3
REAL, DIMENSION(nc-1, nr-1) :: delxatc
REAL, DIMENSION(nc-1, nr-1) :: delyatc
REAL, DIMENSION(nc-1, nr-1) :: depmeanatc
REAL, DIMENSION(nc, nr-1) :: depmeanatu
REAL, DIMENSION(nc-1, nr) :: depmeanatv
LOGICAL, DIMENSION(nc-1, nr-1) :: maskatc
LOGICAL, DIMENSION(nc, nr-1) :: maskatu
LOGICAL, DIMENSION(nc-1, nr) :: maskatv
REAL :: gacc_mean
REAL :: density_ref

```

Note that the arrays are defined on the global physical grid. The following data are always needed

`gxcoord` X-coordinates of the model grid at corner nodes [m or degrees longitude].

gycoord	Y-coordinates of the model grid at corner nodes [m or degrees latitude].
delxatc	Grid spacings at the C-nodes in the X-direction [m].
delyatc	Grid spacings at the C-nodes in the Y-direction [m].
depmeanatc	Mean water depths at C-nodes [m].
depmeanatu	Mean water depths at U-nodes [m].
depmeanatv	Mean water depths at V-nodes [m].
maskatc	Grid mask (.TRUE. at sea and .FALSE. at land C-node grid cells).
maskatu	Grid mask (.TRUE. at sea and .FALSE. at land/coastal U-node cell faces).
maskatv	Grid mask (.TRUE. at sea and .FALSE. at land/coastal land V-node cell faces).
gacc_mean	Acceleration of gravity taken as constant throughout the domain [m/s ²]. No default.
density_ref	Reference water density taken as constant throughout the domain [km/m ³]. No default.

The σ -coordinate arrays needed for 3-D simulations depend on the type of the σ -grid:

1. horizontally uniform σ -grid

gsigcoordatc σ -coordinates along the vertical C-nodes
gsigcoordatw σ -coordinates along the vertical W-nodes. Bottom and surface values must be 0, respectively 1.

2. horizontally non-uniform σ -grid

gscoordatc σ -coordinates at the C-nodes
gscoordatu σ -coordinates at the U-nodes
gscoordatv σ -coordinates at the V-nodes
gscoordatw σ -coordinates at the W-nodes. Bottom and surface values must be 0, respectively 1.

25.1.8 External hydrodynamic data for off-line applications

The routine `usrdef_particle_phys_data` is called when `iopt_part_model=3,4` and

```

modfiles(io_parphs,1,1)%status = 'N'

FORTRAN template

SUBROUTINE usrdef_particle_phys_data(ciodatetime)
CHARACTER (LEN=lentime), INTENT(OUT) :: ciodatetime

IMPLICIT NONE

USE currents
USE density
USE depths
USE gridpars
USE iopars
USE partswitches
USE partvars
USE switches

procname(pglev+1) = 'usrdef_particle_phys_data'
CALL log_timer_in()
...
CALL log_timer_out()

END SUBROUTINE usrdef_particle_phys_data

```

where

ciodatetime Current date and time given as a 23-character string
 (YYYY/MM/DD:HH:MM:SS:mmm).

When the particle module runs in off-line mode, physical data need to be supplied either from a previous COHERENS run or from simulations with a non-COHERENS model. In the first case, the data are available in standard format and this routine does not to be called. In the second case, the data need to be provided by this routine. As stated above, the only constraints for the external grid are that it must be, like COHERENS, rectilinear or curvilinear in the horizontal and must use σ -coordinates in the vertical. The routine will be called initially and after **icpart** time steps.

The following arrays must or may be defined here.

```

! used when
REAL, DIMENSION(nc-1, nr-1) :: deptotatc
REAL, DIMENSION(nc, nr-1) :: deptotatu

```

```

REAL, DIMENSION(nc-1,nr) :: deptotatv
REAL, DIMENSION(nc,nr-1,nz) :: uvel
REAL, DIMENSION(nc-1,nr,nz) :: vvel
REAL, DIMENSION(nc-1,nr-1,nz+1) :: wvel      ! iopt_grid_nodim=3
REAL, DIMENSION(nc-1,nr-1,nz+1) :: vdifcoefpart ! iopt_part_vdif=2
REAL, DIMENSION(nc-1,nr-1,nz) :: dens        ! iopt_part_dens=2
LOGICAL, DIMENSION(nc-1,nr-1) :: maskatc     ! iopt_fld=1
LOGICAL, DIMENSION(nc,nr-1) :: maskatu       ! iopt_fld=1
LOGICAL, DIMENSION(nc-1,nr) :: maskatv       ! iopt_fld=1

```

1. Always

deptotatc Mean water depth at C-nodes [m].
deptotatu Mean water depth at U-nodes [m].
deptotatv Mean water depth at V-nodes [m].
uvel X-component of the current [m/s].
vvel Y-component of the current [m/s].

2. 3-D case

wvel Transformed vertical current ω at W-nodes [m/s]. Bottom and surface values must be zero.

3. 3-D case using random walk method for vertical diffusion and an external diffusion coefficient

vdifcoefpart Vertical diffusion coefficient at W-nodes [m^2/s].

4. 3-D case including buoyancy with a non-uniform density field

dens Water density at C-nodes [kg/m^3].

5. With drying/wetting scheme activated

maskatc Grid mask (.TRUE. at sea and .FALSE. at dry/land C-node grid cells).

maskatu Grid mask (.TRUE. at sea and .FALSE. at dry/land/coastal U-node cell faces).

maskatv Grid mask (.TRUE. at sea and .FALSE. at dry/land/coastal V-node cell faces).

Chapter 26

Structure and discharge manual

This chapter explains the setup of the structure module. The following routines are defined in *Usrdef_Structures.f90*:

- `usrdef_dry_cells`: setup of the dry cells module
- `usrdef_thin_dams`: setup of the thin dams module
- `usrdef_weirs`: setup of the weirs/barriers module
- `usrdef_dischr_spec`: discharge locations
- `usrdef_dischr_data`: defines discharge input data

26.1 Dry cells

The routine `usrdef_dry_cells` is called if

```
iopt_dry_cells = 1
modfiles(io_drycel,1,1)%status='N'
```

FORTRAN template

```
SUBROUTINE usrdef_dry_cells
```

```
USE iopars
USE structures
USE time_routines, ONLY: log_timer_in, log_timer_out
IMPLICIT NONE
```

```

procname(pglev+1) = 'usrdef_dry_cells'
CALL log_timer_in()
...
CALL log_timer_out()

RETURN

END SUBROUTINE usrdef_dry_cells

```

The following setup arrays are defined

```

INTEGER, DIMENSION(numdry) :: idry
INTEGER, DIMENSION(numdry) :: jdry

```

where `numdry` is the number of dry cells at C-nodes and
`idry` (Global) X-indices of the dry cells at C-nodes.
`jdry` (Global) Y-indices of the dry cells at C-nodes.

26.2 Thin dams

The routine `usrdef_thindams` is called if

```

iopt_thndam = 1
modfiles(io_thndam,1,1)%status='N'
FORTRAN template
SUBROUTINE usrdef_thndams

USE iopars
USE structures
USE time_routines, ONLY: log_timer_in, log_timer_out

IMPLICIT NONE

procname(pglev+1) = 'usrdef_thndams'
CALL log_timer_in()
...
CALL log_timer_out()

RETURN

END SUBROUTINE usrdef_thndams

```

The following setup arrays are defined

```
INTEGER, DIMENSION(numthinu) :: ithinu
INTEGER, DIMENSION(numthinu) :: jthinu
INTEGER, DIMENSION(numthinv) :: ithinv
INTEGER, DIMENSION(numthinv) :: jthinv
```

where **numthinu** and **numthinv** are the the number of thin dams at U- respectively V-nodes

ithinu	(Global) X-indices of thin dams at U-nodes.
jthinu	(Global) Y-indices of thin dams at U-nodes.
ithinv	(Global) X-indices of thin dams at V-nodes.
jthinv	(Global) Y-indices of thin dams at V-nodes.

Remarks

- Thin dams can only be specified along lines parallel to one of the numerical grid axes.
- No thin dams can (obviously) be defined at open boundaries or at the edges of the computational grid but thin dams perpendicular to open boundaries are allowed.

26.3 Weirs and barriers

The routine `usrdef_weirs` is called if

```
iopt_weibar = 1
modfiles(io_weibar,1,1)%status='N'

FORTRAN template
SUBROUTINE usrdef_weirs

USE iopars
USE structures
USE time_routines, ONLY: log_timer_in, log_timer_out

IMPLICIT NONE

procname(pglev+1) = 'usrdef_weirs'
CALL log_timer_in()
```

```

...
CALL log_timer_out()

RETURN

END SUBROUTINE usrdef_weirs

```

The following arrays are defined over here

```

INTEGER, DIMENSION(numwbaru) :: iwbaru
INTEGER, DIMENSION(numwbaru) :: jwbaru
REAL, DIMENSION(numwbaru) :: oricoefu
REAL, DIMENSION(numwbaru) :: oriheightu
REAL, DIMENSION(numwbaru) :: oriwidthu
REAL, DIMENSION(numwbaru) :: wbarcoefu
REAL, DIMENSION(numwbaru) :: wbarcrestu
REAL, DIMENSION(numwbaru) :: wbarmodlu
INTEGER, DIMENSION(numwbarv) :: iwbarv
INTEGER, DIMENSION(numwbarv) :: jwbarv
REAL, DIMENSION(numwbarv) :: oricoefv
REAL, DIMENSION(numwbarv) :: oriheightv
REAL, DIMENSION(numwbarv) :: oriwidthv
REAL, DIMENSION(numwbarv) :: wbarcoefv
REAL, DIMENSION(numwbarv) :: wbarcrestv
REAL, DIMENSION(numwbarv) :: wbarmodlv

```

where `numwbaru` and `numwbarv` are the the number of weirs/barriers at U-respectively V-nodes and

<code>iwbaru</code>	(Global) X-indices of weirs or barriers at U-nodes.
<code>jwbaru</code>	(Global) Y-indices of weirs or barriers at U-nodes.
<code>oricoefu</code>	Discharge coefficient C_e for orifices at U-nodes.
<code>oriheightu</code>	Sill heighth O_h at U-nodes [m].
<code>oriwidthu</code>	Orifice width O_w at U-nodes [m].
<code>wbarcoefu</code>	Discharge coefficient C_{st} at U-nodes [$m^{1/2}/s$].
<code>wbarcrestu</code>	Heighth of the weir crests h_{cr} at U-nodes [m].
<code>wbarmodlu</code>	Modular limit m at the U-nodes.
<code>iwbarv</code>	(Global) X-indices of weirs or barriers at V-nodes.
<code>jwbarv</code>	(Global) Y-indices of weirs or barriers at V-nodes.

oricoefv Discharge coefficient C_e for orifices at V-nodes.
oriheightv Sill height O_h at V-nodes [m].
oriwidthv Orifice width O_w at V-nodes [m].
wbarcoefv Discharge coefficient C_{st} at V-nodes [$m^{1/2}/s$].
wbarcrestv Height of the weir crests h_{cr} at V-nodes [m].
wbarmodlv Modular limit m at V-nodes.

Remarks

- The structure is defined as a barrier if the corresponding value of **oriwidthu** or **oriwidthv** has a non-negative value, otherwise it becomes a weir.
- Energy loss at weirs and barriers can generate a strong flow convergence (retardation) and therefore large currents magnitudes and gradients. This may require small time steps.
- For the same reason, it is recommended to define weirs and barriers sufficiently away from the open boundaries.
- There are no defaults.

26.4 Specifiers for discharges

The type of vertical location of the discharges is defined in the routine `usrdef_dischr_spec` if

```
modfiles(io_disspec,1,1)%status = 'N'
```

or in the **CIF** block `dischr_spec` if it exists and the **CIF** is activated, or from a standard forcing file if

```
modfiles(io_disspc,1,1)%status = 'R'
```

The routine `usrdef_dischr_spec` is called if `iopt_dischr=1` and

```
modfiles(io_disspec,1,1)%status='N'
```

FORTRAN template

```

SUBROUTINE usrdef_dischr_spec

USE iopars
USE structures
USE time_routines, ONLY: log_timer_in, log_timer_out

IMPLICIT NONE

procname(pglev+1) = 'usrdef_dischr_spec'
CALL log_timer_in()
...
CALL log_timer_out()

RETURN

END SUBROUTINE usrdef_dischr_spec

```

The following arrays are defined

INTEGER, DIMENSION(numdis) :: kdistype

where numdis is the number of discharge locations and

kdistype Selects the type of vertical location of the discharge

- 0: uniformly distributed over the vertical
- 1: at the fixed grid cell
- 2: at a fixed distance from the sea bed
- 3: at a fixed distance from the sea surface.

26.5 Discharge data

The routine **usrdef_dischr_data** is called if

```

iopt_dischr = 1
modfiles(iddesc,1,1)%status='N'

```

where **iddesc** is the file descriptor defined below.

FORTRAN template

```

SUBROUTINE usrdef_dischr_data(iddesc,ifil,ciodate,disdata,nodat,novars)
CHARACTER (LEN=lentime), INTENT(OUT) :: ciodate

```

```

INTEGER, INTENT(IN) :: iddesc, ifil, nodat, novars
REAL, INTENT(INOUT), DIMENSION(nodat,novars) :: disdata

procname(pglev+1) = 'usrdef_dischr_data'
CALL log_timer_in()
...
CALL log_timer_out()

RETURN

END SUBROUTINE usrdef_dischr_data

```

The **INTENT(IN)** arguments have the following meaning:

iddesc The file descriptor of the corresponding data file

- io_disloc** discharge locations
- io_disvol** volume discharge data
- io_discur** momentum discharge data
- io_dissal** salinity discharge data
- io_distmp** temperature discharge data
- io_dissed** sediment discharge data.

ofil The file index.

nodat Number of discharge locations (must be equal to **numdis**).

novars Number of input data variables which depends on the value of **iddesc** (see below).

The following **INTENT(OUT)** variables must be defined here

ciodatetime date/time in string format¹

disdata discharge data.

The supplied data must be stored in **disdata** and in the order given below. The type of data depends on the value of **iddesc**

io_disloc **xdiscoord** X-coordinates of the discharge locations [m or degrees longitude].

ydiscoord Y-coordinates of the discharge locations [m or degrees latitude].

¹If the parameter **time_zone** is defined with a non-zero value, the time of the input data must be given in local time.

zdiscoord value depends on the value of **kdistype**

- 0: Not used (discharge is spread over the vertical).
- 1: At the vertical grid level where the discharge takes place (value is converted to **INTEGER**)
- 2: Distance to the bottom where the discharge takes place [m].
- 3: Distance to the surface where the discharge takes place [m].

io_disvol	disvol	Volume discharge [m^3/s].
io_discur	disarea	Area over which the discharge takes place [m^2].
	disdir	Discharge direction with respect to X-axis (Eastward direction in the spherical case) [rad].
io_dissal	dissal	Salinity of the discharged water mass [PSU].
io_distmp	distmp	Temperature of the discharged water mass [$^{\circ}\text{C}$].
io_dissed	dissed	Discharged volume of sediment concentrations for each fraction [m^3/m^3]. Note that novars equals in this case the number of sediment fractions nf .

The names of the arrays in the listing above correspond to the ones used in the model. They are only given here for clarity.

Chapter 27

User output

27.1 Introduction

Three types of “standard” user output can be defined in COHERENS:

1. The source file *Usrdef_Time_Series.f90* defines setup parameters and data for time series output. Time series output is enabled when `iopt_out_tsers=1` (default value).
2. The source file *Usrdef_Time_Averages.f90* defines setup parameters and data for time averaged output. Time averaged output is enabled when `iopt_out_avrgd=1`.
3. The source file *Usrdef_Harmonic_Analysis.f90* defines setup parameters and data for harmonic analysis and output. Harmonic analysis and output are enabled when `iopt_out_anal=1`.

For each of the 3 types of output, the user can define one or more “output” sets. For each output set the user must

- Select which variables are used for output and define their attributes.
- Define the attributes of the output file(s).
- Define the attributes of the output grid both in space and time (regular or non-regular, resolution in space and time, . . .).

These attributes are stored in a series of derived arrays discussed in Section 27.2.

There are three categories of output variables and files

- One-dimensional (0-D) data, i.e. variables without spatial dimension in the output file.
- Two-dimensional (2-D) data, i.e. variables having a horizontal but no vertical dimension in the output file.
- Three-dimensional (3-D) data, i.e. variables having a horizontal and vertical dimension in the output file.

In each set the user can define no more than one 0-D, one 2-D or one 3-D output file.

The `usrdef` routines for each type are discussed in Sections 27.3-27.5. The output files not only contain the model data and metadata but also the coordinates of the output grid. In this way each output file can be considered as self-consistent and used by independent postprocessing programs for display and analysis. The structure of a standard COHERENS output file is discussed in Section 27.6.

To allow the user to create and write output in a user specified (non-standard) format, an empty `usrdef_output` routine has been provided in COHERENS. This is further discussed in Section 27.7

27.2 Output attributes

27.2.1 Output variables

The user must select which variables are used for output and define a number of attributes for each variable. They are stored in a derived type vector array of type `VariableAtts`.

```
TYPE :: VariableAtts
  LOGICAL :: convert
  CHARACTER (LEN=lenname) :: f90_name
  CHARACTER (LEN=lendesc) :: long_name, vector_name
  CHARACTER (LEN=lenunit) :: units
  INTEGER :: isub, ivarid, klev, nrank, oopt
  REAL :: dep
  REAL, DIMENSION(2) :: convert_fac
END TYPE VariableAtts
```

where

ivarid	Key id of the variable. The FORTRAN name of a variable key id is <code>iarr_*</code> where <code>*</code> denotes the FORTRAN name of the variable (see below), e.g. <code>iarr_temp</code> for temperature. A full listing of the variables names which can be used for output is given in Table ???. If zero, the variable is non-standard and has to be defined by the user. Default is zero.
f90_name	The FORTRAN name of the variable. Must only be provided if <code>ivarid=0</code> .
long_name	A long descriptive name of the variable. Must only be provided if <code>ivarid=0</code> .
units	A character string specifying the unit of the variable. Must only be provided if <code>ivarid=0</code> or <code>convert</code> is <code>.TRUE.</code> (see below).
vector_name	If the variable represents the component of a vector, the character string specifies a long descriptive name of the corresponding vector. Must only be provided if <code>ivarid=0</code> .
convert	Converts the output data to a new unit if <code>.TRUE.</code> . Default is <code>.FALSE.</code>
convert_facs	Converts output data to a new unit, if <code>convert</code> is <code>.TRUE.</code> , using the formula <code>newval = convert_facs(1)*oldval + convert_facs(2)</code> No default.
nrank	Dimension (rank) of the output variable which can take the values 0, 2, 3. Details are given below.
isub	This attribute must be defined if the variable is defined in the model with an extra “variable” dimension. This is the case for variables used in the sediment module where the last index of some sediment variables represents the fraction index.
oopt	Output values can be obtained by applying an operator on the model variable. The following values are allowed <ul style="list-style-type: none"> oopt_null No operator is applied. The <code>nrank</code> attribute equals the “natural” (0-D, 2-D, 3-D) rank of the variable as used in <code>COHERENS</code>. Default. oopt_mean Result depends on the natural rank of the model variable and the rank of the output data. <ul style="list-style-type: none"> 0: The output value is the domain average in case of a 3-D or the surface average in case of a 2-D variable.

	2: The output value is the depth-mean of a 3-D variable.
<code>oopt_int</code>	Result depends on the natural rank of the model variable and the rank of the output data. 0: The output value is the domain integrated value in case of a 3-D or the surface integrated value in case of a 2-D variable. 2: The output value is the depth-integrated value of a 3-D variable.
<code>oopt_max</code>	Result depends on the natural rank of the model variable and the rank of the output data. 0: The output value is the domain maximum in case of a 3-D or the surface maximum in case of a 2-D variable. 2: The result is the maximum of a 3-D variable over the water depth.
<code>oopt_min</code>	Result depends on the natural rank of the model variable and the rank of the output data. 0: The output value is the domain minimum in case of a 3-D or the surface minimum in case of a 2-D variable. 2: The result is the minimum over the water depth of a 3-D variable.
<code>oopt_klev</code>	Rank of the output data is 2. The output value is the value of a 3-D variable at a specified vertical grid level.
<code>oopt_dep</code>	Rank of the output data is 2. The output value is the value of a 3-D variable at a specified depth below the surface grid level.
<code>klev</code>	Defines the output vertical level if <code>oopt</code> equals <code>oopt_klev</code> . No default.
<code>dep</code>	Defines the water depth (below the surface) if <code>oopt</code> equals <code>oopt_dep</code> . No default.

Remarks

- The variables in the array must be defined in increasing order of rank, i.e. firstly variables of rank 0 (if any), next of rank 2 (if any), third of rank 3 (if any).

- Output data of rank 1 are considered in COHERENS as profile data defined at specific “station” locations (see Section 27.2.4).
- As already mentioned above, the rank of the output data is not the same as the one of the corresponding model variable. This is the case when an operator is applied on the model data which is different from `oopt_null` or in case of a multi-variable model array (such as sediment concentrations). If output is requested for the same model variable with different operators or with different values of the `isub` attribute, the attributes must be defined for each case separately with the same `ivarid`.
- The values of the `long_name` and `units` attributes for standard variables have been selected in compliance with COARDS, UDUNITS and CF (Climate and Forecast) international standards. For details see [Eaton et al. \(2011\)](#), [Pincus & Rew \(2008\)](#).

27.2.2 Output files

Attributes of the output files are defined through of a derived type array of type `FileParams`.

```
TYPE :: FileParams
  LOGICAL :: defined
  INTEGER :: rtype
  CHARACTER (LEN=lenform) :: form
  CHARACTER (LEN=leniofile) :: filename
END TYPE FileParams
```

where

`defined` Output data will be written when the attribute is set to `.TRUE.`.
Default is `.FALSE.`.

`form` File format.

'A': ASCII (portable, sequential).
'U': Unformatted binary (non-portable,sequential).
'N': netCDF format (portable, non-sequential).

`filename` File name. Default names are given in the next sections below for each type of output.

rtype This attribute determines whether output data are written in single (32 bits) or in double precision (64 bits) mode. Values are

`real_type` : Single precision.

`rlong_type`: Double precision.

`float_type` : Single or double precision depending on whether COHERENS is compiled in single or double precision mode. Default.

27.2.3 Output space and time grid

Output grid and time resolution are selected through of a derived type array of type `OutGridParams`.

```
TYPE :: OutGridParams
  LOGICAL :: gridded, packing, time_grid
  CHARACTER (LEN=lentime) :: refdate
  INTEGER :: nostats, time_format, vcoord
  INTEGER, DIMENSION(3) :: tlims, xlims, ylims, zlims
END TYPE OutGridParams
TYPE (OutGridParams), DIMENSION(nosetstsr) :: tsrgpars
```

where

gridded If `.TRUE.`, output data are defined on a regular sub-grid part of the model grid or on the whole model grid. If `.FALSE.`, the data are taken at a number of irregularly spaced locations, denoted below as “stations”. Default is `.FALSE.`.

packing A land mask will be applied if `.TRUE.` and `gridded=.TRUE.`. This means that the data on the horizontal grid will be stored as a vector excluding land points. A 3-D or 2-D output array is then written as a 2-D respectively 1-D array. Advantage is a reduction of storage. Disadvantage is that the data cannot be viewed without postprocessing. A 1-D mask array is supplied so that the output data can be reconverted to their original shape by postprocessing. Default is `.FALSE.`.

xlims Start/end/step grid index in the X-direction index in case `gridded=.TRUE.`. This defines the output grid in the X-direction. Default values are `(1,nr-1,1)` which means that the output and model grid are the same in the X-direction.

ylims	Start/end/step grid index in the Y-direction index in case <code>gridded=TRUE</code> . This defines the output grid in the Y-direction. Default values are <code>(1,nr-1,1)</code> which means that the output and model grid are the same in the Y-direction.
zlims	Start/end/step grid index in the Z-direction index. This defines the output grid in the vertical direction. Default values are <code>(1,nz,1)</code> which means that the output and model grid are the same in the vertical direction.
tlims	Selects the resolution in time and the period in case of time averaging or harmonic analysis. Its meaning is discussed below for each type of output. There is no default.
vcoord	Defines how the vertical grid is defined in the output file. <ol style="list-style-type: none"> 1: Using z-coordinates with respect to the mean or to the actual water depth depending on the value of <code>time_grid</code> (see below). Default. 2: Using σ-coordinates.
nostats	Number of data stations in case <code>gridded=FALSE</code> .
time_format	Defines the format of the time coordinate. If 0 or 7, the time is defined using an absolute format. Otherwise the time is defined in a numerical format as the time passed since a reference date (see below). The time unit depends on the value <code>time_format</code> . <ol style="list-style-type: none"> 0: as a date/time 23-character string (YYYY/MM/DD;HH:MM:SS:mmm) 1: seconds. Default. 2: minutes 3: hours 4: days 5: weeks 6: months (one month equals 2629800 seconds) 7: years 8: year number in decimal format. If <code>time_format</code> is between 1 and 7, the time in the output file is defined as the time passed since a given reference date (see below).
refdate	Reference date/time used when a relative time is used for output (<code>time_format</code> between 1 and 7). Default is the start date of the

run CStartTime. The date is rounded to the nearest minute, hour, ... depending on the value of `time_format`.

`time_grid` This attribute can only be defined for time series output. If `.TRUE.`, the output data grid is taken as time-dependent since the vertical locations in a σ -grid depend on time. Otherwise, the vertical location is taken with respect to the mean water level. For more information see Section 27.6.1. Default is `.FALSE.`.

27.2.4 Output data stations

Station attributes have to be defined if `gridded=.FALSE.` for at least one output set. They are stored in a derived type array of type `StationLocs`.

```
TYPE :: StationLocs
  INTEGER :: xpos, ypos
  CHARACTER (LEN=lennname) :: name
END TYPE StationLocs
```

where

`xpos` X-coordinate of the station locations [m or degrees longitude, positive East].

`ypos` Y-coordinate of the station locations [m or degrees latitude, positive North].

`name` Descriptive names of the stations. Default is ‘Station X’ where ‘X’ is the station’s array index.

27.2.5 Output procedures

The procedure for obtaining the output data depends on the `gridded` attribute:

- If `.TRUE.`, the output values are given at C-nodes. Interpolation is performed if the variable is defined at a U-, V-, W-, or UV-node. Data are flagged if the C-node cell is a land or a temporarily dry cell in case that the drying-wetting scheme is activated.
- If `.FALSE.`, variables not defined at C-nodes (e.g. vector components) are interpolated at C-nodes. The output value at the station location (specified by the `xpos` and `ypos` attributes) is obtained by bi-linear interpolation. If one or more, but not all, of the interpolating cells are

located at a land or at a temporarily dry cell (in case that the drying-wetting scheme is activated), a reduced form of interpolation is used. If all surrounding cells are land or dry, the output value is flagged.

In case of a standard variable, the `ivarid` attribute is given by the user while the output value is automatically produced by the program and no further action is required. When `ivarid` is zero, the variable is non-standard and has to be defined by the user. Three `usrdef_` routines have been implemented for this purpose for each type of output (time series, time averaging, harmonic analysis). They are further described below.

27.3 Time series output

The following routines, defined in `Usrdef_Time_Series.f90`, are available for setting up time series output:

- `usrdef_tsr_params`: Defines, for each output set, the output variable attributes, the attributes of the output files, the attributes of the output grid, the attributes of the output stations and other specific arrays.
- `usrdef_tsr0d_vals`: Definition of non-standard 0-D output variables.
- `usrdef_tsr2d_vals`: Definition of non-standard 2-D output variables.
- `usrdef_tsr3d_vals`: Definition of non-standard 3-D output variables.

The following parameters were already defined in `usrdef_mod_params` or in the **CIF**

- | | |
|-------------------------|---|
| <code>nosettsr</code> | Number of file sets for time series output. |
| <code>novarstsr</code> | Number of variables used for time series output. |
| <code>nostatstsr</code> | Number of output stations for non-gridded (station) time series output (see below). |

27.3.1 Specifiers for time series output

Setup parameters for time series output are defined, when `iopt_out_tsers=1`, in the routine `usrdef_tsr_params` or in the **CIF** block `tsr_params` if it exists and the **CIF** is activated.

FORTRAN template

```

SUBROUTINE usrdef_out_params

USE iopars
USE time_routines, ONLY: log_timer_in, log_timer_out

IMPLICIT NONE

procname(pglev+1) = 'usrdef_out_params'
CALL log_timer_in()
...
CALL log_timer_out()

RETURN

END SUBROUTINE usrdef_out_params

```

The following arrays are defined here

```

TYPE (VariableAtts), DIMENSION(novarstsr) :: tsrvars
TYPE (FileParams), DIMENSION(nosetstsr) :: tsr0d, tsr2d, tsr3d
TYPE (OutGridParams), DIMENSION(nosetstsr) :: tsrgpars
TYPE (StationLocs), DIMENSION(nostatstsr) :: tsrstatlocs
INTEGER, DIMENSION(novarstsr,nosetstsr) :: ivarstsr
INTEGER, DIMENSION(nostatstsr,nosetstsr) :: lstatstsr

```

where

- | | |
|-------------|--|
| tsrvars | Attributes of the output variables. See Section 27.2.1 above. |
| tsr0d | Attributes of the 0-D output files. A 0-D output file can be created if at least one of the output variables has rank 0. See Section 27.2.2 above. |
| tsr2d | Attributes of the 2-D output files. A 2-D output file can be created if at least one of the output variables has rank 2. See Section 27.2.2 above. |
| tsr3d | Attributes of the 3-D output files. A 3-D output file can be created if at least one of the output variables has rank 3. See Section 27.2.2 above. |
| tsrgpars | Attributes of the output grid and time resolution (see Section 27.2.3 above). |
| tsrstatlocs | Station attributes (see Section 27.2.4 above). |

- ivarsts** Let **lvar** = **ivarsts**(**ivar**,**iset**). This means that the variable whose attributes are defined in **tsrvrs**(**lvar**) is selected for output in set **iset**. Zero values are ignored. The number of non-zero values must be not be larger then **novarsts**. Non-defined values are automatically set to zero.
- lstatsts** Let **lstat**=**lstatsts**(**lstat**,**iset**). This means that the station whose attributes are defined in **tsrstatlocs**(**lstat**) is selected for output in set **iset**. Zero values are ignored. The number of non-zero values must be equal to **tsrgpars**(**iset**)%**nostats**. Non-defined values are automatically set to zero.

Remarks

- If the **filename** attribute is not defined, the following default name is used

```
filename = TRIM(outtitle)//'_//TRIM(cset)//'.'//&
& 'tsout'//cdim//'.'//TRIM(suffix)
```

where **outtitle** is the title for output files, defined in **usrdef_mod_params**, **cset** the number of the output set, **cdim** the rank of the output data (0, 2 or 3) and **suffix** depends on the value of the **form** attribute, i.e. '**txt**' for ASCII, '**bin**' for unformatted binary and '**nc**' for netCDF files.

- The **tlms** vector vattribute of **tsrgpars** represent the start/end/step time index¹. Output will be written between time step **tlms**(1) and **tlms**(2) at time step intervals of **tlms**(3) i.e. at time intervals of **timestep*****tlms**(3) seconds.

27.3.2 Time series output data

Three **usrdef_** routines, described below, have been implemented where the user defines the values (at C-nodes) of non-standard output variables (i.e. with a variable id **varid** equal to zero). Contrary to the standard procedures in **COHERENS** there exists no alternative with the **CIF** or standard forcing file method.

27.3.2.1 0-D time series data

The subroutine **usrdef_tsr0d_vals** defines the 0-D (spatially independent) data for time series output and is called when **iopt_out_tsers**=1 and the key id **tsrvrs**(**ivar**)%**varid** for at least one 0-D variable **ivar** equals zero.

¹The time index is defined as the number of base time steps since the start of the run.

FORTRAN template

```
SUBROUTINE usrdef_tsr0d_vals(out0ddat,n0vars)

INTEGER, INTENT(IN) :: n0vars
REAL, INTENT(OUT), DIMENSION(n0vars) :: out0ddat

IMPLICIT NONE

RETURN

END SUBROUTINE usrdef_tsr0d_vals
```

where

n0vars Number of 0-D output variables.

out0ddat 0-D output data. The variables are given in the same order as they are defined in the array **tsrvs** so that **out0ddat(1)** is the value of the first 0-D variable and **out0ddat(n0vars)** of the last 0-D variable. Output data of standard variables are ignored and don't need to be defined here.

27.3.2.2 2-D time series data

The subroutine **usrdef_tsr2d_vals** defines the 2-D (horizontal, but vertically independent) data for time series output and is called when **iopt_out_tsers=1** and the key id **tsrvs(ivar)%varid** for at least one 2-D variable **ivar** equals zero.

FORTRAN template

```
SUBROUTINE usrdef_tsr2d_vals(out2ddat,i,j,n2vars)

INTEGER, INTENT(IN) :: i, j, n2vars
REAL, INTENT(OUT), DIMENSION(n2vars) :: out2ddat

IMPLICIT NONE

RETURN

END SUBROUTINE usrdef_tsr2d_vals
```

where

n2vars Number of 2-D output variables.
i X-index of the data location on the *local* (in the parallel case) model grid (between 1 and **ncloc**).
j Y-index of the data location on the *local* (in the parallel case) model grid (between 1 and **nrloc**).
out2ddat 2-D output data. The variables are given in the same order as they are defined in the array **tsrvs** so that **out2ddat(1)** is the value of the first 2-D variable and **out2ddat(n2vars)** of the last 2-D variable. Output data of standard variables are ignored and don't need to be defined here.

27.3.2.3 3-D time series data

The subroutine **usrdef_tsr3d_vals** defines the 3-D data for time series output and is called when **iopt_out_tsers=1** and the key id **tsrvs(ivars)%varid** for at least one 3-D variable **ivar** equals zero.

FORTRAN template

```

SUBROUTINE usrdef_tsr3d_vals(out3ddat,i,j,k,n3vars)

INTEGER, INTENT(IN) :: i, j, k, n3vars
REAL, INTENT(OUT), DIMENSION(n3vars) :: out3ddat

IMPLICIT NONE

RETURN

END SUBROUTINE usrdef_tsr3d_vals

```

where

n3vars Number of 3-D output variables.
i X-index of the data location on the *local* (in the parallel case) model grid (between 1 and **ncloc**).
j Y-index of the data location on the *local* (in the parallel case) model grid (between 1 and **nrloc**).
k Vertical index of the data location on the model grid (between 1 and **nz**).

`out3ddat` 3-D output data. The variables are given in the same order as they are defined in the array `tsrvars` so that `out3ddat(1)` is the value of the first 3-D variable and `out3ddat(n3vars)` of the last 3-D variable. Output data of standard variables are ignored and don't need to be defined here.

27.4 Time averaged output

Time averaging is defined through three time parameters:

1. A start time t_1 for the first averaging.
2. An end time t_2 for the last averaging.
3. The averaging period T .

The number of averaging periods is then defined by $L = (t_2 - t_1)/T$. Averaging is then performed for the consecutive intervals $[t_1 + (l - 1)T, t_1 + lT]$ with $1 \leq l \leq L$.

The following routines, defined in `Usrdef_Time_Averages.f90`, are available for setting up time averaged output:

- `usrdef_avr_params`: Defines, for each output set, the output variable attributes, the attributes of the output files, the attributes of the output grid, the attributes of the output stations and other specific arrays.
- `usrdef_avr_params`: Defines, for each output set, the output variable attributes, the attributes of the output files and the attributes of the output grid and other specific arrays.
- `usrdef_avr0d_vals`: Definition of non-standard 0-D output variables.
- `usrdef_avr2d_vals`: Definition of non-standard 2-D output variables.
- `usrdef_avr3d_vals`: Definition of non-standard 3-D output variables.

The following parameters were already defined in `usrdef_mod_params` or in the CIF

- | | |
|-------------------------|---|
| <code>nosetsavr</code> | Number of file sets for time averaged output. |
| <code>novarsavr</code> | Number of variables used for time averaged output. |
| <code>nostatsavr</code> | Number of output stations for non-gridded (station) time averaged output (see below). |

27.4.1 Specifiers for time averaged output

Setup parameters for time averaged output are defined, when `iopt_out_avrgd=1`, in the routine `usrdef_avr_params` or in the CIF block `avr_params` if it exists and the CIF is activated.

FORTRAN template

```
SUBROUTINE usrdef_avr_params
```

```
USE iopars
USE time_routines, ONLY: log_timer_in, log_timer_out

IMPLICIT NONE

procname(pglev+1) = 'usrdef_avr_params'
CALL log_timer_in()
...
CALL log_timer_out()

RETURN

END SUBROUTINE usrdef_avr_params
```

The following arrays are defined here

```
TYPE (VariableAtts), DIMENSION(novarsavr) :: avrvars
TYPE (FileParams), DIMENSION(nosetsavr) :: avr0d, avr2d, avr3d
TYPE (OutGridParams), DIMENSION(nosetsavr) :: avrgpars
TYPE (StationLocs), DIMENSION(nostatsavr) :: avrstatlocs
INTEGER, DIMENSION(novarsavr,nosetsavr) :: ivarsavr
INTEGER, DIMENSION(nostatsavr,nosetsavr) :: lstatsavr
```

where

- `avrvars` Attributes of the output variables. See Section 27.2.1 above.
- `avr0d` Attributes of the 0-D output files. A 0-D output file can be created if at least one of the output variables has rank 0. See Section 27.2.2 above.
- `avr2d` Attributes of the 2-D output files. A 2-D output file can be created if at least one of the output variables has rank 2. See Section 27.2.2 above.
- `avr3d` Attributes of the 3-D output files. A 3-D output file can be created if at least one of the output variables has rank 3. See Section 27.2.2 above.

avrgpars	Attributes of the output grid and time resolution (see section 27.2.3 above).
avrstatlocs	Station attributes (see Section 27.2.4 above).
ivarsavr	Let $lvar = \text{ivarsavr}(ivar, iset)$. This means that the variable whose attributes are defined in avrvars(lvar) is selected for output in set iset . Zero values are ignored. The number of non-zero values must be not be larger then novarsavr . Non-defined values are automatically set to zero.
lstatsavr	Let $lstat = \text{lstatsavr}(lstat, iset)$. This means that the station whose attributes are defined in avrstatlocs(lstat) is selected for output in set iset . Zero values are ignored. The number of non-zero values must be equal to avrgpars(iset)%nostats . Non-defined values are automatically set to zero.

Remarks

- If the **filename** attribute is not defined, the following default name is used

```
filename = TRIM(outtitle)//'_//TRIM(cset)//'.//&
& 'avrgd'//cdim//'.//TRIM(suffix)
```

where **outtitle** is the title for output files, defined in **usrdef_mod_params**, **cset** the number of the output set, **cdim** the dimension of the output data (0, 2 or 3) and **suffix** depends on the value of the **form** attribute, i.e. 'txt' for ASCII, 'bin' for unformatted binary and 'nc' for **netCDF** files.

- The **tlims** vector attribute of **avrgpars** represent the start/end time index² and averaging period. The elements of **tlims** then refer to the following times
 - 1: The start time t_1 of the first period divided by **timestep**.
 - 2: The end time t_2 of the last period divided by **timestep**.
 - 3: The averaging period divided by **timestep**.

27.4.2 Time averaged output data

Three **usrdef_** routines, described below, have been implemented where the user defines the values (at C-nodes) of non-standard output variables (i.e.

²The time index is defined as the number of base time steps since the start of the run.

with a variable id `varid` equal to zero). Contrary to the standard procedures in `COHERENS` there exists no alternative with the `CIF` or standard forcing file method.

27.4.2.1 0-D time averaged data

The subroutine `usrdef_avr0d_vals` defines the 0-D (spatially independent) data for time averaged output and is called when `iopt_out_avrgd=1` and the key id `avrvars(ivar)%varid` for at least one 0-D variable `ivar` equals zero.

FORTRAN template

```
SUBROUTINE usrdef_avr0d_vals(out0ddat,n0vars)

INTEGER, INTENT(IN) :: n0vars
REAL, INTENT(OUT), DIMENSION(n0vars) :: out0ddat

IMPLICIT NONE

RETURN

END SUBROUTINE usrdef_avr0d_vals
```

where

`n0vars` Number of 0-D output variables.

`out0ddat` 0-D output data. The variables are given in the same order as they are defined in the array `avrvars` so that `out0ddat(1)` is the value of the first 0-D variable and `out0ddat(n0vars)` of the last 0-D variable. Output data of standard variables are ignored and don't need to be defined here.

27.4.2.2 2-D time averaged data

The subroutine `usrdef_avr2d_vals` defines the 2-D (horizontally dependent, but vertically indendent) data for time averaged output and is called when `iopt_out_avrgd=1` and the key id `avrvars(ivar)%varid` for at least one 2-D variable `ivar` equals zero.

FORTRAN template

```
SUBROUTINE usrdef_avr2d_vals(out2ddat,i,j,n2vars)
```

```

INTEGER, INTENT(IN) :: i, j, n2vars
REAL, INTENT(OUT), DIMENSION(n2vars) :: out2ddat

IMPLICIT NONE

RETURN

END SUBROUTINE usrdef_avr2d_vals

```

where

n2vars Number of 2-D output variables.
i X-index of the data location on the *local* (in the parallel case) model grid (between 1 and **ncloc**).
j Y-index of the data location on the *local* (in the parallel case) model grid (between 1 and **nrloc**).
out2ddat 2-D output data. The variables are given in the same order as they are defined in the array **avrvars** so that **out2ddat(1)** is the value of the first 2-D variable and **out2ddat(n2vars)** of the last 2-D variable. Output data of standard variables are ignored and don't need to be defined here. The variables are given in the same order as they are defined in the array **avrvars** where index 1 refers to the first 2-D variable and the index **n2vars** to the last 2-D variable in **avrvars**. Output data of standard variables are ignored and don't need to be defined here.

27.4.2.3 3-D time averaged data

The subroutine **usrdef_avr3d_vals** defines the 3-D data for time averaged output and is called when **iopt_out_avrgd=1** and the key id **avrvars(ivar)%varid** for at least one 3-D variable **ivar** equals zero.

FORTRAN template

```

SUBROUTINE usrdef_avr3d_vals(out3ddat,i,j,k,n3vars)

INTEGER, INTENT(IN) :: i, j, k, n3vars
REAL, INTENT(OUT), DIMENSION(n3vars) :: out3ddat

IMPLICIT NONE

```

RETURN

END SUBROUTINE `usrdef_avr3d_vals`

where

n3vars Number of 3-D output variables.
i X-index of the data location on the *local* (in the parallel case) model grid (between 1 and **ncloc**).
j Y-index of the data location on the *local* (in the parallel case) model grid (between 1 and **nrloc**).
k Vertical index of the data location on the model grid (between 1 and **nz**).
out3ddat 3-D output data. The variables are given in the same order as they are defined in the array **avrvars** so that **out3ddat(1)** is the value of the first 3-D variable and **out3ddat(n3vars)** of the last 3-D variable. Output data of standard variables are ignored and don't need to be defined here.

27.5 Harmonic analysis and output

The following routines, defined in *Usrdef_Harmonic_Analysis.f90*, are available for setting up harmonic analysis and output

- **usrdef_anal_freqs**: Defines the frequencies for harmonic analysis and their attributes.
- **usrdef_anal_params**: Defines, for each output set, the output variable attributes, the attributes of the output files, the attributes of the output grid, the attributes of the output stations and other specific arrays.
- **usrdef_anal0d_vals**: Definition of non-standard 0-D output variables.
- **usrdef_anal2d_vals**: Definition of non-standard 2-D output variables.
- **usrdef_anal3d_vals**: Definition of non-standard 3-D output variables.

The following parameters were already defined in **usrdef_mod_params** or in the **CIF**

nosetsanal Number of file sets for harmonic analysis.

nofreqsanal Number of frequencies used in the harmonic analysis.

- novarsanal** Number of variables used for harmonic output.
nostatsanal Number of output stations for non-gridded (station) harmonic output (see below).

27.5.1 Harmonic frequencies

This section describes the parameters used to define the frequencies for harmonic setup. They are defined, when `iopt_out_anal=1`, in the routine `usrdef_anal_freqs` or in the CIF block `anal_freqs` if it exists and the CIF is activated.

FORTRAN template

```
SUBROUTINE usrdef_anal_freqs
```

```
!---frequencies used for harmonic analysis
USE iopars
USE time_routines, ONLY: log_timer_in, log_timer_out

IMPLICIT NONE

procname(pglev+1) = 'usrdef_anal_freqs'
CALL log_timer_in()
...
CALL log_timer_out()

RETURN
```

```
END SUBROUTINE usrdef_anal_freqs
```

The following arrays are defined

```
INTEGER, DIMENSION(nofreqsanal) :: index_anal
INTEGER, DIMENSION(nosetsanal) :: nofreqsharm
INTEGER, DIMENSION(nofreqsanal,nosetsanal) :: ifreqsharm
```

where

- index_anal** Key ids of the constituents used in harmonic analysis (as defined in `tide.f90`).
- nofreqsharm** Number of frequencies used in each output set.
- ifreqsharm** Frequency indices used in each output set. For each set the values to be defined are `ifreqsharm(1:nofreqsharm(iset),iset)`. The tidal constituent related to `icon=ifreqsharm(ifreq,iset)` is then given by `index_anal(icon)`.

27.5.2 Specifiers for harmonic output

Setup parameters for harmonic output are defined, when `iopt_out_tanal=1`, in the routine `usrdef_anal_params` or in the CIF block `anal_params` if it exists and the CIF is activated.

FORTRAN template

```
SUBROUTINE usrdef_anal_params

!---parameters and attributes for harmonic analysis
USE iopars
USE time_routines, ONLY: log_timer_in, log_timer_out

IMPLICIT NONE

procname(pglev+1) = 'usrdef_anal_params'
CALL log_timer_in()
...
CALL log_timer_out()

RETURN

END SUBROUTINE usrdef_anal_params
```

The following arrays are defined here

```
TYPE (VariableAtts), DIMENSION(novarsanal) :: analvars
TYPE (FileParams), DIMENSION(nosetsanal) :: res0d, res2d, res3d
TYPE (FileParams), DIMENSION(nofreqsanal,nosetsanal) :: amp0d, amp2d, amp3d
TYPE (FileParams), DIMENSION(nofreqsanal,nosetsanal) :: pha0d, pha2d, pha3d
TYPE (FileParams), DIMENSION(nofreqsanal,nosetsanal) :: ell2d, ell3d
TYPE (OutGridParams), DIMENSION(nosetsanal) :: analgpars
TYPE (StationLocs), DIMENSION(nostatsanal) :: analstatlocs
INTEGER, DIMENSION(novarsanal,nosetsanal) :: ivarsanal
INTEGER, DIMENSION(nostatsanal,nosetsanal) :: lstatsanal
INTEGER, DIMENSION(14,nosetsanal) :: ivarsell
INTEGER, DIMENSION(2,nosetsanal) :: ivec112d, ivec113d
```

where

`analvars` Attributes of the output variables. See Section 27.2.1 above.

`res0d` Attributes of the 0-D output file(s) for tidal residuals. A 0-D output file can be created if at least one of the output variables has rank 0. See Section 27.2.2 above.

amp0d	Attributes of the 0-D output file(s) for tidal amplitudes. A 0-D output file can be created if at least one of the output variables has rank 0. See Section 27.2.2 above.
pha0d	Attributes of the 0-D output file(s) for tidal phases [degrees]. A 0-D output file can be created if at least one of the output variables has rank 0. See Section 27.2.2 above.
res2d	Attributes of the 2-D output file(s) for tidal residuals. A 2-D output file can be created if at least one of the output variables has rank 2. See Section 27.2.2 above.
amp2d	Attributes of the 2-D output file(s) for tidal amplitudes. A 2-D output file can be created if at least one of the output variables has rank 2. See Section 27.2.2 above.
pha2d	Attributes of the 2-D output file(s) for tidal phases [degrees]. A 2-D output file can be created if at least one of the output variables has rank 2. See Section 27.2.2 above.
ell2d	Attributes of the 2-D output file(s) for tidal ellips parameters. A 2-D output file can be created if at least one of the output variables has rank 2. See Section 27.2.2 above.
res3d	Attributes of the 3-D output files for tidal residuals. A 3-D output file can be created if at least one of the output variables has rank 3. See Section 27.2.2 above.
amp3d	Attributes of the 3-D output files for tidal amplitudes. A 3-D output file can be created if at least one of the output variables has rank 3. See Section 27.2.2 above.
pha3d	Attributes of the 3-D output files for tidal phases [degrees]. A 3-D output file can be created if at least one of the output variables has rank 3. See Section 27.2.2 above.
ell3d	Attributes of the 3-D output files for tidal ellipse parameters. A 3-D output file can be created if at least one of the output variables has rank 3. See Section 27.2.2 above.
analgpars	Attributes of the output grid and time resolution (see section 27.2.3 above).
analstatlocs	Station attributes (see Section 27.2.4 above).
ivarsanal	Let $lvar = ivarsanal(ivars, iset)$. This means that the variable whose attributes are defined in <code>analvars(lvar)</code> is selected for output in set <code>iset</code> . Zero values are ignored. The number of non-zero values must be not be larger than <code>novarsanal</code> . Non-defined values are automatically set to zero.

- lstatsanal** Let `lstat=lstatsanal(istat,iset)`. This means that the station whose attributes are defined in `analstatlocs(lstat)` is selected for output in set `iset`. Zero values are ignored. The number of non-zero values must be equal to `analgpars(iset)%nostats`. Non-defined values are automatically set to zero.
- ivarsell** Selects which elliptic parameters are used for each output set. Values are between 1–7 for 2-D and 8–14 for 3-D output. Their meaning is explained in Table 27.1.
- ivecell2d** The array elements `ivecell2d(1,iset)` and `ivecell2d(2,iset)` are the variable indices in `analvars` of the X- and Y-component of the 2-D vector used to obtain the harmonic ellipses in case of 2-D output. See Section 5.12.2.
- ivecell3d** The array elements `ivecell3d(1,iset)` and `ivecell3d(2,iset)` are the variable indices in `analvars` of the X- and Y-component of the 3-D vector used to obtain the harmonic ellipses in case of 3-D output. See Section 5.12.2.

Table 27.1: List of available elliptic parameters (number, variable key id, description of the parameter and the dimension of the vector which determines the ellipse).

Number	Key id	Description	Dimension	Unit
1	<code>iarr_ellmaj2d</code>	Major axis	2	<code>m/s</code>
2	<code>iarr_ellmin2d</code>	Minor axis	2	<code>m/s</code>
3	<code>iarr_ellip2d</code>	Ellipticity	2	–
4	<code>iarr_ellinc2d</code>	Inclination	2	degrees
5	<code>iarr_ellpha2d</code>	Elliptic phase	2	degrees
6	<code>iarr_ellcc2d</code>	Cyclonic component	2	<code>m/s</code>
7	<code>iarr_ellac2d</code>	Anticyclonic component	2	<code>m/s</code>
8	<code>iarr_ellmaj3d</code>	Major axis	3	<code>m/s</code>
9	<code>iarr_ellmin3d</code>	Minor axis	3	<code>m/s</code>
10	<code>iarr_ellip3d</code>	Ellipticity	3	–
11	<code>iarr_ellinc3d</code>	Inclination	3	degrees
12	<code>iarr_ellpha3d</code>	Elliptic phase	3	degrees
13	<code>iarr_ellcc3d</code>	Cyclonic component	3	<code>m/s</code>
14	<code>iarr_ellac3d</code>	Anticyclonic component	3	<code>m/s</code>

Remarks

- If the `filename` attribute is not defined, the following default name is used for respectively tidal residuals, amplitudes, phases and elliptic parameters

```

filename = TRIM(outtitle)//'_'//TRIM(cset)//'.'//&
           & 'resid'//cdim//'.//TRIM(suffix)
filename = TRIM(outtitle)//'_'//TRIM(cset)//'.'//&
           & freqnam//'amplt'//cdim//'.//TRIM(suffix)
filename = TRIM(outtitle)//'_'//TRIM(cset)//'.'//&
           & freqnam//'phase'//cdim//'.//TRIM(suffix)
filename = TRIM(outtitle)//'_'//TRIM(cset)//'.'//&
           & freqnam//'ellip'//cdim//'.//TRIM(suffix)

```

where `outtitle` is the title for output files, defined in `usrdef_mod_params`, `cset` the number of the output set, `cfreq` the name of the tidal constituent (e.g. 'M2'), `cdim` the dimension of the output data (0, 2 or 3) and `suffix` depends on the value of the `form` attribute, i.e. 'txt' for ASCII, 'bin' for unformatted binary and 'nc' for netCDF files.

- The `tlims` vector attribute represents the start/end time index³ and period used for harmonic analysis. The elements of `tlims` have the following meaning
 - 1: Start time t_1 of the first period divided by `timestep`.
 - 2: End time t_2 of the last period divided by `timestep`.
 - 3: Harmonic period divided by `timestep`.
- The arrays `ivarsell`, `iveccell2d`, `iveccell3d` only need to be defined if elliptic parameters are requested for output.
- The procedures for harmonic analysis are discussed in Section 5.12.

27.5.3 Harmonic output data

Three `usrdef_` routines, described below, have been implemented where the user defines the values (at C-nodes) of non-standard output variables (i.e. with a variable id `varid` equal to zero). Contrary to the standard procedures in **COHERENS** there exists no alternative with the **CIF** or standard forcing file method.

³The time index is defined as the number of base time steps since the start of the run.

27.5.3.1 0-D harmonic data

The subroutine `usrdef_anal0d_vals` defines the 0-D (space-independent) data for harmonic output and is called when `iopt_out_anal=1` and the key id `analvars(ivar)%varid` for at least one 0-D variable `ivar` equals zero.

FORTRAN template

```
SUBROUTINE usrdef_anal0d_vals(out0ddat,n0vars)

!---0-D data for harmonic analysis
INTEGER, INTENT(IN) :: n0vars
REAL, INTENT(OUT), DIMENSION(n0vars) :: out0ddat

IMPLICIT NONE

RETURN

END SUBROUTINE usrdef_anal0d_vals
```

where

`n0vars` Number of 0-D output variables.

`out0ddat` 0-D output data. The variables are given in the same order as they are defined in the array `analvars` so that `out0ddat(1)` is the value of the first 0-D variable and `out0ddat(n0vars)` of the last 0-D variable. Output data of standard variables are ignored and don't need to be defined here.

27.5.3.2 2-D harmonic data

The subroutine `usrdef_anal2d_vals` defines the 2-D (horizontally dependent, but vertically indendent) data for harmonic output and is called when `iopt_out_anal=1` and the key id `analvars(ivar)%varid` for at least one 2-D variable `ivar` equals zero.

FORTRAN template

```
SUBROUTINE usrdef_anal2d_vals(out2ddat,i,j,n2vars)

!---2-D data for harmonic analysis
INTEGER, INTENT(IN) :: i, j, n2vars
REAL, INTENT(OUT), DIMENSION(n2vars) :: out2ddat
```

```

IMPLICIT NONE

RETURN

END SUBROUTINE usrdef_anal2d_vals

```

where

n2vars Number of 2-D output variables.
i X-index of the data location on the *local* (in the parallel case) model grid (between 1 and **ncloc**).
j Y-index of the data location on the *local* (in the parallel case) model grid (between 1 and **ncloc**).
out2ddat 2-D output data. The variables are given in the same order as they are defined in the array **analvars** so that **out2ddat(1)** is the value of the first 2-D variable and **out2ddat(n2vars)** of the last 2-D variable. Output data of standard variables are ignored and don't need to be defined here.

27.5.3.3 3-D harmonic data

The subroutine **usrdef_anal3d_vals** defines the 3-D (spatially dependent) data for harmonic output and is called when **iopt_out_anal=1** and the key id **analvars(ivar)%varid** for at least one 3-D variable **ivar** equals zero.

FORTRAN template

```
SUBROUTINE usrdef_anal3d_vals(out3ddat,i,j,k,n3vars)
```

```

!---3-D data for harmonic analysis
INTEGER, INTENT(IN) :: i, j, k, n3vars
REAL, INTENT(OUT), DIMENSION(n3vars) :: out3ddat

```

IMPLICIT NONE

RETURN

```
END SUBROUTINE usrdef_anal3d_vals
```

where

n3vars	Number of 3-D output variables.
i	X-index of the data location on the <i>local</i> (in the parallel case) model grid (between 1 and ncloc).
j	Y-index of the data location on the <i>local</i> (in the parallel case) model grid (between 1 and nrloc).
k	Vertical index of the data location on the model grid (between 1 and nz).
out3ddat	3-D output data. The variables are given in the same order as they are defined in the array analvars so that out3ddat(1) is the value of the first 3-D variable and out3ddat(n3vars) of the last 3-D variable. Output data of standard variables are ignored and don't need to be defined here.

27.6 Output data grid, metadata and output formats

Firstly, it is strongly recommended to select **netCDF** format for output, since several graphical software programs (python, MATLAB, FERRET, ncview, ...) accept **netCDF** data without a need for an intermediate post-processing program.

27.6.1 Output data grid

The following coordinate arrays are written, depending on the rank of the output grid and the **time_grid** attribute

1. **time**: Output times. Format is defined by the **time_format** attribute and is either absolute when it is written as a calendar string or as the year number, or (following the CF-conventions) relative as the time since the reference date given by the **refdate** attribute. This is the only coordinate array used for 0-D output.
2. **x**: X-coordinates of the data grid when a Cartesian grid is selected [m].
3. **lon**: X-coordinates of the data grid when a spherical grid is selected [degrees longitude, positive East].
4. **y**: X-coordinates of the data grid when a Cartesian grid is selected [m.]
5. **lat**: Y-coordinates of the data grid when a spherical grid is selected [degrees latitude, positive North].

6. **depth**: Mean water depth h (bathymetry) [m].
7. **zmean**: Mean Z-coordinate when the **vcoord** attribute is set to 1 and the **time_grid** is `.FALSE.` [m]. The mean Z-coordinate is obtained using $z = h(\sigma - 1)$ and only written at the first output time.
8. **z**: Z-coordinate when the **vcoord** attribute is set to 1 and the **time_grid** is `.TRUE.` [m]. The Z-coordinate is obtained using $z = \sigma H - h$ and written at all output times.
9. **lev**: The σ -coordinate vector when **vcoord** is set to 2. For compatibility with the standard (CF) conventions its value is given by $\sigma^* = \sigma - 1 = (z - \zeta)/H$ and varies between -1 and 0.
10. **zetout**: The water elevation ζ [m]. The variable is only included when **time_grid** is `.TRUE.` and **vcoord** is set to 2. The z -coordinate can then be determined in a postprocessing program using $z = (h + \zeta)\sigma^* - h$.
11. **seapoint**: This integer vector array is used when the data are provided in compact form (omitting permanent land cells) if the **packing** attribute is switched on. The size of the vector is given by the number of sea cells **nowetout**. Values are the indices of the wet points in **FORTRAN** order. The grid indices i, j of each sea point on the 2-D output grid can be recovered from this array in a post-processing program using

```

DO l=1,nowetout
  n = seapoint(l)
  j = (n-1)/ncout + 1
  i = n - (j-1)*ncout
ENDDO

```

where **ncout,nrout** are the horizontal dimensions of the output grid.

12. **statnames**: Names of the data stations when an irregular data grid is selected (**gridded** attribute set to `.FALSE.`).

27.6.2 Data file info

The data file is composed of two sections. The first is the header section containing the metadata with general information and properties of the (coordinate and data) variables written to the output file.

27.6.2.1 metadata

In case of a netCDF output file, the metadata are created using the standard rules for netCDF files. Details are found in (Russ *et al.*, 2004; Eaton *et al.*, 2011). An example of the header section in ASCII format is shown below⁴.

```

nGlbatts      : 6
Conventions   : CF-1.6
title         : Time series data: plume1B
history       : created 2021-07-22 10:57:49
institution   : RBINS - Operational Directorate Natural Environment
source        : Coherens version V2.12
nDimensions   : 4
Dimnames      : time,x,y,z
Dimensions    : 25,120,40,1
nCoordinates  : 5
nVariables    : 4
time_unit     : hours
StartDate     : 2003/01/03;00:00:00:000
time_step     : 3.
packing       : F
realtype      : real
FillValue     : 0.996921E+37
varid         : 1
DataType      : real
Rank          : 1
Shape         : 120
f90_name      : x
long_name     : X-coordinate
units         : m
format        : (50G16.7)
varid         : 2
DataType      : real
Rank          : 1
Shape         : 40
f90_name      : y
long_name     : Y-coordinate
units         : m
format        : (50G16.7)

```

⁴The same header info is written, now in binary format in case of an unformatted 'U' output file.

```
varid          : 3
DataType       : real
Rank           : 2
Shape          : 120,40
f90_name       : depth
long_name      : Mean water depth
units          : m
format         : (50G16.7)
FillValue      : 0.996921E+37
varid          : 4
DataType       : real
Rank           : 1
Shape          : 1
f90_name       : lev
long_name      : Sigma coordinate
units          : -
format         : (50G16.7)
varid          : 5
DataType       : double
Rank           : 1
Shape          : 25
f90_name       : time
long_name      : Time
units          : hours since 2003-01-03 00:00:00
format         : (50E23.14E3)
varid          : 6
DataType       : real
Rank           : 4
Shape          : 120,40,1,25
f90_name       : uvel
long_name      : X-component of current
units          : m s-1
format         : (50G16.7)
vector_name    : Current
FillValue      : 0.996921E+37
varid          : 7
DataType       : real
Rank           : 4
Shape          : 120,40,1,25
f90_name       : vvel
long_name      : Y-component of current
```

```

units          : m s-1
format        : (50G16.7)
vector_name   : Current
FillValue     : 0.996921E+37
varid         : 8
DataType      : real
Rank          : 4
Shape         : 120,40,1,25
f90_name      : wphys
long_name     : Physical vertical velocity
units          : m s-1
format        : (50G16.7)
vector_name   : Current
FillValue     : 0.996921E+37
varid         : 9
DataType      : real
Rank          : 4
Shape         : 120,40,1,25
f90_name      : sal
long_name     : Salinity
units          : PSU
format        : (50G16.7)
FillValue     : 0.996921E+37
#

```

Example 27.1: Example of a header section in ASCII fotrmat for a user-defined output file.

Global attributes are defined on lines 2–6 following the CF-standards. The “institute” attribute can be reset by the user using the `CF_institute` model parameter (see Section 19.3.16). Other global parameters are defined on lines 7–17 (number and names of dimensions, number of coordinates and data variables, time unit, start date, output time time step, value of the packing attribute, value of the data flag used for land or temporiraly dry cells. The `packing` attribute is `.TRUE.` when the data are written in compressed format (see sections above). In the remaining part of the header the properties of firstly the coordinate and next the data variables are given.

27.6.2.2 data

When `netCDF` format is used, the data are written using the routines defined in the `netCDF` manual (Russ *et al.*, 2004; Eaton *et al.*, 2011). In case of ASCII

or unformatted binary files, the data are written in the following order:

1. Time-independent coordinate arrays.
2. Time-dependent coordinate `time` and, if `time_grid=.TRUE.`, either `zout` or `zetout` depending on the value of `vcoord` attribute.
3. Variable arrays in the order defined in the header.

In case of ASCII or unformatted binary files, each data output is preceded by a line with the name of the (coordinate or data variable). Output format is either by default or selected by the user (see Section 19.3.17).

27.7 User-defined output

The routine `usrdef_output` is intended for users who like to define output in their own format. The user is responsible for implementing the code. There are no general rules, except that the routine is called at each base time step. Examples are provided for most test cases and can be found in the subdirectories of `setups`.

Chapter 28

Central Input File

28.0.1 Syntax of a CIF

As shown in the example 28.1 below, each data line in the CIF has the following syntax

```
varname = value 1, value 2, ..., value n
```

where **varname** is the FORTRAN name of a model parameter and **value 1** to **value n** are the input data values for the parameter, separated by the data separator ','. The file is read line-wise. The data strings **value 1**, **value 2**, ... are converted to the appropriate (numeric, logical, character) data format associated with the FORTRAN variable **varname**. If **varname** corresponds to a scalar, it is obvious that only one value needs to be given and there is no data separator. The following rules apply

- If a comment character '!' appears on the data line, all characters in the string, starting from this character are ignored. However, the comment character can only appear at the first position of the data line (in which case the entire line is ignored) or after the last character of the last data string.
- No error occurs if a model scalar or array parameter does not appear on any input line in which case the default value is retained.
- If a data string contains only blanks or equals the null string, the value of the corresponding model parameter is undefined, in which case its default value is retained. When the CIF is written by the program, all variables (even defaults) are defined in the data strings.

- The characters in the string **varname** are case insensitive. If the **CIF** is written by the program, the names are always given in upper case characters.
- Data strings corresponding to a logical variable can take either of the forms **T/F**, **.T./.F.** or **.TRUE./.FALSE..**
- Data strings corresponding to a character variable are written without parentheses.
- If **varname** corresponds to a vector, the number of data can be lower than the size of the vector in which case the non-defined values are set to their defaults. However if a vector has a specified “physical” size and there are no specific defaults, all expected data must be given. Examples are the arrays **index_abc** (physical size given by **nconabc**) or **ntrestart** (physical size given by **nrestarts**).
- If the model parameter represents a 2-dimensional array, say A , with **M** columns and **N** rows, there should be **M+1** data strings on the right hand side. The first one is the row index j , the next ones the data values $A(i, j)$ for $j = 1, M$. The full array is read row-wise from N data lines with the same **varname**. The generic format is

```

A = 1, A(1,1), ..., A(M,1)
...
A = N, A(1,N), ..., A(M,N)

```

As before, data are set to their defaults if a data line contains less than $N+1$ values.

- Arrays of non-derived type and of rank higher than 2, are not allowed in the **CIF**.
- The procedures for derived type arrays are case-independent. They will be described in Section 28.2.2 for each case.
- In the **usrdef** method use was made of key ids. In the **CIF**, **integer** key ids are not very usefull. They are therefore replaced by their key name. This is further discussed below for each particular case.
- If consecutive data strings on the right of the **CIF** line have the same value, the line can be shortened by using a repeat counter. The general format is **n\$value**. For example **5\$1** is equivalent to **1,1,1,1,1,.**

- Parameters in a **CIF** block (defined below) can be read from an external file (in **CIF** format) using the **include** utility. Format is

```
include filename
```

where **filename** is the name of the external **CIF**.

- When a **CIF** is written by the program, all setup parameters are included in the file. The values are either the default settings or the re-defined values after a call from the appropriate **usrdef_** routine or the ones reset by the program after a call to a **reset_** routine. Only exception to this rule is the parameter **cold_start** which is always written as **.FALSE.** and can only be changed by editing the **CIF** manually.

The **CIF** utility uses the following special characters:

- ',' Separates the data strings on an input line.
- '=' Separates the string **varname** from the data strings. Must be on all input lines except those starting with a '!' or '#' character.
- '!' Indicates the start of a comment. All characters on the input line at and beyond this character are ignored.
- '#' Block separator (see below). Must always be the first character on a separator line.
- '\$' Used in repeat constructs.

These special characters cannot be used in the string **varname** or in a data string representing a string variable.

28.1 CIF blocks

The **CIF** consists of a number of so-called **CIF** blocks. Each block consists of a number of **CIF** data lines. The name of the block is given on the first line. The block ends with a line starting with **#**.

```
block name 1
...
#
block name 2
...
#
...
```

The maximum number of allowed **CIF** blocks is 15. Each block represents the equivalent of a corresponding **usrdef** routine. The name of a block is also the name of the **usrdef** routine without **usrdef_** prefix. A list of all **CIF** block names is given in Table 28.1. The last column indicates whether data input from a data block is subject to a constraint.

The **CIF** procedure for reading the **CIF** is then as follows

1. The status of the **CIF** is set to 'R' on the input of the **defruns** file (see Section 18.3).
2. Firstly, the program searches which block names in the **CIF** blocks are included. If a name is found, the corresponding block will be activated.
3. Input data will be read for each activated **CIF** block, provided that the constraints, listed in Table 28.1, is .TRUE.. Block reading occurs at different places in the code and terminates at the end of the initialisation part of the simulation.
4. Input from an activated block will be terminated if the input line contains the # character in the first column.

The following rules apply

- A **CIF** block can easily be deactivated by commenting the line containing the block name (i.e. putting a '!') in the first column. In that case the block will not be activated even when the constraint is satisfied.
- While the order of **CIF** lines is irrelevant, **CIF** parameters can only be defined within their respective block.
- If an **include** statement occurs in a block, the corresponding file may only contain input lines associated with this block.

28.2 Special procedures

28.2.1 Key ids

In the **usrdef_** procedure, use was made of key ids which are defined as integer constants. Since they are not very useful for the **CIF**, key ids are replaced by their key names. For example, **io_modgrd** is replaced by **modgrd**, **iarr_uvel** by **uvel**, **icon_M2** by **M2**, **ics_phys** by **phys**, **igrd_meteo** by **meteo**, A full listing is given below.

Table 28.1: List of available CIF blocks

block name	purpose	used when
mod_params	model parameters and formats for all model forcing	none
init_params	setup and formats of monitoring files	none
nstgrd_spec	parameters needed for nesting (number of sub-grid open boundary points, ...)	iopt_nests=1
sed_params	parameters for the sediment model	iopt_sed>0
morph_params	parameters for the morphology model	iopt_morph=1
sed_spec	sediment particle properties	iopt_sed>0
bio_params	parameters for the biological model	iopt_biology=1
part_params	parameters for the particle module	iopt_part_model>0
part_spec	specifiers for particle clouds	iopt_part_model> 0, iopt_part_cloud=1
pout_params	parameters for output particle trajectories	iopt_part_model> 0, iopt_part_out=1
dischr_spec	specifiers for the discharge module	iopt_dischr=1
tsr_params	definition of metadata and output grid for time series output	iopt_out_tsers=1
avr_params	definition of metadata and output grid for time averaged output	iopt_out_avrgd=1
anal_freqs	definition of frequencies and formats for harmonic analysis	iopt_out_anal=1
anal_params	definition of metadata and output grid for harmonic output	iopt_out_anal=1

1. Tidal frequencies

```
index_abc = (/icon_M2,icon_S2,icon_K2,icon_O2,icon_K1/)
```

in the routine `usrdef_mod_params` becomes

```
index_abc = M2,S2,K2,O2,K1
```

in the **CIF**. The same convention is used for `index_astro`.

2. The key id of a forcing file (first array index of `modfiles`), e.g. `io_modgrd` becomes `modgrd`. See Section 19.4.
3. The file number of initial/final conditions files (second array index of `modfiles`), e.g. `ics_sed` becomes `sed`. See Section 19.4.
4. The surface grid descriptor (first array index of `surfacegrids`), e.g. `igrd_waves` is replaced by `waves`. See Section 19.5.1.
5. The `f90_name` attribute of the derived type arrays `tsrvars`, ..., e.g., `iarr_sal` becomes `sal`. See section 27.2.1.
6. The state variable index used in the first array index of `indexbio_nst`, e.g. `isv_no3` becomes `no3`. See Section 24.1.2.

28.2.2 Derived type arrays

Derived type arrays are written in the **CIF** for each component separately. The general format is

```
array_name%component_name
```

For a 1-D derived type array, the array values of each component are either written array-wise on one input line or element wise on separate lines for each array element. Components of higher rank (2 or 3) derived types arrays are always written element-wise. For example

```
modfiles%form = 2uvabc,2,1,A
modfiles%status = modgrd,1,1,N
modfiles%filename = modgrd,1,1,grid.dat
surfacegrids%nhtype = igrd_meteo,2
tsrvars(1)%varid = umvel,vmvel,zeta,uvel,vvel,wphys,sal,temp
tsr3d(1)%defined = 1,T
amp2d(1)%form = 1,2,N
```

```
tsrgpars%gridded = T,T
tsrgpars%xlims = 0,100,1,50,150,2
sedpart(3)%iopt_bedeq = 0,0,1,3,0
```

A list of all derived type arrays which can be used in the CIF is given in Table 28.2. The last column indicates whether the arrays are written array-wise (A) or element-wise (E). An exception of the general rules is used for the `xlims`, `ylims`, `zlims`, `tlims` attributes of the `tsrgpars`, `avrgpars`, `analgpars` arrays which are generally written as

```
tsrgpars%xlims =
tsrgpars(1)%xlims(1),tsrgpars(1)%xlims(2),tsrgpars(1)%xlims(3)
...
tsrgpars(nosetstsr)%xlims(1),tsrgpars(nosetstsr)%xlims(2),
tsrgpars(nosetstsr)%xlims(3)
```

28.2.3 CIF flags

A 'F' in a data string, representing a numerical variable, indicates a "flagged" value. This occurs in the following cases

- Parameter `gacc_ref`

```
gacc_ref = F
```

means that the gravity acceleration will be evaluated as function of latitude (see Section 19.3.9).

- Parameter `depmean_cst`

```
depmean_cst = F
```

means that a spatially non-uniform bathymetry is used. Otherwise all water depths are set to the value of `demean_cst`.

- The attribute `tlims` representing the start/end/count time indices appear in several derived type setup arrays. The second element `tlims(2)` can either be an integer value or a flag. The latter means that its value will be set by the model to the number of base time steps `nstep` in the simulation.

```
modfiles%tlims = 2uvnst,1,2,0,F,10
```

Table 28.2: List of all derived type arrays used in the model setup. The last column indicates whether the values of a array component are array-wise (A) or element-wise (E).

block name	array name	format (A/E)
mod_params	modfiles	E
	surfgrid	E
tsr_params	tsrvars	A
	tsr0d	E
	tsr2d	E
	tsr3d	E
	tsrgpars	A
	tsrstatlocs	A
avr_params	avrvars	A
	avr0d	E
	avr2d	E
	avr3d	E
	avrgpars	A
	avrstatlocs	A
anal_params	analvars	A
	res0d	E
	res2d	E
	res3d	E
	amp2d	E
	amp3d	E
	pha2d	E
	pha3d	E
	ell2d	E
	ell3d	E
	analgpars	A
	analstatlocs	A
sed_spec	sedpart	A
pout_params	part1d	E
	outppars	A

- The attribute `rtype` is used by derived arrays of type `FileParams` to determine the type of real data in user-defined output files see Section 27.2.2. Values are `S` for single precision and `D` for double precision output data. An empty data string means that the default value is used (single or double precision depending on whether `COHERENS` is compiled in single or double precision mode).

```
tsr3d%rtype = 1,S
amp2d%rtype = 1,2,D
ell2d%rtype = 2,2,
```

28.3 Example

An easy way to construct a `CIF` is to set the status of the file to '`W`' on the input line `defruns`. The example file¹ below is produced by `COHERENS` and shows the different formats used in the `CIF`. Once a `CIF` is made up by the model, it can easily be edited by the user for other applications.

¹When `COHERENS` writes a `CIF`, the variable names are in upper case format. If the user creates a `CIF` both lower, upper or mixed cases are allowed, except for data strings.

```
!           CIF example
!           ... represent skipped lines

init_params

! cold/wart start
COLD_START = F

! log files
LEVPROCS_INI = 3
LEVPROCS_RUN = 3
INILOG_FILE = plume1A.inilog
RUNLOG_FILE = plume1A.runlog
...
! error files
MAXERRORS = 50
LEVPROCS_ERR = 2
ERRLOG_FILE = plume1A.errlog

! warning files
WARNING = T
WARLOG_FILE = plume1A.warlog

! monitoring
MONLOG = 0
MONLOG_FILE = plume1A.monlog
...
! timer report
LEVTIMER = 3
TIMING_FILE = plume1A.timing
TIMER_FORMAT = 1

! number of processes for each model component
NPROCSCOH = 1
NPROCSWAV = 0

! switches for model coupling
IOPT_PART_MODEL = 0
IOPT_WAVES_MODEL = 0

#
```

```
mod_params

! grid switches
IOPT_GRID_HTYPE = 1
...
! switches for grid interpolation
IOPT_ARRINT_DEPTHS = 1
...
! hydrodynamic switches
IOPT_CURR = 2
...
! density switches
IOPT_DENS = 1
...
! switches for other modules
IOPT_SED = 0
...
! switches for advection
IOPT_ADV_SCAL = 3
...
! switches for horizontal diffusion
IOPT_HDIF_COEF = 0
...
! switches for vertical diffusion
IOPT_VDIF_COEF = 3
...
! switches for the turbulence module
IOPT_TURB_ALG = 1
...
! switches for bottom boundary conditions
IOPT_BSTRES_DRAG = 3
...
! switches for meteo input
IOPT_METEO = 0
...
! switches for surface fluxes
IOPT_SFLUX_PARS = 1
...
! switches for open boundary conditions
IOPT_OBC_ADVRLX = 0
```

```
...
! switches for nesting
IOPT_NESTS = 0
! switches for 1-D applications
...
! tidal switches
IOPT_ASTRO_PARS = 1
...
! surface wave switches
IOPT_WAVES = 0
...
! switches for drying/wetting
IOPT_FLD = 0
...
! switches for the structure module
IOPT_DISCHR = 0
...
! numerical switches
IOPT_COR_IMPL = 1
...
! MPI switches
IOPT_MPI_ABORT = 0
...
! switches for user output
IOPT_OUT_ANAL = 1
...
! netcdf switches
IOPT_CDF_ABORT = 0
...
! other switches
IOPT_RNG_SEED = 1
...
! time parameters
CSTARTDATETIME = 2003/01/03;00:00:00:000
CENDDATETIME = 2003/01/06;00:00:00:000
...
! model grid parameters
NC = 121
NR = 41
NZ = 20
NOSBU = 80
```

```
NOSBV = 120
NRVBU = 0
NRVBV = 1

! bathymetric parameters
DEPMEAN_CST = 20.
DEPMEAN_FLAG = 0.

! domain decomposition
NPROCSX = 1
NPROCSY = 1

! parameters for diffusion
HDIFMOM_CST = 0.

...
! tidal parameters
...
NCONASTRO = 0
NOASTRODAYS = 0
NCONOBC = 1

! tidal constituents
INDEX_OBC = M2

! open boundary conditions and nesting
DISTRLX_OBC = 0.
NONESTSETS = 0

...
! bottom boundary conditions
BDRAGCOEF_CST = 0.

...
! surface boundary conditions
CCHARNO = 0.14E-01
CDSPARS = 0.,0.113E-02,0.,0.

...
! surface wave parameters
WAVE_PENETRATION_BED = 10.

...
! reference values
ATMPRES_REF = 101325.
```

```
...
! optical parameters
OPTATTCOEF1_CST = 10.
OPTATTCOEF2_CST = 0.67E-01

...
! parameters for the structure module
NUMDIS = 0
NUMDRY = 0
NUMTHINU = 0
NUMTHINV = 0
NUMWBARU = 0
NUMWBARV = 0

...
! parameters for drying/wetting
DCRIT_FLD = 0.

...
! parameters for the multigrid scheme
DZETARESID_CONV = 0.1E-06

...
! numerical parameters
CGRAVRATIO = 0.3E-01

...
! grid transformations
B_SH = 0.1
DLAT_REF = 52.
DLON_REF = 0.

...
! turbulence parameters
ALPHA_BLACK = 0.2

...
! user defined output
INTITLE = plume1A
NOFREQSANAL = 1
NOSETSANAL = 1

...
! output formats
FREEFMT = F

...
! attributes of forcing files
MODFILES%STATUS = inicon,phys,1,N
MODFILES%FORM = inicon,phys,1,U
```

```
MODFILES%FILENAME = inicon,phys,1,plumeA.phsfin.bin
MODFILES%TLIMS = inicon,phys,1,0,0,0
MODFILES%HEADER = inicon,phys,1,T
MODFILES%ENDFILE = inicon,phys,1,0
...
MODFILES%STATUS = 2uvobc,1,1,N
MODFILES%FORM = 2uvobc,1,1,A
MODFILES%FILENAME = 2uvobc,1,1,
MODFILES%TLIMS = 2uvobc,1,1,0,0,0
MODFILES%HEADER = 2uvobc,1,1,T
MODFILES%ENDFILE = 2uvobc,1,1,0
MODFILES%STATUS = 2uvobc,2,1,N
MODFILES%FORM = 2uvobc,2,1,N
MODFILES%FILENAME = 2uvobc,2,1,
MODFILES%TLIMS = 2uvobc,2,1,0,0,1
MODFILES%HEADER = 2uvobc,2,1,T
MODFILES%ENDFILE = 2uvobc,2,1,0
...
MODFILES%STATUS = salobc,1,1,N
MODFILES%FORM = salobc,1,1,A
MODFILES%FILENAME = salobc,1,1,
MODFILES%TLIMS = salobc,1,1,0,0,0
MODFILES%HEADER = salobc,1,1,T
MODFILES%ENDFILE = salobc,1,1,0
MODFILES%STATUS = salobc,2,1,N
MODFILES%FORM = salobc,2,1,N
MODFILES%FILENAME = salobc,2,1,
MODFILES%TLIMS = salobc,2,1,0,0,1
MODFILES%HEADER = salobc,2,1,T
MODFILES%ENDFILE = salobc,2,1,0

! attributes of surface grids
SURFACEGRIDS%DELXDAT = model,1000.

...
#
```

tsr_params

```
! variable attributes
TSRVARS%F90_NAME = width,front,umvel,vmvel,zeta,uvel,vvel,wphys,sal
TSRVARS%NRANK = 0,0,2,2,2,3,3,3,3
```

```

TSRVARS%LONG_NAME = Plume width,Plume length,X-component of depth-mean current
TSRVARS%UNITS = km,km,m s-1,m s-1,m,m s-1,m s-1,PSU
TSRVARS%VECTOR_NAME = , ,Depth-mean current,Depth-mean current,,Current, ...

! variables used for each set
IVARSTSR = 1,6,7,8,9
...
! output file attributes
TSR3D%DEFINED = 1,T
TSR3D%FORM = 1,N
TSR3D%FILENAME = 1,plumeA_1.tsout3.nc
TSR3D%RTYPE = 1,S
TSR3D%DEFINED = 2,T
TSR3D%FORM = 2,N
TSR3D%FILENAME = 2,plumeA_2.tsout3.nc
TSR3D%RTYPE = 2,S
TSR3D%DEFINED = 3,T
TSR3D%FORM = 3,N
TSR3D%FILENAME = 3,plumeA_3.tsout3.nc
TSR3D%RTYPE = 3,S
...
! output grid and time parameters
TSRGPARS%GRIDDED = T,T,T,T
TSRGPARS%NOSTATS = 0,0,0,0
TSRGPARS%PACKING = F,F,F,F
...
TSRGPARS%TIME_FORMAT = 3,3,3,3
TSRGPARS%VCOORD = 2,2,2,2
TSRGPARS%XLIMS = 1,120,1,30,30,1,1,120,1,30,30,1
...
TSRGPARS%TLIMS = 0,864,36,0,864,36,0,864,36,0,864,1
#
anal_freqs

! frequencies used for harmonic analysis
INDEX_ANAL = M2
...
#
anal_params

```

```
! variable attributes
ANALVARS%F90_NAME = umvel,vmvel,zeta,uvel,vvel,wphys,sal
...
! variables used for each set
IVARSANAL = 1,1,2,3,4,5,6,7

! tidal ellipse parameters
IVARSELL = 1,1,10
IVECELL2D = 1,1,2
IVECELL3D = 1,4,5

! output file attributes
RES2D%DEFINED = 1,T
RES2D%FORM = 1,N
RES2D%FILENAME = 1,plumeA_1.resid2.nc
RES2D%RTYPE = 1,S
...
AMP2D%DEFINED = 1,1,T
AMP2D%FORM = 1,1,N
AMP2D%FILENAME = 1,1,plumeA_1.M2amplt2.nc
AMP2D%RTYPE = 1,1,S
...
! output grid and time parameters
ANALGPARS%GRIDDED = T
...
#
sed_params

! switches for the sediment module
IOPT_SED_BETA = 1
...
! parameters for the sediment module
ALPHA_VR = 2.19
...
#
sed_spec

! sediment particle properties
```

```

SEDPART%IOPT_BBC_EQ = 1,1,1,1,1,1
..
! sediment switches per fraction
SEDPART%BSTRES_CR_CST = 0.,0.,0.,0.,0.,0.

..
#
part_params

! particle switches
IOPT_PART_CLOUD = 1
..
! particle parameters
CDRIFT_FAC = 1.
NOCLOUDS = 3
NOLABELS = 3
NOPART = 9000
..
#
part_spec

! particle properties
DIAMCONC = 0.,0.,0.
..
! particle cloud parameters
CLOUD%KDISTYPE = 1,1,1
..
#
pout_params

! output file attributes
PART1D%DEFINED = 1,T
..
! particle output attributes
OUTPPARS%AGING = T
..
#

```

Example 28.1: Generic example of a CIF.

Chapter 29

Layout of standard forcing files

29.1 Introduction

COHERENS allows to use three types of standard forcing files:

1. COHERENS standard forcing files
 - To create this type of forcing file a first run must be made using the `usrdef` method. The file can then be used for a subsequent run.
 - The variables inside the file are those required by the settings of the first run.
 - Although this is the easiest way to create a forcing file, the use is restricted to simulations with the same forcing variables.
 - The file can be created in ASCII (A), unformatted binary (U) and netCDF (N) format.
2. Non-COHERENS standard files created by the user with header information.
 - The file must be created by the user by pre-processing. Instructions are given in Section 29.2
 - Either A, U or N format can be used.
 - In the non-netCDF case, the file must contain a header containing metadata information about the variables in the file and a few general parameters. In the netCDF case, the header is automatically created by netCDF procedures.
 - The names of the forcing variables (given in the header) must match the names used in COHERENS. This checked by the program.

- Contrary to a standard COHERENS file, the data file may contain more or less data variables as required by the setup. Variables not needed for the specific run will be skipped while variables required by the setup but not contained in the file will be set to their default values. In each case, a warning message will be issued.
 - Order of the variables is irrelevant except for the `time` coordinate and `subname` variable (see below).
 - The prime advantage of non-COHERENS standard files is that the same forcing file can be used for different sorts of simulations. For example, a meteorological data file can be made up with a complete set of meteo variables and then e.g. applied for simulations without temperature/salinity flux.
3. Non-COHERENS standard files created by the user without header information.
- This type is only available in `A` or `U` format.
 - Contrary to the previous case the data variables, the ordering and reading of the data inside the file are assumed to be exactly the same as the ones used in a COHERENS standard forcing file and must be read in the correct order.
 - No checking is performed. A read error is issued if data are missing, stored in an incorrect order or have the wrong shape or type. It is up to the user to find the exact source of the error.
 - The no-header type has only be implemented for users preferring an easy format, but has the disadvantage that errors are easily made.

As discussed in Section 19.4 the attributes of a forcing files are defined in the 3-D derived type array `modfiles`. A generic presentation used in this chapter for an element of this array takes the form

```
modfiles(iddesc,ifil,iotype)
```

where `iddesc` is the forcing file key id of the form `io_*` (e.g. `inicon` for an initial condition file), `ifil` the file number and `iotype` equals 1 for an input and 2 for an output file.

A standard COHERENS forcing file will be written if

```
modfiles(iddesc,ifil,2)='W'
```

while a standard (COHERENS or non-COHERENS) will be used for input if

```
modfiles(iddesc,ifil,2)='R'
```

The rules for constructing a non-COHERENS standard forcing file will be given in the Section 29.2. Methods in the case that the file contains multi-variable arrays are given in Section 29.3.

29.2 Structure of a non-COHERENS standard forcing file

29.2.1 ASCII or unformatted binary forcing files

The structure of an ASCII (or unformatted binary) forcing file with header consists of two parts:

1. A header section where information is provided about the forcing variables (number of coordinates and variables, name, data type, shape, read format). The section ends by a line with a '#' in the first column. Lines starting with the comment character '!' are ignored. A line in the header has the following syntax

```
name : value
```

or

```
name : value 1, value 2, ..., value n
```

where **name** is the name of an output variable and **value** or **value 1, ..., value n** the value(s) of the output variable.

2. A data section with the actual data. Assuming **N** variables, the input data are structured as follows:

```
comment line
data for variable 1
comment line
data for variable 2
...
comment line
data for variable N
```

The comment line(s) must be included. The content is arbitrary and may even be empty. If the first variable is the `time` coordinate variable, the sequence is then repeated for the next time. Note that the time data must be provided in increasing chronological order.

The structure of a header section is shown by the following example of a forcing file defining a model grid

```
ncoordinates      : 0
nvariables        : 6
varid             : 1
datatype          : real
rank              : 1
shape              : 31
f90_name          : gsigcoordatw
format            : (50G16.7)
varid             : 2
datatype          : real
rank              : 2
shape              : 108,50
f90_name          : depmeanglb
format            : (50G16.7)
varid             : 3
datatype          : integer
rank              : 1
shape              : 63
f90_name          : iobu
format            : (50I11)
varid             : 4
datatype          : integer
rank              : 1
shape              : 63
f90_name          : jobu
format            : (50I11)
varid             : 5
dataType          : integer
rank              : 1
shape              : 109
f90_name          : iobv
format            : (50I11)
varid             : 6
datatype          : integer
```

29.2. STRUCTURE OF A NON-COHERENS STANDARD FORCING FILE855

```
rank          : 1
shape         : 109
f90_name      : jobv
long_name     : Global Y-index of V-open boundaries
format        : (50I11)
#
#
```

where

ncoordinates Number of coordinate variables. Value is 1 or 0 depending on whether the forcing contains a time coordinate or not. If the line is omitted, its value is 0.

nvariables Number of data variables (excluding the **time** variable if present).

Attributes of the data variables (including the time variable if present) are provided on the remaining lines (starting from line 3):

varid Variable id. The number represents the order of storage of the variable in the forcing file. They must be given in increasing order from 1 to **ncoordinates+nvariables**. The first variable must be the **time** coordinate (if present).

datatype Data type. Allowed values are **character**, **real**, **double**, **integer**, **logical**.

rank Rank of the data array. For a character string its value is 1. For character array strings the rank is 1 plus the rank of the array.

shape Shape of the data array. For a character string this is the length of the string. For a character string array this is the length of the string followed by the shape of the array.

f90_name Name of the variable which must match its **FORTRAN** name used in the code.

format Format specification needed for reading the data in the ASCII case. If set to **free**, the data are read in free (*) format.

The following conventions are made

- Names and values are case-insensitive.
- The position of the ':' character between the name and the value(s) is arbitrary.
- Blanks and blank names are ignored.

- The order of the lines before the first `varid` statement and between two subsequent `varid` statements is arbitrary.
- The `varid` values must be given in increasing order.
- The `time` coordinate, if present, must be declared as the first data variable.

29.2.2 netCDF forcing files

The header (or metadata) section of a netCDF forcing file is automatically created when the file is written by the user or by COHERENS. The metadata section of the example below is obtained using the netCDF `ncdump` utility:

```
ncdump -h best_meteo.nc
```

giving

```
netcdf best_meteo
dimensions:
lentime = 23 ;
time = UNLIMITED ; // (8759 currently)
lon = 281 ;
lat = 217 ;
variables:
char time(time, lentime) ;
float uwindatc(time, lat, lon) ;
float vwindatc(time, lat, lon) ;
float atmpres(time, lat, lon) ;
float airtemp(time, lat, lon) ;
float relhum(time, lat, lon) ;
float cloud_cover(time, lat, lon) ;
float precipitation(time, lat, lon) ;
```

The user may add additional information by adding global and variable attributes. This is the case when the file is created by the model as a COHERENS standard file. The names of the dimensions (except for the `time` dimension) are arbitrary. The variable names, on the other hand, must match their FORTRAN name used in COHERENS.

For users not familiar with netCDF an example FORTRAN code for making up this forcing file is shown below.

29.2. STRUCTURE OF A NON-COHERENS STANDARD FORCING FILE857

```
PROGRAM best_meteo

USE netcdf

IMPLICIT NONE

INTEGER, PARAMETER :: lentime = 23, maxrecs = 8759, nx = 281, ny = 217
CHARACTER (LEN=lentime) :: ciodate
INTEGER :: airtempid, atmpresid, cloudid, ierr, ncid, precipid, &
           & relhumid, timeid, timerec, uwindid, vwindid
INTEGER, DIMENSION(2) :: timedim, xdim
REAL, DIMENSION(nx,ny) :: airtemp, atmpres, cloud_cover, precipitation, &
                           & relhum, uwindatc, vwindatc

!
!1. Write metadata
!-----
!
!1.1 Open data file
!-----
!

ierr = NF90_create('best_meteo.nc',NF90_CLOBBER,ncid)

!
!1.2. Define dimensions
!-----
!
ierr = NF90_def_dim(ncid,'lentime',23,timedim(1))
ierr = NF90_def_dim(ncid,'time',NF90_unlimited,timedim(2))
ierr = NF90_def_dim(ncid,'lon',nx,xdim(1))
ierr = NF90_def_dim(ncid,'lat',ny,xdim(2))

!
!1.3 Define variables
!-----
!
!---time variable is first defined
ierr = NF90_def_var(ncid,'time',NF90_char,timedim,timeid)
ierr = NF90_def_var(ncid,'uwindatc',NF90_real,&
```

```

                & (/xdim(1),xdim(2),timedim(2)/),uwindid)
ierr = NF90_def_var(ncid,'vwindatc',NF90_real,&
                    & (/xdim(1),xdim(2),timedim(2)/),vwindid)
ierr = NF90_def_var(ncid,'atmpres',NF90_real,&
                    & (/xdim(1),xdim(2),timedim(2)/),atmpresid)
ierr = NF90_def_var(ncid,'airtemp',NF90_real,&
                    & (/xdim(1),xdim(2),timedim(2)/),airtempid)
ierr = NF90_def_var(ncid,'relhum',NF90_real,&
                    & (/xdim(1),xdim(2),timedim(2)/),relhumid)
ierr = NF90_def_var(ncid,'cloud_cover',NF90_real,&
                    & (/xdim(1),xdim(2),timedim(2)/),cloudid)
ierr = NF90_def_var(ncid,'precipitation',NF90_real,&
                    & (/xdim(1),xdim(2),timedim(2)/),precipid)

!
!1.4 End define mode
!-----
!

ierr = NF90_enddef(ncid)

!
!2. Read/write data
!-----
!

timerec = 0

DO WHILE (timerec.LT.maxrecs)

!
!2.1 Read data from data source files
!-----
!

! ---time variable
ciodatetime = ...

! ---meteo data
uwindatc = ...
...
precipitation = ...

```

```

!
!2.2 Write data to the forcing file
!-----
!
!   ---time
ierr = NF90_put_var(ncid,timeid,ciodatetime,start=(/1,timerec/),&
& count=(/lentime,1/))
!
!   ---meteo data
ierr = NF90_put_var(ncid,uwindid,uwindatc,start=(/1,1,timerec/),&
& count=(/nx,ny,1/))
ierr = NF90_put_var(ncid,vwindid,vwindatc,start=(/1,1,timerec/),&
& count=(/nx,ny,1/))
ierr = NF90_put_var(ncid,atmpresid,atmpres,start=(/1,1,timerec/),&
& count=(/nx,ny,1/))
ierr = NF90_put_var(ncid,airtempid,airtemp,start=(/1,1,timerec/),&
& count=(/nx,ny,1/))
ierr = NF90_put_var(ncid,relhumid,relhum,start=(/1,1,timerec/),&
& count=(/nx,ny,1/))
ierr = NF90_put_var(ncid,cloudid,cloud_cover,start=(/1,1,timerec/),&
& count=(/nx,ny,1/))
ierr = NF90_put_var(ncid,precipid,precipitation,start=(/1,1,timerec/),&
& count=(/nx,ny,1/))

timerec = timerec + 1

ENDDO

!
!3 Close forcing file
!-----
!
ierr = NF90_close(ncid)

END PROGRAM best_meteo

```

29.3 Structure of forcing files with multi-variable arrays

Sub-variables in forcing files are used in two contexts:

1. COHERENS makes use of so called “multi-variable” arrays where the last index refers to a specific “sub-variable”. This is the case for variables used e.g. in the sediment, morphology, contaminant and (to some extent) biology modules. The last index denotes the fraction or, more generally the “variable” number. The size of the last dimension represents the number of sub-variables `nosubvars`. For example, initial data only need (or may) be defined for sediment fractions in suspended mode. In that case the forcing file only needs to contain these suspended fractions. The same procedure applies for forcing files containing profiles used at open boundaries and for nesting. The sub-variables are stored in the file as one single data array and read by “index” recognition. This is further discussed in Section 29.3.1.
2. Surface meteorological and wave input are defined from forcing data defined on an external surface grid. In case that the surface grid reduces to a single point (i.e. the `nhype` attribute of the surface grid is zero), the data are supplied as one single vector. The names of the data variables stored in the vector are defined in the data variable `subnames`. This method is called data recognition by “name”. Further discussion is given in Section 29.3.2

29.3.1 Recognition by index

A sub-variable in a multi-variable array is identified by an index value stored in the vector `subindex` of size `nosubvars`. This is the case for sediment concentrations where the last array index represents the fraction index or for some biological state variables array where the last index represents the state variable (key id) number. This means that two additional variable attributes, i.e. `nosubvars` and `subindex` must be provided in the header information for each multi-variable.

The example below shows the header of an ASCII forcing file with initial conditions for sediments. The model setup is made with 4 sediment fractions of which 2 are in suspended and two in bed load mode.

```
ncoordinates    : 1
nvariables      : 2
varid          : 1
```

```

datatype      : character
rank         : 1
shape        : 23
f90_name     : time
format       : (500(A,1X))
varid        : 2
datatype      : real
rank         : 4
shape        : 349,446,20,2
f90_name     : cvol
format       : (50G16.7)
nosubvars    : 2
subindex     : 1,3
varid        : 3
dataType     : real
rank         : 4
shape        : 349,446,1,4
f90_name     : bed_fraction
format       : (50G16.7)
nosubvars    : 4
subindex     : 1,2,3,4
#
time
2013/02/01;00:00:00:000
cvol
...
bed_fraction
...

```

There are now two new variable attributes

nosubnames The number of sub-variables.

subindex The indices of the sub-variables stored in the multi-variable array. The size of the **subindex** vector attribute must be equal to **nosubvars**.

Fractions 1 and 3 are in suspended mode and stored in the data array **cvol** representing the initial volume concentrations. On the other hand, the bed fraction needs to be defined for each sediment fraction (array **bed_fraction**).

The next example is for the same setup but now using a **netCDF** forcing file.

```

netcdf sedics.nc
dimensions:
lentime = 23 ;
time = UNLIMITED ; // (1 currently)
nc-1 = 349 ;
nr-1 = 446 ;
nz = 20 ;
nb = 1 ;
nf = 4 ;
nfsus = 2 ;
variables:
char time(time, lentime) ;
float cvol(time, nfsus, nz, nr-1, nc-1) ;
cvol:subindex = 1, 3 ;
float bed_fraction(time, nf, nb, nr-1, nc-1) ;
bed_fraction:subindex = 1, 2, 3, 4 ;

```

The array dimensions `nfsus` and `nf` represent the number of suspended and total fractions. The `nosubvars` attribute does not need to be defined here since it equals the size of the `subindex` attribute. Note again that the dimension names are arbitrary except for the time dimension.

29.3.2 Recognition by name

When a surface data grid reduces to a single point, the forcing data are stored into a single vector whose size is given by the number of surface forcing variables. The procedure is used for

- Meteorological data defined at a single point when
`surfacegrids(igrd_meteo,1)%nhtype = 0.`
- Surface wave data defined at a single point when
`surfacegrids(igrd_waves,1)%nhtype = 0.`

The example shows the header of a meteorological forcing file and how the data are stored.

```

ncoordinates : 1
nvariables : 1
nsubnames : 5
varid : 1
datatype : character

```

```

rank          : 2
Shape         : 11,5
f90_name      : subname
format        : (500(A,1X))
varid         : 2
datatype      : character
rank          : 1
shape         : 23
f90_name      : time
format        : (500(A,1X))
varid         : 3
datatype      : real
rank          : 1
Shape         : 5
f90_name      : meteodata
format        : (50G16.7)
#
subname
uwindatc    vwindatc    airtemp     relhum      cloud_cover
time
1989/01/03;00:00:00:000
meteodata
  4.060000      3.780000      8.080000      0.7613000     0.5000000
time
1989/01/03;03:00:00:000
meteodata
...

```

- The header contains the additional attribute `nosubnames` representing the number of sub-variables.
- The file contains three forcing variables. The first is the character vector array `subname` with the **FORTRAN** names of the forcing data, the second the time variable `time` and the third the vector `meteodata` with the surface surface data. They must be declared in this order.
- The array `subname` must first appear in the data section and (obviously) only be given at the initial time. The calendar date and time followed by the data values are supplied in chronological order.

The next example show the header of the **netCDF** file with the same data

```
netcdf csmet
dimensions:
lenname = 11 ;
nosubnames = 5 ;
lentime = 23 ;
time = UNLIMITED ; // (2416 currently)
variables:
char subname(nosubnames, lenname) ;
char time(time, lentime) ;
float meteodata(time, nosubnames) ;

data:

subname =
"uwindatc" ,
"vwindatc" ,
"airtemp" ,
"relhum" ,
"cloud_cover" ,
" " ;

time = 

meteodata =
. . .
```

Note again that `subname` is the first variable, `time` the second one and `meteodata` the third one. Note that the dimension names `nosubnames` and `time` must be used.

Chapter 30

Contents of standard forcing files

In this chapter the contents of all forcing files are discussed. For each forcing file a listing is given of all data arrays which can be included for a particular type (represented by the forcing file key id `io_*` and file number `ifil`). The order of the listings is the ones used in a standard COHERENS forcing file which should be used also in case of a standard non-COHERENS file without header. For non-COHERENS standard files with header the order is arbitrary except for the time coordinate and, in the case of multi-variable arrays with name recognition, the sub-variable array `subname`.

As discussed earlier a standard COHERENS forcing file will be created if

```
modfiles(iddesc,ifil,2)%status = 'W'
```

where `iddesc` is the forcing key id and `ifil` the file number.

The conditions for which a forcing array needs to be provided are given. As discussed earlier, a forcing array included in the file, but not needed for the simulation, will be skipped, while a forcing array not included in the data file, but needed for the simulation, will be replaced by its default. A warning message will be issued in each case. If an array is defined on the model grid, its shape is the one given in Table 4.1 for the “physical” grid which depends on the type of node ('C', 'U', 'V', 'W', 'UV').

Important to note is that some standard COHERENS forcing files may contain data which cannot be produced by the user for the non-standard case. This is further discussed below for each particular case.

A description is given for each forcing array including its unit. The descriptions closely follow the ones given in Chapters 19–26 but repeated here for clarity.

30.1 Model grid

purpose : model grid and bathymetry
 time coordinate : no
 key id : io_modgrd
 file number : 1
 sub-variables : no

horizontal model grid

```

                                ! used when
REAL, DIMENSION(nc) :: gdelxglb      ! iopt_grid_htype=2
REAL, DIMENSION(nr) :: gdelyglb      ! iopt_grid_htype=2
REAL, DIMENSION(nc, nr) :: gxcoordglb ! iopt_grid_htype=3
REAL, DIMENSION(nc, nr) :: gycoordglb ! iopt_grid_htype=3

```

gdelxglb Grid spacings in the X-direction at C-nodes in case that a non-uniform rectangular grid is selected (`iopt_grid_htype=2`) [m or degrees longitude].
gdelyglb Grid spacings in the Y-direction at C-nodes in case that a non-uniform rectangular grid is selected (`iopt_grid_htype=2`) [m or degrees latitude].
gxcoordglb X-coordinates of the model grid at corner nodes [m or degrees longitude, positive East]. Must be provided in case a curvilinear grid is selected (`iopt_grid_htype=3`).
gycoordglb Y-coordinates of the model grid at corner nodes [m or degrees latitude, positive North]. Must be provided in case a curvilinear grid is selected (`iopt_grid_htype=3`).

vertical model grid

```

                                ! used when
REAL, DIMENSION(nz+1) :: gsigcoordatw      ! iopt_grid_vtype=2,
                                                ! iopt_grid_vtype_transf=20
REAL, DIMENSION(nc-1, nr-1, nz+1) :: gscoordglb ! iopt_grid_vtype=3
                                                ! iopt_grid_vtype_transf=30

```

gsigcoordatw Sigma coordinates at W-nodes in case that a horizontally uniform, but vertically non-uniform grid is selected. If `iopt_grid_vtype_transf` equals 21, 22 or 23, the array is not needed since the vertical grid is constructed by the

program using one of the vertical coordinates transformations described in Section 4.1.4.1.

gscoordglb Sigma grid at W-nodes in case that a horizontally/vertically non-uniform grid is selected. If `iopt_grid_vtype_transf` equals 31 or 32, the array is not needed here since the vertical grid is constructed by the program using one of the coordinates transformations described in Section 4.1.4.2.

bathymetry

`REAL, DIMENSION(nc-1, nr-1) :: depmeanglb`

depmeanglb Mean water depths at C-nodes [m]. If not supplied, a uniform bathymetry is assumed with water depths equal to the parameter `depmean_cst`. Permanent land cells are marked with the flag `depmean_flag`. If the drying/wetting scheme is activated, negative values can be used for locations above the mean sea level (e.g. tidal flats).

open boundary locations

`REAL, DIMENSION(nobu) :: iobu`
`REAL, DIMENSION(nobu) :: jobu`
`REAL, DIMENSION(nobv) :: iobv`
`REAL, DIMENSION(nobv) :: jobv`

iobu (Global) X-index of the West/East open boundary points at U-nodes.

jobu (Global) Y-index of the West/East open boundary points at U-nodes.

iobv (Global) X-index of the South/North open boundary points at V-nodes.

jobv (Global) Y-index of the South/North open boundary points at V-nodes.

For further remarks see Section 20.1.

30.2 Domain decomposition

purpose : domain decomposition
 time coordinate : no
 key id : io_mppmod
 file number : 1
 sub-variables : no

```

INTEGER, DIMENSION(nprocs) :: mgvars%nc1procs
INTEGER, DIMENSION(nprocs) :: mgvars%nc2procs
INTEGER, DIMENSION(nprocs) :: mgvars%nr1procs
INTEGER, DIMENSION(nprocs) :: mgvars%nr2procs
  
```

mgvars%nc1procs Global X-index of the most western cells of the process domain.

mgvars%nc2procs Global X-index of the most eastern cells of the process domain.

mgvars%nr1procs Global Y-index of the most southern cells of the process domain.

mgvars%nr2procs Global Y-index of the most northern cells of the process domain.

The arrays are needed when the program is set up in parallel mode ($nprocs > 1$) and $iopt_MPI_partit=2$. It must be remarked here that it is not straightforward to set up a non-regular domain decomposition. In particular, when $nomgelevels > 1$, care must be taken that the domain boundaries of the coarser (higher level) sub-grids are aligned with domain boundaries of the finer (lower level) grids. In view of this constraint, the procedure for constructing domain decompositions at all grid levels is not evident. It is therefore recommended to define the decomposition using the **gridproc** utility program, which provides the decomposition arrays in standard format. For details see Section 20.2.

30.3 Initial and final conditions

purpose : initial and final conditions
 time coordinate : yes
 key id : io_inicon, io_fincon
 file number : ics_phys, ics_sed, ics_morph, ics_bio, ics_part
 sub-variables : yes/no

Most of the forcing data used in an initial/final conditions file represent model variables which are updated in time at each time step. This means that its value at the previous time must be available.

Key id `io_incon` represents initial data which are read at the start time, `io_fincon` represents “final condition” data written by the model in standard COHERENS format to provide initial data for a subsequent run. The `status` attribute must be set to 'R' for an initial and 'W' for a final condition file.

Several initial data are determined from inside the program and can therefore only be obtained through a final condition file from a previous run.

The file number key id refers to different program modules (physics, sediments, morphology, biology, particle tracers). They will be separately discussed below.

Important to note is that arrays defined on the model grid are provided on the global grid in contrast to the `usrdef` method where they need to be given on the local grid.

Unless specified otherwise, default values are zero.

30.3.1 Physical module

purpose : initial and final conditions for the physical module
time coordinate : yes
key id : io_inicon, io_fincon
file number : ics_phys
sub-variables : no

time First forcing variable is the time coordinate

CHARACTER (LEN=lentime) :: time

currents and surface elevation

Table 30.1: Values of the switches `iop_mode_2D` and `iop_mode_3D`.

<code>iop_curr</code>	<code>iop_grid_nodim</code>	<code>iop_hydro_impl</code>	<code>iop_mode_2D</code>	<code>iop_mode_3D</code>
0	1	0	0	0
0	2	0	0	0
0	3	0	0	0
0	1	1	0	0
0	2	1	0	0
0	3	1	0	0
1	1	0	1	1
1	2	0	1	1
1	3	0	1	1
1	1	1	1	1
1	2	1	1	1
1	3	1	1	1
2	1	0	0	2
2	2	0	2	0
2	3	0	2	2
2	1	1	0	2
2	2	1	2	0
2	3	1	1	2

where `iop_mode_2D` and `iop_mode_3D` are internal switches which depend on the user-defined switches `iop_curr` (type of hydrodynamics), `iop_grid_nodim` (dimension of model grid) and `iop_hydro_impl` (explicit or implicit time integration). Details are given in Table 30.1.

`udvel` Depth-integrated current in the X-direction [m²/s].

`vdvel` Depth-integrated current in the Y-direction [m²/s].

`zeta` Surface elevation [m].

`uvel` Current *u* in the X-direction [m/s].

`vvel` Current *v* in the Y-direction [m/s].

`wvel` Transformed vertical current *ω* [m/s]. Defined only for 3-D applications.

Except for special applications or experimental case studies, these forcings should be obtained from a final condition file. The usual procedure is to split an initial simulation in two runs. In the first one, a “spinup” start is taken which means that elevations and currents are set to their default (zero values). The simulation period is selected such that a

“steady” regime is established at the end of the run. For example, in the case of a tidal regime, one lets the tides enter the domain through the open boundaries during the cold run. A few tidal cycles are usually sufficient to obtain a regular tidal regime. At the end of the cold run a final condition file is written providing all forcing data for a subsequent (actual) run.

density

```
! used when
REAL, DIMENSION(nc-1, nr-1, nz) :: temp ! iopt_temp=1,2
REAL, DIMENSION(nc-1, nr-1, nz) :: sal ! iopt_sal=1,2
```

temp Temperature T [^0C]. Negative (freezing) values are not allowed in the current implementation. Default is **temp_ref**.

sal Salinity S [PSU]. Default is **sal_ref**.

turbulence

```
! used when iopt_vdif_coef=3 and
REAL, DIMENSION(nc-1, nr-1, nz+1) :: tke ! iopt_turb_ntrans=1,2
REAL, DIMENSION(nc-1, nr-1, nz+1) :: zlmix ! iopt_turb_ntrans=2,
                                              ! iopt_turb_param=1
REAL, DIMENSION(nc-1, nr-1, nz+1) :: dissip ! iopt_turb_ntrans=2,
                                              ! iopt_turb_param=2
```

tke Turbulent kinetic energy k [J/kg].

zlmix Mixing length l [m].

dissip Dissipation of turbulent energy ε [W/kg].

These forcing data should not be produced by the user (except for very exceptional case studies) but either by a final conditions file after a cold start or by keeping the default values given in Section 5.11 which are mostly sufficient. See above for further details.

arrays for bottom stress

```
! used when
REAL, DIMENSION(nc-1, nr-1) :: bdragcoefatc ! iopt_bstres_drag=2
REAL, DIMENSION(nc-1, nr-1) :: zroughatc ! iopt_bstres_drag=4,5
```

bdragcoefatc Bottom drag coefficient C_{db} [-].

zroughatc Bottom roughness length z_0 [m].

tidal arrays

! used when
REAL, DIMENSION(nconobc) :: phase.obc ! iopt_astro_pars=1

phase.obc Astronomical phases of the tidal constituents at open boundaries [rad]. Zero by default.

Should only be used for idealised case studies.

weirs and barriers

! used when iopt_weibar=1
REAL, DIMENSION(numwbaru) :: wbarelossu
REAL, DIMENSION(numwbarv) :: wbarelossv

wbarelossu Energy loss sink term of weirs and barriers at U-nodes [s^{-1}].

wbarelossv Energy loss sink term of weirs and barriers at V-nodes [s^{-1}].

Should only be provided by a final condition file.

open boundary arrays

A number of work space arrays at open boundaries need to be initialised since the open boundary conditions at specific nodes require the values of these arrays from the previous time step. They can only be obtained from a final condition file after a first run.

! use depends on type of open boundary conditions and
REAL, DIMENSION(nobu,nz,0:2,1) :: obcsalatu ! iopt_abc_sal=1
REAL, DIMENSION(nobv,nz,0:2,1) :: obcsalatv ! iopt_abc_sal=1
REAL, DIMENSION(nobu,nz,0:2,1) :: obctmpatu ! iopt_abc_temp=1
REAL, DIMENSION(nobv,nz,0:2,1) :: obctmpatv ! iopt_abc_temp=1
REAL, DIMENSION(nobu,2) :: obc2uvatu ! iopt_abc_2D=1
REAL, DIMENSION(nobv,2) :: obc2uvatv ! iopt_abc_2D=1
REAL, DIMENSION(nobu,nz,2) :: obc3uvatu ! iopt_abc_3D=1
REAL, DIMENSION(nobv,nz,2) :: obc3uvatv ! iopt_abc_3D=1

30.3.2 Multi-fraction sediment module

purpose	:	initial and final conditions for the multi-fraction sediment module
time coordinate	:	yes
key id	:	io_inicon, io_fincon
file number	:	ics_sed
sub-variables	:	yes

Note that the multi-fraction model for suspended transport is only enabled when `iopt_sed` is set to 1.

time First forcing variable is the **time** coordinate

```
CHARACTER (LEN=lentime) :: time
```

concentrations and fractions

```
! used when iopt_sed=1
REAL, DIMENSION(nc-1, nr-1, nz, nosubvars) :: cvol ! fractions in suspended mode
REAL, DIMENSION(nc-1, nr-1, nb, nf) :: bed_fraction ! nf>1
```

where `nosubvars` is the number of sub-variables which must be between 1 and the number of fractions in suspended mode (`nfsus`), `nf` the (total) number of fractions used in the simulation and `nb` the number of bed layers (equal to 1 if the morphological module is not activated).

cvol The volumetric sediment concentration for each sediment fraction [m^3/m^3]. The number of sub-variables `nosubvars` must be between 1 and the number of suspended fractions `nfsus`. The `nosubvars` and `subindex` must be provided (see Section 29.3.1).

bed_fraction The relative amount of material in the sediment bed layers per sediment fraction. Note that the sum over all fractions must be equal to 1 or zero (only allowed when the morphology module is activated). The latter case corresponds to an empty bed sediment layer. If the array is not provided, uniform default values equal to $1/nf$ are taken. The `nosubvars` and `subindex` attributes do not need to be defined since the forcing data must be provided for all fractions.

Since it is not straightforward for the user to define initial concentrations, it is recommended to split the simulation in two runs. A useful procedure then is to take zero (default) values at the start of an initial (“spinup”) run. After some time a quasi-equilibrium stage between erosion and deposition may be gradually established. At the end of the run the data are written to a final condition file providing the initial data for the second (actual) run.

open boundary arrays

In analogy with the physics a number of work space arrays at open boundaries may need to be initialised when the open boundary conditions at specific nodes require the values of these arrays from the previous time step. They can only be obtained from a final condition file written by COHERENS.

```
! use depends on type of open boundary conditions and
REAL, DIMENSION(nobu,nz,0:2,nosubvars) :: obcsedatu ! iopt_obi_sed=1
REAL, DIMENSION(nobv,nz,0:2,nosubvars) :: obcsedatv ! iopt_obi_sed=1
```

Remarks

- The variable attributes **nosubvars** with the number of sub-variables and **subindex** with the indices of the corresponding fractions must be defined for those forcing arrays with a sub-variable last dimension.
- By default, all arrays are initialised to zero, except for the bed fractions which are set to the uniform value $1/nf$.

30.3.3 Flocculation module

purpose	:	initial and final conditions for the flocculation module
time coordinate	:	yes
key id	:	io_inicon, io_fincon
file number	:	ics_sed
sub-variables	:	yes

The flocculation model is enabled when **iopt_sed** is set to 2.

time First forcing variable is the **time** coordinate

```
CHARACTER (LEN=lentime) :: time
```

concentrations

```

! always used
REAL, DIMENSION(nc-1, nr-1, nz) :: floc_P
REAL, DIMENSION(nc-1, nr-1, nz) :: floc_F
REAL, DIMENSION(nc-1, nr-1, nz) :: floc_T

```

floc_P Number concentration of flocculi (N_p) [m^{-3}].

floc_F Number concentration of flocs (N_F) [m^{-3}].

floc_T Number concentration of flocculi bound in flocs (N_T) [m^{-3}].

Initial concentrations can be obtained using the procedure outlined above for the multi-fraction model.

open boundary arrays

In analogy with the physics a number of work space arrays at open boundaries may need to be initialised when the open boundary conditions at specific nodes require the values of these arrays from the previous time step. They can only be obtained from a final condition file written by COHERENS.

```

! use depends on type of open boundary conditions and
REAL, DIMENSION(nbu, nz, 0:2, nosubvars) :: obccnumpatu ! iopt_abc_sed=1
REAL, DIMENSION(nbv, nz, 0:2, nosubvars) :: obccnumpatv ! iopt_abc_sed=1

```

The meaning of the arrays **obccnumpatu**, **obccnumpatv** is the same as the scalar arrays defined above for the physics (e.g. **obctmpatu**, **obctmpatv**), except that the arrays are defined for the three fractions used in the flocculation module.

30.3.4 Morphological module

purpose	: initial and final conditions for the morphological module
time coordinate	: yes
key id	: io_inicon , io_fincon
file number	: ics_morph
sub-variables	: no

time First forcing variable is the **time** coordinate

```
CHARACTER (LEN=lentime) :: time
```

forcing arrays

```

! used when
REAL, DIMENSION(nc-1, nr-1, nb) :: bed_layer_thickness
                                         ! iopt_morph_fixed_layer=1
REAL, DIMENSION(nc-1, nr-1, nf) :: bed_update_acc ! always

bed_layer_thickness Thicknesses of the sediment bed layers [m].
bed_update_acc    The change in the bed level elevation accumulated
                   over time [m].

```

work space arrays

A series of arrays are needed when an Adams-Bashforth time is used.

```

! used when iopt_time_init=1
REAL, DIMENSION(nc-1, nr-1, nf) :: depflux_old1 ! iopt_morph_time_int=2,3
REAL, DIMENSION(nc-1, nr-1, nf) :: eroflux_old1 ! iopt_morph_time_int=2,3
REAL, DIMENSION(nc-1, nr-1, nf) :: depflux_old2 ! iopt_morph_time_int=3
REAL, DIMENSION(nc-1, nr-1, nf) :: eroflux_old2 ! iopt_morph_time_int=3

depflux_old1      Source term  $S_+$  in the bed elevation equation at the
                   previous time step [m].
depflux_old2      Source term  $S_+$  in the bed elevation equation at the
                   second previous time step [m].
eroflux_old1      Sink term  $S_-$  in the bed elevation equation at the
                   previous time step [m].
eroflux_old2      Sink term  $S_-$  in the bed elevation equation at the
                   second previous time step [m].

```

The meaning of the S_+ and S_- source/sink terms is explained in Section 8.4.3. These arrays are needed for updating the bed elevation in time and can only be defined using a final condition forcing file. When a simulation is split in different sub-runs, the procedure is as follows:

- The switch `iopt_morph_init` is set to zero in the initial startup run.
- A final condition file is written. In the subsequent run, `iopt_morph_init` is set to 1 and the data are read from the forcing file now used as an initial condition file.

Remarks

- The **nosubvars** and **subindex** attributes don't need to be defined.
- By default, all arrays are initialised to zero.
- The bed layer thicknesses must be defined with non-zero values in case a fixed sediment layer depth is used.
- When a final condition file is used, the accumulated bed elevation is stored for a next tun.
- Note that no open boundary conditions or nesting procedures are required for the morphological module.

30.3.5 Biological module

purpose : initial and final conditions for the biological module
 time coordinate : yes
 key id : io_inicon, io_fincon
 file number : ics_bio
 sub-variables : yes

The biology module is enabled when **iopt_biology** is set to 1.

time First forcing variable is the **time** coordinate

```
CHARACTER (LEN=lentime) :: time
```

3-D state variables

```
REAL, DIMENSION(nc-1, nr-1, nz) :: bsi, dsi, nh4, no3, nsc, om, po4, sc,  
                                zp, chlorophyll
```

bsi	Biogenic silicon [mol BSi/m ³].
dsi	Dissolved silicon [mol Si/m ³].
nh4	Ammonium [mol NH4/m ³].
no3	Nitrate [mol NO3/m ³].
nsc	Non-silicon consuming phytoplankton [mol C/m ³].
om	Organic matter [mol POC/m ³].
po4	Phosphate [mol PO4/m ³].
sc	Silicon consuming phytoplankton [mol C/m ³].
zp	Zooplankton [mol C/m ³].

chlorophyll Chlorophyll [mol Chl/m³]. Default is zero. Should only be obtained from a final condition file.

2-D state variables

```
! used when iopt_bio_benth>0
REAL, DIMENSION(nc-1,nr-1) :: bsis, nh4s, no3s, oms, po4s
```

bsis Biogenic silicon in the sediment layer [mol BSi/m²].

nh4s Ammonium in the sediment layer [mol NH4/m²].

no3s Nitrate in the sediment layer [mol NO3/m²].

oms Organic matter in the sediment layer [mol POC/m²].

po4s Phosphate in the sediment layer [mol Po4/m²].

open boundary arrays

In analogy with the physics a number of work space arrays at open boundaries may need to be initialised when the open boundary conditions at specific nodes require the values of these arrays from the previous time step. They can only be obtained from a final condition file written by COHERENS.

```
! use depends on type of open boundary conditions and
REAL, DIMENSION(nobu,nz,0:2,nosubvars) :: obcbioatu ! iopt_obic_bio=1
REAL, DIMENSION(nobv,nz,0:2,nosubvars) :: obcbioatv ! iopt_obic_bio=1
```

The meaning of the arrays **obcbioatu**, **obcbioatv** is the same as the scalar arrays defined above for the physics (e.g. **obctmpatu**, **obctmpatv**), except that the arrays are defined for the state variables used in the biology module.

Remarks

Sub-variable attributes are only needed for the open boundary arrays. The variable attributes **nosubvars** and **subindex** are only required for a final condition file which is written by COHERENS.

30.3.6 Particle tracer module

purpose	: initial and final conditions for the particle module
time coordinate	: yes

key id	: io_inicon, io_fincon
file number	: ics_part
sub-variables	: no

time First forcing variable is the **time** coordinate

```
CHARACTER (LEN=lentime) :: time
```

particle attributes

```
! used when
INTEGER, DIMENSION(nopart) :: part%state
INTEGER, DIMENSION(nopart) :: part%drift_state
INTEGER, DIMENSION(nopart) :: part%kdistype      ! iopt_grid_nodim = 3
INTEGER, DIMENSION(nopart) :: part%label
DOUBLE PRECISION, DIMENSION(nopart) :: part%xpos
DOUBLE PRECISION, DIMENSION(nopart) :: part%ypos
DOUBLE PRECISION, DIMENSION(nopart) :: part%zpos ! iopt_grid_nodim = 3
CHARACTER (LEN=lentime), DIMENSION(nopart) :: part%starttime
DOUBLE PRECISION, DIMENSION(nopart) :: part%age
```

state Type of drifting. No default.

- 1: Surface floating particle.
- 2: Submerged particle.

drift_state Particle drift state from a previous run. This attribute is defined in a final condition file, but usually not in a non-standard COHERENS initial condition file.

- 1: Particle has not yet been released. Default.
- 2: Particle is released and drifting.
- 3: Particle is located on a dry/land cell.
- 4: Particle is outside the domain.
- 5: Particle is deposited on the sea bed.

kdistype Selects type of initial location in the vertical for submerged particles.

- 0: At a specified z -level [m]. Default.
- 1: At a specified C-node vertical grid cell.
- 2: At a fixed distance from the sea bed.

	3: At a fixed distance from the sea surface.
label	Particle label (between 1 and <code>nolabels</code>). Default is 1.
xpos	The initial X-location of the particle in double precision [m or degrees longitude, positive East]. Default is 0.0.
ypos	The initial Y-location of the particle in double precision [m or degrees latitude, positive North]. Default is 0.0.
zpos	The initial vertical location of the submerged particle in double precision. Value and unit depends on the value of <code>kdis-type</code> attribute. Default is 0.0. 0: z -coordinate between $-h$ and ζ [m]. 1: Grid level between 1 and <code>nz</code> . 2: Distance from the bottom between 0 and H [m]. 3: Distance from the surface between 0 and H [m].
starttime	Time of release of the particle given as a 23-character string (YYYY/MM/DD:HH:MM:SS:mmm). No default.
age	The particle age at the time of release in double precision. Unit of time is determined by the value of the parameter <code>ptime-unit</code> . Default is 0.0.

30.4 Surface grids

30.4.1 Absolute coordinates of surface grids

purpose	: surface grids
time coordinate	: no
forcing key id	: <code>io_metabs</code> , <code>io_sstabs</code> , <code>io_wavabs</code> , <code>io_parabs</code>
file number	: 1
grid key id	: <code>igrd_meteo</code> , <code>igrd_sst</code> , <code>igrd_waves</code> , <code>igrd_par</code>
sub-variables	: no

For each forcing key id corresponds a surface grid key id. Let `igrd` one of the four grid key ids. Forcing arrays are

```
REAL, DIMENSION(n1dat,n2dat) :: xcoord
REAL, DIMENSION(n1dat,n2dat) :: ycoord
```

where

```
n1dat = surfacegrids(igrd,1)%n1dat
n2dat = surfacegrids(igrd,1)%n2dat
```

and

xcoord X-coordinates of the surface grid [m or degrees longitude, positive East].

ycoord Y-coordinates of the surface grid [m or degrees latitude, positive North].

Remarks

- The grid coordinates of the surface grid represent the locations where the data are defined. The type of the grid can be non-uniform rectangular or curvilinear. In case of a uniform rectangular surface grid, the coordinates are defined with the parameters provided in `usrdef.mod_params` (see Section 19.5.2) and do not need to be provided.
- The coordinate units must be the same as the ones used in the model and determined by the value of `iopt_grid_sph`.
- Surface grids need to be defined when

```
iopt_meteo = 1                      ! meteorological grid
iopt_temp_sbc = 2 or 3                ! SST grid
iopt_waves = 1                        ! wave grid
iopt_biology = 1, iopt_bio_par=2      ! PAR grid
surfacegrids(igrd,1)%nhtype=1,2,3
```

where `igrd` is the key id of the surface grid.

30.4.2 Relative coordinates of surface grids

purpose	:	surface grids
time coordinate	:	no
forcing key id	:	io_metrel, io_sstrel, io_wavrel, io_parrel
file number	:	1
grid key id	:	igrd_meteo, igrd_sst, igrd_waves, igrd_par
sub-variables	:	no

```
INTEGER, DIMENSION(nc, nr) :: icoordC
INTEGER, DIMENSION(nc, nr) :: jcoordC
REAL, DIMENSION(2, 2, nc, nr) :: weightsC
```

- icoordC X-index of the model grid cell containing the location of the data values.
- jcoordC Y-index of the model grid cell containing the location of the data values.
- weightsC Weight factors used for bilinear interpolation.

Remarks

- The coordinate units must be the same as the ones used in the model and determined by the value of `iopt_grid_sph`. The forcing arrays are used for interpolation of the surface data on the model. See Chapter 15 for further details.
- Surface grids need to be defined when

```
iopt_meteo = 1                                ! meteorological grid
iopt_temp_sbc = 2 or 3                         ! SST grid
iopt_waves = 1                                  ! wave grid
iopt_biolgy = 1, iopt_bio_par=2                ! PAR grid
surfacegrids(igrd,1)%nhtype=1,2,3
```

where `igrd` is the grid key id.

- Clearly, it is easier to define the surface grid using absolute coordinates instead of relative ones. The reason for providing the two options is that when absolute coordinates are used as input, the relative coordinates, needed for interpolation of the surface data, need to be calculated inside the code. In case of curvilinear or irregular data grid, the calculation can take quite some time (even in parallel mode). To avoid this, the following procedure can be adopted (taken a meteo grid as example):
 1. Define a cold start run by setting `cold_start=TRUE..`
 2. Use absolute coordinates as input by setting `modfiles(io_metabs,1,1)%status` to 'N' or 'R'.
 3. Set `modfiles(io_metrel,1,2)%status = 'W'`.
 4. After performing the cold run a standard forcing file with relative coordinates is created.

5. Use this forcing file in the actual simulations with `cold_start=.FALSE.` by setting

```
modfiles(io_metrel,1,1)%status = 'R'
modfiles(io_metabs,1,1)%status = '0'
```

30.5 Nesting

30.5.1 Number of nesting locations

purpose	: number of nest locations
time coordinate	: no
forcing key id	: <code>io_nstspc</code>
file number	: 1
sub-variables	: no

The forcing file is used for input when

```
iopt_nests = 1
modfiles(io_nstgrd_spec,1,1)%status = 'R'
```

and either the CIF is not activated or the CIF block `nstgrd_spec` does not exist.

```
INTEGER, DIMENSION(nonestsets) :: nohnstglbc
INTEGER, DIMENSION(nonestsets) :: nohnstglbu
INTEGER, DIMENSION(nonestsets) :: nohnstglbv
INTEGER, DIMENSION(nonestsets) :: nohnstglbx
INTEGER, DIMENSION(nonestsets) :: nohnstglby
INTEGER, DIMENSION(nonestsets) :: novnst
INTEGER, DIMENSION(nonestsets) :: inst2dtype
```

where `nonestsets` is the number of nested sub-grids and

`nohnstglbc` Number of sub-grid open boundary locations at C-nodes. Only needed when nesting is performed for baroclinic currents and C-node scalar (temperature, salinity, sediments, biological) model variables.

`nohnstglbu` Number of sub-grid open boundary locations at U-nodes. Only needed when nesting is performed for surface elevation and U-node (transport) model variables.

- nohnstglbv** Number of sub-grid open boundary locations at V-nodes. Only needed when nesting is performed for surface elevation and V-node (transport) model variables.
- nohnstglbx** Number of sub-grid open boundary locations at X-nodes. Only needed when tangential open boundary conditions are imposed on the sub-grid.
- nohnstglby** Number of sub-grid open boundary locations at Y-nodes. Only needed when tangential open boundary conditions are imposed on the sub-grid.
- novnst** Number of vertical levels on the sub-grid. Only needed when open boundary conditions on the sub-grid are applied for 3-D variables.
- inst2dtype** Selects the type of data for 2-D nesting.
- 1: transports and elevations
 - 2: elevations
 - 3: transports

30.5.2 Locations of sub-grid open boundaries

purpose	: locations of sub-grid open boundaries
time coordinate	: no
forcing key id	: <code>io_nstgrd</code>
file number	: 1, ..., <code>nonestsets</code>
sub-variables	: no

Let for a particular sub-grid with index `iset`,

```
nohnstc = nohnstglbc(iset)
nohnstu = nohnstglbu(iset)
nohnstv = nohnstglbv(iset)
nohnstx = nohnstglbx(iset)
nohnsty = nohnstglby(iset)
novnsth = novnsth(iset)
```

the following forcing arrays can be defined

```
REAL, DIMENSION(nohnstc) :: xcoordatc
REAL, DIMENSION(nohnstc) :: ycoordatc
REAL, DIMENSION(nohnstc,novnsth) :: scoordatc
REAL, DIMENSION(nohnstu) :: xcoordatu
```

```

REAL, DIMENSION(nohnstu) :: ycoordatu
REAL, DIMENSION(nohnstu,novnstz) :: scoordatu
REAL, DIMENSION(nohnstv) :: xcoordatv
REAL, DIMENSION(nohnstv) :: ycoordatv
REAL, DIMENSION(nohnstv,novnstz) :: scoordatv
REAL, DIMENSION(nohnstx) :: xcoordatx
REAL, DIMENSION(nohnstx) :: ycoordatx
REAL, DIMENSION(nohnstx,novnstz) :: scoordatx
REAL, DIMENSION(nohnsty) :: xcoordaty
REAL, DIMENSION(nohnsty) :: ycoordaty
REAL, DIMENSION(nohnsty,novnstz) :: scoordaty

```

where

- xcoordatc** X-coordinates of the sub-grid open boundary data points at C-nodes [m or degrees longitude, positive East]. Only provided if `nohnstglbc(iset)` is non-zero.
- ycoordatc** Y-coordinates of the sub-grid open boundary data points at C-nodes [m or degrees latitude, positive North]. Only provided if `nohnstglbc(iset)` is non-zero.
- scoordatc** σ -coordinates of the sub-grid open boundary data points at C-nodes. Only provided if `nohnstglbc(iset)` is non-zero and 3-D nesting is performed. Default is a spatially uniform σ -grid.
- xcoordatu** X-coordinates of the sub-grid open boundary data points at U-nodes [m or degrees longitude, positive East]. Only provided if `nohnstglbu(iset)` is non-zero.
- ycoordatu** Y-coordinates of the sub-grid open boundary data points at U-nodes [m or degrees latitude, positive North]. Only provided if `nohnstglbu(iset)` is non-zero.
- scoordatu** σ -coordinates of the sub-grid open boundary data points at U-nodes. Only provided if `nohnstglbu(iset)` is non-zero and 3-D nesting is performed. Default is a spatially uniform σ -grid.
- xcoordatv** X-coordinates of the sub-grid open boundary data points at V-nodes [m or degrees longitude, positive East]. Only provided if `nohnstglbv(iset)` is non-zero.
- ycoordatv** Y-coordinates of the sub-grid open boundary data points at V-nodes [m or degrees latitude, positive North]. Only provided if `nohnstglbv(iset)` is non-zero.

- scoordatv** σ -coordinates of the sub-grid open boundary data points at V-nodes. Only provided if `nohnstglbv(iset)` is non-zero and 3-D nesting is performed. Default is a spatially uniform σ -grid.
- xcoordatx** X-coordinates of the sub-grid open boundary data points at X-nodes [m or degrees longitude, positive East]. Only provided if `nohnstglbx(iset)` is non-zero.
- ycoordatx** Y-coordinates of the sub-grid open boundary data points at X-nodes [m or degrees latitude, positive North]. Only provided if `nohnstglbx(iset)` is non-zero.
- scoordatx** σ -coordinates of the sub-grid open boundary data points at X-nodes. Only provided if `nohnstglbx(iset)` is non-zero and 3-D nesting is performed. Default is a spatially uniform σ -grid.
- xcoordaty** X-coordinates of the sub-grid open boundary data points at Y-nodes [m or degrees longitude, positive East]. Only provided if `nohnstglby(iset)` is non-zero.
- ycoordaty** Y-coordinates of the sub-grid open boundary data points at Y-nodes [m or degrees latitude, positive North]. Only provided if `nohnstglby(iset)` is non-zero.
- scoordaty** σ -coordinates of the sub-grid open boundary data points at Y-nodes. Only provided if `nohnstglby(iset)` is non-zero and 3-D nesting is performed. Default is a spatially uniform σ -grid.

30.5.3 Sediment particle attributes

<code>purpose</code>	:	switches and parameters attributes of sediment fractions
<code>time coordinate</code>	:	no
<code>forcing key id</code>	:	<code>io_sedspc</code>
<code>file number</code>	:	1
<code>sub-variables</code>	:	no

The forcing file is used for input when

```
iopt_sed = 1
modfiles(io_sed_spec,1,1)%status = 'R'
```

and either the CIF is not activated or the CIF block `sed_spec` does not exist.

```
REAL, DIMENSION(nf) :: sedpart%bstres_cr
REAL, DIMENSION(nf) :: sedpart%dp
REAL, DIMENSION(nf) :: sedpart%rhos
```

```

REAL, DIMENSION(nf) :: sedpart%ws_cst
INTEGER, DIMENSION(nf) :: sedpart%iopt_bbc_eq
INTEGER, DIMENSION(nf) :: sedpart%iopt_bbc_ref
INTEGER, DIMENSION(nf) :: sedpart%iopt_bedeq
INTEGER, DIMENSION(nf) :: sedpart%iopt_bstres_cr
INTEGER, DIMENSION(nf) :: sedpart%iopt_hidexp
INTEGER, DIMENSION(nf) :: sedpart%iopt_slope
INTEGER, DIMENSION(nf) :: sedpart%iopt_suseq
INTEGER, DIMENSION(nf) :: sedpart%iopt_toteq
INTEGER, DIMENSION(nf) :: sedpart%iopt_type
INTEGER, DIMENSION(nf) :: sedpart%iopt_ws
INTEGER, DIMENSION(nf) :: sedpart%iopt_ws_floc
INTEGER, DIMENSION(nf) :: sedpart%iopt_ws_hindset
INTEGER, DIMENSION(nf) :: sedpart%iopt_ws_lim

```

bstres_cr_cst	Constant critical bed shear stress used when <code>iopt_bstres_cr=1</code> [m^2/s^2]. Default is 0.0.
dp	Particle diameter [m]. Default is 0.0.
rhos	Particle density [kg/m^3]. Default is 2650.0.
ws_cst	Constant falll velocity used when <code>iopt_ws=1</code> .
iopt_bbc_eq	Selects the formulation for the equilibrium sediment concentration. <ul style="list-style-type: none"> 1: Rouse profile. Default. 2: Using q_t/U determined with the equation of Engelund & Hansen (1967) and $\theta_{*,n} = \theta_n$. 3: Using q_t/U determined with the equation of Engelund & Hansen (1967) and using the modified version of Chollet & Cunge (1979) for $\theta_{*,n}$. 4: Using q_t/U determined with the equation of Ackers & White (1973). 5: Using q_t/U determined using (7.82) from Madsen & Grant (1976).
iopt_bbc_ref	Selects the formulation for the reference concentration $c_{a,n}$. <ul style="list-style-type: none"> 1: Smith & McLean (1977). Default. 2: Van Rijn (1984a).
iopt_bedeq	Type of bedload equation.

- 0: Disabled. Default.
- 1: [Meyer-Peter & Müller \(1948\)](#).
- 2: [Engelund & Fredsøe \(1976\)](#).
- 3: [Van Rijn \(1984b\)](#).
- 4: [Wu *et al.* \(2000\)](#).
- 5: [Soulsby \(1997\)](#). This equation includes explicit wave effects.
- 6: [Van Rijn \(1993\)](#).

iop_tbstres_cr Selects the formulation for the critical bottom shear stress.

- 1: User-defined value for each fraction. Default.
- 2: [Brownlie \(1981\)](#) as given by (7.31).
- 3: [Soulsby & Whitehouse \(1997\)](#) as given by (7.32).
- 4: Constant value for the critical Shield parameter from [Wu *et al.* \(2000\)](#).

iop_t hidexp Selects the type of model for hiding and exposure.

- 0: Disabled. Default.
- 1: [Wu *et al.* \(2000\)](#) as given by (7.37).
- 2: [Ashida & Michiue \(1972\)](#) as given by (7.38).

iop_slope Selects the use of a slope factor for the critical shear stress and bed load transport.

- 0: Disabled. Default.
- 1: Enabled using (7.39).

iop_suseq Disables/enables suspended transport.

- 0: Disabled. Default.
- 1: Enabled.

iop_toteq Method for total load transport.

- 0: Disabled. Default.
- 1: [Engelund & Hansen \(1967\)](#) with $\theta_{*,n} = \theta_n$.
- 2: [Engelund & Hansen \(1967\)](#) using the modified version of [Chollet & Cunge \(1979\)](#) for $\theta_{*,n}$.

	3: Ackers & White (1973) .
	4: Madsen & Grant (1976) .
	5: Wu <i>et al.</i> (2000) . Total load is calculated as the sum of suspended and bed load.
iopt_type	Selects type of sediment.
	1: Sand.
	2: Mud.
iopt_ws	Type of method for the settling velocity.
	1: User-defined value for each fraction. Default.
	2: Camenen (2007) formulation (7.40). The coefficients A , B , m depend on the type of sediment selected with the iopt_type attribute.
	3: Stokes formula (7.41).
	4: Soulsby (1997) formula (7.42).
	5: Zhang & Xie (1993) equation (7.43).
iopt_ws_floc	Type of flocculation factor for the settling velocity
	0: Flocculation effect disabled. Default.
	1: Van Leussen (1994) equation (7.47).
	2: Van Rijn (2007b) equation (7.49).
iopt_ws_hindset	Formulation for hindered settling.
	0: Hindered settling disabled. Default.
	1: Richardson & Zaki (1954) equation (7.45).
	2: Winterwerp & van Kesteren (2004) formula (7.46).
iopt_ws_lim	Disables/enables the limitation of the settling velocity for shallow waters.
	0: Disabled. Default.
	1: Enabled.

30.5.4 Properties of particle labels and clouds

purpose : particle labels and clouds
 time coordinate : no

```
forcing key id      : io_parspc
file number        : 1
sub-variables      : no
```

The forcing file is used for input when

```
iopt_part_model>0
modfiles(io_parspec,1,1)%status = 'R'
```

and either the CIF is not activated or the CIF block `part_spec` does not exist.

particle labels

```
! used when
REAL, DIMENSION(nolabels) :: diamconc ! iopt_part_conc>1 or
                                         ! iopt_part_dens>0
REAL, DIMENSION(nolabels) :: rhoconc
```

where

`diamconc` Particle diameter [m]. No default.

`rhoconc` Particle mass concentration [kg/m³]. No default.

particle clouds

```
! used when iopt_part_cloud=1
INTEGER, DIMENSION(noclouds) : cloud%kdistype ! iopt_grid_nodim=1 or 3
INTEGER, DIMENSION(noclouds) : cloud%label
REAL, DIMENSION(noclouds) :: cloud%length
INTEGER, DIMENSION(noclouds) : cloud%nopart
INTEGER, DIMENSION(noclouds) : cloud%noreleases
REAL, DIMENSION(noclouds) :: cloud%orientation
INTEGER, DIMENSION(noclouds) : cloud%state
REAL, DIMENSION(noclouds) :: cloud%thick
REAL, DIMENSION(noclouds) :: cloud%width
```

`kdistype` Selects the type of location in the vertical if the `state` attribute equals 2. Default is 0.

0: At a specified *z*-level.

1: At a specified C-node vertical grid level.

2: At a fixed distance from the sea bed.

	3: At a fixed distance from the sea surface.
label	Label of the particles released in each cloud. Default is 1.
length	Semi-major axis L_{cl} of the ellipsoid as defined by (10.20) [m or degrees longitude]. No default.
nopart	Number of particles released by the clouds during all releases. The sum over all clouds must be equal to the value of <code>nopart</code> defined in <code>usrdef_part_params</code> . No default.
noreleases	Number of releases for each cloud. No default.
orientation	Angle of the semi-major axis with respect to the coordinate X-axis [degrees]. Default is 0^0 .
state	State of the particles within each cloud. May be 1 (surface floats) or 2 (submerged). Default is 2.
thick	Thickness T_{cl} of the ellipsoid as defined by (10.20) [m]. No default.
width	Semi-minor axis W_{cl} of the ellipsoid as defined by (10.20) [m or degrees latitude]. No default.

30.6 Surface boundary conditions for 1-D applications

30.6.1 Tidal surface forcing

purpose	: harmonic data for 1-D surface forcing
time coordinate	: no
forcing key id	: <code>io_luvsur</code>
file number	: 1
sub-variables	: no

```

! used when iopt_sbc_1D = 1 or iopt_sbc_1D = 3
REAL, DIMENSION(nconobc) :: gxslope_amp
REAL, DIMENSION(nconobc) :: gxslope_pha
REAL, DIMENSION(nconobc) :: gyslope_amp
REAL, DIMENSION(nconobc) :: gyslope_pha
! used when iopt_sbc_1D = 1 or iopt_sbc_1D = 2
REAL, DIMENSION(nconobc) :: zeta_amp
REAL, DIMENSION(nconobc) :: zeta_pha

```

where

- gxslope_amp** Amplitudes of the X-component of the pressure gradient divided by the reference density ρ_0 [m^2/s].
- gxslope_ph** Phases of the X-component of the pressure gradient [rad].
- gyslope_amp** Amplitudes of the Y-component of the pressure gradient divided by the reference density ρ_0 [m^2/s].
- gyslope_ph** Phases of the Y-component of the pressure gradient [rad]
- zetadat_amp** Amplitudes of the surface elevation [m].
- zetadat_ph** Phases of the surface elevation [rad].

Remarks

- Defaults of all input data are zero.
- The pressure gradient is normalised by the reference density ρ_0 .
- The phases must be given in radians and not in degrees. To make the conversion from degrees to radians, multiply by the system parameter **degtorad** which equals $\pi/180$.

30.6.2 Surface forcing data

purpose : surface data for 1-D surface forcing
time coordinate : yes
forcing key id : **io_1uvsur**
file number : 2
sub-variables : no

First forcing variable is the **time** coordinate

CHARACTER (LEN=lentime) :: time

Forcing data are

REAL, DIMENSION(novars) :: surdat1d

where **novars** denotes the number of data

surdat1d Surface data. The number, type and order of data depends on the value of the switch **iopt_sbc_1D**

- 1: Pressure gradient in the X- and Y-direction divided by the reference density ρ_0 [m^2/s] and surface elevation [m] (**novars**=3).
- 2: Surface elevation [m] (**novars**=1).
- 3: Pressure gradient in the X- and Y-direction divided by the reference density ρ_0 [m^2/s] (**novars**=2).

30.7 Open boundary conditions specifiers

30.7.1 Specifiers for 2-D normal boundary conditions

purpose : specifiers and harmonic data for 2-D
 normal open boundary conditions
 time coordinate : no
 forcing key id : io_2uvobc
 file number : 1
 sub-variables : no

Forcing data are

```

INTEGER, DIMENSION(nobu) :: ityp2dobu
INTEGER, DIMENSION(nobv) :: ityp2dobv
INTEGER, DIMENSION(nobu) :: iloczobu
INTEGER, DIMENSION(nobv) :: iloczobv
INTEGER, DIMENSION(2:nofiles) :: no2dobuv
INTEGER, DIMENSION(2:nofiles) :: iobc2dtype
INTEGER, DIMENSION(nobu+nobv,2:nofiles) :: index2dobuv
REAL, DIMENSION(nobu,nconobc) :: udatobu_amp
REAL, DIMENSION(nobu,nconobc) :: zdatobu_amp
REAL, DIMENSION(nobu,nconobc) :: udatobu_pha
REAL, DIMENSION(nobu,nconobc) :: zdatobu_pha
REAL, DIMENSION(nobv,nconobc) :: vdatobv_amp
REAL, DIMENSION(nobv,nconobc) :: zdatobv_amp
REAL, DIMENSION(nobv,nconobc) :: vdatobv_pha
REAL, DIMENSION(nobv,nconobc) :: zdatobv_pha
INTEGER, DIMENSION(nqsecobu,2) :: iqsecobu
INTEGER, DIMENSION(nqsecobv,2) :: iqsecobv

```

where `nofiles-1` is the number of data files, i.e. files with file number 2, ..., `nofiles` and

`ityp2dobu` Type of open boundary condition at U-nodes. See Section 5.10.1 for details.
 0 : Clamped. Default.
 1 : Zero slope.
 2 : Zero volume flux.
 3 : Specified elevation and local solution for transport.
 4 : Specified transport.

- 5 : Radiation condition using shallow water speed.
- 6 : [Orlanski \(1976\)](#) condition.
- 7 : [Camerlengo & O'Brien \(1980\)](#).
- 8 : [Flather \(1976\)](#) with specified elevation and transport.
- 9 : Flather with specified elevation.
- 10: [Røed & Smedstad \(1984\)](#).
- 11: Characteristic method with specified elevation and transport.
- 12: Characteristic method with specified elevation.
- 13: Characteristic method using a zero normal gradient condition.
- 14: Imposed depth mean current.
- 15: Imposed discharge.
- 16: Imposed discharge distributed along open boundary sections.
- 17: Specified surface slope.

iloczobu	If the elevation has to be specified at the open boundary, the array selects the position of the specified elevation with respect to the open boundary.
	0: Not required.
	1: At the open boundary U-node. Default.
	2: At the “nearest” C-node outside the domain.
ityp2dobv	The same as ityp2dobu now for open boundaries at V-nodes.
iloczobv	The same as iloczobu now for open boundaries at V-nodes.
no2dobuv	Number of data locations within each data file. Only provided if nofiles >1.
iobc2dtype	Identifies the variables within the data file. Only required if nofiles >1. No default.
	1: Depth-integrated currents and elevations.
	2: Elevations only.
	3: Depth-integrated currents only.
index2dobuv	Each data file contains a sub-set of open boundary data points. The element index2dobuv(idat,ifil) maps, for file ifil, the local

data point **idat** into a corresponding global open boundary index (between 1:**nobu** for U- and **nobu+1:nobu+nobv** for V-open boundaries). The actual size of the first dimension, i.e. as used in the simulation, for file **ofil** equals **no2dcbc(ofil)**. Only required if **nofiles>1**.

udatobu_amp	Amplitudes of the depth-integrated current U at U-open boundaries [m^2/s].
zdatobu_amp	Amplitudes of the surface elevation ζ or surface slope $\partial\zeta/\partial x_1$ at U-open boundaries [m] or [-].
udatobu_ph	Phases of the depth-integrated current U at U-open boundaries [rad].
zdatobu_ph	Phases of the surface elevation ζ or surface slope $\partial\zeta/\partial x_1$ at U-open boundaries [rad].
vdatobv_amp	Amplitudes of the depth-integrated current V at V-open boundaries [m^2/s].
zdatobv_amp	Amplitudes of the surface elevation ζ or surface slope $\partial\zeta/\partial x_2$ at V-open boundaries [m] or [-].
vdatobv_ph	Phases of the depth-integrated current V at V-open boundaries [rad].
zdatobv_ph	Phases of the surface elevation ζ or surface slope $\partial\zeta/\partial x_2$ at V-open boundaries [rad].
iqsecobu	Start and end index of the U-open boundary sections where a discharge condition is specified (ityp2dobuv=16).
iqsecobv	Start and end index of the V-open boundary sections where a discharge condition is specified (ityp2dobv=16).

30.7.2 Specifiers for 2-D tangential open boundary conditions

purpose	: specifiers and harmonic data for 2-D tangential open boundary conditions
time coordinate	: no
forcing key id	: io_2xyobc
file number	: 1
sub-variables	: no

```
INTEGER, DIMENSION(nobx) :: ityp2dobx
INTEGER, DIMENSION(noby) :: ityp2doby
```

```

INTEGER, DIMENSION(2:nofiles) :: no2dobxy
INTEGER, DIMENSION(nobx+noby,2:nofiles) :: index2dobxy
REAL, DIMENSION(nobx,nconobc) :: vdatobx_amp
REAL, DIMENSION(nobx,nconobc) :: vdatobx_pha
REAL, DIMENSION(noby,nconobc) :: udatoby_amp
REAL, DIMENSION(noby,nconobc) :: udatoby_pha

```

where `nofiles-1` is the number of data files, i.e. files with file number 2, ..., `nofiles` and

<code>ityp2dobx</code>	Type of 2-D tangential open boundary condition for the evaluation of the cross-stream advection term in the V - or v -momentum equation at X-nodes. See Section 12.3.16.2 for details.
	0: Zero gradient condition. Default.
	1: Using external values for U .
	2: Upwind scheme.
<code>ityp2doby</code>	Type of 2-D tangential open boundary condition for the evaluation of the cross-stream advecting term in the U - or u momentum equation at Y-nodes. See Section 12.3.16.2 for details.
	0: Zero gradient condition. Default.
	1: Using external values for V .
	2: Upwind scheme.
<code>no2dobxy</code>	Number of data locations within each data file. Only required if <code>nofiles>1</code> .
<code>index2dobxy</code>	Each data file contains a sub-set of open boundary data points. The element <code>index2dobxy(idat,ifil)</code> maps, for file <code>ifil</code> , the local data point <code>idat</code> into a corresponding global open boundary index (between 1: <code>nobx</code> for X- and <code>nobx+1:nobx+noby</code> for Y-open boundaries). The actual size of the first dimension, i.e. as used in the simulation, for file <code>ifil</code> equals <code>no2dobxy(ifil)</code> . Only required if <code>nofiles>1</code> .
<code>vdatobx_amp</code>	Amplitudes of the depth-integrated V -current at the exterior V-nodes adjacent to X-node boundaries [m^2/s].
<code>vdatobx_pha</code>	Phases of the depth-integrated V -current at the exterior V-nodes adjacent to X-node boundaries [rad].
<code>udatoby_amp</code>	Amplitudes of the depth-integrated U -current at the exterior U-nodes adjacent to Y-node boundaries [m^2/s].

udatoby_ph Phases of the depth-integrated *U*-current at the exterior *U*-nodes adjacent to *Y*-node boundaries [rad].

Remarks

- Note that 2-D tangential open boundary conditions can only be used in case of a 2-D grid (*iopt_grid_nodim*=2) or when the time integration scheme is set to explicit (*iopt_hydro_impl*=0).
- Phases must be given in radians and not in degrees. Conversion from degrees to radians can be made by multiplying the phases with the parameter *degtorad* equal to $\pi/180$.
- As stated above, the harmonic amplitudes and phases in the tangential case can only be defined if *iopt_abc_2D_tang*=1 and either *iopt_grid_nodim*=2 or *iopt_hydro_impl*=0.

30.7.3 Specifiers for 3-D normal open boundary conditions

30.7.3.1 Without sub-variables

purpose : specifiers for 3-D normal open boundary conditions without sub-variables

time coordinate : no

forcing key id : *io_3uvobc*, *io_salobc*, *io_tmppobc*

file number : 1

sub-variables : no

```
INTEGER, DIMENSION(nobu) :: itypobu
INTEGER, DIMENSION(nobu,1) :: iprofobu
INTEGER, DIMENSION(nobv) :: itypobv
INTEGER, DIMENSION(nobv,1) :: iprofobv
INTEGER, DIMENSION(2:nofiles,1) :: noprofsd
INTEGER, DIMENSION(nobu+nobv,2:nofiles,1) :: indexprof
```

where *nofiles*-1 are the number of data files, i.e. files with file number 2, ..., *nofiles* and

itypobu Selects the type of open boundary condition used at *U*-nodes (normal case) in case that no profile number has been defined with a positive value (see below). The type of condition depends on the value of the forcing key id:

io_3uvobc

- 0: First order zero gradient condition. Default.
- 1: Second order zero gradient condition.
- 2: Local solution.
- 3: Radiation condition using internal wave speed.
- 4: Orlanski type of radiation condition.
- 5: Discharge condition.

io_salobc, io_tmppobc

- 0: First order zero gradient condition. Default.
- 1: Radiation condition using the internal wave speed.
- 2: Orlanski condition.
- 3: Thatcher-Harleman condition.

iprofobu	Profile number used at U-node open boundaries. Negative values are not allowed. If a profile number is zero, the open boundary condition is defined by the value of itypobu . If the profile contains flagged data, i.e. values equal to float_fill , a zero-gradient condition is applied.
itypobv	Selects type of open boundary condition at V-nodes (normal case) in case that no profile number has been defined with a positive value (see below). Definitions are the same as above for itypobu .
iprofobv	Profile number used at V-node open boundaries. Negative values are not allowed. If a profile number is zero, the open boundary condition is defined by the value of itypobv . If the profile contains flagged data, i.e. values equal to float_fill , a zero-gradient condition is applied.
noprofsd	Number of data profiles for each data file. Only required if nofiles >0.
indexprof	Each data file contains a sub-set of open boundary profiles. The element indexprof(iprof,ifil,1) maps, for file ifil , the local profile number iprof into a corresponding “global” index as defined by iprofobu and iprofobv . The actual size of the first dimension for file ifil equals noprofsd(ifil,1) . If not defined and nofiles =2, the program sets indexprof(1:noprofsd(ifil,1),ifil,1)= (/(1,2,...,noprofsd(ifil,1))) . Only required if nofiles >1.

Remarks are given in Section 21.2.

30.7.3.2 With sub-variables

purpose	: specifiers for 3-D normal open boundary conditions with sub-variables
time coordinate	: no
forcing key id	: <code>io_sedobc</code> , <code>io_bioobc</code>
file number	: 1
sub-variables	: yes

The following setup arrays are or may be defined

```
INTEGER, DIMENSION(nobu,nosub1) :: itypobu
INTEGER, DIMENSION(nobu,nosub2) :: iprofobu
INTEGER, DIMENSION(nobv,nosub3) :: itypobv
INTEGER, DIMENSION(nobv,nosub4) :: iprofobv
INTEGER, DIMENSION(2:nofiles,nosub5) :: noprofsd
INTEGER, DIMENSION(nobu+nobv,2:nofiles,nosub5) :: indexprof
```

where `nofiles` -1 are the number of data files, i.e. files with file number 2, ..., `nofiles` and

itypobu Selects the type of open boundary condition used at U-nodes in case that no profile number has been defined with a positive value (see below).

- 0: First order zero gradient condition. Default.
- 1: Radiation condition using the internal wave speed.
- 2: Orlanski condition.
- 3: Thatcher-Harleman condition.

iprofobu Profile number used at U-node open boundaries. Negative values are not allowed. If a profile number is zero, the open boundary condition is defined by the value of **itypobu**. If the profile contains flagged data, i.e. values equal to `float_fill`, a zero-gradient condition is applied.

itypobv Selects type of open boundary condition at V-nodes in case that no profile number has been defined with a positive value (see below). Definitions are the same as above for **itypobu**.

iprofobv Profile number used at V-node open boundaries. Negative values are not allowed. If a profile number is zero, the open boundary condition is defined by the value of **itypobv**. If the profile contains flagged data, i.e. values equal to `float_fill`, a zero-gradient condition is applied.

- noprofsd** Number of data profiles for each data file and sub-variable. Only required if **nofiles**>0.
- indexprof** Each data file contains a sub-set of open boundary profiles. The element **indexprof(iprof,ifil,ivar)** maps, for file **ifil**, the local profile number **iprof** into a corresponding “global” index as defined by **iprofobu** and **iprofobv**. The actual size of the first dimension for file **ifil** equals **noprofsd(ifil,ivar)**. If not defined and **nofiles**=2, the program sets

$$\text{indexprof}(1:\text{noprofsd}(\text{ifil},\text{ivar}),\text{ifil},\text{ivar}) = ((1,2,\dots,\text{noprofsd}(\text{ifil},\text{ivar})))$$
. Only required if **nofiles**>0.

The last dimension of the above arrays **nosubvars1**, ..., **nosubvars5** are the value of the **nosubvars** attribute. Its value must be lower than or equal to the number of suspended sediment fractions **nfsus** in case of multi-fraction sediments, 3 for the flocculation model or the number of 3-D state variables **nobiovars3d** in the biological case). The variable attribute **subindex** with the indices of the corresponding fractions or state variables must be defined for those forcing arrays with a sub-variable last dimension.

Further remarks are given in Section [21.2](#).

30.7.4 Specifiers for 3-D tangential open boundary conditions

purpose	: specifiers for 3-D tangential open boundary conditions without sub-variables
time coordinate	: no
forcing key id	: io_3xyobc
file number	: 1
sub-variables	: no
INTEGER, DIMENSION(nobx,1) :: itypobx	
INTEGER, DIMENSION(nobx,1) :: iprofobx	
INTEGER, DIMENSION(noby,1) :: itypoby	
INTEGER, DIMENSION(noby,1) :: iprofoby	
INTEGER, DIMENSION(2:nofiles,1) :: noprofsd	
INTEGER, DIMENSION(nobx+noby,2:nofiles,1) :: indexprof	

where **nofiles**-1 are the number of data files, i.e. files with file number 2, ..., **nofiles** and

itypobx Selects the type of open boundary condition used at X-nodes (normal case) in case that no profile number has been defined with a positive value (see below).

0:	First order zero gradient condition. Default.
1:	Using an external data profile.
2:	Upwind scheme.
iprofobx	Profile number used at X-node open boundaries. Negative values are not allowed. If a profile number is zero, the open boundary condition is defined by the value of itypobx .
itypobx	Selects type of open boundary condition at Y-nodes (normal case) in case that no profile number has been defined with a positive value (see below). Definitions are the same as above for itypobx .
iprofoby	Profile number used at Y-node open boundaries. Negative values are not allowed. If a profile number is zero, the open boundary condition is defined by the value of itypoby .
noprofsd	Number of data profiles for each data file. Only required if nofiles >0.
indexprof	Each data file contains a sub-set of open boundary profiles. The element indexprof(iprof,ifil,1) maps, for file ifil , the local profile number iprof into a corresponding “global” index as defined by iprofobx and iprofoby . The actual size of the first dimension for file ifil equals noprofsd(ifil,1) . If not defined and nofiles =2, the program sets indexprof(1:noprofsd(ifil,1)) = $((1,2,\dots,noprofsd(ifil,1)))/$. Only required if nofiles >0.

Remarks are given in Section 21.2.

30.8 Open boundary data

30.8.1 2-D normal open boundary data

purpose	:	data for 2-D normal open boundaries
time coordinate	:	yes
forcing key id	:	io_2uvobc
file number	:	2, ..., nofiles
sub-variables	:	no

where **nofiles-1** is the number of data files, i.e. files with file number 2, ..., **nofiles**. First forcing variable is the **time** coordinate.

```
CHARACTER (LEN=lentime) :: time
```

Forcing data are

```
REAL, DIMENSION(nodat,novars) :: obcdatuv2d
```

where `novars` are the number of data variables. In the normal case, its value depends on the value of `iobc2dtype(ifil)`:

- 1: Equals 2 since both depth integrated current and surface elevation data are required.
- 2: Equals 1 since only surface elevation data are required.
- 3: Equals 1 since only depth integrated current data are required.

In the tangential case, `novars=1`.

`obcdatuv2d` Input data for the forcing file with file number `ifil`.

30.8.2 2-D tangential open boundary data

<code>purpose</code>	:	data for 2-D tangential open boundaries
<code>time coordinate</code>	:	yes
<code>forcing key id</code>	:	<code>io_2xyobc</code>
<code>file number</code>	:	<code>2, ..., nfiles</code>
<code>sub-variables</code>	:	no

where `nfiles-1` is the number of data files, i.e. files with file number `2, ..., nfiles`. First forcing variable is the `time` coordinate

```
CHARACTER (LEN=lentime) :: time
```

Forcing data are

```
REAL, DIMENSION(nodat,1) :: obcdatxy2d
```

where

`obcdatxy2d` Input data for forcing file with file number `ifil`.

30.8.3 Profile data for 3-D variables

30.8.3.1 Physical variables

<code>purpose</code>	:	open boundary data profiles for salinity, temperature, currents (normal, tangential)
<code>time coordinate</code>	:	yes
<code>forcing key id</code>	:	<code>io_salobc, io_tmlobc, io_3uvobc, io_3xyobc</code>
<code>file number</code>	:	<code>2,.., nfiles</code>
<code>sub-variables</code>	:	no

where `nofiles-1` is the number of data files, i.e. files with file number `ifil` ranging from `2,.., nofiles`. First forcing variable is the `time` coordinate

```
CHARACTER (LEN=lentime) :: time
```

Forcing data are

```
REAL, DIMENSION(numprofs,nz,1) :: prof3dsal
```

for salinity (key id `io_salobc`),

```
REAL, DIMENSION(numprofs,nz,1) :: prof3dtmp
```

for temperature (key id `io_tmppobc`),

```
REAL, DIMENSION(numprofs,nz,1) :: prof3dveluv
```

for baroclinic currents (key id `io_3uvobc`) at normal boundaries and

```
REAL, DIMENSION(numprofs,nz,1) :: prof3dvelxy
```

for currents (key id `io_3xyobc`) at tangential boundaries.

The array dimension `numprofs` represents the number of profiles in the forcing file given by `noprofsd(ifil,1)` and the name of the forcing variable represents the type of data

`prof3dsal` Salinity profile data for the forcing file with file number `ifil` [PSU].

`prof3dtmp` Temperature profile data for the forcing file with file number `ifil` [$^{\circ}\text{C}$].

`prof3dveluv` Baroclinic current profile data for the forcing file with file number `ifil` [m/s]. Used as boundary condition for normal advective flux at U- and V-nodes.

`prof3dvelxy` Current profile data for the forcing file with file number `ifil` [m/s]. Used as boundary condition for tangential advective fluxes at X- and Y-nodes.

30.8.4 Profile data for the multi-fraction sediments model

purpose : open boundary data profiles for sediments
in the multi-fraction model

time coordinate : yes

forcing key id : `io_sedobc`

file number : `2,.., nofiles`

sub-variables : yes

where `nofiles-1` is the number of data files, i.e. files with file number `ifil` ranging from `2, ..., nofiles`.

First forcing variable is the `time` coordinate

```
CHARACTER (LEN=lentime) :: time
```

Forcing data are

```
REAL, DIMENSION(numprofs,nz,nosubvars) :: prof3dsed
```

where `numprofs` are number the number of profiles in the forcing file given by `SUM(noprofsd(ifil,:))`, `nosubvars` the number of fractions for which open boundary profiles are provided, which must be smaller than the number of suspended fractions `nfsus`, and

`prof3dsed` Profile data of sediment concentrations for the forcing file with file number `ifil` [m^3/m^3].

Note that the `nosubvars` and `subindex` variable attributes must be defined.

30.8.5 Profile data for the flocculation model

<code>purpose</code>	:	open boundary data profiles for the flocculation model
<code>time coordinate</code>	:	yes
<code>forcing key id</code>	:	<code>io_sedobc</code>
<code>file number</code>	:	<code>2, ..., nofiles</code>
<code>sub-variables</code>	:	no

where `nofiles-1` is the number of data files, i.e. files with file number `ifil` ranging from `2, ..., nofiles`.

First forcing variable is the `time` coordinate

```
CHARACTER (LEN=lentime) :: time
```

Forcing data are

```
REAL, DIMENSION(numprofs,nz) :: prof3dflocP
```

```
REAL, DIMENSION(numprofs,nz) :: prof3dflocF
```

```
REAL, DIMENSION(numprofs,nz) :: prof3dflocT
```

where `numprofs` is the total number for profiles given by `noprofsd(ifil,ivar)` and

`prof3dflocP` Open boundary profile data with number concentrations of floculi (N_P) [m^{-3}].

`prof3dflocF` Open boundary profile data with number concentrations of flocs (N_F) [m^{-3}].

`prof3dflocT` Open boundary profile data with number concentrations of floculi in flocs (N_T) [m^{-3}].

30.8.6 Profile data for the biological model

purpose : open boundary data profiles for the biological model
 time coordinate : yes
 forcing key id : io_bioobc
 file number : 2, , nofiles
 sub-variables : no

where `nofiles-1` is the number of data files, i.e. files with file number `ifil` ranging from 2, . . . , `nofiles`.

First forcing variable is the `time` coordinate

```
CHARACTER (LEN=lentime) :: time
```

Forcing data are

```
REAL, DIMENSION(numprofs,nz) :: prof3dbsi
REAL, DIMENSION(numprofs,nz) :: prof3ddsi
REAL, DIMENSION(numprofs,nz) :: prof3dnh4
REAL, DIMENSION(numprofs,nz) :: prof3dno3
REAL, DIMENSION(numprofs,nz) :: prof3dnsc
REAL, DIMENSION(numprofs,nz) :: prof3dom
REAL, DIMENSION(numprofs,nz) :: prof3dpo4
REAL, DIMENSION(numprofs,nz) :: prof3dsc
REAL, DIMENSION(numprofs,nz) :: prof3dzp
```

where `numprofs` is the total number of profiles given by `noprofsd(ifil,ivar)` and

`prof3dbsi` Biogenic silicon [mol Bsi/m³].

`prof3ddsi` Silicon available for biological uptake [mol DSi/m³].

`prof3dnh4` Ammonium [mol NH4/m³].

`prof3dno3` Nitrate [mol NO3/m³].

`prof3dnsc` Non-silicon consuming phytoplankton [mol C/m³].

`prof3dom` Organic matter [mol POC/m³].

`prof3dpo4` Phosphate [mol PO4/m³].

`prof3dsc` Silicon consuming phytoplankton [mol C/m³].

`prof3dzp` Zooplankton [mol C/m³].

It is clear that the actual number of profile data array must be equal to the parameter `nbobc` (see Section 30.7.3.2).

30.9 Surface forcing data

For each type of surface forcing corresponds a surface grid. The grid type and dimensions will be represented by the parameters

```
nhtype = surfacegrids(igrd,1)%nhtype
n1dat = surfacegrids(igrd,1)%n1dat
n2dat = surfacegrids(igrd,1)%n2dat
```

where `igrd` is key id of the grid which may take on of the values `igrd_meteo`, `igrd_sst`, `igrd_waves`, `igrd_par`.

30.9.1 Meteorological forcing

The case of a single point grid (i.e. `n1dat*n2dat=1`) and a grid with multiple cells are treated separately. Note that the number and the type of the forcing variables depends on the settings of a number of switches (see Section 30.9.1.2 below).

30.9.1.1 Single point grid

<code>purpose</code>	:	meteorological forcing for a single point grid
<code>time coordinate</code>	:	yes
<code>key id</code>	:	<code>io_metsur</code>
<code>file number</code>	:	1
<code>sub-variables</code>	:	no

The first forcing variable is

```
CHARACTER (LEN=lenname), DIMENSION(nosubnames) :: subname
```

where `nosubnames` is the number of forcing data variables and `subname` the names of the forcing data variables which can take the following values

<code>uwindatc</code>	X-component of the surface wind [m/s].
<code>vwindatc</code>	Y-component of the surface wind [m/s].
<code>usstresatc</code>	X-component of the surface stress [m^2/s^2].
<code>vsstresatc</code>	Y-component of the surface stress [m^2/s^2].
<code>airtemp</code>	Air temperature [0C].
<code>relhum</code>	Relative humidity (between 0 and 1).
<code>cloud_cover</code>	Cloud cover (between 0 and 1).

qnsol Non-solar upward surface heat flux [W/m²].
qrad Surface solar irradiance [W/m²].
evapminprec Evaporation minus precipitation rate [kg/m²/s].
precipitation Precipitation rate [kg/m²/s] .

Second forcing variable is the **time** coordinate

CHARACTER (LEN=lentime) :: time

The third variable is a vector where the forcing data are stored.

REAL, DIMENSION(nosubnames) :: metedata

30.9.1.2 Multiple point grid

purpose : meteorological forcing for a surface data grid
time coordinate : yes
key id : io_metsur
file number : 1
sub-variables : no

First forcing variable is the **time** coordinate

CHARACTER (LEN=lentime) :: time

Forcing data are

REAL, DIMENSION(n1dat,n2dat) :: uwindatc	! used when ! iopt_meteo_data=1, ! iopt_meteo_stres=1
REAL, DIMENSION(n1dat,n2dat) :: vwindatc	! iopt_meteo_data=1, ! iopt_meteo_stres=1
REAL, DIMENSION(n1dat,n2dat) :: usstresatc	! iopt_meteo_data=2, ! iopt_meteo_stres=1
REAL, DIMENSION(n1dat,n2dat) :: vsstresatc	! iopt_meteo_data=2, ! iopt_meteo_stres=1
REAL, DIMENSION(n1dat,n2dat) :: atmpres	! iopt_meteo_pres=1,nhtype>0
REAL, DIMENSION(n1dat,n2dat) :: airtemp	! iopt_meteo_data=1, ! iopt_meteo_heat=1
REAL, DIMENSION(n1dat,n2dat) :: relhum	! iopt_meteo_data=1, ! iopt_meteo_heat=1
REAL, DIMENSION(n1dat,n2dat) :: cloud_cover	! iopt_meteo_data=1, ! iopt_meteo_heat=1

```

REAL, DIMENSION(n1dat,n2dat) :: qnsol           ! iopt_meteo_data=2,
                                                ! iopt_meteo_heat=1
REAL, DIMENSION(n1dat,n2dat) :: qrad            ! iopt_meteo_data=2,
                                                ! iopt_meteo_heat=1
REAL, DIMENSION(n1dat,n2dat) :: evapminprec    ! iopt_meteo_precip=1
REAL, DIMENSION(n1dat,n2dat) :: precipitation   ! iopt_meteo_precip=2

```

The variables have the same meaning as above except for

atmpres Atmospheric pressure [Pa].

30.9.2 Surface temperature forcing

purpose	:	SST forcing for a single or multiple point grid
time coordinate	:	yes
key id	:	io_sstsur
file number	:	1
sub-variables	:	no

First forcing variable is the **time** coordinate

```
CHARACTER (LEN=lentime) :: time
```

Forcing data are

```

! used when
REAL, DIMENSION(n1dat,n2dat) :: sst ! iopt_temp_sbc=2,3

```

where

sst Sea surface temperature [$^{\circ}$ C].

30.9.3 Surface wave forcing

As above the case of a single point grid (i.e. $n1dat*n2dat=1$) and a grid with multiple cells are treated separately. Note that the number and type of the forcing variables depends on the settings of a number of switches (see Section 30.9.3.2 below).

30.9.3.1 Single point grid

purpose : wave forcing for a single point grid
 time coordinate : yes
 key id : io_wavsur
 file number : 1
 sub-variables : no

The first forcing variable is

`CHARACTER (LEN=lenname), DIMENSION(nosubnames) :: subname`

where `nosubnames` is the number of forcing data variables and `subname` the names of the forcing data variables which can take the following values

`waveheight` Significant wave height [m].
`waveperiod` Peak wave period [s].
`wavedir` Mean wave direction [rad].
`wavevel` Near-bottom wave orbital velocity [m/s].
`waveexcurs` Near-bottom wave excursion velocity [m/s].
`umstokesatc` X-component of the depth-mean Stokes drift [m/s].
`vmstokesatc` Y-component of the depth-mean Stokes drift [m/s].
`wavepres` Wave induced pressure [m^2/s^2].
`umsdissipatc` X-component of the 2-D surface wave dissipation force [m^2/s^2].
`vmsdissipatc` Y-component of the 2-D surface wave dissipation force [m^2/s^2].
`umbwdissipatc` X-component of the 2-D bed wave dissipation force [m^2/s^2].
`vmbwdissipatc` Y-component of the 2-D bed wave dissipation force [m^2/s^2].

The use of these forcing variables by the model depends on the settings of a number of switches (see Section 30.9.3.2 below).

Second forcing variable is the `time` coordinate

`CHARACTER (LEN=lentime) :: time`

The third variable is a vector where the forcing data are stored.

`REAL, DIMENSION(nosubnames) :: wavedata`

30.9.3.2 Surface data grid

purpose : surface wave data for a multiple point grid
 time coordinate : yes
 key id : io_wavsur
 file number : 1
 sub-variables : no

First forcing variable is the time coordinate

```
CHARACTER (LEN=lentime) :: time
```

Forcing data are

```

! used when
REAL, DIMENSION(n1dat,n2dat) :: waveheight
REAL, DIMENSION(n1dat,n2dat) :: waveperiod
REAL, DIMENSION(n1dat,n2dat) :: wavedir
REAL, DIMENSION(n1dat,n2dat) :: wavevel      ! iopt_waves_form=2
REAL, DIMENSION(n1dat,n2dat) :: waveexcurs  ! iopt_waves_form=2
REAL, DIMENSION(n1dat,n2dat) :: umstokesatc ! iopt_waves_curr=1,
                                              ! iopt_waves_form=2
REAL, DIMENSION(n1dat,n2dat) :: vmstokesatc ! iopt_waves_curr=1,
                                              ! iopt_waves_form=2
REAL, DIMENSION(n1dat,n2dat) :: wavepres     ! iopt_waves_pres=1
REAL, DIMENSION(n1dat,n2dat) :: umswdissipatc ! iopt_waves_dissip=1
REAL, DIMENSION(n1dat,n2dat) :: vmswdissipatc ! iopt_waves_dissip=1
REAL, DIMENSION(n1dat,n2dat) :: umbwdissipatc ! iopt_waves_dissip=1
REAL, DIMENSION(n1dat,n2dat) :: vmbwdissipatc ! iopt_waves_dissip=1

```

The variables have the same meaning as above.

30.9.4 Surface biological (PAR) data

purpose : PAR forcing for a single or multiple point grid
 time coordinate : yes
 key id : io_parsur
 file number : 1
 sub-variables : no

First forcing variable is the time coordinate

```
CHARACTER (LEN=lentime) :: time
```

Forcing data are

```
      ! used when
REAL, DIMENSION(n1dat,n2dat) :: peakpar ! iopt_bio_par=2
```

where

peakpar Peak photosynthetically available radiation (PAR) data [mole photons/m²/s].

30.10 Structures

30.10.1 Dry cells

purpose	:	dry cells
time coordinate	:	no
key id	:	io_drycel
file number	:	1
sub-variables	:	no

Forcing data are

```
INTEGER, DIMENSION(numdry) :: idry
INTEGER, DIMENSION(numdry) :: jdry
```

where **numdry** is the number of dry cells and

idry (Global) X-indices of the dry cells.

jdry (Global) Y-indices of the dry cells.

30.10.2 Thin dams

purpose	:	thin dams
time coordinate	:	no
key id	:	io_thndam
file number	:	1
sub-variables	:	no

Forcing data are

```
INTEGER, DIMENSION(numthinu) :: ithinu
INTEGER, DIMENSION(numthinu) :: jthinu
INTEGER, DIMENSION(numthinv) :: ithinv
INTEGER, DIMENSION(numthinv) :: jthinv
```

where `numthinu`, `numthinv` are the number of thin dams at respectively U- and V-nodes and

- `ithinu` (Global) X-indices of thin dams at the U-nodes.
- `jthinu` (Global) Y-indices of thin dams at the U-nodes.
- `ithinv` (Global) X-indices of thin dams at the V-nodes.
- `jthinv` (Global) Y-indices of thin dams at the V-nodes.

30.10.3 Weirs and barriers

purpose	: weirs and barriers
time coordinate	: no
key id	: <code>io_weibar</code>
file number	: 1
sub-variables	: no

Forcing data are

```
INTEGER, DIMENSION(numwbaru) :: iwbaru
INTEGER, DIMENSION(numwbaru) :: jwbaru
REAL, DIMENSION(numwbaru) :: oricoefu
REAL, DIMENSION(numwbaru) :: oriheightu
REAL, DIMENSION(numwbaru) :: oriwidthu
REAL, DIMENSION(numwbaru) :: wbarcoefu
REAL, DIMENSION(numwbaru) :: wbarcrestu
REAL, DIMENSION(numwbaru) :: wbarmodlu
INTEGER, DIMENSION(numwbarv) :: iwbarv
INTEGER, DIMENSION(numwbarv) :: jwbarv
REAL, DIMENSION(numwbarv) :: oricoefv
REAL, DIMENSION(numwbarv) :: oriheightv
REAL, DIMENSION(numwbarv) :: oriwidthv
REAL, DIMENSION(numwbarv) :: wbarcoefv
REAL, DIMENSION(numwbarv) :: wbarcrestv
REAL, DIMENSION(numwbarv) :: wbarmodlv
```

where `numwbaru` and `numwbarv` are the the number of weirs/barriers at U- respectively V-nodes and

- `iwbaru` (Global) X-indices of weirs or barriers at U-nodes.
- `jwbaru` (Global) Y-indices of weirs or barriers at U-nodes.
- `oricoefu` Discharge coefficient C_e for orifices at U-nodes [-].

oriheightu Orifice height O_h at U-nodes [m].
oriwidthu Orifice width O_w at U-nodes [m].
wbarcoefu Discharge coefficient C_{st} at U-nodes [$m^{1/2}/s$].
wbarcrestu Height of the weir crests h_{cr} at U-nodes [m].
wbarmodlu Modular limit m at the U-nodes.
iwbarv (Global) X-indices of weirs or barriers at V-nodes.
jwbarv (Global) Y-indices of weirs or barriers at V-nodes.
oricoefv Discharge coefficient C_e for orifices at V-nodes [–].
oriheightv Orifice height O_h at V-nodes [m].
oriwidthv Orifice width O_w at V-nodes [m].
wbarcoefv Discharge coefficient C_{st} at V-nodes [$m^{1/2}/s$].
wbarcrestv Height of the weir crests h_{cr} at V-nodes [m].
wbarmodlv Modular limit m at V-nodes.

For further remarks see Section 26.3.

30.11 Discharges

30.11.1 Discharge locations

purpose : discharge locations
 time coordinate : yes
 key id : `io_disloc`
 file number : 1
 sub-variables : no

First forcing variable is the time coordinate

`CHARACTER (LEN=lentime) :: time`

Forcing data are

```
REAL, DIMENSION(numdis) :: xdiscoord
REAL, DIMENSION(numdis) :: ydiscoord
REAL, DIMENSION(numdis) :: zdiscoord
```

where `numdis` is the number of discharge locations and

`xdiscoord` X-coordinates of the discharge locations [m or degrees longitude, positive East].

ydiscoord Y-coordinates of the discharge locations [m or degrees latitude, positive North]. or fractional degrees latitude].

zdiscoord Vertical coordinates of the discharge locations (distance from sea bed or sea surface depending on the value of **kdistype** [m]).

Note that (contrary to the general rule) these forcing arrays must always be provided in the order given above.

30.11.2 Discharged volume

purpose : discharged volume
 time coordinate : yes
 key id : io_disvol
 file number : 1
 sub-variables : no

First forcing variable is the **time** coordinate

CHARACTER (LEN=lentime) :: time

Forcing data are

REAL, DIMENSION(numdis) :: disvol

where **numdis** is the number of discharge locations and

disvol Discharged volume [m^3/s].

30.11.3 Discharge area and direction

purpose : discharge area and direction
 time coordinate : yes
 key id : io_discur
 file number : 1
 sub-variables : no

First forcing variable is the **time** coordinate

CHARACTER (LEN=lentime) :: time

Forcing data are

REAL, DIMENSION(numdis) :: disarea

REAL, DIMENSION(numdis) :: disdir

where **numdis** is the number of discharge locations and

disarea Area over which the discharge takes place [m^2].

disdir Discharge direction with respect to the X-axis [rad].

30.11.4 Discharged salinity

purpose : salinity concentration of the discharged volume
 time coordinate : yes
 key id : io_dissal
 file number : 1
 sub-variables : no

First forcing variable is the `time` coordinate

```
CHARACTER (LEN=lentime) :: time
```

Forcing data are

```
REAL, DIMENSION(numdis) :: dissal
```

where `numdis` are the number of discharge locations and

`dissal` Discharged salinity [PSU].

30.11.5 Discharged temperature

purpose : temperature of the discharged volume
 time coordinate : yes
 key id : io_distmp
 file number : 1
 sub-variables : no

First forcing variable is the `time` coordinate

```
CHARACTER (LEN=lentime) :: time
```

Forcing data are

```
REAL, DIMENSION(numdis) :: distmp
```

where `numdis` is the number of discharge locations and

`distmp` Temperature of the discharged volume [$^{\circ}\text{C}$].

30.11.6 Discharged sediment concentrations

purpose : sediment concentration of the discharged volume
 time coordinate : yes
 key id : io_dissed
 file number : 1
 sub-variables : yes

The first forcing variable is the time coordinate

```
CHARACTER (LEN=lentime) :: time
```

Forcing data are

```
REAL, DIMENSION(numdis,nosubvars) :: dissed
```

where numdis are the number of discharge locations.

The two following variable attributes must be defined

nosubvars The number of discharged sediment fractions which must be lower
 than the number of suspended fractions nfsus.
 subindex Vector of size nosubvars containing the fraction indices of the dis-
 charged sediments.

30.12 Particle model

30.12.1 Particle clouds

purpose : location of particle clouds
 time coordinate : yes
 key id : io_parclid
 file number : 1,...,noclouds
 sub-variables : no

First forcing variable is the time coordinate

```
CHARACTER (LEN=lentime) :: time
```

Forcing data are

```
DOUBLE PRECISION, DIMENSION(3) :: clouddata
```

where

- disdata Coordinates of the centre of the cloud. The vector components represent
- 1: X-coordinate in double precision [m or degrees longitude, positive East].
 - 2: Y-coordinate in double precision [m or degrees latitude, positive North].
 - 3: Z-coordinate in double precision. The format and unit depend on the value of `cloud(icld)%kdistype`. No value needs to be given if the cloud contains only surface floating particles (`cloud(icld)%state=1`).

Note that the data must be provided for each cloud. Values for different clouds are stored in different files with the same forcing key id but a different file number.

30.12.2 Particle model grid

purpose : model grid arrays for the particle module
 time coordinate : no
 key id : `io_pargrd`
 file number : 1
 sub-variables : no

The forcing arrays are used for off-line simulations with the particle model (`iopt_part_model=3,4`). The usual procedure is to create a standard COHERENS forcing file from a previous run with COHERENS. If the user prefers to use an alternative physical model, the grid data can be provided by creating a standard non-COHERENS data file, using the prescriptions discussed in Chapter 29.

Forcing data are

```

! used when
REAL, DIMENSION(nc, nr) :: gxcoord
REAL, DIMENSION(nc, nr) :: gycoord
REAL, DIMENSION(nz+1) :: gsigcoordatw      ! iopt_grid_nodim=3,
                                                iopt_grid_vtype=1,2
REAL, DIMENSION(nc-1, nr-1, nz) :: gscoordatc ! iopt_grid_nodim=3
                                                iopt_grid_vtype=3
REAL, DIMENSION(nc, nr-1, nz) :: gscoordatu   ! iopt_grid_nodim=3,
                                                iopt_grid_vtype=3
REAL, DIMENSION(nc-1, nr, nz) :: gscoordatv   ! iopt_grid_nodim=3,

```

```

          iopt_grid_vtype=3
REAL, DIMENSION(nc-1, nr-1, nz+1) :: gscoordatw ! iopt_grid_nodim=3,
                                               iopt_grid_vtype=3
REAL, DIMENSION(nc-1, nr-1) :: delxatc
REAL, DIMENSION(nc-1, nr-1) :: delyatc
REAL, DIMENSION(nc-1, nr-1) :: depmeanatc
REAL, DIMENSION(nc, nr-1) :: depmeanatu
REAL, DIMENSION(nc-1, nr) :: depmeanatv
LOGICAL, DIMENSION(nc-1, nr-1) :: maskatc
LOGICAL, DIMENSION(nc, nr-1) :: maskatu
LOGICAL, DIMENSION(nc-1, nr) :: maskatv
REAL :: gacc_mean
REAL :: density_ref

```

where

gxcoord	X-coordinates of the model grid at corner nodes [m or degrees longitude, positive east].
gycoord	Y-coordinates of the model grid at corner nodes [m or degrees latitude, positive North].
gsigcoordatw	Sigma coordinates at W-nodes in case a vertically non-uniform, but horizontally uniform grid is selected (iopt_grid_vtype=2,3). If iopt_grid_vtype_transf equals 11,12 or 13, the array is not needed here since the vertical grid is constructed by the program using one of the vertical coordinates transformations described in Section 4.1.4.1.
gscoordatc	Sigma coordinates at C-nodes in case a non-uniform horizontal grid is selected and iopt_grid_vtype_transf=0.
gscoordatu	Sigma coordinates at U-nodes in case a non-uniform horizontal grid is selected and iopt_grid_vtype_transf=0.
gscoordatv	Sigma coordinates at V-nodes in case a non-uniform horizontal grid is selected and iopt_grid_vtype_transf=0.
gscoordatw	Sigma coordinates at W-nodes in case a non-uniform horizontal grid is selected and iopt_grid_vtype_transf=0.
delxatc	Grid spacings at the C-nodes in the X-direction in case of a non-uniform horizontal grid [m].
delyatc	Grid spacings at the C-nodes in the Y-direction in case of a non-uniform horizontal grid [m].
depmeanatc	Mean water depths at C-nodes [m].

depmeanatu	Mean water depths at U-nodes [m].
depmeanatv	Mean water depths at V-nodes [m].
maskatc	Grid mask (.TRUE. at sea and .FALSE. at land C-node grid cells).
maskatu	Grid mask (.TRUE. at sea and .FALSE. at land/coastal U-node cell faces).
maskatv	Grid mask (.TRUE. at sea and .FALSE. at land/coastal land V-node cell faces).
density_ref	Uniform reference density [kg/m ³].
gacc_mean	Acceleration of gravity (taken as uniform in the horizontal) [m ² /s].

30.12.3 Physical data for the particle module

purpose	: physical data needed for the particle module
time coordinate	: yes
key id	: io_parphs
file number	: 1
sub-variables	: no

The forcing arrays are used for off-line simulations with the particle model (iopt_part_model=3,4). The usual procedure is to create a standard COHERENS forcing file from a previous run with COHERENS. If the user prefers to use an alternative physical model, the grid data can be provided by creating a standard non-COHERENS data file, using the prescriptions discussed in Chapter 29.

First forcing variable is the time coordinate

```
CHARACTER (LEN=lentime) :: time
```

Forcing data are

```
! used when
REAL, DIMENSION(nc-1, nr-1) :: deptotatc
REAL, DIMENSION(nc, nr-1) :: deptotatu
REAL, DIMENSION(nc-1, nr-1) :: deptotatv
REAL, DIMENSION(nc, nr-1, nz) :: uvel
REAL, DIMENSION(nc-1, nr, nz) :: vvel
REAL, DIMENSION(nc-1, nr-1, nz+1) :: wvel ! iopt_grid_nodim=3
REAL, DIMENSION(nc-1, nr-1, nz+1) :: vdifcoefpart ! iopt_part_vdif=2
```

```

REAL, DIMENSION(nc-1, nr-1, nz) :: dens           ! iopt_part_dens=2
LOGICAL, DIMENSION(nc-1, nr-1) :: maskatc        ! iopt_fld=1
LOGICAL, DIMENSION(nc, nr-1) :: maskatu          ! iopt_fld=1
LOGICAL, DIMENSION(nc-1, nr) :: maskatv          ! iopt_fld=1

```

where

1. Always

deptotatc Mean water depth at C-nodes [m].
deptotatu Mean water depth at U-nodes [m].
deptotatv Mean water depth at V-nodes [m].
uvel X-component of the current [m].
vvel Y-component of the current [m].

2. 3-D case (**iopt_grid_nodim**=3)

wvel Transformed vertical current ω at W-nodes [m/s]. Bottom and surface values must be zero.

3. 3-D case using random walk method for vertical diffusion and an external diffusion coefficient (**iopt_part_vdif**=2)

vdifcoefpart Vertical diffusion coefficient at W-nodes [m²/s].

4. 3-D case including buoyancy with a non-uniform density field (**iopt_part_dens**=2)

dens Water density at C-nodes [kg/m³].

5. With drying/wetting scheme activated (**iopt_fld** = 1)

maskatc Grid mask (.TRUE. at sea and .FALSE. at dry/land C-node grid cells).
maskatu Grid mask (.TRUE. at sea and .FALSE. at dry/land/coastal U-node cell faces).
maskatv Grid mask (.TRUE. at sea and .FALSE. at dry/land/coastal V-node cell faces).

Appendix A

Transformed model equations

A.1 Horizontal transformation

The continuity, momentum and scalar transport equations can be rewritten from Cartesian to orthogonal curvilinear coordinates (ξ_1, ξ_2) with the aid of the general transformation rules (e.g. Batchelor, 1979):

$$\nabla_h = \left(\frac{1}{h_1} \frac{\partial}{\partial \xi_1}, \frac{1}{h_2} \frac{\partial}{\partial \xi_2} \right) \quad (\text{A.1})$$

$$\nabla_h \cdot \mathbf{F}_h = \frac{1}{h_1 h_2} \left[\frac{\partial}{\partial \xi_1} (h_2 F_1) + \frac{\partial}{\partial \xi_2} (h_1 F_2) \right] \quad (\text{A.2})$$

$$\begin{aligned} \mathbf{F}_h \cdot \nabla_h \mathbf{F}_h = & \left[\mathbf{F}_h \cdot \nabla F_1 + \frac{F_2}{h_1 h_2} \left(F_1 \frac{\partial h_1}{\partial \xi_2} - F_2 \frac{\partial h_2}{\partial \xi_1} \right), \right. \\ & \left. \mathbf{F}_h \cdot \nabla F_2 + \frac{F_1}{h_1 h_2} \left(F_2 \frac{\partial h_2}{\partial \xi_1} - F_1 \frac{\partial h_1}{\partial \xi_2} \right) \right] \end{aligned} \quad (\text{A.3})$$

where the subscript h denotes the horizontal component of the associated vector or operator and h_1, h_2 are the metric coefficients defined by (4.7). Substituting the above relations into (5.1)–(5.3) and (5.5) or (5.6) one obtains

$$\frac{1}{h_1 h_2} \left[\frac{\partial}{\partial \xi_1} (h_2 u) + \frac{\partial}{\partial \xi_2} (h_1 v) \right] + \frac{\partial w}{\partial z} = 0 \quad (\text{A.4})$$

$$\begin{aligned} \frac{\partial u}{\partial t} + \frac{u}{h_1} \frac{\partial u}{\partial \xi_1} + \frac{v}{h_2} \frac{\partial u}{\partial \xi_2} + w \frac{\partial u}{\partial z} + \frac{v}{h_1 h_2} \left(u \frac{\partial h_1}{\partial \xi_2} - v \frac{\partial h_2}{\partial \xi_1} \right) - 2\Omega v \sin \phi \\ = -\frac{g}{h_1} \frac{\partial \zeta}{\partial \xi_1} - \frac{1}{\rho_o h_1} \frac{\partial P_a}{\partial \xi_1} - \frac{1}{h_1} \frac{\partial q}{\partial \xi_1} + F_1^t + \frac{\partial}{\partial z} \left(\nu_T \frac{\partial u}{\partial z} \right) \end{aligned}$$

$$+ \mathcal{D}_{mh1}^*(\tau_{11}) + \mathcal{D}_{mh2}^*(\tau_{12}) \quad (A.5)$$

$$\begin{aligned} \frac{\partial v}{\partial t} + \frac{u}{h_1} \frac{\partial v}{\partial \xi_1} + \frac{v}{h_2} \frac{\partial v}{\partial \xi_2} + w \frac{\partial v}{\partial z} + \frac{u}{h_1 h_2} \left(v \frac{\partial h_2}{\partial \xi_1} - u \frac{\partial h_1}{\partial \xi_2} \right) + 2\Omega u \sin \phi \\ = -\frac{g}{h_1} \frac{\partial \zeta}{\partial \xi_2} - \frac{1}{\rho_o h_2} \frac{\partial P_a}{\partial \xi_2} - \frac{1}{h_2} \frac{\partial q}{\partial \xi_2} + F_2^t + \frac{\partial}{\partial z} \left(\nu_T \frac{\partial v}{\partial z} \right) \\ + \mathcal{D}_{mh1}^*(\tau_{21}) + \mathcal{D}_{mh2}^*(\tau_{22}) \end{aligned} \quad (A.6)$$

$$\begin{aligned} \frac{\partial \psi}{\partial t} + \frac{u}{h_1} \frac{\partial \psi}{\partial \xi_1} + \frac{v}{h_2} \frac{\partial \psi}{\partial \xi_2} + w \frac{\partial \psi}{\partial z} = \mathcal{S}(\psi) + \frac{\partial}{\partial z} \left(\lambda_T \frac{\partial \psi}{\partial z} \right) \\ + \frac{1}{h_1 h_2} \left[\frac{\partial}{\partial \xi_1} \left(\lambda_H \frac{h_2}{h_1} \frac{\partial \psi}{\partial \xi_1} \right) + \frac{\partial}{\partial \xi_2} \left(\lambda_H \frac{h_1}{h_2} \frac{\partial \psi}{\partial \xi_2} \right) \right] \end{aligned} \quad (A.7)$$

The horizontal diffusion operators for momentum are defined by (Pacanowski & Griffies, 2000)

$$\mathcal{D}_{mh1}^*(F) = \frac{1}{h_1 h_2^2} \frac{\partial}{\partial \xi_1} (h_2^2 F) \quad (A.8)$$

$$\mathcal{D}_{mh2}^*(F) = \frac{1}{h_1^2 h_2} \frac{\partial}{\partial \xi_2} (h_1^2 F) \quad (A.9)$$

A.2 Vertical transformation

A general vertical coordinate is defined through the transformation

$$(\xi_1, \xi_2, z, t) \longrightarrow (\tilde{\xi}_1, \tilde{\xi}_2, s, \tilde{t}) \quad (A.10)$$

with $\tilde{\xi}_i = \xi_i$, $\tilde{t} = t$ and $s = f(\xi_1, \xi_2, z, t)$ where, as stated in Section 4.1.4.3, the transformed vertical coordinate s is defined by normalising the σ -coordinate, using (4.40) such that (A.13) is valid. Spatial and time derivatives are transformed by applying the chain rule

$$\frac{\partial}{\partial t} = \frac{\partial}{\partial \tilde{t}} + \frac{\partial s}{\partial t} \frac{\partial}{\partial s} \quad (A.11)$$

$$\frac{\partial}{\partial \xi_i} = \frac{\partial}{\partial \tilde{\xi}_i} + \frac{\partial s}{\partial \xi_i} \frac{\partial}{\partial s} \quad (A.12)$$

$$\frac{\partial}{\partial z} = \frac{1}{h_3} \frac{\partial}{\partial s} \quad (A.13)$$

where the derivatives on the left hand side in the first two relations are taken along constant z -surfaces and the first ones on the right side along constant s -surfaces. The following useful relations can be derived from (A.11)–(A.13)

$$\frac{\partial s}{\partial z} = \frac{1}{h_3}, \quad \frac{\partial z}{\partial s} = h_3 \quad (\text{A.14})$$

$$\frac{\partial z}{\partial \tilde{\xi}_i} + h_3 \frac{\partial s}{\partial \xi_i} = 0 \quad (\text{A.15})$$

$$\frac{\partial h_3}{\partial \tilde{t}} + \frac{\partial s}{\partial t} \frac{\partial h_3}{\partial s} = 0 \quad (\text{A.16})$$

$$\frac{\partial h_3}{\partial \tilde{\xi}_i} = \frac{\partial}{\partial \tilde{\xi}_i} \left(\frac{\partial z}{\partial s} \right) = \frac{\partial}{\partial s} \left(\frac{\partial z}{\partial \tilde{\xi}_i} \right) = -\frac{\partial}{\partial s} \left(h_3 \frac{\partial s}{\partial \xi_i} \right) \quad (\text{A.17})$$

$$\frac{\partial z}{\partial \tilde{t}} = -\frac{\partial s}{\partial t} \frac{\partial z}{\partial s} = -h_3 \frac{\partial s}{\partial t} \quad (\text{A.18})$$

A new vertical velocity is defined by

$$\begin{aligned} \omega &= h_3 \frac{ds}{dt} \\ &= h_3 \left(\frac{\partial s}{\partial t} + \frac{u}{h_1} \frac{\partial s}{\partial \xi_1} + \frac{v}{h_2} \frac{\partial s}{\partial \xi_2} + w \frac{\partial s}{\partial z} \right) \\ &= h_3 \left(\frac{\partial s}{\partial t} + \frac{u}{h_1} \frac{\partial s}{\partial \xi_1} + \frac{v}{h_2} \frac{\partial s}{\partial \xi_2} \right) + w \end{aligned} \quad (\text{A.19})$$

from which (5.30) is obtained.

The continuity equation (A.4) is rewritten in the transformed coordinate system with the aid of the previous relations

$$\begin{aligned} 0 &= \frac{1}{h_1 h_2} \left[\frac{\partial}{\partial \xi_1} (h_2 u) + \frac{\partial}{\partial \xi_2} (h_1 v) \right] + \frac{\partial w}{\partial z} \\ &= \frac{1}{h_1 h_2} \left[\frac{\partial}{\partial \tilde{\xi}_1} (h_2 u) + \frac{\partial}{\partial \tilde{\xi}_2} (h_1 v) + \frac{\partial s}{\partial \xi_1} \frac{\partial}{\partial s} (u h_2) + \frac{\partial s}{\partial \xi_2} \frac{\partial}{\partial s} (v h_1) \right] + \frac{1}{h_3} \frac{\partial w}{\partial s} \\ &= \frac{1}{h_1 h_2 h_3} \left[\frac{\partial}{\partial \tilde{\xi}_1} (h_2 h_3 u) + \frac{\partial}{\partial \tilde{\xi}_2} (h_1 h_3 v) + h_3 \frac{\partial s}{\partial \xi_1} \frac{\partial}{\partial s} (u h_2) - u h_2 \frac{\partial h_3}{\partial \tilde{\xi}_1} \right. \\ &\quad \left. + h_3 \frac{\partial s}{\partial \xi_2} \frac{\partial}{\partial s} (v h_1) - v h_1 \frac{\partial h_3}{\partial \tilde{\xi}_2} \right] + \frac{1}{h_3} \frac{\partial w}{\partial s} \\ &= \frac{1}{h_1 h_2 h_3} \left[\frac{\partial}{\partial \tilde{\xi}_1} (h_2 h_3 u) + \frac{\partial}{\partial \tilde{\xi}_2} (h_1 h_3 v) + h_3 \frac{\partial s}{\partial \xi_1} \frac{\partial}{\partial s} (u h_2) + u h_2 \frac{\partial}{\partial s} \left(h_3 \frac{\partial s}{\partial \xi_1} \right) \right. \\ &\quad \left. + h_3 \frac{\partial s}{\partial \xi_2} \frac{\partial}{\partial s} (v h_1) + v h_1 \frac{\partial}{\partial s} \left(h_3 \frac{\partial s}{\partial \xi_2} \right) \right] + \frac{1}{h_3} \frac{\partial w}{\partial s} \end{aligned}$$

$$\begin{aligned}
&= \frac{1}{h_1 h_2 h_3} \left[\frac{\partial}{\partial \tilde{\xi}_1} (h_2 h_3 u) + \frac{\partial}{\partial \tilde{\xi}_2} (h_1 h_3 v) + \frac{\partial}{\partial s} \left(h_2 h_3 u \frac{\partial s}{\partial \xi_1} + h_1 h_3 v \frac{\partial s}{\partial \xi_2} \right) \right] + \frac{1}{h_3} \frac{\partial w}{\partial s} \\
&= \frac{1}{h_1 h_2 h_3} \left[\frac{\partial}{\partial \tilde{\xi}_1} (h_2 h_3 u) + \frac{\partial}{\partial \tilde{\xi}_2} (h_1 h_3 v) \right] + \frac{1}{h_3} \frac{\partial}{\partial s} \left[w + \frac{h_3}{h_1} u \frac{\partial s}{\partial \xi_1} + \frac{h_3}{h_2} v \frac{\partial s}{\partial \xi_2} \right] \\
&= \frac{1}{h_1 h_2 h_3} \left[\frac{\partial}{\partial \tilde{\xi}_1} (h_2 h_3 u) + \frac{\partial}{\partial \tilde{\xi}_2} (h_1 h_3 v) \right] + \frac{1}{h_3} \frac{\partial}{\partial s} \left(\omega - h_3 \frac{\partial s}{\partial t} \right) \\
&= \frac{1}{h_1 h_2 h_3} \left[\frac{\partial}{\partial \tilde{\xi}_1} (h_2 h_3 u) + \frac{\partial}{\partial \tilde{\xi}_2} (h_1 h_3 v) \right] + \frac{1}{h_3} \frac{\partial \omega}{\partial s} - \frac{1}{h_3} \frac{\partial h_3}{\partial s} \frac{\partial s}{\partial t} - \frac{\partial}{\partial s} \left(\frac{\partial s}{\partial t} \right) \\
&= \frac{1}{h_3} \frac{\partial h_3}{\partial \tilde{t}} + \frac{1}{h_1 h_2 h_3} \left[\frac{\partial}{\partial \tilde{\xi}_1} (h_2 h_3 u) + \frac{\partial}{\partial \tilde{\xi}_2} (h_1 h_3 v) \right] + \frac{1}{h_3} \frac{\partial \omega}{\partial s}
\end{aligned} \tag{A.20}$$

which becomes identical to (5.18) by letting $\tilde{\xi}_i = \xi_i$ and $\tilde{t} = t$.

The physical vertical current is given by

$$\begin{aligned}
w &= \frac{dz}{dt} = \frac{\partial z}{\partial \tilde{t}} + \frac{u}{h_1} \frac{\partial z}{\partial \tilde{\xi}_1} + \frac{v}{h_2} \frac{\partial z}{\partial \tilde{\xi}_2} + \frac{\omega}{h_3} \frac{\partial z}{\partial s} \\
&= \frac{\partial z}{\partial \tilde{t}} + \frac{u}{h_1} \frac{\partial z}{\partial \tilde{\xi}_1} + \frac{v}{h_2} \frac{\partial z}{\partial \tilde{\xi}_2} + \omega
\end{aligned} \tag{A.21}$$

Equation (5.31) is recovered by adding (A.21) and z times (A.20)

$$\begin{aligned}
w &= \frac{\partial z}{\partial \tilde{t}} + \frac{u}{h_1} \frac{\partial z}{\partial \tilde{\xi}_1} + \frac{v}{h_2} \frac{\partial z}{\partial \tilde{\xi}_2} + \omega \\
&\quad + \frac{z}{h_3} \left[\frac{1}{h_1 h_2} \left(\frac{\partial}{\partial \tilde{\xi}_1} (h_2 h_3 u) + \frac{\partial}{\partial \tilde{\xi}_2} (h_1 h_3 v) \right) + \frac{\partial \omega}{\partial s} + \frac{\partial h_3}{\partial \tilde{t}} \right] \\
&= \frac{1}{h_3} \left[\omega \frac{\partial z}{\partial s} + z \frac{\partial \omega}{\partial s} + h_3 \frac{\partial z}{\partial \tilde{t}} + z \frac{\partial h_3}{\partial \tilde{t}} + \frac{u h_3}{h_1} \frac{\partial z}{\partial \tilde{\xi}_1} + \frac{z}{h_1 h_2} \frac{\partial}{\partial \tilde{\xi}_1} (h_2 h_3 u) \right. \\
&\quad \left. + \frac{v h_3}{h_2} \frac{\partial z}{\partial \tilde{\xi}_2} + \frac{z}{h_1 h_2} \frac{\partial}{\partial \tilde{\xi}_2} (h_1 h_3 v) \right] \\
&= \frac{1}{h_3} \left[\frac{\partial}{\partial \tilde{t}} (h_3 z) + \frac{1}{h_1 h_2} \frac{\partial}{\partial \tilde{\xi}_1} (h_2 h_3 u z) + \frac{1}{h_1 h_2} \frac{\partial}{\partial \tilde{\xi}_2} (h_1 h_3 v z) + \frac{\partial}{\partial s} (\omega z) \right]
\end{aligned} \tag{A.22}$$

The total derivative of a quantity ψ (velocity component or scalar) transforms according to

$$\begin{aligned}
\frac{d\psi}{dt} &= \frac{\partial \psi}{\partial t} + \frac{u}{h_1} \frac{\partial \psi}{\partial \xi_1} + \frac{v}{h_2} \frac{\partial \psi}{\partial \xi_2} + w \frac{\partial \psi}{\partial z} \\
&= \frac{\partial \psi}{\partial \tilde{t}} + \frac{u}{h_1} \frac{\partial \psi}{\partial \tilde{\xi}_1} + \frac{v}{h_2} \frac{\partial \psi}{\partial \tilde{\xi}_2} + \frac{\partial \psi}{\partial s} \left(\frac{\partial s}{\partial \tilde{t}} + \frac{u}{h_1} \frac{\partial s}{\partial \tilde{\xi}_1} + \frac{v}{h_2} \frac{\partial s}{\partial \tilde{\xi}_2} + \frac{w}{h_3} \right)
\end{aligned}$$

$$\begin{aligned}
&= \frac{\partial \psi}{\partial \tilde{t}} + \frac{u}{h_1} \frac{\partial \psi}{\partial \tilde{\xi}_1} + \frac{v}{h_2} \frac{\partial \psi}{\partial \tilde{\xi}_2} + \frac{\omega}{h_3} \frac{\partial \psi}{\partial s} \\
&= \frac{1}{h_3} \frac{\partial}{\partial \tilde{t}} (h_3 \psi) + \frac{1}{h_1 h_2 h_3} \left[\frac{\partial}{\partial \tilde{\xi}_1} (h_2 h_3 u \psi) + \frac{\partial}{\partial \tilde{\xi}_2} (h_1 h_3 v \psi) \right] + \frac{1}{h_3} \frac{\partial}{\partial s} (\psi \omega) \\
&\quad - \frac{\psi}{h_3} \frac{\partial h_3}{\partial \tilde{t}} - \frac{\psi}{h_1 h_2 h_3} \left[\frac{\partial}{\partial \tilde{\xi}_1} (h_2 h_3 u) + \frac{\partial}{\partial \tilde{\xi}_2} (h_1 h_3 v) \right] - \frac{\psi}{h_3} \frac{\partial \omega}{\partial s} \\
&= \frac{1}{h_3} \frac{\partial}{\partial \tilde{t}} (h_3 \psi) + \frac{1}{h_1 h_2 h_3} \left[\frac{\partial}{\partial \tilde{\xi}_1} (h_2 h_3 u \psi) + \frac{\partial}{\partial \tilde{\xi}_2} (h_1 h_3 v \psi) \right] + \frac{1}{h_3} \frac{\partial}{\partial s} (\psi \omega)
\end{aligned} \tag{A.23}$$

by virtue of (A.20).

The horizontal gradient of a vertically independent quantity obviously does not change. For a 3-D quantity one has

$$\begin{aligned}
\frac{1}{h_i} \frac{\partial \psi}{\partial \xi_i} &= \frac{1}{h_i} \left[\frac{\partial \psi}{\partial \tilde{\xi}_i} + \frac{\partial s}{\partial \xi_i} \frac{\partial \psi}{\partial s} \right] \\
&= \frac{1}{h_i} \left[\frac{1}{h_3} \frac{\partial}{\partial \tilde{\xi}_i} (h_3 \psi) - \frac{\psi}{h_3} \frac{\partial h_3}{\partial \tilde{\xi}_i} + \frac{\partial s}{\partial \xi_i} \frac{\partial \psi}{\partial s} \right] \\
&= \frac{1}{h_i} \left[\frac{1}{h_3} \frac{\partial}{\partial \tilde{\xi}_i} (h_3 \psi) + \frac{\psi}{h_3} \frac{\partial}{\partial s} \left(h_3 \frac{\partial s}{\partial \tilde{\xi}_i} \right) + \frac{\partial s}{\partial \xi_i} \frac{\partial \psi}{\partial s} \right] \\
&= \frac{1}{h_i} \left[\frac{1}{h_3} \frac{\partial}{\partial \tilde{\xi}_i} (h_3 \psi) + \frac{1}{h_3} \frac{\partial}{\partial s} \left(h_3 \psi \frac{\partial s}{\partial \tilde{\xi}_i} \right) \right] \\
&= \frac{1}{h_i h_3} \left[\frac{\partial}{\partial \tilde{\xi}_i} (h_3 \psi) - \frac{\partial}{\partial s} \left(\psi \frac{\partial z}{\partial \tilde{\xi}_i} \right) \right]
\end{aligned} \tag{A.24}$$

from which (5.36) is obtained with $\psi = q$.

Applying the previous rule for the horizontal diffusion terms in the momentum and scalar transport equations one recovers the definitions (5.25), (5.26), (5.40) and (5.41) by making the assumption that diffusion takes place along constant s -surfaces.

Appendix B

Solutions of the RANS equations

B.1 Non-equilibrium method

Using the definitions (5.140) and (5.142), the RANS equations (5.139) can be reduced to two linear equations for the stability functions S_u and S_b . Firstly, the following auxiliary parameters are defined

$$\begin{aligned}
 \alpha_1 &= 1 - c_{21} - c_{23} + \frac{3}{2}c_{22} \\
 \alpha_2 &= (1 - c_{21})^2 + c_{23}(4 - 4c_{21} + c_{23}) \\
 \alpha_3 &= 3 - c_{21} + 2c_{23} - 2c_3 \\
 \alpha_4 &= c_{23}(3 - 2c_{21} + c_{23} - c_3) \\
 \alpha_5 &= \alpha_1 - \frac{\alpha_2}{c_1} \\
 \alpha_6 &= (1 - c_{21})(3 - c_{21} + 2c_{23} - 2c_3) + \alpha_4 \\
 R_\beta^* &= R_\beta(1 - c_{3\beta})
 \end{aligned} \tag{B.1}$$

The stability functions are then obtained as solution of

$$\begin{aligned}
 &\left[c_1^2 c_{1\beta} + \frac{2}{3} c_{1\beta} \alpha_2 \alpha_M + c_1 (1 - c_3) \alpha_N \right] S_u \\
 &+ (1 - c_3) \left[\frac{2}{3} c_{1\beta} (2 - 2c_{21} + c_{23}) + c_1 (1 - c_{21\beta}) \right] \alpha_N S_b = \frac{2}{3} c_1 c_{1\beta} \alpha_1
 \end{aligned} \tag{B.2}$$

$$\left[\frac{2}{3} c_{1\beta} (1 - c_{21} + 2c_{23}) - c_{22\beta} c_1 \right] \alpha_M S_u + \left[c_1 c_{1\beta}^2 - c_1 c_{22\beta} (1 - c_{21\beta}) \alpha_M \right]$$

$$+ 2c_{1\beta} \left(c_1 R_\beta^* + \frac{2}{3}(1 - c_3) \right) \alpha_N \Big] S_b = \frac{2}{3} c_1 c_{1\beta} \quad (B.3)$$

The solution can be written in the form, given by (5.143). The coefficients C_{a1} – C_{a11} take the following values

$$\begin{aligned} C_{a1} &= \frac{2\alpha_1}{3c_1} \\ C_{a2} &= -\frac{2\alpha_1(1 - c_{21\beta})c_{22\beta}}{3c_1 c_{1\beta}^2} \\ C_{a3} &= -\frac{2}{3c_1 c_{1\beta}} \left[(1 - c_3) \left(\frac{2(c_{23} - c_{22})}{c_1} + \frac{1 - c_{21\beta}}{c_{1\beta}} \right) - 2R_\beta^* \alpha_1 \right] \\ C_{a4} &= \frac{2}{3c_{1\beta}} \\ C_{a5} &= \frac{2}{3c_1 c_{1\beta}} \left[\frac{2}{3c_1} \left(\alpha_2 - \alpha_1(1 - c_{21} + 2c_{23}) \right) + \frac{c_{22\beta}}{c_{1\beta}} \alpha_1 \right] \\ C_{a6} &= \frac{2(1 - c_3)}{3c_1 c_{1\beta}^2} \\ C_{a7} &= \frac{2\alpha_2}{3c_1^2} - \frac{c_{22\beta}(1 - c_{21\beta})}{c_{1\beta}^2} \\ C_{a8} &= \frac{1}{c_{1\beta}} \left[\frac{7(1 - c_3)}{3c_1} + 2R_\beta^* \right] \\ C_{a9} &= -\frac{2c_{22\beta}(1 - c_{21\beta})\alpha_2}{3c_1^2 c_{1\beta}^2} \\ C_{a10} &= \frac{2}{3c_1^2 c_{1\beta}} \left\{ (1 - c_3) \left[\frac{2c_{23}}{c_1} (1 - c_{21}) \right. \right. \\ &\quad \left. \left. + \frac{1}{c_{1\beta}} \left(c_{22\beta}(2 - 2c_{21} + c_{23}) - (1 - c_{21\beta})(1 - c_{21} + 2c_{23}) \right) \right] + 2R_\beta^* \alpha_2 \right\} \\ C_{a11} &= \frac{2(1 - c_3)}{c_1 c_{1\beta}^2} \left[\frac{2(1 - c_3)}{3c_1} + R_\beta^* \right] \end{aligned} \quad (B.4)$$

B.2 Quasi-equilibrium method

The governing equations for the algebraic RANS model are given by (5.147) supplemented by the last 7 equations of (5.139). In analogy with the previous

case they can be reduced to the following equations for S_u and S_b .

$$\begin{aligned} c_1[c_1c_{1\beta} + (1 - c_3)\alpha_N]S_u + \left[\frac{2}{3}c_{1\beta}\alpha_6 + c_1(1 - c_3)(1 - c_{21\beta}) \right] \alpha_N S_b \\ = \frac{2}{3}c_1c_{1\beta}\alpha_5 \end{aligned} \quad (\text{B.5})$$

$$\begin{aligned} \left[c_1c_{1\beta}^2 - c_1(1 - c_{21\beta})c_{22\beta}\alpha_M + (2c_1c_{1\beta}R_\beta^* + \frac{2}{3}c_{1\beta}\alpha_3 - c_1c_{22\beta})\alpha_N \right] S_b \\ = \frac{2}{3}c_{1\beta}(c_1 - 1 + c_{21} - 2c_{23}) + c_1c_{22\beta} \end{aligned} \quad (\text{B.6})$$

If $c_{22\beta} = 0$, the solutions can be cast in the form (5.148) where

$$\begin{aligned} C_{b1} &= \frac{2\alpha_5}{3c_1} \\ C_{b2} &= \frac{2}{3c_1c_{1\beta}} \left[\frac{2}{3c_1}(\alpha_3\alpha_5 - \alpha_6) + \frac{2\alpha_6}{3c_1^2}(1 - c_{21} + 2c_{23}) \right. \\ &\quad \left. - \frac{(1 - c_3)(1 - c_{21\beta})}{c_{1\beta}} \left(1 - \frac{1}{c_1}(1 - c_{21} + 2c_{23}) \right) + 2\alpha_5R_\beta^* \right] \\ C_{b3} &= \frac{2}{c_{1\beta}} \left(R_\beta^* + \frac{\alpha_3}{3c_1} \right) \\ C_{b4} &= \frac{1 - c_3}{c_1c_{1\beta}} \\ C_{b5} &= \frac{2}{3c_{1\beta}} \left(1 - \frac{1}{c_1}(1 - c_{21} + 2c_{23}) \right) \end{aligned} \quad (\text{B.7})$$

If $c_{22\beta} \neq 0$ the solutions are given by (5.149)–(5.150) with

$$\begin{aligned} C_{c1} &= \frac{2\alpha_5}{3c_1} \\ C_{c2} &= -\frac{1}{c_1} \left[\frac{2\alpha_6}{3c_1} + \frac{(1 - c_3)(1 - c_{21\beta})}{c_{1\beta}} \right] \\ C_{c3} &= \frac{1 - c_3}{c_1c_{1\beta}} \\ C_{c4} &= -\left[\frac{2\alpha_6}{3c_1^2} + \frac{1 - c_{21\beta}}{c_{1\beta}} \left(\frac{c_{22\beta}}{c_{1\beta}} + \frac{1 - c_3}{c_1} \right) \right] \\ C_{c5} &= -\frac{2}{3c_1c_{1\beta}} \left[\frac{\alpha_6}{c_1} \left(2R_\beta^* + \frac{2\alpha_3}{3c_1} - \frac{c_{22\beta}}{c_{1\beta}} \right) + \frac{(1 - c_3)(1 - c_{21\beta})}{c_{1\beta}} \left(3R_\beta^* + \frac{\alpha_3}{c_1} \right) \right] \\ C_{c6} &= \frac{2\alpha_5}{3c_1} - \frac{(1 - c_{21\beta})c_{22\beta}}{c_{1\beta}^2} \end{aligned}$$

$$\begin{aligned}
C_{c7} &= \frac{2}{3c_1 c_{1\beta}} \left[\frac{2}{3} \left(\frac{\alpha_3 \alpha_5 + \alpha_6}{c_1} - \frac{\alpha_6 (1 - c_{21} + 2c_{23})}{c_1^2} \right) \right. \\
&\quad \left. + \frac{(1 - c_3)(1 - c_{21\beta})}{c_{1\beta}} \left(1 - \frac{1 - c_{21} + 2c_{23}}{c_1} \right) - \frac{c_{22\beta}}{c_{1\beta}} \left(\alpha_5 - \frac{\alpha_6}{c_1} \right) + 2\alpha_5 R_\beta^* \right] \\
C_{c8} &= -\frac{2\alpha_5}{3c_1 c_{1\beta}} \left(\frac{2}{3} - \frac{2}{3c_1} (1 - c_{21} + 2c_{23}) + \frac{c_{22\beta}}{c_{1\beta}} \right)
\end{aligned} \tag{B.8}$$

B.3 Equilibrium method

The coefficients in the quadratic equation (5.156) for κ^2 are obtained from equations (B.5)–(B.6) and the equilibrium relation $S_u \alpha_M - S_b \alpha_N = 1$. One has

$$\begin{aligned}
C_{d1} &= -\frac{2\alpha_5}{3c_1} - \frac{(1 - c_{21\beta})c_{22\beta}}{c_{1\beta}^2} \\
C_{d2} &= \frac{1}{c_{1\beta}} \left(\frac{2}{3} + \frac{7}{3c_1} (1 - c_3) + 2R_\beta^* \right) \\
C_{d3} &= \frac{2(1 - c_{21\beta})c_{22\beta}\alpha_5}{3c_1 c_{1\beta}^2} \\
C_{d4} &= \frac{2}{3c_1 c_{1\beta}} \left[\frac{2}{3c_1} \left(\alpha_6 - \alpha_3 \alpha_5 - \frac{(1 - c_{21} + 2c_{23})\alpha_6}{c_1} \right) \right. \\
&\quad \left. + \frac{(1 - c_3)(1 - c_{21\beta})}{c_{1\beta}} \left(1 - \frac{1 - c_{21} + 2c_{23}}{c_1} \right) + \frac{c_{22\beta}}{c_{1\beta}} \left(\alpha_1 + \frac{1}{c_1} (1 - c_3)(2 - 2c_{21} + c_{23}) \right) \right. \\
&\quad \left. - 2\alpha_5 R_\beta^* \right] \\
C_{d5} &= \frac{1 - c_3}{c_1 c_{1\beta}^2} \left(\frac{2}{3} + \frac{4(1 - c_3)}{3c_1} + 2R_\beta^* \right)
\end{aligned} \tag{B.9}$$

Appendix C

Discretisation of the Coriolis force

The Coriolis terms in the 2-D and 3-D momentum equations are discretised in time using a “quasi”-implicit formulation. The time discretised equations are written as

$$\frac{u^{n+1} - u^n}{\Delta t} - f \left(\theta_c v^{n+1} + (1 - \theta_c) v^n \right) = F_1 \quad (\text{C.1})$$

$$\frac{v^{n+1} - v^n}{\Delta t} + f \left(\theta_c u^{n+1} + (1 - \theta_c) u^n \right) = F_2 \quad (\text{C.2})$$

where F_i represent all other (explicit and implicit) terms (pressure gradient, advection, diffusion, tidal force). The solution proceeds in two steps

1. Equations (C.1)–(C.2) are solved explicitly for the Coriolis force ($\theta_c = 0$). This gives

$$u^* = u^n + \Delta t (F_1 + f v^n) \quad (\text{C.3})$$

$$v^* = v^n + \Delta t (F_2 - f u^n) \quad (\text{C.4})$$

2. Substituting (C.3)–(C.4) into (C.1)–(C.2) one has

$$u^{n+1} - f \theta_c \Delta t v^{n+1} = u^* - f \theta_c \Delta t v^n \quad (\text{C.5})$$

$$v^{n+1} + f \theta_c \Delta t u^{n+1} = v^* + f \theta_c \Delta t u^n \quad (\text{C.6})$$

or

$$u^{n+1} = u^* + \frac{f \theta_c \Delta t (\Delta v - f \theta_c \Delta t \Delta u)}{1 + (f \theta_c \Delta t)^2} \quad (\text{C.7})$$

$$v^{n+1} = v^* - \frac{f\theta_c \Delta t (\Delta u + f\theta_c \Delta t \Delta v)}{1 + (f\theta_c \Delta t)^2} \quad (C.8)$$

where

$$\Delta u = u^* - u^n, \quad \Delta v = v^* - v^n \quad (C.9)$$

The following remarks apply

- The forcing terms F_i on the right hand side of (C.1)–(C.2) contain both explicit and implicit terms. A simplification has been made since the latter one are taken at level $*$ and not at the new time t^{n+1} .
- In the 3-D case the new time level t^{n+1} is replaced by the predictor step t^p .
- A four-point spatial interpolation is needed for the evaluation of the currents in the Coriolis terms. Since the open boundary conditions are applied only after solving the momentum equations at all interior nodes, open boundary values are excluded from the interpolation procedure.

Appendix D

Energy balance equation

The energy balance equation is derived by multiplying equations (5.19), (5.20), (5.18) by respectively $\rho_0 u$, $\rho_0 v$, $\rho_0 g \zeta$ and adding. For simplicity the baroclinic and atmospheric pressure gradients, the tidal force and horizontal diffusion are neglected. After a straightforward calculation one obtains

$$\frac{1}{h_3} \frac{\partial}{\partial t} (h_3 E_k) + \frac{1}{H} \frac{\partial E_p}{\partial t} + \frac{1}{h_1 h_2 h_3} \left[\frac{\partial}{\partial \xi_1} (h_2 h_3 F_1) + \frac{\partial}{\partial \xi_2} (h_1 h_3 F_2) \right] = D \quad (\text{D.1})$$

where

$$E_k = \frac{1}{2} \rho_0 (u^2 + v^2) \quad (\text{D.2})$$

is the kinetic energy,

$$E_p = \frac{1}{2} \rho_0 g \zeta^2 \quad (\text{D.3})$$

the potential energy,

$$(F_1, F_2) = \left(\frac{1}{2} \rho_0 (u^2 + v^2) + \rho_0 g \zeta \right) (u, v, \omega) \quad (\text{D.4})$$

the energy flux vector and

$$D = \rho_0 \left(\frac{u}{h_3} \frac{\partial D_1}{\partial s} + \frac{v}{h_3} \frac{\partial D_2}{\partial s} \right) \quad (\text{D.5})$$

the dissipation of energy by turbulent diffusion. The diffusion fluxes are given by

$$\begin{aligned} (D_1, D_2) &= \left(\frac{\nu_T}{h_3} \frac{\partial u}{\partial s}, \frac{\nu_T}{h_3} \frac{\partial v}{\partial s} \right) && \text{inside the water column} \\ &= (\tau_{s1}, \tau_{s2}) && \text{at the surface} \\ &= (\tau_{b1}, \tau_{b2}) && \text{at the bottom} \end{aligned} \quad (\text{D.6})$$

The vertically integrated form of (D.1) becomes

$$\frac{\partial}{\partial t}(\bar{E}_k + E_p) + \frac{1}{h_1 h_2} \left[\frac{\partial}{\partial \xi_1} (h_2 \bar{F}_1) + \frac{\partial}{\partial \xi_2} (h_1 \bar{F}_2) \right] = \bar{D} \quad (\text{D.7})$$

where

$$\bar{E}_k = \frac{1}{2} \rho_0 \int_0^{N_z} h_3 (u^2 + v^2) ds \quad (\text{D.8})$$

$$(\bar{F}_1, \bar{F}_2) = \int_0^{N_z} h_3 \left[\frac{1}{2} \rho_0 (u^2 + v^2) + \rho_0 g \zeta \right] (u, v) ds \quad (\text{D.9})$$

$$\bar{D} = \rho_0 (u_s \tau_{s1} + v_s \tau_{s2} - u_b \tau_{b1} - v_b \tau_{b2}) - \rho_0 \int_0^{N_z} \frac{\nu_T}{h_3} \left[\left(\frac{\partial u}{\partial s} \right)^2 + \left(\frac{\partial v}{\partial s} \right)^2 \right] ds \quad (\text{D.10})$$

where (u_s, v_s) and (u_b, v_b) are the velocities at respectively the surface and the bottom.

In case of a pure 2-D application, $u \simeq \bar{u}$ and $v \simeq \bar{v}$ in which case (D.7)–(D.10) reduce to

$$\bar{E}_k = \frac{1}{2} \rho_0 H (\bar{u}^2 + \bar{v}^2) \quad (\text{D.11})$$

$$(\bar{F}_1, \bar{F}_2) = H \left[\frac{1}{2} \rho_0 (\bar{u}^2 + \bar{v}^2) + \rho_0 g \zeta \right] (\bar{u}, \bar{v}) \quad (\text{D.12})$$

$$\bar{D} = \rho_0 (u_s \tau_{s1} + v_s \tau_{s2} - u_b \tau_{b1} - v_b \tau_{b2}) \quad (\text{D.13})$$

Finally, the domain integrated forms are obtained by integrating (D.7) over the 2-D horizontal domain. This gives

$$\frac{\partial}{\partial t} (\langle \bar{E}_k \rangle + \langle E_p \rangle) + \int (\bar{F}_1 n_1 + \bar{F}_2 n_2) d\ell = \langle \bar{D} \rangle \quad (\text{D.14})$$

with

$$\langle \dots \rangle = \int_0^{N_x} \int_0^{N_y} (\dots) h_1 h_2 d\xi_1 d\xi_2 \quad (\text{D.15})$$

The second integral is taken along the 2-D boundary of the domain where (n_1, n_2) is the outwards pointing unit normal along the boundary.

Appendix E

List of standard output variables

Table E.1: Key ids of the variables which can be used for defining standard output variables. The variables denoted by a * are W-node variables for which the `node` attribute can be reset to 'W'.

key id	description	unit
0-D variables		
iarr_dissip0d	domain integrated energy dissipation	W
iarr_dryfac	fraction of dry area	—
iarr_edens0d	domain integrated baroclinic energy	J
iarr_ekin0d	domain integrated kinetic energy	J
iarr_epot0d	domain integrated potential energy	J
iarr_etot0d	domain integrated total energy	J
2-D variables		
iarr_alphahtc_fld	drying factor α	
iarr_airtemp	air temperature T_a	$^{\circ}\text{C}$
iarr_atmpres	atmospheric pressure P_a	N/m^2
iarr_bdragcoefatc	bottom drag coefficient C_{db}	—
iarr_bfricatc	bottom friction velocity u_{*b}	m/s
iarr_bstresatc	bottom stress τ_b	N/m^2
iarr_cds	surface drag coefficient C_{ds}	—
iarr_ces	surface exchange coefficient C_e for the latent heat flux	—
iarr_chs	surface exchange coefficient C_h for the sensible heat flux	—
iarr_cloud_cover	cloud cover f_c	—
iarr_depmeanatc	mean water depth h	m
iarr_deptotatc	total water depth H	m

(Continued)

Table E.1: Continued

iarr_deptotatc_err	total water depth error $\delta_e H$	m
iarr_edens2d	vertically integrated baroclinic energy	J/m ²
iarr_edissip2d	vertically integrated energy dissipation	W/m ²
iarr_eflux2du	X-component of the depth-integrated energy flux	J/s/m
iarr_eflux2dv	Y-component of the depth-integrated energy flux	J/s/m
iarr_ekin2d	vertically integrated kinetic energy	J/m ²
iarr_epot2d	vertically integrated potential energy	J/m ²
iarr_etot2d	vertically integrated total energy	J/m ²
iarr_evapminprec	evaporation minus precipitation rate	kg/m ² /s
iarr_hdifcoef2datc	depth mean horizontal diffusion coefficient	m ² /s
hdvelmag	magnitude of the depth-integrated current	m ² /s
hmvelmag	magnitude of the depth-mean current	m/s
iarr_precipitation	precipitation rate R_{pr}	kg/m ² /s
iarr_qlatent	latent heat flux Q_{la}	W/m ²
iarr qlwave	long wave heat flux Q_{lw}	W/m ²
iarr_qnonsol	non-solar heat flux Q_{nsol}	W/m ²
iarr_qrad	surface solar irradiance Q_s	W/m ²
iarr_qsensible	sensible heat flux Q_{se}	W/m ²
iarr_qtot	total downward surface heat flux	W/m ²
iarr_relhum	relative humidity RH	—
iarr_ssalflux	surface salinity flux	PSU m/s
iarr_sstresatc	surface stress τ_s	N/m ²
iarr_udvel	X-component of the depth-integrated current U	m ² /s
iarr_umvel	X-component of the depth-mean current \bar{u}	m/s
iarr_uwindatc	X-component of the surface wind U_{10}	m/s
iarr_vdvel	Y-component of the depth-integrated current V	m ² /s
iarr_vmvel	Y-component of the depth-mean current \bar{v}	m/s
iarr_vortic2d	vertically integrated vorticity	m/s
iarr_vwindatc	Y-component of the surface wind V_{10}	m/s
iarr_wavedir	wave direction	degrees
iarr_wavexcurs	near-bottom wave excursion amplitude	m
iarr_wavefreq	peak wave frequency	rad/s
iarr_waveheight	significant wave height	m
iarr_wavenum	wave number	1/m
iarr_waveperiod	peak wave period	s
iarr_waveuvel	X-component of the near-bottom wave orbital velocity	m/s
iarr_wavevel	near-bottom wave orbital velocity	m/s
iarr_wavevvel	Y-component of the near-bottom wave orbital velocity	m/s

(Continued)

Table E.1: Continued

iarr_zeta	surface elevation ζ	m
iarr_zroughatc	bottom roughness z_0	m
3-D variables		
iarr_beta_sal	salinity expansion coefficient β_S	PSU ⁻¹
iarr_beta_temp	temperature expansion coefficient β_T	°C ⁻¹
iarr_buofreq2*	squared buoyancy frequency N^2	s ⁻²
iarr_dens	mass density ρ	kg/m ³
iarr_dissip*	dissipation of turbulent kinetic energy ε	W/kg
iarr_edens3d	baroclinic energy	J/m ³
iarr_eddisip3d	energy dissipation	W/m ³
iarr_eflux3du	X-component of the energy flux	W/m ³
iarr_eflux3dv	Y-component of the energy flux	W/m ³
iarr_eflux3dw	vertical component of the energy flux	W/m ³
iarr_ekin3d	kinetic energy	J/m ³
iarr_etot3d	total energy	J/m ³
iarr_hdifcoef3datac	horizontal diffusion coefficient ν_H	m ² /s
hvelmag	magnitude of the 3-D current	m/s
iarr_radiance	solar irradiance I	W/m ²
iarr_ricnum*	Richardson number Ri	—
iarr_sal	salinity S	PSU
iarr_shearfreq2*	squared shear frequency M^2	s ⁻²
iarr_temp	temperature T	°C
iarr_tke*	turbulent kinetic energy k	J/kg
iarr_uvel	X-component of the current u	m/s
iarr_vdifcoefmom*	eddy viscosity ν_T	m ² /s
iarr_vdifcoefscal*	eddy diffusivity λ_T	m ² /s
iarr_vdifcoeftke*	vertical diffusion coefficient for turbulence energy ν_k	m ² /s
iarr_vortic3d	vertical vorticity	s ⁻¹
iarr_vvel	Y-component of the current v	m/s
iarr_wphys	physical vertical current w	m/s
iarr_wvel*	transformed vertical current ω	m/s
iarr_zlmix*	mixing length l	m

Table E.2: Key ids of the variables which can be used for defining standard output variables from the sediment module. In case the variables have an extra dimension for different sediment fractions (variables marked by a *), the attribute `numvar` must be defined with a value between 1 and `nf`.

key id	description	unit
2-D variables		
<code>iarr_bdragcoefatc_sed</code>	skin bottom drag coefficient	—
<code>iarr_bed_fraction*</code>	volume sediment fraction at the sea bed	—
<code>iarr_beta_sed</code>	ratio of sediment diffusion to eddy viscosity coefficient	—
<code>iarr_bottom_sed_flux*</code>	erosion minus deposition rate	m/s
<code>iarr_bstresatc_sed</code>	skin bed shear stress	N/m^2
<code>iarr_ceq*</code>	equilibrium concentration	m^3/m^3
<code>iarr_cref*</code>	bottom reference concentration	m^3/m^3
<code>iarr_d50_bed</code>	median particle diameter at the sea bed	m
<code>iarr_height_c*</code>	reference height for bottom concentration	m
<code>iarr_qbedatc*</code>	volumetric bed load transport	m^2/s
<code>iarr_qbedatu*</code>	X-component of volumetric bed load transport	m^2/s
<code>iarr_qbedatv*</code>	Y-component of volumetric bed load transport	m^2/s
<code>iarr_qsusatc*</code>	volumetric suspended load transport	m^2/s
<code>iarr_qtotatc*</code>	volumetric total load transport	m^2/s
<code>iarr_qtotatu*</code>	X-component of volumetric total load transport	m^2/s
<code>iarr_qtotatv*</code>	Y-component of volumetric total load transport	m^2/s
<code>iarr_tau_cr*</code>	critical shear stress	N/m^2
<code>iarr_t_equiv*</code>	dimensionless adaptation time scale	—
<code>iarr_zroughatc_sed</code>	skin roughness length	m
3-D variables		
<code>iarr_beta_state_sed*</code>	sediment expansion coefficient	—
<code>iarr_ctot*</code>	total volumetric sediment concentration	m^3/m^3
<code>iarr_cvol*</code>	volumetric sediment concentration	m^3/m^3
<code>sedsrcuser*</code>	user defined sediment sources	$\text{m}^3/\text{m}^3/\text{s}$
<code>iarr_vdiffcoef_sed*</code>	sediment diffusion coefficient	m^2/s
<code>iarr_wfall*</code>	settling velocity	m/s

Bibliography

- Ackers, P., & White, W.R. 1973. Sediment transport: new approach and analysis. *Journal of the Hydraulics Division*, **99**, 2041–2060.
- Allen, A.A. 2005. *Leeway divergence*. Research and Development Report CG-D-05-05. US Coast Guard.
- Andrews, D.G., & McIntyre, M.E. 1978. An exact theory of nonlinear waves on a Lagrangian-mean flow. *Journal of Fluid Mechanics*, **89**, 609–646.
- Ardhuin, F., Rasclle, N., & Belibassakis, K.A. 2008. Explicit wave-averaged primitive equations using a generalized Lagrangian mean. *Ocean Modelling*, **20**, 35–60.
- Armanini, A. 1995. Non-uniform sediment transport: dynamics of the active layer. *Journal of Hydraulic Research*, **105**, 611–622.
- Ashida, K., & Michiue, M. 1972. An investigation of river bed degradation downstream of a dam. In: *Hydraulic research and its impact on the environment*.
- Bagnold, R.A. 1954. *The physics of blown sand and desert dunes*. Methuen, London.
- Bartnicki, J. 1989. A simple filtering procedure for removing negative values from numerical solutions of the advection equation. *Environmental Software*, **4**, 187–201.
- Beckmann, A., & Haidvogel, D.B. 1993. Numerical simulation of flow around a tall isolated seamount. Part I: Problem formulation and model accuracy. *Journal of Physical Oceanography*, **23**, 1736–1753.
- Behrenfeld, M. J., & Falkowski, P. G. 1997. A consumers guide to phytoplankton primary production models. *Limnology and Oceanography*, **42**(7), 479–1491.

- Bennis, A.-C., Arduin, F., & Dumas, F. 2011. On the coupling of wave and three-dimensional circulation models: Choice of theoretical framework, practical implementation and adiabatic tests. *Ocean Modelling*, **40**, 260–272.
- Bignami, F., Marullo, S., Santoleri, R., & Schiano, M.E. 1995. Longwave radiation budget in the Mediterranean Sea. *JGR*, **100**, 2501–2514.
- Bissinger, J. E., Montagnes, D. J. S., Sharples, J., & Atkinson, D. 2008. Predicting marine phytoplankton maximum growth rates from temperature: Improving on the Eppley curve using quantile regression. *Limnology and Oceanography*, **53**(2), 487–493.
- Blackadar, A.K. 1962. The vertical distribution of wind and turbulent exchange in a neutral atmosphere. *Journal of Geophysical Research*, **67**, 3095–3102.
- Blanc, T.V. 1985. Variation of bulk-derived surface flux, stability, and roughness results due to the use of different transfer coefficient schemes. *Journal of Physical Oceanography*, **15**, 650–669.
- Blom, A. 2003. *A vertical sorting model for rivers with non-uniform sediment and dunes*. Ph.D. thesis, University of Twente.
- Blumberg, A.F., & Mellor, G.L. 1987. A description of a three-dimensional coastal ocean circulation model. *Pages 1–16 of: Heaps, N.S. (ed), Three-dimensional Coastal Ocean Models*. Coastal and Estuarine Sciences. Washington D.C.: Americal Geophysical Union.
- Boris, J.P., & Book, D.L. 1979. Flux corrected transport: I.SHASTA, a fluid transport algorithm that works. *Journal of Computational Physics*, **11**, 38–69.
- Bos, M.G. 1998. *Discharge measurement structures*. Tech. rept. International Institute for Land Reclamation and Improvement, Wageningen, The Netherlands.
- Bowden, K.F., Fairbairn, L.A., & Hughes, P. 1959. The distribution of shearing stresses in a tidal current. *Geophysical Journal of the Royal Astronomical Society*, **2**, 288–305.
- Breugem, W.A. 2012. *Transport of suspended particles in turbulent open channel flows*. Ph.D. thesis, Tech. Univ., Delft, Netherlands.

- Brownlie, W.R. 1981. *Prediction of flow depth and sediment discharge in open channels*. Tech. rept. KH-R-43A. California Institute of Technology.
- Brunt, D. 1932. Notes on radiation in the atmosphere. *Q.J.R. Meteorol. Soc.*, **58**, 389–420.
- Buck, K. R., Chavez, F. P., & Campbell, L. 1996. Basin-wide distributions of living carbon components and the inverted trophic pyramid of the central gyre of the North Atlantic Ocean, summer 1993. *Aquatic Microbial Ecology*, **10**, 283–298.
- Bulleid, N. C. 1984. Deoxygenation and remineralization above the sediment-water interface; an in situ experimental study. *Estuarine, Coastal and Shelf Science*, **19**(1), 15–25.
- Burchard, H. 2001. Simulating the wave-enhanced layer under breaking surface waves with two-equation turbulence models. *Journal of Physical Oceanography*, **31**, 3133–3145.
- Burchard, H. 2002. *Applied Turbulence Modelling in Marine Waters*. Lecture Notes in Earth Sciences, no. 100. Berlin: Springer-Verlag.
- Burchard, H., & Baumert, H. 1995. On the performance of a mixed-layer model based on the $k - \varepsilon$ turbulence closure. *Journal of Geophysical Research*, **100**, 8523–8540.
- Burchard, H., & Bolding, K. 2002. *GETM – A General Estuarine Transport Model. Scientific Documentation*. Tech. rept. Institute for Environment and Sustainability, Joint Research Centre, Ispra, Italy.
- Burchard, H., & Petersen, O. 1999. Models of turbulence in the marine environment – a comparative study of two-equation turbulence models. *Journal of Marine Systems*, **21**, 29–53.
- Burchard, H., Deleersnijder, E., & Meister, E. 2003. A high-order Patankar-type discretisation for stiff systems of production-destruction equations. *Applied Numerical Mathematics*, **47**, 1–30.
- Businger, J.A., Wyngaard, J.C., Izumi, Y., & Bradley, E.F. 1971. Flux-profile relationships in the atmospheric surface layer. *Journal of the Atmospheric Sciences*, **28**, 181–189.
- Calbet, A., & Landry, M. R. 2004. Phytoplankton growth, microzooplankton grazing and carbon cycling in marine systems. *Limnology and Oceanography*, **49**(1), 51–57.

- Camenen, B. 2007. Simple and general formula for the settling velocity of particles. *Journal of Hydraulic Engineering*, **133**, 229–233.
- Camerlengo, A.L., & O'Brien, J.J. 1980. Open boundary conditions in rotating fluids. *Journal of Computational Physics*, **35**, 12–35.
- Canuto, V.M., Howard, A., Cheng, Y., & Dubovikov, M.S. 2001. Ocean turbulence. Part I: One-point closure model – Momentum and heat vertical diffusivities. *Journal of Physical Oceanography*, **31**, 1413–1426.
- Cartwright, D.E., & Edden, A.C. 1973. Corrected tables of tidal harmonics. *Geophysical Journal of the Royal Astronomical Society*, **33**, 253–264.
- Cartwright, D.E., & Tayler, R.J. 1971. New computations of the tide-generating potential. *Geophysical Journal of the Royal Astronomical Society*, **23**, 45–74.
- Casulli, V., & Cheng, R.T. 1992. Semi-implicit finite difference methods for three-dimensional shallow water flow. *International Journal for Numerical Methods in Fluids*, **15**, 629–648.
- Charnock, H. 1955. Wind stress on a water surface. *Quarterly Journal Royal Meteorological Society*, **81**, 639–640.
- Chauton, M. S., Tilstone, G. H. nd Legrand, C., & Johnsen, G. 1997. Changes in pigmentation, bio-optical characteristics and photophysiology, during phytoflagellate succession in mesocosms. *Journal of Plankton Research*, **26**(3), 315–324.
- Chen, C., Beardsley, R.C., & Cowles, G. 2006. *An unstructured Grid, Finite-Volume Coastal Ocean Model*. Tech. rept. School for Marine Science and Technology, Univ. of Massachusetts-Dartmouth, New Bedford, Massachusetts, US.
- Chen, C. A., Wang, S., & Pai, B. 2001. Nutrient budgets for the South China Sea basin. *Marine Chemistry*, **75**, 281–300.
- Chen, X. 2003. A free-surface correction method for simulating shallow water flows. *Journal of Computational Physics*, **189**, 557–579.
- Chollet, J.P., & Cunge, J.A. 1979. New interpretation of some head loss-flow velocity relationships for deformable movable beds. *Journal of Hydraulic Research*, **17**, 1–13.

- Christian, J. R., Verschell, M. A., Murtugudde, R., Busalacchi, A. J., & McClain, C. R. 2002. Biogeochemical modelling of the tropical Pacific Ocean. I: Seasonal and interannual variability. *Deep-Sea Research*, **4**, 509–543.
- Christofferson, J.B., & Jonsson, I.G. 1985. Bed friction and dissipation in a combined current and wave motion. *Ocean Engineering*, **12**, 387–423.
- Chu, P.C., & Fan, C. 1997. Sixth-order difference scheme for sigma coordinate ocean models. *Journal of Physical Oceanography*, **27**, 2064–2071.
- Clark, N.E., Eber, L., Laurs, R.M., Renner, J.A., & Saur, J.F.T. 1974. *Heat exchange between ocean and atmosphere in the eastern North Pacific for 1961–71*. NOAA Tech. Rep. NMFS SSRF-682. U.S. Dep. of Commer., Washington D.C.
- Cloern, J. E., Grenz, C., & Vidergar-Lucal, L. 1995. An empirical model for the phytoplankton chlorophyll:carbon ratio—the conversion factor between productivity and growth rate. *Limnology and Oceanography*, **40**, 1313–1321.
- Colella, P., & Woodward, P.R. 1984. The piecewise parabolic method (PPM) for gas-dynamical simulations. *Journal of Computational Physics*, **54**, 174–201.
- Coleman, N.L. 1970. Flume studies of the sediment transfer coefficient. *Water Resources Research*, **6**(3), 801–809.
- Cox, M.D. 1987. Isopycnal diffusion in a z -coordinate ocean model. *Ocean Modelling*, **74**, 1–5.
- Craig, P.D., & Banner, M.L. 1994. Modeling wave-enhanced turbulence in the ocean surface layer. *Journal of Physical Oceanography*, **24**, 2546–2559.
- Daly, B.J., & Harlow, F.H. 1970. Transport equations in turbulence. *Physics of Fluids*, **13**, 2634–2649.
- Davies, A.M. 1990. On extracting tidal current profiles from vertically integrated two-dimensional hydrodynamic models. *Journal of Geophysical Research*, **95**, 18317–18342.
- Davies, A.M. 1993. A bottom boundary layer-resolving three-dimensional tidal model: A sensitivity study of eddy viscosity formulation. *Journal of Physical Oceanography*, **23**, 1437–1453.

- Davies, A.M., & Jones, J.E. 1991. On the numerical solution of the turbulence energy equations for wave and tidal flows. *International Journal for Numerical Methods in Fluids*, **12**, 17–41.
- Davies, A.M., & Jones, J.E. 1992. A three-dimensional wind driven circulation model of the Celtic and Irish Seas. *Continental Shelf Research*, **12**, 159–188.
- Davies, A.M., Kwong, S.C.M., & Flather, R.A. 1997. Formulation of a variable-function three-dimensional model, with applications to the M_2 and M_4 tide on the North-West European continental shelf. *Continental Shelf Research*, **17**, 165–204.
- Dearman, J. R., Taylor, A. H., & Davidson, K. 2003. Influence of autotroph model complexity on simulations of microbial communities in marine mesocosms. *Marine Ecology Progress Series*, **250**, 13–28.
- Deleersnijder, E. 1993. Numerical mass conservation in a free-surface sigma coordinate marine model with mode splitting. *Journal of Marine Systems*, **4**, 365–370.
- Delft3D. 2009. *Delft3D-FLOW. Simulation of multi-dimensional hydrodynamic flows and transport phenomena, including sediments. User Manual. Version 3.14*. Delft, The Netherlands.
- Dietrich, W.E. 1982. Settling velocity of natural particles. *Water Resources Research*, **18**, 1615–1626.
- Doodson, A.T. 1921. The harmonic development of the tide-generating potential. *Proceedings of the Royal Society of London A*, **100**, 305–329.
- Eaton, B., Gregory, J., Drach, B., Taylor, K., Hankin, S., Caron, J., Signell, R., Bentley, P., Rappa, G., Höck, H., Pamment, A., & Juckes, M. 2011. *NetCDF Climate and Forecast (CF) metadata Conventions. Version 1.6*.
- Egiazaroff, I.V. 1965. Calculation of nonuniform sediment concentrations. *Journal of the Hydraulics Division*, **91**, 225–248.
- Engelund, F., & Fredsøe, J. 1976. A sediment transport model for straight alluvial channels. *Nordic Hydrology*, **7**, 293–306.
- Engelund, F., & Hansen, E. 1967. *A monograph on sediment transport in alluvial streams*. Tech. rept. Denmark Tech. Univ., Hydraul. Lab.

- Eppley, R. W. 1972. Temperature and phytoplankton growth in the sea. *Fisheries Bulletin*, **70**, 1063–1085.
- Faure, V., Pinazoa, C., TorrÃ©ton, J., & Douillet, P. 200. Relevance of various formulations of phytoplankton chlorophyll a:carbon ratio in a 3D marine ecosystem model. *Ecology*, **329**(10), 813–822.
- Ferziger, J. 2005. Turbulence: its origins and structure. *Pages 4–13 of: Baumert, H.Z., Simpson, J., & SÃ¼ndermann, J. (eds), Marine Turbulence. Theories, Observations and Models.* Cambridge University Press.
- Finkel, Z. V., Beardall, J., Flynn, K. J., Quigg, A., Rees, T. A. V., & Ra, J. A. 2010. Phytoplankton in a changing world: cell size and elemental stoichiometry. *Journal of Plankton Research*, **32**(1), 119–137.
- Flather, R.A. 1976. A tidal model of the northwest European continental shelf. *MÃ©moires de la SociÃ©tÃ© Royale des Sciences de LiÃ¨ge*, **10**, 141–164.
- Flynn, K. J. 2005. Castles build on sand: dysfunctionality in plankton models and the inadequacy of dialogue between biologists and modellers. *Journal of Plankton Research*, **27**(12), 1205–1210.
- Foreman, M.G.G., Henry, R.F., Walters, R.A., & Ballantyne, V.A. 1993. A finite element model for tides and resonance along the north coast of British Columbia. *Journal of Geophysical Research*, **98**, 2509–2532.
- Fortunato, AndrÃ© B., & Oliveira, A. 2004. A modeling system for tidally driven long-term morphodynamics. *Journal of Hydraulic Research*, **42**, 426–434.
- Franks, P.S.J. 2009. Planctonic ecosystem models: perplexing parameterizations and a failure to fail. *Journal of Plankton Research*, **31**(11), 1299–1306.
- Friedrichs, C., & Madsen, O. 1992. Nonlinear diffusion of the tidal signal in frictionally dominated embayments. *Journal of Geophysical Research*, **97**, 5637–5650.
- Galappatti, R., & Vreugdenhil, C.B. 1985. A depth-integrated model for suspended sediment transport. *Journal of Hydraulic Research*, **23**, 359–377.
- Galperin, B., Kantha, L.H., Hassid, S., & Rosati, A. 1988. A quasi-equilibrium turbulent energy model for geophysical flows. *Journal of the Atmospheric Sciences*, **45**, 55–62.

- Galperin, B., A., Rosati, Kantha, L.H., & Mellor, G.L. 1989. Modeling rotating stratified turbulent flows with application to oceanic mixed layers. *Journal of Physical Oceanography*, **19**, 901–916.
- Gan, J., Lu, Z., Dai, M., Cheung, A. Y. Y., Liu, H., & Harrison, P. 2010. Biological response to intensified upwelling and to a river plume in the northeastern South China Sea: A modeling study. *Journal of Geophysical Research*, **115**, 1–19.
- Garcia, M., & Parker, G. 1991. Entrainment of bed sediment into suspension. *Journal of Hydraulic Engineering*, **117**, 414–435.
- Geernaert, G.L.. 1990. Bulk parameterizations for the wind stress and heat fluxes. *Pages 91–172 of: Geernaert, G.L., & Plant, W.J. (eds), Surface Waves and Fluxes, Vol. 1 – Current theory.*
- Geernaert, G.L., Katsaros, K.B., & Richter, K. 1986. Variation of the drag coefficient and its dependence on sea state. *Journal of Geophysical Research*, **91**, 7667–7679.
- Geider, R. J., MacIntyre, H. L., & Kana, T. M. 1997. Dynamic model of phytoplankton growth and acclimation: responses of the balanced growth rate and the chlorophyll a:carbon ratio to light, nutrient limitation and temperature. *Marine Ecology Progress Series*, **148**, 187–200.
- Gent, P.R., & McWilliams, J.C. 1990. Isopycnal mixing in ocean circulation models. *Journal of Physical Oceanography*, **20**, 150–155.
- Gerdes, R., Köberle, C., & Willebrand, J. 1991. The influence of numerical advection schemes on the results of ocean general circulation models. *Climate Dynamics*, **5**, 211–226.
- Gill, A.E. 1982. *Atmospheric-Ocean Dynamics*. International Geophysics Series, vol. 30. Orlando: Academic Press.
- Glorioso, P.D., & Davies, A.M. 1995. The influence of eddy viscosity formulation, bottom topography, and wind wave effects upon the circulation of a shallow bay. *Journal of Physical Oceanography*, **25**, 1243–1264.
- Godin, G. 1972. *The Analysis of Tides*. Liverpool University Press.
- Goldstein, E.B., Coco, G., & Murray, A.B. 2013. Prediction of wave ripple characteristics using genetic programming. *Continental Shelf Research*, **71**, 1–15.

- Grangere, K., Lefebvre, S. ans Menesguen, A., & Jouenne, F. 2009. On the interest of using field primary production data to calibrate phytoplankton rate processes in ecosystem models. *Estuarine, Coastal and Shelf Science*, **81**, 169–178.
- Grant, W.D., & Madsen, O.S. 1979. Combined wave and current interaction with a rough bottom. *Journal of Geophysical Research*, **84**, 1797–1808.
- Grant, W.D., & Madsen, O.S. 1982. Movable bed roughness in unsteady oscillatory flow. *Journal of Geophysical Research*, **87**, 469–481.
- Grant, W.D., & Madsen, O.S. 1986. The continental-shelf bottom boundary layer. *Ann. Rev. Fluid Mech.*, **18**, 265–305.
- Griffies, S.M. 1998. The Gent-McWilliams skew flux. *Journal of Physical Oceanography*, **28**, 831–841.
- Griffies, S.M. 2004. *Fundamentals of ocean climate modelling*. Princeton University Press.
- Griffies, S.M., Gnanadesikan, A., Pacanowski, R.C., Larichev, V.D., Dukowicz, J.K., & R.D., Smith. 1998. Isoneutral diffusion in a z-coordinate ocean model. *Journal of Physical Oceanography*, **28**, 805–830.
- Guan, W.B., & Wolanski, S.C. 2005. 3-D fluid-mud dynamics in the Jiaojiang estuary, China. *Estuarine, Coastal and Shelf Science*, **65**, 747–762.
- Haney, R.L. 1991. On the pressure gradient force over steep topography in sigma coordinate ocean models. *Journal of Physical Oceanography*, **21**, 610–619.
- Hannides, C. C. S., Landry, M. R., Benitez-Nelson, C. R., Styles, R. M., Montoya, J. P., & Karl, D. M. 2009. Export stoichiometry and migrant-mediated flux of phosphorus in the North Pacific Subtropical Gyre. *DSR*, **56**, 73–88.
- Hansen, P. J., & Bjornsen, P. K. 1997. Zooplankton grazing and growth: Scaling with the 2–2000 μm body size range. *Limnology and Oceanography*, **4**(4), 687–704.
- Harris, Courtney K., & Wiberg, P. 1997. Approaches to quantifying long-term continental shelf sediment transport with an example from the Northern California STRESS mid-shelf site. *Continental Shelf Research*, 1389–1418.

- Hasselmann, K. 1971. On the mass and momentum transfer between short gravity waves and larger-scale motions. *Journal of Fluid Mechanics*, **50**, 189–205.
- Heathershaw, A.D. 1981. Comparisons of measured and predicted sediment transport rates in tidal currents. *Marine Geology*, **42**, 75–104.
- Hedstrom, G.W. 1979. Nonreflecting boundary conditions for nonlinear hyperbolic systems. *Journal of Computational Physics*, **30**, 222–237.
- Hirano, M. 1971. River bed degradation with armouring. *Trans. Jpn. Soc. Civ. Eng*, **3**, 194–195.
- Hirsch, C. 1990. *Numerical computation of internal and external flows. Volume 2: Computational methods for inviscid and viscous flows*. New York: Wiley.
- Hofmann, M., & Maqueda, M.A.M. 2006. Performance of a second-order moments advection scheme in an ocean general circulation model. *Journal of Geophysical Research*, **111**, doi:10.1029/2005JC003297.
- Hossain, M.S., & Rodi, W. 1982. A turbulence model for buoyant flows and its application to vertical buoyant jets. *Pages 121–178 of: Rodi, W. (ed), Turbulent buoyant jets and plumes*. HMT-Series, vol. 6. Oxford: Pergamon Press.
- Hunt, J.N. 1979. Direct solution of wave dispersion equation. *Journal of the Waterway Port Coastal and Ocean Division*, **105**, 457–459.
- Hunter, J.R. 1997. The application of Lagrangian particle-tracking techniques to modelling of dispersion in the sea. *Pages 257–269 of: Noye, J. (ed), Numerical modelling: application to marine systems*.
- Huthnance, J.M. (ed). 1997. *Processes in Regions of Freshwater Influence – PROFILE, MAS2-CT93-0054. Final Report*. Proudman Oceanographic Laboratory. POL Internal Document No. 102.
- Il'Yash, L. V., Matorin, D. N., Kol'tsova, T.I., & Sham, H. H. 2004. Spatial distribution and daily dynamics of phytoplankton in Nhatrang Bay of the South China Sea. *Oceanology*, **44**(2), 219–229.
- Iselin, C.O. 1939. The influence of vertical and lateral turbulence on the characteristics of the waters at mid-depth. *Trans. Amer. Geophys. Union*, **20**, 414–417.

- ITTC. 1978. *Proceedings of the 15th International Towing Tank Conference. The Hague. Report of the Performance Committee.*
- Jacket, D.R., & McDougall, T.J. 1995. Minimal adjustment of hydrographic profiles to achieve static stability. *Journal of Atmospheric and Oceanic Technology*, **12**, 381–389.
- James, I.D. 1996. Advection schemes for shelf sea models. *Journal of Marine Systems*, **8**, 237–254.
- Jameson, A., Schmidt, W., & Turkel, E. 1981. Numerical solutions of the Euler equations by finite volume methods using Runge-Kutta time-stepping schemes. In: *AIAA paper 81-1259*. AIAA (th Computational Fluid Dynamics Conference, Honolulu, Hawaii.
- Jassby, A.D., & Platt, T. 976. Mathematical formulation of the relationship between photosynthesis and light for phytoplankton. *Limnology and Oceanography*, **21**(4), 540–547.
- Jensen, T.G. 1998. Open boundary conditions in stratified ocean models. *Journal of Marine Systems*, **16**, 297–322.
- Jerlov, N.G. 1968. *Optical Oceanography*. Elsevier.
- Kantha, L.H., & Clayson, C.A. 1994. An improved mixed layer model for geophysical applications. *Journal of Geophysical Research*, **99**, 25235–25266.
- Kantha, L.H., & Clayson, C.A. 2000a. *Small Scale Processes in Geophysical Fluid Flows*. San Diego, California: Academic Press.
- Kantha, L.H., & Clayson, C.A. 2000b. *Numerical Models of Oceans and Oceanic Processes*. San Diego, California: Academic Press.
- Kantha, L.H., Rosati, A., & Galperin, B. 1989. Effect of rotation on vertical mixing and associated turbulence in stratified fluids. *Journal of Geophysical Research*, **94**, 4843–4854.
- Key, J.R., Silcox, R.A., & Stone, R.S. 1996. Evaluation of surface radiative flux parameterizations for use in sea ice models. *Journal of Geophysical Research*, **101**, 3839–3849.
- Kleppel, G. S., Holliday, D. V., & Pieper, R. E. 1991. Trophic interactions between copepods and microplankton: a question about the role of diatoms. *Limnology and Oceanography*, **36**(1), 172–178.

- Kleppel, G. S., Holliday, D. V., & Pieper, R. E. 1993. On the diets of calanoid copepods. *Marine Ecology Progress Series*, **99**, 183–195.
- Kliem, N., & Pietrzak, J.D. 1999. On the pressure gradient error in sigma coordinate ocean models: A comparison with a laboratory experiment. *Journal of Geophysical Research*, **104**, 29781–29799.
- Koch, F.G., & Flokstra, C. 1981. Bed level computations for curved alluvial channels. In: *19th IAHR Congress*, vol. 2.
- Kondo, J. 1975. Air-sea bulk transfer coefficients in diabatic conditions. *Boundary-Layer Meteorology*, **9**, 91–112.
- Lacroix, G., Ruddick, K., Park, Y., Gypens, N., & Lancelot, C. 2007. Validation of the 3D biogeochemical model MIRO&CO with field nutrient and phytoplankton data and MERIS-derived surface chlorophyll a images. *Journal of Marine Systems*, **64**, 66–88.
- Lane, E.M., Restrepo, J.M., & McWilliams, J.C. 2007. Wave-current interaction: A comparison of radiation-stress and vortex-force representations. *Journal of Physical Oceanography*, **37**, 1122–1141.
- Large, W.G., & Pond, S. 1981. Open ocean momentum flux measurements in moderate to strong winds. *Journal of Physical Oceanography*, **11**, 324–336.
- Large, W.G., & Yeager, S. 2004. *Diurnal to decadal global forcing for ocean and sea-ice models: the data sets and flux climatologies*. NCAR Technical Note, NCAR/TN-460+STR. CGD Division of the National Center for Atmospheric Research.
- Large, W.G., McWilliams, J.C., & Doney, S.C. 1994. Oceanic vertical mixing: A review and a model with a nonlocal boundary layer parameterization. *Reviews of Geophysics*, **32**, 363–403.
- Lee, J.L., Toorman, E., Molz, F.J., & Wang, J. 2011. A two-class population balance equation yielding bimodal flocculation of marine or estuarine sediments. *Water Research*, **45**, 2131–2145.
- Lee, J.L., Toorman, E., & Fettweis, M. 2014. Multimodal particle size distributions of fine-grained sediments: mathematical modelling and field investigation. *Ocean Dynamics*, doi:10.1007/s10236-014-0692-y.
- Leonard, B.P. 1979. A stable and accurate convective modelling procedure based on quadratic upstream interpolation. *Computer Methods in Applied Mechanics and Engineering*, **19**, 59–98.

- Li, M.Z., Wright, L.D., & Amos, C.L. 1996. Predicting ripple roughness and sand resuspension under combined flows in a shoreface environment. *Marine Geology*, **130**, 139–161.
- Liu, K., Chen, Y., Tseng, C., Lin, I., Liu, H., & Snidvongs, A. 2007. The significance of phytoplankton photo-adaptation and benthic-pelagic coupling to primary production in the South China Sea: Observations and numerical investigations. *Deep-Sea Research*, **54**, 1546–1574.
- Liu, K. K., Tseng, C. M., Wu, C. R., & Lin, I. I. 2010a. *Carbon and Nutrient fluxes in continental margins. Chapter 8.6 The South China Sea*. Springer-Verlag Berlin.
- Liu, S., Li, T., Huang, H., Guo, Z. L., Huang, L. M., & Wang, W. X. 2010b. Feeding efficiency of a marine copepod *Acartia erythraea* on eight different algal diets. *Acta Ecologica Sinica*, **30**(1), 22–26.
- Liu, W.C., Liu, S.Y., Hsu, M.H., & Kuo, A.Y. 2005. Water quality modelling to determine minimum instream flow for fish survival in tidal rivers. *Journal of Environmental Management*, **76**, 293–308.
- Lu, Z., Gan, J., Dai, M., & Cheung, A. Y. Y. 2010. The influence of coastal upwelling and a river plume on the subsurface chlorophyll maximum over the shelf of the northeastern South China Sea. *Journal of Marine Systems*, **82**, 35–46.
- Luyten, P.J. 1996. An analytical and numerical study of surface and bottom boundary layers with variable forcing and application to the North Sea. *Journal of Marine Systems*, **8**, 171–189.
- Luyten, P.J. 1997. Modelling physical processes of haline stratification in ROFIs with application to the Rhine outflow region. *Journal of Marine Systems*, **12**, 277–298.
- Luyten, P.J. (ed). 1999. *COHERENS – Dissemination and exploitation of a coupled hydrodynamical-ecological model for regional and shelf seas, MAS3-CT97-0088. Final Report*. Management Unit of the Mathematical Models. MUMM internal report.
- Luyten, P.J., Jones, J.E., Proctor, R., Tabor, A., Tett, P., & Wild-Allen, K. 1999. *COHERENS – A coupled hydrodynamical-ecological model for regional and shelf seas: User Documentation*. Tech. rept. Management Unit of the Mathematical Models of the North Sea (MUMM), Belgium.

- Luyten, P.J., Carniel, S., & Umgiesser, G. 2002. Validation of turbulence closure parameterisations for stably stratified flows using the PROVESS turbulence measurements in the North Sea. *Journal of Sea Research*, **47**, 239–267.
- Luyten, P.J., Jones, J.E., & Proctor, R. 2003. A numerical study of the long- and short-term temperature variability and thermal circulation in the North Sea. *Journal of Physical Oceanography*, **33**, 37–56.
- Luyten, P.J., Andreu-Burillo, I., Norro, A., Ponsar, S., & Proctor, R. 2006. A new version of the European public domain code COHERENS. *Pages 474–481 of: Dahlin, H., Flemming, N.C., Marchand, P., & Petersson, S.E. (eds), European Operational Oceanography: Present and Future.*
- Madsen, O.S., & Grant, W.D. 1976. Quantitative description of sediment transport by waves. *Pages 1093–1112 of: Proc. 15th Coastal Eng. Conf., ASCE.*
- Maier-Reimer, E., & Sündermann, J. 1982. *On Tracer Methods in Computational Hydrodynamics*. Engineering Application of Computational Hydraulics 1. Pitman Advanced Publ. Program, Boston.
- Malarkey, J., & Davies, A.G. 2003. A non-iterative procedure for the Wiberg and Harris (1994) oscillatory sand ripple predictor. *JCR*, **19**, 738–739.
- Malarkey, J., & Davies, A.G. 2012. A simple procedure for calculating the mean and maximum bed stress under wave and current conditions for rough turbulent flow based on Soulsby and Clarke's (2005) method. *Computers and Geosciences*, **43**, 101–107.
- Martinsen, E.A., & Engedahl, H. 1987. Implementation and testing of a lateral boundary scheme as an open boundary condition in a barotropic ocean model. *Coastal Engineering*, **11**, 603–627.
- Mathieu, P.-P., & Deleersnijder, E. 1998. What is wrong with isopycnal diffusion in world ocean models ? *Applied Mathematical Modelling*, **22**, 367–378.
- McCalpin, J.D. 1994. A comparison of second-order and fourth-order pressure gradient algorithms in a σ -coordinate ocean model. *International Journal for Numerical Methods in Fluids*, **18**, 361–383.
- McDougall, T.J., Jackett, D.R., Wright, D.G., & Feistel, R. 2003. Accurate and computationally efficient algorithms for potential temperature and

- density of seawater. *Journal of Atmospheric and Oceanic Technology*, **20**, 730–741.
- McWilliams, J.C., Restrepo, J.M., & Lane, E.M. 2004. An asymptotic theory for the interaction of waves and currents in coastal waters. *Journal of Fluid Mechanics*, **511**, 135–178.
- Mellor, G. 2003. The three-dimensional current and surface wave equations. *Journal of Physical Oceanography*, **33**, 1978–1989.
- Mellor, G.L., & Yamada, T. 1974. A hierarchy of turbulence closure models for planetary boundary layers. *Journal of the Atmospheric Sciences*, **31**, 1791–1806.
- Mellor, G.L., & Yamada, T. 1982. Development of a turbulence closure model for geophysical fluid problems. *Reviews of Geophysics and Space Physics*, **20**, 851–875.
- Menesguin, A., Cugier, P., Loyer, S., Vanhoutte-Brunier, A., Hoch, T., Guillaud, J., & Gohin, F. 2007. Two- or three-layered box-models versus fine 3D models for coastal ecological modelling. A comparative study in the English channel (Western Europe). *Journal of Marine Systems*, **64**(1–4), 47–65.
- Mesinger, F., & Arakawa, A. 1976. *Numerical methods used in atmospheric models*. Tech. rept. Global Atmospheric Research Programme (GARP) publication series.
- Meyer-Peter, E., & Müller, R. 1948. Formulas for bed-load transport. *Pages 39–64 of: Proceedings of the 2nd Meeting of the International Association for Hydraulic Structures Research*.
- Millero, F.J., Chen, C.-T., Bradshaw, A., & Schleicher, K. 1980. A new high pressure equation of state for seawater. *Deep-Sea Research*, **27A**.
- Moloney, C. L., & Filed, J. G. 1991. The size based dynamics of plankton food webs; I. Simulation model of carbon and nitrogen flows. *Journal of Plankton Research*, **13**, 1003–1038.
- Monin, A., & Obukhov, A. 1954. Basic turbulent mixing laws in the atmospheric near-surface layer. *Trans. Geophys. Inst. Akad. Nauk USSR*, **151**, 163–187.
- Montgomery, R.B. 1940. The present evidence on the importance of lateral mixing processes in the ocean. *Bull. Amer. Meteor. Soc.*, **21**, 87–94.

- Montheith, J.L., & Unsworth, M.H. 2008. *Principles of Environmental Physics*. third edn. New York: Academic Press.
- Moore, C. M., Suggett, D., Holligan, P. M., Sharples, J., Abraham, E. R., Lucas, M. I., Rippeth, T. P., Fisher, N. R., Simpson, J. H., & Hydes, D. J. 2007. Physical controls on phytoplankton physiology and production at a shelf sea front: a fast repetition rate fluorometer based field study. *Marine Ecology Progress Series*, **259**, 29–45.
- MPI. 1995. *A message-passing interface standard (version 1.1)*. Message Passing Interface Forum, Univ. of Tennessee, Knoxville, Tennessee.
- Munk, W.H., & Anderson, E.R. 1948. Notes on a theory of the thermocline. *Journal of Marine research*, **VII**(3), 276–295.
- Murray, F.W. 1967. On the computation of saturation vapour pressure. *Journal of Applied Meteorology*, **6**, 203–204.
- Naimie, C.E., Loder, J.W., & Lynch, D.R. 1994. Seasonal variation of the three-dimensional residual circulation on Georges Bank. *Journal of Geophysical Research*, **99**, 15967–15989.
- Nielsen, P. 1992. *Coastal bottom boundary layers and sediment transport*. Advanced Series on Ocean Eng. World Scientific, Singapore.
- Ning, X. 2004. Physical–biological oceanographic coupling influencing phytoplankton and primary production in the South China Sea. *Journal of Geophysical Research*, **109**(C10005), doi: 10.1029/2004JC002356.
- Ning, X., Chai, F., Xue, H., Cai, Y., Liu, C., & Shi, J. 2004. Physical–biological oceanographic coupling influencing phytoplankton and primary production in the South China Sea. *Journal of Geophysical Research*, **109**(C10005), doi: 10.1029/2004/JC002365), 1–20.
- Odintsov, V. S., Propp, M. V., & Dultseva, O. A. 1993. Denitrification, Nitrification and Nitrogen Fixation in Sublittoral Sediments of the South China Sea. *Internationale Revue der gesamten Hydrobiologie und Hydrographie*, **77**(3), 379–389.
- Orlanski, I. 1976. A simple boundary condition for unbounded hyperbolic flows. *Journal of Computational Physics*, **21**, 251–269.
- Pacanowski, R., & Griffies, S.M. 2000. *MOMM 3.0 Manual*. Tech. rept. Geophysical Fluid Dynamics Laboratory.

- Pacanowski, R.C., & Philander, S.G.H. 1981. Parameterization of vertical mixing in numerical models of tropical oceans. *Journal of Physical Oceanography*, **11**, 1443–1451.
- Palma, E.D., & Matano, R.P. 1998. On the implementation of passive open boundary conditions for a general circulation model: The barotropic mode. *Journal of Geophysical Research*, **103**, 1319–1341.
- Parker, G. 1991. Selective sorting and abrasion of river gravel. I: Theory. *Journal of Hydraulic Engineering*, **117**, 131–147.
- Partheniades, E. 1965. Erosion and deposition of cohesive soils. *Journal of the Hydraulics Division*, **91**, 105–139.
- Patankar, S.V. 1980. *Numerical Heat Transfer and Fluid Flow*. Lecture Notes in Earth Sciences. New York: McGraw-Hill.
- Paulson, C.A., & Simpson, J.J. 1977. Irradiance measurements in the upper ocean. *Journal of Physical Oceanography*, **7**, 952–956.
- Peters, H., Gregg, M.C., & Toole, J.M. 1988. On the parameterization of equatorial turbulence. *Journal of Geophysical Research*, **93**, 1199–1218.
- Phillips, N.A. 1957. A coordinate system having some special advantages for numerical forecasting. *Journal of Meteorology*, **14**, 184–185.
- Phillips, O.M. 1977. *The Dynamics of the Upper Ocean*. Cambridge University Press.
- Pincus, R., & Rew, R. 2008. *The NetCDF Fortran 90 Interface Guide*. UNIDATA Program Center of the University Corporation for Atmospheric Research.
- Pope, S.B. 2001. *Turbulent Flows*. Cambridge, U.K.: Cambridge University Press.
- Prandle, D. 1982. The vertical structure of tidal currents and other oscillatory flows. *Continental Shelf Research*, **1**, 191–207.
- Prather, M.J. 1986. Numerical advection by conservation of second-order moments. *Journal of Geophysical Research*, **91**, 6671–6681.
- Press, W.H., Teukolsky, S.A., Vetterling, W.T., & Flannery, B.P. 1992. *Numerical Recipes. The art of scientific computing*. 2 edn. Cambridge: Cambridge University Press.

- Proctor, R. (ed). 1997. *NOMADS – North Sea Model Advection-Dispersion study, MAS2-CT94-0105. Technical Report IV: Model Intercomparison*. Proudman Oceanographic Laboratory. POL Internal Document No. 107.
- Proctor, R., & Davies, A.M. 1996. A three dimensional hydrodynamic model of tides off the north-west coast of Scotland. *Journal of Marine Systems*, **7**, 43–66.
- Redi, M.H. 1982. Oceanic isopycnal mixing by coordinate rotation. *Journal of Physical Oceanography*, **12**, 1154–1158.
- Ribberink, J.S. 1987. *Mathematical modelling of one-dimensional morphological changes in rivers with non-uniform sediment*. Ph.D. thesis, Technical University Delft.
- Richardson, J.F., & Zaki, W.N. 1954. Sedimentation and fluidisation: Part I. *Chemical Engineering Research and Design*, **32a**, 35–53.
- Rodi, W. 1984. *Turbulence models and their application in hydraulics*. 2 edn. Delft, Netherlands: International Association for Hydraulic Research.
- Roe, P.L. 1985. Some contributions to the modelling of discontinuous flows. *Pages 163–193 of: Proceedings of 1983 AMS-SIAM summer seminar on large scale computing in fluid mechanics*. Lectures in Applied Mathematics, vol. 22.
- Roe, P.L. 1986. Characteristic-based schemes for the Euler equations. *Annual Review of Fluid Mechanics*, **18**, 337–365.
- Røed, L.P., & Cooper, C.K. 1987. A study of various open boundary conditions for wind-forced barotropic numerical ocean models. *Pages 305–335 of: Nihoul, J.C.J., & Jamart, B.M. (eds), Three-dimensional models of marine and estuarine dynamics*. Amsterdam: Elsevier.
- Røed, L.P., & Smedstad, O.M. 1984. Open boundary conditions for forced waves in a rotating fluid. *SIAM J. Sci. Stat. Comput.*, **5**, 414–426.
- Roelvink, J.A. 2006. Coastal morphodynamic evolution techniques. *Coastal engineering*, **53**, 277–287.
- Roelvink, J.A., & Walstra, D.J.R. 2005. *Keeping it simple by using complex models*. Tech. rept. Univ. of Mississippi.
- Rosati, A., & Miyakoda, K. 1988. A general circulation model for upper ocean simulation. *Journal of Physical Oceanography*, **18**, 1601–1626.

- Rotta, J.C. 1951. Statistische theorie nichthomogener turbulenz. *Zeitshrift für Physik*, **129**, 547–572.
- Ruddick, K.G. 1995. *Modelling of coastal processes influenced by the fresh-water discharge of the Rhine*. Ph.D. thesis, Univ. de Liège, Belgium.
- Ruddick, K.G., Deleersnijder, E., Luyten, P.J., & Ozer, J. 1995. Haline stratification in the Rhine-Meuse freshwater plume: A three-dimensional model sensitivity analysis. *Continental Shelf Research*, **15**, 1597–1630.
- Russ, R., Davis, G., Emmerson, S., & Davis, H. 2004. *The NetCDF User's Guide. Data model, programming interfaces and format for self-describing, portable data*. UNIDATA Program Center of the University Corporation for Atmospheric Research. <http://www.unidata.ucar.edu/sofware/netcdf>.
- Sakshaug, E., Bricaud, A., Dandonneau, Y., Falkowski, P. G., Kiefer, D. A., Legendre, L., Morel, A., Parslow, J., & Takahashi, M. 1997. Parameters of photosynthesis: definitions, theory and interpretation of results. *Journal of Plankton Research*, **19**(11), 1637–1670.
- Sarthou, G., Timmermans, K. R., Blain, S., & TrÃ©guer, P. 200. Growth physiology and fate of diatoms in the ocean: a review. *Journal of Sea Research*, **53**, 25–42.
- Schmittner, A., Oschlies, A., Giraud, X., Eby, L., & Simmon, H. L. 2005. A global model of the marine ecosystem for long-term simulations: Sensitivity to ocean mixing, buoyancy forcing, particle sinking, and dissolved organic matter cycling. *Global biogeochemical cycles*, **19**, GB3004, doi:10.1029/2004GB002283.
- Schureman, P. 1941. *Manual of Harmonic Analysis and Prediction of Tides*. Dept. of Commerce Special Publication 98, U.S. Government Printing Office, Washington D.C.
- Semtner, A.J., & Chervin, R.M. 1988. A simulation of the global ocean circulation with residual eddies. *Journal of Geophysical Research*, **93**, 15502–15522.
- Shchepetkin, A.F., & McWilliams, J.C. 2003. A method for computing the horizontal pressure-gradient force in an oceanic model with a nonaligned vertical coordinate. *Journal of Geophysical Research*, **108**, doi:10.1029/2001JC001047.

- Shields, A. 1936. *Anwendung der Ächlichkeitsmechanik und der Turbulenz Forschung auf die Geschiebebewegung*. Mitteilungen der Preussischen Versuchsanstalt für Wasserbau and Schiffbau.
- Shine, K.P. 1984. Parameterization of the shortwave flux over high albedo surfaces as a function of cloud thickness and surface albedo. *Q.J. Roy. Meteor. Soc.*, **110**, 747–764.
- SIMONA. 2009. *WAQUA/TRIWAQ – two- and three-dimensional shallow water flow model. Technical documentation. Version 3.10*. Ministry of Transport, Public Works and Water Management. The Netherlands. http://www.helpdeskwater.nl/onderwerpen/applicaties-modellen/-water_en_ruimte/simona/.
- Siswanto, E., Ishizaka, J., & Yokouchi, K. 2006. Optimal Primary Production Model and Parameterization in the Eastern China Sea. *Journal of Oceanography*, **62**, 361–372.
- Slørdal, L.H. 1997. The pressure gradient force in sigma-coordinate ocean models. *International Journal for Numerical Methods in Fluids*, **24**, 987–1017.
- Smagorinsky, J. 1963. General circulation experiments with the primitive equations - I. The basic experiment. *Monthly Weather Review*, **91**, 99–164.
- Smayda, T. J. 1997. Harmful algal blooms: their ecophysiology and general relevance to phytoplankton blooms in the sea. *Limnology and Oceanography*, **42**(5), 1137–1153.
- Smith, J.D., & McLean, S.R. 1977. Spatially averaged flow over a wavy surface. *Journal of Geophysical Research*, **82**, 1735–1746.
- Smith, S.D., & Banke, E.G. 1975. Variation of the sea surface drag coefficient with windspeed. *Quarterly Journal Meteorological Society*, **101**, 665–673.
- Song, Y., & Haidvogel, D. 1994. A semi-implicit ocean circulation model using a generalized topography-following coordinate system. *Journal of Computational Physics*, **115**, 228–244.
- Song, Y.T. 1998. A general pressure gradient formulation for ocean models, 1. Scheme design and diagnostic analysis. *Monthly Weather Review*, **126**, 3213–3230.

- Soulsby, R.L. 1983. The bottom boundary layers of shelf seas. *Pages 189–266 of: Johns, B. (ed), Physical Oceanography of Coastal and Shelf Seas*, vol. 35. Elsevier Oceanography Series.
- Soulsby, R.L. 1997. *Dynamics of marine sands: a manual for practical applications*. Thomas Telford, London.
- Soulsby, R.L., & Clarke, S. 2005. *Bed shear-stress under combined waves and currents on smooth and rough beds*. Report TR 137. HR Wallingford, Wallingford, UK.
- Soulsby, R.L., & Whitehouse, R.J.S. 2005. *Prediction of ripple properties in shelf seas. Mark 2 Predictor for time evolution*. Report TR 154. HR Wallingford, Wallingford, UK.
- Soulsby, R.L., & Whitehouse, R.J.S.W. 1997. Threshold of sediment motion in coastal environments. *Pages 149–154 of: Proc. Pacific Coasts and Ports '97 Conf., Christchurch*, vol. 1. Univ. Canterbury, New Zealand.
- Steinhorn, I. 1991. Salt flux and evaporation. *Journal of Physical Oceanography*, **21**, 1681–1683.
- Stelling, G.S., & Van Kester, J.A.T.M. 1994. On the approximation of horizontal gradients in sigma coordinates for bathymetry with steep bottom slopes. *International Journal for Numerical Methods in Fluids*, **18**, 915–935.
- Stokes, G.G. 1847. On the theory of oscillatory waves. *Trans. Camb. Phil. Soc*, **8**, 197–229.
- Sutka, R. L., Ostrom, N. E., Ostrom, P. H., & Phanikumar, M. S. 2004. Stable nitrogen isotope dynamics of dissolved nitrate in a transect from the North Pacific Subtropical Gyre to the Eastern Tropical North Pacific. *Geochimica et Cosmochimica Acta*, **68**(3), 517–527.
- Svendsen, I.A. 2006. *Introduction to nearshore hydrodynamics*. Advanced Series on Ocean Engineering, vol. 24. World scientific.
- Sweby, P.K. 1984. High resolution schemes using flux limiters for hyperbolic conservation laws. *SIAM Journal of Numerical Analysis*, **21**, 995–1011.
- Talstra, H. 2003. *Numerieke modellering van het lange termijn morfologische gedrag van estuaria*. M.Phil. thesis, TU Delft.

- Tan, Y., Huang, L., Chen, Q., & Huang, X. 2004. Seasonal variation in zooplankton composition and grazing impact on phytoplankton standing stock in the Pearl River Estuary, China. *Continental Shelf Research*, **24**(16), 1949–1968.
- Taylor, A. H., Geider, R. J., & Gilbert, F. J. H. 1997. Seasonal and latitudinal dependencies of phytoplankton carbon-to-chlorophyll a ratios: results of a modelling study. *Marine Ecology Progress Series*, **152**, 51–66.
- Thatcher, M.L., & Harleman, R.F. 1972. *A mathematical model for the prediction of unsteady salinity intrusion in estuaries*. Tech. rept. Dept. Civ. Eng. Mass. Inst. Techn., Cambridge, USA.
- Tian, R. C. 2006. Toward standard parameterization in marine biological modeling. *Ecological Modelling*, **193**, 363–386.
- Uchiyama, Y., McWilliams, J.C., & Restrepo, J.M. 2009. Wave-current interaction in nearshore shear instability analyzed with a vortex force formalism. *Journal of Geophysical Research*, **114**, doi:10.1029/2009JC005135.
- Uchiyama, Y., McWilliams, J.C., & Shchepetkin, A.F. 2010. Wave-current interaction in an oceanic circulation model with a vortex-force formalism: Application to the surf zone. *Ocean Modelling*, **34**, 16–35.
- UDUNITS. 1997. *Udunits software package*. UNIDATA Program Center of the University Corporation for Atmospheric Research.
- Uye, S., & Matsuda, O. 1988. Phosphorus content of zooplankton from the Inland Sea of Japan. *Journal of Oceanography*, **44**(6), 280–286.
- Van Leussen, W. 1994. *Estuarine macroflocs and their role in fine-grained sediment transport*. Ph.D. thesis, University of Utrecht, Netherlands.
- Van Rijn, L.C. 1984a. Sediment pick-up functions. *Journal of Hydraulic Engineering*, **110**, 1494–1502.
- Van Rijn, L.C. 1984b. Sediment transport, Part II: Suspended load transport. *Journal of Hydraulic Engineering*, **110**, 1613–1641.
- Van Rijn, L.C. 1993. *Principles of sediment transport in rivers, estuaries and coastal seas*. Aqua Publications, Amsterdam.
- Van Rijn, L.C. 2007b. Unified view of sediment transport by currents and waves. II: Suspended transport. *Journal of Hydraulic Engineering*, **133**, 668–689.

- Villaret, C., & Trowbridge, J.H. 1991. Effects of stratification by suspended sediments on turbulent shear flows. *Journal of Geophysical Research*, **96**, 10659–10680.
- Visser, A.W. 1997. Using random walk models to simulate the vertical distribution of particles in a turbulent water column. *Marine Ecology Progress Series*, **158**, 275–281.
- Wang, S.-D, Shen, Y.-M., Guo, Y-K., & Tang, T. 2008. Three-dimensional numerical simulation for transport of oil spills in seas. *Ocean Engineering*, **35**, 503–510.
- Wiberg, P.L., & Harris, C.K. 1994. Ripple geometry in wave-dominated environments. *Journal of Geophysical Research*, **99**, 775–789.
- Wiberg, P.L., & Rubin, D.M. 1989. Bed roughness produced by saltating sediment. *Journal of Geophysical Research*, **94**, 5011–5016.
- Winterwerp, J.C., & van Kesteren, W.G.M. 2004. *Introduction to the physics of cohesive sediment in the marine environment*. Elsevier Science.
- Wong, G. T. F., Ku, T., Mulholland, M., Tseng, C., & Wang, D. 2007. The SouthEast Asian Time-series Study (SEATS) and the biogeochemistry of the South China Sea – An overview. *Deep-Sea Research II*, **54**, 1434–1447.
- Wu, J. 1980. Wind stress coefficients over the sea surface near neutral conditions – A revisit. *Journal of Physical Oceanography*, **10**, 727–740.
- Wu, W., Wang, S., & Jia, Y. 2000. Nonuniform sediment transport in alluvial rivers. *Journal of Hydraulic Research*, **38**, 427–434.
- Xing, J., & Davies, A.M. 1996. Application of turbulence energy models to the computation of tidal currents and mixing intensities in shelf edge regions. *Journal of Physical Oceanography*, **26**, 417–447.
- Yalin, M.S. 1977. *Mechanics of sediment transport*. Pergamon.
- Yanenko, N.N. 1971. *The method of fractional steps*. New York: Springer Verlag.
- Yool, A., Martin, A. P., Fernandez, C., & Clark, D. R. 200. The significance of nitrification for oceanic new production. *Nature*, **447**, 999–1002.
- Zhang, R., & Xie, J.H. 1993. *Sedimentation research in China: Systematic selections*. China Water and Power Press, Beijing.

- Zillman, J.W. 1972. *A study of some aspects of the radiation and heat budgets of the Southern Hemisphere Oceans.* Tech. rept. Bureau of Meteorology, Dept. of the Interior, Canberra, ACT, Australia.