Lars Umlauf · Hans Burchard · Karsten Bolding

GOTM

Sourcecode and Test Case Documentation

Devel version - pre 4.2

Flow — Mixing — Sediment Transport

Contents

1 Introduction

1.1 What is GOTM?

GOTM is the abbreviation for 'General Ocean Turbulence Model'. It is a one-dimensional water column model for the most important hydrodynamic and thermodynamic processes related to vertical mixing in natural waters. In addition, it has been designed such that it can easily be coupled to 3-D circulation models, and used as a module for the computation of vertical turbulent mixing. The core of the model computes solutions for the one-dimensional versions of the transport equations of momentum, salt and heat. The key component in solving these equations is the model for the turbulent fluxes of these quantities. The strength of GOTM is the vast number of well-tested turbulence models that have been implemented in the code. These models span the range from simple prescribed expressions for the turbulent diffusivities up to complex Reynolds-stress models with several differential transport equations to solve. Even though, evidently, not all turbulence models published in oceanography could be implemented, at least one member of every relevant model family can be found in GOTM (empirical models, energy models, two-equation models, Algebraic Stress Models, K-profile parameterisations, etc).

Besides the classic combination of the hydrodynamic and turbulent part of the model, GOTM has been growing considerably with the years, and new parts have been developed. Sediment transport and the dynamics of sea grass have been added, state-of-the-art numerical schemes have been implemented, and an environment for the assimilation of data and the computation of atmosphere-ocean interactions exists now. In addition, there is a number of scientific research groups that adopted GOTM for their own projects. Even though the modules developed by these groups (biological and bio-geochemical components, air-sea interaction modules, plotting routines, etc) are not part of the core structure of GOTM, as downloadable from our web-site at `www.gotm.net`, they are in most cases available directly from these groups. In that sense, GOTM is an integrated, community based software environment for an almost unlimited range of applications in geophysical turbulence modelling.

1.2 The idea behind GOTM

Computer codes similar to pieces of GOTM can be found at many scientific institutions. However, different researchers have different goals. Some are interested in the development of turbulence models, others in oceanic applications of these models, and yet others want to compare the effects of different turbulence models on different processes in the ocean or in lakes. The attempt to use one of their specialised programs for one's own project resulted in many cases in spending weeks of work for deciphering non-documented FORTRAN lines, scattered with pre-historic fragments of code from more or, in some cases, less talented programmers. Additional time had to be spend for providing components for atmospheric forcing, etc, before the own research project could finally be attacked.

To overcome these problems, the GOTM project was intiated, its purpose being twofold. First, GOTM should provide an integrative environment for all researchers interested in the *application* of a turbulence model in studies of oceanic processes. Such a software should contain a core part for solving transport equations of mean and turbulent quantities, but equally well routines to compute the atmosphere-ocean fluxes from meteorological or measured data, including routines to interpolate and manipulate them. Second, however, GOTM should also be a research tool for those interested in the *development* of turbulence models and numerical algorithms. This implies that GOTM should always contain the state-of-the-art models and algorithms in these disciplines. The current version of GOTM was developed under these premises.

In both cases, a detailed and comprehensible documentation is crucial, and we spent a lot of effort to come close to this goal. All methods and models embedded in GOTM can be traced back to scientific publications, a key requirement for the scientific use of a program. Also, we took great care to make the FORTRAN95 code as safe, easily understandable, and extensible, as possible.

1.3 How to read the documentation

This document is the official scientific documentation of GOTM. Due to the fast and continuing evolution of GOTM, we have been looking for a new and flexible way of giving a comprehensive and up-to-date documentation for GOTM. We decided for the following strategy.

Every module of GOTM is accompanied by an introductory text on the general theory of the subject, including mathematical derivations, bibliographic references, and the definition of the most important variables. These introductory parts, which should give the reader a brief theoretical overview of what is coded in the modules, are expected to be relatively stable. References are given to more comprehensive introductory or advanced media for each subject.

For the actual documentation of the FORTRAN95 code, which is more likely subject to frequent changes and extensions, a different strategy has been followed. For every `module`, internal or external `subroutine` or `function`, a short piece of documentation in LaTeX has been directly written into the code. These fragments of the documentation will be updated every time the code changes. We use a software called PROTEX, which looks into *every* FORTRAN file of GOTM, extracts the LaTeX parts, and compiles some information about the FORTRAN interfaces, public member functions, public data members, defined parameters, etc. All these pieces of information are assembled by PROTEX to yield a nice documentation including table of contents, figures, tables, references, and formulae for each part of the program. The largest part of the report has been created in this way. Note that PROTEX looks for certain key words in the FORTRAN code to organise the structure of the final document. Therefore, don't be confused if you find things like `!DESCRIPTION:`, `!INTERFACE:`, `!PUBLIC DATA MEMBERS:`, etc, in the FORTRAN files. These are always preceded by an exclamation mark, and thus invisible for your FORTRAN compiler.

If you are new to GOTM, we recommend to completely go through the core parts of GOTM described in section 2, section 3, and section 4. In these sections, you will find also references to the relevant introductory literature. The other parts of this documentation should be used like an encyclopedia: you can look up things fast when you need more information about parts of the program. Extra comments can be found in the form of standard FORTRAN comments that should help users to find their way through the lines of the code.

A special status in this documentation has section 12 illustrating particular scenarios prepared for GOTM. This section contains useful information about the theoretical background and the implementation of each scenario currently available in GOTM. Scenarios range from simple test cases, like a turbulent Couette flow, to full oceanic applications including meteorological forcing and comparison to measured data. The most simple scenarios descibed in section 12 serve as a little tutorial, in which the key algorithms of GOTM are introduced in a practical way.

This documentation does not contain information about how to download, compile and run the code and the test cases. All information necessary to run GOTM on a number of well-known platforms is compiled at `www.gotm.net`. If you wish to directly contact to the GOTM developers, please write an e-mail to `gotm-devel@googlegroups.com`. All users of GOTM, who signed up on the GOTM web page, will be on the users' mailing list, `gotm-users@googlegroups.com`. Information about updates, bug fixes, and new versions of GOTM are communicated via this list.

1.4 Acknowledgements

The authors of this report are grateful to the former members of the GOTM Team for their persisting cooperation. These are particularly members from the very first days of GOTM which took place at the Joint Research Centre in Ispra (Italy) in 1998: Manuel Ruiz Villarreal who worked after the Ispra time in Santiago de Compostela (Spain), Lisboa (Portugal), Hamburg (Germany), and Warnemünde (Germany) before he moved back to his home country for working in A Coruña (Spain). Pierre-Phillipe Mathieu who went to Reading (U.K.) for some time before he arrived in Frascati (Italy) recently. We further want to acknowledge those of the almost 200 subscribed users of GOTM from all over the world who helped us to improve GOTM, reported bugs, and motivated us to go on with this zero-budget project. It was also the important role which GOTM played in several projects, mostly funded by the European Commission, which helped a lot to maintain GOTM. These projects were MAS3-CT96-0053 ('PHASE'), MAS3-CT96-0051 ('MTP II-MATER'), MAS3-CT97-0025 ('PROVESS'), and especially CARTUM (Comparative Analysis and Rationalisation of Second-Moment Turbulence Models), a brainstorming activity (MAS3-CT98-0172), which brought together turbulence specialists from all over the world. We are finally grateful to all those other people working on the Public Domain Software without which a project like GOTM would be unthinkable: LaTeX, PROTEX, LINUX and many others.

2 The GOTM main program

2.1 Introduction

The purpose of the main program and its associated module `gotm` is the construction of a solid framework for the interaction of all components of GOTM. Almost no actual computations are carried out inside this part of the program. However, the most important processes are triggered from there: the initialization of all lower-level modules is actuated and the time stepping of the differential equations is managed. Also calls to subroutines responsible for the air-sea interaction and the ouput of the results are managed. Details for each of the (few) routines are given in the following.

2.2 GOTM — the main program

INTERFACE:

```
program main
```

DESCRIPTION:

This is the main program of GOTM. However, because GOTM has been programmed in a modular way, this routine is very short and merely calls internal routines of other modules. Its main purpose is to update the time and to call the internal routines `init_gotm()`, `time_loop()`, and `clean_up()`, which are defined in the module `gotm` as discussed in section 2.3.

USES:

```
use time
use gotm
IMPLICIT NONE
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding & Hans Burchard
```

2.3 Module gotm — the general framework

INTERFACE:

```
module gotm
```

DESCRIPTION:

This is 'where it all happens'. This module provides the internal routines `init_gotm()` to initialise the whole model and `time_loop()` to manage the time-stepping of all fields. These two routines in turn call more specialised routines e.g. of the `meanflow` and `turbulence` modules to delegate the job.
Here is also the place for a few words on FORTRAN 'units' we used. The method of FORTRAN units is quite rigid and also a bit dangerous, but lacking a better alternative we adopted it here. This requires the definition of ranges of units for different purposes. In GOTM we strongly suggest to use units according to the following conventions.

- unit=10 is reserved for reading namelists.
- units 20-29 are reserved for the `airsea` module.
- units 30-39 are reserved for the `meanflow` module.
- units 40-49 are reserved for the `turbulence` module.
- units 50-59 are reserved for the `output` module.
- units 60-69 are reserved for the `extra` modules like those dealing with sediments or sea-grass.
- units 70- are *not* reserved and can be used as you wish.

USES:

```
use meanflow
use observations
use time

use airsea,      only: init_air_sea,do_air_sea,clean_air_sea
use airsea,      only: set_sst,set_ssuv,integrated_fluxes
use airsea,      only: calc_fluxes
use airsea,      only: wind=>w,tx,ty,I_0,cloud,heat,precip,evap
use airsea,      only: bio_albedo,bio_drag_scale

use turbulence,  only: turb_method
use turbulence,  only: init_turbulence,do_turbulence
use turbulence,  only: num,nuh,nus
use turbulence,  only: const_num,const_nuh
use turbulence,  only: gamu,gamv,gamh,gams
use turbulence,  only: kappa
use turbulence,  only: clean_turbulence
```

```
  use kpp,         only: init_kpp,do_kpp,clean_kpp

  use mtridiagonal,only: init_tridiagonal,clean_tridiagonal
  use eqstate,     only: init_eqstate

#ifdef SEAGRASS
  use seagrass
#endif
#ifdef SPM
  use spm_var, only: spm_calc
  use spm, only: init_spm, set_env_spm, do_spm, end_spm
#endif
#ifdef BIO
  use bio
  use bio_fluxes
  use bio_var, only: npar,numc,cc
#endif
#ifdef _FABM_
  use gotm_fabm,only:init_gotm_fabm,init_gotm_fabm_state,set_env_gotm_fabm,do_gotm_fabm,clean_gotm
  use gotm_fabm_input,only:init_gotm_fabm_input,do_gotm_fabm_input
  use gotm_fabm_output,only:init_gotm_fabm_output,do_gotm_fabm_output
#endif

  use output

  IMPLICIT NONE
  private
```

PUBLIC MEMBER FUNCTIONS:

```
  public init_gotm, time_loop, clean_up
```

DEFINED PARAMETERS:

```
  integer, parameter                  :: namlst=10
#ifdef SEAGRASS
  integer, parameter                  :: unit_seagrass=62
#endif
#ifdef SPM
  integer, parameter                  :: unit_spm=64
#endif
#ifdef BIO
  integer, parameter                  :: unit_bio=63
#endif
```

REVISION HISTORY:

```
  Original author(s): Karsten Bolding & Hans Burchard
```

2.3.1 Initialise the model

INTERFACE:

```
   subroutine init_gotm()
```

DESCRIPTION:

This internal routine triggers the initialization of the model. The first section reads the namelists of `gotmrun.nml` with the user specifications. Then, one by one each of the modules are initialised with help of more specialised routines like `init_meanflow()` or `init_turbulence()` defined inside their modules, respectively.
Note that the KPP-turbulence model requires not only a call to `init_kpp()` but before also a call to `init_turbulence()`, since there some fields (fluxes, diffusivities, etc) are declared and the turbulence namelist is read.

USES:

```
  IMPLICIT NONE
```

REVISION HISTORY:

```
   Original author(s): Karsten Bolding & Hans Burchard
```

2.3.2 Manage global time–stepping

INTERFACE:

```
   subroutine time_loop()
```

DESCRIPTION:

This internal routine is the heart of the code. It contains the main time-loop inside of which all routines required during the time step are called. The following main processes are successively triggered.

1. The model time is updated and the output is prepared.
2. Air-sea interactions (flux, SST) are computed.
3. The time step is performed on the mean-flow equations (momentum, temperature).
4. Some quantities related to shear and stratification are updated (shear-number, buoyancy frequency, etc).
5. Turbulence is updated depending on what turbulence closure model has been specified by the user.
6. The results are written to the output files.

Depending on macros set for the Fortran pre-processor, extra features like the effects of sea-grass or sediments are considered in this routine (see section 10).

USES:

```
IMPLICIT NONE
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding & Hans Burchard
```

2.3.3 The run is over — now clean up.

INTERFACE:

```
subroutine clean_up()
```

DESCRIPTION:

This function is just a wrapper for the external routine `close_output()` discussed in section 7. All open files will be closed after this call.

USES:

```
IMPLICIT NONE
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding & Hans Burchard
```

2.3.4 Print the current state of all loaded gotm modules.

INTERFACE:

```
subroutine print_state()
```

DESCRIPTION:

This routine writes the value of all module-level variables to screen.

USES:

```
use airsea,    only: print_state_airsea
use turbulence,only: print_state_turbulence

IMPLICIT NONE
```

REVISION HISTORY:

```
Original author(s): Jorn Bruggeman
```

3 The mean flow model

3.1 Introduction

This module contains the definitions of the most important mean flow variables used in geophysical models. In GOTM, these are

- the mean horizontal velocity components, U and V
- the mean potential temperature, Θ, (or the mean buoyancy, B)
- the mean salinity, S

Note that in general a variable ϕ describing a turbulent field can be decomposed into a mean and a fluctuating part. In GOTM, we use the notation

$$\phi = \langle \phi \rangle + \phi' \ , \tag{1}$$

where $\langle \ \ \rangle$ denotes the ensemble mean and the prime the fluctuating part. In addition, for brevity, we use the following conventions:

$$\begin{array}{rcll} U &=& \langle u \rangle & \text{for the x-velocity} \\ V &=& \langle v \rangle & \text{for the y-velocity} \\ P &=& \langle p \rangle & \text{for the pressure} \\ \Theta &=& \langle \theta \rangle & \text{for the potential temperature} \\ B &=& \langle b \rangle & \text{for the buoyancy} \\ S &=& \langle s \rangle & \text{for the salinity} \end{array} \tag{2}$$

Note that, if not explicitly mentioned, GOTM uses the units kg, m, s, K. Further conventions are introduced in the turbulence chapter section 4. All operations on these meanflow variables are executed and coordinated in the `meanflow` module.

3.1.1 Physics

Due to the one-dimensional character of GOTM, the state-variables listed above are assumed to be horizontally homogeneous, depending only on the vertical z-coordinate. As a consequence, all horizontal gradients have to be taken from observations, or they have to be estimated, parameterised or neglected.
For example, the surface slopes $\partial_x \zeta$ and $\partial_y \zeta$ representing the barotropic pressure-gradients may be determined by means of local observations or results from three-dimensional numerical models. It is also possible to prescribe a time series of the near-bed velocity components for reconstructing the barotropic pressure gradient, see *Burchard* (1999). The implementation of these options for the external pressure gradient is carried out in `extpressure.F90`, described in section 3.7. The internal pressure-gradient, which results from horizontal density gradients, can be prescribed from observations of horizontal gradients of Θ and S or from three-dimensional model results (see `intpressure.F90` in section 3.8). These gradients may also be used for horizontally advecting Θ and S (see section 3.10 and section 3.11).
Another option in GOTM for parameterising the advection of Θ and S is to relax the model results to observations. Evidently, this raises questions about the physical consistency of the model, but it might help to provide a more realistic density field for studies of turbulence dynamics. Nudging is also possible for the horizontal velocity components. This makes sense in order to initialise inertial

oscillations from observed velocity profiles, see section 3.5 and section 3.6. In the momentum equations, advection and horizontal diffusion terms are neglected.
In hydrostatic ocean models, the vertical velocity is calculated by means of the continuity equation, where the horizontal gradients of U and V are needed. Since these are not available or set to zero, the assumption of zero vertical velocity would be consistent. In many applications however, a non-zero vertical velocity is needed in order to reflect the vertical adiabatic motion of e.g. a thermocline. In GOTM, we have thus included the option of prescribing a vertical velocity time series at one height level which might be vertically moving. Vertical velocities at the surface and at the bottom are prescribed according to the kinematic boundary conditions ($w = 0$ at the bottom and $w = \partial_t \zeta$ at the surface), and between these locations and the prescribed vertical velocity at a certain height, linear interpolation is applied, see `updategrid.F90` in section 3.3. This vertical velocity is then used for the vertical advection of all prognostic quantities.
Standard relations according to the law of the wall are used for deriving bottom boundary conditions for the momentum equations (see `friction.F90` in section 3.9). At the sea surface, they have to be prescribed or calculated from meteorological observations with the aid of bulk formulae using the simulated or observed sea surface temperature (see section 5.2). In `stratification.F90` described in section 3.14, the buoyancy b as defined in equation (33) is calculated by means of the UNESCO equation of state (*Fofonoff and Millard* (1983)) or its linearised version. In special cases, the buoyancy may also be calculated from a simple transport equation. `stratification.F90` is also used for calculating the Brunt-Väisälä frequency, N.
The turbulent fluxes are calculated by means of various different turbulence closure models described in great detail in the `turbulence` module, see section 4.7. As a simplifying alternative, mixing can be computed according to the so-called 'convective adjustment' algorithm, see section 3.15.
Furthermore, the vertical grid is also defined in the meanflow module (see `updategrid.F90` in section 3.3). Choices for the numerical grid are so-called σ-coordinates with layers heights having a fixed portion of the water depth throughout the simulation. Equidistant and non-equidistant grids are possible.

3.1.2 Numerics

For the spatial discretisation, the water column is divided into N_i layers of not necessarily equal thickness h_i,

$$h_i = (\gamma_i - \gamma_{i-1})D, \qquad i = 1, \dots, N_i \ , \tag{3}$$

with nondimensional interfaces γ_i with $\gamma_0 = -1$, $\gamma_{i-1} < \gamma_i$ and $\gamma_{N_i} = 0$, see *Burchard and Petersen* (1997).
The discrete values for the mean flow quantities U, V, Θ, and S represent interval means and are therefore located at the centres of the intervals, and the turbulent quantities like k, L, ϵ, ν_t, ν_t', N, P, G, c_μ, and c_μ' are positioned at the interfaces of the intervals (see section 4.7). The indexing is such, that the interface above an interval has the same index as the interval itself. This means that mean flow quantities range from $i = 1, .., N_i$ while turbulent quantities range from $i = 0, .., N_i$ (see figure 1). The staggering of the grid allows for a straight-forward discretisation of the vertical fluxes of momentum and tracers without averaging. However, for the vertical fluxes of e.g. k and ϵ, averaging of the eddy diffusivities is necessary. This is only problematic for the fluxes near the surface and the bottom, where viscosities at the boundaries have to be considered for the averaging. These can however be derived from the law of the wall.
The time stepping is equidistant, based on two time levels and not limited by Courant numbers, because of the absence of advection and an implicit treatment of vertical diffusion, see figure 2. In

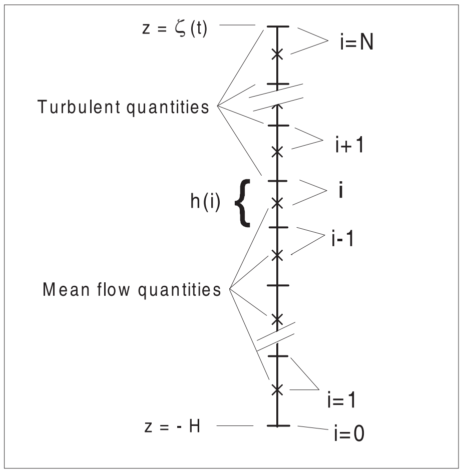

Figure 1: Spatial organisation and indexing of the numerical grid.

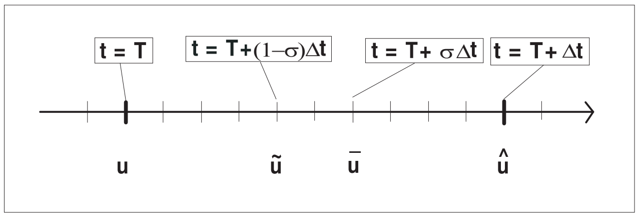

Figure 2: Temporal organisation and indexing of the numerical grid. Here, a time stepping slightly more implicit than the *Crank and Nicolson* (1947) scheme with $\sigma = 0.6$ is shown.

the following, the discretisation of a simple diffusion equation,

$$\frac{\partial X}{\partial t} - \frac{\partial}{\partial z}\left(\nu \frac{\partial X}{\partial z}\right) = 0 \, , \tag{4}$$

will be illustrated for Neumann-type boundary conditions

$$\nu \frac{\partial X}{\partial z} = F_s \qquad \text{for } z = \zeta, \tag{5}$$

and

$$\nu \frac{\partial X}{\partial z} = F_b \qquad \text{for } z = -H. \tag{6}$$

The semi-implicit discretisation of (4) can then be written as

$$\frac{X_{N_i}^{n+1} - X_{N_i}^{n}}{\Delta t} - \frac{F_s - \nu_{N_i-1}^{n} \frac{X_{N_i}^{n+\sigma} - X_{N_i-1}^{n+\sigma}}{0.5(h_{N_i}^{n+1} + h_{N_i-1}^{n+1})}}{h_{N_i}^{n+1}} = \, , \tag{7}$$

$$\frac{X_i^{n+1}-X_i^n}{\Delta t} - \frac{\nu_i^n \frac{X_{i+1}^{n+\sigma}-X_i^{n+\sigma}}{0.5(h_{i+1}^{n+1}+h_i^{n+1})} - \nu_{i-1}^n \frac{X_i^{n+\sigma}-X_{i-1}^{n+\sigma}}{0.5(h_i^{n+1}+h_{i-1}^{n+1})}}{h_i^{n+1}} = 0 \ , \tag{8}$$

$$\frac{X_1^{n+1}-X_1^n}{\Delta t} - \frac{\nu_1^n \frac{X_2^{n+\sigma}-X_1^{n+\sigma}}{0.5(h_2^{n+1}+h_1^{n+1})} - F_b}{h_1^{n+1}} = 0 \ , \tag{9}$$

for $1 < i < N_i$. Here, the semi-implicit time level is defined by

$$X^{n+\sigma} = \sigma X^{n+1} + (1-\sigma)X^n. \tag{10}$$

Thus, for $\sigma = 0$, a fully explicit, for $\sigma = 1$ a fully implicit, and for $\sigma = 0.5$ the *Crank and Nicolson* (1947) second-order scheme are obtained. Figure 2 shows an example for $\sigma = 0.6$. It should be noted that often a time stepping is preferable which is slightly more implicit than the *Crank and Nicolson* (1947) scheme in order to obtain asymptotic stability. The resulting linear system of equations (7) – (9) with tri-diagonal matrix structure is solved by means of the simplified Gaussian elimination.
With the same strategy, a very similar system of equations can be derived for variables located at the interfaces of the grid cells, i.e. variables describing turbulence.

3.2 Module Mean Flow

INTERFACE:

```
   module meanflow
```

DESCRIPTION:

This module provides all variables necessary for the meanflow calculation and also makes the proper initialisations.

USES:

```
   IMPLICIT NONE
   Default all is private.
   private
```

PUBLIC MEMBER FUNCTIONS:

```
   public init_meanflow, clean_meanflow
 ifdef _PRINTSTATE_
   public print_state_meanflow
 endif
```

PUBLIC DATA MEMBERS:

```
   logical, public                              :: grid_ready
   logical, public                              :: init_buoyancy

   coordinate z, layer thicknesses
   REALTYPE, public, dimension(:), allocatable, target  :: ga,z,h,ho

   the velocity components
   REALTYPE, public, dimension(:), allocatable, target  :: u,v,w

   velocity at old time step
   REALTYPE, public, dimension(:), allocatable  :: uo,vo

   potential temperature, salinity
   REALTYPE, public, dimension(:), allocatable, target  :: T,S,rho

   boyancy frequency squared
   (total, from temperature only, from salinity only)
   REALTYPE, public, dimension(:), allocatable  :: NN,NNT,NNS

   shear-frequency squared
   (total, from u only, from v only)
   REALTYPE, public, dimension(:), allocatable  :: SS,SSU,SSV

   buoyancy, short-wave radiation,
   extra production of tke by see-grass etc
```

```
 REALTYPE, public, dimension(:), allocatable  :: buoy,rad,xP

 a dummy array
 (most often used for diffusivities)
 REALTYPE, public, dimension(:), allocatable  :: avh

 grid-related vertical velocity
 REALTYPE, public, dimension(:), allocatable  :: w_grid

 extra friction terms due to e.g. seagrass
 REALTYPE, public, dimension(:), allocatable  :: fric,drag

 shading in the water column
 REALTYPE, public, dimension(:), allocatable, target  :: bioshade

ifdef EXTRA_OUTPUT

 dummies for testing
 REALTYPE, public, dimension(:), allocatable   :: mean1,mean2,mean3,mean4,mean5

endif

 the 'meanflow' namelist
 REALTYPE, public                    :: h0b
 REALTYPE, public                    :: z0s_min
 logical,  public                    :: charnock
 REALTYPE, public                    :: charnock_val
 REALTYPE, public                    :: ddu
 REALTYPE, public                    :: ddl
 integer,  public                    :: grid_method
 REALTYPE, public                    :: c1ad
 REALTYPE, public                    :: c2ad
 REALTYPE, public                    :: c3ad
 REALTYPE, public                    :: c4ad
 REALTYPE, public                    :: Tgrid
 REALTYPE, public                    :: NNnorm
 REALTYPE, public                    :: SSnorm
 REALTYPE, public                    :: dsurf
 REALTYPE, public                    :: dtgrid
 character(LEN=PATH_MAX), public     :: grid_file
 REALTYPE, public                    :: gravity
 REALTYPE, public                    :: rho_0
 REALTYPE, public                    :: cp
 REALTYPE, public                    :: avmolu
 REALTYPE, public                    :: avmolT
 REALTYPE, public                    :: avmolS
 integer,  public                    :: MaxItz0b
 logical,  public                    :: no_shear

 the roughness lengths
```

```
REALTYPE, public                    :: z0b,z0s,za

the coriolis parameter
REALTYPE, public                    :: cori

the friction velocities
REALTYPE, public                    :: u_taub,u_taus

bottom stress
REALTYPE, public, target            :: taub

other stuff
REALTYPE, public                    :: depth0
REALTYPE, public                    :: depth
REALTYPE, public                    :: runtimeu, runtimev
```

DEFINED PARAMETERS:

```
REALTYPE, public, parameter         :: pi=3.141592654
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding & Hans Burchard
```

3.2.1 Initialisation of the mean flow variables

INTERFACE:

```
subroutine init_meanflow(namlst,fn,nlev,latitude)
```

DESCRIPTION:

Allocates memory and initialises everything related to the 'meanflow' component of GOTM.

USES:

```
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer, intent(in)                  :: namlst
character(len=*), intent(in)         :: fn
integer, intent(in)                  :: nlev
REALTYPE, intent(in)                 :: latitude
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding & Hans Burchard
See log for the meanflow module
```

3.2.2 Cleaning up the mean flow variables

INTERFACE:

```
subroutine clean_meanflow()
```

DESCRIPTION:

De-allocates all memory allocated via init_meanflow()

USES:

```
IMPLICIT NONE
```

INPUT PARAMETERS:

REVISION HISTORY:

```
Original author(s): Karsten Bolding & Hans Burchard
See log for the meanflow module
```

3.2.3 Print the current state of the meanflow module.

INTERFACE:

```
subroutine print_state_meanflow()
```

DESCRIPTION:

This routine writes the value of all module-level variables to screen.

USES:

```
IMPLICIT NONE
```

REVISION HISTORY:

```
Original author(s): Jorn Bruggeman
```

3.3 The vertical grid

INTERFACE:

```
subroutine updategrid(nlev,dt,zeta)
```

DESCRIPTION:

This subroutine calculates for each time step new layer thicknesses in order to fit them to the changing water depth. Three different grids can be specified:

1. Equidistant grid with possible zooming towards surface and bottom. The number of layers, `nlev`, and the zooming factors, `ddu`=d_u and `ddl`=d_l, are specified in `gotmmean.nml`. Zooming is applied according to the formula

$$h_k = D\frac{\tanh\left((d_l+d_u)\frac{k}{M}-d_l\right)+\tanh(d_l)}{\tanh(d_l)+\tanh(d_u)}-1 \quad . \tag{11}$$

   From this formula, the following grids are constructed:

   - $d_l = d_u = 0$ results in equidistant discretisations.
   - $d_l > 0, d_u = 0$ results in zooming near the bottom.
   - $d_l = 0, d_u > 0$ results in zooming near the surface.
   - $d_l > 0, d_u > 0$ results in double zooming nea both, the surface and the bottom.

2. Sigma-layers. The fraction that every layer occupies is read-in from file, see `gotmmean.nml`.

3. Cartesian layers. The height of every layer is read in from file, see `gotmmean.nml`. This method is not recommended when a varying sea surface is considered.

Furthermore, vertical velocity profiles are calculated here, if `w_adv_method` is 1 or 2, which has to be chosen in the `w_advspec` namelist in `obs.nml`. The profiles of vertical velocity are determined by two values, the height of maximum absolute value of vertical velocity, `w_height`, and the vertical velocity at this height, `w_adv`. From `w_height`, the vertical velocity is linearly decreasing towards the surface and the bottom, where is value is zero.

USES:

```
use meanflow,     only: grid_ready
use meanflow,     only: depth0,depth
use meanflow,     only: ga,z,h,ho,ddu,ddl,grid_method
use meanflow,     only: NN,SS,w_grid,grid_file,w
use observations, only: zeta_method,w_adv_method
use observations, only: w_adv,w_height,w_adv_discr
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer, intent(in)                 :: nlev
REALTYPE, intent(in)                :: dt,zeta
```

REVISION HISTORY:

```
Original author(s): Hans Burchard & Karsten Bolding
```

3.4 The Coriolis rotation

INTERFACE:

```
subroutine coriolis(nlev,dt)
```

DESCRIPTION:

This subroutine carries out the Coriolis rotation by applying a 2×2 rotation matrix with the angle $f\Delta t$ on the horizontal velocity vector (U, V).

USES:

```
USE meanflow, only: u,v,cori
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer, intent(in)                :: nlev
REALTYPE, intent(in)               :: dt
```

REVISION HISTORY:

```
Original author(s): Hans Burchard & Karsten Bolding
```

3.5 The U-momentum equation

INTERFACE:

```
   subroutine uequation(nlev,dt,cnpar,tx,num,gamu,Method)
```

DESCRIPTION:

This subroutine computes the transport of momentum in x-direction according to

$$\dot{U} = \mathcal{D}_U - g\frac{\partial \zeta}{\partial x} + \int_z^{\zeta} \frac{\partial B}{\partial x}\, dz' - \frac{1}{\tau_R^U}(U - U_{obs}) - C_f U \sqrt{U^2 + V^2} \ , \tag{12}$$

where $\dot{U}$ denotes the material derivative of U, ζ the free surface elevation and B the mean buoyancy defined in (33). $\mathcal{D}_U$ is the sum of the turbulent and viscous transport terms modelled according to

$$\mathcal{D}_U = \frac{\partial}{\partial z}\left((\nu_t + \nu)\frac{\partial U}{\partial z} - \tilde{\Gamma}_U \right) \quad . \tag{13}$$

In this equation, ν_t and ν are the turbulent and molecular diffusivities of momentum, respectively, and $\tilde{\Gamma}_U$ denotes the non-local flux of momentum, see section 4.
Coriolis rotation is accounted for as described in section 3.4. The external pressure gradient (second term on right hand side) is applied here only if surface slopes are directly given. Otherwise, the gradient is computed as described in section 3.7, see *Burchard* (1999). The internal pressure gradient (third term on right hand side) is calculated in `intpressure.F90`, see section 3.8. The fifth term on the right hand side allows for nudging the velocity to observed profiles with the relaxation time scale τ_R^U. This is useful for initialising velocity profiles in case of significant inertial oscillations. Bottom friction is implemented implicitly using the fourth term on the right hand side. Implicit friction may be applied on all levels in order to allow for inner friction terms such as seagrass friction (see section 10.1).
Diffusion is numerically treated implicitly, see equations (7)- (9). The tri-diagonal matrix is solved then by a simplified Gauss elimination. Vertical advection is included, and it must be non-conservative, which is ensured by setting the local variable `adv_mode=0`, see section 8.5 on page 209.

USES:

```
   use meanflow,     only: gravity,avmolu
   use meanflow,     only: h,u,uo,v,w,avh
   use meanflow,     only: drag,SS,runtimeu
   use observations, only: w_adv_method,w_adv_discr
   use observations, only: uProf,vel_relax_tau,vel_relax_ramp
   use observations, only: idpdx,dpdx
   use util,         only: Dirichlet,Neumann
   use util,         only: oneSided,zeroDivergence

   IMPLICIT NONE
```

INPUT PARAMETERS:

```
number of vertical layers
integer, intent(in)                :: nlev

time step (s)
REALTYPE, intent(in)               :: dt

numerical "implicitness" parameter
REALTYPE, intent(in)               :: cnpar

wind stress in x-direction
divided by rho_0 (m^2/s^2)
REALTYPE, intent(in)               :: tx

diffusivity of momentum (m^2/s)
REALTYPE, intent(in)               :: num(0:nlev)

non-local flux of momentum (m^2/s^2)
REALTYPE, intent(in)               :: gamu(0:nlev)

method to compute external
pressure gradient
integer, intent(in)                :: method
```

DEFINED PARAMETERS:

```
REALTYPE, parameter                :: long=1.0D15
```

REVISION HISTORY:

```
Original author(s): Lars Umlauf
                    (re-write after first version of
                     Hans Burchard and Karsten Bolding)
```

3.6 The V-momentum equation

INTERFACE:

```
subroutine vequation(nlev,dt,cnpar,ty,num,gamv,Method)
```

DESCRIPTION:

This subroutine computes the transport of momentum in y-direction according to

$$\dot{V} = \mathcal{D}_V - g\frac{\partial \zeta}{\partial y} + \int_z^\zeta \frac{\partial B}{\partial y}\,dz' - \frac{1}{\tau_R^V}(V - V_{obs}) - C_f V\sqrt{U^2+V^2}\ , \tag{14}$$

where $\dot{V}$ denotes the material derivative of V, ζ the free surface elevation and B the mean buoyancy defined in (33). $\mathcal{D}_V$ is the sum of the turbulent and viscous transport terms modelled according to

$$\mathcal{D}_V = \frac{\partial}{\partial z}\left((\nu_t + \nu)\frac{\partial V}{\partial z} - \tilde{\Gamma}_V\right) \quad . \tag{15}$$

In this equation, ν_t and ν are the turbulent and molecular diffusivities of momentum, respectively, and $\tilde{\Gamma}_V$ denotes the non-local flux of momentum, see section 4.
Coriolis rotation is accounted for as described in section 3.4. All other terms are completely analogous to those described in section 3.5.

USES:

```
use meanflow,     only: gravity,avmolu
use meanflow,     only: h,v,vo,u,w,avh
use meanflow,     only: drag,SS,runtimev
use observations, only: w_adv_method,w_adv_discr
use observations, only: vProf,vel_relax_tau,vel_relax_ramp
use observations, only: idpdy,dpdy
use util,         only: Dirichlet,Neumann
use util,         only: oneSided,zeroDivergence

IMPLICIT NONE
```

INPUT PARAMETERS:

```
number of vertical layers
integer, intent(in)                :: nlev

time step (s)
REALTYPE, intent(in)               :: dt

numerical "implicitness" parameter
REALTYPE, intent(in)               :: cnpar

wind stress in y-direction
divided by rho_0 (m^2/s^2)
```

```
REALTYPE, intent(in)                :: ty

diffusivity of momentum (m^2/s)
REALTYPE, intent(in)                :: num(0:nlev)

non-local flux of momentum (m^2/s^2)
REALTYPE, intent(in)                :: gamv(0:nlev)

method to compute external
pressure gradient
integer, intent(in)                 :: method
```

DEFINED PARAMETERS:

```
REALTYPE, parameter                 :: long=1.0D15
```

REVISION HISTORY:

```
Original author(s): Lars Umlauf
                    (re-write after first version of
                     Hans Burchard and Karsten Bolding)
```

3.7 The external pressure-gradient

INTERFACE:

```
subroutine extpressure(method,nlev)
```

DESCRIPTION:

This subroutine calculates the external pressure-gradient. Two methods are implemented here, relating either to the velocity vector at a given height above bed prescribed or to the vector for the vertical mean velocity. In the first case, `dpdx` and `dpdy` are x- and y-components of the prescribed velocity vector at the height `h_press` above the bed. The velocity profile will in this routive be shifted by a vertically constant vector such that the resulting profile has an (interpolated) velocity at `h_press` which is identical to the prescribed value. In the second case, `dpdx` and `dpdy` are x- and y-components of the prescribed vertical mean velocity vector, and `h_press` is not used. Here the velocity profile is shifted in such a way that the resulting mean velocty vector is identical to `dpdx` and `dpdy`.
For both cases, this is a recalculation of the external pressure gradient, since at all points the same acceleration has been applied in this operator split method.
If the external pressure-gradient is prescribed by the surface slope, then it is directly inserted in (12) and (14).
For details of this method, see *Burchard* (1999).

USES:

```
use meanflow,     only: u,v,h
use observations, only: dpdx,dpdy,h_press
IMPLICIT NONE
```

INPUT PARAMETERS:

```
method to compute external
pressure gradient
integer, intent(in)                :: method

number of vertical layers
integer, intent(in)                :: nlev
```

REVISION HISTORY:

```
Original author(s): Hans Burchard & Karsten Bolding
```

3.8 The internal pressure-gradient

INTERFACE:

```
subroutine intpressure(nlev)
```

DESCRIPTION:

With the hydrostatic assumption

$$\frac{\partial P}{\partial z} + g\langle\rho\rangle = 0 \; , \tag{16}$$

where P denotes the mean pressure, $g = 9.81\mathrm{m\,s}^{-2}$ the gravitational acceleration and $\langle\rho\rangle$ the mean density, the components of the pressure-gradient may be expressed as

$$-\frac{1}{\rho_0}\frac{\partial P}{\partial x} = -g\frac{\partial \zeta}{\partial x} + \int_z^{\zeta} \frac{\partial B}{\partial x}\, dz' \tag{17}$$

and

$$-\frac{1}{\rho_0}\frac{\partial P}{\partial y} = -g\frac{\partial \zeta}{\partial y} + \int_z^{\zeta} \frac{\partial B}{\partial y}\, dz' \; , \tag{18}$$

where ζ is the surface elevation and B the mean buoyancy as defined in (33).
The first term on the right hand side in (17) and (18) is the external pressure-gradient due to surface slopes, and the second the internal pressure-gradient due to the density gradient. The internal pressure-gradient will only be established by gradients of mean potential temperature Θ and mean salinity S. Sediment concentration is assumed to be horizontally homogeneous.
In this subroutine, first, the horizontal buoyancy gradients, $\partial_x B$ and $\partial_y B$, are calculated from the prescribed gradients of salinity, $\partial_x S$ and $\partial_y S$, and temperature, $\partial_x \Theta$ and $\partial_y \Theta$, according to the finite-difference expression

$$\frac{\partial B}{\partial x} \approx \frac{B(S+\Delta_x S, \Theta + \Delta_x \Theta, P) - B(S,\Theta,P)}{\Delta x} \; , \tag{19}$$

$$\frac{\partial B}{\partial y} \approx \frac{B(S+\Delta_y S, \Theta + \Delta_y \Theta, P) - B(S,\theta,P)}{\Delta y} \; , \tag{20}$$

where the defintions

$$\Delta_x S = \Delta x \partial_x S \; , \quad \Delta_y S = \Delta y \partial_y S \; , \tag{21}$$

and

$$\Delta_x \Theta = \Delta x \partial_x \Theta \; , \quad \Delta_y \Theta = \Delta y \partial_y \Theta \; , \tag{22}$$

have been used. Δx and Δy are ”small enough”, but otherwise arbitrary length scales. The buoyancy gradients computed with this method are then vertically integrated according to (17) and (18).
The horizontal salinity and temperature gradients have to supplied by the user, either as constant values or as profiles given in a file (see `obs.nml`).

USES:

```
use meanflow,      only: T,S
use meanflow,      only: gravity,rho_0,h
use observations,  only: dsdx,dsdy,dtdx,dtdy
use observations,  only: idpdx,idpdy,int_press_method
use eqstate,       only: eqstate1
IMPLICIT NONE
```

INPUT PARAMETERS:

```
number of vertical layers
integer, intent(in)                 :: nlev
```

REVISION HISTORY:

```
Original author(s): Hans Burchard & Karsten Bolding
```

3.9 The vertical friction

INTERFACE:

```
   subroutine friction(kappa,avmolu,tx,ty)
```

DESCRIPTION:

This subroutine updates the bottom roughness

$$z_0^b = 0.1\frac{\nu}{u_*^b} + 0.03h_0^b + z_a \quad . \tag{23}$$

The first term on the right hand side of (23) represents the limit for hydraulically smooth surfaces, the second term the limit for completely rough surfaces. Note that the third term, z_a, is the contribution of suspended sediments to the roughness length, see *Smith and McLean* (1977). It is updated during calls to the sediment-routines.
The law-of-the-wall relations are used to compute the friction velocity

$$u_*^b = r\sqrt{U_1^2 + V_1^2}\ , \tag{24}$$

where U_1 and V_1 are the components of the mean velocity at the center of the lowest cell. We used the abbreviation

$$r = \frac{\kappa}{\ln\left(\frac{0.5h_1+z_0^b}{z_0^b}\right)}\ , \tag{25}$$

where κ is the von Kármán constant and the index '1' indicates values at the center of the first grid box at the bottom (version 1). Another expression for r can be derived using the mean value of the velocity in the lowest grid box, and not its value in the middle of the box (version 2). Also this method is supported in `friction()` and can be activated by uncommenting one line in the code.
If no breaking surface waves are considered, the law of the wall also holds at the surface. The surface roughness length may be calculated according to the *Charnock* (1955) formula,

$$z_0^s = \alpha\frac{(u_*^s)^2}{g} \quad . \tag{26}$$

The model constant α is read in as `charnock_val` from the `meanflow` namelist.

USES:

```
   use meanflow,      only: h,z0b,h0b,MaxItz0b,z0s,za
   use meanflow,      only: u,v,rho,gravity
   use meanflow,      only: u_taub,u_taus,drag,taub
   use meanflow,      only: charnock,charnock_val,z0s_min

   IMPLICIT NONE
```

INPUT PARAMETERS:

```
   REALTYPE, intent(in)                :: kappa,avmolu,tx,ty
```

REVISION HISTORY:

```
   Original author(s): Hans Burchard & Karsten Bolding
```

3.10 The temperature equation

INTERFACE:

```
   subroutine temperature(nlev,dt,cnpar,I_0,heat,nuh,gamh,rad)
```

DESCRIPTION:

This subroutine computes the balance of heat in the form

$$\dot{\Theta} = \mathcal{D}_\Theta - \frac{1}{\tau_R^\Theta}(\Theta - \Theta_{obs}) + \frac{1}{C_p \rho_0} \frac{\partial I}{\partial z} \ , \tag{27}$$

where $\dot{\Theta}$ denotes the material derivative of the mean potential temperature Θ, and $\mathcal{D}_\Theta$ is the sum of the turbulent and viscous transport terms modelled according to

$$\mathcal{D}_\Theta = \frac{\partial}{\partial z} \left(\left(\nu_t^\Theta + \nu^\Theta \right) \frac{\partial \Theta}{\partial z} - \tilde{\Gamma}_\Theta \right) \quad . \tag{28}$$

In this equation, ν_t^Θ and ν^Θ are the turbulent and molecular diffusivities of heat, respectively, and $\tilde{\Gamma}_\Theta$ denotes the non-local flux of heat, see section 4.
Horizontal advection is optionally included (see `obs.nml`) by means of prescribed horizontal gradients $\partial_x \Theta$ and $\partial_y \Theta$ and calculated horizontal mean velocities U and V. Relaxation with the time scale τ_R^Θ towards a precribed profile Θ_{obs}, changing in time, is possible.
The sum of latent, sensible, and longwave radiation is treated as a boundary condition. Solar radiation is treated as an inner source, $I(z)$. It is computed according the exponential law (see *Paulson and Simpson* (1977))

$$I(z) = I_0 \bigg(A e^{z/\eta_1} + (1-A) e^{z/\eta_2} \bigg) B(z). \tag{29}$$

The absorbtion coefficients η_1 and η_2 depend on the water type and have to be prescribed either by means of choosing a *Jerlov* (1968) class (see *Paulson and Simpson* (1977)) or by reading in a file through the namelist `extinct` in `obs.nml`. The damping term due to bioturbidity, $B(z)$ is calculated in the biogeochemical routines, see section **??**.
Diffusion is numerically treated implicitly, see equations (7)- (9). The tri-diagonal matrix is solved then by a simplified Gauss elimination. Vertical advection is included, and it must be non-conservative, which is ensured by setting the local variable `adv_mode=0`, see section 8.5 on page 209.

USES:

```
   use meanflow,     only: avmolt,rho_0,cp
   use meanflow,     only: h,u,v,w,T,S,avh
   use meanflow,     only: bioshade
   use observations, only: dtdx,dtdy,t_adv
   use observations, only: w_adv_discr,w_adv_method
   use observations, only: tprof,TRelaxTau
   use observations, only: A,g1,g2
   use util,         only: Dirichlet,Neumann
   use util,         only: oneSided,zeroDivergence

   IMPLICIT NONE
```

INPUT PARAMETERS:

```
number of vertical layers
integer, intent(in)                :: nlev

time step (s)
REALTYPE, intent(in)               :: dt

numerical "implicitness" parameter
REALTYPE, intent(in)               :: cnpar

surface short waves radiation  (W/m^2)
REALTYPE, intent(in)               :: I_0

surface heat flux (W/m^2)
(negative for heat loss)
REALTYPE, intent(in)               :: heat

diffusivity of heat (m^2/s)
REALTYPE, intent(in)               :: nuh(0:nlev)

non-local heat flux (Km/s)
REALTYPE, intent(in)               :: gamh(0:nlev)
```

OUTPUT PARAMETERS:

```
shortwave radiation profile (W/m^2)
REALTYPE                           :: rad(0:nlev)
```

REVISION HISTORY:

```
Original author(s): Hans Burchard & Karsten Bolding
```

3.11 The salinity equation

INTERFACE:

```
subroutine salinity(nlev,dt,cnpar,nus,gams)
```

DESCRIPTION:

This subroutine computes the balance of salinity in the form

$$\dot{S} = \mathcal{D}_S - \frac{1}{\tau_R^S}(S - S_{obs}) \ , \tag{30}$$

where $\dot{S}$ denotes the material derivative of the salinity S, and $\mathcal{D}_S$ is the sum of the turbulent and viscous transport terms modelled according to

$$\mathcal{D}_S = \frac{\partial}{\partial z}\left(\left(\nu_t^S + \nu^S\right) \frac{\partial S}{\partial z} - \tilde{\Gamma}_S \right) \quad . \tag{31}$$

In this equation, ν_t^S and ν^S are the turbulent and molecular diffusivities of salinity, respectively, and $\tilde{\Gamma}_S$ denotes the non-local flux of salinity, see section 4. In the current version of GOTM, we set $\nu_t^S = \nu_t^\Theta$ for simplicity.
Horizontal advection is optionally included (see `obs.nml`) by means of prescribed horizontal gradients $\partial_x S$ and $\partial_y S$ and calculated horizontal mean velocities U and V. Relaxation with the time scale τ_R^S towards a precribed (changing in time) profile S_{obs} is possible.
Inner sources or sinks are not considered. The surface freshwater flux is given by means of the precipitation - evaporation data read in as $P - E$ through the `airsea.nml` namelist:

$$\mathcal{D}_S = S(P - E), \qquad \text{at } z = \zeta, \tag{32}$$

with $P-E$ given as a velocity (note that $\mathcal{D}_S$ is the flux in the direction of z, and thus positive for a *loss* of salinity) . Diffusion is numerically treated implicitly, see equations (7)-(9). The tri-diagonal matrix is solved then by a simplified Gauss elimination. Vertical advection is included, and it must be non-conservative, which is ensured by setting the local variable `adv_mode=0`, see section 8.5 on page 209.

USES:

```
use meanflow,     only: avmols
use meanflow,     only: h,u,v,w,S,avh
use observations, only: dsdx,dsdy,s_adv
use observations, only: w_adv_discr,w_adv_method
use observations, only: sprof,SRelaxTau
use airsea,       only: precip,evap
use util,         only: Dirichlet,Neumann
use util,         only: oneSided,zeroDivergence

IMPLICIT NONE
```

INPUT PARAMETERS:

```
number of vertical layers
integer, intent(in)                :: nlev

 time step (s)
REALTYPE, intent(in)               :: dt

numerical "implicitness" parameter
REALTYPE, intent(in)               :: cnpar

diffusivity of salinity (m^2/s)
REALTYPE, intent(in)               :: nus(0:nlev)

non-local salinity flux (psu m/s)
REALTYPE, intent(in)               :: gams(0:nlev)
```

REVISION HISTORY:

```
Original author(s): Hans Burchard & Karsten Bolding
```

3.12 The buoyancy equation

INTERFACE:

```
subroutine buoyancy(nlev,dt,cnpar,nub,gamb)
```

DESCRIPTION:

This subroutine solves a transport equation for the mean potential buoyancy,

$$B = -g \frac{\langle \rho \rangle - \rho_0}{\rho_0} \ , \tag{33}$$

where g is the accelaration of gravity, and $\langle \rho \rangle$ and ρ_0 are the mean potential density and the reference density, respectively. A simplified transport equation for B can be written as

$$\dot{B} = \mathcal{D}_B \ , \tag{34}$$

where $\dot{B}$ denotes the material derivative of B, and $\mathcal{D}_b$ is the sum of the turbulent and viscous transport terms modelled according to

$$\mathcal{D}_B = \frac{\partial}{\partial z} \left((\nu_t^B + \nu^B) \frac{\partial B}{\partial z} - \tilde{\Gamma}_B \right) \quad . \tag{35}$$

In this equation, ν_t^B and ν^B are the turbulent and molecular diffusivities of buoyancy, respectively, and $\tilde{\Gamma}_B$ denotes the non-local flux of buoyancy, see section 4. In the current version of GOTM, we set $\nu_t^B = \nu_t^\Theta$ for simplicity. Source and sink terms are completely disregarded, and thus (34) mainly serves as a convenient tool for some idealized test cases in GOTM.
Diffusion is treated implicitly in space (see equations (7)- (9)), and then solved by a simplified Gauss elimination. Vertical advection is included, and it must be non-conservative, which is ensured by setting the local variable `adv_mode=0`, see section 8.5 on page 209.

USES:

```
use meanflow,      only: h,w,buoy,T,avh,init_buoyancy
use meanflow,      only: w_grid,grid_method
use observations,  only: b_obs_NN,b_obs_surf,b_obs_sbf
use observations,  only: w_adv_discr,w_adv_method
use util,          only: Dirichlet,Neumann
use util,          only: oneSided,zeroDivergence
IMPLICIT NONE
```

INPUT PARAMETERS:

```
number of vertical layers
integer, intent(in)                 :: nlev

 time step (s)
REALTYPE, intent(in)                :: dt

numerical "implicitness" parameter
```

```
REALTYPE, intent(in)                :: cnpar

diffusivity of buoyancy (m^2/s)
REALTYPE, intent(in)                :: nub(0:nlev)

non-local buoyancy flux (m^2/s^3)
REALTYPE, intent(in)                :: gamb(0:nlev)
```

REVISION HISTORY:

```
Original author(s): Hans Burchard & Karsten Bolding
```

3.13 Calculation of the vertical shear

INTERFACE:

```
subroutine shear(nlev,cnpar)
```

DESCRIPTION:

The (square of the) shear frequency is defined as

$$M^2 = \left(\frac{\partial U}{\partial z}\right)^2 + \left(\frac{\partial V}{\partial z}\right)^2 \quad . \tag{36}$$

It is an important parameter in almost all turbulence models. The U- and V-contributions to M^2 are computed using a new scheme which guarantees conservation of kinetic energy for the conversion from mean to turbulent kinetic energy, see *Burchard* (2002a). The shear is calculated by dividing the energy-consistent form of the shear production (see equation (14) by *Burchard* (2002a), but note the typo in that equation) by the eddy viscosity. The correct form of the right hand side of equation (14) of *Burchard* (2002a) should be:

$$\begin{aligned}
(D_{kin})_j &= \frac{\nu_{j+1/2}}{2}\frac{\sigma\left(\hat U_{j+1}-\hat U_j\right)\left(\hat U_{j+1}-U_j\right)+(1-\sigma)\left(U_{j+1}-U_j\right)\left(U_{j+1}-\hat U_j\right)}{(z_{j+1/2}-z_{j-1/2})(z_{j+1}-z_j)}\\
&+\frac{\nu_{j-1/2}}{2}\frac{\sigma\left(\hat U_j-\hat U_{j-1}\right)\left(U_j-\hat U_{j-1}\right)+(1-\sigma)\left(U_j-U_{j-1}\right)\left(\hat U_j-U_{j-1}\right)}{(z_{j+3/2}-z_{j+1/2})(z_{j+1}-z_j)}\\
&= P^l_{j+1/2}+P^u_{j-1/2},
\end{aligned} \tag{37}$$

with the mean kinetic energy dissipation, $(D_{kin})_j$. The two terms on the right hand side are the contribution of energy dissipation from below the interface at $j+1/2$ and the contribution from above the interface at $j-1/2$. With (37), an energy-conserving discretisation of the shear production at $j+1/2$ should be

$$P_{j+1/2} = P^l_{j+1/2}+P^u_{j+1/2}, \tag{38}$$

such that a consistent discretisation of the square of the shear in x-direction should be

$$\begin{aligned}
\left(\frac{\partial U}{\partial z}\right)^2 &\approx \frac{P_{j+1/2}}{\nu_{j+1/2}}\\
&= \frac{1}{2}\frac{\sigma\left(\hat U_{j+1}-\hat U_j\right)\left(\hat U_{j+1}-U_j\right)+(1-\sigma)\left(U_{j+1}-U_j\right)\left(U_{j+1}-\hat U_j\right)}{(z_{j+1/2}-z_{j-1/2})(z_{j+1}-z_j)}\\
&+\frac{1}{2}\frac{\sigma\left(\hat U_{j+1}-\hat U_j\right)\left(U_{j+1}-\hat U_j\right)+(1-\sigma)\left(U_{j+1}-U_j\right)\left(\hat U_{j+1}-U_j\right)}{(z_{j+3/2}-z_{j+1/2})(z_{j+1}-z_j)}.
\end{aligned} \tag{39}$$

The V-contribution is computed analogously. The shear obtained from (39) plus the V-contribution is then used for the computation of the turbulence shear production, see equation (148).

USES:

```
use meanflow,   only: h,u,v,uo,vo
use meanflow,   only: SS,SSU,SSV

IMPLICIT NONE
```

INPUT PARAMETERS:

```
number of vertical layers
integer,  intent(in)                :: nlev

numerical "implicitness" parameter
REALTYPE, intent(in)                :: cnpar
```

REVISION HISTORY:

```
Original author(s): Lars Umlauf
```

3.14 Calculation of the stratification

INTERFACE:

```
   subroutine stratification(nlev,buoy_method,dt,cnpar,nub,gamB)
```

DESCRIPTION:

This routine computes the mean potential density, $\langle\rho\rangle$, the mean potential buoyancy, B, defined in (33), and the mean buoyancy frequency,

$$N^2 = -\frac{g}{\rho_0}\frac{\partial\rho}{\partial z} = \frac{\partial B}{\partial z} \ , \tag{40}$$

which is based on potential density or buoyancy such that for $N^2 = 0$, the entropy is constant in the whole water column and mixing does not work against buoyancy forces. If GOTM used as a turbulence library in your own three-dimensional model, you have to insure that the N^2 computed by you, and passed to the turbulence routines in GOTM, is consistent with the concept of potential density and your equation of state.
The mean potential density is evaluated from the equation of state, (244), according to

$$\langle\rho\rangle = \hat{\rho}(\Theta, S, P_R) \ , \tag{41}$$

where Θ denotes the mean potential temperature, S the mean salinity and P_R the mean reference pressure. The buoyancy frequency defined in (40) can be decomposed into contributions due to potential temperature and salinity stratification,

$$N^2 = N_\Theta^2 + N_S^2 \ , \tag{42}$$

where we introduced the quantities

$$N_\Theta^2 = -\frac{g}{\rho_0}\frac{\partial\rho}{\partial z}\bigg|_S = g\alpha(\Theta, S, P_R)\frac{\partial\Theta}{\partial z} \ , \tag{43}$$

with the thermal expansion coefficient defined in (246), and

$$N_S^2 = -\frac{g}{\rho_0}\frac{\partial\rho}{\partial z}\bigg|_\Theta = -g\beta(\Theta, S, P_R)\frac{\partial S}{\partial z} \ , \tag{44}$$

with the saline contraction coefficient defined in (247). It is important to note that in the actual code the reference pressure, P_R, has been replaced by the (approximate) hydrostatic pressure. Only if this dependence is replaced by the constant reference pressure at the surface in the equation of state, see section 8.8, the model is truely based on potential temperature and density. Otherwise, the model is based on *in-situ* quantities.
Alternatively to the procedure outlined above, depending on the values of the parameter `buoy_method`, the buoyancy may be calculated by means of the transport equation (34). This equation then replaces the computation of Θ and S and is only recommended for idealized studies.

USES:

```
use meanflow,   only: h,S,T,buoy,rho
use meanflow,   only: NN,NNT,NNS
use meanflow,   only: gravity,rho_0
use eqstate,    only: eqstate1
IMPLICIT NONE
```

INPUT PARAMETERS:

```
number of vertical layers
integer,  intent(in)                :: nlev

method to compute buoyancy
integer,  intent(in)                :: buoy_method

 time step (s)
REALTYPE, intent(in)                :: dt

numerical "implicitness" parameter
REALTYPE, intent(in)                :: cnpar

diffusivity of buoyancy (m^2/s)
REALTYPE, intent(in)                :: nub(0:nlev)

non-local buoyancy flux (m^2/s^3)
REALTYPE, intent(in)                :: gamb(0:nlev)
```

OUTPUT PARAMETERS:

REVISION HISTORY:

```
Original author(s): Karsten Bolding & Hans Burchard
```

3.15 Convective adjustment

INTERFACE:

```
subroutine convectiveadjustment(nlev,num,nuh,const_num,const_nuh, &
                                buoy_method,g,rho_0)
```

DESCRIPTION:

In this subroutine, convective adjustment is performed for the temperature, Θ, and the salinity, S, or alternatively for the buoyancy, B, if a dynamic equation is solved for this quantity. Beginning from the first interface below the surface, the water column is checked for static instability. If the Brunt-Väisälä frequency squared, N^2, is negative, the two adjacent boxes are completely mixed. The stability for the interface below this homogenised upper part of the water column is then analysed, and, if needed, mixing is performed again. By doing so, the water column is scanned until the first interface with statically stable or neutral stratification or the bottom is reached. An equation of state described in section 8.8 is used for calculating the Brunt-Väisälä frequency.
The constant values `const_num` and `const_nuh` are then imposed for the eddy viscosity ν_t and the eddy diffusivity ν_t', respectively.

USES:

```
use meanflow, only: h,t,s,buoy,NN
use eqstate, only: eqstate1
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer, intent(in)                :: nlev,buoy_method
REALTYPE, intent(in)               :: g,rho_0
REALTYPE, intent(in)               :: const_num,const_nuh
```

OUTPUT PARAMETERS:

```
REALTYPE, intent(out)              :: num(1:nlev),nuh(1:nlev)
```

REVISION HISTORY:

```
Original author(s): Hans Burchard & Karsten Bolding
```

4 The turbulence model

To close the differential equations for momentum, heat, and salt, parameterisations of the turbulent fluxes of momentum, $\langle u'w' \rangle$, heat $\langle w'\theta' \rangle$, and salinity $\langle w's' \rangle$ are required. Since turbulence 'feels' the effects of temperature and salinity fluctuations essentially through buoyancy fluctuations, only the buoyancy flux, $\langle w'b' \rangle$, is discussed in the following. The assumptions under which one can infer the fluxes of heat and salinity from the buoyancy flux are addressed below.

4.1 Introduction

There are different types and levels of closure models available in GOTM to compute the vertical turbulent fluxes. Simple models rely on the idea that theses fluxes can be computed as the product of a positive turbulent diffusivity and a mean flow gradient. Contributions to the fluxes that are not 'down-gradient', are summarized in so-called counter-gradient terms. Using these assumptions, the fluxes of momentum and buoyancy can be expressed as

$$\langle u'w' \rangle = -\nu_t \frac{\partial u}{\partial z} + \tilde{\Gamma}_U \ , \quad \langle v'w' \rangle = -\nu_t \frac{\partial v}{\partial z} + \tilde{\Gamma}_V \ , \quad \langle w'b' \rangle = -\nu_t^B \frac{\partial B}{\partial z} + \tilde{\Gamma}_B \ , \tag{45}$$

where $\tilde{\Gamma}_{(U,V,B)}$ denote the counter-gradient fluxes. They can be important under very strong stratification and in the case of convection. Note, that the current version of GOTM identifies the diffusivities of heat and salt with ν_t^B (see section 3.10 and section 3.11).
Using an analogy to the kinetic theory of gases, the vertical turbulent diffusivities, ν_t and ν_t^B, are often assumed to be the product of a typical velocity scale of turbulence, q, times a typical length scale, l, see *Tennekes and Lumley* (1972). The velocity scale q can e.g. be identified with the average value of the turbulent fluctuations expressed by the turbulent kinetic energy, $k = q^2/2$. Then, the diffusivities of momentum and heat can be written as

$$\nu_t = c_\mu k^{\frac{1}{2}} l \ , \quad \nu_t^B = c'_\mu k^{\frac{1}{2}} l \ , \tag{46}$$

where the dimensionless quantities c_μ and $c_\mu{}'$ are usually referred to as the 'stability functions'. Depending on the level of turbulent closure, these stability functions can be either constants, empirical functions, or functions of some non-dimensional flow parameters resulting from a higher-order turbulence model. The same applies to the counter-gradient fluxes $\tilde{\Gamma}_{(U,V,B)}$ defined in (45).
There are different possibilities in GOTM to compute the scales q (or k) and l appearing in (46). According to the level of complexity, they are ordered in GOTM in the following fashion.

1. Both, k and l are computed from algebraic relations. The algebraic equation for k is based on a simplified form of the transport equation of the turbulent kinetic energy. The equation for the length-scale may result from different approaches. The most simple models assume an empirically motivated, prescribed vertical distribution of the length-scale. This level of closure corresponds to the 'level 2' model of *Mellor and Yamada* (1982), but also to more recent approaches, see *Cheng et al.* (2002). Algebraic models are an over-simplification in numerous situations.

2. At the next level, k is computed from the differential transport equation for the turbulent kinetic energy ('energy models'). As before, the length-scale is computed from an empirically or theoretically based relation. Models of this type are quite popular in geophysical modelling. A description is given in section 4.19.

3. In the so-called two-equation models, both, k and l, are computed from differential transport equations. As before, k follows from the transport equation of the turbulent kinetic energy. Now, however, also the length-scale is determined from a differential transport equation. This equation is usually not directly formulated for the length-scale, but for a related, length-scale determining variable. Presently, there are different possibilities for the length-scale determining variables implemented in GOTM, such as the rate of dissipation, ϵ, or the product kl. They are discussed in section 4.7.9.

   The main advantage of the two-equation models is their greater generality. There are, for example, a number of fundamental flows which cannot be reproduced with an algebraically prescribed length-scale. Examples are the temporal decay of homogeneous turbulence, the behaviour of turbulence in stratified homogeneous shear flows, and the spatial decay of shear-free turbulence from a planar source. A discussion of these points is given in section 4.7.3 and section 4.7.4. Also see *Umlauf et al.* (2003) and *Umlauf and Burchard* (2003).

In addition to the hierarchy of turbulence models in terms of their methods used to compute the turbulent kinetic energy and the length-scale, GOTM also supports an ordering scheme according to the extent to which transport equations for the turbulent fluxes are solved.

1. At the lowest level of this scheme, it is postulated that $c_\mu = c_\mu^0$ and $c'_\mu = c'^0_\mu$ are constant. Because these models implicitly assume an isotropic tensor relation between the velocity gradient and the tensor of the Reynolds-stresses, they usually fail in situations of strong anisotropy, most notably in stably stratified, curved or shallow flows. In unstratified flows with balanced aspect ratios (which seldom occur in nature), however, they may compute reasonable results. Models of this type are referred to as the 'standard' models in the following.

2. Some problems associated with standard versions of the models can be ameliorated by making c_μ and c'_μ empirical functions of one or several significant non-dimensional flow parameters. At this level, the simplest approach would be to formulate empirical relations suggested from observations in the field or in the laboratory. An example of such a relation is the model of *Schumann and Gerz* (1995) which has been implemented in GOTM (see section 4.29).

3. Another, more consistent, approach results from the solution of simplified forms of the transport equations for the Reynolds-stresses and the turbulent heat fluxes in addition to the transport equations for k and the length-scale determining variable. Surprisingly, it turns out that under some assumptions, and after tedious algebra, the turbulent fluxes computed by these models can be expressed by (46). The important difference is, however, that the existence of vertical eddy diffusivities is not a postulate, but a consequence of the model. The stability functions c_μ and c'_μ can be shown to become functions of some non-dimensional numbers like

$$\alpha_M = \frac{k^2}{\epsilon^2} M^2 \ , \quad \alpha_N = \frac{k^2}{\epsilon^2} N^2 \ , \quad \alpha_b = \frac{k k_b}{\epsilon^2} \ , \tag{47}$$

   with the shear-frequency, M, and the buoyancy frequency, N, computed as described in section 3.5 and section 3.14, respectively. k and k_b are the turbulent kinetic energy and the buoyancy variance, respectively and ϵ denotes the rate of dissipation.

   The most well-known models of this type have been implemented into GOTM. An up-to-date account of their derivation can be found in *Canuto et al.* (2001). Their evaluation for the oceanic mixed layer has been extensively discussed by *Burchard and Bolding* (2001).

4. Even more complete models include further differential equations for the buoyancy variance and for some or all of the turbulent fluxes. These models cannot be reduced to the form (46). The derivation of models of the type discussed in the latter two points are reviewed in section 4.2

Evidently, this short introduction cannot serve as an introductory text on one-point turbulence modelling. It serves merely as a place to define the most important quantities and relations used in this manual. Readers not familiar with this subject will certainly feel the need for a more in-depth discussion. An excellent introduction to turbulence is still the book of *Tennekes and Lumley* (1972). A modern and detailed approach to one and two-equation models for unstratified flows is given in the book of *Wilcox* (1998), and the effects of stratification are discussed e.g. by *Rodi* (1987) and by *Burchard* (2002b).

4.2 Second-order models

Since one-point second-order models are an essential part of GOTM, this section is devoted to a detailed discussion of the derivation and the properties of these models. Second-order models result from the full or approximate solution of the transport equations for the turbulent fluxes like $\langle u'u' \rangle$, $\langle u'w' \rangle$, $\langle w'b' \rangle$, etc. Model equations for the turbulent momentum fluxes follow directly from the Navier-Stokes equations. The derivation of these equations for stratified and rotating fluids is discussed e.g. in *Sander* (1998).
Considering the one-point correlations for the velocity fluctuations u'_i, the momentum fluxes can be expressed as

$$\dot{\langle u'_i u'_j \rangle} - \mathcal{D}_{ij} = P_{ij} + G_{ij} + F_{ij} + \Phi_{ij} - \epsilon_{ij} \ , \tag{48}$$

where $\mathcal{D}_{ij}$ is the sum of the viscous and turbulent transport terms and $\dot{\langle \cdots \rangle}$ denots the material derivative of the ensemble average. The shear-production, P_{ij}, and the buoyancy production, G_{ij}, on the right hand side are defined as

$$P_{ij} = -\langle u'_i u'_m \rangle \frac{\partial U_j}{\partial x_m} - \langle u'_j u'_m \rangle \frac{\partial U_i}{\partial x_m} \ , \quad G_{ij} = \delta_{i3} \langle u'_j b' \rangle + \delta_{j3} \langle u'_i b' \rangle \ , \tag{49}$$

where b' is the fluctuating part of the buoyancy, defined analogously to the mean buoyancy, B, in (33). The tensor of the dissipation rate is assumed to be isotropic, leading to $\epsilon_{ij} = 2/3 \epsilon \delta_{ij}$. Φ_{ij} denotes the pressure redistribution terms discussed below. The influence of the Coriolis-acceleration can be summarized in the tensor F_{ij} which is, however, neglected in the current version of GOTM. The contraction of (48) yields the equation for the turbulent kinetic energy, (152), with production terms defined by

$$P = \frac{1}{2} P_{ii} \ , \quad G = \frac{1}{2} G_{ii} \quad . \tag{50}$$

Similar to (48), the transport equation for the turbulent buoyancy flux is given by

$$\dot{\langle u'_i b' \rangle} - \mathcal{D}^b_i = -\langle u'_i u'_m \rangle \frac{\partial B}{\partial x_m} - \langle u'_m b' \rangle \frac{\partial U_i}{\partial x_m} + F^b_i + 2\delta_{i3} k_b + \Phi^b_i - \epsilon^b_i \ , \tag{51}$$

where $\mathcal{D}^b_i$ denotes the viscous and turbulent transport terms, see *Sander* (1998). For the dissipation, one has $\epsilon^b_i = 0$, following from isotropy. The redistribution terms Φ^b_i are discussed below. As in (48), the Coriolis term F^b_i is neglected in the current version of GOTM.

Note that k_b is half the buoyancy variance and relates to the turbulent potential energy, E_p, according to

$$k_b = \langle b'^2 \rangle / 2 = E_p N^2 \ , \tag{52}$$

where the square of the buoyancy frequency, N^2, is defined in (40).
The crucial point in (48) is the model for the pressure-strain correlation. The most popular models in engineering trace back to suggestions by *Launder et al.* (1975) and *Gibson and Launder* (1976). With the modifications suggested of *Speziale et al.* (1991), this model can be written as

$$\Phi_{ij} = -c_1 \tau_u^{-1} k \, b_{ij} + c_2 k S_{ij} + c_3 k \Sigma_{ij} + c_4 k Z_{ij} + c_5 k N_{ij} + c_6 \Gamma_{ij} \ , \tag{53}$$

usually extended by the last term to account for the effects of buoyancy, see *Gibson and Launder* (1976), *Gibson and Launder* (1978). The model (53) is expressed here in terms of the dimensionless tensor of the stress anisotropies,

$$b_{ij} = \frac{\langle u_i' u_j' \rangle}{2k} - \frac{1}{3} \delta_{ij} \ , \tag{54}$$

and two traceless and symmetric tensors,

$$\Sigma_{ij} = S_{im} b_{mj} + S_{jm} b_{mi} - \frac{2}{3} S_{mn} b_{mn} \delta_{ij} \ , \quad Z_{ij} = W_{im} b_{mj} + W_{jm} b_{mi} \ , \tag{55}$$

which depend on the symmetric and the anti-symmetric parts of the velocity gradient,

$$S_{ij} = \frac{1}{2} \left(L_{ij} + L_{ji} \right) \ , \quad W_{ij} = \frac{1}{2} \left(L_{ij} - L_{ji} \right) \quad \text{with} \quad L_{ij} = \frac{\partial U_i}{\partial x_j} \quad . \tag{56}$$

Buoyancy enters via the symmetric and traceless tensor

$$\Gamma_{ij} = - \left(G_{ij} - \frac{2}{3} G \delta_{ij} \right) \ , \tag{57}$$

with G_{ij} as defined in (49). In view of the derivation of Explicit Algebraic Models (EASMs), the models implemented in GOTM neglect the term N_{ij} on the right hand side of (53), which is non-linear in b_{ij}, see *Speziale et al.* (1991). c_1–c_6 are model constants. In geophysical applications, in contrast to engineering, virually all authors used $c_1^* = 0$ in (53). In GOTM, the return-to-isotropy time scale τ_u is identified with the dynamic dissipation time scale

$$\tau = \frac{k}{\epsilon} \ , \tag{58}$$

which is a reasonable model assumption in many applications (*Canuto et al.* (2001), *Jin et al.* (2003)).
For Explicit Algebraic Heat Flux Models, a quite general model for the pressure buoyancy-gradient correlation appearing in (51) can be written as

$$\begin{aligned} \Phi_i^b \quad = \quad & -c_{b1} \tau_b^{-1} \, \langle u_i' b' \rangle + c_{b2} S_{ij} \langle u_j' b' \rangle + c_{b3} W_{ij} \langle u_j' b' \rangle \\ & + c_{b4} \langle u_i' u_j' \rangle \frac{\partial B}{\partial x_j} - 2 c_{b5} k_b \delta_{i3} \ , \end{aligned} \tag{59}$$

where $\tau_b = \tau$ is adopted for the return-to-isotropy time scale.

The models (53) and (59) correspond to some recent models used in theoretical and engineering studies (*So et al.* (2003), *Jin et al.* (2003)), and generalize all *explicit* models so far adopted by the geophysical community (see *Burchard* (2002b), *Burchard and Bolding* (2001)). With all model assumptions inserted, (48) and (51) constitute a closed system of 9 coupled differential equations, provided the dissipation time scale τ and the buoyancy variance k_b are known. Models for the latter two quantities and simplifying assumptions reducing the differential equations to algebraic expressions are discussed in the following subsection.

4.3 Algebraic Models

The key assumptions in deriving algebraic models have been formulated by *Rodi* (1976) and *Gibson and Launder* (1976). These authors suggested to simplify the right hand sides of (48) and (51) according to

$$\dot{\langle u_i'u_j' \rangle} - \mathcal{D}_{ij} = \frac{\langle u_i'u_j' \rangle}{k}\left(\dot{k} - \mathcal{D}_k\right) \ , \quad \dot{\langle u_i'b' \rangle} - \mathcal{D}_i^b = \frac{\langle u_i'b' \rangle}{2}\left(\frac{\dot{k} - \mathcal{D}_k}{k} + \frac{\dot{k}_b - \mathcal{D}_b}{k_b}\right) \ , \tag{60}$$

which are reasonable approximations in many situations. Moreover, (60) can be shown to hold exactly in stably stratified, homogeneous shear flows, when the flow approaches the so-called weak-equilibrium limit, see *Shih et al.* (2000). Using (60) and the pressure-strain model (53), it can be shown after some algebra that the transport equations for the momentum flux (48) reduces to

$$\mathcal{N} b_{ij} = -a_1 \overline{S}_{ij} - a_2 \overline{\Sigma}_{ij} - a_3 \overline{Z}_{ij} - a_4 \overline{N}_{ij} - a_5 \overline{\Gamma}_{ij} \tag{61}$$

in dimensionless form. The a_i relate to the coefficients used in (53) according to $a_1 = 2/3 - c_2/2$, $a_2 = 1 - c_3/2$, $a_3 = 1 - c_4/2$, $a_4 = c_5/2$, and $a_5 = 1/2 - c_6/2$. The dimensionless, traceless and symmetric tensors appearing on the right hand side of (61) are defined as

$$\overline{S}_{ij} = \frac{k}{\epsilon} S_{ij} \ , \quad \overline{\Sigma}_{ij} = \frac{k}{\epsilon} \Sigma_{ij} \ , \quad \overline{Z}_{ij} = \frac{k}{\epsilon} Z_{ij} \quad . \tag{62}$$

Additionally,

$$\overline{\Gamma}_{ij} = \Gamma_{ij}/\epsilon = \begin{pmatrix} -\frac{2}{3}\gamma_3 & 0 & \gamma_1 \\ 0 & -\frac{2}{3}\gamma_3 & \gamma_2 \\ \gamma_1 & \gamma_2 & \frac{4}{3}\gamma_3 \end{pmatrix} \ , \quad \gamma_i = -\frac{\langle u_i'b' \rangle}{\epsilon} \tag{63}$$

has been introduced in (61) for convenience. Here, the γ_i correspond to the *mixing efficiencies* in each coordinate direction, respectively. Note, that the vertical component,

$$\gamma_3 = \gamma = -\frac{\langle w'b' \rangle}{\epsilon} = -\frac{G}{\epsilon} = \frac{R_f}{1 - R_f} \ , \quad R_f = -\frac{G}{P} \ , \tag{64}$$

can be identified with the classical mixing efficiency used in many studies of stratified fluids.
Most authors proceed know in deriving, with the help of (60), a dimensionless equation for the normalised turbulent buoyancy flux, $\zeta_i = \langle u_i'b' \rangle / \sqrt{(k k_b)}$, see *So et al.* (2002), *Jin et al.* (2003). It can be shown, however, that the resulting algebraic equations alternatively can be expressed, without further assumptions, in the form of equations for the mixing efficiencies,

$$\mathcal{N}_b \gamma_i = -a_{b1} \overline{S}_{ij} \gamma_j - a_{b2} \overline{W}_{ij} \gamma_j + a_{b3} b_{ij} \overline{N}_j + \frac{1}{3} a_{b3} \overline{N}_i - a_{b4} \overline{T} \delta_{i3} \quad . \tag{65}$$

Since efficiencies γ_i are the primary variables appearing on the right hand side of (61) through the presence of the tensor $\overline{\Gamma}_{ij}$ defined in (63), and since they are variables with a clear physical interpretation, we prefer (65) to a mathematicall equivalent equation for the normalised buoyancy flux, ζ_i.
The new dimensionless quantities entering the problem via (65) are

$$\overline{N}_i = \frac{k^2}{\epsilon^2}\frac{\partial B}{\partial x_i} \; , \quad \overline{T} = \frac{k k_b}{\epsilon^2} \quad . \tag{66}$$

Note that the vertical component of $\overline{N}_i$ can be identified with the square of the buoyancy frequency, N^2, made dimensionless with the dynamic dissipation time scale $\tau = k/\epsilon$.
(61) and (65) are linear in b_{ij} and γ_i, with a non-linear coupling introduced by the terms

$$\begin{aligned} \mathcal{N} &= \frac{P+G}{\epsilon} + \frac{c_1}{2} - 1 \\ \mathcal{N}_b &= \frac{1}{2}\left(\frac{P+G}{\epsilon} - 1\right) + c_{b1} + \frac{1}{2r}\left(\frac{P_b}{\epsilon_b} - 1\right) \quad . \end{aligned} \tag{67}$$

The production-to-dissipation ratios appearing in these expression are exclusively related to known quantities and thus introduce no new independent variables. However, the time scale ratio,

$$r = \frac{k_b}{\epsilon_b}\frac{\epsilon}{k} \tag{68}$$

needs to be described.
(61) and (65) are a system of 9 coupled algebraic equations for the anisotropies b_{ij} and the mixing efficiencies γ_i, depending solely on the non-dimensional tensors $\overline{S}_{ij}$, $\overline{W}_{ij}$, the vector $\overline{N}_i$, and the scalar $\overline{T}$. This system is linear, if $\mathcal{N}$ and $\mathcal{N}_b$ are treated as knowns and if the nonlinear term N_{ij} in (61) is neglected, $a_4 = 0$. No closed solution of the complete system in three dimensions has been reported so far in the literature. Nevertheless, separate solutions in three dimensions for (61) and (65), respectively, have been reported (see *Jin et al.* (2003) and the references therein).
In geophysical applications, the system (61) and (65) can be considerably simplified by assuming that the fluid is horizontally homogeneous (boundary layer approximation), and closed solutions can be obtained (see *Cheng et al.* (2002)). The procedure to obtain such solutions is discussed in the following subsection.

4.4 Explicit models for vertical shear and stratification

In the following, we restrict ourselves to flows with vertical shear and stratification, and assume that mean quantities are horizontally homogeneous. Under these conditions, (66) yiels $\overline{N}_1 = \overline{N}_2 = 0$ and

$$\overline{N}_3 = \frac{k^2}{\epsilon^2}\frac{\partial B}{\partial z} = \frac{k^2}{\epsilon^2}N^2 \quad . \tag{69}$$

The velocity gradient simplifies to

$$L_{ij} = \begin{pmatrix} 0 & 0 & S_U \\ 0 & 0 & S_V \\ 0 & 0 & 0 \end{pmatrix} \; , \tag{70}$$

where $S_U = \partial U/\partial z$ and $S_V = \partial V/\partial z$ are the vertical shear in U and V, respectively.
Under these conditions, and using the conventions

$$\overline{S}_U = \frac{k}{\epsilon} S_U \ , \quad \overline{S}_V = \frac{k}{\epsilon} S_V \ , \quad \overline{N}^2 = \overline{N_3} \ , \tag{71}$$

(61) reduces to

$$\begin{aligned}
\mathcal{N} b_{11} &= -\left(\frac{a_2}{3} + a_3\right) b_{13}\overline{S}_U + \frac{2}{3} a_2 b_{23}\overline{S}_V + \frac{2}{3} a_5 \gamma_3 \ , \\
\mathcal{N} b_{22} &= -\left(\frac{a_2}{3} + a_3\right) b_{23}\overline{S}_V + \frac{2}{3} a_2 b_{13}\overline{S}_U + \frac{2}{3} a_5 \gamma_3 \ , \\
\mathcal{N} b_{33} &= -\left(\frac{a_2}{3} - a_3\right) b_{13}\overline{S}_U - \left(\frac{a_2}{3} - a_3\right) b_{23}\overline{S}_V - \frac{4}{3} a_5 \gamma_3 \ , \\
\mathcal{N} b_{12} &= -\frac{a_2 + a_3}{2} b_{13}\overline{S}_V - \frac{a_2 + a_3}{2} b_{23}\overline{S}_U \ , \\
\mathcal{N} b_{13} &= -\frac{a_2 - a_3}{2} b_{11}\overline{S}_U - \frac{a_2 + a_3}{2} b_{33}\overline{S}_U - \frac{a_2 - a_3}{2} b_{12}\overline{S}_V - \frac{1}{2} a_1 \overline{S}_U - a_5\gamma_1 \ , \\
\mathcal{N} b_{23} &= -\frac{a_2 - a_3}{2} b_{22}\overline{S}_V - \frac{a_2 + a_3}{2} b_{33}\overline{S}_V - \frac{a_2 - a_3}{2} b_{12}\overline{S}_U - \frac{1}{2} a_1 \overline{S}_V - a_5\gamma_2 \quad .
\end{aligned} \tag{72}$$

Similarly, for the mixing efficiencies, (65) yields

$$\begin{aligned}
\mathcal{N}_b \gamma_1 &= -\frac{a_{b1} + a_{b2}}{2} \gamma_3 \overline{S}_U + a_{b3} b_{13} \overline{N}^2 \ , \\
\mathcal{N}_b \gamma_2 &= -\frac{a_{b1} + a_{b2}}{2} \gamma_3 \overline{S}_V + a_{b3} b_{23} \overline{N}^2 \ , \\
\mathcal{N}_b \gamma_3 &= -\frac{a_{b1} - a_{b2}}{2} \gamma_1 \overline{S}_U - \frac{a_{b1} - a_{b2}}{2} \gamma_2 \overline{S}_V + a_{b3} b_{33} \overline{N}^2 + \frac{a_{b3}}{3} \overline{N}^2 - a_{b4} \overline{T} \quad .
\end{aligned} \tag{73}$$

In geophysical applications, a reasonable assumption is $P_b = \epsilon_b$ to elimmate the dependence of (73) on $\overline{T}$. From (160), using (64) and (68), it follows that $\overline{T}$ can be expressed in the form

$$\overline{T} = r \gamma_3 \overline{N}^2 \quad . \tag{74}$$

With the help of (74), the last of (73) can be re-written as

$$\mathcal{N}_b \gamma_3 = -\frac{a_{b1} - a_{b2}}{2} \gamma_1 \overline{S}_U - \frac{a_{b1} - a_{b2}}{2} \gamma_2 \overline{S}_V + a_{b3} b_{33} \overline{N}^2 + \frac{a_{b3}}{3} \overline{N}^2 - a_{b5} \gamma_3 \overline{N}^2 \quad . \tag{75}$$

Note that the new parameter $a_{b5} = r a_{b4}$ depends on the time scale ratio, r, and is, in general, not constant. Nevertheless, constant $r = c_b$ is frequently assumed (see below).
In the general case, (72) and (73) can be inverted directly to yield a solution of the form

$$b_{13} = -\frac{1}{2} \hat{c}_\mu \overline{S}_U \ , \quad b_{23} = -\frac{1}{2} \hat{c}_\mu \overline{S}_V \ , \quad \gamma_3 = \hat{c}'_\mu \overline{N}^2 - \Gamma \ , \tag{76}$$

from which, by insertion into (72) and (73), all other quantities can be determined. Since $\mathcal{N}$ and $\mathcal{N}_b$ defined in (67) have been treated as known, the solution is not yet completely explicit. In the numerical scheme of GOTM, they are updated from their values at past time steps. By identifying

$$\nu_t = \hat{c}_\mu \frac{k^2}{\epsilon} \ , \quad \nu_t' = \hat{c}_\mu' \frac{k^2}{\epsilon} \ , \quad \tilde{\Gamma} = \epsilon \Gamma \ , \tag{77}$$

(76) corresponds in form exactly to (45). Note that, adopting the equilibrium assumption (74), the dependence on Γ drops in (76). From (46) and (77), and using the definition of the dissipation rate (155), it is clear that

$$\hat{c}_\mu = (c_\mu^0)^3 c_\mu \ , \quad \hat{c}_\mu' = (c_\mu^0)^3 c_\mu' \ , \qquad . \tag{78}$$

The structure of the dimensionless parameter functions apearing in (76) is given by

$$\hat{c}_\mu = \frac{N_n}{D} \ , \quad \hat{c}_\mu' = \frac{N_b}{D} \ , \quad \Gamma = \frac{N_\Gamma}{D} \ , \tag{79}$$

where the numerators and the denominator are polynomials of the square of the shear number, $\alpha_M = \overline{S}^2 = \overline{S}_U^2 + \overline{S}_V^2$, the square of the buoyancy number, $\alpha_N = \overline{N}^2$, the mixed scalar, $\alpha_B = \overline{T}$, and the functions $\mathcal{N}$ and $\mathcal{N}_b$. The latter two functions depend on the production-to-dissipation ratios for k and k_b, which for vertical shear and stratification can be written as

$$\begin{aligned} \frac{P}{\epsilon} &= -2b_{13}\overline{S}_U - 2b_{23}\overline{S}_V = \hat{c}_\mu \overline{S}^2 \ , \\ \frac{G}{\epsilon} &= -\gamma_3 = -\hat{c}_\mu' \overline{N}^2 + \Gamma \ , \qquad . \\ \frac{P_b}{\epsilon_b} &= -\frac{G}{\epsilon}\frac{\epsilon}{\epsilon_b} N^2 = -r \frac{G}{\epsilon} \frac{\overline{N}^2}{\overline{T}} \end{aligned} \tag{80}$$

Once k and k_b (and their dissipation ratios, ϵ and ϵ_b) are known, also the time scale ratio r defined in (68) can be computed, and the problem can be solved. Different possibilities to derive these quantities are discussed in the following.

4.4.1 Equilibrium states

Some authors use simplifying assumptions to derive more compact forms of the expressions for the solution in (76). In the following, a few examples, which are special cases of the general solution discussed here, are reviewed.
In deriving their version of the general solution (76), *Canuto et al.* (2001) e.g. assumed $P_b = \epsilon_b$ and constant r. Under these conditions, because of (74), the dependence on $\overline{T}$ dissapears, and the counter-gradient term Γ_B in (76) drops. It was further assumed that $P + G = \epsilon$ in (67) only, leading to $\mathcal{N} = (c_1 + c_1^*)/2$ and $\mathcal{N}_b = c_{b1}$. These particularly simple expressions linearize the system, and a fully explicit solution can be obtained, provided k and ϵ are known. *Burchard and Bolding* (2001) adopted the solution of *Canuto et al.* (2001) and complemented it by k and ϵ computed from dynamical equations ('k-ϵ model').
In contrast, *Canuto et al.* (2001) and *Cheng et al.* (2002) decided for a further simplification. They solved (76) with k and ϵ from algebraic expressions. In their case, k followed from the approximation $P + G = \epsilon$ of (152) (see section 4.17), and ϵ from a prescribed length-scale.

Using (76), (79), and (80), it is easy to show that the assumption $(P+G)/\epsilon$ leads to

$$N_n\overline{S}^2 - N_b\overline{N}^2 - D = 0 \ , \tag{81}$$

which is polynomial equation in $\overline{S}$ and $\overline{N}$. This expression can be used to replace one of the latter two variables by the other. An interesting consequence is the fact that all non-dimensional turbulent quantities can be expressed in terms of the Richardson number $Ri = \overline{N}^2/\overline{S}^2$ only. Replacing $\overline{N}^2$ by $\overline{S}^2 Ri$ in (81), a quadratic equation for $\alpha_M = \overline{S}^2$ in terms for Ri can be established (see e.g. *Cheng et al.* (2002). Using the definitions given in section 4.26, this equation can be written as

$$\alpha_M^2\left(-d_5+n_2-(d_3-n_1+n_{b2})\,Ri-(d_4+n_{b1})\,Ri^2\right)+\alpha_M\left(-d_2+n_0-(d_1+n_{b0})\,Ri\right)-d_0=0 \quad . \tag{82}$$

The solution for α_M can, via (81), be used to expressed also $\overline{N}^2$ in terms of Ri. This implies that also the stability functions and hence the complete solution of the problem only depends on Ri. Investigating the solution of the quadratic equation (82), it can be seen that α_M becomes infinite if the factor in front of α_M^2 vanishes. This is the case for a certain value of the Richardson number, $Ri = Ri_c$, following from

$$-d_5+n_2-(d_3-n_1+n_{b2})\,Ri_c-(d_4+n_{b1})\,Ri_c^2=0 \quad . \tag{83}$$

Solutions of this equation for some popular models are given in table 1. For $Ri = Ri_c$, equilibrium models predict complete extinction of turbulence. For non-equilibrium models solving dynamical equations like (152), however, Ri_c has no direct signifcance, because turbulence may be sustainned by turbulent transport and/or the rate term.

GL78	KC94	CHCD01A	CHCD01B	CCH02
0.47	0.24	0.85	1.02	0.96

Table 1: Critical Richardson number for some models

4.4.2 Stability of explicit models

A physically reasonable condition for an explicit second order model expressed the fact that increasing (non-dimensional) shear $\overline{S}$ should lead to increasing vertical shear-anisotropies of turbulence, b_{13} and b_{23}. It has been shown by *Burchard and Deleersnijder* (2001) that a violation of this condition may lead to numerical instabilities of the models.
Mathematically, the shear-condition is expressed by

$$\frac{\partial(b_{13}^2+b_{23}^2)^{\frac{1}{2}}}{\partial\overline{S}} = \frac{1}{2}\frac{\partial\tilde{c}_\mu\overline{S}}{\partial\overline{S}} \geq 0 \ , \tag{84}$$

where (76) has been used. Using the equilibrium form of the stability function described in section 4.26, this condition leads to a cubic equation in $\alpha_M = \overline{S}^2$. A simpler condition can be obtained, when this equation is solved after terms multiplied by d_5 and n_2, which usually are very small, are neglected.
The resulting approximate condition is

$$\alpha_M \leq \frac{d_0n_0+(d_0n_1+d_1n_0)\alpha_N+(d_1n_1+d_4n_0)\alpha_N^2+d_4n_1\alpha_N^3}{d_2n_0+(d_2n_1+d_3n_0)\alpha_N+d_3n_1\alpha_N^2} \quad . \tag{85}$$

Burchard and Deleersnijder (2001) showed that using (85) the most well-known models yield numerically stable results. However, for some models like those of *Mellor and Yamada* (1982) and *Kantha and Clayson* (1994), the limiter (85) is almost always 'active', and hence replaces the actual turbulence model in a questionable way.

4.5 Parameter conversion for other models

Virtually all pressure-redistribution models used in engineering and geophysical applications can be considered as special cases of (53) and (59). However, most authors adopted a different notation and different parameter values. In this section, paramater conversions for the most well-known models are discussed.

4.5.1 The model of *Gibson and Launder* (1978)

The pressure-strain model of this important class of engineering models has been originally suggested by *Launder et al.* (1975). It can be written as

$$\Phi_{ij} = -2\tilde{c}_1 \epsilon b_{ij} - \tilde{c}_2 k S_{ij} - \tilde{c}_3 \left(P_{ij} - \frac{2}{3} P \delta_{ij} \right) - \tilde{c}_4 \left(D_{ij} - \frac{2}{3} P \delta_{ij} \right) + \tilde{c}_6 \Gamma_{ij} \ , \tag{86}$$

where that last term has been added by *Gibson and Launder* (1978) to account for the effects of gravity in stratified fluids. This term is identical to the last term in (53). The new production-of-anisotropy tensor D_{ij} is defined as

$$D_{ij} = -\langle u'_i u'_m \rangle \frac{\partial U_m}{\partial x_j} - \langle u'_j u'_m \rangle \frac{\partial U_m}{\partial x_i} \quad . \tag{87}$$

Using the tensor relations

$$\begin{aligned} P_{ij} &= -2k\Sigma_{ij} - 2kZ_{ij} + \frac{2}{3} P \delta_{ij} - \frac{4}{3} k S_{ij} \ , \\ D_{ij} &= -2k\Sigma_{ij} + 2kZ_{ij} + \frac{2}{3} P \delta_{ij} - \frac{4}{3} k S_{ij} \ , \end{aligned} \tag{88}$$

(86) can be re-written in the form

$$\Phi_{ij} = -2\tilde{c}_1 \epsilon b_{ij} + \left(\frac{4}{3} (\tilde{c}_3 + \tilde{c}_4) - \tilde{c}_2 \right) k S_{ij} + 2 \left(\tilde{c}_3 + \tilde{c}_4 \right) k \Sigma_{ij} + 2 \left(\tilde{c}_3 - \tilde{c}_4 \right) k Z_{ij} + \tilde{c}_6 \Gamma_{ij} \quad . \tag{89}$$

Comparting with (53), the following relations can be estabilished: $c_1 = 2\tilde{c}_1$, $c_2 = 4/3(\tilde{c}_3 + \tilde{c}_4) - \tilde{c}_2$, $c_3 = 2(\tilde{c}_3 + \tilde{c}_4)$, $c_3 = 2(\tilde{c}_3 - \tilde{c}_4)$, $c_5 = 0$, and $c_6 = \tilde{c}_6$.
Gibson and Launder (1978) use a slightly different notation for the pressure-scambling model (59). Their model is somewhat simplified form of the model of *Jin et al.* (2003), which can be written as

$$\begin{aligned} \Phi_i^b \quad = \quad & -\tilde{c}_{b1} \frac{\epsilon}{k} \langle u'_i b' \rangle + \tilde{c}_{b2} L_{ij} \langle u'_j b' \rangle + \tilde{c}_{b3} L_{ji} \langle u'_j b' \rangle \\ & + \tilde{c}_{b4} \langle u'_i u'_j \rangle \frac{\partial B}{\partial x_j} - 2\tilde{c}_{b5} k_b \delta_{i3} \quad . \end{aligned} \tag{90}$$

Using the decomposition of the velocity gradient in its symmetric and anti- symmetric part, (56), the following parameter relation are evident: $c_{b1} = \tilde{c}_{b1}$, $c_{b2} = \tilde{c}_{b2} + \tilde{c}_{b3}$, $c_{b3} = \tilde{c}_{b2} - \tilde{c}_{b3}$, $c_{b4} = \tilde{c}_{b4}$, $c_{b5} = \tilde{c}_{b5}$.

	$\tilde{c}_1$	$\tilde{c}_2$	$\tilde{c}_3$	$\tilde{c}_4$	$\tilde{c}_6$	$\tilde{c}_{b1}$	$\tilde{c}_{b2}$	$\tilde{c}_{b3}$	$\tilde{c}_{b4}$	$\tilde{c}_{b5}$	r
GL78	1.8	0	0.6	0	0.5	3	0.33	0	0	0.33	0.8
GLNEW	1.8	0	0.78	0.2545	0.3	3.28	0.4	0	0	0.4	0.8

Table 2: Some parameter sets for the model of *Gibson and Launder* (1978)

Parameter values for this model are compiled in table 3. 'GLNEW' denotes the revised parameter set for the pressure-strain model given in *Wilcox* (1998) and for the pressure-buoyancy gradient model in *Zhao et al.* (2001).

4.5.2 The model of *Canuto et al.* (2001)

Canuto et al. (2001) and *Cheng et al.* (2002) use a model that is virtually identical to the traditional model of *Launder et al.* (1975) and *Gibson and Launder* (1978). The values of their model parameters and their notation, however, are somewhat different.
Looking for conversion relations, it should be noted that the anisotropy tensor b_{ij}^{CCHD} used by *Canuto et al.* (2001) is twice the tensor defined in (54), $b_{ij}^{\mathrm{CCHD}} = 2kb_{ij}$. Also the dissipative time scale τ^{CCHD} of *Canuto et al.* (2001) is twice the time scal defined in (58), $\tau^{\mathrm{CCHD}} = 2\tau$. If one further notes that the turbulent heat flux $h_i = \langle u_i'\theta'\rangle$ is related to the buoyancy flux according to $\langle u_i'b'\rangle = \alpha g h_i$, relations between the model parameters can be found.
With these relations, equation (10a) of *Canuto et al.* (2001) can be re-written as

$$b_{ij} = -\lambda_1\overline{S}_{ij} - 2\lambda_2\overline{\Sigma}_{ij} - 2\lambda_3\overline{Z}_{ij} - \lambda_4\overline{\Gamma}_{ij} \quad . \tag{91}$$

The return-to-isotropy part of the pressure-strain model of *Canuto et al.* (2001) reads

$$\Phi_{ij} = -\frac{2}{\lambda}\epsilon b_{ij} \; , \tag{92}$$

from which, by comparing with (53), it follows that $c_1 = 2/\lambda$ and $c_1^* = 0$, and hence from (67) $\mathcal{N} = 1/\lambda$. Thus, adopting the relations $a_1 = \lambda_1/\lambda$, $a_2 = 2\lambda_2/\lambda$, $a_3 = 2\lambda_3/\lambda$, $a_4 = 0$, and $a_5 = \lambda_4/\lambda$, (91) corresponds exactly to (61),
Similarly, equation (10a) of *Cheng et al.* (2002) can be re-expressed in the form

$$\frac{\lambda_5}{2}\gamma_i = -\lambda_6\overline{S}_{ij}\gamma_j - \lambda_7\overline{W}_{ij}\gamma_j + 2b_{ij}\overline{N}_j + \frac{2}{3}\overline{N}_i - \lambda_0\overline{T}\delta_{i3} \quad . \tag{93}$$

The somewhat simpler model of *Canuto et al.* (2001) adopts the equilibrium assumption (74), and replaces the last term in (93) by $-\lambda_0 r\gamma_3\overline{N}^2\delta_{i3}$ and, assuming constant r, identifies $\lambda_0 r = \lambda_8$. The time scale ratio r is computed in equation (20a) of *Canuto et al.* (2001).
The return-to-isotropy part of this model (see equation (6c) of *Cheng et al.* (2002)) reads

$$\Phi_i^b = -\frac{\lambda_5}{2}\frac{\epsilon}{k}\langle u_i'b'\rangle \; , \tag{94}$$

from which follows, by comparison with (59) and (67), that $\mathcal{N}_b = c_{b1} = \lambda_5/2$. Comparing (93) with (65) one finds, by inspection, the relations $a_{b1} = \lambda_6$, $a_{b2} = \lambda_7$, $a_{b3} = 2$, $a_{b4} = 2\lambda_0$, and $a_{b5} = 2\lambda_8$. Some parameter sets for this model are compiled in table 3.

	λ	λ_0	λ_1	λ_2	λ_3	λ_4	λ_5	λ_6	λ_7	λ_8
CHCD01A	0.4	2/3	0.107	0.0032	0.0864	0.12	11.9	0.4	0	0.48
CHCD01B	0.4	2/3	0.127	0.00336	0.0906	0.101	11.2	0.4	0	0.318
CCH02	0.4	2/3	0.107	0.0032	0.0864	0.1	11.04	0.786	0.643	0.547

Table 3: Some parameter sets for the model of *Canuto et al.* (2001)

4.5.3 The model of *Mellor and Yamada* (1982)

The pressure-strain model of *Mellor and Yamada* (1982) is expressed in terms of $q^2 = 2k$ and the dissipation length scale $l = q^3/(B_1\epsilon)$, where B_1 is a model constant. The time scale ratio in this model is set to $r = c_b = B_1/B_2$. Using these expression, their model can be re-written as

$$\Phi_{ij} = -\frac{B_1}{3A_1}\epsilon b_{ij} + 4C_1 k S_{ij} \ , \tag{95}$$

which, by comparison with (53), yields $c_1 = B_1/(3A_1)$ and $c_2 = 4C_1$. All other parameters are zero.
Similarly, the pressure-scrambling model of *Mellor and Yamada* (1982) (using the extensions suggested by *Kantha and Clayson* (1994) and *Kantha* (2003)) reads

$$\Phi_i^b = -\frac{B_1}{6A_2}\frac{\epsilon}{k}\langle u_i' b'\rangle + C_2(S_{ij} + W_{ij})\langle u_j' b'\rangle - 2C_3 k_b \delta_{i3} \ , \tag{96}$$

which can be compared to (59) to obtain $c_{b1} = B_1/(6A_2)$ and $c_{b2} = C_2$, $c_{b3} = C_2$, $c_{b5} = C_3$. All other parameters are zero.
Several parameter sets suggested for this model are compiled in table 4

	A_1	A_2	B_1	B_2	C_1	C_2	C_3
MY82	0.92	0.74	16.55	10.1	0.08	0	0
KC94	0.92	0.74	16.55	10.1	0.08	0.7	0.2
K03	0.58	0.62	16.55	11.6	0.038	0.7	0.2

Table 4: Some parameter sets for the model of *Mellor and Yamada* (1982)

4.6 Numerics

The numerical approximation of the turbulence equations is in principle carried out as explained in section 3.1.2. One basic difference is however due to the fact that turbulent quantities are generally non-negative such that it is necessary that the discretised forms of the physical equations retain the principle of non-negativity. A typical model problem would be the following:

$$\frac{\partial X}{\partial t} = P - QX, \quad P, Q > 0 \tag{97}$$

with X denoting any non-negative quantity, P a non-negative source term, QX a non-negative linear sink term, and t denoting time. P and Q may depend on X and t. It can easily be shown that with (97), X remains non-negative for any non-negative initial value X_0 and limited Q. For the q^2l-equation and the ϵ-equation (described in section 4.14 and section 4.15), Q would be proportional to q/l and ϵ/k, repsectively.

A straight-forward, explicit discretisation in time of (97) can be written as

$$\frac{X^{n+1}-X^n}{\Delta t} = P^n - Q^n X^n \tag{98}$$

with the superscripts denoting the old (n) and the new ($n+1$) time level and Δt denoting the time step. In this case, the numerical solution on the new time level would be

$$X_i^{n+1} = X_i^n(1-\Delta t Q_i^n) + \Delta t P_i^n \ , \tag{99}$$

which is negative for negative right hand side of (98), provided that

$$\Delta t > \frac{X^n}{X^n Q^n - P^n} \quad . \tag{100}$$

Since it is computationally unreasonable to restrict the time step in such a way that (100) is avoided, a numerical procedure first published by *Patankar* (1980) is generally applied

$$\frac{X^{n+1}-X^n}{\Delta t} = P^n - Q^n X^{n+1} \ , \tag{101}$$

which yields an always non-negative solution for X^{n+1},

$$X^{n+1} = \frac{X^n + \Delta t P^n}{1+\Delta t Q^n} \quad . \tag{102}$$

Thus, the so-called quasi-implicit formulation (101) by *Patankar* (1980) is a sufficient condition for positivity applied in almost all numerical turbulence models.

4.7 Module turbulence: its all in here ...

INTERFACE:

```
   module turbulence
```

DESCRIPTION:

In this module, variables of the turbulence model and some member functions to manipulate them are defined. The key-functions are `init_turbulence()`, which initialises the model, and `do_turbulence()`, which manages the time step for the whole procedure. These two functions are the only 'public' member functions i.e. they are callable from outside the module. There are many more internal functions, for which descriptions are provided seperately.
It should be pointed out that the turbulence module of GOTM may be used in combination with virtually any shallow-wate 3-D circulation model using a structured grid in the vertical direction. To this end, a clear interface separating the mean flow and the turbulence part of GOTM is required. Vertical columns of the three-dimensional fields have to copied into one-dimensional vectors, which are passed to GOTM. With the help of this information, GOTM updates the turbulent fields and returns one-dimensional vectors of the turbulent diffusivities and/or the turbulent fluxes to the 3-D model. The 'door' between the 3-D model and GOTM is the function `do_turbulence()`, which has been designed with these ideas in mind.

USES:

```
   IMPLICIT NONE

   default: all is private.
   private
```

PUBLIC MEMBER FUNCTIONS:

```
   public init_turbulence, do_turbulence
   public k_bc,q2over2_bc,epsilon_bc,psi_bc,q2l_bc
   public clean_turbulence
 ifdef _PRINTSTATE_
   public print_state_turbulence
 endif
```

PUBLIC DATA MEMBERS:

```
   TKE, rate of dissipation, turbulent length-scale
   REALTYPE, public, dimension(:), allocatable   :: tke,eps,L

   TKE at old time level
   REALTYPE, public, dimension(:), allocatable   :: tkeo

   buoyancy variance and its destruction
   REALTYPE, public, dimension(:), allocatable   :: kb,epsb

   shear and buoyancy production
```

```
 of tke and buoyancy variance
 REALTYPE, public, dimension(:), allocatable   :: P,B,Pb

 turbulent diffusivities
 of momentum, temperature, salinity
 REALTYPE, public, dimension(:), allocatable         :: num
 REALTYPE, public, dimension(:), allocatable, target :: nuh
 REALTYPE, public, dimension(:), allocatable         :: nus

 non-local fluxes of momentum
 REALTYPE, public, dimension(:), allocatable   :: gamu,gamv

 non-local fluxes
 of buoyancy, temperature, salinity
 REALTYPE, public, dimension(:), allocatable   :: gamb,gamh,gams

 non-dimensional  stability functions
 REALTYPE, public, dimension(:), allocatable   :: cmue1,cmue2

 non-dimensional counter-gradient term
 REALTYPE, public, dimension(:), allocatable   :: gam

 alpha_M, alpha_N, and alpha_B
 REALTYPE, public, dimension(:), allocatable   :: as,an,at

 time scale ratio r
 REALTYPE, public, dimension(:), allocatable   :: r

 the gradient Richardson number
 REALTYPE, public, dimension(:), allocatable   :: Rig

 the flux Richardson number
 REALTYPE, public, dimension(:), allocatable   :: xRf

 turbulent velocity variances
 REALTYPE, public, dimension(:), allocatable   :: uu,vv,ww

ifdef EXTRA_OUTPUT

 dummies for testing
 REALTYPE, public, dimension(:), allocatable   :: turb1,turb2,turb3,turb4,turb5

endif

 some additional constants
 REALTYPE, public                              :: cm0,cmsf,cde,rcm, b1

 Prandtl-number in neutrally stratified flow
 REALTYPE, public                              :: Prandtl0
```

```
parameters for wave-breaking
REALTYPE, public                             :: craig_m,sig_e0

the 'turbulence' namelist
integer, public                              :: turb_method
integer, public                              :: tke_method
integer, public                              :: len_scale_method
integer, public                              :: stab_method

the 'bc' namelist
integer, public                              :: k_ubc
integer, public                              :: k_lbc
integer, public                              :: kb_ubc
integer, public                              :: kb_lbc
integer, public                              :: psi_ubc
integer, public                              :: psi_lbc
integer, public                              :: ubc_type
integer, public                              :: lbc_type

the 'turb_param' namelist
REALTYPE, public                             :: cm0_fix
REALTYPE, public                             :: Prandtl0_fix
REALTYPE, public                             :: cw
logical                                      :: compute_kappa
REALTYPE, public                             :: kappa
logical                                      :: compute_c3
REALTYPE                                     :: ri_st
logical,  public                             :: length_lim
REALTYPE, public                             :: galp
REALTYPE, public                             :: const_num
REALTYPE, public                             :: const_nuh
REALTYPE, public                             :: k_min
REALTYPE, public                             :: eps_min
REALTYPE, public                             :: kb_min
REALTYPE, public                             :: epsb_min

the 'generic' namelist
logical                                      :: compute_param
REALTYPE, public                             :: gen_m
REALTYPE, public                             :: gen_n
REALTYPE, public                             :: gen_p
REALTYPE, public                             :: cpsi1
REALTYPE, public                             :: cpsi2
REALTYPE, public                             :: cpsi3minus
REALTYPE, public                             :: cpsi3plus
REALTYPE                                     :: sig_kpsi
REALTYPE, public                             :: sig_psi
REALTYPE                                     :: gen_d
REALTYPE                                     :: gen_alpha
REALTYPE                                     :: gen_l
```

```
the 'keps' namelist
REALTYPE, public                              :: ce1
REALTYPE, public                              :: ce2
REALTYPE, public                              :: ce3minus
REALTYPE, public                              :: ce3plus
REALTYPE, public                              :: sig_k
REALTYPE, public                              :: sig_e
logical,  public                              :: sig_peps

the 'my' namelist
REALTYPE, public                              :: e1
REALTYPE, public                              :: e2
REALTYPE, public                              :: e3
REALTYPE, public                              :: sq
REALTYPE, public                              :: sl
integer,  public                              :: my_length
logical,  public                              :: new_constr

the 'scnd' namelist
integer                                       ::  scnd_method
integer                                       ::  kb_method
integer                                       ::  epsb_method
integer                                       ::  scnd_coeff
REALTYPE ,public                              ::  cc1
REALTYPE, public                              ::  ct1,ctt
REALTYPE, public                              ::  cc2,cc3,cc4,cc5,cc6
REALTYPE, public                              ::  ct2,ct3,ct4,ct5

the a_i's for the ASM
REALTYPE, public                              ::  a1,a2,a3,a4,a5
REALTYPE, public                              ::  at1,at2,at3,at4,at5


the 'iw' namelist
integer,  public                              :: iw_model
REALTYPE, public                              :: alpha
REALTYPE, public                              :: klimiw
REALTYPE, public                              :: rich_cr
REALTYPE, public                              :: numiw
REALTYPE, public                              :: nuhiw
REALTYPE, public                              :: numshear
```

DEFINED PARAMETERS:

```
general outline of the turbulence model
integer, parameter, public                    :: convective=0
integer, parameter, public                    :: algebraic=1
integer, parameter, public                    :: first_order=2
integer, parameter, public                    :: second_order=3
```

```
method to update TKE
integer, parameter, public                    :: tke_local_eq=1
integer, parameter, public                    :: tke_keps=2
integer, parameter, public                    :: tke_MY=3

stability functions
integer, parameter, public                    :: Constant=1
integer, parameter, public                    :: MunkAnderson=2
integer, parameter, public                    :: SchumGerz=3
integer, parameter, public                    :: EiflerSchrimpf=4

method to update length scale
integer, parameter                            :: Parabola=1
integer, parameter                            :: Triangle=2
integer, parameter                            :: Xing=3
integer, parameter                            :: RobertOuellet=4
integer, parameter                            :: Blackadar=5
integer, parameter                            :: BougeaultAndre=6
integer, parameter                            :: ispra_length=7
integer, parameter, public                    :: diss_eq=8
integer, parameter, public                    :: length_eq=9
integer, parameter, public                    :: generic_eq=10

boundary conditions
integer, parameter, public                    :: Dirichlet=0
integer, parameter, public                    :: Neumann=1
integer, parameter, public                    :: viscous=0
integer, parameter, public                    :: logarithmic=1
integer, parameter, public                    :: injection=2

type of second-order model
integer, parameter                            :: quasiEq=1
integer, parameter                            :: weakEqKbEq=2
integer, parameter                            :: weakEqKb=3

method to solve equation for k_b
integer, parameter                            :: kb_algebraic=1
integer, parameter                            :: kb_dynamic=2

method to solve equation for epsilon_b
integer, parameter                            :: epsb_algebraic=1
integer, parameter                            :: epsb_dynamic=2
```

BUGS:

```
      The algebraic equation for the TKE is not safe
      to use at the moment. Use it only in connection
      with the prescribed length-scale profiles. The
      functions report_model() will report wrong things
      for the algebraic TKE equation. To be fixed with
      the next version.
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding, Hans Burchard,
                    Manuel Ruiz Villarreal,
                    Lars Umlauf
```

4.7.1 Initialise the turbulence module

INTERFACE:

```
subroutine init_turbulence(namlst,fn,nlev)
```

DESCRIPTION:

Initialises all turbulence related stuff. This routine reads a number of namelists and allocates memory for turbulence related vectors. The core consists of calls to the the internal functions `generate_model()` and `analyse_model()`, discussed in great detail in section 4.7.3 and section 4.7.4, respectively. The former function computes the model coefficients for the generic two-equation model (see *Umlauf et al.* (2003)) from physically motivated quantities like the von Kármán constant, κ, the decay rate in homogeneous turbulence, d, the steady-state Richardson number, Ri_{st}, and many others. The latter function does the inverse: it computes the physically motivated quantities from the model constants of any model currently available in GOTM. After the call to either function, all relevant model parameters are known to GOTM. Then, the function `report_model()` is called, which displays all results on the screen.

USES:

```
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer,          intent(in)        :: namlst
character(len=*), intent(in)        :: fn
integer,          intent(in)        :: nlev
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding, Hans Burchard,
                    Manuel Ruiz Villarreal,
                    Lars Umlauf
```

4.7.2 Initialize the second-order model

INTERFACE:

```
subroutine init_scnd(scnd_coeff)
```

DESCRIPTION:

This subroutine computes the a_i's defined in (61) and the $a_b i$'s defined in (65) from the model parameters of the pressure redistribution models (53) and (59). Parameter sets from different authors are converted to the GOTM notation according to the relations discussed in section 4.5.

USES:

```
   IMPLICIT NONE
```

INPUT PARAMETERS:

```
   integer, intent(in)                :: scnd_coeff
```

DEFINED PARAMETERS:

```
   REALTYPE,  parameter               :: cc1GL78     =  3.6000
   REALTYPE,  parameter               :: cc2GL78     =  0.8000
   REALTYPE,  parameter               :: cc3GL78     =  1.2000
   REALTYPE,  parameter               :: cc4GL78     =  1.2000
   REALTYPE,  parameter               :: cc5GL78     =  0.0000
   REALTYPE,  parameter               :: cc6GL78     =  0.5000
   REALTYPE,  parameter               :: ct1GL78     =  3.0000
   REALTYPE,  parameter               :: ct2GL78     =  0.3333
   REALTYPE,  parameter               :: ct3GL78     =  0.3333
   REALTYPE,  parameter               :: ct4GL78     =  0.0000
   REALTYPE,  parameter               :: ct5GL78     =  0.3333
   REALTYPE,  parameter               :: cttGL78     =  0.8000

   REALTYPE,  parameter               :: cc1MY82     =  6.0000
   REALTYPE,  parameter               :: cc2MY82     =  0.3200
   REALTYPE,  parameter               :: cc3MY82     =  0.0000
   REALTYPE,  parameter               :: cc4MY82     =  0.0000
   REALTYPE,  parameter               :: cc5MY82     =  0.0000
   REALTYPE,  parameter               :: cc6MY82     =  0.0000
   REALTYPE,  parameter               :: ct1MY82     =  3.7280
   REALTYPE,  parameter               :: ct2MY82     =  0.0000
   REALTYPE,  parameter               :: ct3MY82     =  0.0000
   REALTYPE,  parameter               :: ct4MY82     =  0.0000
   REALTYPE,  parameter               :: ct5MY82     =  0.0000
   REALTYPE,  parameter               :: cttMY82     =  0.6102

   REALTYPE,  parameter               :: cc1KC94     =  6.0000
   REALTYPE,  parameter               :: cc2KC94     =  0.3200
   REALTYPE,  parameter               :: cc3KC94     =  0.0000
   REALTYPE,  parameter               :: cc4KC94     =  0.0000
   REALTYPE,  parameter               :: cc5KC94     =  0.0000
   REALTYPE,  parameter               :: cc6KC94     =  0.0000
   REALTYPE,  parameter               :: ct1KC94     =  3.7280
   REALTYPE,  parameter               :: ct2KC94     =  0.7000
   REALTYPE,  parameter               :: ct3KC94     =  0.7000
```

```
REALTYPE,  parameter                :: ct4KC94     =  0.0000
REALTYPE,  parameter                :: ct5KC94     =  0.2000
REALTYPE,  parameter                :: cttKC94     =  0.6102

REALTYPE,  parameter                :: cc1LDOR96   =  3.0000
REALTYPE,  parameter                :: cc2LDOR96   =  0.8000
REALTYPE,  parameter                :: cc3LDOR96   =  2.0000
REALTYPE,  parameter                :: cc4LDOR96   =  1.1180
REALTYPE,  parameter                :: cc5LDOR96   =  0.0000
REALTYPE,  parameter                :: cc6LDOR96   =  0.5000
REALTYPE,  parameter                :: ct1LDOR96   =  3.0000
REALTYPE,  parameter                :: ct2LDOR96   =  0.3333
REALTYPE,  parameter                :: ct3LDOR96   =  0.3333
REALTYPE,  parameter                :: ct4LDOR96   =  0.0000
REALTYPE,  parameter                :: ct5LDOR96   =  0.3333
REALTYPE,  parameter                :: cttLDOR96   =  0.8000

REALTYPE,  parameter                :: cc1CHCD01A  =  5.0000
REALTYPE,  parameter                :: cc2CHCD01A  =  0.8000
REALTYPE,  parameter                :: cc3CHCD01A  =  1.9680
REALTYPE,  parameter                :: cc4CHCD01A  =  1.1360
REALTYPE,  parameter                :: cc5CHCD01A  =  0.0000
REALTYPE,  parameter                :: cc6CHCD01A  =  0.4000
REALTYPE,  parameter                :: ct1CHCD01A  =  5.9500
REALTYPE,  parameter                :: ct2CHCD01A  =  0.6000
REALTYPE,  parameter                :: ct3CHCD01A  =  1.0000
REALTYPE,  parameter                :: ct4CHCD01A  =  0.0000
REALTYPE,  parameter                :: ct5CHCD01A  =  0.3333
REALTYPE,  parameter                :: cttCHCD01A  =  0.7200

REALTYPE,  parameter                :: cc1CHCD01B  =  5.0000
REALTYPE,  parameter                :: cc2CHCD01B  =  0.6983
REALTYPE,  parameter                :: cc3CHCD01B  =  1.9664
REALTYPE,  parameter                :: cc4CHCD01B  =  1.0940
REALTYPE,  parameter                :: cc5CHCD01B  =  0.0000
REALTYPE,  parameter                :: cc6CHCD01B  =  0.4950
REALTYPE,  parameter                :: ct1CHCD01B  =  5.6000
REALTYPE,  parameter                :: ct2CHCD01B  =  0.6000
REALTYPE,  parameter                :: ct3CHCD01B  =  1.0000
REALTYPE,  parameter                :: ct4CHCD01B  =  0.0000
REALTYPE,  parameter                :: ct5CHCD01B  =  0.3333
REALTYPE,  parameter                :: cttCHCD01B  =  0.4770

REALTYPE,  parameter                :: cc1CCH02    =  5.0000
REALTYPE,  parameter                :: cc2CCH02    =  0.7983
REALTYPE,  parameter                :: cc3CCH02    =  1.9680
REALTYPE,  parameter                :: cc4CCH02    =  1.1360
REALTYPE,  parameter                :: cc5CCH02    =  0.0000
REALTYPE,  parameter                :: cc6CCH02    =  0.5000
REALTYPE,  parameter                :: ct1CCH02    =  5.5200
```

```
REALTYPE,  parameter                 :: ct2CCH02    =  0.2134
REALTYPE,  parameter                 :: ct3CCH02    =  0.3570
REALTYPE,  parameter                 :: ct4CCH02    =  0.0000
REALTYPE,  parameter                 :: ct5CCH02    =  0.3333
REALTYPE,  parameter                 :: cttCCH02    =  0.8200

integer, parameter                   :: LIST        = 0
integer, parameter                   :: GL78        = 1
integer, parameter                   :: MY82        = 2
integer, parameter                   :: KC94        = 3
integer, parameter                   :: LDOR96      = 4
integer, parameter                   :: CHCD01A     = 5
integer, parameter                   :: CHCD01B     = 6
integer, parameter                   :: CCH02       = 7
```

REVISION HISTORY:

```
Original author(s): Lars Umlauf
```

4.7.3 Generate a two-equation model

INTERFACE:

```
subroutine generate_model
```

DESCRIPTION:

Computes the parameters of an instance of the 'generic' two-equation model according to the specifications set in `gotmturb.nml`. This model solves (152) for the k and (168) for the generic length-scale defined in section 4.16 together with an Algebraic Stress Model. For several simple turbulent flows, analytical solutions of this models exist and can be used to calibrate the model coefficients. The method is described in great detail in *Umlauf and Burchard* (2003). Also users that are not interested in the generic part of GOTM should have a look in this section, because results derived here are referenced in later parts of the manual.
After the call to `generate_model()`, all parameters of the generic two-equation model are known. The user has full control over specific properties of the resulting model (see section 4.16).
In the following sections, the effects of model parameters on the behaviour of two-equation models in specific situations are briefly reviewed. For a more in-depth discussion, see *Umlauf and Burchard* (2003).

The logarithmic boundary layer

In the logarithmic boundary layer one has $P = \epsilon$ and $k \propto u_*^2$ by defintion. Under these conditions it is easy to show that a solution of (152) is

$$k = \frac{u_*^2}{(c_\mu^0)^2} , \tag{103}$$

and a solution of (168) can only be obtained if the condition

$$\sigma_\psi = \frac{n^2\kappa^2}{(c_\mu^0)^2(c_{\psi 2} - c_{\psi 1})} \tag{104}$$

is satisfied. (103) can be conveniently used to obtain boundary conditions for k, whereas (104) yields for example the value for the turbulent Schmidt-number σ_ψ as a function of the von Kármán constant (provided, of course, that the other constants are known). The value of the von Kármán constant is usually assumed to be $\kappa \approx 0.4$.

Decay of homogeneous turbulence

Another example of a simple but fundamental turbulent situation is the temporal decay of isotropic, homogeneous turbulence (approximated by the spatial decay of turbulence behind grids in laboratory settings). At large times, t, data from many experiments are well described by a power law of the form

$$\frac{k}{k_0} = A\left(\frac{t}{\tau_0}\right)^d , \tag{105}$$

with constant A and initial values of the kinetic energy, k_0, and the *eddy turnover time*, τ_0. The decay rates, d, have been thoroughly documented. Experiments (*Bradshaw* (1975), *Townsend* (1976), *Domaradzki and Mellor* (1984), *Mohamed and Larue* (1990)) suggest that d is in the range $-1.3 < d < -1$. DNS, generally conducted at low Reynolds numbers, produce consistently higher values. For example, *Briggs et al.* (1996) obtain a value near -1.5 from their DNS.
In homogeneous decaying turbulence, (152) and (168) reduce to a balance between the rate and dissipation terms, respectively. The coupled system of ordinary differential equations can be solved for given initial values k_0 and ψ_0 (see e.g. *Wilcox* (1998)). The solution can be shown to reduce to (105) at large times. Then, the decay exponent, d, is determined by

$$d = -\frac{2n}{2m + n - 2c_{\psi 2}} , \tag{106}$$

and thus depends only on the exponents m and n defined in (167) and the model constant $c_{\psi 2}$. For given exponents m and n, the experimental values of d can be used to derive the value of the model constant $c_{\psi 2}$. Note, that the predicted decay rate, d, is completely independent of the ASM (or the stability functions in other words).

Homogeneous turbulent shear-flows

A natural extension of decaying homogeneous turbulence is the inclusion of a homogeneous shear and an aligned homogeneous stratification. Since turbulence is still assumed to be homogeneous, the divergence of any turbulent transport term vanishes and the interplay between the stabilizing effects of stratification and the destabilizing action of shear can be isolated. Thus, it is not surprising that this interesting special case of turbulence has been explored extensively by laboratory experiments (*Tavoularis and Corrsin* (1981a,b), *Tavoularis and Karnik* (1989), *Rohr et al.* (1988)), by Direct Numerical Simulation (*Gerz et al.* (1989), *Holt et al.* (1991), *Jacobitz et al.* (1997), *Shih* *et al.* (2000)) and by Large-Eddy Simulation (*Kaltenbach et al.* (1994)). That flows of this kind are crucial also in many oceanographic flows has been pointed out by *Baumert and Peters* (2000). In the context of the generic two-equation model, this turbulent flow is mathematically established by neglecting the turbulent transport terms and the advective part of the material time derivative. Then, (152) and (168) reduce to a set of ordinary differential equations. Using the chain rule of differentiation, the relation

$$\frac{1}{l}\frac{\mathrm{d}l}{\mathrm{d}t} = \frac{1}{n}\frac{1}{\psi}\frac{\mathrm{d}\psi}{\mathrm{d}t} - \frac{m}{n}\frac{1}{k}\frac{\mathrm{d}k}{\mathrm{d}t} \tag{107}$$

for the mixing length, l, follows immediately from (167). With (107), the generic model expressed by (152) and (168) can be used to derive an evolution equation for the integral length scale, l,

$$\begin{aligned}\frac{1}{l}\frac{\mathrm{d}l}{\mathrm{d}t} &= -\left(\frac{1}{n}c_{\psi_2} - \frac{m}{n}\right)\frac{\epsilon}{k} \\ &+ \frac{1}{k}\left(\left(\frac{1}{n}c_{\psi_1} - \frac{m}{n}\right)P + \left(\frac{1}{n}c_{\psi_3} - \frac{m}{n}\right)G\right) \quad .\end{aligned} \tag{108}$$

Tennekes (1989) derived an equation similar to (108), however only for the special case of the k-ϵ model applied to unstratified flows, and stated that *'on dimensional grounds, l cannot depend upon the shear because the shear is homogeneous and cannot impose a length scale'*. This argument requires immediately

$$c_{\psi 1} = m \ , \tag{109}$$

which is used in the subroutine to determine the model parameter $c_{\psi 1}$. A more detailed discussion of this method is given in *Umlauf and Burchard* (2003).

Shear-free turbulence, wave-breaking

The first step in understanding the behaviour of two-equation models in the surface layer affected by breaking gravity waves is the investigation of a special case, in which turbulence decays spatially away from a planar source *without mean shear.* Turbulence generated by an oscillating grid in a water tank has been used in various laboratory settings to study the spatial decay of velocity fluctuations in this basic turbulent flow, where turbulent transport and dissipation balance exactly. For a summary of these results, see *Umlauf et al.* (2003).
All grid stirring experiments confirmed a power law for the decay of k and a linear increase of the length scale, l, according to

$$k = K(z + z_0)^{\alpha} \ , \quad l = L(z + z_0) \ , \tag{110}$$

where K, L, and z_0 are constants, and the source of turbulence has been assumed to be at $z = 0$. In these experiments, $z_0 = l/L$ at $z = 0$ is not related to any kind of surface roughness length. Rather, it is connected to the length scale of injected turbulence, uniquely determined by the spectral properties of turbulence at the source. Experiments suggest that the decay rate for the turbulent kinetic energy is likely to be in the range $-3 < \alpha < -2$. The value of L, i.e. the slope of the turbulent length scale, l, was found to be consistently smaller than in wall-bounded shear flows, $L < \kappa \approx 0.4$, see *Umlauf et al.* (2003).
In stationary, shear-free, unstratified turbulence, the generic model simplifies to a balance between the turbulent transport terms and the dissipative terms in (152) and (168). Using the definition

of ψ in (167) and the scaling for the rate of dissipation, (155), the transport and dissipation of k and ψ are balanced according to

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}z}\left(\frac{c_\mu}{\sigma_k^\psi}k^{\frac{1}{2}}l\frac{\mathrm{d}k}{\mathrm{d}z}\right) &= (c_\mu^0)^3\frac{k^{\frac{3}{2}}}{l}\ ,\\ \frac{\mathrm{d}}{\mathrm{d}z}\left(\frac{c_\mu}{\sigma_\psi}k^{\frac{1}{2}}l\frac{\mathrm{d}}{\mathrm{d}z}\left((c_\mu^0)^p k^m l^n\right)\right) &= c_{\psi 2}(c_\mu^0)^{p+3}k^{m+\frac{1}{2}}l^{n-1}\quad .\end{aligned} \tag{111}$$

Note, that in shear-free turbulence, the shear-number defined in (47) is $\alpha_M = 0$ by definition, and stability functions always reduce to a constant which is, however, different from the constant c_μ^0 approached in the logarithmic boundary layer, see section 4.7.13.
For the solution of this non-linear system , we inserted the expressions (110) in (111). From (155) and (46), power-laws follow then also for $\epsilon = E(z+z_0)^\beta$ and $\nu_t = N(z+z_0)^\gamma$.
Inserting (110) into $(111)_1$ yields the equation

$$(\alpha L)^2 = \frac{2}{3}(c_\mu^0)^2 R\,\sigma_k^\psi\ , \tag{112}$$

where the constant ratio $R = c_\mu^0/c_\mu$ follows uniquely from the respective ASM. The power-law (110) can also be inserted in $(111)_2$ to yield

$$(\alpha m + n)\left(\left(\frac{1}{2}+m\right)\alpha + n\right)L^2 = \left(c_\mu^0\right)^2 R\,\sigma_\psi c_{\psi 2}\quad . \tag{113}$$

We note that with the help of (106) and (109), the relation (104) can be rewritten as

$$\sigma_\psi = \frac{2\kappa^2 d}{(c_\mu^0)^2(d+2)}n\quad . \tag{114}$$

Expressing now σ_ψ with (114) and $c_{\psi 2}$ with the help of (106) on the right hand side of (113), an equation expressing the exponent m in terms of n (or vice-versa) can be obtained. The result for n can be written as

$$\begin{aligned} n &= -\frac{1}{4(2+d)(\kappa^2R - L^2)}\Bigg(4d\kappa^2 R\,m - (1+4m)(2+d)\alpha L^2\\ &+\sqrt{8m(1+2m)(2+d)^2(\kappa^2R-L^2)\alpha^2L^2 + \Big(-4d\kappa^2R\ m + (2+d)(1+4m)\alpha L^2\Big)^2}\Bigg)\quad .\end{aligned} \tag{115}$$

After assigning appropriate values for the von Kármán constant, κ, the decay coefficient of homogeneous turbulence, d, the spatial decay rate, α, and the slope, L, an infinite number of pairs of m and n satisfying (115) can be derived. Each corresponds to a different two-equation model. Some example are given in table 5 (see *Umlauf and Burchard* (2003)).
Even though each line in this table represents a different two-equation model with completely different model constants, each of the two groups of models (with $\alpha = -2.0$ and $\alpha = -2.5$, respectively) *performs completely identical in all situations discussed until here.* Thus, the generic model allows for the formulation of groups of two-equation models with fully controlled properties from the outset. As discussed by *Umlauf and Burchard* (2003), one more constraint is necessary to obtain the final values of all parameters, including the exponents m and n. These authors

α	L	m	n	$c_{\psi 2}$	σ_k^ψ	σ_ψ
-2.0	0.20	1.00	-0.67	1.22	0.80	1.07
-2.0	0.20	2.00	-1.09	2.36	0.80	1.75
-2.5	0.20	1.00	-1.05	1.35	1.25	1.68
-2.5	0.20	2.00	-1.74	2.58	1.25	2.78

Table 5: Some parameter sets for the generic model with $\kappa = 0.4$, $d = -1.2$, $(c_\mu^0)^2 = 0.3$, $c_{\psi_1} = m$ and obeying the log-layer compatibility relation (114).

suggested that the first line in table 5 yields a model with excellent properties in all flows they considered.

Mixed layer deepending

The correct prediction of mixed layer deepening into a stratified fluid due to a wind stress at the surface is one of the most crucial requirements for an oceanic turbulence model. This situation has been frequently interpreted by analogy with the classical experiment of *Kato and Phillips* (1969) and its re-interpretation by *Price* (1979), in which the entrainment in a linearly stratified fluid subject to a constant surface stress was investigated. The results of this experiment have been used by numerous authors to calibrate their turbulence models.
In particular, it has been shown by *Burchard and Bolding* (2001) for the k-ϵ model of *Rodi* (1987), by *Burchard* (2001b) for the q^2l model of *Mellor and Yamada* (1982), and by *Umlauf et al.* (2003) for the k-ω model of *Wilcox* (1988) that, remarkably, the mixed layer depth predicted by these models depends almost exclusively on the value of the Richardson number, $Ri = N^2/M^2$, computed in a *homogeneous*, stratified shear-flow in steady-state. This value is usually referred to as the steady-state Richardson number, Ri_{st} (*Rohr et al.* (1988), *Kaltenbach et al.* (1994), *Jacobitz et al.* (1997), *Shih et al.* (2000)).
Umlauf et al. (2003) showed that in the context of models considered in GOTM, the steady-state Richardson number is determined by the relation

$$Ri_{st} = \frac{c_\mu}{c_\mu{}'} \frac{c_{\psi 2} - c_{\psi 1}}{c_{\psi 2} - c_{\psi 3}} \quad . \tag{116}$$

Since it is well-known that, with the equilibrium assumption $P + G = \epsilon$, stability functions reduce to functions of Ri only (*Mellor and Yamada* (1974), *Galperin et al.* (1988)), (116) is a non-linear equation for the model constant $c_{\psi 3}$ for given Ri_{st}. Note, that the structure parameters, m and n, do not appear in (116). This implies that the type of the two-equation model is irrelevant for the prediction of the mixed layer depth, as long as (116) is fulfilled for identical Ri_{st}. Numerical examples with very different values of m and n confirmed indeed that the mixed layer depth only depends on Ri_{st}. The experiment of *Kato and Phillips* (1969) could almost perfectly be reproduced, provided the parameter $c_{\psi 3}$ was chosen to correspond to $Ri_{st} \approx 0.25$, see *Umlauf et al.* (2003).
Note, that in instable situations, a different value of the parameter $c_{\psi 3}$ needs to be used. This does not cause a discontinuity in the model because the buoyancy term in (168) is zero at the transition. An evaluation of the length-scale equations in convective flows, however, is intimately related to the third-order modelling of the triple correlation terms, a topic outside the scope of

this documentation.

USES:

```
IMPLICIT NONE
```

REVISION HISTORY:

```
Original author(s): Lars Umlauf
```

4.7.4 Analyse the turbulence models

INTERFACE:

```
subroutine analyse_model
```

DESCRIPTION:

This routine analyses all models in GOTM for their physical properties implied by chosen model parameters. These results can be displayed by calling the internal routine `report_model()`, also defined in the `turbulence` module (see section 4.7.5).
In most cases, the relations connecting model parameters and physical properties have already been derived in section 4.7.3: the von Kármán constant, κ, follows from (104), the decay rate in homogeneous turbulence , d, from (106), and the steady-state Richardson-number from (116). These relations have been obtained in 'generic' form (see section 4.16), but relations for specific models, like the k-ϵ model or the k-ω model, can be derived by simply adopting the parameters compiled in table 8 and table 9 in section 4.16.
The decay rates α and L in shear-free turbulence follow from the physically meaningful roots of (112) and (113), which are

$$\begin{aligned} \alpha &= -\frac{4n(\sigma_k^\psi)^{\frac{1}{2}}}{(1+4m)(\sigma_k^\psi)^{\frac{1}{2}} - (\sigma_k^\psi + 24\sigma_\psi c_{\psi 2})^{\frac{1}{2}}} \ , \\ L &= c_\mu^0 R^{\frac{1}{2}} \left(\frac{(1+4m+8m^2)\sigma_k^\psi + 12\sigma_\psi c_{\psi 2} - (1+4m)(\sigma_k^\psi(\sigma_k^\psi + 24\sigma_\psi c_{\psi 2}))^{\frac{1}{2}}}{12n^2} \right)^{\frac{1}{2}} \ , \end{aligned} \tag{117}$$

where it should be recalled that $R = c_\mu^0/c_\mu$. For the standard models (without ASM), $R = 1$ may be assumed. Then, with the values from table 8 and table 9, solutions for the k-ϵ model of *Rodi* (1987), and the k-ω model of *Umlauf et al.* (2003) can be directly recovered as special cases of this equation.
Due to its wall-functions, the model of *Mellor and Yamada* (1982) described in section 4.14 requires a slightly more complicated analysis. For this model, the von Kármán constant is computed according to

$$\kappa = \sqrt{\frac{E_2 - E_1 + 1}{S_l B_1}} \quad . \tag{118}$$

The decay rates in shear-free turbulence can be shown to be

$$
\begin{aligned}
\alpha &= \frac{5\kappa B_1^{\frac{1}{2}} S_l + \left(12E_2\left(2S_l - S_q\right) + B_1\kappa^2 S_l\left(S_l + 12S_q\right)\right)^{\frac{1}{2}}}{3\kappa B_1^{\frac{1}{2}}\left(S_q - 2S_l\right)} \\
L &= \kappa\left(\frac{\mathcal{N}}{6S_q(E_2 - B_1\kappa^2 S_l)^2}\right)^{\frac{1}{2}} ,
\end{aligned}
\tag{119}
$$

where we introduced the abbreviation

$$
\begin{aligned}
\mathcal{N} &= 6E_2\left(2S_l - S_q\right) + B_1\kappa^2 S_l\left(13S_l + 6S_q\right) \\
&- 5B_1^{\frac{1}{2}}\kappa S_l\left(12E_2(2S_l - S_q) + B_1\kappa^2 S_l(S_l + 12S_q)\right)^{\frac{1}{2}} \quad .
\end{aligned}
\tag{120}
$$

These equations replace (117) for the model of *Mellor and Yamada* (1982). Decay-rates for this model do not at all depend on the stability functions. However, they depend on the parameter E_2 of the wall-functions. This parameter, however, has been derived for wall-bounded shear flows, and it is not very plausible to find it in an expression for *shear-free* flows.
The routine `analyse_model()` works also for one-equation models, where the length-scale, l, is prescribed by an analytical expression (see section 4.19). However, some attention has to be paid in interpreting the results. First, it is clear that these models cannot predict homogeneous turbulence, simply because all formulations rely on some type of modified boundary layer expressions for the length-scale. This impies that a well-defined decay rate, d, and a steady-state Richardson-number, Ri_{st}, cannot be computed. Second, the von Kármán constant, κ, does not follow from (104) or (118), because κ now relates directly to the prescribed slope of the length-scale close to the bottom or the surface. Third, in shear-free flows, $(117)_1$ or $(119)_1$ remain valid, provided the planar source of the spatially decaying turbulence is located at $z = 0$. Then, the slope of the length-scale, L, defined in (110) can be identified with the prescribed slope, κ, and $(117)_1$ or $(119)_1$ are identical to the solutions suggested by *Craig and Banner* (1994).
In this context, it should be pointed out that the shear-free solutions also have a direct relation to an important oceanic situation. If the planar source of turbulence is assumed to be located at $z = 0$, and if the injected turbulence is identified with turbulence caused by breaking surface-waves, then it can be shown that (117) or (119) are valid in a thin boundary layer adjacent to the suface. Further below, to classical law of the wall determines the flow, see *Craig and Banner* (1994) and citeUmlaufetal2003.

USES:

```
IMPLICIT NONE
```

INPUT PARAMETERS:

REVISION HISTORY:

```
Original author(s): Lars Umlauf
```

4.7.5 Report turbulence model

INTERFACE:

```
   subroutine report_model
```

DESCRIPTION:

This routine reports on the parameters and the propeties of all turbulence models implemented in GOTM. Results are written to the screen.

USES:

```
   IMPLICIT NONE
```

REVISION HISTORY:

```
   Original author(s): Lars Umlauf
```

4.7.6 Manage turbulence time-stepping

INTERFACE:

```
   subroutine do_turbulence(nlev,dt,depth,u_taus,u_taub,z0s,z0b,h,      &
                            NN,SS,xP)
```

DESCRIPTION:

This routine is the central point of the turbulence scheme. It determines the order, in which turbulence variables are updated, and calls other member functions updating the TKE, the length-scale, the dissipation rate, the ASM etc. Note, that the list of arguments in `do_turbulence()` corresponds exactly to those mean flow and grid-related variables required to update the turbulent quantities. These variables have to be passed from a 3-D model, if the `turbulence` module of GOTM is used for the computation of the turbulent fluxes. Do not forget to call `init_turbulence()` from the 3-D model before the first call to `do_turbulence()`.
The variable `turb_method` determines the essential structure of the calls in `do_turbulence()`. At the moment, the following model types are available:

- `turb_method = 0` corresponds to the "convective adjustment" algorithm, see section 3.15. Since this model is not a real one-point turbulence closure, it is not called from `do_turbulence` but directly from the main GOTM loop.
- `turb_method = 1` corresponds to a purely algebraic description of the turbulent diffusivities.
- `turb_method = 2` corresponds to models computing the diffusivities from the TKE and the turbulent length scale according to (46). TKE and length scale are computed from dynamic PDEs or algebraic relations, an empirical (i.e. not derived from a second-order model) stability function is used, see section 4.7.12.
- `turb_method = 3` corresponds to a second-order model for the turbulent fluxes.

The second-order models fall into different categories, depending on the value of `second_method`. These models, discussed in detail in section 4.4, are listed in the following.

- `second_method = 1` corresponds to algebraic quasi-equilibrium models with scaling in the spirit of *Galperin et al.* (1988), see section 4.27.
- `second_method = 2` corresponds to algebraic models assuming $P_b = \epsilon_b$, and hence using (74). Furthermore, full equilibrium $P + G = \epsilon$ and $P_b = \epsilon_b$ is assumed for the computation of $\mathcal{N}$ and $\mathcal{N}_b$ in (67), see section 4.26
- `second_method = 3` corresponds to algebraic models assuming full equilibrium $P + G = \epsilon$ and $P_b = \epsilon_b$ for the computation of $\mathcal{N}$ and $\mathcal{N}_b$ in (67). Now, however, also an equation for (half) the buoyancy variance k_b is solved, leading to the appearance of the counter-gradient term in (76), see section 4.25. This model is not yet fully tested and therefore not available.

Depending on the values of `kb_method` and `epsb_method`, different algebraic or differential equations for k_b and ϵ_b are solved for `second_method = 3,4`.

USES:

```
IMPLICIT NONE

interface
   subroutine production(nlev,NN,SS,xP)
     integer,  intent(in)                :: nlev
     REALTYPE, intent(in)                :: NN(0:nlev)
     REALTYPE, intent(in)                :: SS(0:nlev)
     REALTYPE, intent(in), optional      :: xP(0:nlev)
   end subroutine production
end interface
```

INPUT PARAMETERS:

```
number of vertical layers
integer,  intent(in)                :: nlev

time step (s)
REALTYPE, intent(in)                :: dt

distance between surface
and bottom(m)
REALTYPE, intent(in)                :: depth

surface and bottom
friction velocity (m/s)
REALTYPE, intent(in)                :: u_taus,u_taub

surface and bottom
roughness length (m)
REALTYPE, intent(in)                :: z0s,z0b
```

```
layer thickness (m)
REALTYPE, intent(in)                :: h(0:nlev)

boyancy frequency squared (1/s^2)
REALTYPE, intent(in)                :: NN(0:nlev)

shear-frequency squared (1/s^2)
REALTYPE, intent(in)                :: SS(0:nlev)

TKE production due to seagrass
friction (m^2/s^3)
REALTYPE, intent(in), optional      :: xP(0:nlev)
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding, Hans Burchard,
                    Lars Umlauf
```

4.7.7 Update the turbulent kinetic energy

INTERFACE:

```
subroutine do_tke(nlev,dt,u_taus,u_taub,z0s,z0b,h,NN,SS)
```

DESCRIPTION:

Based on user input, this routine calls the appropriate routines for calculating the turbulent kinetic energy. The user has the choice between an algebraic equation described in section 4.17, and two versions of the dynamic transport equation of the TKE described in section 4.11 and section 4.12. The former uses k-ϵ notation, the latter the notation of *Mellor and Yamada* (1982). Apart from this, both equations are identical and update the vectors `tke` and `tkeo`, which is the value of the tke at the old time step.

USES:

```
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer,  intent(in)                :: nlev
REALTYPE, intent(in)                :: dt,u_taus,u_taub,z0s,z0b
REALTYPE, intent(in)                :: h(0:nlev)
REALTYPE, intent(in)                :: NN(0:nlev),SS(0:nlev)
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding, Hans Burchard,
                    Manuel Ruiz Villarreal, Lars Umlauf
```

4.7.8 Update the buoyancy variance

INTERFACE:

```
   subroutine do_kb(nlev,dt,u_taus,u_taub,z0s,z0b,h,NN,SS)
```

DESCRIPTION:

Based on the value of `kb_method`, this routine calls the appropriate routines for calculating (half) the buoyancy variance k_b defined in (52). The user has the choice between a simple algebraic expression, described in section 4.18, and a dynamic equation for k_b, described in section 4.13.

USES:

```
   IMPLICIT NONE
```

INPUT PARAMETERS:

```
   integer,  intent(in)                :: nlev
   REALTYPE, intent(in)                :: dt,u_taus,u_taub,z0s,z0b
   REALTYPE, intent(in)                :: h(0:nlev)
   REALTYPE, intent(in)                :: NN(0:nlev),SS(0:nlev)
```

REVISION HISTORY:

```
   Original author(s): Lars Umlauf
```

4.7.9 Update the dissipation length-scale

INTERFACE:

```
   subroutine do_lengthscale(nlev,dt,depth,u_taus,u_taub, z0s,z0b,h,NN,SS)
```

DESCRIPTION:

Based on the value of `len_scale_method`, this routine calls the appropriate routines for calculating the turbulent length-scale, l, and the rate of dissipation, ϵ. The user has the choice between several algebraic equations described in section 4.19, and several differential transport equations for a length-scale determining variable. At the moment, GOTM implements equations for the rate of dissipation, described in section 4.15, for the Mellor-Yamada model described in section 4.14, and for the generic scale formulated by *Umlauf and Burchard* (2003) and described in section 4.16. This last transport equation generalises all of the previously mentioned models. For example, the k-ϵ model and the k-ω model can be recovered as special cases of the generic equation, see *Umlauf and Burchard* (2003).

USES:

```
   IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer,  intent(in)               :: nlev
REALTYPE, intent(in)               :: dt,depth,u_taus,u_taub,z0s,z0b
REALTYPE, intent(in)               :: h(0:nlev)
REALTYPE, intent(in)               :: NN(0:nlev),SS(0:nlev)
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding, Hans Burchard,
                    Manuel Ruiz Villarreal,
                    Lars Umlauf
```

4.7.10 Update the desctruction rate of buoyancy variance

INTERFACE:

```
subroutine do_epsb(nlev,dt,u_taus,u_taub,z0s,z0b,h,NN,SS)
```

DESCRIPTION:

Based on the value of `epsb_method`, this routine calls the appropriate routines for calculating the molecular destruction rate of k_b, defined in (160). Presently, only a simple algebraic expression, described in section 4.20, is available in GOTM.

USES:

```
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer,  intent(in)               :: nlev
REALTYPE, intent(in)               :: dt,u_taus,u_taub,z0s,z0b
REALTYPE, intent(in)               :: h(0:nlev)
REALTYPE, intent(in)               :: NN(0:nlev),SS(0:nlev)
```

REVISION HISTORY:

```
Original author(s): Lars Umlauf
```

4.7.11 Update diffusivities (Kolmogorov-Prandtl relation)

INTERFACE:

```
subroutine kolpran(nlev)
```

DESCRIPTION:

Eddy viscosity and diffusivity are calculated by means of the relation of Kolmogorov and Prandtl from the updated values of k, l and the stability functions according to (46). In addition, the counter-gradient term $\tilde{\Gamma}_B = \epsilon\Gamma$ is updated, see (35) and (77).

Note, that this routine relies on the fact that the lowest and uppermost values of the stability functions and of k, l, and Γ have been computed using the correct boundary conditions. No special treatment of ν_t, ν_t^B, and $\tilde{\Gamma}_B$ at the boundaries is processed.

USES:

```
   IMPLICIT NONE
```

INPUT PARAMETERS:

```
   integer, intent(in)       :: nlev
```

REVISION HISTORY:

```
   Original author(s): Karsten Bolding, Hans Burchard,
                       Manuel Ruiz Villarreal, Lars Umlauf
```

4.7.12 Update stability functions

INTERFACE:

```
   subroutine stabilityfunctions(nlev)
```

DESCRIPTION:

Based on the user's specifications in `gotmtub.nml`, this internal routine selects the desired stability functions defined in (46). These simple functions depend on α_M and α_N defined in (47), which are in most cases only used to compute the Richardson-number

$$Ri = \frac{\alpha_N}{\alpha_M} \quad . \tag{121}$$

A description of individual stability functions starts from section 4.28.

USES:

```
   IMPLICIT NONE
```

INPUT PARAMETERS:

```
   integer, intent(in)                 :: nlev
```

REVISION HISTORY:

```
   Original author(s): Hans Burchard, Karsten Bollding, Lars Umlauf
```

4.7.13 Compute special values of stability functions

INTERFACE:

```
   subroutine compute_cm0(turb_method,stab_method,scnd_method)
```

DESCRIPTION:

Computes the values of the stability function c_μ defined in (46) in the logarithmic boundary-layer, c_μ^0, and in shear-free, spatially decaying turbulence, $c_\mu^{\rm sf}$ (see section 4.7.4).
c_μ^0 is the value of c_μ in unstratified equilibrium flows, i.e. in the logarithmic wall region. It can be obtained from the relation $P = \epsilon$, according to (80) written in the form

$$\frac{P}{\epsilon} = \hat{c}_\mu \alpha_M = 1 \quad . \tag{122}$$

In unstratified flows, $\hat{c}_\mu$ only depends on α_M (see sections 4.24–4.26), and (122) is a polynomial equation for the value of α_M in equilibrium. Its solution is

$$\alpha_M = \frac{3\mathcal{N}^2}{a_2^2 - 3a_3^2 + 3a_1\mathcal{N}} \ , \tag{123}$$

where, according to (67) in equilibrium $\mathcal{N} = (c_1 + c_1^*)/2$. The value of the stability function in equilibrium follows directly from (122),

$$\hat{c}_\mu^0 = \frac{a_2^2 - 3a_3^2 + 3a_1\mathcal{N}}{3\mathcal{N}^2} \quad . \tag{124}$$

Note that $\hat{c}_\mu^0 = (c_\mu^0)^4$ according to (78).
Algebraic Stress Models exhibit an interesting behaviour in unstratified, shear-free turbulence. Clearly, in the absence of shear, these models predict isotropic turbulence, $b_{ij} = 0$, according to (61). This is a direct consequence of the assumption (60), implying an infinitely small return-to-isotropy time scale. Formally, however, the limit of the stability function $\hat{c}_\mu$ for $\alpha_M \to 0$ follows from (76) and the definitions given in sections 4.24–4.26. The limiting value is

$$\lim_{\alpha_M \to 0} \hat{c}_\mu = \hat{c}_\mu^{\rm sf} = \frac{a_1}{\mathcal{N}} \ , \tag{125}$$

where, according to (67), one has either $\mathcal{N} = c_1/2 - 1$ or $\mathcal{N} = (c_1 + c_1^*)/2$, see section 4.24 and section 4.26, respectively. The above limit corresponds to nearly isotropic turbulence supporting a very small momentum flux caused by a very small shear.
Note that $\hat{c}_\mu^{\rm sf} = (c_\mu^0)^3 c_\mu^{\rm sf}$ according to (78).

USES:

```
   IMPLICIT NONE
```

INPUT PARAMETERS:

```
   integer, intent(in)                 :: turb_method
   integer, intent(in)                 :: stab_method
   integer, intent(in)                 :: scnd_method
```

REVISION HISTORY:

```
   Original author(s): Lars Umlauf
```

4.7.14 Boundary conditons for the k-equation (k-epsilon style)

INTERFACE:

```
REALTYPE function k_bc(bc,type,zi,z0,u_tau)
```

DESCRIPTION:

Computes prescribed and flux boundary conditions for the transport equation (152). The formal parameter `bc` determines whether `Dirchlet` or `Neumann`-type boundary conditions are computed. Depending on the physical properties of the boundary-layer, the parameter `type` relates either to a `visous`, a `logarithmic`, or an `injection`-type boundary-layer. In the latter case, the flux of TKE caused by breaking surface waves has to be specified. Presently, there is only one possibility to do so implemented in GOTM. It is described in section 4.34. All parameters that determine the boundary layer have to be set in `gotmturb.nml`.
Note that in this section, for brevity, z denotes the distance from the wall (or the surface), and *not* the standard coordinate of the same name used in GOTM.

Viscous boundary-layers

This type is not implemented yet in GOTM.

Logarithmic boundary-layers

The Dirichlet (prescribed) boundary condition follows from (103) as

$$k = \frac{u_*^2}{(c_\mu^0)^2} \quad . \tag{126}$$

The Neumann (flux) boundary condition can be derived from the constancy of k in the logarithmic region. This fact can be written as

$$F_k = -\frac{\nu_t}{\sigma_k}\frac{\partial k}{\partial z} = 0 \quad . \tag{127}$$

Shear-free boundary-layers with injection of TKE

The Dirichlet (prescribed) boundary condition follows simply from the power-law in (110),

$$k = K(z + z_0)^\alpha \quad . \tag{128}$$

The Neumann (flux) boundary condition can be written as

$$F_k = -\frac{\nu_t}{\sigma_k}\frac{\partial k}{\partial z} = -\frac{c_\mu}{\sigma_k} K^{\frac{3}{2}} L\alpha (z + z_0)^{\frac{3}{2}\alpha} \ , \tag{129}$$

which follows immediately from (110) and the expression for the turbulent diffusivity, (46). The parameter K can be determined from an evaluation of (129) at $z = 0$. The result is

$$K = \left(-\frac{\sigma_k}{c_\mu \alpha L} F_k \right)^{\frac{2}{3}} \frac{1}{z_0^\alpha} \ , \tag{130}$$

where the specification of the flux F_k and the value of z_0 have to be determined from a suitable model of the wave breaking process.

USES:

```
     IMPLICIT NONE
```

INPUT PARAMETERS:

```
   integer, intent(in)                  :: bc,type
   REALTYPE, intent(in)                 :: zi,z0,u_tau
```

REVISION HISTORY:

```
   Original author(s): Lars Umlauf
```

4.7.15 Boundary conditons for the k-equation (Mellor-Yamada style)

INTERFACE:

```
   REALTYPE function  q2over2_bc(bc,type,zi,z0,u_tau)
```

DESCRIPTION:

Computes prescribed and flux boundary conditions for the transport equation (156). The formal parameter `bc` determines whether `Dirchlet` or `Neumann`-type boundary conditions are computed. Depending on the physical properties of the boundary-layer, the parameter `type` relates either to a `visous`, a `logarithmic`, or an `injection`-type boundary-layer. In the latter case, the flux of TKE caused by breaking surface waves has to be specified. Presently, there is only one possibility to do so implemented in GOTM. It is described in section 4.34. All parameters that determine the boundary layer have to be set in `gotmturb.nml`.
Note that in this section, for brevity, z denotes the distance from the wall (or the surface), and *not* the standard coordinate of the same name used in GOTM.

Viscous boundary-layers

This type is not implemented yet in GOTM.

Logarithmic boundary-layers

The Dirichlet (prescribed) boundary condition follows from (103) and (159) as

$$q^2/2 = \frac{u_*^2 B_1^{\frac{2}{3}}}{2} \quad . \tag{131}$$

The Neumann (flux) boundary condition can be derived from the constancy of $q^2/2$ in the logarithmic region. This fact can be written as

$$F_q = -S_q q l \frac{\partial k}{\partial z} = 0 \quad . \tag{132}$$

Shear-free boundary-layers with injection of TKE

The Dirichlet (prescribed) boundary condition follows simply from the power-law in (110),

$$\frac{q^2}{2} = k = K(z + z_0)^\alpha \quad . \tag{133}$$

The Neumann (flux) boundary condition can be written as

$$F_q = -S_q q l \frac{\partial k}{\partial z} = -\sqrt{2} S_q K^{\frac{3}{2}} \alpha L (z + z_0)^{\frac{3}{2}\alpha} \ , \tag{134}$$

which follows immediately from (110). The parameter K can be determined from an evaluation of (134) at $z = 0$. The result is

$$K = \left(- \frac{F_q}{\sqrt{2} S_q \alpha L} \right)^{\frac{2}{3}} \frac{1}{z_0^\alpha} \ , \tag{135}$$

where the specification of the flux F_q and the value of z_0 have to be determined from a suitable model of the wave breaking process.

USES:

```
    IMPLICIT NONE
```

INPUT PARAMETERS:

```
  integer, intent(in)                     :: bc,type
  REALTYPE, intent(in)                    :: zi,z0,u_tau
```

REVISION HISTORY:

```
   Original author(s): Lars Umlauf
```

4.7.16 Boundary conditons for the epsilon-equation

INTERFACE:

```
   REALTYPE function  epsilon_bc(bc,type,zi,ki,z0,u_tau)
```

DESCRIPTION:

Computes prescribed and flux boundary conditions for the transport equation (165). The formal parameter `bc` determines whether `Dirchlet` or `Neumann`-type boundary conditions are computed. Depending on the physical properties of the boundary-layer, the parameter `type` relates either to a `visous`, a `logarithmic`, or an `injection`-type boundary-layer. In the latter case, the flux of TKE caused by breaking surface waves has to be specified. Presently, there is only one possibility to do so implemented in GOTM. It is described in section 4.34. All parameters that determine the boundary layer have to be set in `gotmturb.nml`.
Note that in this section, for brevity, z denotes the distance from the wall (or the surface), and *not* the standard coordinate of the same name used in GOTM.

Viscous boundary-layers

This type is not implemented yet in GOTM.

Logarithmic boundary-layers

The Dirichlet (prescribed) boundary condition follows from (155) as

$$\epsilon = \frac{(c_\mu^0)^3 k^{\frac{3}{2}}}{\kappa(z+z_0)} \ , \tag{136}$$

where we used the law-of-the-wall relation $l = \kappa(z + z_0)$.
The Neumann (flux) boundary condition can be expressed as

$$F_\epsilon = -\frac{\nu_t}{\sigma_\epsilon}\frac{\partial \epsilon}{\partial z} = \frac{(c_\mu^0)^4}{\sigma_\epsilon}\frac{k^2}{z+z_0} \ , \tag{137}$$

by inserting $l = \kappa(z + z_0)$ into the expression for the diffusivity in (46). Note, that in (136) and (137), we use `ki`, the value of k at the current time step, to compute the boundary conditions. By means of (103), it would have been also possible to express the boundary conditions in terms of the friction velocity, u_*. This, however, causes numerical difficulties in case of a stress-free surface boundary-layer as for example in the pressure-driven open channel flow.

Shear-free boundary-layers with injection of TKE

The Dirichlet (prescribed) boundary condition follows simply from the power-law (110) inserted in (155). This yields

$$\epsilon = (c_\mu^0)^3 K^{\frac{3}{2}} L^{-1} (z+z_0)^{\frac{3}{2}\alpha - 1} \quad . \tag{138}$$

The Neumann (flux) boundary condition is

$$F_\epsilon = -\frac{\nu_t}{\sigma_\epsilon}\frac{\partial \epsilon}{\partial z} = -\frac{c_\mu (c_\mu^0)^3}{\sigma_\epsilon} K^2 \left(\frac{3}{2}\alpha - 1\right)(z+z_0)^{2\alpha-1} \ , \tag{139}$$

which follows from (110) and (46). The parameter K is computed as described in the context of (130).

USES:

```
   IMPLICIT NONE
```

INPUT PARAMETERS:

```
   integer, intent(in)                  :: bc,type
   REALTYPE, intent(in)                 :: zi,ki,z0,u_tau
```

REVISION HISTORY:

```
   Original author(s): Lars Umlauf
```

4.7.17 Boundary conditons for the psi-equation

INTERFACE:

```
   REALTYPE function  psi_bc(bc,type,zi,ki,z0,u_tau)
```

DESCRIPTION:

Computes prescribed and flux boundary conditions for the transport equation (168). The formal parameter `bc` determines whether `Dirchlet` or `Neumann`-type boundary conditions are computed. Depending on the physical properties of the boundary-layer, the parameter `type` relates either to a `visous`, a `logarithmic`, or an `injection`-type boundary-layer. In the latter case, the flux of TKE caused by breaking surface waves has to be specified. Presently, there is only one possibility to do so implemented in GOTM. It is described in section 4.34. All parameters that determine the boundary layer have to be set in `gotmturb.nml`.
Note that in this section, for brevity, z denotes the distance from the wall (or the surface), and *not* the standard coordinate of the same name used in GOTM.

Viscous boundary-layers

This type is not implemented yet in GOTM.

Logarithmic boundary-layers

The Dirichlet (prescribed) boundary condition follows from (167) as

$$\psi = (c_\mu^0)^p \kappa^n k^m \, (z + z_0)^n \ , \tag{140}$$

where we used the law-of-the-wall relation $l = \kappa(z + z_0)$.
Neumann (flux) boundary condition can be written as

$$F_\psi = -\frac{\nu_t}{\sigma_\psi}\frac{\partial \psi}{\partial z} = -\frac{n(c_\mu^0)^{p+1}\kappa^{n+1}}{\sigma_\psi} k^{m+\frac{1}{2}} (z + z_0)^n \tag{141}$$

by inserting $l = \kappa(z + z_0)$ into the expression for the diffusivity in (46). Note, that in (140) and (141), we use `ki`, the value of k at the current time step, to compute the boundary conditions. By means of (103), it would have been also possible to express the boundary conditions in terms of the friction velocity, u_*. This, however, causes numerical difficulties in case of a stress-free surface boundary-layer as for example in the pressure-driven open channel flow.

Shear-free boundary-layers with injection of TKE

The Dirichlet (prescribed) boundary condition follows simply from the power-law (110) inserted in (167). This yields

$$\psi = (c_\mu^0)^p K^m L^n (z + z_0)^{m\alpha+n} \quad . \tag{142}$$

The Neumann (flux) boundary condition is

$$F_\psi = -\frac{\nu_t}{\sigma_\psi}\frac{\partial \psi}{\partial z} = -\frac{c_\mu (c_\mu^0)^p}{\sigma_\psi} \left(m\alpha + n\right) K^{m+\frac{1}{2}} L^{n+1} (z + z_0)^{(m+\frac{1}{2})\alpha+n} \ , \tag{143}$$

USES:

```
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer, intent(in)                :: bc,type
REALTYPE, intent(in)               :: zi,ki,z0,u_tau
```

REVISION HISTORY:

```
Original author(s): Lars Umlauf
```

4.7.18 Boundary conditons for the q2l-equation

INTERFACE:

```
REALTYPE function  q2l_bc(bc,type,zi,ki,z0,u_tau)
```

DESCRIPTION:

Computes prescribed and flux boundary conditions for the transport equation (162). The formal parameter `bc` determines whether `Dirchlet` or `Neumann`-type boundary conditions are computed. Depending on the physical properties of the boundary-layer, the parameter `type` relates either to a `visous`, a `logarithmic`, or an `injection`-type boundary-layer. In the latter case, the flux of TKE caused by breaking surface waves has to be specified. Presently, there is only one possibility to do so implemented in GOTM. It is described in section 4.34. All parameters that determine the boundary layer have to be set in `gotmturb.nml`.
Note that in this section, for brevity, z denotes the distance from the wall (or the surface), and *not* the standard coordinate of the same name used in GOTM.

Viscous boundary-layers

This type is not implemented yet in GOTM.

Logarithmic boundary-layers

The Dirchlet (prescribed) boundary conditions can be written as

$$q^2 l = 2\kappa k (z + z_0) \ , \tag{144}$$

where we used the law-of-the-wall relation $l = \kappa(z + z_0)$.
Neumann (flux) boundary condition can be written as

$$F_l = -S_l q l \frac{\partial q^2 l}{\partial z} = -2\sqrt{2} S_l \kappa^2 k^{\frac{3}{2}} (z + z_0) \tag{145}$$

by inserting $l = \kappa(z + z_0)$ (q is constant in the log-layer). Note, that in (144) and (145), we use `ki`, the value of k at the current time step, to compute the boundary conditions. By means of (103), it would have been also possible to express the boundary conditions in terms of the friction velocity, u_*. This, however, causes numerical difficulties in case of a stress-free surface boundary-layer as for example in the pressure-driven open channel flow.

Shear-free boundary-layers with injection of TKE

The Dirichlet (prescribed) boundary condition follows simply from the power-law (110), yielding

$$q^2 l = 2KL(z + z_0)^{\alpha+1} \quad . \tag{146}$$

Neumann (flux) boundary condition is

$$F_l = -S_l q l \frac{\partial q^2 l}{\partial z} = -2\sqrt{2} S_l (\alpha + 1) K^{\frac{3}{2}} L^2 (z + z_0)^{\frac{3}{2}\alpha+1} \ , \tag{147}$$

which follows from (110). The parameter K is computed as described in the context of (135).

USES:

```
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer, intent(in)                :: bc,type
REALTYPE, intent(in)               :: zi,ki,z0,u_tau
```

REVISION HISTORY:

```
Original author(s): Lars Umlauf
```

4.7.19 Clean up the turbulence module

INTERFACE:

```
subroutine clean_turbulence()
```

DESCRIPTION:

De-allocate all memory allocated in init_turbulence().

USES:

```
IMPLICIT NONE
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding
```

4.7.20 Print the current state of the turbulence module.

INTERFACE:

```
subroutine print_state_turbulence()
```

DESCRIPTION:

This routine writes the value of all module-level variables to screen.

USES:

```
IMPLICIT NONE
```

REVISION HISTORY:

```
Original author(s): Jorn Bruggeman
```

4.8 Update turbulence production

INTERFACE:

```
subroutine production(nlev,NN,SS,xP)
```

DESCRIPTION:

This subroutine calculates the production terms of turbulent kinetic energy as defined in (154) and the production of buoayancy variance as defined in (161). The shear-production is computed according to

$$P = \nu_t (M^2 + \alpha_w N^2) + X_P \ , \tag{148}$$

with the turbulent diffusivity of momentum, ν_t, defined in (46). The shear-frequency, M, is discretised as described in section 3.13. The term multiplied by α_w traces back to a parameterisation of breaking internal waves suggested by *Mellor* (1989). X_P is an extra production term, connected for example with turbulence production caused by sea-grass, see (274) in section 10.1. `xP` is an `optional` argument in the FORTRAN code.
Similarly, according to (80), the buoyancy production is computed from the expression

$$G = -\nu_t^B N^2 + \tilde{\Gamma}_B \ , \tag{149}$$

with the turbulent diffusivity, ν_t^B, defined in (46). The second term in (149) represents the non-local buoyancy flux. The buoyancy-frequency, N, is discretised as described in section 3.14.
The production of buoyancy variance by vertical meanflow gradients follows from (80) and (149)

$$P_b = -G N^2 \quad . \tag{150}$$

Thus, according to the definition of the potential energy (52), the buoyancy production G describes the conversion between turbulent kinetic and potential energy in (152) and (160), respectively.

USES:

```
use turbulence, only: P,B,Pb
use turbulence, only: num,nuh
use turbulence, only: alpha,iw_model
IMPLICIT NONE
```

INPUT PARAMETERS:

```
number of vertical layers
integer,  intent(in)                :: nlev

boyancy frequency squared (1/s^2)
REALTYPE, intent(in)                :: NN(0:nlev)

shear-frequency squared (1/s^2)
REALTYPE, intent(in)                :: SS(0:nlev)

TKE production due to seagrass
friction (m^2/s^3)
REALTYPE, intent(in), optional      :: xP(0:nlev)
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding, Hans Burchard
```

4.9 Update dimensionless alpha's

INTERFACE:

```
subroutine alpha_mnb(nlev,NN,SS)
```

DESCRIPTION:

This subroutine updates the dimensionless numbers α_M, α_N, and α_b according to (47). Note that according to (66) and (69) the following identities are valid

$$\alpha_M = \overline{S}^2 \; , \quad \alpha_N = \overline{N}^2 \; , \quad \alpha_b = \overline{T} \quad . \tag{151}$$

USES:

```
use turbulence,  only:     tke,eps,kb
use turbulence,  only:     as,an,at
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer,  intent(in)      :: nlev
REALTYPE, intent(in)      :: NN(0:nlev),SS(0:nlev)
```

REVISION HISTORY:

```
Original author(s): Lars Umlauf
```

4.10 Update time scale ratio

INTERFACE:

```
subroutine r_ratio(nlev)
```

DESCRIPTION:

This routine updates the ratio r of the dissipation time scales as defined in (68).

USES:

```
use turbulence,  only:     tke,eps,kb,epsb
use turbulence,  only:     r

IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer, intent(in)        :: nlev
```

REVISION HISTORY:

```
Original author(s): Lars Umlauf
```

4.11 The dynamic k-equation

INTERFACE:

```
   subroutine tkeeq(nlev,dt,u_taus,u_taub,z0s,z0b,h,NN,SS)
```

DESCRIPTION:

The transport equation for the turbulent kinetic energy, k, follows immediately from the contraction of the Reynolds-stress tensor. In the case of a Boussinesq-fluid, this equation can be written as

$$\dot{k} = \mathcal{D}_k + P + G - \epsilon \ , \tag{152}$$

where $\dot{k}$ denotes the material derivative of k. P and G are the production of k by mean shear and buoyancy, respectively, and ϵ the rate of dissipation. $\mathcal{D}_k$ represents the sum of the viscous and turbulent transport terms. For horizontally homogeneous flows, the transport term $\mathcal{D}_k$ appearing in (152) is presently expressed by a simple gradient formulation,

$$\mathcal{D}_k = \frac{\partial}{\partial z}\left(\frac{\nu_t}{\sigma_k}\frac{\partial k}{\partial z}\right) \ , \tag{153}$$

where σ_k is the constant Schmidt-number for k.
In horizontally homogeneous flows, the shear and the buoyancy production, P and G, can be written as

$$\begin{aligned} P &= -\langle u'w'\rangle\frac{\partial U}{\partial z} - \langle v'w'\rangle\frac{\partial V}{\partial z} \ , \\ G &= \langle w'b'\rangle \ , \end{aligned} \tag{154}$$

see (50). Their computation is discussed in section 4.8.
The rate of dissipation, ϵ, can be either obtained directly from its parameterised transport equation as discussed in section 4.15, or from any other model yielding an appropriate description of the dissipative length-scale, l. Then, ϵ follows from the well-known cascading relation of turbulence,

$$\epsilon = (c_\mu^0)^3 \frac{k^{\frac{3}{2}}}{l} \ , \tag{155}$$

where c_μ^0 is a constant of the model.

USES:

```
   use turbulence,   only: P,B,num
   use turbulence,   only: tke,tkeo,k_min,eps
   use turbulence,   only: k_bc, k_ubc, k_lbc, ubc_type, lbc_type
   use turbulence,   only: sig_k
   use util,         only: Dirichlet,Neumann

   IMPLICIT NONE
```

INPUT PARAMETERS:

```
number of vertical layers
integer,  intent(in)                :: nlev

time step (s)
REALTYPE, intent(in)                :: dt

surface and bottom
friction velocity (m/s)
REALTYPE, intent(in)                :: u_taus,u_taub

surface and bottom
roughness length (m)
REALTYPE, intent(in)                :: z0s,z0b

layer thickness (m)
REALTYPE, intent(in)                :: h(0:nlev)

square of shear and buoyancy
frequency (1/s^2)
REALTYPE, intent(in)                :: NN(0:nlev),SS(0:nlev)
```

REVISION HISTORY:

```
Original author(s): Lars Umlauf
                   (re-write after first version of
                    H. Burchard and K. Bolding)
```

4.12 The dynamic q2/2-equation

INTERFACE:

```
subroutine q2over2eq(nlev,dt,u_taus,u_taub,z0s,z0b,h,NN,SS)
```

DESCRIPTION:

The transport equation for the TKE $q^2/2 = k$ can be written as

$$\dot{\overline{q^2/2}} = \mathcal{D}_q + P + G - \epsilon \; , \tag{156}$$

where $\dot{\overline{q^2/2}}$ denotes the material derivative of $q^2/2$. With P and G following from (154), evidently, this equation is formally identical to (152). The only reason why it is discretized seperately here, is the slightly different down-gradient model for the transport term,

$$\mathcal{D}_q = \frac{\partial}{\partial z}\left(q l S_q \frac{\partial q^2/2}{\partial z} \right) \; , \tag{157}$$

where S_q is a model constant. The notation has been chosen according to that introduced by *Mellor and Yamada* (1982). Using their notation, also (155) can be expressed in mathematically identical form as

$$\epsilon = \frac{q^3}{B_1 l} \; , \tag{158}$$

where B_1 is a constant of the model. Note, that the equivalence of (155) and (158) requires that

$$(c_\mu^0)^{-2} = \frac{1}{2} B_1^{\frac{2}{3}} \quad . \tag{159}$$

USES:

```
use turbulence,   only: P,B
use turbulence,   only: tke,tkeo,k_min,eps,L
use turbulence,   only: q2over2_bc, k_ubc, k_lbc, ubc_type, lbc_type
use turbulence,   only: sq
use util,         only: Dirichlet,Neumann

IMPLICIT NONE
```

INPUT PARAMETERS:

```
number of vertical layers
integer,  intent(in)               :: nlev

time step (s)
REALTYPE, intent(in)               :: dt

surface and bottom
```

```
friction velocity (m/s)
REALTYPE, intent(in)                :: u_taus,u_taub

surface and bottom
roughness length (m)
REALTYPE, intent(in)                :: z0s,z0b

layer thickness (m)
REALTYPE, intent(in)                :: h(0:nlev)

square of shear and buoyancy
frequency (1/s^2)
REALTYPE, intent(in)                :: NN(0:nlev),SS(0:nlev)
```

REVISION HISTORY:

```
Original author(s): Lars Umlauf
```

4.13 The dynamic kb-equation

INTERFACE:

```
subroutine kbeq(nlev,dt,u_taus,u_taub,z0s,z0b,h,NN,SS)
```

DESCRIPTION:

The transport equation for (half the) buoyancy variance, $k_b = \langle b'^2 \rangle / 2$, follows from the equation for the buoyancy fluctations (see *Sander* (1998)). In the case of a Boussinesq-fluid, this equation can be written as

$$\dot{k_b} = \mathcal{D}_b + P_b - \epsilon_b \ , \tag{160}$$

where $\dot{k_b}$ denotes the material derivative of k_b. P_b is the production of k_b be mean density gradients, and ϵ_b the rate of molecular destruction. $\mathcal{D}_b$ represents the sum of the viscous and turbulent transport terms. It is presently evaluated with a simple down gradient model in GOTM.
The production of buoyancy variance by the vertical density gradient is

$$P_b = -\langle w'b' \rangle \frac{\partial B}{\partial z} = -\langle w'b' \rangle N^2 \quad . \tag{161}$$

Its computation is discussed in section 4.8.
The rate of molecular destruction, ϵ_b, can be computed from either a transport equation or a algebraic expression, section 4.7.10.

USES:

```
use turbulence,   only: Pb,epsb,nuh
use turbulence,   only: kb,kb_min
use turbulence,   only: k_ubc, k_lbc, ubc_type, lbc_type
use util,         only: Dirichlet,Neumann

IMPLICIT NONE
```

INPUT PARAMETERS:

```
number of vertical layers
integer,  intent(in)              :: nlev

time step (s)
REALTYPE, intent(in)              :: dt

surface and bottom
friction velocity (m/s)
REALTYPE, intent(in)              :: u_taus,u_taub

surface and bottom
roughness length (m)
REALTYPE, intent(in)              :: z0s,z0b
```

```
layer thickness (m)
REALTYPE, intent(in)                :: h(0:nlev)

square of shear and buoyancy
frequency (1/s^2)
REALTYPE, intent(in)                :: NN(0:nlev),SS(0:nlev)
```

REVISION HISTORY:

```
Original author(s): Lars Umlauf
```

4.14 The dynamic q2l-equation

INTERFACE:

```
   subroutine lengthscaleeq(nlev,dt,depth,u_taus,u_taub,z0s,z0b,h,NN,SS)
```

DESCRIPTION:

Following suggestions of *Rotta* (1951), *Mellor and Yamada* (1982) proposed an equation for the product q^2l expressed by

$$\dot{\overline{q^2l}} = \mathcal{D}_l + l(E_1P + E_3G - E_2F\epsilon) \ , \tag{162}$$

where $\dot{\overline{q^2l}}$ denotes the material derivative of q^2l. The production terms P and G follow from (154), and ϵ can be computed either directly from (158), or from (155) with the help (159).
The so-called wall function, F, appearing in (162) is defined by

$$F = 1 + E_2 \left(\frac{l}{\kappa \mathcal{L}_z} \right)^2 \ , \tag{163}$$

κ being the von Kármán constant and $\mathcal{L}_z$ some measure for the distance from the wall. Different possiblities for $\mathcal{L}_z$ are implemented in GOTM, which can be activated be setting the parameter `MY_length` in `gotmturb.nml` to appropriate values. Close to the wall, however, one always has $\mathcal{L}_z = \overline{z}$, where $\overline{z}$ is the distance from the wall.
For horizontally homogeneous flows, the transport term $\mathcal{D}_l$ appearing in (162) is expressed by a simple gradient formulation,

$$\mathcal{D}_l = \frac{\partial}{\partial z} \left(q l S_l \frac{\partial q^2 l}{\partial z} \right) \ , \tag{164}$$

where S_l is a constant of the model. The values for the model constants recommended by *Mellor and Yamada* (1982) are displayed in table 6. They can be set in `gotmturb.nml`. Note, that the parameter E_3 in stably stratifed flows is in principle a function of the so-called steady state Richardson-number, as discussed by *Burchard* (2001b), see discussion in the context of (116).

	B_1	S_q	S_l	E_1	E_2	E_3
Mellor and Yamada (1982)	16.6	0.2	0.2	1.8	1.33	1.8

Table 6: Constants appearing in (162) and (158)

At the end of this routine the length-scale can be constrained according to a suggestion of *Galperin et al.* (1988). This feature is optional and can be activated by setting `length_lim = .true.` in `gotmturb.nml`.

USES:

```
   use turbulence, only: P,B
   use turbulence, only: tke,tkeo,k_min,eps,eps_min,L
   use turbulence, only: kappa,e1,e2,e3,b1
   use turbulence, only: MY_length,cm0,cde,galp,length_lim
```

```
use turbulence, only: q2l_bc, psi_ubc, psi_lbc, ubc_type, lbc_type
use turbulence, only: sl
use util,       only: Dirichlet,Neumann

IMPLICIT NONE
```

INPUT PARAMETERS:

```
number of vertical layers
integer,  intent(in)                :: nlev

time step (s)
REALTYPE, intent(in)                :: dt

local water depth (m)
REALTYPE, intent(in)                :: depth

surface and bottom
friction velocity (m/s)
REALTYPE, intent(in)                :: u_taus,u_taub

surface and bottom
roughness length (m)
REALTYPE, intent(in)                :: z0s,z0b

layer thickness (m)
REALTYPE, intent(in)                :: h(0:nlev)

square of shear and buoyancy
frequency (1/s^2)
REALTYPE, intent(in)                :: NN(0:nlev),SS(0:nlev)
```

REVISION HISTORY:

```
Original author(s): Lars Umlauf
                   (re-write after first version of
                    H. Burchard and K. Bolding
```

4.15 The dynamic epsilon-equation

INTERFACE:

```
subroutine dissipationeq(nlev,dt,u_taus,u_taub,z0s,z0b,h,NN,SS)
```

DESCRIPTION:

The k-ϵ model in its form suggested by *Rodi* (1987) has been implemented in GOTM. In this model, the rate of dissipation is balanced according to

$$\dot{\epsilon} = \mathcal{D}_\epsilon + \frac{\epsilon}{k}(c_{\epsilon 1} P + c_{\epsilon 3} G - c_{\epsilon 2} \epsilon) \ , \tag{165}$$

where $\dot{\epsilon}$ denotes the material derivative of ϵ. The production terms P and G follow from (154) and $\mathcal{D}_\epsilon$ represents the sum of the viscous and turbulent transport terms.
For horizontally homogeneous flows, the transport term $\mathcal{D}_\epsilon$ appearing in (165) is presently expressed by a simple gradient formulation,

$$\mathcal{D}_\epsilon = \frac{\partial}{\partial z}\left(\frac{\nu_t}{\sigma_\epsilon}\frac{\partial \epsilon}{\partial z}\right) \ , \tag{166}$$

where σ_ϵ is the constant Schmidt-number for ϵ.
It should be pointed out that not all authors retain the buoyancy term in (165), see e.g. *Gibson and Launder* (1976). Similar to the model of *Mellor and Yamada* (1982), *Craft et al.* (1996) set $c_{\epsilon 1} = c_{\epsilon 3}$. However, in both cases, the k-ϵ model cannot predict a proper state of full equilibrium in stratified flows at a predefined value of the Richardson number (see *Umlauf et al.* (2003) and discussion around (116)). Model constants are summarised in table 7.

	c_μ^0	σ_k	σ_ϵ	$c_{\epsilon 1}$	$c_{\epsilon 2}$
Rodi (1987)	0.5577	1.0	1.3	1.44	1.92

Table 7: Constants appearing in (165) and (155).

At the end of this routine the length-scale can be constrained according to a suggestion of *Galperin et al.* (1988). This feature is optional and can be activated by setting `length_lim = .true.` in `gotmturb.nml`.

USES:

```
use turbulence, only: P,B,num
use turbulence, only: tke,tkeo,k_min,eps,eps_min,L
use turbulence, only: ce1,ce2,ce3plus,ce3minus
use turbulence, only: cm0,cde,galp,length_lim
use turbulence, only: epsilon_bc, psi_ubc, psi_lbc, ubc_type, lbc_type
use turbulence, only: sig_e,sig_e0,sig_peps
use util,       only: Dirichlet,Neumann

IMPLICIT NONE
```

INPUT PARAMETERS:

```
number of vertical layers
integer,  intent(in)                :: nlev

time step (s)
REALTYPE, intent(in)                :: dt

surface and bottom
friction velocity (m/s)
REALTYPE, intent(in)                :: u_taus,u_taub

surface and bottom
roughness length (m)
REALTYPE, intent(in)                :: z0s,z0b

layer thickness (m)
REALTYPE, intent(in)                :: h(0:nlev)

square of shear and buoyancy
frequency (1/s^2)
REALTYPE, intent(in)                :: NN(0:nlev),SS(0:nlev)
```

REVISION HISTORY:

```
Original author(s): Lars Umlauf
                   (re-write after first version of
                    H. Burchard and K. Bolding
```

4.16 The dynamic psi-equation

INTERFACE:

```
subroutine genericeq(nlev,dt,u_taus,u_taub,z0s,z0b,h,NN,SS)
```

DESCRIPTION:

This model has been formulated by *Umlauf and Burchard* (2003), who introduced a 'generic' variable,

$$\psi = (c_\mu^0)^p k^m l^n \ , \tag{167}$$

where k is the turbulent kinetic energy computed from (152) and l is the dissipative length-scale defined in (155). For appropriate choices of the exponents p, m, and n, the variable ψ can be directly identified with the classic length-scale determining variables like the rate of dissipation, ϵ, or the product kl used by *Mellor and Yamada* (1982) (see section 4.14 and section 4.15). Some examples are compiled in table 8.

ψ	two-equation model by:	p	m	n
ω	*Wilcox* (1988)	-1	$\frac{1}{2}$	-1
kl	*Mellor and Yamada* (1982)	0	1	1
ϵ	*Rodi* (1987)	3	$\frac{3}{2}$	-1
$k\tau$	*Zeierman and Wolfshtein* (1986)	-3	$\frac{1}{2}$	1

Table 8: Exponents p, n, m defined in (167), and their relation to the variable of the second equation in some well-known two-equation models.

The transport equation for ψ can written as

$$\dot{\psi} = \mathcal{D}_\psi + \frac{\psi}{k}(c_{\psi_1} P + c_{\psi_3} G - c_{\psi 2}\epsilon) \ , \tag{168}$$

where $\dot{\psi}$ denotes the material derivative of ψ, see *Umlauf and Burchard* (2003). The production terms P and G follow from (154). $\mathcal{D}_\psi$ represents the sum of the viscous and turbulent transport terms. The rate of dissipation can computed by solving (167) for l and inserting the result into (155).
For horizontally homogeneous flows, the transport terms $\mathcal{D}_\psi$ appearing in (168) are expressed by a simple gradient formulation,

$$\mathcal{D}_\psi = \frac{\partial}{\partial z}\left(\frac{\nu_t}{\sigma_\psi}\frac{\partial \psi}{\partial z}\right) \quad . \tag{169}$$

For appropriate choices of the parameters, most of the classic transport equations can be directly recovered from the generic equation (168). An example is the transport equation for the inverse turbulent time scale, $\omega \propto \epsilon/k$, which has been formulated by *Wilcox* (1988) and extended to buoyancy affected flows by *Umlauf et al.* (2003). The precise definition of ω follows from table 8, and its transport equation can be written as

$$\dot{\omega} = \mathcal{D}_\omega + \frac{\omega}{k}(c_{\omega_1} P + c_{\omega_3} G - c_{\omega 2}\epsilon) \ , \tag{170}$$

which is clearly a special case of (168). Model constants for this and other traditional models are given in table 9. Apart from having to code only one equation to recover all of the traditional

	c_μ^0	σ_k^ψ	σ_ψ	$c_{\psi 1}$	$c_{\psi 2}$	$c_{\psi 3}$
k-ϵ, *Rodi* (1987) :	0.5477	1.0	1.3	1.44	1.92	(see eq. (116))
k-kl, *Mellor and Yamada* (1982) :	0.5544	1.96	1.96	0.9	0.5	0.9
k-ω, *Wilcox* (1988) :	0.5477	2	2	0.555	0.833	(see eq. (116))
k-τ *Zeierman and Wolfshtein* (1986):	0.5477	1.46	10.8	0.173	0.225	(—)

Table 9: Model constants of some standard models, converted to the notation used here. The Schmidt-numbers for the model of *Mellor and Yamada* (1982) are valid only in the logarithmic boundary-layer, because the diffusion models (157) and (164) are slightly different from (153) and (169). There is no indication that one class of diffusion models is superior.

models, the main advantage of the generic equation is its flexibility. After choosing meaningful values for physically relevant parameters like the von Kármán constant, κ, the temporal decay rate for homogeneous turbulence, d, some parameters related to breaking surface waves, etc, a two-equation model can be generated, which has exactly the required properties. This is discussed in great detail in *Umlauf and Burchard* (2003). All algorithms have been implemented in GOTM and are described in section 4.7.3.

USES:

```
   use turbulence, only: P,B,num
   use turbulence, only: tke,tkeo,k_min,eps,eps_min,L
   use turbulence, only: cpsi1,cpsi2,cpsi3plus,cpsi3minus,sig_psi
   use turbulence, only: gen_m,gen_n,gen_p
   use turbulence, only: cm0,cde,galp,length_lim
   use turbulence, only: psi_bc, psi_ubc, psi_lbc, ubc_type, lbc_type
   use util,       only: Dirichlet,Neumann

   IMPLICIT NONE
```

INPUT PARAMETERS:

```
   number of vertical layers
   integer,  intent(in)                :: nlev

   time step (s)
   REALTYPE, intent(in)                :: dt

   surface and bottom
   friction velocity (m/s)
   REALTYPE, intent(in)                :: u_taus,u_taub

   surface and bottom
   roughness length (m)
```

```
REALTYPE, intent(in)                :: z0s,z0b

layer thickness (m)
REALTYPE, intent(in)                :: h(0:nlev)

square of shear and buoyancy
frequency (1/s^2)
REALTYPE, intent(in)                :: NN(0:nlev),SS(0:nlev)
```

REVISION HISTORY:

```
Original author(s): Lars Umlauf and Hans Burchard
```

4.17 The algebraic k-equation

INTERFACE:

```
   subroutine tkealgebraic(nlev,u_taus,u_taub,NN,SS)
```

DESCRIPTION:

This subroutine computes the turbulent kinetic energy based on (152), but using the local equilibrium assumption

$$P + G - \epsilon = 0 \quad . \tag{171}$$

This statement can be re-expressed in the form

$$k = (c_\mu^0)^{-3}\, l^2 (c_\mu M^2 - c'_\mu N^2) \; , \tag{172}$$

were we used the expressions in (154) together with (45) and (46). The rate of dissipaton, ϵ, has been expressed in terms of l via (155). This equation has been implemented to update k in a diagnostic way. It is possible to compute the value of k as the weighted average of (172) and the value of k at the old timestep. The weighting factor is defined by the `parameter c_filt`. It is recommended to take this factor small (e.g. `c_filt = 0.2`) in order to reduce the strong oscillations associated with this scheme, and to couple it with an algebraically prescribed length scale with the length scale limitation active (`length_lim=.true.` in `gotmturb.nml`, see *Galperin et al.* (1988)).

USES:

```
   use turbulence,   only: tke,tkeo,L,k_min
   use turbulence,   only: cmue2,cde,cmue1,cm0

   IMPLICIT NONE
```

INPUT PARAMETERS:

```
   number of vertical layers
   integer,  intent(in)                :: nlev

   surface and bottom
   friction velocity (m/s)
   REALTYPE, intent(in)                :: u_taus,u_taub

   square of shear and buoyancy
   frequency (1/s^2)
   REALTYPE, intent(in)                :: NN(0:nlev),SS(0:nlev)
```

DEFINED PARAMETERS:

```
   REALTYPE , parameter                :: c_filt=1.0
```

REVISION HISTORY:

```
   Original author(s): Hans Burchard & Karsten Bolding
```

4.18 The algebraic kb-equation

INTERFACE:

```
   subroutine kbalgebraic(nlev)
```

DESCRIPTION:

The algebraic equation for k_b simply assumes equilibrium in (160),

$$P_b = \epsilon_b \quad . \tag{173}$$

This equation can be re-written as

$$k_b = \frac{k_b \epsilon}{k \epsilon_b} \frac{k}{\epsilon} P_b = r \frac{k}{\epsilon} P_b = c_b \frac{k}{\epsilon} P_b \ , \tag{174}$$

where we used the definition of the time scale ratio r in (68), and assumed that $r = c_b$ is a constant.

USES:

```
   use turbulence,  only:     tke,eps,kb,Pb
   use turbulence,  only:     ctt,kb_min

  IMPLICIT NONE
```

INPUT PARAMETERS:

```
  number of vertical layers
   integer,  intent(in)                :: nlev
```

REVISION HISTORY:

```
   Original author(s): Lars Umlauf
```

4.19 Some algebraic length-scale relations

INTERFACE:

```
   subroutine algebraiclength(method,nlev,z0b,z0s,depth,h,NN)
```

DESCRIPTION:

This subroutine computes the vertical profile of the turbulent scale l from different types of analytical expressions. These range from simple geometrical forms to more complicated expressions taking into account the effects of stratification and shear. The users can select their method in the input file `gotmturb.nml`. For convenience, we define here d_b and d_s as the distance from the bottom and the surface, respectively. The water depth is then given by $H = d_b + d_s$, and z_0^b and z_0^s are the repective roughness lengths. With these abbreviations, the expressions implemented in GOTM are as follows.

1. The parabolic profile is defined according to

$$l = \kappa \frac{(d_s + z_0^s)(d_b + z_0^b)}{d_s + d_b + z_0^b + z_0^s} \ , \tag{175}$$

   where it should be noted that only for large water depth this equation converges to $\kappa(z + z_0)$ near the bottom or near the surface.

2. The triangular profile is defined according to

$$l = \kappa \ \min(d_s + z_0^s, d_b + z_0^b) \ , \tag{176}$$

   which converges always to $\kappa(z + z_0)$ near the bottom or near the surface.

3. A distorted parabola can be constructed by using a slightly modified form of the equation used by *Xing and Davies* (1995),

$$l = \kappa \frac{(d_s + z_0^s)(d_b^{\text{Xing}} + z_0^b)}{d_s + d_b^{\text{Xing}} + z_0^s + z_0^b} \ , \quad d_b^{\text{Xing}} = d_b \exp\left(-\beta \frac{d_b}{H}\right) \ , \tag{177}$$

   where it should be noted that only for large water depth this equation converges to $\kappa(z + z_0)$ near the bottom or near the surface. The constant β is a form parameter determining the distortion of the profile. Currently we use $\beta = 2$ in GOTM.

4. A distorted parabola can be constructed by using a slightly modified form of the equation used by *Robert and Ouellet* (1987),

$$l = \kappa(d_b + z_0^b)\sqrt{1 - \frac{d_b - z_0^s}{H}} \ , \tag{178}$$

   where it should be noted that only for large water depth this equation converges to $\kappa(z + z_0)$ near the bottom. Near the surface, the slope of l is always different from the law of the wall, a fact that becomes important when model solutions for the case of breaking waves are computed, see section 4.7.4.

5. Also the famous formula of *Blackadar* (1962) is based on a parabolic shape, extended by an extra length–scale l_a. Using the form of *Luyten et al.* (1996), the algebraic relation is expressed by

$$l = \left(\frac{1}{\kappa(d_s + z_0^s)} + \frac{1}{\kappa(d_b + z_0^b)} + \frac{1}{l_a} \right) \ , \tag{179}$$

where

$$l_a = \gamma_0 \frac{\int_{-H}^{\eta} k^{\frac{1}{2}} z dz}{\int_{-H}^{\eta} k^{\frac{1}{2}} dz} \tag{180}$$

is the natural kinetic energy scale resulting from the first moment of the rms turbulent velocity. The constant γ_0 usually takes the value $\gamma_0 = 0.2$. It should be noted that this expression for l converges to $\kappa(z + z_0)$ at the surface and the bottom only for large water depth, and when l_a plays only a minor role.

6. The so–called ISPRAMIX method to compute the length–scale is described in detail in section 4.22.

After the length–scale has been computed, it is optionally limited by the method suggested by *Galperin et al.* (1988). This option can be activated in `gotmturb.nml` by setting `length_lim = .true.` The rate of dissipation is computed according to (155).

USES:

```
use turbulence, only: L,eps,tke,k_min,eps_min
use turbulence, only: cde,galp,kappa,length_lim
IMPLICIT NONE
```

INPUT PARAMETERS:

```
type of length scale
integer,  intent(in)               :: method

number of vertical layers
integer,  intent(in)               :: nlev

surface and bottom roughness (m)
REALTYPE, intent(in)               :: z0b,z0s

local depth (m)
REALTYPE, intent(in)               :: depth

layer thicknesses (m)
REALTYPE, intent(in)               :: h(0:nlev)

buoyancy frequency (1/s^2)
REALTYPE, intent(in)               :: NN(0:nlev)
```

DEFINED PARAMETERS:

```
integer, parameter                 :: Parabola=1
integer, parameter                 :: Triangle=2
```

```
   integer, parameter                  :: Xing=3
   integer, parameter                  :: RobertOuellet=4
   integer, parameter                  :: Blackadar=5
   integer, parameter                  :: ispra_length=7
```

REVISION HISTORY:

```
   Original author(s):  Manuel Ruiz Villarreal, Hans Burchard
```

4.20 The algebraic epsilonb-equation

INTERFACE:

```
subroutine epsbalgebraic(nlev)
```

DESCRIPTION:

The algebraic equation for ϵ_b, the molecular rate of destruction of buoyancy variance, see (160), simply assumes a constant time scale ratio $r = c_b$, see (68). From this assumption, it follows immediately that

$$\epsilon_b = \frac{1}{c_b} \frac{\epsilon}{k} k_b \quad . \tag{181}$$

USES:

```
use turbulence,  only:     tke,eps,kb,epsb
use turbulence,  only:     ctt,epsb_min

IMPLICIT NONE
```

INPUT PARAMETERS:

```
number of vertical layers
integer,  intent(in)                :: nlev
```

REVISION HISTORY:

```
Original author(s): Lars Umlauf
```

4.21 The algebraic velocity variances

INTERFACE:

```
   subroutine variances(nlev,SSU,SSV)
```

DESCRIPTION:

Using (72) and the solution shown in (76) and the variances of the turbulent velocity fluctations can be evaluated according to

$$\begin{aligned}
\frac{\langle u'^2 \rangle}{k} &= \frac{2}{3} + \frac{1}{\mathcal{N}\varepsilon}\left(\left(\frac{a_2}{3} + a_3\right)\nu_t\left(\frac{\partial U}{\partial z}\right)^2 - \frac{2}{3}a_2\nu_t\left(\frac{\partial V}{\partial z}\right)^2 - \frac{4}{3}a_5 G\right) \ , \\
\frac{\langle v'^2 \rangle}{k} &= \frac{2}{3} + \frac{1}{\mathcal{N}\varepsilon}\left(\left(\frac{a_2}{3} + a_3\right)\nu_t\left(\frac{\partial V}{\partial z}\right)^2 - \frac{2}{3}a_2\nu_t\left(\frac{\partial U}{\partial z}\right)^2 - \frac{4}{3}a_5 G\right) \ , \\
\frac{\langle w'^2 \rangle}{k} &= \frac{2}{3} + \frac{1}{\mathcal{N}\varepsilon}\left(\left(\frac{a_2}{3} - a_3\right)P + \frac{8}{3}a_5 G\right) \ ,
\end{aligned} \tag{182}$$

where the diffusivities are computed according to (46) (also see section 4.26 and section 4.27), and the buoyancy production, G, follows from (149).

USES:

```
  use turbulence,  only:     uu,vv,ww
  use turbulence,  only:     tke,eps,P,B,num
  use turbulence,  only:     cc1,ct1,a2,a3,a5
  IMPLICIT NONE
```

INPUT PARAMETERS:

```
  number of vertical layers
  integer,  intent(in)                :: nlev

  square of shear frequency (1/s^2)
  (from u- and v-component)
  REALTYPE, intent(in)               :: SSU(0:nlev),SSV(0:nlev)
```

REVISION HISTORY:

```
   Original author(s): Lars Umlauf
```

4.22 Algebraic length-scale from ISPRAMIX

INTERFACE:

```
subroutine ispralength(nlev,NN,h,depth)
```

DESCRIPTION:

This subroutine calculates the lengthscale used in the ISPRAMIX model, see *Eifler and Schrimpf* (1992) and *Demirov et al.* (1998). In both mixing regions (close to the surface and the bottom), l is obtained from the formula

$$l = \frac{\kappa \tilde{z}}{1 + \frac{\kappa \tilde{z}}{c_2 \cdot h_m}} (1 - R_f)^e \qquad (183)$$

where $\tilde{z}$ is the distance from the interface (surface or bottom). The fraction in (183) predicts an approximation to a linear behavior of l near boundaries and a value proportional to the thickness of the mixed layer far from the interface, $l = c_2 h_m$, where $c_2 = 0.065$ is estimated from experimental data as discussed in *Eifler and Schrimpf* (1992). The factor $(1 - R_f)$, with the flux Richardson number $R_f = -G/P$, accounts for the effect of stratification on the length-scale. The parameter e is here a tuning parameter (pers. comm. Walter Eifler, JRC, Ispra, Italy) which is usually set to $e = 1$.

USES:

```
use turbulence, only: L,tke,k_min,eps_min,xRF,kappa,cde

IMPLICIT NONE
```

INPUT PARAMETERS:

```
number of vertical layers
integer,  intent(in)              :: nlev

buoyancy frequency (1/s^2)
REALTYPE, intent(in)              :: NN(0:nlev)

layer thickness (m)
REALTYPE, intent(in)              :: h(0:nlev)

local depth (m)
REALTYPE, intent(in)              :: depth
```

REVISION HISTORY:

```
Original author(s):  Manuel Ruiz Villarreal, Hans Burchard
```

4.23 Algebraic length-scale with two master scales

INTERFACE:

```
   subroutine potentialml(nlev,z0b,z0s,h,depth,NN)
```

DESCRIPTION:

Computes the length scale by defining two master length scales l_u and l_d

$$\begin{aligned} &\int_{z_0}^{z_0+l_u(z_0)} (b(z_0)-b(z))dz = k(z_0) \ , \\ &\int_{z_0-l_d(z_0)}^{z_0} (b(z)-b(z_0))dz = k(z_0) \end{aligned} \tag{184}$$

From l_u and l_d two length–scales are defined: l_k, a characteristic mixing length, and l_ϵ, a characteristic dissipation length. They are computed according to

$$\begin{aligned} &l_k(z_0) = \mathrm{Min}(l_d(z_0), l_u(z_0)) \ , \\ &l_\epsilon(z_0) = (l_d(z_0)l_u(z_0))^{\frac{1}{2}} \quad . \end{aligned} \tag{185}$$

l_k is used in `kolpran()` to compute eddy viscosity/difussivity. l_ϵ is used to compute the dissipation rate, ϵ according to

$$\epsilon = C_\epsilon k^{3/2} l_\epsilon^{-1} \ , \quad C_\epsilon = 0.7 \quad . \tag{186}$$

USES:

```
   use turbulence, only: L,eps,tke,k_min,eps_min
   use turbulence, only: cde,galp,kappa,length_lim

   IMPLICIT NONE
```

INPUT PARAMETERS:

```
   number of vertical layers
   integer,  intent(in)               :: nlev

   bottom and surface roughness (m)
   REALTYPE, intent(in)               :: z0b,z0s

   layer thickness (m)
   REALTYPE, intent(in)               :: h(0:nlev)

   local depth (m)
   REALTYPE, intent(in)               :: depth

   buoyancy frequency (1/s^2)
   REALTYPE, intent(in)               :: NN(0:nlev)
```

REVISION HISTORY:

```
   Original author(s):  Manuel Ruiz Villarreal, Hans Burchard
```

4.24 The non-local, exact weak-equilibrium stability function

INTERFACE:

```
subroutine cmue_a(nlev)
```

DESCRIPTION:

The solution of (72) and (73) has the shape indicated by (76). This subroutine is used to update the quantities c_μ, c'_μ and Γ, defined in (76), from which all turbulent fluxes can be computed. The non-linear terms $\mathcal{N}$ and $\mathcal{N}_b$ are updated by evaluating the right hand side of (67) at the old time step.
The numerators and the denominator appearing in (79) are polynomials of the form

$$\begin{aligned}
D &= d_0 + d_1\overline{N}^2 + d_2\overline{S}^2 + d_3\overline{N}^2\overline{S}^2 + d_4\overline{N}^4 + d_5\overline{S}^4 \ , \\
N_n &= n_0 + n_1\overline{N}^2 + n_2\overline{S}^2 + n_3\overline{T} \ , \\
N_b &= n_{b0} + n_{b1}\overline{N}^2 + n_{b2}\overline{S}^2 \ , \\
N_\Gamma &= (g_0 + g_1\overline{N}^2 + g_2\overline{S}^2)\overline{T} \quad .
\end{aligned} \tag{187}$$

The coefficients of D are given by

$$\begin{aligned}
d_0 &= 36\mathcal{N}^3\mathcal{N}_b^2 \ , \\
d_1 &= 84a_5a_{b3}\mathcal{N}^2\mathcal{N}_b \ , \\
d_2 &= 9(a_{b2}^2 - a_{b1}^2)\mathcal{N}^3 + 12(3a_3^2 - a_2^2)\mathcal{N}\mathcal{N}_b^2 \ , \\
d_3 &= 12(a_2a_{b1} - 3a_3a_{b2})a_5a_{b3}\mathcal{N} + 12(a_3^2 - a_2^2)a_5a_{b3}\mathcal{N}_b \ , \\
d_4 &= 48a_5^2a_{b3}^2\mathcal{N} \ , \\
d_5 &= 3(3a_3^2 - a_2^2)(a_{b2}^2 - a_{b1}^2)\mathcal{N} \quad .
\end{aligned} \tag{188}$$

The coefficients of the numerators N_n and N_b can be expressed as

$$\begin{aligned}
n_0 &= 36a_1\mathcal{N}^2\mathcal{N}_b^2 \ , \\
n_1 &= -12a_5a_{b3}(a_{b1} + a_{b2})\mathcal{N}^2 - 8a_5a_{b3}(-6a_1 + a_2 + 3a_3)\mathcal{N}\mathcal{N}_b \ , \\
n_2 &= 9a_1(a_{b2}^2 - a_{b1}^2)\mathcal{N}^2 \ , \\
n_3 &= 36a_5a_{b4}(a_{b1} + a_{b2})\mathcal{N}^2 + 24a_5a_{b4}(a_2 + 3a_3)\mathcal{N}\mathcal{N}_b \ ,
\end{aligned} \tag{189}$$

$$\begin{aligned}
n_{b0} &= 12a_{b3}\mathcal{N}^3\mathcal{N}_b \ , \\
n_{b1} &= 12a_5a_{b3}^2\mathcal{N}^2 \ , \\
n_{b2} &= 9a_1a_{b3}(a_{b1} - a_{b2})\mathcal{N}^2 + a_{b3}(6a_1(a_2 - 3a_3) - 4(a_2^2 - 3a_3^2))\mathcal{N}\mathcal{N}_b \ ,
\end{aligned} \tag{190}$$

and the numerator of the term Γ is

$$
\begin{aligned}
g_0 &= 36 a_{b4} \mathcal{N}^3 \mathcal{N}_b \ , \\
g_1 &= 36 a_5 a_{b3} a_{b4} \mathcal{N}^2 \ , \\
g_2 &= 12 a_{b4} (3 a_3^2 - a_2^2) \mathcal{N} \mathcal{N}_b \quad .
\end{aligned}
\tag{191}
$$

USES:

```
use turbulence, only: eps
use turbulence, only: P,B,Pb,epsb
use turbulence, only: an,as,at,r
use turbulence, only: cmue1,cmue2,gam
use turbulence, only: cm0
use turbulence, only: cc1
use turbulence, only: ct1,ctt
use turbulence, only: a1,a2,a3,a4,a5
use turbulence, only: at1,at2,at3,at4,at5

IMPLICIT NONE
```

INPUT PARAMETERS:

```
number of vertical layers
integer, intent(in)        :: nlev
```

BUGS:

```
Test stage. Do not yet use.
```

REVISION HISTORY:

```
Original author(s): Lars Umlauf
```

4.25 The non-local, approximate weak-equilibrium stability function

INTERFACE:

```
   subroutine cmue_b(nlev)
```

DESCRIPTION:

This subroutine is used to update the quantities c_μ, c'_μ and Γ, defined in (76), from which all turbulent fluxes can be computed. This done exactly as described in section 4.24, with the exception that equilibrium $P + G = \epsilon$ and $P_b = \epsilon_b$ is assumed in computing the non-linear terms in (67), leading to the particularly simple expressions

$$\mathcal{N} = \frac{c_1}{2} \; , \quad \mathcal{N}_b = c_{b1} \quad . \tag{192}$$

USES:

```
   use turbulence, only: an,as,at
   use turbulence, only: cmue1,cmue2,gam
   use turbulence, only: cm0
   use turbulence, only: cc1
   use turbulence, only: ct1,ctt
   use turbulence, only: a1,a2,a3,a4,a5
   use turbulence, only: at1,at2,at3,at4,at5

   IMPLICIT NONE
```

INPUT PARAMETERS:

```
   number of vertical layers
   integer, intent(in)       :: nlev
```

BUGS:

```
  Test stage. Do not yet use.
```

REVISION HISTORY:

```
   Original author(s): Lars Umlauf
```

4.26 The local, weak-equilibrium stability functions

INTERFACE:

```
   subroutine cmue_c(nlev)
```

DESCRIPTION:

This subroutine updates the explicit solution of (72) and (73) with shape indicated by (76). In addition to the simplifications discussed in section 4.25, $P_b = \epsilon_b$ is assumed in (73) to eliminate the dependency on $\overline{T}$ according to (74). As discussed in section 4.4, this implies that the last of (73) is replaced by (75). Thus, the Γ-term in (76) drops out, and the solution is characterized by c_μ and c'_μ only.
As a consequence, the numerators and the denominator appearing in (79) are of somewhat different form compared to the result in section 4.24. They can be written as

$$\begin{aligned} D &= d_0 + d_1\overline{N}^2 + d_2\overline{S}^2 + d_3\overline{N}^2\overline{S}^2 + d_4\overline{N}^4 + d_5\overline{S}^4 \ , \\ N_n &= n_0 + n_1\overline{N}^2 + n_2\overline{S}^2 \ , \\ N_b &= n_{b0} + n_{b1}\overline{N}^2 + n_{b2}\overline{S}^2 \quad . \end{aligned} \tag{193}$$

The coefficients of D are given by

$$\begin{aligned} d_0 &= 36\mathcal{N}^3\mathcal{N}_b^2 \ , \\ d_1 &= 84a_5a_{b3}\mathcal{N}^2\mathcal{N}_b + 36a_{b5}\mathcal{N}^3\mathcal{N}_b \ , \\ d_2 &= 9(a_{b2}^2 - a_{b1}^2)\mathcal{N}^3 + 12(3a_3^2 - a_2^2)\mathcal{N}\mathcal{N}_b^2 \ , \\ d_3 &= 12(a_2a_{b1} - 3a_3a_{b2})a_5a_{b3}\mathcal{N} + 12(a_3^2 - a_2^2)a_5a_{b3}\mathcal{N}_b \\ &+ 12(3a_3^2 - a_2^2)a_{b5}\mathcal{N}\mathcal{N}_b \ , \\ d_4 &= 48a_5^2a_{b3}^2\mathcal{N} + 36a_5a_{b3}a_{b5}\mathcal{N}^2 \ , \\ d_5 &= 3(3a_3^2 - a_2^2)(a_{b2}^2 - a_{b1}^2)\mathcal{N} \ , \end{aligned} \tag{194}$$

and the coefficients of the numerators are

$$\begin{aligned} n_0 &= 36a_1\mathcal{N}^2\mathcal{N}_b^2 \ , \\ n_1 &= -12a_5a_{b3}(a_{b1} + a_{b2})\mathcal{N}^2 - 8a_5a_{b3}(-6a_1 + a_2 + 3a_3)\mathcal{N}\mathcal{N}_b \\ &+ 36a_1a_{b5}\mathcal{N}^2\mathcal{N}_b \ , \\ n_2 &= 9a_1(a_{b2}^2 - a_{b1}^2)\mathcal{N}^2 \end{aligned} \tag{195}$$

and

$$\begin{aligned} n_{b0} &= 12a_{b3}\mathcal{N}^3\mathcal{N}_b \ , \\ n_{b1} &= 12a_5a_{b3}^2\mathcal{N}^2 \ , \\ n_{b2} &= 9a_1a_{b3}(a_{b1} - a_{b2})\mathcal{N}^2 + a_{b3}(6a_1(a_2 - 3a_3) - 4(a_2^2 - 3a_3^2))\mathcal{N}\mathcal{N}_b \ , \end{aligned} \tag{196}$$

These polynomials correspond to a slightly generalized form of the solution suggested by *Canuto et al.* (2001) and *Cheng et al.* (2002). For cases with unstable stratification, the same clipping conditions on α_N is applied as described in section 4.27. For the cases of extreme shear, the limiter described in the context of (85) is active.

USES:

```
use turbulence, only: an,as,at
use turbulence, only: cmue1,cmue2
use turbulence, only: cm0
use turbulence, only: cc1
use turbulence, only: ct1,ctt
use turbulence, only: a1,a2,a3,a4,a5
use turbulence, only: at1,at2,at3,at4,at5

IMPLICIT NONE
```

INPUT PARAMETERS:

```
number of vertical layers
integer, intent(in)       :: nlev
```

DEFINED PARAMETERS:

```
REALTYPE, parameter       :: asLimitFact=1.0d0
REALTYPE, parameter       :: anLimitFact=0.5d0
```

REVISION HISTORY:

```
Original author(s): Lars Umlauf
```

4.27 The quasi-equilibrium stability functions

INTERFACE:

```
subroutine cmue_d(nlev)
```

DESCRIPTION:

This subroutine updates the explicit solution of (72) and (73) under the same assumptions as those discussed in section 4.26. Now, however, an additional equilibrium assumption is invoked. With the help of (80), one can write the equilibrium condition for the TKE as

$$\frac{P+G}{\epsilon} = \hat{c}_\mu(\alpha_M,\alpha_N)\alpha_M - \hat{c}'_\mu(\alpha_M,\alpha_N)\alpha_N = 1 \ , \tag{197}$$

where (151) has been used. This is an implicit relation to determine α_M as a function of α_N. With the definitions given in section 4.26, it turns out that $\alpha_M(\alpha_N)$ is a quadratic polynomial that is easily solved. The resulting value for α_M is substituted into the stability functions described in section 4.26. For negative α_N (convection) the shear number α_M computed in this way may become negative. The value of α_N is limited such that this does not happen, see *Umlauf and Burchard* (2005).

USES:

```
use turbulence, only: an,as,at
use turbulence, only: cmue1,cmue2
use turbulence, only: cm0
use turbulence, only: cc1
use turbulence, only: ct1,ctt
use turbulence, only: a1,a2,a3,a4,a5
use turbulence, only: at1,at2,at3,at4,at5

IMPLICIT NONE
```

INPUT PARAMETERS:

```
number of vertical layers
integer, intent(in)       :: nlev
```

DEFINED PARAMETERS:

```
REALTYPE, parameter       :: anLimitFact = 0.5D0
REALTYPE, parameter       :: small       = 1.0D-10
```

REVISION HISTORY:

```
Original author(s): Lars Umlauf
```

4.28 The Munk and Anderson (1948) stability function

INTERFACE:

```
subroutine cmue_ma(nlev)
```

DESCRIPTION:

This subroutine computes the stability functions according to *Munk and Anderson* (1948). These are expressed by the empirical relations

$$
\begin{aligned}
&c_\mu = c_\mu^0 \ , \\
&c'_\mu = \frac{c_\mu}{Pr_t^0} \frac{(1+10Ri)^{1/2}}{(1+3.33Ri)^{3/2}} \ , \qquad && Ri \geq 0 \\
&c'_\mu = c_\mu \ , && Ri < 0 \ ,
\end{aligned}
\tag{198}
$$

where where Ri is the gradient Richardson-number and Pr_t^0 is the turbulent Prandtl-number for $Ri \to 0$. Pr_t^0 and the fixed value c_μ^0 have to be set in `gotmturb.nml`.

USES:

```
use turbulence, only: cm0_fix,Prandtl0_fix
use turbulence, only: cmue1,cmue2,as,an
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer, intent(in)                :: nlev
```

REVISION HISTORY:

```
Original author(s): Hans Burchard & Karsten Bolding
```

4.29 The Schumann and Gerz (1995) stability function

INTERFACE:

```
subroutine cmue_sg(nlev)
```

DESCRIPTION:

This subroutine computes stability functions according to

$$c_\mu = c_\mu^0 , \qquad c'_\mu = \frac{c_\mu^0}{Pr_t} \tag{199}$$

with constant c_μ^0. Based simulation data on stratified homogeneous shear-flows, *Schumann and Gerz* (1995) proposed the empirical relation for the turbulent Prandtl–number,

$$Pr_t = Pr_t^0 \exp\left(- \frac{Ri}{Pr_t^0 Ri^\infty} \right) - \frac{Ri}{Ri^\infty} \ , \tag{200}$$

where where Ri is the gradient Richardson–number and Pr_t^0 is the turbulent Prandtl–number for $Ri \rightarrow 0$. Pr_t^0 and the fixed value c_μ^0 have to be set in `gotmturb.nml`. *Schumann and Gerz* (1995) suggested $Pr_t^0 = 0.74$ and $Ri^\infty = 0.25$.

USES:

```
use turbulence, only: Prandtl0_fix,cm0_fix
use turbulence, only: cmue1,cmue2,as,an
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer, intent(in)                :: nlev
```

REVISION HISTORY:

```
Original author(s): Hans Burchard & Karsten Bolding
```

4.30 Flux Richardson number stability function

INTERFACE:

```
subroutine cmue_rf(nlev)
```

DESCRIPTION:

In the ISPRAMIX ocean model (see *Eifler and Schrimpf* (1992)), another approach is used for considering stability effects on vertical mixing. The stability functions in this model are of the form:

$$c_\mu = \text{const} = 0.5, \tag{201}$$

$$c'_\mu = c_\mu f(R_f) = c_\mu \frac{1}{P_r^0}(1 - R_f)^{1/2}. \tag{202}$$

The neutral Prandtl number used there is $P_r^0 = 0.7143$. The function $f(R_f)$ is assumed to lay between the values 0.18 (corresponding to a supercritically stratified situation) and 2.0 (preventing it from growing too much under unstable conditions).
A formulation for $(1 - R_f)$ can be derived from the definition of the flux Richardson number

$$R_f = \frac{c'_\mu}{c_\mu} R_i \tag{203}$$

and (202), see *Beckers* (1995):

$$(1 - R_f) = [(\tilde{R}_i^2 + 1)^{1/2} - \tilde{R}_i]^2 \tag{204}$$

with

$$\tilde{R}_i = \frac{0.5}{P_r^0} R_i \tag{205}$$

where R_i is the gradient Richardson number.

USES:

```
use turbulence, only: cm0_fix,Prandtl0_fix,xRF
use turbulence, only: cmue1,cmue2,an,as
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer, intent(in)                :: nlev
```

REVISION HISTORY:

```
Original author(s):  Manuel Ruiz Villarreal, Hans Burchard
```

4.31 Calculate c3 from steady-state Richardson number

INTERFACE:

```
REALTYPE function compute_cpsi3(c1,c2,Ri)
```

DESCRIPTION:

Numerically computes $c_{\psi 3}$ for two-equation models from given steady-state Richardson-number Ri_{st} and parameters $c_{\psi 1}$ and $c_{\psi 2}$ according to (116). A Newton-iteration is used to solve the resulting implicit non-linear equation.

USES:

```
use turbulence, only:           an,as,cmue1,cmue2
use turbulence, only:           cm0,cm0_fix,Prandtl0_fix
use turbulence, only:           turb_method,stab_method
use turbulence, only:           Constant
use turbulence, only:           MunkAnderson
use turbulence, only:           SchumGerz
use turbulence, only:           EiflerSchrimpf
IMPLICIT NONE
```

INPUT PARAMETERS:

```
REALTYPE, intent(in)            :: c1,c2,Ri
```

REVISION HISTORY:

```
Original author(s): Hans Burchard, Lars Umlauf
```

4.32 Calculate steady-state Richardson number from c3

INTERFACE:

```
   REALTYPE function  compute_rist(c1,c2,c3)
```

DESCRIPTION:

Numerically computes the steady-state Richardson-number Ri_{st} for two-equations models from the given $c_{\psi 3}$ and the parameters $c_{\psi 1}$ and $c_{\psi 2}$ according to (116). A (very tricky) double Newton-iteration is used to solve the resulting implicit non-linear equation.

USES:

```
   use turbulence, only:           as,an,cmue1,cmue2
   use turbulence, only:           cm0
   use turbulence, only:           turb_method,stab_method
   use turbulence, only:           cm0_fix,Prandtl0_fix
   use turbulence, only:           Constant
   use turbulence, only:           MunkAnderson
   use turbulence, only:           SchumGerz
   use turbulence, only:           EiflerSchrimpf
   IMPLICIT NONE
```

INPUT PARAMETERS:

```
   REALTYPE, intent(in)           :: c1,c2,c3
```

REVISION HISTORY:

```
   Original author(s): Hans Burchard, Lars Umlauf
```

4.33 Update internal wave mixing

INTERFACE:

```
subroutine internal_wave(nlev,NN,SS)
```

DESCRIPTION:

Imposes eddy viscosity and diffusivity characteristic of internal wave activity and shear instability when there is extinction of turbulence as suggested by *Kantha and Clayson* (1994). In this case, the new values of ν_t and $\nu'_t = \nu^B_t$, defined in (45), are used instead of those computed with the model.
When k is small (extinction of turbulence, diagnosed by $k <$`klimiw`), ν_t and ν'_t are set to empirical values typical in the presence of internal wave activity (IW) and shear instability (SI). This model is described by

$$\nu_t = \nu_t^{IW} + \nu_t^{SI} \ , \quad \nu'_t = \nu_t'^{IW} + \nu_t'^{SI} \ , \tag{206}$$

where

$$\nu_t^{IW} = 10^{-4} \ , \quad \nu_t'^{IW} = 5 \cdot 10^{-5} \quad . \tag{207}$$

The 'SI' parts are functions of the Richardson number according to

$$\nu_t^{SI} = \nu_t'^{SI} = 0 \ , \qquad R_i > 0.7 \ , \tag{208}$$

$$\nu_t^{SI} = \nu_t'^{SI} = 5 \cdot 10^{-3} \left(1 - \left(\frac{R_i}{0.7} \right)^2 \right)^3 \ , \qquad 0 < R_i < 0.7 \ , \tag{209}$$

$$\nu_t^{SI} = \nu_t'^{SI} = 5 \cdot 10^{-3} \ , \qquad R_i < 0 \quad . \tag{210}$$

The unit of all diffusivities is $\mathrm{m^2s^{-1}}$.

USES:

```
use turbulence,    only:            iw_model,alpha,klimiw,rich_cr
use turbulence,    only:            numiw,nuhiw,numshear
use turbulence,    only:            tke,num,nuh
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer,  intent(in)                :: nlev
REALTYPE, intent(in)                :: NN(0:nlev),SS(0:nlev)
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding, Hans Burchard,
                    Manuel Ruiz Villarreal
```

4.34 TKE flux from wave-breaking

INTERFACE:

```
   REALTYPE  function fk_craig(u_tau,eta)
```

DESCRIPTION:

This functions returns the flux of k caused by breaking surface waves according to

$$F_k = \eta u_*^3 \quad . \tag{211}$$

This form has also been used by *Craig and Banner* (1994), who suggested $\eta \approx 100$.

USES:

```
   IMPLICIT NONE
```

INPUT PARAMETERS:

```
   REALTYPE, intent(in)                :: u_tau
   REALTYPE, intent(in)                :: eta
```

REVISION HISTORY:

```
   Original author(s): Lars Umlauf
```

4.35 Module kpp: the KPP-turbulence model

INTERFACE:

```
module kpp
```

DESCRIPTION:

This implentation of the KPP turbulence parameterisation is based on the publications of *Large et al.* (1994) and *Durksi et al.* (2004). The general expression for the turbulent fluxes used in the KPP model is identical to that suggested in (45). It assumes that the turbulent flux is the sum of a down-gradient flux and a non-local contribution,

$$\langle w'\phi' \rangle = -\nu_t^\phi \frac{\partial \langle \phi \rangle}{\partial z} + \tilde{\Gamma}_\phi \ , \tag{212}$$

where the super- or subscript ϕ is a placeholder for the symbols m, h, and s, indicating whether a quantity relates to momentum, heat, or salinity (or any other tracer), respectively. Note that turbulence parameters due to salinity stratification are updated only if the pre-processor macro `KPP_SALINITY` has been defined in `cppdefs.h`.
In the notation of the KPP model, the non-local flux is expressed as

$$\tilde{\Gamma}_\phi = \nu_t^\phi \gamma_\phi \ , \tag{213}$$

where independent models are valid for ν_t^ϕ and γ_ϕ. The KPP model assumes that the turbulent diffusivity, ν_t^ϕ, inside the surface or bottom boundary layer is determined by a relation of the form

$$\nu_t^\phi = h w_\phi(\sigma) G(\sigma) \ , \tag{214}$$

where h denotes the thickness of the boundary layer, computed according to the algorithm discussed below. The non-dimensional boundary layer coordinate σ is defined according to

$$\sigma = \frac{d}{h} \ , \tag{215}$$

where d is the distance from the free surface (or the bottom boundary). The velocity scale, w_ϕ, in (214) is computed as described in section 4.35.6. The dimensionless shape function G is a cubic polynomial,

$$G(\sigma) = a_0 + a_1\sigma + a_2\sigma^2 + a_3\sigma^3 \quad . \tag{216}$$

Physical arguments discussed in *Large et al.* (1994) require $a_0 = 0$, $a_1 = 1$. The remaining two parameters a_2 and a_3 may be re-expressed in terms of the value of G and its derivative, G', at the edge of the boundary layer, $\sigma = 1$. Then, (216) can be re-expressed as

$$G(\sigma) = \sigma\Big[1 + \sigma\Big((\sigma - 2) + (3 - 2\sigma)\, G(1) + (\sigma - 1)\, G'(1)\Big)\Big] \tag{217}$$

Apart from the boundary layer diffusivities, the KPP model also computes ”interior” diffusivities, here denoted by ν_{ti}^ϕ. The function G and its derivative can be evaluted from the requirement that, at the edge of the boundary layer, the boundary layer diffusivity and its derivative correspond exactly to the interior diffusivity and its derivative, respectively.

Continuity of the boundary and interior diffusivites is obviously insured, see (214), if we require that

$$G(1) = \frac{1}{hw_\phi(1)} \nu_{ti}^\phi(z_{bl}) \ , \tag{218}$$

where z_{bl} denotes the vertical coordinate of the surface (or bottom) boundary layer.
A condition for the continuity of the derivatives of ν_t^ϕ and ν_{ti}^ϕ can be obtained by carrying out the derivative with respect to z of (214), and setting it equal to the z-derivative of ν_{ti}^ϕ. For the surface layer this results in

$$G'(1) = -\frac{G(1)}{w(1)} \frac{\partial w}{\partial \sigma}\bigg|_{\sigma=1} - \frac{1}{w(1)} \frac{\partial \nu_{ti}^\phi}{\partial z}\bigg|_{z=z_{bl}} \ , \tag{219}$$

where we used the relation

$$\frac{\partial}{\partial z} = -\frac{1}{h} \frac{\partial}{\partial \sigma} \ , \tag{220}$$

if the motion of the free surface is ignored.
The derivative of w_ϕ appearing in (219) can be evaluted with the help of the formulae given in section 4.35.6. As discussed in section 4.35.6, at $\sigma = 1$, the derivative of w_ϕ is different from zero only for stably stratified flows. Then, the non-dimensional function Φ_ϕ appearing in (227) is given by (229), and it is easy to show that

$$\frac{\partial w}{\partial \sigma}\bigg|_{\sigma=1} = -5hw_\phi(1) \frac{B_f}{u_*^4} \ , \tag{221}$$

valid for both, bottom and surface boundary layers. Note that in the original publication of *Large et al.* (1994), erroneously, there appears an additional factor κ in this relation.
With the help of (221), one can re-write (219) as

$$G'(1) = \frac{B_f}{u_*^4} \nu_{ti}^\phi\bigg|_{z=z_{bl}} - \frac{1}{w(1)} \frac{\partial \nu_{ti}^\phi}{\partial z}\bigg|_{z=z_{bl}} \ , \tag{222}$$

valid only for the surface boundary layer. For the bottom boundary layer, the minus sign in (220) disappears, with the consequence that the minus sign in (222) has to be replaced by a plus. Note that if the pre-processor macro `KPP_CLIP_GS` is defined in `cppdef.h`, the slope of G is set to zero for negative slopes. For stably stratified flows with a stabilizing buoyancy flux, this limiter breaks the continuity of the first derivatives.
The non-local transport term defined in (213) is computed as described in *Large et al.* (1994), if the pre-processor macro `NONLOCAL` is defined. Otherwise, non-local transport is ignored.
The position of the surface boundary layer depth, z_{bl}, corresponds to the position where the bulk Richardson number,

$$Ri_b = \frac{(B_r - B(z_{bl}))d}{|\boldsymbol{U}_r - \boldsymbol{U}(z_{bl})|^2 + V_t^2(z_{bl})} \ , \tag{223}$$

defined by *Large et al.* (1994), reaches the critical value Ri_c. The subscript "r" in (223) denotes a certain reference value of the buoyancy and velocity close to the surface. The choice of this reference value is not unique, and several possibilities have been implemented in numerical models. Presently, GOTM uses the uppermost grid point as the reference value. The bulk Richardson-number is then computed at the grid faces by linear interpolation of quantities defined at the centers (if `KPP_TWOPOINT_REF` is defined) or by simply identifying the neighbouring center-value with the value at the face. The "turbulent velocity shear", V_t, is computed as described by *Large et al.* (1994). The value of z_{bl} is then found from (223) by linear interpolation.

To check the boundary layer limit according to the condition

$$Ri_b(z_{bl}) = \frac{Ri_{\text{top}}(z_{bl})}{Ri_{\text{bot}}(z_{bl})} = Ri_c \ , \tag{224}$$

two methods have been implemented in GOTM. The first method simply evaluates (224) with a linear interpolation scheme. The second method is activated if the pre-processor macro `KPP_IP_FC` is defined. Then, the condition (224) is reformulated as

$$F_c(z_{bl}) = Ri_{\text{top}}(z_{bl}) - Ri_c Ri_{\text{bot}}(z_{bl}) = 0 \quad . \tag{225}$$

The position where the function F_c changes sign is computed from linear interpolation. This method has been suggested in the ROMS code as the numerically more stable alternative. Clearly, all approaches are grid-depending, a difficulty that cannot be overcome with the KPP model. Finally, provided `clip_mld=.true.` in `kpp.nml`, the boundary layer is cut if it exceeds the Ekman or the Monin-Obukhov length scale, see *Large et al.* (1994).

USES:

```
  use turbulence,   only: num,nuh,nus
  use turbulence,   only: gamu,gamv,gamh,gams
  use turbulence,   only: Rig
  use turbulence,   only: kappa

 ifdef EXTRA_OUTPUT
  use turbulence,   only: turb1,turb2,turb3,turb4,turb5
 endif

   use eqstate

  IMPLICIT NONE

  private
```

PUBLIC MEMBER FUNCTIONS:

```
  public init_kpp, do_kpp, clean_kpp
```

PUBLIC DATA MEMBERS:

```
  z-position of surface boundary layer depth
  REALTYPE, public                :: zsbl

  z-position of bottom boundary layer depth
  REALTYPE, public                :: zbbl
```

DEFINED PARAMETERS:

```
non-dimensional extent of the surface layer (epsilon=0.1)
REALTYPE, parameter :: epsilon = 0.1

critical gradient Richardson number below which turbulent
mixing occurs (Ri0=0.7)
REALTYPE, parameter :: Ri0     = 0.7

value of double-diffusive density ratio where mixing goes
to zero in salt fingering (Rrho0=1.9)
REALTYPE, parameter :: Rrho0    = 1.9

buoancy frequency (1/s2) limit for convection (bvfcon=-2.0E-5)
REALTYPE, parameter :: bvfcon  = -2.0E-5

scaling factor for double diffusion of temperature in salt
fingering case (fdd=0.7)
REALTYPE, parameter :: fdd     = 0.7

maximum interior convective viscosity and diffusivity
due to shear instability (nu0c=0.01)
REALTYPE, parameter :: nu0c    = 0.01

maximum interior viscosity (m2/s) due to shear
instability (nu0m=10.0E-4)
REALTYPE, parameter :: nu0m    = 10.0E-4

maximum interior diffusivity (m2/s) due to shear
instability (nu0s=10.0E-4)
REALTYPE, parameter :: nu0s    = 10.0E-4

scaling factor for double diffusion in salt fingering (nu=1.5E-6)
REALTYPE, parameter :: nu      = 1.5E-6

scaling factor for double diffusion in salt fingering (nuf=10.0E-4)
REALTYPE, parameter :: nuf     = 10.0E-4

interior viscosity (m2/s) due to wave breaking (nuwm=1.0E-5)
REALTYPE, parameter :: nuwm    = 1.0E-5

interior diffusivity (m2/s) due to wave breaking (nuwm=1.0E-6)
REALTYPE, parameter :: nuws    = 1.0E-6

double diffusion constant for salinity in diffusive
convection case (sdd1=0.15)
REALTYPE, parameter :: sdd1    = 0.15

double diffusion constant for salinity in diffusive convection
(sdd2=1.85)
REALTYPE, parameter :: sdd2    = 1.85
```

double diffusion constant for salinity in diffusive convection
(sdd3=0.85)
```
REALTYPE, parameter :: sdd3    = 0.85
```

double diffusion constant for temperature in diffusive convection
(tdd1=0.909)
```
REALTYPE, parameter :: tdd1    = 0.909
```

double diffusion constant for temperature in diffusive convection
(tdd2=4.6)
```
REALTYPE, parameter :: tdd2    = 4.6
```

double diffusion constant for temperature in diffusive convection case
(tdd3=0.54).
```
REALTYPE, parameter :: tdd3    = 0.54
```

proportionality coefficient parameterizing nonlocal transport
(Cstar=10.0)
```
REALTYPE, parameter :: Cstar   = 10.0
```

ratio of interior Brunt-Vaisala frequency to that
at entrainment depth (Cv=1.5-1.6)
```
REALTYPE, parameter :: Cv      = 1.6
```

ratio of entrainment flux to surface buoyancy flux (betaT=-0.2)
```
REALTYPE, parameter :: betaT   = -0.2
```

constant for computation of Ekman scale (cekman=0.7)
```
REALTYPE, parameter :: cekman  = 0.7
```

constant for computation of Monin-Obukhov scale (cmonob = 1.0)
```
REALTYPE, parameter :: cmonob  = 1.0
```

coefficient of flux profile for momentum in their
1/3 power law regimes (am=1.26)
```
REALTYPE, parameter :: am      = 1.257
```

coefficient of flux profile for momentum in their
1/3 power law regimes (as=-28.86)
```
REALTYPE, parameter :: as      = -28.86
```

coefficient of flux profile for momentum in their
1/3 power law regimes (cm=8.38)
```
REALTYPE, parameter :: cm      = 8.38
```

coefficient of flux profile for momentum in their
1/3 power law regimes (cs=98.96)
```
REALTYPE, parameter :: cs      = 98.96
```

maximum stability parameter "zeta" value of the 1/3

```
power law regime of flux profile for momentum (zetam=-0.2)
REALTYPE, parameter :: zetam   = -0.2

maximum stability parameter "zeta" value of the 1/3
power law regime of flux profile for tracers (zetas=-1.0)
REALTYPE, parameter :: zetas   = -1.0
```

REVISION HISTORY:

```
Original author(s): Lars Umlauf
```

LOCAL VARIABLES:

```
proportionality coefficient for
parameterizing non-local transport
REALTYPE                              ::    Cg

coefficient from computation of
turbulent shear velocity
REALTYPE                              ::    Vtc

acceleration of gravity
REALTYPE                              ::    g

reference density
REALTYPE                              ::    rho_0

g/rho_0
REALTYPE                              ::    gorho0

critical bulk Richardson number
REALTYPE                              ::    Ric

compute surface and bottom BBL
logical                               ::    kpp_sbl,kpp_bbl

compute internal mixing
logical                               ::    kpp_interior

use clipping of MLD at Ekman and Monin-Oboukhov scale
logical                               ::    clip_mld

positions of grid faces and centers
REALTYPE, dimension(:), allocatable   ::    z_w,z_r

distance between centers
REALTYPE, dimension(:), allocatable   ::    h_r

integer                               ::    ksblOld
REALTYPE                              ::    zsblOld
```

4.35.1 Initialise the KPP module

INTERFACE:

```
subroutine init_kpp(namlst,fn,nlev,h0,h,kpp_g,kpp_rho_0)
```

DESCRIPTION:

This routine first reads the namelist `kpp`, which has to be contained in a file with filename specified by the string `fn` (typically called `kpp.nml`). Since the `kpp` module uses fields defined in the `turbulence` module, it has to allocate dynamic memory for them. Apart from this, this routine reports the model settings and initialises a number of parameters needed later in the time loop.
If you call the GOTM KPP routines from a three-dimensional model, make sure that this function is called *after* the call to `init_turbulence()`. Also make sure that you pass the correct arguments.

USES:

```
IMPLICIT NONE
```

INPUT PARAMETERS:

```
namelist reference
integer,          intent(in)        :: namlst

filename containing namelist
character(len=*), intent(in)        :: fn

number of grid cells
integer,          intent(in)        :: nlev

bathymetry (m)
REALTYPE,         intent(in)        :: h0

size of grid cells (m)
REALTYPE,         intent(in)        :: h(0:nlev)

acceleration of gravity (m/s^2)
REALTYPE,         intent(in)        :: kpp_g

reference density (kg/m^3)
REALTYPE,         intent(in)        :: kpp_rho_0
```

REVISION HISTORY:

```
Original author(s): Lars Umlauf
```

4.35.2 Loop over the KPP-algorithm

INTERFACE:

```
   subroutine do_kpp(nlev,h0,h,rho,u,v,NN,NNT,NNS,SS,u_taus,u_taub,  &
                     tFlux,btFlux,sFlux,bsFlux,tRad,bRad,f)
```

DESCRIPTION:

Here, the time step for the KPP model is managed. If `kpp_interior=.true.` in `kpp.nml`, the mixing algorithm for the computation of the interior diffusivities is called first. This algorithm is described in section 4.35.3. Then, if `kpp_sbl=.true.` and `kpp_bbl=.true.`, the algorithms for the surface and bottom boundary layer are called. They are described in section 4.35.4 and section 4.35.5, respectively.
If this routine is called from a three-dimensional code, it is essential to pass the correct arguments. The first 3 parameters relate to the numerical grid, discussed in section 3.1.2. Note that `h0` denotes the local bathymetry, i.e. the positive distance between the reference level $z = 0$ and the bottom. The next three parameters denote the potential density, ρ, and the two mean velocity components, U and V. The buoyancy frequency, N^2, and the different contributions to it, N^2_Θ and N^2_S, have to be computed from the potential density as discussed in section 3.14. The shear frequency, M^2, is defined in (36). The vertical discretisation does not necessary have to follow (39), since in the KPP model no TKE equation is solved and thus energy conservation is not an issue. All three-dimensional fields have to be interpolated "in a smart way" to the water column defined by GOTM. The corresponding interpolation schemes may be quite different for the different staggered grids, finite volume, and finite element approaches used in the horizontal. Therefore, we cannot offer a general recipe here.
The bottom friction velocity is computed as described in section 3.9. If this parameter is passed from a three-dimensional code, it has to be insured that the parameter r in (24) is computed consistently, see (25).
All fluxes without exception are counted positive, if they enter the water body. Note that for consistency, the equations of state in GOTM cannot be used if the KPP routines are called from a 3-D model. Therefore, it is necessary to pass the temperature and salinity fluxes, as well as the corresponding buoyancy fluxes. The same applies to the radiative fluxes. The user is responsible for performing the flux conversions in the correct way. To get an idea have a look at section 8.10. The last argument is the Coriolis parameter, f. It is only used for clippling the mixing depth at the Ekman depth.

USES:

```
   IMPLICIT NONE
```

INPUT PARAMETERS:

```
   number of grid cells
   integer                                      :: nlev

   bathymetry (m)
   REALTYPE                                     :: h0

   thickness of grid cells (m)
   REALTYPE                                     :: h(0:nlev)
```

```
potential density at grid centers (kg/m^3)
REALTYPE                                      :: rho(0:nlev)

velocity components at grid centers (m/s)
REALTYPE                                      :: u(0:nlev),v(0:nlev)

square of buoyancy frequency (1/s^2)
REALTYPE                                      :: NN(0:nlev)

square of buoyancy frequency caused by
temperature and salinity stratification
REALTYPE                                      :: NNT(0:nlev),NNS(0:nlev)

square of shear frequency (1/s^2)
REALTYPE                                      :: SS(0:nlev)

surface and bottom friction velocities (m/s)
REALTYPE                                      :: u_taus,u_taub

surface temperature flux (K m/s) and
salinity flux (psu m/s) (negative for loss)
REALTYPE                                      :: tFlux,sFlux

surface buoyancy fluxes (m^2/s^3) due to
heat and salinity fluxes
REALTYPE                                      :: btFlux,bsFlux

radiative flux [ I(z)/(rho Cp) ] (K m/s)
and associated buoyancy flux (m^2/s^3)
REALTYPE                                      :: tRad(0:nlev),bRad(0:nlev)

Coriolis parameter (rad/s)
REALTYPE                                      :: f
```

REVISION HISTORY:

```
Original author(s): Lars Umlauf
```

4.35.3 Compute interior fluxes

INTERFACE:

```
subroutine interior(nlev,NN,NNT,NNS,SS)
```

DESCRIPTION:

Here, the interior diffusivities (defined as the diffusivities outside the surface and bottom boundary layers) are computed. The algorithms are identical to those suggested by *Large et al.* (1994). For numerical efficiency, the algorithms for different physical processes are active only if certain pre-processor macros are defined in `cppdefs.h`.

- The shear instability algorithm is active if the macro `KPP_SHEAR` is defined.
- The internal wave algorithm is active if the macro `KPP_INTERNAL_WAVE` is defined.
- The convective instability algorithm is active if the macro `KPP_CONVEC` is defined.
- The double-diffusion algorithm is active if the macro `KPP_DDMIX` is defined. Note that in this case, the macro `SALINITY` has to be defined as well.

USES:

```
IMPLICIT NONE
```

INPUT PARAMETERS:

```
number of grid cells
integer                                      :: nlev

square of buoyancy frequency (1/s^2)
REALTYPE                                     :: NN(0:nlev)

square of buoyancy frequencies caused by
temperature and salinity stratification
REALTYPE                                     :: NNT(0:nlev),NNS(0:nlev)

square of shear frequency (1/s^2)
REALTYPE                                     :: SS(0:nlev)
```

REVISION HISTORY:

```
Original author(s): Lars Umlauf
```

4.35.4 Compute turbulence in the surface layer

INTERFACE:

```
subroutine surface_layer(nlev,h0,h,rho,u,v,NN,u_taus,u_taub,       &
                         tFlux,btFlux,sFlux,bsFlux,tRad,bRad,f)
```

DESCRIPTION:

In this routine all computations related to turbulence in the surface layer are performed. The algorithms are described in section 4.35. Note that these algorithms are affected by some pre-processor macros defined in `cppdefs.inp`, and by the parameters set in `kpp.nml`, see section 4.35.

USES:

```
IMPLICIT NONE
```

INPUT PARAMETERS:

```
number of grid cells
integer                                      :: nlev

bathymetry (m)
REALTYPE                                     :: h0

thickness of grid cells (m)
REALTYPE                                     :: h(0:nlev)

potential density at grid centers (kg/m^3)
REALTYPE                                     :: rho(0:nlev)

velocity components at grid centers (m/s)
REALTYPE                                     :: u(0:nlev),v(0:nlev)

square of buoyancy frequency (1/s^2)
REALTYPE                                     :: NN(0:nlev)

surface and bottom friction velocities (m/s)
REALTYPE                                     :: u_taus,u_taub

surface temperature flux (K m/s) and
salinity flux (sal m/s) (negative for loss)
REALTYPE                                     :: tFlux,sFlux

surface buoyancy fluxes (m^2/s^3) due to
heat and salinity fluxes
REALTYPE                                     :: btFlux,bsFlux

radiative flux [ I(z)/(rho Cp) ] (K m/s)
and associated buoyancy flux (m^2/s^3)
REALTYPE                                     :: tRad(0:nlev),bRad(0:nlev)

Coriolis parameter (rad/s)
REALTYPE                                     :: f
```

REVISION HISTORY:

```
Original author(s): Lars Umlauf
```

4.35.5 Compute turbulence in the bottom layer

INTERFACE:

```
subroutine bottom_layer(nlev,h0,h,rho,u,v,NN,u_taus,u_taub, &
                        tFlux,btFlux,sFlux,bsFlux,tRad,bRad,f)
```

DESCRIPTION:

In this routine all computations related to turbulence in the bottom layer are performed. The algorithms are described in section 4.35. Note that these algorithms are affected by some pre-processor macros defined in `cppdefs.inp`, and by the parameters set in `kpp.nml`, see section 4.35. The computation of the bulk Richardson number is slightly different from the surface boundary layer, since for the bottom boundary layer this quantity is defined as,

$$Ri_b = \frac{(B(z_{bl}) - B_r)d}{|\boldsymbol{U}(z_{bl}) - \boldsymbol{U}_r|^2 + V_t^2(z_{bl})} \ , \qquad (226)$$

where z_{bl} denotes the position of the edge of the bottom boundary layer.
Also different from the surface layer computations is the absence of non-local fluxes.

USES:

```
IMPLICIT NONE
```

INPUT PARAMETERS:

```
number of grid cells
integer                                     :: nlev

bathymetry (m)
REALTYPE                                    :: h0

thickness of grid cells (m)
REALTYPE                                    :: h(0:nlev)

potential density at grid centers (kg/m^3)
REALTYPE                                    :: rho(0:nlev)

velocity components at grid centers (m/s)
REALTYPE                                    :: u(0:nlev),v(0:nlev)

square of buoyancy frequency (1/s^2)
REALTYPE                                    :: NN(0:nlev)

surface and bottom friction velocities (m/s)
REALTYPE                                    :: u_taus,u_taub

bottom temperature flux (K m/s) and
salinity flux (sal m/s) (negative for loss)
REALTYPE                                    :: tFlux,sFlux

bottom buoyancy fluxes (m^2/s^3) due to
heat and salinity fluxes
REALTYPE                                    :: btFlux,bsFlux
```

```
radiative flux [ I(z)/(rho Cp) ] (K m/s)
and associated buoyancy flux (m^2/s^3)
REALTYPE                                :: tRad(0:nlev),bRad(0:nlev)

Coriolis parameter (rad/s)
REALTYPE                                :: f
```

REVISION HISTORY:

```
Original author(s): Lars Umlauf
```

4.35.6 Compute the velocity scale

INTERFACE:

```
subroutine wscale(Bfsfc,u_taus,d,wm,ws)
```

DESCRIPTION:

This routine computes the turbulent velocity scale for momentum and tracer as a function of the turbulent friction velocity, u_*, the "limited" distance, d_{lim}, and the total buoyancy flux, B_f, according to

$$w_\phi = \frac{\kappa u_*}{\Phi_\phi(\zeta)} \quad . \tag{227}$$

In this equation, Φ_ϕ is a non-dimensional function of the stability parameter $\zeta = d_{\mathrm{lim}}/L$, using the Monin-Obukhov length,

$$L = \frac{u_*^3}{\kappa B_f} \quad . \tag{228}$$

In stable situations, $B_f \geq 0$, the length scale d_{lim} is just the distance from the surface or bottom, d. Then, the non-dimensional function is of the form

$$\Phi_\phi = 1 + \zeta \; , \tag{229}$$

and identical for momentum, heat, and tracers.
In unstable situations, $B_f < 0$, the scale d_{lim} corresponds to the distance from surface or bottom only until it reaches the end of the surface (or bottom) layer at $d = \epsilon h$. Then it stays constant at this maximum value.
The different functional forms of $\Phi_\phi(\zeta)$ for unstable flows are discussed in *Large et al.* (1994).

USES:

```
IMPLICIT NONE
```

INPUT PARAMETERS:

```
buoyancy flux (m^2/s^3)
REALTYPE, intent(in)                :: Bfsfc

friction velocity (m/s)
REALTYPE, intent(in)                :: u_taus

(limited) distance (m)
REALTYPE, intent(in)                :: d
```

OUTPUT PARAMETERS:

```
velocity scale (m/s)
for momentum and tracer
REALTYPE, intent(out)               :: wm, ws
```

REVISION HISTORY:

```
Original author(s): Lars Umlauf
```

4.36 Printing GOTM library version

INTERFACE:

```
subroutine gotm_lib_version(unit)
```

DESCRIPTION:

Simply prints the version number of the GOTM turbulence library to unit.

USES:

```
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer, intent(in)                 :: unit
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding & Hans Burchard
```

5 Air–Sea interaction

5.1 Introduction

This module provides the surface forcing for GOTM. For all dynamic equations, surface boundary conditions need to be specified. For the momentum equations described in section 3.5 and section 3.6, these are the surface momentum fluxes τ_x^s and τ_y^s in $\mathrm{N\,m^{-2}}$. For the temperature equation described in section 3.10, it is the total surface heat flux,

$$Q_{tot} = Q_E + Q_H + Q_B \tag{230}$$

in $\mathrm{W\,m^{-2}}$ that has to be determined[1]. The total surface heat flux Q_{tot} is calculated as the sum of the latent heat flux Q_E, the sensible heat flux Q_H, and the long wave back radiation Q_B. In contrast to the total surface heat flux Q_{tot}, the net short wave radiation at the surface, I_0, is not used as a boundary condition but as a source of heat, as calculated by means of equation (29), see *Paulson and Simpson* (1977). For the salinity equation described in section 3.11, the fresh water fluxes at the surface are given by the difference of the evaporation and the precipitation, p_e, given in $\mathrm{m\,s^{-1}}$, see also the surface boundary condition for salinity, (32). The way how boundary conditions for the transport equations of turbulent quantities are derived, is discussed in section 4. There are basically two ways of calculating the surface heat and momentum fluxes implemented into GOTM. They are either prescribed (as constant values or to be read in from files) or calculated on the basis of standard meteorological data which have to be read in from files. The necessary parameters are the wind velocity vector at 10 m height in $\mathrm{m\,s^{-1}}$, the sea surface temperature (SST in Celsius), air temperature in Celsius), air humidity (either as relative humidity in %, as wet bulb temperature or as dew point temperature in Celsius) and air pressure (in hectopascal), each at 2 m height above the sea surface, and the wind velocity vector at 10 m height in $\mathrm{m\,s^{-1}}$. Instead of the observed SST, also the SST from the model may be used. For the calculation of these fluxes, the bulk formulae of *Kondo* (1975) or **?** are used.

[1]Note, that Q_{tot} has to be divided by the mean density and the specific heat capacity to be used as a boundary condition in (27), since this equation is formulated in terms of the temperature, and the the internal energy

5.2 Module airsea — atmospheric fluxes

INTERFACE:

```
   module airsea
```

DESCRIPTION:

This module calculates the heat, momentum and freshwater fluxes between the ocean and the atmosphere as well as the incoming solar radiation. Fluxes and solar radiation may be prescribed. Alternatively, they may be calculated by means of bulk formulae from observed or modelled meteorological parameters and the solar radiation may be calculated from longitude, latitude, time and cloudiness. For the prescibed fluxes and solar radiation, values may be constant or read in from files. All necessary setting have to be made in the namelist file `airsea.nml`.

USES:

```
   use airsea_variables
   use time,         only: julian_day, time_diff
   use observations, only: read_obs
   IMPLICIT NONE

   default: all is private.
   private
```

PUBLIC MEMBER FUNCTIONS:

```
   public                              :: init_air_sea
   public                              :: do_air_sea
   public                              :: clean_air_sea
   public                              :: set_sst
   public                              :: set_ssuv
   public                              :: integrated_fluxes
 ifdef _PRINTSTATE_
   public                              :: print_state_airsea
 endif
```

PUBLIC DATA MEMBERS:

```
   logical,  public                    :: calc_fluxes
   wind speed (m/s)
   REALTYPE, public, target            :: w
   surface short-wave radiation
   and surface heat flux (W/m^2)
   REALTYPE, public, target            :: I_0
   REALTYPE, public                    :: heat

   surface stress components (Pa)
   REALTYPE, public                    :: tx,ty

   precipitation and  evaporation
```

```
(m/s)
REALTYPE, public, target            :: precip
REALTYPE, public, target            :: evap

sea surface temperature (degC), sea surface salinity (psu),
sea surface current components (m/s)
REALTYPE, public                    :: sst
REALTYPE, public                    :: sss
REALTYPE, public                    :: ssu
REALTYPE, public                    :: ssv

integrated precipitationa and
evaporation + sum (m)
REALTYPE, public                    :: int_precip,int_evap,int_fwf

integrated short-wave radiation,
surface heat flux (J/m^2)
REALTYPE, public                    :: int_swr,int_heat

sum of short wave radiation
and surface heat flux (J/m^2)
REALTYPE, public                    :: int_total
REALTYPE, public                    :: cloud
feedbacks to drag and albedo by biogeochemistry
REALTYPE, target, public            :: bio_drag_scale,bio_albedo
```

DEFINED PARAMETERS:

```
integer,  parameter                 :: meteo_unit=20
integer,  parameter                 :: swr_unit=21
integer,  parameter                 :: heatflux_unit=22
integer,  parameter                 :: momentum_unit=23
integer,  parameter                 :: precip_unit=24
integer,  parameter                 :: sst_unit=25
integer,  parameter                 :: sss_unit=26
integer,  parameter                 :: CONSTVAL=1
integer,  parameter                 :: FROMFILE=2
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding, Hans Burchard
```

LOCAL VARIABLES:

```
logical                   :: init_saved_vars
integer                   :: swr_method
integer                   :: fluxes_method
integer                   :: back_radiation_method
integer                   :: heat_method
integer                   :: momentum_method
integer                   :: precip_method
integer                   :: sst_method
integer                   :: sss_method
integer                   :: ssuv_method
integer                   :: hum_method
logical, public           :: rain_impact
logical, public           :: calc_evaporation

character(len=PATH_MAX)   :: meteo_file
character(len=PATH_MAX)   :: swr_file
character(len=PATH_MAX)   :: heatflux_file
character(len=PATH_MAX)   :: momentumflux_file
character(len=PATH_MAX)   :: precip_file
character(len=PATH_MAX)   :: sss_file
character(len=PATH_MAX)   :: sst_file

REALTYPE                  :: u10,v10
REALTYPE                  :: airp
REALTYPE                  :: airt,twet,tdew
REALTYPE                  :: cloud_obs
REALTYPE                  :: rh

REALTYPE                  :: const_swr
REALTYPE                  :: swr_factor
REALTYPE                  :: const_heat
REALTYPE                  :: const_tx,const_ty
REALTYPE                  :: const_precip
REALTYPE                  :: precip_factor
REALTYPE                  :: dlon,dlat

short_wave_radiation has an optional argument [swr] and therefore needs an explicit interface
interface
   function short_wave_radiation(jul,secs,dlon,dlat,cloud,bio_albedo) result(swr)
      integer, intent(in)                :: jul,secs
      REALTYPE, intent(in)               :: dlon,dlat
      REALTYPE, intent(in)               :: cloud
      REALTYPE, intent(in)               :: bio_albedo
      REALTYPE                           :: swr
   end function short_wave_radiation
end interface
```

BUGS:

```
Wind speed - w - is not entirely correct.
No temporal interpolation is done. If the momentum fluxes tx,ty are
specified w=0.
The Fairall and Kondo methods calculate their own w internally.
w is used by e.g. bio.F90
```

5.2.1 Initialise the air–sea interaction module

INTERFACE:

```
   subroutine init_air_sea(namlst,lat,lon)
```

DESCRIPTION:

This routine initialises the air-sea module by reading various variables from the namelist `airsea.nml` and opens relevant files. These parameters are:

`calc_fluxes`	`.true.`: Sensible, latent and back-radiation are calculated by means of bulk formulae. In this case, `meteo_file` must be given and `hum_method` must be specified.
	`.false.`: Surface fluxes and solar radiation are prescribed.
`fluxes_method`	Select which parameterisation to use for latent and sensible fluxes:
	1: Kondo (1975)
	2: Fairall et al. (1996)
`back_radiation_method`	Select which parameterisation to use:
	1: Clark et al. (1974)
	2: Hastenrath and Lamb (1978)
	3: Bignami et al. (1995)
	4: Berliandand Berliand (1952)
`meteo_file`	file with meteo data (for `calc_fluxes=.true.`) with
	date: `yyyy-mm-dd hh:mm:ss`
	x-component of wind (10 m) in $\mathrm{m\,s^{-1}}$
	y-component of wind (10 m) in $\mathrm{m\,s^{-1}}$
	air pressure (2 m) in hectopascal
	dry air temperature (2 m) in Celsius
	rel. hum. in % or wet bulb temp. in C or dew point temp. in C
	cloud cover in 1/10
	Example:
	`1998-01-01 00:00:00 6.87 10.95 1013.0 6.80 73.2 0.91`
`hum_method`	1: relative humidity given as 7. column in `meteo_file`
	2: wet bulb temperature given as 7. column in `meteo_file`
	3: dew point temperature given as 7. column in `meteo_file`
`heat_method`	0: heat flux not prescribed
	1: constant value for short wave radiation (`const_swr`) and surface heat flux (`const_qh`)
	2: `swr`, `heat` are read from `heatflux_file`
`rain_impact`	`.true.`: include sensible- and momentum-flux from precipitation
`calc_evaporation`	`.true.`: calculate evaporation/condensation (m/s)
`swr_method`	1: constant value for short wave radiation (`const_swr`)
	2: read short wave radiation from file
	3: Solar radiation is calculated from time, longitude, latitude, and cloud cover.
`const_swr`	constant value for short wave radiation in $\mathrm{W\,m^{-2}}$ (always positive)
`swr_file`	file with short wave radiation in $\mathrm{W\,m^{-2}}$
`swr_factor`	scales data read from file to $\mathrm{W\,m^{-2}}$ - defaults to 1
`const_heat`	constant value for surface heat flux in $\mathrm{W\,m^{-2}}$ (negative for heat loss)
`heatflux_file`	file with date and `heat` in $\mathrm{W\,m^{-2}}$
`momentum_method`	0: momentum flux not prescribed
	1: constant momentum fluxes `const_tx`, `const_tx` given
	2: surface momentum fluxes given from file `momentumflux_file`
`const_tx`	x-component of constant surface momentum flux in $\mathrm{N\,m^{-2}}$
`const_ty`	y-component of constant surface momentum flux in $\mathrm{N\,m^{-2}}$
`momentumflux_file`	File with date, `tx` and `ty` given
`precip_method`	0: precipitation not included == precip=0.
	1: constant value for precipitation in $\mathrm{m\,s^{-1}}$
	2: values for precipitation read from file `precip_file`
`const_precip`	value for precip in $\mathrm{m\,s^{-1}}$
`precip_file`	file with date and `precip` in $\mathrm{m\,s^{-1}}$
`precip_factor`	scales data read from file to $\mathrm{m\,s^{-1}}$ - defaults to 1
`sst_method`	0: no independent SST observation is read from file
	2: independent SST observation is read from file, only for output

USES:

```
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer, intent(in)                :: namlst
REALTYPE, intent(in)               :: lat,lon
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding
```

5.2.2 Obtain the air–sea fluxes

INTERFACE:

```
subroutine do_air_sea(jul,secs)
```

DESCRIPTION:

Depending on the value of the boolean variable `calc_fluxes`, the subroutines for the calculation of the fluxes and the short wave radiation are called or the fluxes are directly read in from the namelist `airsea.nml` as constants or read in from files. Furthermore, the surface freshwater flux is set to a constant value or is read in from a file.

USES:

```
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer, intent(in)                :: jul,secs
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding
```

5.2.3 Finish the air–sea interactions

INTERFACE:

```
subroutine clean_air_sea
```

DESCRIPTION:

All files related to air-sea interaction which have been opened are now closed by this routine.

USES:

```
IMPLICIT NONE
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding
```

5.2.4 Read meteo data, interpolate in time

INTERFACE:

```
   subroutine flux_from_meteo(jul,secs)
```

DESCRIPTION:

For `calc_fluxes=.true.`, this routine reads meteo data from `meteo_file` and calculates for each time step in `meteo_file` the fluxes of heat and momentum, and the long-wave back radiation by calling the routines `humidity`, `back_radiation` and `airsea_fluxes`, see sections section 5.4, section 5.5, and section 5.6, a wrapper routine for using the bulk fomulae from either *Kondo* (1975) or **?**. Afterwards, the airsea fluxes are interpolated to the actual time step of GOTM. Finally, the incoming short-wave radiation is calculated by using the interpolated cloud cover and the actual UTC time of GOTM, see the routine `short_wave_radiation` described in section 5.9.

USES:

```
   IMPLICIT NONE
```

INPUT PARAMETERS:

```
   integer, intent(in)                :: jul,secs
```

REVISION HISTORY:

```
   Original author(s): Karsten Bolding
```

5.2.5 Read short wave radiation, interpolate in time

INTERFACE:

```
   subroutine read_swr(jul,secs,swr)
```

DESCRIPTION:

This routine reads the short wave radiation (in $\mathrm{W\,s^{-2}}$) from `swr_file` and interpolates in time.

USES:

```
   IMPLICIT NONE
```

INPUT PARAMETERS:

```
   integer, intent(in)                :: jul,secs
```

OUTPUT PARAMETERS:

```
   REALTYPE,intent(out)               :: swr
```

REVISION HISTORY:

```
   Original author(s): Karsten Bolding
```

5.2.6 Read heat flux data, interpolate in time

INTERFACE:

```
subroutine read_heat_flux(jul,secs,heat)
```

DESCRIPTION:

For `calc_fluxes=.false.`, this routine reads in the sum of sensible, latent and long-wave back-radiation flux in $\mathrm{W\,m^{-2}}$ from `heatflux_file` and interpolate in time.

USES:

```
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer, intent(in)                :: jul,secs
```

OUTPUT PARAMETERS:

```
REALTYPE, intent(out)              :: heat
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding
```

5.2.7 Read momentum flux data, interpolate in time

INTERFACE:

```
subroutine read_momentum_flux(jul,secs,tx,ty)
```

DESCRIPTION:

For `calc_fluxes=.false.`, this routine reads momentum fluxes in $\mathrm{N\,m^{-2}}$ from `momentumflux_file` and interpolates them in time.

USES:

```
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer,intent(in)                 :: jul,secs
```

OUTPUT PARAMETERS:

```
REALTYPE,intent(out)               :: tx,ty
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding
```

LOCAL VARIABLES:

```
integer                  :: yy,mm,dd,hh,min,ss
REALTYPE                 :: t,alpha
REALTYPE, save           :: dt
integer, save            :: mom_jul1,mom_secs1
integer, save            :: mom_jul2,mom_secs2
REALTYPE, save           :: obs1(2),obs2(2)
integer                  :: rc
```

5.2.8 Read precipitation, interpolate in time

INTERFACE:

```
subroutine read_precip(jul,secs,precip)
```

DESCRIPTION:

This routine reads the precipitation (in $\mathrm{m\,s^{-1}}$) from `precip_file` and interpolates in time.

USES:

```
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer, intent(in)               :: jul,secs
```

OUTPUT PARAMETERS:

```
REALTYPE,intent(out)              :: precip
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding
```

5.2.9 Read SST, interpolate in time

INTERFACE:

```
subroutine read_sst(jul,secs,sst)
```

DESCRIPTION:

For `calc_fluxes=.false.`, this routine reads sea surface temperature (SST) from `sst_file` and interpolates in time.

USES:

```
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer, intent(in)                :: jul,secs
```

OUTPUT PARAMETERS:

```
REALTYPE,intent(out)               :: sst
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding
```

5.2.10 Read SSS, interpolate in time

INTERFACE:

```
subroutine read_sss(jul,secs,sss)
```

DESCRIPTION:

For `calc_fluxes=.false.`, this routine reads sea surface salinity (SSS) from `sss_file` and interpolates in time.

USES:

```
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer, intent(in)                :: jul,secs
```

OUTPUT PARAMETERS:

```
REALTYPE,intent(out)               :: sss
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding
```

5.2.11 Integrate short–wave and sea surface fluxes

INTERFACE:

```
subroutine integrated_fluxes(dt)
```

DESCRIPTION:

This utility routine integrates the short–wave radiation and heat–fluxes over time.

USES:

```
IMPLICIT NONE
```

INPUT PARAMETERS:

```
REALTYPE, intent(in)                :: dt
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding
```

5.2.12 Set the SST to be used from model.

INTERFACE:

```
subroutine set_sst(temp)
```

DESCRIPTION:

This routine sets the simulated sea surface temperature (SST) to be used for the surface flux calculations.

USES:

```
IMPLICIT NONE
```

INPUT PARAMETERS:

```
REALTYPE, intent(in)                :: temp
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding
```

5.2.13 Set the surface velocities to be used from model.

INTERFACE:

```
subroutine set_ssuv(uvel,vvel)
```

DESCRIPTION:

This routine sets the simulated sea surface velocities to be used for the surface flux calculations.

USES:

```
IMPLICIT NONE
```

INPUT PARAMETERS:

```
REALTYPE, intent(in)                :: uvel,vvel
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding
```

5.2.14 Print the current state of the air–sea interaction module.

INTERFACE:

```
subroutine print_state_airsea()
```

DESCRIPTION:

This routine writes the value of all module-level variables to screen.

USES:

```
IMPLICIT NONE
```

REVISION HISTORY:

```
Original author(s): Jorn Bruggeman
```

5.3 Module airsea_variables

INTERFACE:

```
module airsea_variables
```

DESCRIPTION:

Here, number of public variables in the airsea module is declared.

USES:

```
IMPLICIT NONE

default: all is private.
private
```

PUBLIC DATA MEMBERS:

```
REALTYPE, public, parameter         :: cpa=1008.
REALTYPE, public, parameter         :: cpw=3985.
REALTYPE, public, parameter         :: emiss=0.97
REALTYPE, public, parameter         :: bolz=5.67e-8
REALTYPE, public, parameter         :: kelvin=273.16
REALTYPE, public, parameter         :: const06=0.62198
REALTYPE, public, parameter         :: rgas = 287.1
REALTYPE, public, parameter         :: g = 9.81          [m/s2]
REALTYPE, public, parameter         :: rho_0 = 1025.     [kg/m3]
REALTYPE, public, parameter         :: kappa = 0.41      von Karman
REALTYPE, public                    :: es
REALTYPE, public                    :: ea
REALTYPE, public                    :: qs
REALTYPE, public                    :: qa
REALTYPE, public                    :: L
REALTYPE, public                    :: rhoa
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding, Hans Burchard
```

5.4 Calculate the humidity

INTERFACE:

```
subroutine humidity(hum_method,hum,airp,tw,ta)
```

DESCRIPTION:

This routine calculated the saturation vapour pressure at SST and at air temperature, as well as the saturation specific humidty and the specific humidity. For the latter, four methods are implemented, and the method has to be chosen in the namelist file `airsea.nml` as parameter `hum_method`, see section 5.2.1 for details.

USES:

```
use airsea_variables, only: kelvin,const06,rgas
use airsea_variables, only: es,ea,qs,qa,rhoa
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer, intent(in)                :: hum_method
REALTYPE, intent(in)               :: hum,airp,tw,ta
```

OUTPUT PARAMETERS:

REVISION HISTORY:

```
Original author(s): Adolf Stips, Hans Burchard & Karsten Bolding
```

DEFINED PARAMETERS:

```
REALTYPE, parameter       :: a1=6.107799961
REALTYPE, parameter       :: a2=4.436518521e-1
REALTYPE, parameter       :: a3=1.428945805e-2
REALTYPE, parameter       :: a4=2.650648471e-4
REALTYPE, parameter       :: a5=3.031240396e-6
REALTYPE, parameter       :: a6=2.034080948e-8
REALTYPE, parameter       :: a7=6.136820929e-11
```

LOCAL VARIABLES:

```
REALTYPE        :: rh,twet,twet_k,dew,dew_k
```

5.5 Calculate the long-wave back radiation

INTERFACE:

```
   subroutine back_radiation(method,dlat,tw,ta,cloud,qb)
```

DESCRIPTION:

Here, the long-wave back radiation is calculated by means of one out of four methods, which depend on the value given to the parameter `method`: `method`=1: *Clark et al.* (1974), `method`=2: *Hastenrath and Lamb* (1978), `method`=3: *Bignami et al.* (1995), `method`=4: *Berliand and Berliand* (1952). It should ne noted that the latitude must here be given in degrees.

USES:

```
   use airsea_variables, only: emiss,bolz
   use airsea_variables, only: ea,qa
   IMPLICIT NONE
```

INPUT PARAMETERS:

```
   integer, intent(in)                :: method
   REALTYPE, intent(in)               :: dlat,tw,ta,cloud
```

OUTPUT PARAMETERS:

```
   REALTYPE, intent(out)              :: qb
```

REVISION HISTORY:

```
   Original author(s): Adols Stips, Hans Burchard & Karsten Bolding
```

LOCAL VARIABLES:

```
   integer, parameter   :: clark=1        Clark et al, 1974
   integer, parameter   :: hastenrath=2   Hastenrath and Lamb, 1978
   integer, parameter   :: bignami=3      Bignami et al., 1995 - Medsea
   integer, parameter   :: berliand=4     Berliand and Berliand, 1952 - ROMS


   REALTYPE, parameter, dimension(91)  :: cloud_correction_factor = (/ &
     0.497202,     0.501885,     0.506568,     0.511250,     0.515933, &
     0.520616,     0.525299,     0.529982,     0.534665,     0.539348, &
     0.544031,     0.548714,     0.553397,     0.558080,     0.562763, &
     0.567446,     0.572129,     0.576812,     0.581495,     0.586178, &
     0.590861,     0.595544,     0.600227,     0.604910,     0.609593, &
     0.614276,     0.618959,     0.623641,     0.628324,     0.633007, &
     0.637690,     0.642373,     0.647056,     0.651739,     0.656422, &
     0.661105,     0.665788,     0.670471,     0.675154,     0.679837, &
     0.684520,     0.689203,     0.693886,     0.698569,     0.703252, &
     0.707935,     0.712618,     0.717301,     0.721984,     0.726667, &
     0.731350,     0.736032,     0.740715,     0.745398,     0.750081, &
     0.754764,     0.759447,     0.764130,     0.768813,     0.773496, &
```

```
    0.778179,     0.782862,     0.787545,     0.792228,     0.796911, &
    0.801594,     0.806277,     0.810960,     0.815643,     0.820326, &
    0.825009,     0.829692,     0.834375,     0.839058,     0.843741, &
    0.848423,     0.853106,     0.857789,     0.862472,     0.867155, &
    0.871838,     0.876521,     0.881204,     0.885887,     0.890570, &
    0.895253,     0.899936,     0.904619,     0.909302,     0.913985, &
    0.918668 /)

  REALTYPE                 :: ccf
  REALTYPE                 :: x1,x2,x3
```

5.6 Wrapper for air-sea fluxes calculations

INTERFACE:

```
   subroutine airsea_fluxes(method,sst,airt,u10,v10,precip, &
                            evap,taux,tauy,qe,qh)
```

DESCRIPTION:

A wrapper around the different methods for calculating momentum fluxes and sensible and latent heat fluxes at the air-sea interface. To have a complete air-sea exchange also the short wave radiation and back-wave radiation must be calculated.

USES:

```
   IMPLICIT NONE
```

INPUT PARAMETERS:

```
   integer, intent(in)                :: method
   REALTYPE, intent(in)               :: sst,airt,u10,v10,precip
```

INPUT/OUTPUT PARAMETERS:

```
   REALTYPE, intent(inout)            :: evap
```

OUTPUT PARAMETERS:

```
   REALTYPE, intent(out)              :: taux,tauy,qe,qh
```

REVISION HISTORY:

```
   Original author(s): Karsten Bolding
```

5.7 Heat and momemtum fluxes according to Kondo

INTERFACE:

```
subroutine kondo(sst,airt,u10,v10,precip,evap,taux,tauy,qe,qh)
```

DESCRIPTION:

Based on the model sea surface temperature, the wind vector at 10 m height, the air pressure at 2 m, the dry air temperature and the air pressure at 2 m, and the relative humidity (either directly given or recalculated from the wet bulb or the dew point temperature), this routine first computes the transfer coefficients for the surface momentum flux vector, (τ_x^s, τ_y^s) (c_{dd}), the latent heat flux, Q_e, (c_{ed}) and the sensible heat flux, Q_h, (c_{hd}) heat flux according to the *Kondo* (1975) bulk formulae. Afterwards, these fluxes are calculated according to the following formulae:

$$\begin{aligned} \tau_x^s &= c_{dd}\rho_a W_x W \\ \tau_y^s &= c_{dd}\rho_a W_y W \\ Q_e &= c_{ed} L \rho_a W (q_s - q_a) \\ Q_h &= c_{hd} C_{pa} \rho_a W (T_w - T_a) \end{aligned} \tag{231}$$

with the air density ρ_a, the wind speed at 10 m, W, the x- and the y-component of the wind velocity vector, W_x and W_y, respectively, the specific evaporation heat of sea water, L, the specific saturation humidity, q_s, the actual specific humidity q_a, the specific heat capacity of air at constant pressure, C_{pa}, the sea surface temperature, T_w and the dry air temperature, T_a.

USES:

```
use airsea_variables, only: kelvin,const06,rgas,rho_0
use airsea_variables, only: qs,qa,rhoa
use airsea_variables, only: cpa,cpw
use airsea, only: rain_impact,calc_evaporation
IMPLICIT NONE
```

INPUT PARAMETERS:

```
REALTYPE, intent(in)              :: sst,airt,u10,v10,precip
```

INPUT/OUTPUT PARAMETERS:

```
REALTYPE, intent(inout)           :: evap
```

OUTPUT PARAMETERS:

```
REALTYPE, intent(out)             :: taux,tauy,qe,qh
```

REVISION HISTORY:

```
Original author(s): Hans Burchard and Karsten Bolding
```

LOCAL VARIABLES:

```
   REALTYPE                  :: w,L
   REALTYPE                  :: s,s0
   REALTYPE                  :: ae_d,be_d,pe_d
   REALTYPE                  :: ae_h,be_h,ce_h,pe_h
   REALTYPE                  :: ae_e,be_e,ce_e,pe_e
   REALTYPE                  :: x,x1,x2,x3
   REALTYPE                  :: ta,ta_k,tw,tw_k
   REALTYPE                  :: cdd,chd,ced
   REALTYPE                  :: tmp,rainfall,cd_rain
   REALTYPE, parameter       :: eps=1.0e-12
```

5.8 Heat and momentum fluxes according to Fairall et al.

INTERFACE:

```
   subroutine fairall(sst,airt,u10,v10,precip,evap,taux,tauy,qe,qh)
```

DESCRIPTION:

The surface momentum flux vector, (τ_x^s, τ_y^s), in $[\mathrm{N\,m^{-2}}]$, the latent heat flux, Q_e, and the sensible heat flux, Q_h, both in $[\mathrm{W\,m^{-2}}]$ are calculated here according to the *Fairall et al.* (1996b) bulk formulae, which are build on the Liu-Katsaros-Businger (*Liu et al.* (1979)) method. Cool skin and warm layer effects are considered according to the suggestions of *Fairall et al.* (1996a).
The air temperature airt and the sea surface temperature sst may be given in Kelvin or Celcius: if they are > 100 - Kelvin is assumed.
This piece of code has been adapted from the COARE code originally written by David Rutgers and Frank Bradley - see http://www.coaps.fsu.edu/COARE/flux_algor/flux.html.

USES:

```
   use airsea_variables, only: kelvin,const06,rgas,rho_0,g,rho_0,kappa
   use airsea_variables, only: qs,qa,rhoa
   use airsea_variables, only: cpa,cpw
   use airsea, only: rain_impact,calc_evaporation
   IMPLICIT NONE
```

INPUT PARAMETERS:

```
   REALTYPE, intent(in)              :: sst,airt,u10,v10,precip
```

INPUT/OUTPUT PARAMETERS:

```
   REALTYPE, intent(inout)           :: evap
```

OUTPUT PARAMETERS:

```
   REALTYPE, intent(out)             :: taux,tauy,qe,qh
```

REVISION HISTORY:

```
   Original author(s): Adolf Stips
```

DEFINED PARAMETERS:

```
   Fairall LKB roughness Reynolds number to Von Karman
   REALTYPE,parameter        :: fdg = 1.0            non-dimensional

   Beta parameter evaluated from Fairall low windspeed turbulence data.
   REALTYPE,parameter        :: beta = 1.2           non-dimensional

   Zabl      Height (m) of atmospheric boundary layer.
   REALTYPE,parameter        :: Zabl = 600.0         in [m]

   REALTYPE, parameter       :: r3 = 1.0/3.0
```

```
  Liu et al. (1979) look-up table coefficients to compute roughness
  Reynolds number for temperature (rt) and moisture (rq) as a
  function of wind Reynolds number (rr):
       rt = Liu_a(:,1) * Rr   ** Liu_b(:,1)    temperature
       rq = Liu_a(:,2) * Rr   ** Liu_b(:,2)    moisture
  REALTYPE,parameter, dimension(8,2) :: Liu_a = reshape ( &
              (/ 0.177,  1.376,    1.026,      1.625,   &
                 4.661, 34.904, 1667.190, 588000.0,     &
                 0.292,  1.808,    1.393,      1.956,   &
                 4.994, 30.709, 1448.680, 298000.0 /),  &
              (/ 8, 2 /) )

  REALTYPE,parameter, dimension(8,2) :: Liu_b = reshape ( &
              (/  0.0,    0.929, -0.599, -1.018,        &
                 -1.475, -2.067, -2.907, -3.935,        &
                  0.0,    0.826, -0.528, -0.870,        &
                 -1.297, -1.845, -2.682, -3.616 /),     &
              (/ 8, 2 /) )

  REALTYPE,parameter, dimension(9) :: Liu_Rr =            &
              (/    0.0,  0.11,   0.825,   3.0,         &
                   10.0, 30.0,  100.0,   300.0,         &
                 1000.0 /)
  Height (m) of surface air temperature measurement.
  REALTYPE, parameter       ::  zt= 2.0
  Height (m) of surface air humidity measurement
  REALTYPE, parameter       ::  zq= 2.0
  Height (m) of surface winds measurement
  REALTYPE, parameter       ::  zw= 10.0
  integer,  parameter       :: itermax = 20
ifdef GUSTINESS
  REALTYPE, parameter       :: wgust=0.2
else
  REALTYPE, parameter       :: wgust=0.0
endif

  REALTYPE,external         :: psi
```

LOCAL VARIABLES:

```
  REALTYPE                  :: tmp,cff,wgus
  REALTYPE                  :: L
  REALTYPE                  :: Cd
  REALTYPE                  :: ta,ta_k,tw,tw_k
  integer                   :: ier,iter,k
  REALTYPE                  :: vis_air
  REALTYPE                  :: tpsi,qpsi,wpsi,ZWoL,oL,ZToL,ZQoL,ZoW,ZoT, ZoQ
  REALTYPE                  :: Wstar,Tstar, Qstar, delQ, delT, rr,rt,rq
  REALTYPE                  :: TVstar,Bf, upvel,delw,Wspeed, w
  REALTYPE                  :: ri,cd_rain
```

```
   REALTYPE                 :: x1,x2,x3
   REALTYPE                 :: x
   REALTYPE                 :: rainfall
   REALTYPE, parameter      :: eps=1.0e-12
```

5.9 Calculate the short–wave radiation

INTERFACE:

```
function short_wave_radiation(jul,secs,dlon,dlat,cloud,bio_albedo) result(swr)
```

DESCRIPTION:

This subroutine calculates the short–wave net radiation based on latitude, longitude, time, fractional cloud cover and albedo. The albedo monthly values from *Payne* (1972) are given here as means of the values between at 30° N and 40° N for the Atlantic Ocean (hence the same latitudinal band of the Mediterranean Sea). The basic formula for the short-wave radiation at the surface, Q_s, has been taken from *Rosati and Miyakoda* (1988), who adapted the work of *Reed* (1977) and *Simpson and Paulson* (1999):

$$Q_s = Q_{tot}(1 - 0.62C + 0.0019\beta)(1 - \alpha), \tag{232}$$

with the total radiation reaching the surface under clear skies, Q_{tot}, the fractional cloud cover, C, the solar noon altitude, β, and the albedo, α. This piece of code has been taken the MOM-I (Modular Ocean Model) version at the INGV (Istituto Nazionale di Geofisica e Vulcanologia, see `http://www.bo.ingv.it/`).

USES:

```
use time, only: calendar_date
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer, intent(in)                 :: jul,secs
REALTYPE, intent(in)                :: dlon,dlat
REALTYPE, intent(in)                :: cloud
REALTYPE, intent(in)                :: bio_albedo
```

OUTPUT PARAMETERS:

```
REALTYPE                            :: swr
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding & Hans Burchard
```

LOCAL VARIABLES:

```
REALTYPE, parameter       :: pi=3.14159265358979323846
REALTYPE, parameter       :: deg2rad=pi/180.
REALTYPE, parameter       :: rad2deg=180./pi

REALTYPE, parameter       :: solar=1350.
REALTYPE, parameter       :: eclips=23.439*deg2rad
REALTYPE, parameter       :: tau=0.7
REALTYPE, parameter       :: aozone=0.09
```

```
REALTYPE                 :: th0,th02,th03,sundec
REALTYPE                 :: thsun,coszen,zen,dzen,sunbet
REALTYPE                 :: qatten,qzer,qdir,qdiff,qtot,qshort
REALTYPE                 :: albedo
integer                  :: jab
integer                  :: yy,mm,dd
REALTYPE                 :: yrdays,days,hour,tjul
REALTYPE                 :: rlon,rlat

integer, parameter       :: yday(12) = &
              (/ 0,31,59,90,120,151,181,212,243,273,304,334 /)

REALTYPE, parameter      :: alb1(20) = &
              (/.719,.656,.603,.480,.385,.300,.250,.193,.164, &
                .131,.103,.084,.071,.061,.054,.039,.036,.032,.031,.030 /)

REALTYPE, parameter      :: za(20) = &
              (/90.,88.,86.,84.,82.,80.,78.,76.,74.,70.,  &
                66.,62.,58.,54.,50.,40.,30.,20.,10.,0.0 /)

REALTYPE                 :: dza(19)
data           dza/8*2.0, 6*4.0, 5*10.0/
```

6 Working with observed data in GOTM

In the context of GOTM, the term ‘observations’ should be understood in a broad sense: it may refer to data either measured in nature or generated artificially. The inclusion of such data into GOTM can serve different purposes. Examples are time-series of external pressure-gradients, which can be used to drive the model, or observed profiles of the temperature to which model results can be relaxed.
Two different types of ‘observations’ are considered so far in GOTM: time series of scalar data and time series of profile data. The first type is used to introduce, for example, sea surface elevations into the model. The latter is used to include, for example, temperature or velocity fields.
All specifications concerning the ‘observations’ are done via the namelist file `obs.inp`. Each of type of variable has its own namelist in `obs.inp`. Common for all namelists is a member with the suffix `_method`, used to specify the action performed to generate or acquire the variable, respectively. Observations can be, for example, read-in from files or computed according to an analytical expression. Some types of observations (e.g. turbulent dissipation rates) are not used directly during the calculations in GOTM. but can be conveniently interpolated to the numerical grid to allow for an easy comparison of measured data and model results.
For all types of observations, one `_method` is always ‘from file’. All input-files are in ASCII with a very straight-forward format. The necessary interpolation in space is performed as an integral part of the general reading routines. Temporal interpolation is performed as part of the specific reading routines, e.g. `get_s_profile.F90`.

6.1 Module observations — the 'real' world

INTERFACE:

```
module observations
```

DESCRIPTION:

This module provides the necessary subroutines for communicating 'observations' to GOTM. The module operates according to the general philosophy used in GOTM, i.e. it provides `init_observations()` to be called in the overall initialisation routine and `get_all_obs()` to be called in the time loop to actually obtain the 'observations'. In addition to these subroutines the module also provides two routines for reading scalar-type observations and profile-type observations. Each observation has a date stamp with the format `yyyy-mm-dd hh:dd:mm`. The module uses the `time` module (see section 8.11) to convert the time string to the internal time representation of GOTM. Profiles are interpolated to the actual GOTM model grid. Free format is used for reading-in the actual data.

USES:

```
IMPLICIT NONE

default: all is private.
private
```

PUBLIC MEMBER FUNCTIONS:

```
   public init_observations, get_all_obs, read_obs, read_profiles,&
          clean_observations
 ifdef _PRINTSTATE_
   public print_state_observations
 endif
```

PUBLIC DATA MEMBERS:

```
logical, public                                :: init_saved_vars
'observed' salinity profile
REALTYPE, public, dimension(:), allocatable, target :: sprof

'observed' temperature profile
REALTYPE, public, dimension(:), allocatable   :: tprof

'observed' oxygen profile
REALTYPE, public, dimension(:), allocatable   :: o2_prof

'observed' horizontal salinity  gradients
REALTYPE, public, dimension(:), allocatable   :: dsdx,dsdy

'observed' horizontal temperarure  gradients
REALTYPE, public, dimension(:), allocatable   :: dtdx,dtdy
```

```
  internal horizontal pressure gradients
  REALTYPE, public, dimension(:), allocatable   :: idpdx,idpdy

  horizontal velocity profiles
  REALTYPE, public, dimension(:), allocatable   :: uprof,vprof

  observed profile of turbulent dissipation rates
  REALTYPE, public, dimension(:), allocatable   :: epsprof

  ralaxation times for salinity and temperature
  REALTYPE, public, dimension(:), allocatable, target :: SRelaxTau
  REALTYPE, public, dimension(:), allocatable         :: TRelaxTau

  sea surface elevation, sea surface gradients and height of velocity obs.
  REALTYPE, public          :: zeta,dpdx,dpdy,h_press

  vertical advection velocity
  REALTYPE, public          :: w_adv,w_height

  Parameters for water classification - default Jerlov type I
  REALTYPE, public, target  :: A,g1,g2

ifdef BIO
  'observed' biological profiles
  REALTYPE, public, dimension(:,:), allocatable :: bioprofs
endif

-------------------------------------------------------------------------------
 the following data are not all public,
 but have been included for clarity here
-------------------------------------------------------------------------------

  Salinity profile(s)
  integer, public           :: s_prof_method
  integer, public           :: s_analyt_method
  character(LEN=PATH_MAX)   :: s_prof_file
  REALTYPE                  :: z_s1,s_1,z_s2,s_2
  REALTYPE                  :: s_obs_NN
  REALTYPE                  :: SRelaxTauM
  REALTYPE                  :: SRelaxTauS
  REALTYPE                  :: SRelaxTauB
  REALTYPE                  :: SRelaxSurf
  REALTYPE                  :: SRelaxBott

  Temperature profile(s)
  integer, public           :: t_prof_method
  integer, public           :: t_analyt_method
  character(LEN=PATH_MAX)   :: t_prof_file
  REALTYPE                  :: z_t1,t_1,z_t2,t_2
  REALTYPE                  :: t_obs_NN
  REALTYPE                 :: TRelaxTauM
  REALTYPE                 :: TRelaxTauS
  REALTYPE                 :: TRelaxTauB
  REALTYPE                 :: TRelaxSurf
  REALTYPE                 :: TRelaxBott

  Oxygen profile(s)
  integer, public          :: o2_prof_method
  integer, public          :: o2_units
  character(LEN=PATH_MAX)  :: o2_prof_file

  External pressure - 'press' namelist
  integer, public          :: ext_press_method,ext_press_mode
  character(LEN=PATH_MAX)  :: ext_press_file
  REALTYPE, public         :: PressConstU
  REALTYPE, public         :: PressConstV
  REALTYPE, public         :: PressHeight
  REALTYPE, public         :: PeriodM
  REALTYPE, public         :: AmpMu
  REALTYPE, public         :: AmpMv
  REALTYPE, public         :: PhaseMu
  REALTYPE, public         :: PhaseMv
  REALTYPE, public         :: PeriodS
  REALTYPE, public         :: AmpSu
  REALTYPE, public         :: AmpSv
  REALTYPE, public         :: PhaseSu
  REALTYPE, public         :: PhaseSv

  Internal pressure - 'internal_pressure' namelist
  integer, public          :: int_press_method
  character(LEN=PATH_MAX)  :: int_press_file
  REALTYPE, public         :: const_dsdx
  REALTYPE, public         :: const_dsdy
  REALTYPE, public         :: const_dtdx
  REALTYPE, public         :: const_dtdy
  logical, public          :: s_adv
  logical, public          :: t_adv

  Light extinction - the 'extinct' namelist
  integer                  :: extinct_method
  character(LEN=PATH_MAX)  :: extinct_file

  Vertical advection velocity - 'w_advspec' namelist
  integer, public          :: w_adv_method
  REALTYPE, public         :: w_adv0
  REALTYPE, public         :: w_adv_height0
  character(LEN=PATH_MAX)  :: w_adv_file
  integer, public          :: w_adv_discr

  Sea surface elevations - 'zetaspec' namelist
```

```
  integer,public           :: zeta_method
  character(LEN=PATH_MAX)  :: zeta_file
  REALTYPE, public         :: zeta_0
  REALTYPE, public         :: period_1
  REALTYPE, public         :: amp_1
  REALTYPE, public         :: phase_1
  REALTYPE, public         :: period_2
  REALTYPE, public         :: amp_2
  REALTYPE, public         :: phase_2

  Wind waves - 'wave_nml' namelist
  integer,public           :: wave_method
  character(LEN=PATH_MAX)  :: wave_file
  REALTYPE, public         :: Hs
  REALTYPE, public         :: Tz
  REALTYPE, public         :: phiw

  Observed velocity profile profiles - typically from ADCP
  integer                  :: vel_prof_method
  CHARACTER(LEN=PATH_MAX)  :: vel_prof_file
  REALTYPE, public         :: vel_relax_tau
  REALTYPE, public         :: vel_relax_ramp

  Observed dissipation profiles
  integer                  :: e_prof_method
  REALTYPE                 :: e_obs_const
  CHARACTER(LEN=PATH_MAX)  :: e_prof_file

  Buoyancy - 'bprofile' namelist
  REALTYPE, public         :: b_obs_surf,b_obs_NN
  REALTYPE, public         :: b_obs_sbf

ifdef BIO
  Observed biological profiles
  integer, public          :: bio_prof_method
  CHARACTER(LEN=PATH_MAX)  :: bio_prof_file
endif

  REALTYPE,public, parameter:: pi=3.141592654d0
```

DEFINED PARAMETERS:

```
  Unit numbers for reading observations/data.
  integer, parameter       :: s_prof_unit=30
  integer, parameter       :: t_prof_unit=31
  integer, parameter       :: ext_press_unit=32
  integer, parameter       :: int_press_unit=33
  integer, parameter       :: extinct_unit=34
  integer, parameter       :: w_adv_unit=35
  integer, parameter       :: zeta_unit=36
  integer, parameter        :: wave_unit=37
  integer, parameter        :: vel_prof_unit=38
  integer, parameter        :: e_prof_unit=39
  integer, parameter        :: o2_prof_unit=40
ifdef BIO
  integer, parameter        :: bio_prof_unit=41
endif

  pre-defined parameters
  integer, parameter        :: READ_SUCCESS=1
  integer, parameter        :: END_OF_FILE=-1
  integer, parameter        :: READ_ERROR=-2
  integer, parameter        :: NOTHING=0
  integer, parameter        :: ANALYTICAL=1
  integer, parameter        :: CONSTANT=1
  integer, parameter        :: FROMFILE=2
  integer, parameter        :: CONST_PROF=1
  integer, parameter        :: TWO_LAYERS=2
  integer, parameter        :: CONST_NN=3
```

REVISION HISTORY:

```
  Original author(s): Karsten Bolding & Hans Burchard
```

6.1.1 Initialise the observation module

INTERFACE:

```
  subroutine init_observations(namlst,fn,julday,secs,                &
                               depth,nlev,z,h,gravity,rho_0)
```

DESCRIPTION:

The `init_observations()` subroutine basically reads the `obs.nml` file with a number of different namelists and takes actions according to the specifications in the different namelists. In this routine also memory is allocated to hold the 'observations'. Finally, all variables are initialised to sane values, either by reading from files, by prescribing constant values, or by using analytical expressions.

USES:

```
  IMPLICIT NONE
```

INPUT PARAMETERS:

```
  integer, intent(in)               :: namlst
  character(len=*), intent(in)      :: fn
  integer, intent(in)               :: julday,secs
  REALTYPE, intent(in)              :: depth
  integer, intent(in)               :: nlev
  REALTYPE, intent(in)              :: z(0:nlev),h(0:nlev)
  REALTYPE, intent(in)              :: gravity,rho_0
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding & Hans Burchard
```

6.1.2 get_all_obs

INTERFACE:

```
subroutine get_all_obs(julday,secs,nlev,z)
```

DESCRIPTION:

During the time integration this subroutine is called each time step to update all 'observation'. The routine is basically a wrapper routine which calls the variable specific routines. The only input to this routine is the time (in internal GOTM representation) and the vertical grid. It is up to each of the individual routines to use this information and to provide updated 'observations'.

USES:

```
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer, intent(in)                :: julday,secs
integer, intent(in)                :: nlev
REALTYPE, intent(in)               :: z(:)
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding & Hans Burchard
```

6.1.3 read_obs

INTERFACE:

```
subroutine read_obs(unit,yy,mm,dd,hh,min,ss,N,obs,ierr)
```

DESCRIPTION:

This routine will read all non-profile observations. The routine allows for reading more than one scalar variable at a time. The number of data to be read is specified by `N`. Data read-in are returned in the 'obs' array. It is up to the calling routine to assign meaning full variables to the individual elements in `obs`.

USES:

```
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer, intent(in)                :: unit
integer, intent(in)                :: N
```

OUTPUT PARAMETERS:

```
integer, intent(out)               :: yy,mm,dd,hh,min,ss
REALTYPE,intent(out)               :: obs(:)
integer, intent(out)               :: ierr
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding & Hans Burchard
```

6.1.4 read_profiles

INTERFACE:

```
subroutine read_profiles(unit,nlev,cols,yy,mm,dd,hh,min,ss,z, &
                         profiles,lines,ierr)
```

DESCRIPTION:

Similar to `read_obs()` but used for reading profiles instead of scalar data. The data will be interpolated on the grid specified by nlev and z. The data can be read 'from the top' or 'from the bottom' depending on a flag in the actual file.

USES:

```
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer, intent(in)                :: unit
integer, intent(in)                :: nlev,cols
REALTYPE, intent(in)               :: z(:)
```

INPUT/OUTPUT PARAMETERS:

```
integer, intent(inout)             :: lines
```

OUTPUT PARAMETERS:

```
integer, intent(out)               :: yy,mm,dd,hh,min,ss
REALTYPE, intent(out)              :: profiles(:,:)
integer, intent(out)               :: ierr
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding & Hans Burchard
```

6.1.5 Clean up the observation module

INTERFACE:

```
subroutine clean_observations()
```

DESCRIPTION:

De-allocates memory allocated in init_observations().

USES:

```
IMPLICIT NONE
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding & Hans Burchard
```

6.1.6 Print the current state of the observations module.

INTERFACE:

```
subroutine print_state_observations()
```

DESCRIPTION:

This routine writes the value of all module-level variables to screen.

USES:

```
IMPLICIT NONE
```

REVISION HISTORY:

```
Original author(s): Jorn Bruggeman
```

6.1.7 get_s_profile

INTERFACE:

```
subroutine get_s_profile(unit,jul,secs,nlev,z)
```

DESCRIPTION:

This routine is responsible for providing sane values to an 'observed' salinity profile. The subroutine is called in the `get_all_obs()` subroutine as part of the main integration loop. In case of observations from file the temporal interpolation is done in this routine.

USES:

```
use time
use observations, only: init_saved_vars,read_profiles,sprof
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer, intent(in)                :: unit
integer, intent(in)                :: jul,secs
integer, intent(in)                :: nlev
REALTYPE, intent(in)               :: z(0:nlev)
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding
```

6.1.8 get_t_profile

INTERFACE:

```
subroutine get_t_profile(unit,jul,secs,nlev,z)
```

DESCRIPTION:

This routine is responsible for providing sane values to an 'observed' temperature profile. The subroutine is called in the `get_all_obs()` subroutine as part of the main integration loop. In case of observations from file the temporal interpolation is done in this routine.

USES:

```
use time
use observations, only: init_saved_vars,read_profiles,tprof
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer, intent(in)                :: unit
integer, intent(in)                :: jul,secs
integer, intent(in)                :: nlev
REALTYPE, intent(in)               :: z(0:nlev)
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding
```

6.1.9 get_o2_profile

INTERFACE:

```
   subroutine get_o2_profile(unit,jul,secs,nlev,z)
```

DESCRIPTION:

This routine is responsible for providing sane values to an 'observed' oxygen profile. The subroutine is called in the `get_all_obs()` subroutine as part of the main integration loop. In case of observations from file the temporal interpolation is done in this routine.

USES:

```
   use time
   use observations, only: init_saved_vars,read_profiles,o2_prof,o2_units
   IMPLICIT NONE
```

INPUT PARAMETERS:

```
   integer, intent(in)                :: unit
   integer, intent(in)                :: jul,secs
   integer, intent(in)                :: nlev
   REALTYPE, intent(in)               :: z(0:nlev)
```

REVISION HISTORY:

```
   Original author(s): Karsten Bolding
```

6.1.10 get_ext_pressure

INTERFACE:

```
   subroutine get_ext_pressure(method,unit,jul,secs)
```

DESCRIPTION:

This routine will provide the external pressure-gradient, either from analytical expressions or read-in from a file. The subroutine is called in `get_all_obs()` as part of the main integration loop. In case of observations from file the temporal interpolation is done in this routine.

USES:

```
   use time,         only: time_diff,julian_day,fsecs
   use observations, only: init_saved_vars,read_obs
   use observations, only: pi,h_press,dpdx,dpdy
   use observations, only: AmpMu,AmpMv,PhaseMu,PhaseMv,PeriodM
   use observations, only: AmpSu,AmpSv,PhaseSu,PhaseSv,PeriodS
   use observations, only: PressConstU,PressConstV,PressHeight
   IMPLICIT NONE
```

INPUT PARAMETERS:

```
   integer, intent(in)                :: method,unit,jul,secs
```

REVISION HISTORY:

```
   Original author(s): Karsten Bolding
```

6.1.11 get_int_pressure

INTERFACE:

```
subroutine get_int_pressure(method,unit,jul,secs,nlev,z)
```

DESCRIPTION:

This routine will provide the internal pressure-gradients, either analytically prescribed or read from a file. The subroutine is called in the `get_all_obs()` subroutine as part of the main integration loop. The spatial interpolation is done via the reading routine and the temporal interpolation is done in this routine.

USES:

```
use time, only: time_diff,julian_day
use observations, only: init_saved_vars,read_profiles
use observations, only: dsdx,dsdy,dtdx,dtdy
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer, intent(in)                :: method
integer, intent(in)                :: unit
integer, intent(in)                :: jul,secs
integer, intent(in)                :: nlev
REALTYPE, intent(in)               :: z(0:nlev)
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding
```

6.1.12 read_extinction

INTERFACE:

```
subroutine read_extinction(unit,jul,secs)
```

DESCRIPTION:

This routine will provide the light extinction coefficients. It is only called if no Jerlov class has been specified in `obs.nml`.

USES:

```
use time
use observations, only : read_obs,init_saved_vars
use observations, only : A,g1,g2
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer, intent(in)                :: unit,jul,secs
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding
```

6.1.13 get_w_adv

INTERFACE:

```
subroutine get_w_adv(method,unit,jul,secs)
```

DESCRIPTION:

This routine is responsible for providing sane values to 'observed' vertical velocity which will then be applied for vertical advection of mean flow properties. A height and a vertical velocity value are either set to constant values or read from a file. The height will be assigned to be the position of maximum vertical velocity, and the vertical profiles of vertical velocity will be then constructed in such a way that the velocity is linearly decreasing away from this height, with zero values at the surface and the bottom. The subroutine is called in the `get_all_obs()` subroutine as part of the main integration loop. In case of observations from file the temporal interpolation is done in this routine.

USES:

```
use time,         only: time_diff,julian_day
use observations, only: init_saved_vars,read_obs
use observations, only: w_adv,w_adv0,w_adv_height0,w_height
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer, intent(in)                :: method,unit,jul,secs
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding
```

6.1.14 get_zeta

INTERFACE:

```
subroutine get_zeta(method,unit,jul,secs)
```

DESCRIPTION:

This routine will provide sea surface elevation - either by an analytical expression or read from file. The subroutine is called in the `get_all_obs()` subroutine as part of the main integration loop. The spatial interpolation is done via the reading routine and the temporal interpolation is done in this routine.

USES:

```
use time,         only: time_diff,julian_day,fsecs
use observations, only: pi,init_saved_vars,read_obs
use observations, only: period_1,amp_1,phase_1,period_2,amp_2,phase_2
use observations, only: zeta,zeta_0
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer, intent(in)                 :: method,unit,jul,secs
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding
```

6.1.15 get_wave

INTERFACE:

```
subroutine get_wave(unit,jul,secs)
```

DESCRIPTION:

This routine is responsible for providing sane values to 'observed' wind generated waves. The observations consist of significant wave height (Hs), mean zero-crossing period (Tz) and mean direction (phiw). The variables can be set to constant values (wave_method=1) or read from file (wave_method=2). For wave_method=0 nothing is done. The subroutine is called in the `get_all_obs()` subroutine as part of the main integration loop. In case of observations from file the temporal interpolation is done in this routine.

USES:

```
use time,         only: time_diff,julian_day
use observations, only: init_saved_vars,read_obs
use observations, only: Hs,Tz,phiw
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer, intent(in)                 :: unit,jul,secs
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding
```

LOCAL VARIABLES:

```
integer                    :: yy,mm,dd,hh,min,ss
REALTYPE                   :: t
REALTYPE, save             :: dt
integer, save              :: jul1,secs1
integer, save              :: jul2,secs2
REALTYPE, save             :: alpha(3)
REALTYPE, save             :: obs1(3),obs2(3)
integer                    :: rc
```

6.1.16 get_vel_profile

INTERFACE:

```
subroutine get_vel_profile(unit,jul,secs,nlev,z)
```

DESCRIPTION:

This routine is responsible for providing sane values to 'observed' velocity profiles. The subroutine is called in the `get_all_obs` subroutine as part of the main integration loop. In case of observations from file the temporal interpolation is done in this routine.

USES:

```
use time
use observations, only: init_saved_vars,read_profiles,uprof,vprof
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer, intent(in):: unit
integer, intent(in):: jul,secs
integer, intent(in):: nlev
REALTYPE, intent(in):: z(0:nlev)
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding
```

6.1.17 get_eps_profile

INTERFACE:

```
subroutine get_eps_profile(unit,jul,secs,nlev,z)
```

DESCRIPTION:

This routine will get the observed dissipation profiles. The subroutine is called in the `get_all_obs` subroutine as part of the main integration loop. The spatial interpolation is done via the reading routine and the temporal interpolation is done in this routine.

USES:

```
   use time
   use observations, only: init_saved_vars,read_profiles,epsprof
   IMPLICIT NONE
```

INPUT PARAMETERS:

```
   integer, intent(in)                :: unit
   integer, intent(in)                :: jul,secs
   integer, intent(in)                :: nlev
   REALTYPE, intent(in)               :: z(0:nlev)
```

REVISION HISTORY:

```
   Original author(s): Karsten Bolding
```

6.1.18 analytical_profile

INTERFACE:

```
   subroutine analytical_profile(nlev,z,z1,v1,z2,v2,prof)
```

DESCRIPTION:

This routine creates a vertical profile `prof` with value `v1` in a surface layer down to depth `z1` and a bottom layer of value `v2` reaching from depth `z2` down to the bottom. Both layers are connected by an intermediate layer reaching from `z1` to `z2` with values linearly varying from `v1` to `v2`.

USES:

```
   IMPLICIT NONE
```

INPUT PARAMETERS:

```
   integer,  intent(in)               :: nlev
   REALTYPE, intent(in)               :: z(0:nlev)
   REALTYPE, intent(in)               :: z1,v1,z2,v2
```

OUTPUT PARAMETERS:

```
   REALTYPE, intent(out)              :: prof(0:nlev)
```

REVISION HISTORY:

```
   Original author(s): Karsten Bolding
```

6.1.19 const_NNT

INTERFACE:

```
subroutine const_NNT(nlev,z,T_top,S_const,NN,gravity,rho_0,T)
```

DESCRIPTION:

This routine creates a vertical profile `prof` with value `v1`

USES:

```
use eqstate
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer,  intent(in)              :: nlev
REALTYPE, intent(in)              :: z(0:nlev)
REALTYPE, intent(in)              :: T_top,S_const,NN
REALTYPE, intent(in)              :: gravity,rho_0
```

OUTPUT PARAMETERS:

```
REALTYPE, intent(out)             :: T(0:nlev)
```

REVISION HISTORY:

```
Original author(s): Lars Umlauf
```

6.1.20 const_NNS

INTERFACE:

```
subroutine const_NNS(nlev,z,S_top,T_const,NN,gravity,rho_0,S)
```

DESCRIPTION:

This routine creates a vertical profile `prof` with value `v1`

USES:

```
use eqstate
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer,  intent(in)              :: nlev
REALTYPE, intent(in)              :: z(0:nlev)
REALTYPE, intent(in)              :: S_top,T_const,NN
REALTYPE, intent(in)              :: gravity,rho_0
```

OUTPUT PARAMETERS:

```
REALTYPE, intent(out)             :: S(0:nlev)
```

REVISION HISTORY:

```
Original author(s): Lars Umlauf
```

7 Saving the results

GOTM provides an easily extendible interface for storing calculated results. The main specifications are given via the `output` namelist in `gotmrun.inp`. The most important member in this namelist is the integer `out_fmt`. Changing this variable will select the output format — presently ASCII and NetCDF are supported.
In GOTM output is triggered by `do_output()` called inside the main integration loop (see section 2.3). Completely separated from the core of GOTM, a format specific subroutine is called to do the actual output. We strongly recommend to use the NetCDF format — mainly because it is well established and save — but also because a large number of graphical programmes can read NetCDF. Another reason is the powerful package 'nco' which provides some nice programs for manipulating NetCDF files. Information about how to install and use NetCDF and nco can be found at

- `http://www.unidata.ucar.edu/packages/netcdf` and
- `http://nco.sourceforge.net`.

7.1 Module output — saving the results

INTERFACE:

```
  module output
```

DESCRIPTION:

This module acts as an interface between GOTM and modules/routines doing the actual output. In order to add a new output format it is only necessary to add hooks in this module and write the actual output routines. It is not necessary to change anything in GOTM itself.

USES:

```
   use time, ONLY: write_time_string,julianday,secondsofday,timestep
   use asciiout
 ifdef NETCDF_FMT
   use ncdfout, ONLY:  init_ncdf,do_ncdf_out,close_ncdf
 endif

   IMPLICIT NONE
```

PUBLIC DATA MEMBERS:

```
   logical                              :: write_results
   integer, public, parameter           :: ascii_unit=50
   integer, public, parameter           :: grads_unit=51
   character(len=19)                    :: ts
   integer                              :: out_fmt
   character(len=PATH_MAX)              :: out_dir
   character(len=PATH_MAX)              :: out_fn
   integer                              :: nsave
   logical                              :: diagnostics
   integer                              :: mld_method
   REALTYPE                             :: diff_k
   REALTYPE                             :: Ri_crit
   logical                              :: rad_corr
   logical                              :: init_diagnostics
```

REVISION HISTORY:

```
   Original author(s): Karsten Bolding, Hans Burchard
```

7.1.1 Initialize the output module

INTERFACE:

```
   subroutine init_output(title,nlev,latitude,longitude)
```

DESCRIPTION:

Calls the initialization routine based on output format selected by the user.

USES:

```
IMPLICIT NONE
```

INPUT/OUTPUT PARAMETERS:

```
character(len=*), intent(in)        :: title
integer, intent(in)                 :: nlev
REALTYPE, intent(in)                :: latitude,longitude
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding & Hans Burchard
```

7.1.2 Set some variables related to output

INTERFACE:

```
subroutine prepare_output(n)
```

DESCRIPTION:

This routine check whether output should be written at the current time step. If this is the case, the model time is written to a string for display on the screen.

USES:

```
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer(kind=8), intent(in)         :: n
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding & Hans Burchard
```

7.1.3 Save the model results in file

INTERFACE:

```
subroutine do_output(n,nlev)
```

DESCRIPTION:

Calls the routine, which will do the actual storing of results, depending on the output format.

USES:

```
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer(kind=8), intent(in)        :: n
integer, intent(in)                :: nlev
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding & Hans Burchard
```

LOCAL VARIABLES:

```
REALTYPE                  :: secs
```

7.1.4 Close files used for saving model results

INTERFACE:

```
subroutine close_output()
```

DESCRIPTION:

Call routines for closing any open output files.

USES:

```
IMPLICIT NONE
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding & Hans Burchard
```

7.1.5 Compute various diagnostic/integrated variables

INTERFACE:

```
subroutine do_diagnostics(n,nlev,BuoyMeth,dt,u_taus,u_taub,I_0,heat)
```

DESCRIPTION:

This subroutine calculates the following diagnostic/integrated variables.

USES:

```
use airsea,       only: sst
use meanflow,     only: gravity,rho_0,cp
use meanflow,     only: h,u,v,s,t,NN,SS,buoy,rad
use turbulence,   only: kappa
use turbulence,   only: tke
use observations, only: tprof,b_obs_sbf
use eqstate,      only: eqstate1
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer(kind=8), intent(in)          :: n
integer, intent(in)                  :: nlev,BuoyMeth
REALTYPE, intent(in)                 :: dt
REALTYPE, intent(in)                 :: u_taus,u_taub
REALTYPE, intent(in)                 :: I_0,heat
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding & Hans Burchard
```

7.2 Module asciiout — saving the results in ASCII

INTERFACE:

```
MODULE asciiout
```

DESCRIPTION:

This module contains three subroutines for writing model output in ASCII format. The authors do not encourage using ASCII for output — instead we recommend NetCDF.

USES:

```
IMPLICIT NONE
Default all is private.
private
```

PUBLIC MEMBER FUNCTIONS:

```
public init_ascii, do_ascii_out, close_ascii

integer :: set
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding & Hans Burchard
```

7.2.1 Open the file unit for writing

INTERFACE:

```
subroutine init_ascii(fn,title,unit)
IMPLICIT NONE
```

DESCRIPTION:

Opens a file giving in the `output` namelist and connects it with a unit number.

INPUT PARAMETERS:

```
character(len=*), intent(in)        :: fn,title
```

INPUT/OUTPUT PARAMETERS:

```
integer, intent(in)                 :: unit
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding & Hans Burchard
See asciiout module
```

7.2.2 Save the model results to file

INTERFACE:

```
subroutine do_ascii_out(nlev,timestr,unit)
```

DESCRIPTION:

Writes all calculated data to an ASCII file.

USES:

```
  use meanflow,     only: depth0,h,u,v,z,S,T,NN,buoy
  use turbulence,   only: num,nuh,tke,eps,L
  use turbulence,   only: kb,epsb
  use observations, only: tprof,sprof,uprof,vprof,epsprof
#ifdef SEDIMENT
   use sediment, only: ascii_sediment
#endif
#ifdef SEDIMENT
   use seagrass, only: ascii_seagrass
#endif

  IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer, intent(in)                :: nlev
CHARACTER(len=*), intent(in)       :: timestr
integer, intent(in)                :: unit
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding & Hans Burchard
See asciiout module
```

7.2.3 Close files used for saving model results

INTERFACE:

```
subroutine close_ascii(unit)
IMPLICIT NONE
```

DESCRIPTION:

Close the open ASCII file.

INPUT PARAMETERS:

```
integer, intent(in)                :: unit
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding & Hans Burchard
See asciiout module
```

7.3 Module ncdfout — saving the results in NetCDF

INTERFACE:

```
module ncdfout
```

DESCRIPTION:

This module provides routines for saving the GOTM results using NetCDF format. A hack has been provided for saving in a way that can be used by the GrADS graphics software. The `sdfopen()` interface to GrADS does not allow for smaller time units than 1 hour, so if GrADS output is selected the units for time are set to `hours` and not `secs`.
In both cases, the type and number of variables appearing in the output file depends on the turbulence model and the output flags set by the user. If you use, for example, the KPP turbulence module no information for the TKE, the dissipation rate, the turbulence production terms are saved, because the KPP model does not provide information about these quantities.
Note that if you `#define EXTRA_OUTPUT` in `cppdef.h`, then you will find the a number of dummy fields called `mean1, mean2, ...` and `turb1, turb2, ...` in the netCDF output file after re-compiling and runnign GOTM. These extra variables are public members of the `meanflow` and `turbulence` modules and are convenient for testing and debuging.

USES:

```
use turbulence, only: turb_method
use netcdf
IMPLICIT NONE
```

PUBLIC MEMBER FUNCTIONS:

```
public init_ncdf, do_ncdf_out, close_ncdf
public define_mode, new_nc_variable, set_attributes, store_data
```

PUBLIC DATA MEMBERS:

```
netCDF file id
integer, public                    :: ncid

dimension ids
integer                            :: lon_dim,lat_dim,z_dim,z1_dim
integer                            :: time_dim
integer                            :: dim1d(1)
integer                            :: dim3d(3)
integer                            :: dim4d(4)
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding & Hans Burchard
```

7.3.1 Create the NetCDF file

INTERFACE:

```
subroutine init_ncdf(fn,title,lat,lon,nlev,start_time,time_unit)
IMPLICIT NONE
```

DESCRIPTION:

Opens and creates the NetCDF file, and initialises all dimensions and variables for the core GOTM model.

INPUT PARAMETERS:

```
character(len=*), intent(in)        :: fn,title,start_time
REALTYPE, intent(in)                :: lat,lon
integer, intent(in)                 :: nlev,time_unit
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding & Hans Burchard
```

7.3.2 Save model results to file

INTERFACE:

```
subroutine do_ncdf_out(nlev,secs)
```

DESCRIPTION:

Write the GOTM core variables to the NetCDF file.

USES:

```
 use airsea,       only: tx,ty,I_0,heat,precip,evap,sst,sss
 use airsea,       only: int_precip,int_evap,int_fwf
 use airsea,       only: int_swr,int_heat,int_total
 use meanflow,     only: depth0,u_taub,u_taus,rho_0,gravity
 use meanflow,     only: h,u,v,z,S,T,buoy,SS,NN
 use turbulence,   only: P,B,Pb
 use turbulence,   only: num,nuh,nus
 use turbulence,   only: gamu,gamv,gamh,gams
 use turbulence,   only: tke,kb,eps,epsb,L,uu,vv,ww
 use kpp,          only: zsbl,zbbl
 use observations, only: zeta,uprof,vprof,tprof,sprof,epsprof,o2_prof
 use eqstate,      only: eqstate1
ifdef EXTRA_OUTPUT
 use meanflow,     only: mean1,mean2,mean3,mean4,mean5
 use turbulence,   only: turb1,turb2,turb3,turb4,turb5
endif
 IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer,  intent(in)                :: nlev
REALTYPE, intent(in)                :: secs
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding & Hans Burchard
```

7.3.3 Close files used for saving model results

INTERFACE:

```
subroutine close_ncdf()
IMPLICIT NONE
```

DESCRIPTION:

Closes the NetCDF file.

REVISION HISTORY:

```
Original author(s): Karsten Bolding & Hans Burchard
```

7.3.4 Begin or end define mode

INTERFACE:

```
integer function define_mode(ncid,action)
```

DESCRIPTION:

Depending on the value of the argument `action`, this routine put NetCDF in the 'define' mode or not.

USES:

```
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer, intent(in)        :: ncid
logical, intent(in)        :: action
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding & Hans Burchard
```

7.3.5 Define a new NetCDF variable

INTERFACE:

```
   integer function new_nc_variable(ncid,name,data_type,dims,id)
```

DESCRIPTION:

This routine is used to define a new variable to store in a NetCDF file.

USES:

```
   IMPLICIT NONE
```

INPUT PARAMETERS:

```
   integer, intent(in)                :: ncid
   character(len=*), intent(in)       :: name
   integer, intent(in)                :: data_type
   integer, intent(in)                :: dims(:)
```

OUTPUT PARAMETERS:

```
   integer, intent(out)               :: id
```

REVISION HISTORY:

```
   Original author(s): Karsten Bolding & Hans Burchard
```

7.3.6 Set attributes for a NetCDF variable.

INTERFACE:

```
   integer function set_attributes(ncid,id,                          &
                                  units,long_name,                  &
                                  valid_min,valid_max,valid_range, &
                                  scale_factor,add_offset,         &
                                  FillValue,missing_value,         &
                                  C_format,FORTRAN_format)
```

DESCRIPTION:

This routine is used to set a number of attributes for variables. The routine makes heavy use of the `optional` keyword. The list of recognized keywords is very easy to extend. We have included a sub-set of the COARDS conventions.

USES:

```
   IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer, intent(in)                :: ncid,id
character(len=*), optional         :: units,long_name
REALTYPE, optional                 :: valid_min,valid_max
REALTYPE, optional                 :: valid_range(2)
REALTYPE, optional                 :: scale_factor,add_offset
REALTYPE, optional                 :: FillValue,missing_value
character(len=*), optional         :: C_format,FORTRAN_format
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding & Hans Burchard
```

LOCAL VARIABLES:

```
integer                   :: len,iret
REAL_4B                   :: vals(2)
```

7.3.7 Store values in a NetCDF file

INTERFACE:

```
integer function store_data(ncid,id,var_shape,nlev, &
                            iscalar,iarray,scalar,array)
```

DESCRIPTION:

This routine is used to store a variable in the NetCDF file. The subroutine uses `optional` parameters to find out which data type to save.

USES:

```
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer, intent(in)                :: ncid,id,var_shape,nlev
integer, optional                  :: iscalar
integer, optional                  :: iarray(0:nlev)
REALTYPE, optional                 :: scalar
REALTYPE, optional                 :: array(0:nlev)
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding & Hans Burchard
```

8 Utilities

8.1 Introduction

In this section, different utility modules and routines are assembled, such as the `time` module (see `time.F90`), keeping track of all time calculations, the `mtridiagonal` module with a Gaussian solver for systems of equations with tri-diagonal matrices (see `tridiagonal.F90`), and the `eqstate` module (see `eqstate.F90`) with different versions of the equation of state.
Also discussed are advection and diffusion routines, such as `diff_center()` and `adv_center()` for variables located at the centers of the grid cells, i.e. in general mean flow variables.

8.2 Module util — parameters and interfaces for utilities

INTERFACE:

```
MODULE util
```

DESCRIPTION:

This module is an encapsulation of a number of parameters used by different routines found in the `util` directory. It should make it easier to read the code, since finding a line like

```
if (method.eq.UPSTREAM) then ...
```

in a subroutine for advection methods tells you more than reading only

```
if (method.eq.1) then ...
```

USES:

```
IMPLICIT NONE
```

DEFINED PARAMETERS:

```
type of advection scheme
integer,parameter                   :: UPSTREAM       = 1
integer,parameter                   :: P1             = 2
integer,parameter                   :: P2             = 3
integer,parameter                   :: Superbee       = 4
integer,parameter                   :: MUSCL          = 5
integer,parameter                   :: P2_PDM         = 6

boundary condition type
for diffusion scheme
integer,parameter                   :: Dirichlet      = 0
integer,parameter                   :: Neumann        = 1

boundary condition type
for advection schemes
integer,parameter                   :: flux           = 1
integer,parameter                   :: value          = 2
integer,parameter                   :: oneSided       = 3
integer,parameter                   :: zeroDivergence = 4
```

REVISION HISTORY:

```
Original author(s): Lars Umlauf
```

8.3 Diffusion schemes — grid centers

INTERFACE:

```
subroutine diff_center(N,dt,cnpar,posconc,h,Bcup,Bcdw, &
                       Yup,Ydw,nuY,Lsour,Qsour,Taur,Yobs,Y)
```

DESCRIPTION:

This subroutine solves the one-dimensional diffusion equation including source terms,

$$\frac{\partial Y}{\partial t} = \frac{\partial}{\partial z}\left(\nu_Y \frac{\partial Y}{\partial z}\right) - \frac{1}{\tau_R}(Y - Y_{obs}) + Y L_{\text{sour}} + Q_{\text{sour}} \ , \tag{233}$$

for al variables defined at the centers of the grid cells, and a diffusion coefficient ν_Y defined at the faces. Relaxation with time scale τ_R towards observed values Y_{obs} is possible. L_{sour} specifies a linear source term, and Q_{sour} a constant source term. Central differences are used to discretize the problem as discussed in section 3.1.2. The diffusion term, the linear source term, and the linear part arising from the relaxation term are treated with an implicit method, whereas the constant source term is treated fully explicit.
The input parameters `Bcup` and `Bcdw` specify the type of the upper and lower boundary conditions, which can be either Dirichlet or Neumann-type. `Bcup` and `Bcdw` must have integer values corresponding to the parameters `Dirichlet` and `Neumann` defined in the module `util`, see section 8.2. `Yup` and `Ydw` are the values of the boundary conditions at the surface and the bottom. Depending on the values of `Bcup` and `Bcdw`, they represent either fluxes or prescribed values. The integer `posconc` indicates if a quantity is non-negative by definition (`posconc`=1, such as for concentrations) or not (`posconc`=0). For `posconc`=1 and negative boundary fluxes, the source term linearisation according to *Patankar* (1980) is applied.
Note that fluxes *entering* a boundary cell are counted positive by convention. The lower and upper position for prescribing these fluxes are located at the lowest und uppermost grid faces with index "0" and index "N", respectively. If values are prescribed, they are located at the centers with index "1" and index "N", respectively.

USES:

```
use util,          only  : Dirichlet, Neumann
use mtridiagonal

IMPLICIT NONE
```

INPUT PARAMETERS:

```
number of vertical layers
integer,  intent(in)              :: N

time step (s)
REALTYPE, intent(in)              :: dt

"implicitness" parameter
REALTYPE, intent(in)              :: cnpar
```

```
1: non-negative concentration, 0: else
integer, intent(in)                :: posconc

layer thickness (m)
REALTYPE, intent(in)               :: h(0:N)

type of upper BC
integer,  intent(in)               :: Bcup

type of lower BC
integer,  intent(in)               :: Bcdw

value of upper BC
REALTYPE, intent(in)               :: Yup

value of lower BC
REALTYPE, intent(in)               :: Ydw

diffusivity of Y
REALTYPE, intent(in)               :: nuY(0:N)

linear source term
(treated implicitly)
REALTYPE, intent(in)               :: Lsour(0:N)

constant source term
(treated explicitly)
REALTYPE, intent(in)               :: Qsour(0:N)

relaxation time (s)
REALTYPE, intent(in)               :: Taur(0:N)

observed value of Y
REALTYPE, intent(in)               :: Yobs(0:N)
```

INPUT/OUTPUT PARAMETERS:

```
REALTYPE                           :: Y(0:N)
```

REVISION HISTORY:

```
Original author(s): Lars Umlauf
```

8.4 Diffusion schemes — grid faces

INTERFACE:

```
   subroutine diff_face(N,dt,cnpar,h,Bcup,Bcdw,Yup,Ydw,nuY,Lsour,Qsour,Y)
```

DESCRIPTION:

USES:

```
   use util,          only  : Dirichlet, Neumann
   use mtridiagonal

   IMPLICIT NONE
```

INPUT PARAMETERS:

```
   number of vertical layers
   integer,  intent(in)                :: N

   time step (s)
   REALTYPE, intent(in)                :: dt

   "implicitness" parameter
   REALTYPE, intent(in)                :: cnpar

   layer thickness (m)
   REALTYPE, intent(in)                :: h(0:N)

   type of upper BC
   integer,  intent(in)                :: Bcup

   type of lower BC
   integer,  intent(in)                :: Bcdw

   value of upper BC
   REALTYPE, intent(in)                :: Yup

   value of lower BC
   REALTYPE, intent(in)                :: Ydw

   diffusivity of Y
   REALTYPE, intent(in)                :: nuY(0:N)

   linear source term
   (treated implicitly)
   REALTYPE, intent(in)                :: Lsour(0:N)

   constant source term
   (treated explicitly)
```

```
REALTYPE, intent(in)                :: Qsour(0:N)
```

INPUT/OUTPUT PARAMETERS:

```
REALTYPE                            :: Y(0:N)
```

REVISION HISTORY:

```
Original author(s): Lars Umlauf
```

8.5 Advection schemes — grid centers

INTERFACE:

```
subroutine adv_center(N,dt,h,ho,ww,Bcup,Bcdw,Yup,Ydw,method,mode,Y)
```

DESCRIPTION:

This subroutine solves a one-dimensional advection equation. There are two options, depending whether the advection should be conservative or not. Conservative advection has to be applied when settling of sediment or rising of phytoplankton is considered. In this case the advection is of the form

$$\frac{\partial Y}{\partial t} = -\frac{\partial F}{\partial z} \ , \tag{234}$$

where $F = wY$ is the flux caused by the advective velocity, w.
Non-conservative advective transport has to be applied, when the water has a non-zero vertical velocity. In three-dimensional applications, this transport would be conservative, since vertical divergence would be compensated by horizontal convergence and vice versa. However, the key assumption of one-dimensional modelling is horizontal homogeneity, such that we indeed have to apply a vertically non-conservative method, which is of the form

$$\frac{\partial Y}{\partial t} = -w\frac{\partial Y}{\partial z} = -\left(\frac{\partial F}{\partial z} - Y\frac{\partial w}{\partial z}\right) . \tag{235}$$

The discretized form of (234) is

$$Y_i^{n+1} = Y_i^n - \frac{\Delta t}{h_i}\left(F_i^n - F_{i-1}^n\right) \ , \tag{236}$$

where the integers n and i correspond to the present time and space level, respectively.
For the non-conservative form (235), an extra term needs to be included:

$$Y_i^{n+1} = Y_i^n - \frac{\Delta t}{h_i}\left(F_i^n - F_{i-1}^n - Y_i^n\left(w_k - w_{k-1}\right)\right) . \tag{237}$$

Which advection method is applied is decided by the flag `mode`, which gives conservative advection (236) for `mode=1` and non-conservative advection (237) for `mode=0`.
Fluxes are defined at the grid faces, the variable Y_i is defined at the grid centers. The fluxes are computed in an upstream-biased way,

$$F_i^n = \frac{1}{\Delta t}\int_{z_i^{\text{Face}}-w\Delta t}^{z_i^{\text{Face}}} Y(z')dz' \quad . \tag{238}$$

For a third-order polynomial approximation of Y (see *Pietrzak* (1998)), these fluxes can be written the in so-called Lax-Wendroff form as

$$\begin{aligned} F_i &= w_i\left(Y_i + \frac{1}{2}\Phi_i^+\left(1-|c_i|\right)\left(Y_{i+1}-Y_i\right)\right) \qquad \text{for} \quad w_i > 0 \ , \\ F_i &= w_i\left(Y_{i+1} + \frac{1}{2}\Phi_i^-\left(1-|c_i|\right)\left(Y_i-Y_{i+1}\right)\right) \qquad \text{for} \quad w_i < 0 \ , \end{aligned} \tag{239}$$

where $c_i = 2w_i\Delta t/(h_i + h_{i+1})$ is the Courant number. The factors appearing in (239) are defined as

$$\Phi_i^+ = \alpha_i + \beta_i r_i^+ \ , \quad \Phi_i^- = \alpha_i + \beta_i r_i^- \ , \tag{240}$$

where

$$\alpha_i = \frac{1}{2} + \frac{1}{6}\left(1 - 2|c_i|\right) \ , \quad \beta_i = \frac{1}{2} - \frac{1}{6}\left(1 - 2|c_i|\right) \quad . \tag{241}$$

The upstream and downstream slope parameters are

$$r_i^+ = \frac{Y_i - Y_{i-1}}{Y_{i+1} - Y_i} \ , \quad r_i^- = \frac{Y_{i+2} - Y_{i+1}}{Y_{i+1} - Y_i} \quad . \tag{242}$$

To obtain monotonic and positive schemes also in the presence of strong gradients, so-called slope limiters are aplied for the factors Φ_i^+ and Φ_i^-. The two most obvious cases are the first-order upstream discretisation with $\Phi_i^+ = \Phi_i^- = 0$ and the Lax-Wendroff scheme with $\Phi_i^+ = \Phi_i^- = 1$. The subroutine `adv_center.F90` provides six different slope-limiters, all discussed in detail by *Pietrzak* (1998):

- first-order upstream (`method=UPSTREAM`)
- second-order upstream-biased polynomial scheme (`method=P1`, not yet implemented)
- third-order upstream-biased polynomial scheme (`method=P2`)
- third-order scheme (TVD) with Superbee limiter (`method=Superbee`)
- third-order scheme (TVD) with MUSCL limiter (`method=MUSCL`)
- third-order scheme (TVD) with ULTIMATE QUICKEST limiter (`method=P2_PDM`)

If during a certain time step the maximum Courant number is larger than one, a split iteration will be carried out which guarantees that the split step Courant numbers are just smaller than 1. Several kinds of boundary conditions are implemented for the upper and lower boundaries. They are set by the integer values `Bcup` and `Bcdw`, that have to correspond to the parameters defined in the module `util`, see section 8.2. The following choices exist at the moment:
For the value `flux`, the boundary values `Yup` and `Ydw` are interpreted as specified fluxes at the uppermost and lowest interface. Fluxes into the boundary cells are counted positive by convention. For the value `value`, `Yup` and `Ydw` specify the value of Y at the interfaces, and the flux is computed by multiplying with the (known) speed at the interface. For the value `oneSided`, `Yup` and `Ydw` are ignored and the flux is computed from a one-sided first-order upstream discretisation using the speed at the interface and the value of Y at the center of the boundary cell. For the value `zeroDivergence`, the fluxes into and out of the respective boundary cell are set equal. This corresponds to a zero-gradient formulation, or to zero flux divergence in the boundary cells.
Be careful that your boundary conditions are mathematically well defined. For example, specifying an inflow into the boundary cell with the speed at the boundary being directed outward does not make sense.

USES:

```
use util
IMPLICIT NONE
```

INPUT PARAMETERS:

```
number of vertical layers
integer,  intent(in)                :: N

time step (s)
REALTYPE, intent(in)                :: dt

layer thickness (m)
REALTYPE, intent(in)                :: h(0:N)

old layer thickness (m)
REALTYPE, intent(in)                :: ho(0:N)

vertical advection speed
REALTYPE, intent(in)                :: ww(0:N)

type of upper BC
integer,  intent(in)                :: Bcup

type of lower BC
integer,  intent(in)                :: Bcdw

value of upper BC
REALTYPE, intent(in)                :: Yup

value of lower BC
REALTYPE, intent(in)                :: Ydw

type of advection scheme
integer,  intent(in)                :: method

advection mode (0: non-conservative, 1: conservative)
integer,  intent(in)                :: mode
```

INPUT/OUTPUT PARAMETERS:

```
REALTYPE                            :: Y(0:N)
```

DEFINED PARAMETERS:

```
REALTYPE,     parameter             :: one6th=1.0d0/6.0d0
integer,      parameter             :: itmax=100
```

REVISION HISTORY:

```
Original author(s): Lars Umlauf
```

8.6 Lagrangian particle random walk

INTERFACE:

```
subroutine lagrange(nlev,dt,zlev,nuh,w,npar,active,zi,zp)
```

DESCRIPTION:

Here a Lagrangian particle random walk for spatially inhomogeneous turbulence according to *Visser* (1997) is implemented. With the random walk, the particle i is moved from the vertical position z_i^n to z_i^{n+1} according to the following algorithm:

$$\begin{aligned} z_i^{n+1} &= z_i^n + \partial_z \nu_t(z_i^n)\Delta t \\ &+ R\left\{2r^{-1}\nu_t(z_i^n + \tfrac{1}{2}\partial_z\nu_t(z_i^n)\Delta t)\Delta t\right\}^{1/2}, \end{aligned} \tag{243}$$

where R is a random process with $\langle R\rangle = 0$ (zero mean) and and the variance $\langle R^2\rangle = r$. Set `visc_corr=.true.` for evaluating eddy viscosity in a semi-implicit way. A background viscosity (`visc_back`) may be set. The variance r of the random walk scheme (`rnd_var`) has to be set manually as well here.

USES:

```
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer, intent(in)                :: nlev
REALTYPE, intent(in)               :: dt
REALTYPE, intent(in)               :: zlev(0:nlev)
REALTYPE, intent(in)               :: nuh(0:nlev)
REALTYPE, intent(in)               :: w
integer, intent(in)                :: npar
logical, intent(in)                :: active(npar)
```

INPUT/OUTPUT PARAMETERS:

```
integer, intent(inout)             :: zi(npar)
REALTYPE, intent(inout)            :: zp(npar)
```

REVISION HISTORY:

```
Original author(s): Hans Burchard & Karsten Bolding
```

LOCAL VARIABLES:

```
integer            :: i,n
REALTYPE           :: rnd(npar),rnd_var_inv
REALTYPE,parameter :: visc_back=0.e-6,rnd_var=0.333333333
REALTYPE           :: depth,dz(nlev),dzn(nlev),step,zp_old
REALTYPE           :: visc,rat,dt_inv,zloc
logical,parameter  :: visc_corr=.false.
```

8.7 Module mtridiagonal — solving the system

INTERFACE:

```
MODULE mtridiagonal
```

DESCRIPTION:

Solves a linear system of equations with a tridiagonal matrix using Gaussian elimination.

PUBLIC MEMBER FUNCTIONS:

```
public init_tridiagonal, tridiagonal, clean_tridiagonal
```

PUBLIC DATA MEMBERS:

```
REALTYPE, dimension(:), allocatable     :: au,bu,cu,du
```

REVISION HISTORY:

```
Original author(s): Hans Burchard & Karsten Bolding
```

8.7.1 Allocate memory

INTERFACE:

```
subroutine init_tridiagonal(N)
```

DESCRIPTION:

This routines allocates memory necessary to perform the Gaussian elimination.

USES:

```
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer, intent(in)                 :: N
```

REVISION HISTORY:

```
Original author(s): Hans Burchard & Karsten Bolding
```

8.7.2 Simplified Gaussian elimination

INTERFACE:

```
subroutine tridiagonal(N,fi,lt,value)
```

DESCRIPTION:

A linear equation with tridiagonal matrix structure is solved here. The main diagonal is stored on `bu`, the upper diagonal on `au`, and the lower diagonal on `cu`, the right hand side is stored on `du`. The method used here is the simplified Gauss elimination, also called *Thomas algorithm*.

USES:

```
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer, intent(in)                 :: N,fi,lt
```

OUTPUT PARAMETERS:

```
REALTYPE                            :: value(0:N)
```

REVISION HISTORY:

```
Original author(s): Hans Burchard & Karsten Bolding
```

8.7.3 De-allocate memory

INTERFACE:

```
subroutine clean_tridiagonal()
```

DESCRIPTION:

De-allocates memory allocated in init_tridiagonal.

USES:

```
IMPLICIT NONE
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding
```

8.8 Module eqstate — the equation of state

INTERFACE:

```
MODULE eqstate
```

DESCRIPTION:

Computes the density, $\langle\rho\rangle$, and buoyancy from the salinity, S, the temperature, Θ, and the thermodynamic pressure, P, according to an *equation of state*,

$$\langle\rho\rangle = \hat{\rho}(S, \Theta, P) \quad . \tag{244}$$

The following remark on the thermodynamic interpretation of density, temperature, and pressure is useful here. If Θ is identified with the in-situ temperature, and P with the in-situ pressure, then $\langle\rho\rangle$ will be the in-situ density. On the other hand, if P is identified with the surface pressure, and Θ with the potential temperature, the same equation of state, (244), will yield $\langle\rho\rangle$ as the potential density. Note that the quantity `sigma_t` found in the GOTM output is simply computed from $\langle\rho\rangle$ - 1000 kg m^{-3}, and may therefore adopt different meanings.
At present, two different models for the equation of state (”modes”), and four different ”methods” how to evalute the equation of state are implemented.
Modes:

1. The UNESCO equation of state according to *Fofonoff and Millard* (1983)
2. The *Jackett et al.* (2005) equation of state

Methods:

1. the full equation of state — including pressure effects
2. the full equation of state — without pressure effects
3. the linearised equation of state
4. a general linear form of the equation of state

USES:

```
IMPLICIT NONE

default: all is private.
private
```

PUBLIC MEMBER FUNCTIONS:

```
public init_eqstate,eqstate1,eos_alpha,eos_beta,unesco,rho_feistel
```

REVISION HISTORY:

```
Original author(s): Hans Burchard & Karsten Bolding
```

8.8.1 Read the namelist `eqstate`

INTERFACE:

```
subroutine init_eqstate(namlst)
```

DESCRIPTION:

Here, the namelist `eqstate` in the namelist file `gotmrun.nml` is read.

USES:

```
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer, optional, intent(in)       :: namlst
```

REVISION HISTORY:

```
Original author(s): Hans Burchard & Karsten Bolding
```

8.8.2 Select an equation of state

INTERFACE:

```
REALTYPE function eqstate1(S,T,p,g,rho_0)
```

DESCRIPTION:

Calculates the in-situ buoyancy according to the selected method. `S` is salinity S in psu, `T` is potential temperature θ in °C (ITS-90), `p` is gauge pressure (absolute pressure - 10.1325 bar), `g` is the gravitational acceleration in $\mathrm{m\,s^{-2}}$ and `rho_0` the reference density in $\mathrm{kg\,m^{-3}}$. `eqstate1` is the in-situ-density in $\mathrm{kg\,m^{-3}}$. For `eq_state_method`=1, the UNESCO equation of state is used, for `eq_state_method`=2, the *Jackett et al.* (2005) equation of state is used. Here, some care is needed, since the UNESCO equation used bar for pressure and the *Jackett et al.* (2005) uses dbar for pressure. For values of `eq_state_method` ranging from 1 to 4, one of the following methods will be used.

1. the full equation of state for sea water including pressure dependence.
2. the equation of state for sea water with the pressure evaluated at the sea surface as reference level. This is the choice for computations based on potential temperature and density.
3. a linearised equation of state. The parameters `T0`, `S0` and `p0` have to be specified in the namelist.
4. a linear equation of state with prescribed `rho0`, `T0`, `S0`, `dtr0`, `dsr0` according to

$$\rho = \rho_0 + \mathtt{dtr0}(T - T_0) + \mathtt{dsr0}(S - S_0) \quad . \tag{245}$$

USES:

```
IMPLICIT NONE
```

INPUT PARAMETERS:

```
REALTYPE,intent(in)                 :: S,T,p
REALTYPE,optional,intent(in)        :: g,rho_0
```

REVISION HISTORY:

```
Original author(s): Hans Burchard & Karsten Bolding
```

8.8.3 Compute thermal expansion coefficient

INTERFACE:

```
REALTYPE function eos_alpha(S,T,p,g,rho_0)
```

DESCRIPTION:

Computes the thermal expansion coefficient defined by

$$\alpha = -\frac{1}{\rho_0}\left(\frac{\partial \rho_{is}}{\partial T}\right)_S = \frac{1}{g}\left(\frac{\partial B_{is}}{\partial T}\right)_S \quad , \tag{246}$$

where B_{is} denotes the in-situ buoyancy. The computation is carried out by a finite difference approximation of (246), requiring two evaluations of the equation of state. Note, that comparing (246) with (245) it follows that $\alpha = -\texttt{dtr0}/\rho_0$.

USES:

```
IMPLICIT NONE
```

INPUT PARAMETERS:

```
REALTYPE,intent(in)                 :: S,T,p
REALTYPE,optional,intent(in)        :: g,rho_0
```

REVISION HISTORY:

```
Original author(s): Lars Umlauf
```

8.8.4 Compute saline contraction coefficient

INTERFACE:

```
REALTYPE function eos_beta(S,T,p,g,rho_0)
```

DESCRIPTION:

Computes the saline contractioncoefficient defined by

$$\beta = \frac{1}{\rho_0} \left(\frac{\partial \rho_{is}}{\partial S} \right)_T = -\frac{1}{g} \left(\frac{\partial B_{is}}{\partial S} \right)_T \ , \tag{247}$$

where B_{is} denotes the in-situ buoyancy. The computation is carried out by a finite difference approximation of (247), requiring two evaluations of the equation of state. Note, that comparing (247) with (245) it follows that $\beta = \texttt{dsr0}/\rho_0$.

USES:

```
IMPLICIT NONE
```

INPUT PARAMETERS:

```
REALTYPE,intent(in)                :: S,T,p
REALTYPE,optional,intent(in)       :: g,rho_0
```

REVISION HISTORY:

```
Original author(s): Lars Umlauf
```

8.8.5 The UNESCO equation of state

INTERFACE:

```
REALTYPE function unesco(S,T,p,UNPress)
```

DESCRIPTION:

Computes the in-situ density in (244) according to the UNESCO equation of state for sea water (see *Fofonoff and Millard* (1983)). The pressure dependence can be switched on (`UNPress=.true.`) or off (`UNPress=.false.`). `S` is salinity S in psu, `T` is potential temperature θ in °C (ITS-90), `p` is gauge pressure (absolute pressure - 10.1325 bar) and `unesco` is the in-situ density in kg m^{-3}. The check value is `unesco(35,25,1000) = 1062.53817` .

USES:

```
IMPLICIT NONE
```

INPUT PARAMETERS:

```
REALTYPE, intent(in)               :: S,T,p
LOGICAL, intent(in)                :: UNPress
```

REVISION HISTORY:

```
Original author(s): Hans Burchard & Karsten Bolding
```

8.8.6 The *Jackett et al.* (2005) equation of state

INTERFACE:

```
REALTYPE function rho_feistel(s,th,p,UNPress)
```

DESCRIPTION:

Computes the in-situ density in (244) according to the *Jackett et al.* (2005) equation of state for sea water, which is based on the Gibbs potential developed by *Feistel* (2003). The pressure dependence can be switched on (`UNPress=.true.`) or off (`UNPress=.false.`). `s` is salinity S in psu, `th` is potential temperature θ in °C (ITS-90), `p` is gauge pressure (absolute pressure - 10.1325 dbar) and `rho_feistel` is the in-situ density in kg m^{-3}. The check value is `rho_feistel(20,20,1000) = 1017.728868019642` .

USES:

```
IMPLICIT NONE
```

INPUT PARAMETERS:

```
REALTYPE, intent(in)                :: s,th,p
LOGICAL, intent(in)                 :: UNPress
```

REVISION HISTORY:

```
Original author(s): Hans Burchard & Karsten Bolding
```

8.9 Interpolate from observation space to model grid

INTERFACE:

```
subroutine gridinterpol(N,cols,obs_z,obs_prof,nlev,model_z,model_prof)
```

DESCRIPTION:

This is a utility subroutine in which observational data, which might be given on an arbitrary, but structured grid, are linearly interpolated and extrapolated to the actual (moving) model grid.

USES:

```
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer,  intent(in)              :: N,cols
REALTYPE, intent(in)              :: obs_z(0:N),obs_prof(0:N,cols)
integer,  intent(in)              :: nlev
REALTYPE, intent(in)              :: model_z(0:nlev)
```

OUTPUT PARAMETERS:

```
REALTYPE, intent(out)             :: model_prof(0:nlev,cols)
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding & Hans Burchard
```

8.10 Convert between buoyancy fluxes and others

INTERFACE:

```
  subroutine  convert_fluxes(nlev,g,cp,rho_0,heat,p_e,rad,T,S,           &
                             tFlux,sFlux,btFlux,bsFlux,tRad,bRad)
```

DESCRIPTION:

This subroutine computes the buoyancy fluxes that are due to

1. the surface heat flux,
2. the surface salinity flux caused by the value of P-E (precipitation-evaporation),
3. and the short wave radiative flux.

Additionally, it outputs the temperature flux (`tFlux`) corresponding to the surface heat flux, the salinity flux (`sFlux`) corresponding to the value P-E, and the profile of the temperature flux (`tRad`) corresponding to the profile of the radiative heat flux.
This function is only called when the KPP turbulence model is used. When you call the KPP routines from another model outside GOTM, you are on your own in computing the fluxes required by the KPP model, because they have to be consistent with the equation of state used in your model.

USES:

```
  use eqstate
  IMPLICIT NONE
```

INPUT PARAMETERS:

```
  integer,  intent(in)              :: nlev
  REALTYPE, intent(in)              :: g,cp,rho_0
  REALTYPE, intent(in)              :: heat,p_e
  REALTYPE, intent(in)              :: rad(0:nlev)
  REALTYPE, intent(in)              :: T(0:nlev)
  REALTYPE, intent(in)              :: S(0:nlev)
```

OUTPUT PARAMETERS:

```
  REALTYPE, intent(out)            ::  tFlux,sFlux
  REALTYPE, intent(out)             :: btFlux,bsFlux
  REALTYPE, intent(out)             :: tRad(0:nlev)
  REALTYPE, intent(out)             :: bRad(0:nlev)
```

REVISION HISTORY:

```
   Original author(s): Lars Umlauf
```

8.11 Module time — keep control of time

INTERFACE:

```
  MODULE time
```

DESCRIPTION:

This module provides a number of routines/functions and variables related to the mode time in GOTM. The basic concept used in this module is that time is expressed as two integers — one is the true Julian day and the other is seconds since midnight. All calculations with time then become very simple operations on integers.

USES:

```
  IMPLICIT NONE
  default: all is private.
  private
```

PUBLIC MEMBER FUNCTIONS:

```
  public                            :: init_time, calendar_date
  public                            :: julian_day, update_time
  public                            :: write_time_string
  public                            :: time_diff
  public                            :: sunrise_sunset
 ifdef _PRINTSTATE_
  public                            :: print_state_time
 endif
```

PUBLIC DATA MEMBERS:

```
  character(len=19), public         :: timestr
  character(len=19), public         :: start
  character(len=19), public         :: stop
  REALTYPE,          public         :: timestep
  REALTYPE,          public         :: fsecs,simtime
  integer,target,    public         :: julianday,secondsofday
  integer,target,    public         :: yearday
  integer,           public         :: timefmt
  integer,           public         :: MinN,MaxN
```

REVISION HISTORY:

```
  Original author(s): Karsten Bolding & Hans Burchard
```

8.11.1 Initialise the time system

INTERFACE:

```
subroutine init_time(MinN,MaxN)
```

DESCRIPTION:

The subroutine `init_time()` initialises the time module by reading a namelist and take actions according to the specifications. On exit from this subroutine the two variables MinN and MaxN have well defined values and can be used in the time loop.

USES:

```
IMPLICIT NONE
```

INPUT/OUTPUT PARAMETERS:

```
integer, intent(inout)    :: MinN,MaxN
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding & Hans Burchard
```

8.11.2 Convert true Julian day to calendar date

INTERFACE:

```
subroutine calendar_date(julian,yyyy,mm,dd)
```

DESCRIPTION:

Converts a Julian day to a calendar date — year, month and day. Based on a similar routine in *Numerical Recipes*.

USES:

```
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer                              :: julian
```

OUTPUT PARAMETERS:

```
integer                              :: yyyy,mm,dd
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding & Hans Burchard
```

8.11.3 Convert a calendar date to true Julian day

INTERFACE:

```
subroutine julian_day(yyyy,mm,dd,julian)
```

DESCRIPTION:

Converts a calendar date to a Julian day. Based on a similar routine in *Numerical Recipes*.

USES:

```
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer                             :: yyyy,mm,dd
```

OUTPUT PARAMETERS:

```
integer                             :: julian
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding & Hans Burchard
```

8.11.4 Keep track of time (Julian days and seconds)

INTERFACE:

```
subroutine update_time(n)
```

DESCRIPTION:

Based on a starting time this routine calculates the actual time in a model integration using the number of time steps, `n`, and the size of the time step, `timestep`. More public variables can be updated here if necessary.

USES:

```
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer(kind=8), intent(in)         :: n
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding & Hans Burchard
```

8.11.5 Convert a time string to Julian day and seconds

INTERFACE:

```
subroutine read_time_string(timestr,jul,secs)
```

DESCRIPTION:

Converts a time string to the true Julian day and seconds of that day. The format of the time string must be: `yyyy-mm-dd hh:hh:ss` .

USES:

```
IMPLICIT NONE
```

INPUT PARAMETERS:

```
character(len=19)                   :: timestr
```

OUTPUT PARAMETERS:

```
integer, intent(out)                :: jul,secs
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding & Hans Burchard
```

8.11.6 Convert Julian day and seconds into a time string

INTERFACE:

```
subroutine write_time_string(jul,secs,timestr)
```

DESCRIPTION:

Formats Julian day and seconds of that day to a nice looking character string.

USES:

```
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer, intent(in)                 :: jul,secs
```

OUTPUT PARAMETERS:

```
character(len=19)                   :: timestr
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding & Hans Burchard
```

8.11.7 Return the time difference in seconds

INTERFACE:

```
REALTYPE FUNCTION time_diff(jul1,secs1,jul2,secs2)
```

DESCRIPTION:

This functions returns the time difference between two dates in seconds. The dates are given as Julian day and seconds of that day.

USES:

```
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer, intent(in)              :: jul1,secs1,jul2,secs2
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding & Hans Burchard
```

8.11.8 Return the times of sunrise and sunset

INTERFACE:

```
subroutine sunrise_sunset(latitude,declination,sunrise,sunset)
```

DESCRIPTION:

This functions returns the time difference between two dates in seconds. The dates are given as Julian day and seconds of that day.

USES:

```
IMPLICIT NONE
```

INPUT PARAMETERS:

```
REALTYPE, intent(in)             :: latitude,declination
```

OUTPUT PARAMETERS:

```
REALTYPE, intent(out)            :: sunrise,sunset
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding & Hans Burchard
```

LOCAL VARIABLES:

```
REALTYPE                   :: omega,hour
```

8.11.9 Print the current state of the time module.

INTERFACE:

```
subroutine print_state_time()
```

DESCRIPTION:

This routine writes the value of all module-level variables to screen.

USES:

```
IMPLICIT NONE
```

REVISION HISTORY:

```
Original author(s): Jorn Bruggeman
```

8.12 Lagrangian particle random walk

INTERFACE:

```
   subroutine lagrange(nlev,dt,zlev,nuh,w,npar,active,zi,zp)
```

DESCRIPTION:

Here a Lagrangian particle random walk for spatially inhomogeneous turbulence according to *Visser* (1997) is implemented. With the random walk, the particle i is moved from the vertical position z_i^n to z_i^{n+1} according to the following algorithm:

$$\begin{aligned} z_i^{n+1} &= z_i^n + \partial_z \nu_t(z_i^n)\Delta t \\ &+ R\left\{2r^{-1}\nu_t(z_i^n + \tfrac{1}{2}\partial_z \nu_t(z_i^n)\Delta t)\Delta t\right\}^{1/2}, \end{aligned} \tag{248}$$

where R is a random process with $\langle R \rangle = 0$ (zero mean) and and the variance $\langle R^2 \rangle = r$. Set `visc_corr=.true.` for evaluating eddy viscosity in a semi-implicit way. A background viscosity (`visc_back`) may be set. The variance r of the random walk scheme (`rnd_var`) has to be set manually as well here.

USES:

```
   IMPLICIT NONE
```

INPUT PARAMETERS:

```
   integer, intent(in)                  :: nlev
   REALTYPE, intent(in)                 :: dt
   REALTYPE, intent(in)                 :: zlev(0:nlev)
   REALTYPE, intent(in)                 :: nuh(0:nlev)
   REALTYPE, intent(in)                 :: w
   integer, intent(in)                  :: npar
   logical, intent(in)                  :: active(npar)
```

INPUT/OUTPUT PARAMETERS:

```
   integer, intent(inout)               :: zi(npar)
   REALTYPE, intent(inout)              :: zp(npar)
```

REVISION HISTORY:

```
   Original author(s): Hans Burchard & Karsten Bolding
```

LOCAL VARIABLES:

```
   integer            :: i,n
   REALTYPE           :: rnd(npar),rnd_var_inv
   REALTYPE,parameter :: visc_back=0.e-6,rnd_var=0.333333333
   REALTYPE           :: depth,dz(nlev),dzn(nlev),step,zp_old
   REALTYPE           :: visc,rat,dt_inv,zloc
   logical,parameter  :: visc_corr=.false.
```

8.13 General ODE solver

INTERFACE:

```
subroutine ode_solver(solver,numc,nlev,dt,cc,right_hand_side_rhs,right_hand_side_ppdd)
```

DESCRIPTION:

Here, 10 different numerical solvers for the right hand sides of the biogeochemical models are implemented for computing the ordinary differential equations (ODEs) which are calculated as the second step of the operational split method for the complete biogeochemical models. The remaining ODE is

$$\partial_t c_i = P_i(\vec{c}) - D_i(\vec{c}), \quad i = 1, \ldots, I, \tag{249}$$

with c_i denoting the concentrations of state variables. The right hand side denotes the reaction terms, which are composed of contributions $d_{i,j}(\vec{c})$, which represent reactive fluxes from c_i to c_j, and in turn, $p_{i,j}(\vec{c})$ are reactive fluxes from c_j received by c_i, see equation (267).
These methods are:

1. First-order explicit (not unconditionally positive)
2. Second order explicit Runge-Kutta (not unconditionally positive)
3. Fourth-order explicit Runge-Kutta (not unconditionally positive)
4. First-order Patankar (not conservative)
5. Second-order Patankar-Runge-Kutta (not conservative)
6. Fourth-order Patankar-Runge-Kutta (does not work, not conservative)
7. First-order Modified Patankar (conservative and positive)
8. Second-order Modified Patankar-Runge-Kutta (conservative and positive)
9. Fourth-order Modified Patankar-Runge-Kutta (does not work, conservative and positive)
10. First-order Extended Modified Patankar (stoichiometrically conservative and positive)
11. Second-order Extended Modified Patankar-Runge-Kutta (stoichiometrically conservative and positive)

The schemes 1 - 5 and 7 - 8 have been described in detail by *Burchard et al.* (2003). Later, *Bruggeman et al.* (2006) could show that the Modified Patankar schemes 7 - 8 are only conservative for one limiting nutrient and therefore they developed the Extended Modified Patankar (EMP) schemes 10 and 11 which are also stoichiometrically conservative. Patankar and Modified Patankar schemes of fourth order have not yet been developed, such that choices 6 and 9 do not work yet.
The call to `ode_solver()` requires a little explanation. The first argument `solver` is an integer and specifies which solver to use. The arguments `numc` and `nlev` gives the dimensions of the data structure `cc` i.e. `cc(1:numc,0:nlev)`. `dt` is simply the time step. The last argument is the most complicated. `right_hand_side` is a subroutine with a fixed argument list. The subroutine evaluates the right hand side of the ODE and may be called more than once during one time-step - for higher order schemes. For an example of a correct `right_hand_side` have a look at e.g. `do_bio_npzd()`

USES:

```
   IMPLICIT NONE
```

INPUT PARAMETERS:

```
   integer, intent(in)                 :: solver,nlev,numc
   REALTYPE, intent(in)                :: dt
  !INPUT/OUTPUT PARAMETER:
   REALTYPE, intent(inout)             :: cc(1:numc,0:nlev)

   interface
      subroutine right_hand_side_ppdd(first,numc,nlev,cc,pp,dd)
         logical, intent(in)                  :: first
         integer, intent(in)                  :: numc,nlev
         REALTYPE, intent(in)                 :: cc(1:numc,0:nlev)

         REALTYPE, intent(out)                :: pp(1:numc,1:numc,0:nlev)
         REALTYPE, intent(out)                :: dd(1:numc,1:numc,0:nlev)
      end
   end interface

   interface
      subroutine right_hand_side_rhs(first,numc,nlev,cc,rhs)
         logical, intent(in)                  :: first
         integer, intent(in)                  :: numc,nlev
         REALTYPE, intent(in)                 :: cc(1:numc,0:nlev)
         REALTYPE, intent(out)                :: rhs(1:numc,0:nlev)
      end
   end interface
```

REVISION HISTORY:

```
   Original author(s): Hans Burchard, Karsten Bolding
```

LOCAL VARIABLES:

8.13.1 First-order Euler-forward scheme

INTERFACE:

```
   subroutine euler_forward(dt,numc,nlev,cc,right_hand_side)
```

DESCRIPTION:

Here, the first-order Euler-forward (E1) scheme is coded, with one evaluation of the right-hand sides per time step:

$$c_i^{n+1} = c_i^n + \Delta t \left\{ P_i\left(\underline{c}^n\right) - D_i\left(\underline{c}^n\right) \right\}. \tag{250}$$

USES:

```
   IMPLICIT NONE
```

INPUT PARAMETERS:

```
   integer, intent(in)                 :: numc,nlev
   REALTYPE, intent(in)                :: dt
  !INPUT/OUTPUT PARAMETER:
   REALTYPE, intent(inout)             :: cc(1:numc,0:nlev)

   interface
      subroutine right_hand_side(first,numc,nlev,cc,rhs)
         logical, intent(in)                 :: first
         integer, intent(in)                 :: numc,nlev
         REALTYPE, intent(in)                :: cc(1:numc,0:nlev)
         REALTYPE, intent(out)               :: rhs(1:numc,0:nlev)
      end
   end interface
```

REVISION HISTORY:

```
   Original author(s): Hans Burchard, Karsten Bolding
```

LOCAL VARIABLES:

```
  logical  :: first
  REALTYPE :: rhs(1:numc,0:nlev)
  integer  :: i,ci
```

8.13.2 Second-order Runge-Kutta scheme

INTERFACE:

```
   subroutine runge_kutta_2(dt,numc,nlev,cc,right_hand_side)
```

DESCRIPTION:

Here, the second-order Runge-Kutta (RK2) scheme is coded, with two evaluations of the right hand side per time step:

$$\left.\begin{array}{rcl} c_i^{(1)} & = & c_i^n + \Delta t \left\{ P_i\left(\underline{c}^n\right) - D_i\left(\underline{c}^n\right)\right), \\ c_i^{n+1} & = & c_i^n + \dfrac{\Delta t}{2}\left\{ P_i\left(\underline{c}^n\right) + P_i\left(\underline{c}^{(1)}\right) - D_i\left(\underline{c}^n\right) - D_i\left(\underline{c}^{(1)}\right)\right\}. \end{array}\right\} \quad (251)$$

USES:

```
   IMPLICIT NONE
```

INPUT PARAMETERS:

```
   REALTYPE, intent(in)                :: dt
   integer, intent(in)                 :: numc,nlev
  !INPUT/OUTPUT PARAMETER:
   REALTYPE, intent(inout)             :: cc(1:numc,0:nlev)

   interface
      subroutine right_hand_side(first,numc,nlev,cc,rhs)
         logical, intent(in)                  :: first
         integer, intent(in)                  :: numc,nlev
         REALTYPE, intent(in)                 :: cc(1:numc,0:nlev)
         REALTYPE, intent(out)                :: rhs(1:numc,0:nlev)
      end
   end interface
```

REVISION HISTORY:

```
   Original author(s): Hans Burchard, Karsten Bolding
```

LOCAL VARIABLES:

```
  logical  :: first
  REALTYPE :: rhs(1:numc,0:nlev),rhs1(1:numc,0:nlev)
  REALTYPE :: cc1(1:numc,0:nlev)
  integer  :: i,ci
```

8.13.3 Fourth-order Runge-Kutta scheme

INTERFACE:

```
   subroutine runge_kutta_4(dt,numc,nlev,cc,right_hand_side)
```

DESCRIPTION:

Here, the fourth-order Runge-Kutta (RK4) scheme is coded, with four evaluations of the right hand sides per time step:

$$\left.\begin{aligned}
c_i^{(1)} &= c_i^n + \Delta t \left\{P_i\left(\underline{c}^n\right) - D_i\left(\underline{c}^n\right)\right\} \\
c_i^{(2)} &= c_i^n + \Delta t \left\{P_i\left(\frac{1}{2}\left(\underline{c}^n + \underline{c}^{(1)}\right)\right) - D_i\left(\frac{1}{2}\left(\underline{c}^n + \underline{c}^{(1)}\right)\right)\right\} \\
c_i^{(3)} &= c_i^n + \Delta t \left\{P_i\left(\frac{1}{2}\left(\underline{c}^n + \underline{c}^{(2)}\right)\right) - D_i\left(\frac{1}{2}\left(\underline{c}^n + \underline{c}^{(2)}\right)\right)\right\} \\
c_i^{(4)} &= c_i^n + \Delta t \left\{P_i\left(\underline{c}^{(3)}\right) - D_i\left(\underline{c}^{(3)}\right)\right\} \\
c_i^{n+1} &= \frac{1}{6}\left\{c_i^{(1)} + 2c_i^{(2)} + 2c_i^{(3)} + c_i^{(4)}\right\}.
\end{aligned}\right\} \tag{252}$$

USES:

```
   IMPLICIT NONE
```

INPUT PARAMETERS:

```
   REALTYPE, intent(in)              :: dt
   integer, intent(in)               :: numc,nlev
  !INPUT/OUTPUT PARAMETER:
   REALTYPE, intent(inout)           :: cc(1:numc,0:nlev)

   interface
      subroutine right_hand_side(first,numc,nlev,cc,rhs)
         logical, intent(in)                 :: first
         integer, intent(in)                 :: numc,nlev
         REALTYPE, intent(in)                :: cc(1:numc,0:nlev)
         REALTYPE, intent(out)               :: rhs(1:numc,0:nlev)
      end
   end interface
```

REVISION HISTORY:

```
   Original author(s): Hans Burchard, Karsten Bolding
```

LOCAL VARIABLES:

```
  logical  :: first
  REALTYPE :: rhs(1:numc,0:nlev),rhs1(1:numc,0:nlev)
  REALTYPE :: rhs2(1:numc,0:nlev),rhs3(1:numc,0:nlev)
  REALTYPE :: cc1(1:numc,0:nlev)
  integer  :: i,ci
```

8.13.4 First-order Patankar scheme

INTERFACE:

```
   subroutine patankar(dt,numc,nlev,cc,right_hand_side)
```

DESCRIPTION:

Here, the first-order Patankar-Euler scheme (PE1) scheme is coded, with one evaluation of the right hand sides per time step:

$$c_i^{n+1} = c_i^n + \Delta t \left\{ P_i\left(\underline{c}^n\right) - D_i\left(\underline{c}^n\right) \frac{c_i^{n+1}}{c_i^n} \right\}. \tag{253}$$

USES:

```
   IMPLICIT NONE
```

INPUT PARAMETERS:

```
   REALTYPE, intent(in)                :: dt
   integer, intent(in)                 :: numc,nlev
  !INPUT/OUTPUT PARAMETER:
   REALTYPE, intent(inout)             :: cc(1:numc,0:nlev)

   interface
      subroutine right_hand_side(first,numc,nlev,cc,pp,dd)
         logical, intent(in)                  :: first
         integer, intent(in)                  :: numc,nlev
         REALTYPE, intent(in)                 :: cc(1:numc,0:nlev)
         REALTYPE, intent(out)                :: pp(1:numc,1:numc,0:nlev)
         REALTYPE, intent(out)                :: dd(1:numc,1:numc,0:nlev)
      end subroutine right_hand_side
   end interface
```

REVISION HISTORY:

```
   Original author(s): Hans Burchard, Karsten Bolding
```

LOCAL VARIABLES:

```
  logical  :: first
  REALTYPE :: ppsum,ddsum
  REALTYPE :: pp(1:numc,1:numc,0:nlev),dd(1:numc,1:numc,0:nlev)
  integer  :: i,j,ci
```

8.13.5 Second-order Patankar-Runge-Kutta scheme

INTERFACE:

```
   subroutine patankar_runge_kutta_2(dt,numc,nlev,cc,right_hand_side)
```

DESCRIPTION:

Here, the second-order Patankar-Runge-Kutta (PRK2) scheme is coded, with two evaluations of the right hand sides per time step:

$$\left.\begin{array}{rcl} c_i^{(1)} & = & c_i^n + \Delta t \left\{ P_i\left(\underline{c}^n\right) - D_i\left(\underline{c}^n\right) \dfrac{c_i^{(1)}}{c_i^n} \right\}, \\ c_i^{n+1} & = & c_i^n + \dfrac{\Delta t}{2} \left\{ P_i\left(\underline{c}^n\right) + P_i\left(\underline{c}^{(1)}\right) - \left(D_i\left(\underline{c}^n\right) + D_i\left(\underline{c}^{(1)}\right)\right) \dfrac{c_i^{n+1}}{c_i^{(1)}} \right\}. \end{array}\right\} \qquad (254)$$

USES:

```
   IMPLICIT NONE
```

INPUT PARAMETERS:

```
   REALTYPE, intent(in)                :: dt
   integer, intent(in)                 :: numc,nlev
  !INPUT/OUTPUT PARAMETER:
   REALTYPE, intent(inout)             :: cc(1:numc,0:nlev)

   interface
      subroutine right_hand_side(first,numc,nlev,cc,pp,dd)
         logical, intent(in)                  :: first
         integer, intent(in)                  :: numc,nlev
         REALTYPE, intent(in)                 :: cc(1:numc,0:nlev)
         REALTYPE, intent(out)                :: pp(1:numc,1:numc,0:nlev)
         REALTYPE, intent(out)                :: dd(1:numc,1:numc,0:nlev)
      end subroutine right_hand_side
   end interface
```

REVISION HISTORY:

```
   Original author(s): Hans Burchard, Karsten Bolding
```

LOCAL VARIABLES:

```
  logical  :: first
  REALTYPE :: ppsum(1:numc,0:nlev),ddsum(1:numc,0:nlev)
  REALTYPE :: pp(1:numc,1:numc,0:nlev),dd(1:numc,1:numc,0:nlev)
  REALTYPE :: cc1(1:numc,0:nlev)
  integer  :: i,j,ci
```

8.13.6 Fourth-order Patankar-Runge-Kutta scheme

INTERFACE:

```
   subroutine patankar_runge_kutta_4(dt,numc,nlev,cc,right_hand_side)
```

DESCRIPTION:

This subroutine should become the fourth-order Patankar Runge-Kutta scheme, but it does not yet work.

USES:

```
   IMPLICIT NONE
```

INPUT PARAMETERS:

```
   REALTYPE, intent(in)                :: dt
   integer, intent(in)                 :: numc,nlev
  !INPUT/OUTPUT PARAMETER:
   REALTYPE, intent(inout)             :: cc(1:numc,0:nlev)

   interface
```

```
    subroutine right_hand_side(first,numc,nlev,cc,pp,dd)
       logical, intent(in)                :: first
       integer, intent(in)                :: numc,nlev
       REALTYPE, intent(in)               :: cc(1:numc,0:nlev)
       REALTYPE, intent(out)              :: pp(1:numc,1:numc,0:nlev)
       REALTYPE, intent(out)              :: dd(1:numc,1:numc,0:nlev)
    end subroutine right_hand_side
 end interface
```

REVISION HISTORY:

```
Original author(s): Hans Burchard, Karsten Bolding
```

LOCAL VARIABLES:

```
logical  :: first
REALTYPE :: ppsum(1:numc,0:nlev),ddsum(1:numc,0:nlev)
REALTYPE :: ppsum1(1:numc,0:nlev),ddsum1(1:numc,0:nlev)
REALTYPE :: ppsum2(1:numc,0:nlev),ddsum2(1:numc,0:nlev)
REALTYPE :: ppsum3(1:numc,0:nlev),ddsum3(1:numc,0:nlev)
REALTYPE :: pp(1:numc,1:numc,0:nlev),dd(1:numc,1:numc,0:nlev)
REALTYPE :: cc1(1:numc,0:nlev)
integer  :: i,j,ci
```

8.13.7 First-order Modified Patankar scheme

INTERFACE:

```
subroutine modified_patankar(dt,numc,nlev,cc,right_hand_side)
```

DESCRIPTION:

Here, the first-order Modified Patankar-Euler scheme (MPE1) scheme is coded, with one evaluation of the right hand side per time step:

$$c_i^{n+1} = c_i^n + \Delta t \left\{ \sum_{\substack{j=1 \\ j\neq i}}^{I} p_{i,j}\left(\underline{c}^n\right)\frac{c_j^{n+1}}{c_j^n} + p_{i,i}\left(\underline{c}^n\right) - \sum_{j=1}^{I} d_{i,j}\left(\underline{c}^n\right)\frac{c_i^{n+1}}{c_i^n} \right\}. \tag{255}$$

USES:

```
IMPLICIT NONE
```

INPUT PARAMETERS:

```
 REALTYPE, intent(in)              :: dt
 integer, intent(in)               :: numc,nlev
!INPUT/OUTPUT PARAMETER:
 REALTYPE, intent(inout)             :: cc(1:numc,0:nlev)

 interface
    subroutine right_hand_side(first,numc,nlev,cc,pp,dd)
       logical, intent(in)                :: first
       integer, intent(in)                :: numc,nlev
       REALTYPE, intent(in)               :: cc(1:numc,0:nlev)
       REALTYPE, intent(out)              :: pp(1:numc,1:numc,0:nlev)
       REALTYPE, intent(out)              :: dd(1:numc,1:numc,0:nlev)
    end subroutine right_hand_side
 end interface
```

REVISION HISTORY:

```
   Original author(s): Hans Burchard, Karsten Bolding
```

LOCAL VARIABLES:

```
  logical  :: first
  REALTYPE :: pp(1:numc,1:numc,0:nlev),dd(1:numc,1:numc,0:nlev)
  REALTYPE :: a(1:numc,1:numc),r(1:numc)
  integer  :: i,j,ci
```

8.13.8 Second-order Modified Patankar-Runge-Kutta scheme

INTERFACE:

```
   subroutine modified_patankar_2(dt,numc,nlev,cc,right_hand_side)
```

DESCRIPTION:

Here, the second-order Modified Patankar-Runge-Kutta (MPRK2) scheme is coded, with two evaluations of the right hand sides per time step:

$$\left.\begin{aligned}
c_i^{(1)} &= c_i^n + \Delta t \left\{ \sum_{\substack{j=1\\ j\neq i}}^{I} p_{i,j}\left(\underline{c}^n\right) \frac{c_j^{(1)}}{c_j^n} + p_{i,i}\left(\underline{c}^n\right) - \sum_{j=1}^{I} d_{i,j}\left(\underline{c}^n\right) \frac{c_i^{(1)}}{c_i^n} \right\}, \\
c_i^{n+1} &= c_i^n + \frac{\Delta t}{2} \left\{ \sum_{\substack{j=1\\ j\neq i}}^{I} \left(p_{i,j}\left(\underline{c}^n\right) + p_{i,j}\left(\underline{c}^{(1)}\right) \right) \frac{c_j^{n+1}}{c_j^{(1)}} + p_{i,i}\left(\underline{c}^n\right) + p_{i,i}\left(\underline{c}^{(1)}\right) \right. \\
&\qquad \left. - \sum_{j=1}^{I} \left(d_{i,j}\left(\underline{c}^n\right) + d_{i,j}\left(\underline{c}^{(1)}\right) \right) \frac{c_i^{n+1}}{c_i^{(1)}} \right\}.
\end{aligned}\right\} \tag{256}$$

USES:

```
   IMPLICIT NONE
```

INPUT PARAMETERS:

```
   REALTYPE, intent(in)               :: dt
   integer, intent(in)                :: numc,nlev
  !INPUT/OUTPUT PARAMETER:
   REALTYPE, intent(inout)            :: cc(1:numc,0:nlev)

   interface
      subroutine right_hand_side(first,numc,nlev,cc,pp,dd)
         logical, intent(in)                :: first
         integer, intent(in)                :: numc,nlev
         REALTYPE, intent(in)               :: cc(1:numc,0:nlev)
         REALTYPE, intent(out)              :: pp(1:numc,1:numc,0:nlev)
         REALTYPE, intent(out)              :: dd(1:numc,1:numc,0:nlev)
      end subroutine right_hand_side
   end interface
```

REVISION HISTORY:

```
   Original author(s): Hans Burchard, Karsten Bolding
```

LOCAL VARIABLES:

```
  logical  :: first
  REALTYPE :: pp(1:numc,1:numc,0:nlev),dd(1:numc,1:numc,0:nlev)
  REALTYPE :: pp1(1:numc,1:numc,0:nlev),dd1(1:numc,1:numc,0:nlev)
  REALTYPE :: a(1:numc,1:numc),r(1:numc)
  REALTYPE :: cc1(1:numc,0:nlev)
  integer  :: i,j,ci
```

8.13.9 Fourth-order Modified Patankar-Runge-Kutta scheme

INTERFACE:

```
   subroutine modified_patankar_4(dt,numc,nlev,cc,right_hand_side)
```

DESCRIPTION:

This subroutine should become the fourth-order Modified Patankar Runge-Kutta scheme, but it does not yet work.

USES:

```
   IMPLICIT NONE
```

INPUT PARAMETERS:

```
   REALTYPE, intent(in)                 :: dt
   integer, intent(in)                  :: numc,nlev
  !INPUT/OUTPUT PARAMETER:
   REALTYPE, intent(inout)              :: cc(1:numc,0:nlev)

   interface
      subroutine right_hand_side(first,numc,nlev,cc,pp,dd)
         logical, intent(in)                   :: first
         integer, intent(in)                   :: numc,nlev
         REALTYPE, intent(in)                  :: cc(1:numc,0:nlev)
         REALTYPE, intent(out)                 :: pp(1:numc,1:numc,0:nlev)
         REALTYPE, intent(out)                 :: dd(1:numc,1:numc,0:nlev)
      end subroutine right_hand_side
   end interface
```

REVISION HISTORY:

```
   Original author(s): Hans Burchard, Karsten Bolding
```

LOCAL VARIABLES:

```
  logical  :: first
  REALTYPE :: pp(1:numc,1:numc,0:nlev),dd(1:numc,1:numc,0:nlev)
  REALTYPE :: pp1(1:numc,1:numc,0:nlev),dd1(1:numc,1:numc,0:nlev)
  REALTYPE :: pp2(1:numc,1:numc,0:nlev),dd2(1:numc,1:numc,0:nlev)
  REALTYPE :: pp3(1:numc,1:numc,0:nlev),dd3(1:numc,1:numc,0:nlev)
  REALTYPE :: a(1:numc,1:numc),r(1:numc)
  REALTYPE :: cc1(1:numc,0:nlev)
  integer  :: i,j,ci
```

8.13.10 First-order Extended Modified Patankar scheme

INTERFACE:

```
   subroutine emp_1(dt,numc,nlev,cc,right_hand_side)
```

DESCRIPTION:

Here, the first-order Extended Modified Patankar scheme for biogeochemical models is coded, with one evaluation of the right-hand side per time step:

$$\begin{array}{c} \vec{c}^{n+1} = \vec{c}^n + \Delta t\, \vec{f}(t^n, \vec{c}^n) \displaystyle\prod_{j \in J^n} \frac{c_j^{n+1}}{c_j^n} \\ \text{with } J^n = \{i : f_i(t^n, \vec{c}^n) < 0, i \in \{1, ..., I\}\} \end{array} \tag{257}$$

This system of non-linear implicit equations is solved in auxiliary subroutine findp_bisection, using the fact this system can be reduced to a polynomial in one unknown, and additionally using the restrictions imposed by the requirement of positivity. For more details, see *Bruggeman et al.* (2006).

USES:

```
   IMPLICIT NONE
```

INPUT PARAMETERS:

```
   REALTYPE, intent(in)              :: dt
   integer, intent(in)               :: numc,nlev
  !INPUT/OUTPUT PARAMETER:
   REALTYPE, intent(inout)           :: cc(1:numc,0:nlev)

   interface
      subroutine right_hand_side(first,numc,nlev,cc,rhs)
         logical, intent(in)                 :: first
         integer, intent(in)                 :: numc,nlev
         REALTYPE, intent(in)                :: cc(1:numc,0:nlev)
         REALTYPE, intent(out)               :: rhs(1:numc,0:nlev)
      end
   end interface
```

REVISION HISTORY:

```
   Original author(s): Jorn Bruggeman
```

LOCAL VARIABLES:

```
  logical  :: first
  REALTYPE :: derivative(1:numc,0:nlev)
  integer  :: ci
  REALTYPE :: pi
```

8.13.11 Second-order Extended Modified Patankar scheme

INTERFACE:

```
   subroutine emp_2(dt,numc,nlev,cc,right_hand_side)
```

DESCRIPTION:

Here, the second-order Extended Modified Patankar scheme for biogeochemical models is coded, with two evaluations of the right-hand side per time step:

$$\begin{aligned}
\vec{c}^{(1)} &= \vec{c}^{n} + \Delta t \, \vec{f}(t^n, \vec{c}^n) \prod_{j \in J^n} \frac{c_j^{(1)}}{c_j^n} \\
\vec{c}^{n+1} &= \vec{c}^{n} + \frac{\Delta t}{2} \left(\vec{f}(t^n, \vec{c}^n) + \vec{f}(t^{n+1}, \vec{c}^{(1)}) \right) \prod_{k \in K^n} \frac{c_k^{n+1}}{c_k^{(1)}}
\end{aligned} \tag{258}$$

where

$$
\begin{aligned}
J^n &= \{i : f_i(t^n, \vec{c}^n) < 0, i \in \{1, ..., I\}\} \\
K^n &= \left\{i : f_i(t^n, \vec{c}^n) + f_i(t^{n+1}, \vec{c}^{(1)}) < 0, i \in \{1, ..., I\}\right\}.
\end{aligned} \tag{259}
$$

The first step is identical to a step with the first-order EMP scheme. The second step mathmatically identical to a step with the first-order scheme if we rewrite it as

$$
\begin{aligned}
&\vec{c}^{n+1} = \vec{c}^n + \Delta t \, \vec{h}(t^n, t^{n+1}, \vec{c}^n, \vec{c}^{(1)}) \prod_{k \in K^n} \frac{c_k^{n+1}}{c_k^n} \\
&\quad \text{with } \vec{h}(t^n, t^{n+1}, \vec{c}^n, \vec{c}^{(1)}) = \frac{1}{2}\left(\vec{f}(t^n, \vec{c}^n) + \vec{f}(t^{n+1}, \vec{c}^{(1)})\right) \prod_{k \in K^n} \frac{c_k^n}{c_k^{(1)}}.
\end{aligned} \tag{260}
$$

Therefore, this scheme can be implemented as two consecutive steps with the first-order scheme, the second using $\vec{h}(t^n, t^{n+1}, \vec{c}^n, \vec{c}^{(1)})$. The non-linear problem of each consecutive step is solved in auxiliary subroutine findp_bisection.
For more details, see *Bruggeman et al.* (2006).

USES:

```
   IMPLICIT NONE
```

INPUT PARAMETERS:

```
   REALTYPE, intent(in)                :: dt
   integer, intent(in)                 :: numc,nlev
  !INPUT/OUTPUT PARAMETER:
   REALTYPE, intent(inout)              :: cc(1:numc,0:nlev)

   interface
      subroutine right_hand_side(first,numc,nlev,cc,rhs)
         logical, intent(in)                  :: first
         integer, intent(in)                  :: numc,nlev
         REALTYPE, intent(in)                 :: cc(1:numc,0:nlev)
         REALTYPE, intent(out)                :: rhs(1:numc,0:nlev)
      end
   end interface
```

REVISION HISTORY:

```
   Original author(s): Jorn Bruggeman
```

LOCAL VARIABLES:

```
  logical  :: first
  integer  :: i,ci
  REALTYPE :: pi, rhs(1:numc,0:nlev), cc_med(1:numc,0:nlev), rhs_med(1:numc,0:nlev)
```

8.13.12 Calculation of the EMP product term 'p'

INTERFACE:

```
subroutine findp_bisection(numc, cc, derivative, dt, accuracy, pi)
```

DESCRIPTION:

Auxiliary subroutine for finding the Extended Modified Patankar product term p with the bisection technique.
This subroutine solves the non-linear problem

$$\begin{aligned} \vec{c}^{n+1} &= \vec{c}^{n} + \Delta t\, \vec{f}(t^n, \vec{c}^n) \prod_{j \in J^n} \frac{c_j^{n+1}}{c_j^n} \\ &\text{with } J^n = \{i : f_i(t^n, \vec{c}^n) < 0, i \in \{1, ..., I\}\} \end{aligned} \tag{261}$$

using the fact that it can be reduced to the problem of finding the root of a polynomial in one unknown $p := \prod\limits_{j \in J^n} c_j^{n+1}/c_j^n$:

$$g(p) = \prod_{j \in J^n} \left(1 + \frac{\Delta t\, f_j(t^n, \vec{c}^n)}{c_j^n} p \right) - p = 0, \tag{262}$$

with

$$J^n = \{i : f_i(t^n, \vec{c}^n) < 0, i \in \{1, ..., I\}\}, \tag{263}$$

Additionally, it makes use of the the positivity requirement $\vec{c}_i^{n+1} > 0 \;\forall\, i$, which imposes restriction

$$p \in \left(0, \min\left(1, \min_{j \in J^n} \left(-\frac{c_j^n}{\Delta t\, f_j(t^n, \vec{c}^n)} \right)\right)\right). \tag{264}$$

It has been proved that there exists exactly one p for which the above is true, see *Bruggeman et al.* (2006). The resulting problem is solved using the bisection scheme, which is guaranteed to converge.

USES:

```
implicit none
```

INPUT PARAMETERS:

```
 integer, intent(in)   :: numc
 REALTYPE, intent(in)  :: cc(1:numc), derivative(1:numc)
 REALTYPE, intent(in)  :: dt, accuracy
!OUTPUT PARAMETER:
 REALTYPE, intent(out) :: pi
```

REVISION HISTORY:

```
Original author(s): Jorn Bruggeman
```

LOCAL VARIABLES:

```
   REALTYPE :: pileft, piright, fnow
   REALTYPE :: relderivative(1:numc)
   integer  :: iter, i, potnegcount
```

8.13.13 Matrix solver

INTERFACE:

```
   subroutine matrix(n,a,r,c)
```

DESCRIPTION:

This is a Gaussian solver for multi-dimensional linear equations.

USES:

```
   IMPLICIT NONE
```

INPUT PARAMETERS:

```
   integer, intent(in)                 :: n
  INPUT/OUTPUT PARAMETERS:
  REALTYPE                             :: a(1:n,1:n),r(1:n)
  OUTPUT PARAMETERS:
  REALTYPE, intent(out)                :: c(1:n)
```

REVISION HISTORY:

```
   Original author(s): Hans Burchard, Karsten Bolding
```

LOCAL VARIABLES:

```
  integer  :: i,j,k
```

9 Biogeochemical models

In this module, biogeochemical models are implemented, in a two-way coupled mode.

9.1 Mathematical formulation

The general structure of a biogeochemical model with I state variables expressed as ensemble averaged concentrations is given by the following set of equations:

$$\partial_t c_i + \partial_z \left(m_i c_i - K_V \partial_z c_i\right) = P_i(\vec{c}) - D_i(\vec{c}), \quad i = 1, \ldots, I, \quad j, k = 1, \ldots, 3, \tag{265}$$

with c_i denoting the concentrations of state variables. Furthermore, m_i represents the autonomous motion of the ecosystem component c_i (e.g. sinking or active swimming), and K_V represents the eddy diffusivity. The source and sink terms of the ecosystem component c_i are summarised in $P_i(\vec{c})$ and $D_i(\vec{c})$, respectively. For three-dimensional models, advection with the flow field and horizontal advection would have to be accounted for additionally. In many biogeochemical models, some of the state variables have positive lower limits. In order to account for this, we defined all state variables as the difference between the actual value and their lower limit, such that (for non-negative state variables only) the model value c_i represents a concentration of $c_i + c_i^{\min}$ where $c_i^{\min}$ is the lower limit of c_i.
The gradient term on the left hand side of (265) is the total transport, for which typically surface and bottom boundary conditions

$$K_V \partial_z c_i \big|_{z=\eta} = F_i^s, \qquad K_V \partial_z c_i \big|_{z=-H} = -F_i^b, \tag{266}$$

with surface and bottom fluxes, F_i^s and F_i^b, respectively, are applied. The right hand side denotes the reaction terms, which are composed of contributions $d_{i,j}(\vec{c})$, which represent reactive fluxes from c_i to c_j, and in turn, $p_{i,j}(\vec{c})$ are reactive fluxes from c_j received by c_i:

$$P_i(\vec{c}) = \sum_{j=1}^{I} p_{i,j}(\vec{c}), \quad D_i(\vec{c}) = \sum_{j=1}^{I} d_{i,j}(\vec{c}), \tag{267}$$

with $d_{i,j} \geq 0$ for all i, j and $p_{i,j} \geq 0$ for all $i \neq j$.
We basically consider two types of ecosystem models. In the simple NPZ (nutrient-phytoplankton-zooplankton) type models all state variables are based on the same measurable unit such as [mmol N m^{-3}] for nitrogen-based models. In such NPZ models the reactive terms do only exchange mass between state variables with

$$p_{i,j}(\vec{c}) = d_{j,i}(\vec{c}), \quad \text{for } i \neq j \quad \text{and} \quad p_{i,i}(\vec{c}) = d_{i,i}(\vec{c}) = 0, \quad \text{for } i = j. \tag{268}$$

Neglecting for a moment all transport terms, it is easily seen that this simple type of model is conserving mass:

$$d_t\left(\sum_{i=1}^{I} c_i\right) = \sum_{i=1}^{I}\left(P_i(\vec{c}) - D_i(\vec{c})\right) = \sum_{i=1}^{I}\sum_{j=1}^{I}\left(p_{i,j}(\vec{c}) - d_{i,j}(\vec{c})\right) = \sum_{i=1}^{I}\left(p_{i,i}(\vec{c}) - d_{i,i}(\vec{c})\right) = 0. \tag{269}$$

The NPZD model (see section **??**) and the *Fasham et al.* (1990) model discussed in section **??** are such fully conservative models.

In many biogeochemical models most state variables are known to be positive or at least non-negative quantities. For non-negative initial conditions $c_i(0) \geq 0$ one can easily show by a simple contradiction argument that the condition

$$d_{i,j}(\vec{c}) \longrightarrow 0 \ \text{ for } \ c_i \longrightarrow 0 \tag{270}$$

guarantees that the quantities $c_i(t) \geq 0$, remain non-negative for all t. A typical example is $d_{i,j}(\vec{c}) = f(\vec{c})c_i$ with a non-negative, bounded function f which might depend on all c_i.
However, for many applications such simple models are too restrictive. Often different spatial references are involved for the state variables, such as the detritus concentration in the water column, measured in [mmol N m^{-3}] and the fluff layer concentration at the bed, measured in [mmol N m^{-2}]. Many biogeochemical processes involve more than two substances such as the photosynthesis where different nutrients (e.g. nitrate and phosphorus) are taken up by phytoplankta and oxygen is produced. The ratios between these substances dissipated or produced are usually fixed, in the example of photosynthesis uptake of 16 mmol m^{-3} nitrate is connected to an uptake of 1 mmol m^{-3} phosphorus and a production of 8.125 mmol m^{-3} oxygen.
For state variables which may be negative (such as oxygen concentration which also includes oxygen demand units, all sink and source terms are added up in the production terms $p_{i,j}$, with a negative sign for the sink terms. For the *Neumann et al.* (2002) model discussed in sections **??**, further deviations from the conservation formulation are introduced since biogeochemical reactions include substances which are not budgeted by the model (mostly because they are assumed to be not limiting). One typical example is nitrogen fixated by blue-green algae which builds up biomass by using atmospheric nitrogen which is later recycled to nitrate. Such non-conservative terms are lumped into the diagonal terms $p_{i,i}$ and $d_{i,i}$.

9.2 Numerical aspects

Two basic aspects which are included in the mathematical formulation for the biogeochemical equations discussed in section 9.1 are to be reproduced by the numerical methods applied: conservation and positivity. Another constraint for the choice of numerical methods is that they should be sufficiently stable and accurate. In order to facilitate this, a split method is applied separating the numerical solution of the transport part (advection, diffusion) and the reaction part. By doing so, we take splitting errors into account which should however be not significant as long as the typical reaction time scales are much longer than the constant model time step Δt.
In the transport step in which the right hand side is set to zero, finite volume discretisations are used such that conservation of mass is guaranteed. The spatial discretisation is carried out by separating the water column into N not necessarily equidistant intervals of height h_k. The state variables, represented by layer-averaged values, are located in the centres of these intervals, the advective and diffusive fluxes are located at the interfaces in between. The transport step itself is subject to operator splitting. The autonomous motion of the state variables (including sinking or rising due to negative or positive buoyancy, respectively) is discretised by means of TVD (Total Variation Diminishing) advection schemes, for which several choices are available, see *Pietrzak* (1998). These TVD schemes are positivity conserving due to their TVD property. The most accurate among those schemes is the so-called PDM-limited P$_2$ scheme which has been described in detail by *Leonard* (1991).
For the diffusion, a central in space scheme is used which is slightly biased towards a backward in time scheme in order to avoid asymptotic instability (see *Samarskij* (1984)). By doing so, positivity is obtained and the schemes are practically second order in time and space.
With the discretisations of the transport terms given above, accuracy, positivity and conservation of the state variables are guaranteed by means of standard schemes. For the reaction terms, *Burchard*

et al. (2003) recently developed schemes which also fulfil these requirements. Due to the operator split between transport and reaction terms, only ordinary differential equations (ODEs) have to be treated numerically for the latter terms. For the case of conservative biogeochemical models with $p_{i,i} = d_{i,i} = 0$, these schemes are identical to those given by *Burchard et al.* (2003). For $p_{i,i} \neq 0$, some modifications are necessary. Three classes of schemes are considered: Explicit schemes such as the Euler-forward scheme and second- and fourth-order Runge-Kutta schemes (see section 8.13). These schemes are known to be conservative, but for sufficiently large time steps they may compute negative values of state variables also for non-negative state variables. This may be avoided by small time stepping, which however usually leads to an enormous increase of the computational effort such that these schemes lose their practical relevance in this context. In order to solve this problem, *Patankar* (1980) had suggested the first-order in time positive definite scheme (253), and *Burchard et al.* (2003) have extended this to second order, see (254). However, these schemes are not conservative, since source and sink terms are numerically treated in a different way. Fully conservative and non-negative schemes in first- and second-oder in time have thus been suggested and tested for ordinary differential equations by *Burchard et al.* (2003), with $p_{i,i} = d_{i,i} = 0$ in equations. (255) and (256) in section 8.13. This equal numerical treatment of sources and sinks results in implicit linear systems of equations. Since only ordinary differential equations are to be solved in each grid point, these systems have small dimensions, for example $I = 7$ for the *Fasham et al.* (1990) model (see section **??**) and $I = 10$ for the *Neumann et al.* (2002) model (see section **??**). Thus, these linear systems may be directly solved by Gaussian elimination schemes. Nevertheless, one can also employ iterative methods. Especially for the linear system arising in the context of the present type of equations it is proven in *Burchard et al.* (2003) that the involved matrix is always non-singular and the standard Jacobi-type method converge to the unique solution of the system. Later, *Bruggeman et al.* (2006) found that the Modified Patankar schemes as described in equations (255) and (256) are only conservative for systems with one model currency (e.g. nitrogen in the model of *Fasham et al.* (1990)), but do not conserve stoichiometric ratios, when several limiting nutrients are present. To solve that problem, *Bruggeman et al.* (2006) developed first- and second-order Extended Modified Patankar (EMP) schemes, which are stoiciometrically conservative and explicit, such that they do not need to solve implicit systems of linear equations.

9.3 Computational aspects

The computational structure of the coupled physical-biogeochemical model system implemented here has been designed under consideration of various objectives. In this section a description of the various design related decisions is given. The major objectives are:

- to provide a well-defined interface between the one-dimensional physical model and a sufficiently generic biogeochemical model,
- to allow for easy extensions of the system with new biogeochemical models without changing the over-all structure, such as the *Fasham et al.* (1990) model (see section **??**) and the *Neumann et al.* (2002) model, see section **??**),
- to provide a number of solution methods for the 'process part' of the biogeochemical model, i.e. solvers for ordinary differential equations as discussed in section 9.2 and section 8.13,
- to obtain fast and efficient execution of the coupled model,
- to design the system in such a way that three-dimensional models can easily be interfaced with it.

For the biogeochemical model, we have adopted the same design strategy as has been used for the turbulence module of GOTM. The interface between an application using the turbulence module consists of two subroutine calls only: *init_turbulence()* and *do_turbulence()*. The subroutine *init_turbulence()* is responsible for initialising the parameters of the turbulence module and called as part of the initialisation of the entire model. For the turbulence module the initialisation includes reading *namelists* with the model configuration, allocating memory for all necessary variables and initialising these variables with sensible values. *init_turbulence()* should only be called once during program execution and after this call all public and private variables of the turbulence module should be in a consistently initialised state. During the time integration of the model, *do_turbulence()* has to be called at each time step. It is called with a number of parameters to transfer information e.g. from the mean-flow module to the turbulence module but also to receive the variables updated by the turbulence module.

Using the same strategy for the biogeochemical module has some problematic implications which are described here. In analogy to the turbulence module, the interface is given via the two subroutines *init_bio()* and *do_bio()*. The major difference between the turbulence module and the biogeochemical module in terms of implementation is that in the former the number of variables are known at compilation time and the dimensions are specified at run time where as in the latter both the number of state variables and their dimension are known only at run time. The general interface has to be able to handle not only the different biogeochemical models implemented at present but also to provide a framework for developing future models. There are two major items to address: 1. how to initialise the biogeochemical model and 2. how to select the right biogeochemical model during the time integration and use the selected ordinary differential equation (ODE) solver.

To solve the initialisation problem we have chosen a two-level initialisation approach. At the first level variables not specific to any of the biogeochemical models are initialised. The single most important variable during this phase is *bio_model*, which contains the identification number for all implemented biogeochemical models. Depending on the value of *bio_model*, the second level of initialisation is started. At this level all model specific variables (such as process rates) are initialised. The most important variable at the second level is I (number of state variables). After this step, the system returns to the first level, and now all information is available for allocating memory and initialising all variables. The most important data structure provided to the individual biogeochemical models will briefly be mentioned here. $c_{i,k}$ with $1 \leq i \leq I$ and $1 \leq k \leq N$ (number of vertical layers) is a two-dimensional array containing the concentrations of each variable at each depth. I is supplied by the individual biogeochemical model and N is transferred in the call to *init_bio()* from the physical model.

After the initialisation, all variables are initialised in a common data structure where the only link to the specific model is via *bio_model* and I. The next step is to design the actual time integration in such a way that selected biogeochemical model operates on the common data structure using the selected ODE solver in a transparent way.

Figure 3 shows a sketch of how this is organised in the model source code. The sketch should be be read from left to right. At the left side we have the interface *do_bio()*, which is the only connection to the calling program. The next level shows a sequence of steps necessary to do the time integration. It should be noted that not all biogeochemical models necessarily have to execute all the steps, some models do e.g. not need any surface fluxes or short-wave radiation. For the diffusion/advection part a general subroutine is called which is also used by the physical model. After having calculated I_{PAR} and B (see eqs. (29) and (**??**), the next step is the step at which the production and destruction terms ($p_{i,j,k}$ and $d_{i,j,k}$) of the biogeochemical models are calculated. This is done via a call to *ode_solver()*. After the call to *ode_solver()*, $c_{i,k}$ has been updated with the new values of all variables in the biogeochemical model. Which ODE solver to use is determined during the initialisation phase (*ode_method*). It should be noted that for some of

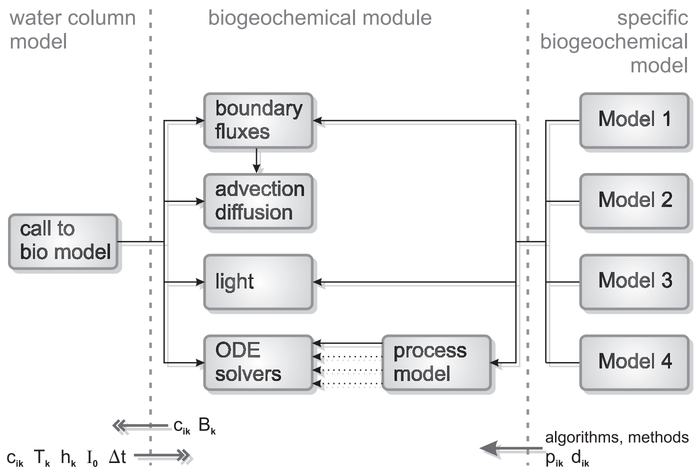

Figure 3: The structure of the *do_bio()* subroutine. This subroutine is responsible for updating all variables in the biogeochemical model in question at each time-step. *do_bio()* essentially works as a wrapper around all biogeochemical models implemented. The hatched arrows from *process model* to *ODE solvers* indicate that between one and four calls of *process model* per time step are performed, depending on the order of the chosen ODE solver.

the solution methods the biogeochemical processes have to be evaluated more than once. Instead of having *ode_solver()* being responsible for calling the chosen biogeochemical model, an additional subroutine has been introduced: *process_model*, is a simple wrapper routine calling the selected biogeochemical model.
The implementation of this biogeochemical module into three-dimensional models is straightforward. The 3D model has to take care of storing all three-dimensional state variables and calculate their advection with the mean flow and the horizontal diffusion. Settling, migration, vertical diffusion and the production/destruction processes are calculated by the biogeochemical module which has to be called by means of a loop over all horizontal grid boxes of the 3D model. This text has been adapted from *Burchard et al.* (2006).

10 Extra features

Here, some extra features are stored which are up to now

- the `seagrass` module.

The `seagrass` scenario in section 10.1 investigates the *Verduin and Backhaus* (2000) seagrass-current simulations.

10.1 Module seagrass — sea grass dynamics

INTERFACE:

```
module seagrass
```

DESCRIPTION:

In this module, seagrass canopies are treated as Lagrangian tracers, which either advect passively with the horizontal current speed or rest at their excursion limits and thus exert friction on the mean flow, see *Verduin and Backhaus* (2000). Turbulence generation due to seagrass friction is possible, see namelist file `seagrass.nml`. The extra production term in the balance of TKE, (152), is included as described in section 4.8.

USES:

```
default: all is private.
private
```

PUBLIC MEMBER FUNCTIONS:

```
public init_seagrass, do_seagrass, save_seagrass, end_seagrass
logical, public                     :: seagrass_calc
```

REVISION HISTORY:

```
Original author(s): Hans Burchard & Karsten Bolding
REALTYPE, dimension(:), allocatable :: xx,yy
REALTYPE, dimension(:), allocatable :: exc,vfric,grassz,xxP
logical                   :: init_output
from a namelist
character(len=PATH_MAX)   :: grassfile='seagrass.dat'
REALTYPE                  :: XP_rat
integer                   :: grassind
integer                   :: grassn
integer                   :: out_unit
integer                   :: maxn
```

10.1.1 Initialise the sea grass module

INTERFACE:

```
subroutine init_seagrass(namlst,fname,unit,nlev,h)
```

DESCRIPTION:

Here, the seagrass namelist `seagrass.nml` is read and memory is allocated for some relevant vectors. Afterwards, excursion limits and friction coefficients are read from a file. The uppermost grid related index for the seagrass canopy is then calculated.

USES:

```
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer,          intent(in)   :: namlst
character(len=*), intent(in)   :: fname
integer,          intent(in)   :: unit
integer,          intent(in)   :: nlev
REALTYPE,         intent(in)   :: h(0:nlev)
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding & Hans Burchard
```

LOCAL VARIABLES:

```
integer                    :: i,rc
REALTYPE                   :: z
namelist /canopy/  seagrass_calc,grassfile,XP_rat
```

10.1.2 Update the sea grass model

INTERFACE:

```
subroutine do_seagrass(nlev,dt)
```

DESCRIPTION:

Here the time depending seagrass equation suggested by *Verduin and Backhaus* (2000) is calculated. In order to explain the basic principle, an idealised example is examined here with a simplified momentum equation,

$$\partial_t u - \partial_z(\nu_t \partial_z u) = -g\partial_x \zeta - C_f u|u| \ , \tag{271}$$

and the Lagrangian tracer equation for seagrass,

$$\partial_t X = \begin{cases} u & \text{for } |X| < X_{\max} \text{ or } X \cdot u < 0, \\ 0 & \text{else }, \end{cases} \tag{272}$$

where X is the Langrangian horizontal excursion of the seagrass. The seagrass friction coefficient, C_f, is only non–zero at heights where seagrass tracers are at their excursion limits:

$$C_f = \begin{cases} C_f^{\max} & \text{for } |X| = X_{\max} \ , \\ 0 & \text{else } \ . \end{cases} \tag{273}$$

The maximum excursion limits $X_{\max}$ and the friction coefficients $C_f^{\max}$ are read from a file.
The production of turbulence is calculated here as the sum of shear production and friction loss at the seagrass leaves,

$$X_P = \alpha_{sg} C_f |u|^3 \ , \tag{274}$$

which is added to the usual shear–production term as illustrated in (148). The efficiency coefficient of turbulence production by sea–grass friction, α_{sg}, is denoted as `xP_rat` in the code. It has to be read–in from the `canopy` namelist. For details and example calculations, see *Burchard and Bolding* (2000).

USES:

```
   use meanflow, only:     u,v,h,drag,xP
   IMPLICIT NONE
```

INPUT PARAMETERS:

```
   integer,  intent(in)     :: nlev
   REALTYPE, intent(in)     :: dt
```

REVISION HISTORY:

```
   Original author(s): Karsten Bolding & Hans Burchard
```

LOCAL VARIABLES:

```
   integer                   :: i
   REALTYPE                  :: dist
   REALTYPE                  :: grassfric(0:nlev)
   REALTYPE                  :: excur(0:nlev)
   REALTYPE                  :: z(0:nlev)
```

10.1.3 Storing the results

INTERFACE:

```
   subroutine save_seagrass
```

DESCRIPTION:

Here, storing of the seagrass profiles to an ascii or a netCDF file is managed.

USES:

```
   use meanflow, only:     h
   use output, only: out_fmt,ascii_unit,ts
 ifdef NETCDF_FMT
   use netcdf
   use ncdfout, only: ncid
   use ncdfout, only: lon_dim,lat_dim,z_dim,time_dim,dim4d
   use ncdfout, only: define_mode,new_nc_variable,set_attributes,store_data
 endif
   IMPLICIT NONE
```

REVISION HISTORY:

```
   Original author(s): Karsten Bolding & Hans Burchard
```

LOCAL VARIABLES:

```
integer, save             :: x_excur_id,y_excur_id,n
integer                   :: i,iret
REALTYPE                  :: zz
REALTYPE, parameter       :: miss_val = -999.0
```

10.1.4 Finish the sea grass calculations

INTERFACE:

```
subroutine end_seagrass
```

DESCRIPTION:

Nothing done yet — supplied for completeness.

USES:

```
IMPLICIT NONE
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding & Hans Burchard
```

11 Running GOTM in a GUI

11.1 Introduction

11.2 Software required

12 GOTM scenarios

In this section, all scenarios included in the GOTM homepage for download are briefly discussed here. An overview is given in table 10. Information about how to install and run the scenarios can be found on the GOTM homepage, `www.gotm.net`.

Section	Title	Scenario name
12.1.1	Couette-flow	`couette`
12.1.2	Pressure-gradient driven channel flow	`channel`
12.1.3	Breaking surface-waves	`wave_breaking`
12.1.4	Some entrainment scenarios	`entrainment`
12.1.5	Estarine dynamics	`estuary`
12.1.6	seagrass canopy dynamics	`seagrass`
12.2.1	Fladenground Experiment	`flex`
12.2.2	Annual North Sea simulation	`nns_annual`
12.2.3	Seasonal North Sea simulation	`nns_seasonal`
12.2.4	Liverpool Bay	`liverpool_bay`
12.2.5	Gotland Deep in Baltic Sea	`gotland_deep`
12.2.6	Middelbank in Baltic Sea	`reynolds`
12.3.1	Ocean Weather Ship Papa	`ows_papa`
12.4.1	Lago Maggiore	`lago_maggiore`

Table 10: List of GOTM scenarios described in this section

12.1 Idealised scenarios

In this subsection, the performance of GOTM in some idealised turbulent flows is discussed. In these flows there are regions, where certain analytical solutions, like the law of the wall or the Rouse profile, apply. These solutions can be used to test the correctness of the implementation and the accuracy of the numerical schemes. The theoretical background is discussed in section 4 and in the review article of *Umlauf and Burchard* (2005).
The first few of these idealised flows serve also as a short tutorial for new GOTM users. We supplied several input files for these scenarios to illustrate the performance of different turbulence models for the same flow. It is recommended to start with the Couette-flow described next.

12.1.1 Couette-flow

This is the simplest example designed for new users. It will tell you about how to run a simple unstratified flow with the most frequently used turbulence models. The term Couette-flow flow traditionally denotes an uni-directional, unstratified, non-rotating flow confined between two plates, of which one is moving with constant velocity. No pressure-gradient is applied. It is clear that this flow can also serve as a very simple model of the steady-state flow in a horizontally infinite ocean of finite depth, driven solely by a shear-stress at the surface.

A set of GOTM input files (containing all specifications needed for the runs) has been provided for 3 different turbulence models in the sub-directories `kepsilon_nml/`, `komega_nml/` and `MellorYamada_nml/`. Copy all files from the subdirectory `kepsilon_nml/` to the directory with the GOTM executable. We will call this directory the *current directory* in the following. How to install GOTM and create the executable is described on the GOTM web page at `www.gotm.net`. Take some time to have a look at the contents of these files.
In our example, the prescribed surface stress is $\tau_x = 1.027$ Pa, a quantity that can be set in the input file `airsea.nml`. This file contains many other variables that are related to the air-sea fluxes driving the model.
Parameters concerning the run are set in the input file `gotmrun.nml`. There, you will find for example the specification of the water depth (10 m in this case) and the date and time of this run (24 hours until a steady-state is reached). The input file `gotmrun.nml` contains mainly parameters concerning the model run, the time step, the model time, the output format, etc.
All information about the turbulence models is read-in from the file `gotmturb.nml`. Having a look in this file, you see that we selected `tke_method = 2` and `length_scale_method = 8`, which corresponds exactly to the k-ϵ model described in section 4.15. The model parameters are given in the `keps` namelist. In this simple example, no Explicit Algebraic Stress Model (see section 4.2) is solved in addition to the transport equations for k and ϵ. If you compare this `gotmturb.nml` with those found in the other sub-directories (e.g. for the Mellor-Yamada model) it is easy to see how different turbulence models can be activated by changing e.g. the value for `length_scale_method`.
If you run this scenario, GOTM will write information about the run and the turbulence model to your screen: What are the parameters of the run, like time step, date, layers, etc? What are the model parameters of the turbulence model? What value has the von Kármán constant, κ? What value has the decay rate in homogeneous turbulence, d? And so on. All other output is written to files called `couette.out` or `couette.nc`, depending on whether you selected ASCII or NetCDF output in `gotmrun.nml`.
If you analyse the results, you will find that the turbulent kinetic energy is constant over the whole depth, whereas the profiles of the turbulent diffusivity and the length scale are approximately parabolic. The length scale approaches the constant slope $\kappa \approx 0.433$ near the boundaries. If you want to change this value, you can set `compute_kappa = .false.` in `gotmturb.nml`. Then, GOTM will automatically change the model constants of the k-ϵ model to compute the value of κ prescribed in `gotmturb.nml` (see section 4.7.4).
There are other models you can use to calculate the Couette-flow. If you copy all files from the directory `MellorYamada_nml/` to the current directory, GOTM will use the Mellor-Yamada model described in section 4.14 with parameters set in `gotmturb.nml`. A special role plays the so-called 'generic model' described in section 4.16. Other model like the k-ϵ model or the k-ω model by *Umlauf et al.* (2003) can be considered as special cases of the generic model. If you copy e.g. the files from `komega_nml/` to the current directory, the k-ω model is run for the couette case. For this simple flow, however, model results will be quite similiar in all cases.

12.1.2 Pressure-gradient driven channel flow

A pressure-gradient driven open channel flow is investigated here with a prescribed surface slope $\partial_x\zeta = -10^{-5}$ at a fixed water depth of 10 m. No surface stress is applied, and rotation and stratification are neglected. The simulation is run for 24 h until a steady-state is reached. The specification of all these parameters and those related to the turbulence models by use of the nml-files is analogous to section 12.1.1.
The surface slope is set in the namelist `ext_pressure` in the input file `obs.nml`. How the parameters given in this file are interpreted by GOTM is described in section 3.7 and briefly also in `obs.nml`. This file typically contains information about "observed" quantities that are used to

either force the model (like internal and external pressure gradients) or for comparision with computed results. In the latter case, “observed” quantities are displayed in the output file next to the computed quantities.
If you want to try out the different turbulence models mentioned in the couette-case (see section 12.1.1), simply copy the corresponding files from the respective subdirectories to the current directory with the GOTM executable. Note that in `gotmturb.nml` we now set `turb_method = 3`. This implies that the turbulent fluxes are computed from a second-order turbulence model. A new thing in GOTM 3.2 is that parameters for the second-order model can now be directly specified via the “scnd” namelist in `gotmturb.nml`. For the theoretical background of this, please see section 4.2
In the following publications some of the results in comparison to laboratory data are shown: *Burchard et al.* (1998), *Burchard et al.* (1999), *Burchard* (2002b). The simulation has been motivated by the work of *Baumert and Radach* (1992).

12.1.3 Turbulence under breaking surface waves

In this scenario, it is demonstrated how the effect of breaking surface waves is parameterised in one- and two-equation models. This is usually done by injecting turbulent kinetic energy (TKE) at the surface, see *Craig and Banner* (1994) and *Craig* (1996). The rate of TKE injected is proportional to the surface friction velocity cubed, as defined in (211). Injection of TKE at the surface leads to a thin surface boundary layer, in which the vertical transport of TKE and its dissipation approximately balance. This layer is sometimes called the *transport layer.* Even though there can be considerable shear in this layer, shear-production of turbulence is negligible by definition (also see section 4.7.4).
Different types of models are available in GOTM for the wave-breaking scenario. The key change in `gotmturb.nml` for runs with TKE injection is to set `ubc_type = 2`, telling GOTM to set the type of the upper boundary to TKE injection. The decay rates of the TKE and the dissipation rate in the wave-affected layer are then an natural outcome of the model. Note that with the KPP model, this scenario cannot be run.

- For the one-equation models, as discussed in *Craig and Banner* (1994), a linearly increasing macro length scale, l, is postulated with a slope of $\kappa = 0.4$. This is analogous to the law of the wall, even though there is no physical evidence for the assumption that the length-scale under breaking waves behaves identically as in wall-bounded shear-flows. As shown by *Craig and Banner* (1994), an analytical solution for the one-equation model can be derived, but only inside the transport layer, according to which the TKE (and all other turbulence quantities) follows a power-law (see discussion in section 4.7.3 and section 4.7.4).

  If you want to simulate wave breaking with a model of this type, simply copy all files from `prescribed_nml/` to the current directoy, and run GOTM. A dynamic equation for k is used, but the length scale is fixed, and prescribed by a triangular shape with slope κ (`length_scale_method = 2` in `gotmturb.nml`, see section 4.19).

- For two-equation models, the slope of the length scale in the transport layer is not simply prescribed and generally not equal to κ. *Umlauf et al.* (2003) generalized the solution of *Craig and Banner* (1994) and derived analytical solutions for the non-linear system of equations describing the behaviour of two-equation models for injection of TKE at the surface. They showed that the TKE follows a power-law and that the length scale increases linearly, however, with a slope $L \neq \kappa$. They also compared the spatial decay of turbulence in grid stirring experiments (thought as an analogy to wave-breaking) to the results of several two-equation models.

A numerical solution of the k-ϵ model can be obtained by copying the files in `kepspilon_nml` to the current directory, and insuring that `compute_kappa = .true.` and `sig_peps = .false.` in `gotmturb.nml`. Because the spatial decay rate of the TKE is very large for this model, the wave-affected layer is very small, and of the order of only a few tens of centimeters for this scenario. As discussed by *Umlauf et al.* (2003), this disadvantage can be overcome by using the k-ω model with parameters given in `gotmturb.nml` in the directory `komega_nml/`. The decay rates of this model nicely correspond to those measured in the laboratory grid strirring experiments. The Mellor-Yamada model has also been investigated by *Umlauf et al.* (2003), but for this model, again, decay was shown to be too strong. In addition, the decay rate depends in an unphysical way on the wall-function required in this model.

- As an alternative to the standard k-ϵ model, *Burchard* (2001a) suggested to make the turbulent Schmidt number for the ϵ-equation, (165), a function of the production-to-dissipation ratio, P/ϵ. As shown in detail in this paper, the variable Schmidt number can be used to "force" the k-ϵ model to compute κ for the slope of the length scale, even under breaking waves. Then, obviously, the solution of the k-ϵ model corresponds to the solution of the simpler one-equation model investigated by *Craig and Banner* (1994). Note again, however, that there is no physical evidence for $l = \kappa(z + z_0)$ in the wave-affected boundary layer.

  If you want to simulate wave breaking with this model, simply copy the files from `kepspilon_nml/` to the current directory, and make sure that you set `compute_kappa = .false.` and `sig_peps = .true.` in `gotmturb.nml`. Results are quite similar to those with the prescribed length scale.

- *Umlauf and Burchard* (2003) analysed the properties of a whole class of two-equation models for the case of TKE injection at the surface. They suggested a 'generic' model which could satisfy the power-laws under breaking waves for any desired decay rate, α, and length scale slope, L. This model is activated with the input files from `generic_nml/`. Users can select any reasonable values for α and L (and many others parameters like κ and d), and GOTM will automatically generate a two-equation model with exactly the desired properties. Parameters are computed according to the formulae described in section 4.7.3.

In all cases a surface-stress of $\tau_x = 1.027 \ \mathrm{N\,m^{-2}}$ was applied. After a runtime of 2 days, a steady-state with a constant stress over the whole water column of 20 m depth is reached. The wave affected layer can be found in the uppermost meter or so, and because of the strong gradients in this region we used a refined grid close to the surface. The parameters for such a 'zoomed grid' can be set in the input file `gotmmean.nml` according to the decription in section 3.3. If you want to compare the computed profiles with the analytical solutions in (110), you'll need a specification of the parameter K. This parameter is computed in `k_bc()` to be foundăin `turbulence.F90`, where you can add a few FORTRAN lines to write it out.

12.1.4 Some entrainment scenarios

This test case describes three idealised entrainment scenarios as discussed in the review paper of *Umlauf and Burchard* (2005). These are: wind-driven entrainment into a linearly stratified fluid, wind-driven entrainment into a two-layer fluid, and entrainment in free convection. As in the cases before, the input files for different turbulence closure models are contained in a number of sub-directories. The `entrainment` test cases is also the first test for the GOTM implementation of the KPP turbulence model described in section 4.35.
For all input files, the default is a linear density stratification due to a not necessarly linear temperature stratification (because the equation of state is not necessarily linear). The stratification

corresponds to $N^2 = 1 \cdot 10^{-4}$ s^{-2}. Salinity is constant. Have look into `obs.nml` to understand how different types of initial stratifcations can be specified in GOTM. The water depth is $H = 50$ m, deep enough for the surface induced mixing not to reach the bed within the 24 h of simulation. Rotation is neglected. By default, a constant wind stress of $\tau_x = 0.1027$ Pa is set in `airsea.nml`. Note, that for all turbulence models, except the Mellor-Yamada model, we set `compute_c3 = .true.` in `gotmturb.nml`, which means that the model constant $c_{\epsilon 3}$ in (165) (or its counterpart in all other models) is computed from a prescribed steady-state Richardson-number, Ri_{st} (see discussion in the context of (116)). Some more discussion is given in *Burchard and Bolding* (2001) and *Umlauf and Burchard* (2005). As pointed out in these papers, it is the value of the steady-state Richardson number (and thus the value of $c_{\epsilon 3}$) that determines the mixed layer depth in almost all shear-driven entrainment scenarios.
To run the Mellor-Yamada model, use the input files in `MellorYamada_nml/`. Looking at the results you will realize that this model is not at all in accordance with the experimental results of *Price* (1979) for the entrainment in a linearly stratified fluid. The reason can be traced back to the behaviour of the turbulent length scale in the strongly stratified thermocline. *Galperin et al.* (1988) suggested to clip the length scale at a certain value to circumvent this problem. Their solution can be activated by setting `length_lim = .true.` in `gotmturb.nml`. A second solution has been suggested by *Burchard* (2001b), who computed the model constant E_3 in (162) from the steady-state Richardson-number as described above. To activate this method, select `compute_c3 = .true.` (and `length_lim = .false.` because clipping is not needed any more).
This scenario has been used by us in several publications as a test for vertical mixing schemes, see *Burchard et al.* (1998), *Burchard et al.* (1999), *Burchard and Petersen* (1999), *Burchard and Bolding* (2001) *Burchard and Deleersnijder* (2001), *Deleersnijder and Burchard* (2003), and *Umlauf et al.* (2003).
The second entrainment scenario discussed in *Umlauf and Burchard* (2005) is essentially identical to the one just described, however, it starts from a two layer stratification. To use this kind of initial condition, first set `analyt_method=2` in `obs.nml`, and specify the desired temperatures, `t_1` and `t_2` for the upper and lower layer, respectively. The thickness of the upper layer is `z_t1`. For a pure two-layer stratification, set `z_t2=z_t1`, otherwise you will get a linear transition between the upper and the lower layer.
For convective entrainment, you simply need to set the momentum flux, `const_tx`, to zero and specify an appropriate negative heat flux, `const_heat`, in `airsea.nml`, see *Umlauf and Burchard* (2005).
If you run the KPP-model, some model parameters can be set in the extra input file `kpp.nml` found in kpp_nml/. With this model, the depth of the mixing layer depends mostly on the value of the critical bulk Richardson number that can also be set in this file. When you work with the KPP-model in free convection, don't forget to check if the pre-processor macro `NONLOCAL` is defined `cppdefs.h` (after changes in this file, don't forget to re-compile the whole code!). If `NONLOCAL` is defined, the KPP model also computes the non-local fluxes of heat (and salinity, if the salinity equation is active). In any case, the z-coordinate of the edges of the upper and lower mixing layers are given as `zsbl` and `zbbl`, respectively, in the netCDF output file.

12.1.5 Estuarine dynamics

In this idealised experiment, an estuarine circulation is simulated, mainly in order to demonstrate how to use tracer advection and internal pressure gradients in GOTM, but also to show the basic physical process of tidal asymmetries and its impact on SPM dynamics.
The average water depth is $H = 15$ m, the model is run for 10 days. The forcing is a M$_2$ tide (of period 44714 s) which prescribes sinusoidal time series for the vertically averaged momentum in west-east direction with an amplitude of 1.5 m s$^{-1$} and an offset of 0.05 m s^{-1} directed to the west

in order to simulate river run-off. The surface elevation is sinusoidal as well with an amplitude of 1 m and a phase shift of 1.5 hours compared to the velocity. A constant in time and space horizontal west-east salinity gradient of -0.0005 ppt m^{-1} is prescribed, advection of salinity is turned on. In order not to obtain negative salinities, relaxation to the initial salinity profile of 15 ppt is made. In order to avoid strong stratification near the surface, a small wind stress of 0.01027 $\mathrm{N/m}^2$ is applied.
A simple suspended matter equation with constant settling velocity is calculated in order to show the effect of tidal asymmetries on SPM transport. It is clearly seen that during flood SPM is mixed higher up into the water column than during ebb, an effect which has been described by *Geyer* (1993).
It is recommended to go through the description in the routines computing the external and internal pressure gradients, see section 3.7 and section 3.8, to understand the corresponding settings in the input file `obs.nml`. The relaxation scheme for salinity is described in section 3.11. Essential for this case is also the parametrisation of horizontal advection, which is set in `obs.nml` and described in section 3.11. Note that horizontal advection is calculated from the same horizontal salinity gradient that causes the internal pressure gradient.
The result is that estuarine circulation is set on and near bed residual velocity is directed upstream. It is interesting to have a look into the resulting buoyancy production or Brunt-Väisälä frequency. The effect of lateral advection on stratification leads to either production or supression of turbulence, and thus to an asymmetric time series of the turbulent diffusivity.
For two-dimensional simulations of estuarine circulation, see e.g. *Burchard and Baumert* (1998) and *Burchard et al.* (2004).

12.1.6 Seagrass canopy dynamics

The seagrass-current interaction has been successfully simulated by *Verduin and Backhaus* (2000) by means of coupling an ocean circulation model (HAMSOM) and a Lagrangian tracer model. The model set-up was basically two-dimensional with a vertical and a horizontal coordinate. A harmonic swell wave travelling into the direction of the positive x-coordinate had been specified at one open boundary.
The seagrass was represented by passive Lagrangian tracers which freely followed the flow as long as they were located inside prescribed excursion limits. The movement was simply frozen when the excursion limit was reached and the flow tendency was to carry them even further out. Only in that situation, the seagrass tracers had an effect on the current speed by exerting a quadratic friction on the flow.
The basic result of *Verduin and Backhaus* (2000) for a location inside the seagrass meadow was, that the mean kinetic energy had a local maximum just above the upper reach of the seagrass. That was found to be in good agreement with field measurements.
It is interesting to perform the following two experiments:

A: Now extra turbulence is produced by leaf-current friction, $\alpha = 0$.

B: All friction losses between leaves and current are converted to turbulence, $\alpha = 1$.

The results for these two experiments are shown in *Burchard and Bolding* (2000). The sensivity to α seems to be small, only the profiles of averaged turbulent kinetic energy are significantly influenced. The results of *Verduin and Backhaus* (2000) are basically reproduced. Especially, the local maximaum of mean kinetic energy just above the upper reach of seagrass is well simulated. The only striking difference is that in our model the seagrass shows an asymmetry for the excursion which is following the residual transport caused by the waves travelling from left to right.

Data files:

`seagrass.dat`	height above bed in m, excursion limit in m, friction coefficient in m^{-1}.

12.2 Shelf sea scenarios

All shelf sea scenarios briefly discussed here are from the Irish Sea and the North Sea. A Baltic Sea mixed layer scenario is in preparation.

12.2.1 Fladenground Experiment

A data set which has been used throughout the last 20 years as a calibration for mixing parameterisations has been collected during the measurements of the Fladenground Experiment 1976 (FLEX'76) campaign. These measurements of meteorological forcing and potential temperature profiles were carried out in spring 1976 in the northern North Sea at a water depth of about 145 m and a geographical position at 58°55'N and 0°32'E. The simulation is run from April 6 to June 7, 1976. The *Kondo* (1975) bulk formulae have been used for calculating the surface fluxes. For further details concerning the measurements, see *Soetje and Huber* (1980) and *Brockmann et al.* (1984). This FLEX'76 data set has been used by several authors in order to test different mixing schemes (see e.g. *Friedrich* (1983), *Frey* (1991), *Burchard and Baumert* (1995), *Pohlmann* (1997), *Burchard and Petersen* (1999), *Mellor* (2001)).

Data files:

`momentumflux.dat`	surface stress components, τ_x and τ_y in $\mathrm{N\,m}^{-2}$
`heatflux.dat`	solar radiation and outgoing heat flux, Q_{in} and Q_{out} in $\mathrm{W\,m}^{-2}$
`sst.dat`	observed SST in °C
`pressure.dat`	time series of surface slopes fitted to the local spring-neap cycle
`tprof.dat`	profiles of measured potential temperature for initial conditions and validation, data are reanalysed and low pass filtered
`sprof.dat`	profiles of idealised salinity for initial conditions and relaxation
`tprof_ctd`	CTD-profiles of potential temperature, with some gaps
`sprof_ctd`	CTD-profiles of salinity, with some gaps
`extinction.dat`	extinction coefficients fitted to measurements

12.2.2 Annual North Sea simulation

Here the annual simulation of the Northern Sea at 59°20" N and 1°17' E during the year 1998 as discussed by *Bolding et al.* (2002) is performed.
For this simulation, time series of surface slopes $\partial_x\zeta$ and $\partial_y\zeta$ were extrapolated from observations during autumn 1998 based on four partial tides by means of harmonic analysis (the program for doing this was kindly provided by Frank Janssen, now at the Baltic Sea Research Institute Warnemünde). All necessary meteorological data are from the UK Meteorological Office Model. For calculating the resulting surface fluxes, the bulk formulae from *Kondo* (1975) are used here. Since no observations for the sea surface temperature (SST) are available for the whole year 1998 at station NNS, the simulated SST is used as input into the bulk formulae. For the evolution of the vertical salinity profile, which is known to stabilise stratification during summer months, a relaxation to results obtained with a prognostic three-dimensional model of the North Sea by *Pohlmann* (1996). By doing so, the horizontal advection, which is the dominant process for salinity dynamics in the Northern North Sea, is parameterised.

Data files:

`sprof.dat`	salinity in ppt from three-dimensional model of *Pohlmann* (1996)
`tprof.dat`	potential temperature in °C from the three-dimensional model of *Pohlmann* (1996)
`pressure.dat`	sea surface slopes from tidal analysis of observations
`meteonns.dat`	meteorological data from UK Met Office model
`sst.dat`	sea surface temperature in °C from analysis by Bundesamt für Seeschifffahrt und Hydrographie, Hamburg, Germany

12.2.3 Seasonal North Sea simulation

This Northern North Sea Experiment has been carried out in the framework of the PROVESS (PROcesses of VErtical mixing in Shealf Seas) project (MAS3-CT97-0025, 1998-2001) which has been funded by the European Communities MAST-III program.
The observations in the Northern North Sea were carried out in September and October 1998. Here, a period of 20 days from October 7 - 27, 1998 is simulated. All forcing and validation data have been carefully processed from observations during this PROVESS-NNS experiment.
Two different dissipation rate data sets are included:

`eps_fly.dat`	data from a FLY profiler, processed by School of Ocean Sciences, University of Bangor, Wales
`eps_mst.dat`	data from an MST profiler, processed by JRC, Ispra, Italy.

These files can be read in into GOTM through the `eobs` namelist in `obs.nml`. The dissipation rate has only been observed at short intervals, periods without observations are set to minimum values in the files. These dissipation rate observations are read in into GOTM in order to allow for proper interpolation to the temporal and spatial output steps, and they are not used for any type of nudging.
The data files are prepared such that the maximum simulation interval can be extended to September 7 at 10.00 h – November 2 at 13.00 h, 1998.
For details on dissipation rate data processing, see *Prandke et al.* (2000).
For discussions of various model simulations, see *Burchard et al.* (2002) and also the annual simulation in section 12.2.2 and *Bolding et al.* (2002).

Other data files:

`sprof.dat`	salinity in ppt from CTDs and microstructure shear probes from several ships
`tprof.dat`	potential temperature in °C from CTDs and microstructure shear probes from several ships
`pressure.dat`	sea surface elevation gradients analysed from a triangle of pressure gauges
`w_adv.dat`	vertical velocities analysed from vertical displacement of pycnocline
`velprof.dat`	horizontal velocities from bottom mounted ADCP
`meteo_dana.dat`	meteorological observations from R/V Dana, only used for `calc_fluxes=.true.`

12.2.4 Liverpool Bay

The observations for this scenario have been carried out by *Rippeth et al.* (2001) in the Liverpool Bay ROFI on July 5 and 6, 1999 at a position of 53°28.4'N, 3°39.2'W. This period is about three days after spring tide, with calm weather and clear sky. The dissipation rate measurements were carried out with a FLY shear probe mounted on a free-falling profiler. Sensors for temperature and conductivity attached to the profiler give detailed information on the vertical density distribution during each cast. Nearby, an ADCP was mounted on the bottom, giving information on the vertical velocity structure. Some accompanying CTD casts were made in order to achieve estimates for the horizontal gradients of temperature and salinity. For further details concerning the observations, see *Rippeth et al.* (2001).
The surface fluxes are based on ship observations and from a nearby meteorological station at Hawarden. From the ship, wind speed and direction at 10 m above the sea surface and air pressure have been taken. From Hawarden station, observations of dry air, wet bulb and dew point temperature are used. Since the surface fluxes are calculated externally by means of bulk formulae of *Kondo* (1975), the sea surface temperature from measurements (FLY profiler) has been used. The bed roughness has been estimated from near-bed ADCP measurements as $z_0^b \approx 0.0025$ m by means of fits to the law of the wall. The external pressure gradient due to surface slopes is estimated according to a method suggested by *Burchard* (1999) by means of adjustment to near bed velocity observations. The CTD casts carried out during the campaign did only allow for rough estimates of the horizontal density gradient. The horizontal salinity and temperature gradients for a typical summer situation have been estimated by *Sharples* (1992) to $\partial_s S = 0.0425\ \mathrm{ppt\,km}^{-1}$ and $\partial_s T = -0.0575\ \mathrm{K\,km}^{-1}$, respectively. Here, s is the gradient into the direction $\alpha = 78°$ rotated anti-clockwise from North.

Data files:

`sprof.dat`	salinity in ppt from free-falling shear-probe
`tprof.dat`	potential temperature in ° from free-falling shear-probe
`pressure.dat`	near-bed velocity from ADCP for external pressure forcing
`zeta.dat`	sea surface elevation from pressure gauge
`velprof.dat`	horizontal velocities from bottom mounted ADCP
`eprof.dat`	observed dissipation rates in $\mathrm{W\,kg}^{-1}$
`heatflux.dat`	surface heat fluxes calculated accroding to *Kondo* (1975)
`momentumflux.dat`	surface momentum fluxes calculated according to *Kondo* (1975)

The numerical simulations of this scenario has been described in *Simpson et al.* (2002).

12.2.5 Gotland Deep

These simulations are made for the location of station 271 Central Eastern Gotland Sea of the Baltic Sea at 20 E and 57.3 N with a water depth of about 250 m. Initial conditions for temperature and salinity are derived from measurements. Meteorological forcing was available from the ERA15 reanalysis data set (http://wms.ecmwf.int/research/era/Era-15.html). For the penetration of solar radiation into the water column, fairly turbid water (Jerlov type IB) is assumed. Salinity concentrations are nudged to observations with a time scale of $\tau_R = 2$ days.
For the comparison of simulated temperature and salinity and observations we have used mainly data from the COMBINE program. All environmental monitoring within HELCOM and the Baltic marine environment is carried out under the COMBINE program. The COMBINE program runs under the umbrella of HELCOM. HELCOM is the governing body of the *Convention on the Protection of the Marine Environment of the Baltic Sea Area* - more usually known as the Helsinki Com-

mission (`www.helcom.fi`). In a regular schedule data from stations in the Baltic Sea are collected. Parts of these data are maintained inter alia at the Baltic Sea Research Institute Warnemünde and can be used for scientific work.
Model results and observations are compared for the years 1994-1996. For the discretisation, the water column has been divided into 100 vertical layers, with a strong zooming towards the surface, resulting in a mean near-surface resolution of less than 0.5 m. The time step for these simulations is set to $\Delta t = 1$ hour.

Data files:

`meteo.dat`	meteorological data extracted from the ERA15 reanalysis data set
`sprof_271.dat`	deep salinity profiles at station 271
`sprof_271_all.dat`	all salinity profiles at station 271
`sprof_GB.dat`	all salinity profiles in Gotland basin, within 57°8.3'N - 57°28.3'N and 19°54.6'E - 20°14.6'E
`tprof_271.dat`	deep temperature profiles at station 271
`tprof_271_all.dat`	all temperature profiles at station 271
`tprof_GB.dat`	all temperature profiles in Gotland basin, within 57°8.3'N - 57°28.3'N and 19°54.6'E - 20°14.6'E

The meteorological data have been compiled by Frank Janssen (IOW, Baltic Sea Research Institute Warnemünde, Germany), and the temperature and salinity profiles have been collected from the IOW data bank by Iris Theil (University of Hamburg, Germany).
These data have been used for simulating the Gotland Deep ecosystem dynamics for the years 1983-1991, see *Burchard et al.* (2006), see also section 12.5.4.

12.2.6 Middelbank

Here a campaign (REYNOLDS, funded by the German Federal Ministry for Education and Research, chief-scientist Hans Ulrich Lass, IOW) in the Eastern Bornholm Basin (55° 35' N, 16° 39' E, mean water depth: 55 m) is simulated. The simulation period is August 30, 2001 at 17 h to September 9, 2001 at 14 h. The water column is characterised by a thermocline at about 25 m depth and a halocline at about 50 m depth. The simulation period is charaterised by storms up to 0.2 $\mathrm{N\,m^{-2}}$. As forcing, surface stress, heat fluxes and solar radiation has been calculated on the basis of meteorological observations according to *Kondo* (1975). The barotropic pressure gradient has been recalculated from vertically averaged observed velocity profiles, see section 3.1.1. As initial conditions, observed temperature, salinity and velocity profiles are used. Additionally the vertical velocity at the thermocline has been diagnosed from temperature observations and is used for vertical advection, see section 3.1.1. The turbulent dissipation rate ε has been observed during two sub-periods, such that turbulence model results may be compared with observations.

Data files:

`eprof.dat`	profiles of observed dissipation rate in $\mathrm{W\,kg^{-1}}$
`heatflux.dat`	surface heat flux and solar radiation in $\mathrm{W\,m^{-2}}$
`momentumflux.dat`	surface momentum flux in $\mathrm{N\,m^{-2}}$
`pressure.dat`	vertically averaged velocity components in $\mathrm{m\,s^{-1}}$
`sprof.dat`	profiles of observed salinity in psu
`sss.dat`	time series of sea surface salinity in psu
`sst.dat`	time series of sea surface temperature in °C
`tprof.dat`	profiles of observed temperature in °C
`velprof.dat`	profiles of observed velocity components in $\mathrm{m\,s^{-1}}$
`vertvel.dat`	profiles of diagnosed vertical velocity at thermocline depth in $\mathrm{m\,s^{-1}}$

So far, these data have not yet been published.

12.3 Open ocean scenarios

The two open ocean scenarios introduced here are two classical test cases from the Northern Pacific Ocean. For an overview, see *Martin* (1985).

12.3.1 Ocean Weather Ship Papa

This scenario is a classical scenario for the Northern Pacific, for which long term observations of meteorological parameters and temperature profiles are available. The station Papa at 145°W, 50°N has the advantage that it is situated in a region where the horizontal advection of heat and salt is assumed to be small. Various authors used these data for validating turbulence closure schemes (*Denman* (1973), *Martin* (1985), *Gaspar et al.* (1990), *Large et al.* (1994), *Kantha and Clayson* (1994), *d'Alessio et al.* (1998), *Burchard et al.* (1999), *Villarreal* (2000), *Axell and Liungman* (2001), *Burchard and Bolding* (2001)).
The way how bulk formulae for the surface momentum and heat fluxes have been used here is discussed in detail in *Burchard et al.* (1999).
For mixing below the thermocline, an internal wave and shear instability parameterisation as suggested by *Large et al.* (1994) has been used. The maximum simulation time allowed by the included surface forcing file and the temperature profile file is January 1 (17.00 h), 1960 - December 31 (12.00 h), 1968. In this scenario, the simulation time is run from March 25, 1961 (0.00 h) to March 25, 1962 (0.00 h).

Data files:

`sprof.dat`	salinity profiles in ppt of monthly climatology from Levitus data set. First profile interpolated to January, 1st
`tprof.dat`	profiles of measured potential temperature for initial conditions and relaxation
`heatflux.dat`	surface heat fluxes calculated according to *Kondo* (1975)
`momentumflux.dat`	surface momentum fluxes calculated according to *Kondo* (1975)

This scenario has been discussed in detail by *Burchard et al.* (1999). We are grateful to Paul Martin for providing the meteorological data and the temperature profiles, see also *Martin* (1985).

12.4 Lake scenarios

So far, the Lago Maggiore scenario discussed in section 12.4.1 is the only lake scenario.

12.4.1 Lago Maggiore

The measurements for this Lago Maggiore scenario were made during three days in winter 1995 (December 18-21) at the shore of Ispra (45° 49,244'N, 8° 36,377'E). The measurements were carried out with an uprising profiler located 150 m from the shore at a water depth of 42 m. Such the sampled depth interval ranged from 30 m up to the surface. On the profiler, an MST shear probe, a fast temperature sensor and temperature and conductivity probes were mounted such that profiles of turbulent dissipation rate ϵ, temperature variance ϵ_θ, mean temperature θ and mean salinity S could be derived. For a detailed description of the data analysis, see *Stips et al.* (2002).
Wind speed was measured from a small buoy about 30 m away from the probe location with an anemometer at a height of 95 cm above the water surface. The accuracy is ± 0.1 m s^{-1}. Air temperature and relative humidity were recorded at the measurement location on shore at a height of 10 m above lake surface. The cloud cover has been estimated every hour. Incident solar radiation was measured at the meteorological station in Pallanza, in a distance of about 10 km from the measuring site. An analysis of heat fluxes obtained by various bulk formulae showed however a significant deviation between the heat content of the water column and accumulation of these heat fluxes. This could be due to the fact that these bulk formulae are designed for oceanic conditions such that they are not valid for a lake with weak wind conditions. Thus, instead of using the calculated surface heat fluxes from bulk formulae, they were calculated from the heat gain of the water column under consideration of the solar radiation.

Data files:

`salz_lmd95.dat`	profiles of measured salinity in ppt for initial conditions and relaxation
`temp_lmd95.dat`	profiles of measured potential temperature for initial conditions and relaxation
`eps_lmd95.dat`	profiles of measured dissipation rate for validation
`hflu2_05lt.dat`	surface heat fluxes calculated according to *Kondo* (1975)
`momentumflux.dat`	surface momentum fluxes calculated according to *Kondo* (1975)

For a discussion of the simulation, see *Stips et al.* (2002).

12.5 Biogeochemical scenarios

The functionality of biogeochemical models in GOTM is demonstrated here for four different scenarios:

- The idealised channel flow scenario introduced in section 12.1.2 is here complemented with a simple suspended matter equation, see section 12.5.1.
- The PROVESS Northern North Sea scenario (see section 12.2.2) is here coupled to the NPZD model, see section 12.5.2.
- The Fladenground Experiment (FLEX'76), see section 12.2.1 is here coupled to the *Fasham et al.* (1990) biogeochemical model, see section 12.5.3.
- The multiannual Gotland Deep scenario (see section 12.2.5) is here coupled to the biogeochemical model by *Neumann et al.* (2002), with adaptations by *Burchard et al.* (2006), see section 12.5.4.

12.5.1 Channel flow - Rouse profile

In this scenario, the water depth and the surface slope have been chosen identical to those in the purely physical `channel` scenario introduced in section 12.1.2.
Under certain conditions, the suspended matter equation

$$\partial_t C + \partial_z \left(w_c C - \nu_t \partial_z C \right) = 0, \tag{275}$$

has an analytical solution:

- Constant settling velocity w_c
- Parabolic eddy diffusivity ν_t
- Reflective bottom and surface
- Steady-state solution
- No sources and sinks

Let the eddy diffusivity profile be parabolic with

$$\nu_t = \kappa u_* (-z) \frac{D + z_0 + z}{D + z_0} \tag{276}$$

with the depth D, the bottom roughness length z_0, the van Karman number κ and the bottom friction velocity u_* and the vertical coordinate z. Then the analytical solution of (275) is

$$\frac{C}{C_0} = \left(\frac{-z}{D + z_0 + z} \right)^{-w_c/(\kappa u_*)}, \tag{277}$$

where C_0 is the suspended matter concentration at $z = -(D + z_0)/2$i and depends on the initial conditions for C. The Rouse number is then defined as:

$$R = \frac{-w_c}{u_*}. \tag{278}$$

When running the `rouse` scenario with a two-equation turbulence closure model, then the analytical solution for the Rouse profile is only approximated, since the eddy diffusivity deviates from (277). In order to be closer to the analytical solution, it is ncessary to chose in `gotmturb.nml` the analytical parabolic profile for eddy diffusivity, which is done by the follwing settings in `gotmturb.nml`:

```
&turbulence
 turb_method=     2,
 tke_method=      1,
 len_scale_method=4,
 stab_method=     1
```

This Rouse scenario may be calculated with Eulerian concentrations or with Lagrangian particles, depending on the setting of `bio_eulerian` in `bio.nml`. When using the Lagrangian particle method, it is advisable to average the concentration over all time steps which belong to one output time step, by setting `bio_lagrange_mean=.true.`
It is also possible to include a sink term at the bed (for simulating the effect of grazing by benthic filter feeder), the seetings for this have to be made in `mussels.nml`.

12.5.2 Northern North Sea - NPZD model

In order to provide a typical and clearly defined physical environment for testing the NPZD model discussed in section **??**, we use an annual simulation of the water column in the Northern North Sea. The setup is similar to the one described in section 12.2.2 and validated in detail by *Bolding et al.* (2002). The photosynthetically available radiation (PAR) has been simply taken as visible part of the short-wave radiation, a feedback of increased turbidity due to plankton blooms to the light absorption has not been considered for the heat budget. For further details of the physical model setup, see section 12.2.2.
As shown by *Burchard et al.* (2005), the explicit ODE solvers introduced in section 8.13 do not guarantee non-negative solutions for biogeochemical concentrations when the biogeochemical model is stiff, i.e. when time scales are involved which may be shorter than the time step. In order to make the NPZD model stiff, *Burchard et al.* (2005) chose a half-saturation nutrient concentration α of 0.02 mmol N m^{-3}, whereas the typical values for α would be between 0.2 and 1.5 mmol N m^{-3}. This has the consequence that nutrient is taken up by phytoplankton even at low concentrations, which strongly decreases the time scale of this process. The overall phytoplankton evolution over an annual cycle is not much affected by this manipulation, except from the fact that now the summer surface nutrient concentrations are much lower. It should be noted that such low half saturation concentrations for nutrients have actually been observed in the oceanic mixed layer. *Harrison et al.* (1996) calculated for the mixed layer of the North Atlantic mean half saturation concentrations for nitrate and ammonium as small as 0.02 mmol N m^{-3}.
In order to demonstrate the advantages of the Modified Patankar schemes over the fully explicit schemes, the simulation carried out here is based on a time step of 2 h for the physical part. For the biogeochemical part, a time splitting is used (set `split_factor` in `bio.nml`) such that fractional time steps are possible for the biogeochemical part. Thus, by using a time step of $\Delta^{phy}t = 7200$ s for the physical part and iterating the biogeochemical part 1,4 and 36 times per physical time step with unchanged physical forcing, biogochemical time steps of $\Delta^{bio}t = 7200$ s, $\Delta^{bio}t = 1800$ s and $\Delta^{bio}t = 200$ s, respectively, can be obtained. By doing so, it is possible to use exactly the same physical forcing for all ODE solvers and all biogeochemical time steps.
When using the explicit ODE solver (to do so, set `ode_method` in `bio.nml` to 1, 2 or 3), the summer nutrient concentration goes down to negative values. This does not happen for the other conservative methods 7 and 8 (first- and second-order Modified Patankar schemes) and 10 and 11 (first- and second-oder Extended Modified Patankar schemes).

12.5.3 Fladenground Experiment - *Fasham et al.* (1990) model

During the Fladenground Experiment 1976, extensive biogeochemical observations have been carried out as well. Here, the modelling work of *Kühn and Radach* (1997) will be reproduced with GOTM. All GOTM modelling details have been documented by *Burchard et al.* (2006).
The bloom at the Fladenground started with the onset of thermal stratification of the water column at 19 April (Julian day 110), reached its maximum at 1 May with an depth-integrated phytoplankton biomass of about 11 g C m^{-2} (*Radach et al.* (1980)). At 16 May the phytoplankton stock reached again the pre-bloom level. This main bloom was dominated by diatoms; flagellates constituted a smaller secondary bloom some weeks later. In accordance with the rapid production of organic matter the nitrate pool was depleted: from 8 mmol N m^{-3} before the bloom to less than 0.1 mmol N m^{-3} at the end of the bloom (*Brockmann et al.* (1983)). The observed average daily primary production during the bloom was 1.2 g C m^{-2} d^{-1} (*Weigel and Hagmeier* (1980)). The part of organic substance settling to the bottom was estimated from sediment trap measurements to about 20 ± 10 % of the primary production during that period (*Davies and Paine* (1984), *Radach et al.* (1984)). The remaining 80 % were obviously used by the zooplankton and partly

– via dissolution of the PON – also by bacteria. From estimates of the grazing pressure of the dominant copepod, *Calanus finmarchicus* (see *Krause and Radach* (1980)) it was concluded that additionally herbivorous microzooplankton must have played an essential role in grazing on the phytoplankton stock. The role of advection on the ecosystem dynamics was discussed by *Eberlein et al.* (1980). Thus the total FLEX '76 period could be divided into two phases: the first 6 weeks (until 6 May) with only weak horizontal advection, and the following 4 weeks with distinctively larger influence of advection on the nutrient concentrations.
This biogeochemical scenario has been reproduced by *Kühn and Radach* (1997) by means of the *Fasham et al.* (1990) biogeochemical model (see also section **??**) together with a one-equation turbulence closure model. *Burchard et al.* (2006), from which most of the present section has been adapted documented the implementation of this model into GOTM.

12.5.4 Gotland Deep - *Neumann et al.* (2002) model

The Gotland Deep scenario (Central Baltic Sea) which has been described in section 12.2.5 has been used for demonstrating the implementation of the *Neumann et al.* (2002) biogeochemical model into GOTM, see *Burchard et al.* (2006). For the description of the implementation into GOTM, see section **??**.
Some adaptations to the *Neumann et al.* (2002) had to be made, in order to provide acceptable results for the ecosystem simulations in the Gotland Deep. The the maximum growth rate of diatoms from a value of $r_1^{\max} = 1.5$ d^{-1} to $r_1^{\max} = 2.0$ d^{-1}. This is due to the fact that the simple turbulence closure scheme (*Pacanowsci and Philander* (1981)) used by *Neumann et al.* (2002) mixed substantially less than the k-ε used here. The surface fluxes of nutrients have been calibrated in such a way that winter nutrient concentrations are close to observations. By doing do, the effect of lateral nutrient transport is parameterised.
For further details, see *Burchard et al.* (2006).

References

Axell, L., and O. Liungman, A one-equation turbulence model for geophysical applications: Comparison with data and the k-epsilon model, *Environmental Fluid Mechanics*, *1*, 71–106, 2001.

Baumert, H., and H. Peters, Second-moment closures and length scales for weakly stratified turbulent shear flows, *J. Geophys. Res.*, *105*(C3), 6453–6468, 2000.

Baumert, H., and G. Radach, Hysteresis of turbulent kinetic energy in nonrotational tidal flows, *J. Geophys. Res.*, *97*(C), 3669–3677, 1992.

Beckers, J.-M., La méditerranée occidentale: de la modélisation mathématique à la simulation numérique, Ph.D. thesis, Université de Liège, Belgium, collection des publications de la Faculté des Sciences Appliquées No. 136, 1995.

Berliand, M. E., and T. G. Berliand, Measurement of the effective radiation of the earth with consideration of the effect of cloudiness (in russian), *Izv. Akad. Nauk SSSR, Ser. Geofiz.*, *1*, 1952.

Bignami, F., S. Marullo, R. Santoleri, and M. E. Schiano, Long-wave radiation budget in the Mediterranean Sea, *J. Geophys. Res.*, *100*, 2501–2514, 1995.

Blackadar, A. K., The vertical distribution of wind and turbulent exchange in a neutral atmosphere, *J. Geophys. Res.*, *67*, 3095–3102, 1962.

Bolding, K., H. Burchard, T. Pohlmann, and A. Stips, Turbulent mixing in the Northern North Sea: a numerical model study, *Cont. Shelf Res.*, *22*, 2707–2724, 2002.

Bradshaw, P., *An Introduction to Turbulence and its Measurement*, Pergamon, 1975.

Briggs, D. A., J. H. Ferziger, J. R. Koseff, and S. G. Monismith, Entrainment in a shear-free turbulent mixing layer, *J. Fluid Mech.*, *310*, 215–241, 1996.

Brockmann, U. H., V. Ittekott, G. Kattner, K. Eberlein, and K. D. Hammer, Release of dissolved organic substances in the course of phytoplankton blooms, in *North Sea Dynamics*, edited by J. Sündermann and W. Lenz, pp. 530–548, Springer, 1983.

Brockmann, U. H., K. Eberlein, K. Huber, H.-J. Neubert, G. Radach, and K. Schulze (Eds.), *JONSDAP '76: FLEX/INOUT Atlas, Vol. 1*, no. 63 in ICES Oceanographic Data Lists and Inventories, 450 pp. pp., Conseil International pour l'Exploration de la Mer, Copenhagen, Denmark, 1984.

Bruggeman, J., H. Burchard, B. Kooi, and B. Sommeijer, A second-order, unconditionally stable, mass-conserving integration scheme for biochemical systems, *Appl. Num. Math*, *57*, 36–58, 2006.

Burchard, H., Recalculation of surface slopes as forcing for numerical water column models of tidal flow, *App. Math. Modelling*, *23*, 737–755, 1999.

Burchard, H., Simulating the wave-enhanced layer under breaking surface waves with two-equation turbulence models, *J. Phys. Oceanogr.*, *31*, 3133–3145, 2001a.

Burchard, H., Note on the q^2l equation by Mellor and Yamada [1982], *J. Phys. Oceanogr.*, *31*, 1377–1387, 2001b.

Burchard, H., Energy-conserving discretisation of turbulent shear and buoyancy production, *Ocean Modelling*, *4*, 347–361, 2002a.

Burchard, H., *Applied Turbulence Modelling in Marine Waters*, no. 100 in Lecture Notes in Earth Sciences, Springer, 2002b.

Burchard, H., and H. Baumert, On the performace of a mixed-layer model based on the k-ϵ turbulence closure, *J. Geophys. Res. (C5)*, *100*, 8523–8540, 1995.

Burchard, H., and H. Baumert, The formation of estuarine turbidity maxima due to density effects in the salt wedge. A hydrodynamic process study, *J. Phys. Oceanogr.*, *28*, 309–321, 1998.

Burchard, H., and K. Bolding, Implementation of the Verduin and Backhaus seagrass-current interaction into the General Ocean Turbulence Model (GOTM). A short feasability study, unpublished manuscript, 2000.

Burchard, H., and K. Bolding, Comparative analysis of four second-moment turbulence closure models for the oceanic mixed layer, *J. Phys. Oceanogr.*, *31*, 1943–1968, 2001.

Burchard, H., and E. Deleersnijder, Stability of algebraic non-equilibrium second-order closure models, *Ocean Modelling*, *3*, 33–50, 2001.

Burchard, H., and O. Petersen, Hybridisation between σ and z coordinates for improving the internal pressure gradient calculation in marine models with steep bottom slopes, *Int. J. Numer. Meth. Fluids*, *25*, 1003–1023, 1997.

Burchard, H., and O. Petersen, Models of turbulence in the marine enviroment - a comparative study of two-equation turbulence models, *J. Mar. Syst.*, *21*, 29–53, 1999.

Burchard, H., O. Petersen, and T. P. Rippeth, Comparing the performance of the Mellor-Yamada and the $k-\epsilon$ two-equation turbulence models, *J. Geophys. Res. (C5)*, *103*, 10,543–10,554, 1998.

Burchard, H., K. Bolding, and M. R. Villarreal, GOTM – a general ocean turbulence model. Theory, applications and test cases, *Tech. Rep. EUR 18745 EN*, European Commission, 1999.

Burchard, H., K. Bolding, T. P. Rippeth, A. Stips, J. H. Simpson, and J. Sündermann, Microstructure of turbulence in the Northern North Sea: A comparative study of observations and model simulations, *Journal of Sea Research*, *47*, 223–238, 2002.

Burchard, H., E. Deleersnijder, and A. Meister, A high-order conservative Patankar-type discretisation for stiff systems of production-destruction equations, *47*, 1–30, 2003.

Burchard, H., K. Bolding, and M. R. Villarreal, Three-dimensional modelling of estuarine turbidity maxima in a tidal estuary, *Ocean Dynamics*, *54*, 250–265, 2004.

Burchard, H., E. Deleersnijder, and A. Meister, Application of Modified Patankar schemes to stiff biogeochemical models for the water column, *Ocean Dynamics*, *55*, 326–337, 2005.

Burchard, H., K. Bolding, W. Kühn, A. Meister, T. Neumann, and L. Umlauf, Description of a flexible and extendable physical-biogeochemical model system for the water column, *61*, 180–211, 2006.

Canuto, V. M., A. Howard, Y. Cheng, and M. S. Dubovikov, Ocean turbulence. Part I: One-point closure model—momentum and heat vertical diffusivities, *J. Phys. Oceanogr.*, *31*(6), 1413–1426, 2001.

Charnock, H., Wind stress on a water surface, *Q. J. R. Meteorol. Soc.*, *81*, 639–640, 1955.

Cheng, Y., V. M. Canuto, and A. M. Howard, An improved model for the turbulent PBL, *J. Atmos. Sci.*, *59*, 1550–1565, 2002.

Clark, N. E., L. Eber, R. M. Laurs, J. A. Renner, and J. F. T. Saur, Heat exchange between ocean and atmoshere in the Eastern North Pacific for 1961-1971, *Tech. Rep. NMFS SSRF-682*, NOAA, U.S. Dept. of Commerce, Washington, D.C., 1974.

Craft, T. J., N. Z. Ince, and B. E. Launder, Recent developments in second-moment closure for buoyancy-affected flows, *Dynamics of Atmospheres and Oceans*, *23*, 99–114, 1996.

Craig, P. D., Velocity profiles and surface roughness under breaking waves, *J. Geophys. Res.*, *101*, 1265–1277, 1996.

Craig, P. D., and M. L. Banner, Modeling wave-enhanced turbulence in the ocean surface layer, *J. Phys. Oceanogr.*, *24*, 2546–2559, 1994.

Crank, J., and P. Nicolson, A practical method for numerical evaluation of solutions of partial differential equations of the heat-conduction type, *Proc. Cambridge Philos. Soc.*, *43*, 50–67, re-published in: John Crank 80th birthday special issue *Adv. Comput. Math.* **6** (1997) 207-226, 1947.

d'Alessio, S. J. D., K. Abdella, and N. A. McFarlane, A new second-order turbulence closure scheme for modeling the oceanic mixed layer, *J. Phys. Oceanogr.*, *28*, 1624–1641, 1998.

Davies, J. M., and R. Paine, Supply of organic matter in the northern North Sea during a spring phytoplankton bloom, *78*, 315–324, 1984.

Deleersnijder, E., and H. Burchard, Reply to Mellor's comments on *stability of algebraic non-equilibrium second-order closure models*, *Ocean Modelling*, *5*, 291–293, 2003.

Demirov, E., W. Eifler, M. Ouberdous, and N. Hibma, Ispramix — a three–dimensional free surface model for coastal ocean simulations and satellite data assimilation on parallel computers, *Tech. Rep. EUR 18129 EN*, European CommissionJoint Reseach Center, Ispra, Italy, 1998.

Denman, K. L., A time-dependent model of the upper ocean, *J. Phys. Oceanogr.*, *3*, 173–184, 1973.

Domaradzki, J. A., and G. L. Mellor, A simple turbulence closure hypothesis for the triple velocity correlation functions in homogeneous isotropic turbulence, *J. Fluid Mech.*, *140*, 45–61, 1984.

Durksi, S. M., S. M. Glenn, and D. Haidvogel, Vertical mixing schemes in the coastal ocean: Comparision of the level 2.5 Mellor-Yamada scheme with an enhanced version of the K profile parameterization, *J. Geophys. Res.*, *109*(C01015), doi:10.1029/2002JC001702, 2004.

Eberlein, K., G. Kattner, U. Brockmann, and K. Hammer, Nitrogen and phosphorus in different water layers at the central station during FLEX '76, *"Meteor"-Forsch.-Ergebnisse, Reihe A*, *22*, 87–98, 1980.

Eifler, W., and W. Schrimpf, Ispramix, a hydrodynamic program for computing regional sea circulation patterns and transfer processes, *Tech. Rep. EUR 14856 EN*, European Commission Joint Reseach Center, Ispra, Italy, 1992.

Fairall, C. W., E. F. Bradley, J. Godfrey, G. A. Wick, J. B. Edson, and G. Young, Cool-skin and warm-layer effects on sea surface temperature, *J. Geophys. Res.*, *101*, 1295–1308, 1996a.

Fairall, C. W., E. F. Bradley, D. P. Rogers, J. B. Edson, and G. S. Young, Bulk parameterization of air-sea fluxes for TOGA-COARE, *J. Geophys. Res.*, *101*, 3747–3764, 1996b.

Fasham, M. J. R., H. W. Ducklow, and S. M. McKelvie, A nitrogen-based model of plankton dynamics in the oceanic mixed layer, *J. Mar. Res.*, *48*, 591–639, 1990.

Feistel, R., A new extended Gibbs thermodynamic potential of seawater, *Prog. Oceanogr.*, *58*, 43–115, http://authors.elsevier.com/sd/article/S0079661103000880 corrigendum 61 (2004) 99, 2003.

Fofonoff, N. P., and R. C. Millard, Algorithms for the computation of fundamental properties of seawater, *Unesco technical papers in marine sciences*, *44*, 1–53, 1983.

Frey, H., A three-dimensional, baroclinic shelf sea circulation model — 1. The turbulence closure scheme and the one-dimensional test model, *Cont. Shelf Res.*, *11*(4), 365–395, 1991.

Friedrich, H., Simulation of the thermal stratification at the FLEX central station with a one-dimensional integral model, in *North Sea Dynamics*, edited by J. Sündermann and W. Lenz, pp. 396–411, Springer, 1983.

Galperin, B., L. H. Kantha, S. Hassid, and A. Rosati, A quasi-equilibrium turbulent energy model for geophysical flows, *J. Atmos. Sci.*, *45*(1), 55–62, 1988.

Gaspar, P., Y. Gregoris, and J. Lefevre, A simple eddy kinetic energy model for simulations of the oceanic vertical mixing: Tests at station Papa and long-term upper ocean study site, *J. Geophys. Res.*, *95*, 16,179–16,193, 1990.

Gerz, T., U. Schumann, and S. E. Elghobashi, Direct numerical simulation of stratified homogeneous turbulent shear flows, *J. Fluid Mech.*, *200*, 563–594, 1989.

Geyer, W. R., The importance of suppression of turbulence by stratification on the estuarine turbidity maximum, *Estuaries*, *16*, 113–125, 1993.

Gibson, M. M., and B. E. Launder, On the calculation of horizontal, turbulent, free shear flows under gravitational influence, *J. Heat Transfer*, *98C*, 81–87, 1976.

Gibson, M. M., and B. E. Launder, Ground effects on pressure fluctuations in the atmospheric boundary layer, *J. Fluid Mech.*, *86*, 491–511, 1978.

Harrison, W. G., L. Harris, and B. D. Irwin, The kinetics of nitrogen utilization in the oceanic mixed layer: Nitrate and ammonium interactions at nanomolar concentrations, *41*, 16–32, 1996.

Hastenrath, S., and P. J. Lamb, Heat budget atlas of the tropical Atlantic and Eastern Pacific Oceans, *Tech. rep.*, University of Wisconsin, Madison, 1978.

Holt, S. E., J. R. Koseff, and J. H. Ferziger, A numerical study of the evolution and structure of homogeneous stably stratified sheared turbulence, *J. Fluid Mech.*, *237*, 499–539, 1991.

Jackett, D. R., T. J. McDougall, R. Feistel, D. G. Wright, and S. M. Griffies, Updated algorithms for density, potential temperature, conservative temperature and freezing temperature of seawater, *Journal of Atmospheric and Oceanic Technology*, submitted, 2005.

Jacobitz, F. C., S. Sarkar, and C. W. van Atta, Direct numerical simulations of the turbulence evolution in a uniformly sheared and stably stratifed flow, *J. Fluid Mech.*, *342*, 231–261, 1997.

Jerlov, N. G., *Optical oceanography*, Elsevier, 1968.

Jin, L. H., R. M. C. So, and T. B. Gatski, Equilibrium states of turbulent homogeneous buoyant flows, *J. Fluid Mech.*, *482*, 207–233, 2003.

Kaltenbach, H.-J., T. Gerz, and U. Schumann, Large-Eddy simulation of homogeneous turbulence and diffusion in stably stratified shear flow, *J. Fluid Mech.*, *280*, 1–40, 1994.

Kantha, L. H., On an improved model for the turbulent pbl, *J. Atmos. Sci.*, *60*(17), 2239–2246, 2003.

Kantha, L. H., and C. A. Clayson, An improved mixed layer model for geophysical applications, *J. Geophys. Res.*, *99*(C12), 25,235–25,266, 1994.

Kato, H., and O. M. Phillips, On the penetration of a turbulent layer into stratified fluid, *J. Fluid Mech.*, *37*(4), 643–655, 1969.

Kondo, J., Air-sea bulk transfer coefficients in diabatic conditions, *Bound. Layer Meteor.*, *9*, 91–112, 1975.

Krause, M., and G. Radach, On the succession of developmental stages of herbivorous zooplankton in the northern north sea during flex '76, *"Meteor"-Forsch.-Ergebnisse, Reihe A*, *22*, 133–149, 1980.

Kühn, W., and G. Radach, A one-dimensional physical-biological model study of the pelagic nitrogen cycling during the spring bloom in the northern North Sea (FLEX'76), *J. Mar. Res.*, *55*, 687–734, 1997.

Large, W. G., J. C. McWilliams, and S. C. Doney, Oceanic vertical mixing: a review and a model with nonlocal boundary layer parameterisation, *Rev. Geophys.*, *32*, 363–403, 1994.

Launder, B. E., G. J. Reece, and W. Rodi, Progress in the development of Reynolds stress turbulent closure, *J. Fluid Mech.*, *68*, 537–566, 1975.

Leonard, B. P., The ULTIMATE conservative difference scheme applied to unsteady one-dimensional advection, *88*, 17–74, 1991.

Liu, W. T., K. B. Katsaros, and J. A. Businger, Bulk parameterization of the air-sea exchange of heat and water vapor including the molecular constraints at the interface, *J. Atmos. Sci.*, *36*, 1722–1735, 1979.

Luyten, P. J., E. Deleersnijder, J. Ozer, and K. G. Ruddik, Presentation of a family of turbulence closure models for stratified shallow water flows and preliminary application to the Rhine outflow region, *Cont. Shelf Res.*, *16*(1), 1996.

Martin, P. J., Simulation of the mixed layer at OWS November and Papa with several models, *J. Geophys. Res.*, *90*(C1), 903–916, 1985.

Mellor, G. L., Retrospect on oceanic boundary layer modeling and second moment closure, in *Parameterization of Small–Scale Processes; Proc. of the Aha Hulikoa Hawaiian Winter Workshop*, edited by P. Mueller and D. Henderson, pp. 251–271, University of Hawaii at Manoa, Honolulu, 1989.

Mellor, G. L., One-dimensional ocean surface layer modeling, a problem and a solution, *J. Phys. Oceanogr.*, *31*(3), 790–809, 2001.

Mellor, G. L., and T. Yamada, A hierarchy of turbulence closure models for planetary boundary layers, *J. Atmos. Sci.*, *31*, 1791–1806, 1974.

Mellor, G. L., and T. Yamada, Development of a tubulence closure model for geophysical fluid problems, *Reviews of Geophysics and Space Physics*, *20*(4), 851–875, 1982.

Mohamed, M. S., and J. C. Larue, The decay power law in grid-generated turbulence, *J. Fluid Mech.*, *219*, 195–214, 1990.

Munk, W. H., and E. R. Anderson, Notes on the theory of the thermocline, *J. Mar. Res.*, *3*, 276–295, 1948.

Neumann, T., W. Fennel, and C. Kremp, Experimental simulations with an ecosystem model of the Baltic Sea: A nutrient load reduction experiment, *Global Biogeochemical Cycles*, *16*, 10.1029/2001GB001,450, 2002.

Pacanowsci, R. C., and S. G. H. Philander, Parameterization of vertical mixing in numerical models of tropical oceans, *J. Phys. Oceanogr.*, *11*, 1443–1451, 1981.

Patankar, S. V., *Numerical Heat Transfer and Fluid Flow*, Taylor & Francis, 1980.

Paulson, C. A., and J. J. Simpson, Irradiance measurements in the upper ocean, *J. Phys. Oceanogr.*, *7*, 952–956, 1977.

Payne, R. E., Albedo of the sea surface, *J. Atmos. Sci.*, *9*, 959–970, 1972.

Pietrzak, J., The use of TVD limiters for forward-in-time upstream-biased advection schemes in ocean modeling, *Monthly Weather Review*, *126*, 812–830, 1998.

Pohlmann, T., Predicting the thermocline in a circulation model of the North Sea – Part I: Model description, calibration and verification, *Cont. Shelf Res.*, *16*, 131–146, 1996.

Pohlmann, T., Estimating the influence of advection during FLEX'76 by means of a three-dimensional shelf sea circulation model, *Dtsch. Hydrogr. Z.*, *49*, 215–226, 1997.

Prandke, H., K. Holtsch, and A. Stips, MITEC technology development: The microstructure/turbulence measuring system mss, *Tech. Rep. EUR 19733 EN*, European Commission, Joint Research Centre, Ispra, Italy, 2000.

Price, J. F., On the scaling of stress driven entrainment experiments, *J. Fluid Mech.*, *90*(3), 509–529, 1979.

Radach, G., J. Trahms, and A. Weber, The chloroophyll development at the central station during FLEX '76 – Two data sets., *CES C.M.*, *C3*, 3–21, 1980.

Radach, G., J. Berg, B. Heinemann, and M. Krause, On the relation of primary production to grazing during the Fladenground Experiment 1976 (FLEX '76), in *Flows of Engergy and Materials in Marine Ecosystems*, edited by M. J. R. Fasham, pp. 597–625, NATO Conf. Ser. IV: Marine Sciences 13, New York, 1984.

Reed, R. K., On estimating insolation over the ocean, *J. Phys. Oceanogr.*, *7*, 482–485, 1977.

Rippeth, T. P., N. Fisher, and J. H. Simpson, The semi-diurnal cycle of turbulent dissipation in the presence of tidal straining, *J. Phys. Oceanogr.*, *31*, 2458–2471, 2001.

Robert, J. L., and Y. Ouellet, A three–dimensional finite element model for the study of steady and non–steady natural flows, in *Three–dimensional models of marine and estuarine dynamics*, edited by J. C. Nihoul and B. M. Jamart, no. 45 in Elsevier Oceanography Series, Elsevier, 1987.

Rodi, W., A new algebraic relation for calculating the Reynolds stresses, *Z. angew. Math. Mech.*, *56*, T 219–T 221, 1976.

Rodi, W., Examples of calculation methods for flow and mixing in stratified fluids, *J. Geophys. Res. (C5)*, *92*, 5305–5328, 1987.

Rohr, J. J., E. C. Itsweire, K. N. Helland, and C. W. van Atta, Growth and decay of turbulence in a stably stratified shear flow, *J. Fluid Mech.*, *195*, 77–111, 1988.

Rosati, A., and K. Miyakoda, A general circulation model for upper ocean simulation, *J. Phys. Oceanogr.*, *18*, 1601–1626, 1988.

Rotta, J., Statistische Theorie nichthomogener Turbulenz. 1. Mitteilung, *Z. Phys.*, *129*, 547–572, 1951.

Samarskij, A. A., *Theorie der Differenzenverfahren*, Akademische Verlagsgesellschaft Geest and Portig, Leipzig, 1984.

Sander, J., Dynamical equations and turbulent closures in geophysics, *Continuum Mech. Thermodyn.*, *10*, 1–28, 1998.

Schumann, U., and T. Gerz, Turbulent mixing in stably stratified shear flows, *J. Appl. Meteorol.*, *34*, 33–48, 1995.

Sharples, J., Time-dependent stratification in regions of large horizontal density gradient, Ph.D. thesis, School of Ocean Sciences, University of Wales, Bangor, 1992.

Shih, L. H., J. R. Koseff, J. H. Ferziger, and C. R. Rehmann, Scaling and parameterization of stratified homogeneous turbulent shear flow, *J. Fluid Mech.*, *412*, 1–20, 2000.

Simpson, J. H., H. Burchard, N. R. Fisher, and T. P. Rippeth, The semi-diurnal cycle of dissipation in a ROFI: model-measurement comparisons, *Cont. Shelf Res.*, *22*, 1615–1628, 2002.

Simpson, J. J., and C. A. Paulson, Mid-ocean observations of atmosphere radiation, *Quart. J. Roy. Meteor. Soc.*, *105*, 487–502, 1999.

Smith, J. D., and S. R. McLean, Spatially averaged flow over a wavy surface, *J. Geophys. Res.*, *82*, 1735–1746, 1977.

So, R. M. C., P. Vimala, L. H. Jin, and C. Y. Zhao, Accounting for buoyancy effects in the explicit algebraic stress model: homogeneous turbulent shear flows, *Theoret. Comput. Fluid Dynamics*, *15*, 283–302, 2002.

So, R. M. C., L. H. Jin, and T. B. Gatski, An explicit algebraic model for turbulent buoyant flows, in *Proceedings of the FEDSM '03: 4th ASME-JSME Joint Fluids Engineering Conference*, Honolulu, Hawaii, USA, 2003.

Soetje, K. C., and K. Huber, A compilation of data on the thermal stratification at the central station in the northern North Sea during FLEX'76, *"Meteor"-Forsch.-Ergebnisse, Reihe A*, *22*, 69–77, 1980.

Speziale, C. G., S. Sarkar, and T. B. Gatski, Modeling the pressure-strain correlation of turbulence: an invariant dynamical systems approach, *J. Fluid Mech.*, *227*, 245–272, 1991.

Stips, A., H. Burchard, K. Bolding, and W. Eifler, Modelling of convective turbulence with a two-equation k-ε turbulence closure scheme, *Ocean Dynamics*, *52*, 153–168, 2002.

Tavoularis, S., and S. Corrsin, Experiments in a nearly homogenous turbulent shear flow with a uniform mean temperature gradient. Part 1, *J. Fluid Mech.*, *104*, 311–348, 1981a.

Tavoularis, S., and S. Corrsin, Experiments in a nearly homogenous turbulent shear flow with a uniform mean temperature gradient. Part 2. The fine structure, *J. Fluid Mech.*, *104*, 349–367, 1981b.

Tavoularis, S., and U. Karnik, Further experiments on the evolution of turbulent stresses and scales in uniformly sheared turbulence, *J. Fluid Mech.*, *204*, 457–478, 1989.

Tennekes, H., The decay of turbulence in plane homogeneous shear flow, in *Lecture Notes on Turbulence*, edited by J. R. Herring and J. C. McWilliams, pp. 32–35, World Scientific, 1989.

Tennekes, H., and J. L. Lumley, *A First Course in Turbulence*, MIT Press, 1972.

Townsend, A. A., *The Structure of Turbulent Shear flow*, Cambridge University Press, 1976.

Umlauf, L., and H. Burchard, A generic length-scale equation for geophysical turbulence models, *J. Mar. Res.*, *61*, 235–265, 2003.

Umlauf, L., and H. Burchard, Second-order turbulence closure models for geophysical boundary layers. a review of recent work, *Cont. Shelf. Res.*, *25*, 795–827, 2005.

Umlauf, L., H. Burchard, and K. Hutter, Extending the k-ω turbulence model towards oceanic applications, *Ocean Modelling*, *5*, 195–218, 2003.

Verduin, J. J., and J. O. Backhaus, Dynamics of plant-flow interactions for the seagrass *amphibolis antarctica*: Field observations and model simulations, *Estuarine, Coastal and Shelf Science*, *50*, 185–204, 2000.

Villarreal, M. R., Parameterisation of turbulence in the ocean and application of a 3D baroclinic model to the Ria de Pontevedra, Ph.D. thesis, Departamento de Fisica da Materia Condensada, Grupo de Fisica Non-Lineal, Universidade de Santiago de Compostela, 2000.

Visser, A. W., Using random walk models to simulate the vertical distribution of particles in a turbulent water column, *158*, 275–281, 1997.

Weigel, P., and E. Hagmeier, Primary production measurements in the Fladen Ground area (North Sea) during the first phase of a spring phytoplankton bloom, *"Meteor"-Forsch.-Ergebnisse, Reihe A*, *22*, 79–86, 1980.

Wilcox, D. C., Reassessment of the scale-determining equation for advanced turbulence models, *AIAA Journal*, *26*(11), 1299–1310, 1988.

Wilcox, D. C., *Turbulence Modeling for CFD*, 2nd ed., DCW Industries, Inc., 1998.

Xing, J., and A. N. Davies, Application of three dimensional turbulence energy models to the determination of tidal mixing and currents in a shallow sea, *Prog. Oceanogr.*, *35*, 153–205, 1995.

Zeierman, S., and M. Wolfshtein, Turbulent time scale for turbulent-flow calculations, *AIAA J.*, *24*(10), 1606–1610, 1986.

Zhao, C. Y., R. M. C. So, and T. B. Gatski, Turbulence modeling effects on the prediction of equilibrium states of buoyant shear flows, *Theoret. Comput. Fluid Dynamics*, *14*, 399–422, 2001.