

Reference Manual

Generated by Doxygen 1.8.11

Contents

1	Modular arbitrary-order ocean-atmosphere model: MAOOAM -- Fortran implementation	1
2	Modular arbitrary-order ocean-atmosphere model: The Tangent Linear and Adjoint model	5
3	Modules Index	7
3.1	Modules List	7
4	Data Type Index	9
4.1	Data Types List	9
5	File Index	11
5.1	File List	11
6	Module Documentation	13
6.1	aotensor_def Module Reference	13
6.1.1	Detailed Description	14
6.1.2	Function/Subroutine Documentation	14
6.1.2.1	a(i)	14
6.1.2.2	add_count(i, j, k, v)	14
6.1.2.3	coeff(i, j, k, v)	14
6.1.2.4	compute_aotensor(func)	15
6.1.2.5	init_aotensor	15
6.1.2.6	kdelta(i, j)	16
6.1.2.7	psi(i)	16
6.1.2.8	t(i)	16
6.1.2.9	theta(i)	16

6.1.3	Variable Documentation	17
6.1.3.1	aotensor	17
6.1.3.2	count_elems	17
6.1.3.3	real_eps	17
6.2	ic_def Module Reference	17
6.2.1	Detailed Description	18
6.2.2	Function/Subroutine Documentation	18
6.2.2.1	load_ic	18
6.2.3	Variable Documentation	19
6.2.3.1	exists	19
6.2.3.2	ic	19
6.3	inprod_analytic Module Reference	20
6.3.1	Detailed Description	21
6.3.2	Function/Subroutine Documentation	21
6.3.2.1	b1(Pi, Pj, Pk)	21
6.3.2.2	b2(Pi, Pj, Pk)	22
6.3.2.3	calculate_a	22
6.3.2.4	calculate_b	22
6.3.2.5	calculate_c_atm	23
6.3.2.6	calculate_c_oc	23
6.3.2.7	calculate_d	24
6.3.2.8	calculate_g	24
6.3.2.9	calculate_k	26
6.3.2.10	calculate_m	26
6.3.2.11	calculate_n	27
6.3.2.12	calculate_o	27
6.3.2.13	calculate_s	28
6.3.2.14	calculate_w	28
6.3.2.15	deallocate_inprod	29
6.3.2.16	delta(r)	30

6.3.2.17	flambda(r)	30
6.3.2.18	init_inprod	30
6.3.2.19	s1(Pj, Pk, Mj, Hk)	31
6.3.2.20	s2(Pj, Pk, Mj, Hk)	32
6.3.2.21	s3(Pj, Pk, Hj, Hk)	32
6.3.2.22	s4(Pj, Pk, Hj, Hk)	32
6.3.3	Variable Documentation	32
6.3.3.1	atmos	32
6.3.3.2	awavenum	32
6.3.3.3	ocean	33
6.3.3.4	owavenum	33
6.4	params Module Reference	33
6.4.1	Detailed Description	35
6.4.2	Function/Subroutine Documentation	36
6.4.2.1	init_nml	36
6.4.2.2	init_params	36
6.4.3	Variable Documentation	37
6.4.3.1	ams	37
6.4.3.2	betp	37
6.4.3.3	ca	37
6.4.3.4	co	38
6.4.3.5	cpa	38
6.4.3.6	cpo	38
6.4.3.7	d	38
6.4.3.8	dp	38
6.4.3.9	dt	39
6.4.3.10	epsa	39
6.4.3.11	f0	39
6.4.3.12	g	39
6.4.3.13	ga	39

6.4.3.14	go	40
6.4.3.15	gp	40
6.4.3.16	h	40
6.4.3.17	k	40
6.4.3.18	kd	40
6.4.3.19	kdp	41
6.4.3.20	kp	41
6.4.3.21	l	41
6.4.3.22	lambda	41
6.4.3.23	lpa	41
6.4.3.24	lpo	42
6.4.3.25	lr	42
6.4.3.26	lsbpa	42
6.4.3.27	lsbpo	42
6.4.3.28	n	42
6.4.3.29	natm	43
6.4.3.30	nbatm	43
6.4.3.31	nboc	43
6.4.3.32	ndim	43
6.4.3.33	noc	43
6.4.3.34	oms	44
6.4.3.35	phi0	44
6.4.3.36	phi0_npi	44
6.4.3.37	pi	44
6.4.3.38	r	44
6.4.3.39	rp	45
6.4.3.40	rr	45
6.4.3.41	rra	45
6.4.3.42	sb	45
6.4.3.43	sbpa	45

6.4.3.44	sbpo	46
6.4.3.45	sc	46
6.4.3.46	scale	46
6.4.3.47	sig0	46
6.4.3.48	t_run	46
6.4.3.49	t_trans	47
6.4.3.50	ta0	47
6.4.3.51	to0	47
6.4.3.52	tw	47
6.4.3.53	writeout	47
6.5	rk2_integrator Module Reference	47
6.5.1	Detailed Description	48
6.5.2	Function/Subroutine Documentation	48
6.5.2.1	init_integrator	48
6.5.2.2	step(y, t, dt, res)	48
6.5.2.3	tendencies(t, y, res)	49
6.5.3	Variable Documentation	49
6.5.3.1	buf_f0	49
6.5.3.2	buf_f1	50
6.5.3.3	buf_y1	50
6.6	rk2_tl_ad_integrator Module Reference	50
6.6.1	Detailed Description	50
6.6.2	Function/Subroutine Documentation	51
6.6.2.1	ad_step(y, ystar, t, dt, res)	51
6.6.2.2	init_tl_ad_integrator	51
6.6.2.3	tl_step(y, ystar, t, dt, res)	51
6.6.3	Variable Documentation	52
6.6.3.1	buf_f0	52
6.6.3.2	buf_f1	52
6.6.3.3	buf_y1	52

6.7	rk4_integrator Module Reference	52
6.7.1	Detailed Description	53
6.7.2	Function/Subroutine Documentation	53
6.7.2.1	init_integrator	53
6.7.2.2	step(y, t, dt, res)	53
6.7.2.3	tendencies(t, y, res)	54
6.7.3	Variable Documentation	54
6.7.3.1	buf_ka	54
6.7.3.2	buf_kb	55
6.7.3.3	buf_y1	55
6.8	rk4_tl_ad_integrator Module Reference	55
6.8.1	Detailed Description	56
6.8.2	Function/Subroutine Documentation	56
6.8.2.1	ad_step(y, ystar, t, dt, res)	56
6.8.2.2	init_tl_ad_integrator	56
6.8.2.3	tl_step(y, ystar, t, dt, res)	57
6.8.3	Variable Documentation	57
6.8.3.1	buf_ka	57
6.8.3.2	buf_kb	57
6.8.3.3	buf_y1	58
6.9	stat Module Reference	58
6.9.1	Detailed Description	58
6.9.2	Function/Subroutine Documentation	59
6.9.2.1	acc(x)	59
6.9.2.2	init_stat	59
6.9.2.3	iter()	59
6.9.2.4	mean()	59
6.9.2.5	reset	60
6.9.2.6	var()	60
6.9.3	Variable Documentation	60

6.9.3.1	i	60
6.9.3.2	m	60
6.9.3.3	mprev	60
6.9.3.4	mtmp	61
6.9.3.5	v	61
6.10	tensor Module Reference	61
6.10.1	Detailed Description	62
6.10.2	Function/Subroutine Documentation	62
6.10.2.1	copy_tensor(src, dst)	62
6.10.2.2	jsparse_mul(coolist_ijk, arr_j, jcoo_ij)	63
6.10.2.3	jsparse_mul_mat(coolist_ijk, arr_j, jcoo_ij)	64
6.10.2.4	mat_to_coo(src, dst)	64
6.10.2.5	simplify(tensor)	65
6.10.2.6	sparse_mul2(coolist_ij, arr_j, res)	66
6.10.2.7	sparse_mul3(coolist_ijk, arr_j, arr_k, res)	66
6.10.3	Variable Documentation	67
6.10.3.1	real_eps	67
6.11	tl_ad_tensor Module Reference	67
6.11.1	Detailed Description	68
6.11.2	Function/Subroutine Documentation	68
6.11.2.1	ad(t, ystar, deltay, buf)	68
6.11.2.2	ad_add_count(i, j, k, v)	69
6.11.2.3	ad_add_count_ref(i, j, k, v)	69
6.11.2.4	ad_coeff(i, j, k, v)	70
6.11.2.5	ad_coeff_ref(i, j, k, v)	70
6.11.2.6	compute_adtensor(func)	71
6.11.2.7	compute_adtensor_ref(func)	71
6.11.2.8	compute_tltensor(func)	71
6.11.2.9	init_adtensor	71
6.11.2.10	init_adtensor_ref	72

6.11.2.11	<code>init_tltensor</code>	72
6.11.2.12	<code>jacobian(ystar)</code>	73
6.11.2.13	<code>jacobian_mat(ystar)</code>	73
6.11.2.14	<code>tl(t, ystar, deltat, buf)</code>	73
6.11.2.15	<code>tl_add_count(i, j, k, v)</code>	74
6.11.2.16	<code>tl_coeff(i, j, k, v)</code>	74
6.11.3	Variable Documentation	75
6.11.3.1	<code>adtensor</code>	75
6.11.3.2	<code>count_elems</code>	75
6.11.3.3	<code>real_eps</code>	75
6.11.3.4	<code>tltensor</code>	75
6.12	util Module Reference	75
6.12.1	Detailed Description	76
6.12.2	Function/Subroutine Documentation	76
6.12.2.1	<code>init_one(A)</code>	76
6.12.2.2	<code>init_random_seed()</code>	76
6.12.2.3	<code>rstr(x, fm)</code>	76
6.12.2.4	<code>str(k)</code>	76
7	Data Type Documentation	77
7.1	<code>inprod_analytic::atm_tensors</code> Type Reference	77
7.1.1	Detailed Description	77
7.1.2	Member Data Documentation	77
7.1.2.1	<code>a</code>	77
7.1.2.2	<code>b</code>	77
7.1.2.3	<code>c</code>	78
7.1.2.4	<code>d</code>	78
7.1.2.5	<code>g</code>	78
7.1.2.6	<code>s</code>	78
7.2	<code>inprod_analytic::atm_wavenum</code> Type Reference	78
7.2.1	Detailed Description	78

7.2.2	Member Data Documentation	78
7.2.2.1	h	78
7.2.2.2	m	79
7.2.2.3	nx	79
7.2.2.4	ny	79
7.2.2.5	p	79
7.2.2.6	typ	79
7.3	tensor::coolist Type Reference	79
7.3.1	Detailed Description	80
7.3.2	Member Data Documentation	80
7.3.2.1	elems	80
7.3.2.2	nelems	80
7.4	tensor::coolist_elem Type Reference	80
7.4.1	Detailed Description	81
7.4.2	Member Data Documentation	81
7.4.2.1	j	81
7.4.2.2	k	81
7.4.2.3	v	81
7.5	inprod_analytic::ocean_tensors Type Reference	81
7.5.1	Detailed Description	82
7.5.2	Member Data Documentation	82
7.5.2.1	c	82
7.5.2.2	k	82
7.5.2.3	m	82
7.5.2.4	n	82
7.5.2.5	o	82
7.5.2.6	w	83
7.6	inprod_analytic::ocean_wavenum Type Reference	83
7.6.1	Detailed Description	83
7.6.2	Member Data Documentation	83
7.6.2.1	h	83
7.6.2.2	nx	83
7.6.2.3	ny	83
7.6.2.4	p	83

8 File Documentation	85
8.1 aotensor_def.f90 File Reference	85
8.2 doc/gen_doc.md File Reference	86
8.3 doc/tl_ad_doc.md File Reference	86
8.4 ic_def.f90 File Reference	86
8.5 inprod_analytic.f90 File Reference	86
8.6 LICENSE.txt File Reference	88
8.6.1 Function Documentation	89
8.6.1.1 files(the""Software"")	89
8.6.1.2 License(MIT) Copyright(c) 2015-2016 Lesley De Cruz and Jonathan Demaeyer Permission is hereby granted	89
8.6.2 Variable Documentation	89
8.6.2.1 charge	89
8.6.2.2 CLAIM	90
8.6.2.3 conditions	90
8.6.2.4 CONTRACT	90
8.6.2.5 copy	90
8.6.2.6 distribute	90
8.6.2.7 FROM	90
8.6.2.8 IMPLIED	90
8.6.2.9 KIND	91
8.6.2.10 LIABILITY	91
8.6.2.11 MERCHANTABILITY	91
8.6.2.12 merge	91
8.6.2.13 modify	91
8.6.2.14 OTHERWISE	91
8.6.2.15 publish	91
8.6.2.16 restriction	92
8.6.2.17 so	92
8.6.2.18 Software	92
8.6.2.19 sublicense	92

8.6.2.20 use	92
8.7 maoam.f90 File Reference	92
8.7.1 Function/Subroutine Documentation	92
8.7.1.1 maoam	92
8.8 params.f90 File Reference	93
8.9 rk2_integrator.f90 File Reference	95
8.10 rk2_tl_ad_integrator.f90 File Reference	96
8.11 rk4_integrator.f90 File Reference	96
8.12 rk4_tl_ad_integrator.f90 File Reference	97
8.13 stat.f90 File Reference	97
8.14 tensor.f90 File Reference	98
8.15 test_aotensor.f90 File Reference	99
8.15.1 Function/Subroutine Documentation	99
8.15.1.1 test_aotensor	99
8.16 test_inprod_analytic.f90 File Reference	99
8.16.1 Function/Subroutine Documentation	100
8.16.1.1 inprod_analytic_test	100
8.17 test_tl_ad.f90 File Reference	100
8.17.1 Function/Subroutine Documentation	100
8.17.1.1 gasdev(idum)	100
8.17.1.2 ran2(idum)	101
8.17.1.3 test_tl_ad	101
8.18 tl_ad_tensor.f90 File Reference	101
8.19 util.f90 File Reference	102
8.19.1 Function/Subroutine Documentation	103
8.19.1.1 lcg(s)	103

Chapter 1

Modular arbitrary-order ocean-atmosphere model: MAOOAM -- Fortran implementation

About

(c) 2013-2016 Lesley De Cruz and Jonathan Demaeyer

See [LICENSE.txt](#) for license information.

This software is provided as supplementary material with:

- De Cruz, L., Demaeyer, J. and Vannitsem, S.: The Modular Arbitrary-Order Ocean-Atmosphere Model: MAOOAM v1.0, Geosci. Model Dev., 9, 2793-2808, [doi:10.5194/gmd-9-2793-2016](https://doi.org/10.5194/gmd-9-2793-2016), 2016.

Please cite this article if you use (a part of) this software for a publication.

The authors would appreciate it if you could also send a reprint of your paper to lesley.deacruz@meteo.be, jonathan.demaeyer@meteo.be and svn@meteo.be.

Consult the MAOOAM [code repository](#) for updates, and [our website](#) for additional resources.

A pdf version of this manual is available [here](#).

Installation

The program can be installed with Makefile. We provide configuration files for two compilers : gfortran and ifort.

By default, gfortran is selected. To select one or the other, simply modify the Makefile accordingly. If gfortran is selected, the code should be compiled with gfortran 4.7+ (allows for allocatable arrays in namelists). If ifort is selected, the code has been tested with the version 14.0.2 and we do not guarantee compatibility with older compiler version.

To install, unpack the archive in a folder, and run: make

Remark: The command "make clean" removes the compiled files.

For Windows users, a minimalistic GNU development environment (including gfortran and make) is available at www.mingw.org.

Description of the files

The model tendencies are represented through a tensor called aotensor which includes all the coefficients. This tensor is computed once at the program initialization.

- [maooam.f90](#) : Main program.
- [aotensor_def.f90](#) : Tensor aotensor computation module.
- [IC_def.f90](#) : A module which loads the user specified initial condition.
- [inprod_analytic.f90](#) : Inner products computation module.
- [rk2_integrator.f90](#) : A module which contains the Heun integrator for the model equations.
- [rk4_integrator.f90](#) : A module which contains the RK4 integrator for the model equations.
- Makefile : The Makefile.
- gfortran.mk : Gfortran compiler options file.
- ifort.mk : Ifort compiler options file.
- [params.f90](#) : The model parameters module.
- [tl_ad_tensor.f90](#) : Tangent Linear (TL) and Adjoint (AD) model tensors definition module
- [rk2_tl_ad_integrator.f90](#) : Heun Tangent Linear (TL) and Adjoint (AD) model integrators module
- [rk4_tl_ad_integrator.f90](#) : RK4 Tangent Linear (TL) and Adjoint (AD) model integrators module
- [test_tl_ad.f90](#) : Tests for the Tangent Linear (TL) and Adjoint (AD) model versions
- README.md : A read me file.
- [LICENSE.txt](#) : The license text of the program.
- [util.f90](#) : A module with various useful functions.
- [tensor.f90](#) : Tensor utility module.
- [stat.f90](#) : A module for statistic accumulation.
- [params.nml](#) : A namelist to specify the model parameters.
- [int_params.nml](#) : A namelist to specify the integration parameters.
- [modeselection.nml](#) : A namelist to specify which spectral decomposition will be used.

Usage

The user first has to fill the [params.nml](#) and [int_params.nml](#) namelist files according to their needs. Indeed, model and integration parameters can be specified respectively in the [params.nml](#) and [int_params.nml](#) namelist files. Some examples related to already published article are available in the [params](#) folder.

The [modeselection.nml](#) namelist can then be filled :

- NBOC and NBATM specify the number of blocks that will be used in respectively the ocean and the atmosphere. Each block corresponds to a given x and y wavenumber.
- The OMS and AMS arrays are integer arrays which specify which wavenumbers of the spectral decomposition will be used in respectively the ocean and the atmosphere. Their shapes are OMS(NBOC,2) and AMS(NBATM,2).

- The first dimension specifies the number attributed by the user to the block and the second dimension specifies the x and the y wavenumbers.
- The VDDG model, described in Vannitsem et al. (2015) is given as an example in the archive.
- Note that the variables of the model are numbered according to the chosen order of the blocks.

Finally, the IC.nml file specifying the initial condition should be defined. To obtain an example of this configuration file corresponding to the model you have previously defined, simply delete the current IC.nml file (if it exists) and run the program :

```
./maoam
```

It will generate a new one and start with the 0 initial condition. If you want another initial condition, stop the program, fill the newly generated file and restart :

```
./maoam
```

It will generate two files :

- evol_field.dat : the recorded time evolution of the variables.
- mean_field.dat : the mean field (the climatology)

The tangent linear and adjoint models of MAOOAM are provided in the [tl_ad_tensor](#), [rk2_tl_ad_integrator](#) and [rk4_tl_ad_integrator](#) modules. It is documented [here](#).

Implementation notes

As the system of differential equations is at most bilinear in y_j ($j = 1..n$), \mathbf{y} being the array of variables, it can be expressed as a tensor contraction :

$$\frac{dy_i}{dt} = \sum_{j,k=0}^{ndim} \mathcal{T}_{i,j,k} y_k y_j$$

with $y_0 = 1$.

The tensor [aotensor_def::aotensor](#) is the tensor \mathcal{T} that encodes the differential equations is composed so that:

- $\mathcal{T}_{i,j,k}$ contains the contribution of dy_i/dt proportional to $y_j y_k$.
- Furthermore, y_0 is always equal to 1, so that $\mathcal{T}_{i,0,0}$ is the constant contribution to dy_i/dt
- $\mathcal{T}_{i,j,0} + \mathcal{T}_{i,0,j}$ is the contribution to dy_i/dt which is linear in y_j .

Ideally, the tensor [aotensor_def::aotensor](#) is composed as an upper triangular matrix (in the last two coordinates).

The tensor for this model is composed in the [aotensor_def](#) module and uses the inner products defined in the [inprod_analytic](#) module.

Final Remarks

The authors would like to thank Kris for help with the lua2fortran project. It has greatly reduced the amount of (error-prone) work.

No animals were harmed during the coding process.

Chapter 2

Modular arbitrary-order ocean-atmosphere model: The Tangent Linear and Adjoint model

Description :

The Tangent Linear and Adjoint model are implemented in the same way as the nonlinear model, with a tensor storing the different terms. The Tangent Linear (TL) tensor $\mathcal{T}_{i,j,k}^{TL}$ is defined as:

$$\mathcal{T}_{i,j,k}^{TL} = \mathcal{T}_{i,k,j} + \mathcal{T}_{i,j,k}$$

while the Adjoint (AD) tensor $\mathcal{T}_{i,j,k}^{AD}$ is defined as:

$$\mathcal{T}_{i,j,k}^{AD} = \mathcal{T}_{j,k,i} + \mathcal{T}_{j,i,k}.$$

where $\mathcal{T}_{i,j,k}$ is the tensor of the nonlinear model.

These two tensors are used to compute the trajectories of the models, with the equations

$$\frac{d\delta y_i}{dt} = \sum_{j=1}^{ndim} \sum_{k=0}^{ndim} \mathcal{T}_{i,j,k}^{TL} y_k^* \delta y_j.$$

$$-\frac{d\delta y_i}{dt} = \sum_{j=1}^{ndim} \sum_{k=0}^{ndim} \mathcal{T}_{i,j,k}^{AD} y_k^* \delta y_j.$$

where y^* is the point where the Tangent model is defined (with $y_0^* = 1$).

Implementation :

The two tensors are implemented in the module [tl_ad_tensor](#) and must be initialized (after calling [params::init_↵](#) [params](#) and [aotensor_def::aotensor](#)) by calling [tl_ad_tensor::init_tlensor\(\)](#) and [tl_ad_tensor::init_adtensor\(\)](#). The tendencies are then given by the routine [tl\(t,ystar,deltay,buf\)](#) and [ad\(t,ystar,deltay,buf\)](#). An integrator with the Heun method is available in the module [rk2_tl_ad_integrator](#) and a fourth-order Runge-Kutta integrator in [rk4_tl_ad_↵](#) [integrator](#). An example on how to use it can be found in the test file [test_tl_ad.f90](#)

Chapter 3

Modules Index

3.1 Modules List

Here is a list of all modules with brief descriptions:

aotensor_def	The equation tensor for the coupled ocean-atmosphere model with temperature which allows for an extensible set of modes in the ocean and in the atmosphere	13
ic_def	Module to load the initial condition	17
inprod_analytic	Inner products between the truncated set of basis functions for the ocean and atmosphere streamfunction fields. These are partly calculated using the analytical expressions from Cehelsky, P., & Tung, K. K. : Theories of multiple equilibria and weather regimes-A critical reexamination. Part II: Baroclinic two-layer models. Journal of the atmospheric sciences, 44(21), 3282-3303, 1987	20
params	The model parameters module	33
rk2_integrator	Module with the integration routines	47
rk2_tl_ad_integrator	Tangent Linear (TL) and Adjoint (AD) model versions of MAOOAM. Integrators module	50
rk4_integrator	Module with the RK4 integration routines	52
rk4_tl_ad_integrator	Tangent Linear (TL) and Adjoint (AD) model versions of MAOOAM. Integrators module	55
stat	Statistics accumulators	58
tensor	Tensor utility module	61
tl_ad_tensor	Tangent Linear (TL) and Adjoint (AD) model versions of MAOOAM. Tensors definition module .	67
util	Utility module	75

Chapter 4

Data Type Index

4.1 Data Types List

Here are the data types with brief descriptions:

inprod_analytic::atm_tensors	
Type holding the atmospheric inner products tensors	77
inprod_analytic::atm_wavenum	
Atmospheric bloc specification type	78
tensor::coolist	
Coordinate list. Type used to represent the sparse tensor	79
tensor::coolist_elem	
Coordinate list element type. Elementary elements of the sparse tensors	80
inprod_analytic::ocean_tensors	
Type holding the oceanic inner products tensors	81
inprod_analytic::ocean_wavenum	
Oceanic bloc specification type	83

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

aotensor_def.f90	85
ic_def.f90	86
inprod_analytic.f90	86
maooam.f90	92
params.f90	93
rk2_integrator.f90	95
rk2_tl_ad_integrator.f90	96
rk4_integrator.f90	96
rk4_tl_ad_integrator.f90	97
stat.f90	97
tensor.f90	98
test_aotensor.f90	99
test_inprod_analytic.f90	99
test_tl_ad.f90	100
tl_ad_tensor.f90	101
util.f90	102

Chapter 6

Module Documentation

6.1 aotensor_def Module Reference

The equation tensor for the coupled ocean-atmosphere model with temperature which allows for an extensible set of modes in the ocean and in the atmosphere.

Functions/Subroutines

- integer function **psi** (i)
Translate the $\psi_{a,i}$ coefficients into effective coordinates.
- integer function **theta** (i)
Translate the $\theta_{a,i}$ coefficients into effective coordinates.
- integer function **a** (i)
Translate the $\psi_{o,i}$ coefficients into effective coordinates.
- integer function **t** (i)
Translate the $\delta T_{o,i}$ coefficients into effective coordinates.
- integer function **kdelta** (i, j)
Kronecker delta function.
- subroutine **coeff** (i, j, k, v)
*Subroutine to add element in the **aotensor** $\mathcal{T}_{i,j,k}$ structure.*
- subroutine **add_count** (i, j, k, v)
*Subroutine to count the elements of the **aotensor** $\mathcal{T}_{i,j,k}$. Add +1 to `count_elems(i)` for each value that is added to the tensor *i*-th component.*
- subroutine **compute_aotensor** (func)
*Subroutine to compute the tensor **aotensor**.*
- subroutine, public **init_aotensor**
*Subroutine to initialise the **aotensor** tensor.*

Variables

- integer, dimension(:), allocatable **count_elems**
Vector used to count the tensor elements.
- real(kind=8), parameter **real_eps** = 2.2204460492503131e-16
Epsilon to test equality with 0.
- type(**coolist**), dimension(:), allocatable, public **aotensor**
 $\mathcal{T}_{i,j,k}$ - Tensor representation of the tendencies.

6.1.1 Detailed Description

The equation tensor for the coupled ocean-atmosphere model with temperature which allows for an extensible set of modes in the ocean and in the atmosphere.

Copyright

2015 Lesley De Cruz & Jonathan Demaeyer. See [LICENSE.txt](#) for license information.

Remarks

Generated Fortran90/95 code from aotensor.lua

6.1.2 Function/Subroutine Documentation

6.1.2.1 integer function aotensor_def::a (integer i) [private]

Translate the $\psi_{o,i}$ coefficients into effective coordinates.

Definition at line 76 of file aotensor_def.f90.

```
76      INTEGER :: i,a
77      a = i + 2 * natm
```

6.1.2.2 subroutine aotensor_def::add_count (integer, intent(in) i , integer, intent(in) j , integer, intent(in) k , real(kind=8), intent(in) v) [private]

Subroutine to count the elements of the [aotensor](#) $\mathcal{T}_{i,j,k}$. Add +1 to count_elems(i) for each value that is added to the tensor i -th component.

Parameters

i	tensor i index
j	tensor j index
k	tensor k index
v	value that will be added

Definition at line 124 of file aotensor_def.f90.

```
124      INTEGER, INTENT(IN) :: i,j,k
125      REAL(KIND=8), INTENT(IN) :: v
126      IF (abs(v) .ge. real_eps) count_elems(i)=count_elems(i)+1
```

6.1.2.3 subroutine aotensor_def::coeff (integer, intent(in) i , integer, intent(in) j , integer, intent(in) k , real(kind=8), intent(in) v) [private]

Subroutine to add element in the [aotensor](#) $\mathcal{T}_{i,j,k}$ structure.

Parameters

i	tensor i index
j	tensor j index
k	tensor k index
v	value to add

Definition at line 99 of file aotensor_def.f90.

```

99      INTEGER, INTENT(IN) :: i,j,k
100      REAL(KIND=8), INTENT(IN) :: v
101      INTEGER :: n
102      IF (.NOT. ALLOCATED(aotensor)) stop "*** coeff routine : tensor not yet allocated ***"
103      IF (.NOT. ALLOCATED(aotensor(i)%elems)) stop "*** coeff routine : tensor not yet allocated ***"
104      IF (abs(v) .ge. real_eps) THEN
105          n=(aotensor(i)%elems)+1
106          IF (j .LE. k) THEN
107              aotensor(i)%elems(n)%j=j
108              aotensor(i)%elems(n)%k=k
109          ELSE
110              aotensor(i)%elems(n)%j=k
111              aotensor(i)%elems(n)%k=j
112          END IF
113          aotensor(i)%elems(n)%v=v
114          aotensor(i)%elems=n
115      END IF

```

6.1.2.4 subroutine aotensor_def::compute_aotensor (external func) [private]

Subroutine to compute the tensor [aotensor](#).

Parameters

<i>func</i>	External function to be used
-------------	------------------------------

Definition at line 132 of file aotensor_def.f90.

6.1.2.5 subroutine, public aotensor_def::init_aotensor ()

Subroutine to initialise the [aotensor](#) tensor.

Remarks

This procedure will also call [params::init_params\(\)](#) and [inprod_analytic::init_inprod\(\)](#) . It will finally call [inprod_analytic::deallocate_inprod\(\)](#) to remove the inner products, which are not needed anymore at this point.

Definition at line 202 of file aotensor_def.f90.

```

202      INTEGER :: i
203      INTEGER :: allocstat
204
205      CALL init_params ! Iniatialise the parameter
206
207      CALL init_inprod ! Initialise the inner product tensors
208
209      ALLOCATE(aotensor(ndim),count_elems(ndim), stat=allocstat)
210      IF (allocstat /= 0) stop "*** Not enough memory ! ***"

```

```

211     count_elems=0
212
213     CALL compute_aotensor(add_count)
214
215     DO i=1,ndim
216         ALLOCATE(aotensor(i)%elems(count_elems(i)), stat=allocstat)
217         IF (allocstat /= 0) stop "*** Not enough memory ! ***"
218     END DO
219
220     DEALLOCATE(count_elems, stat=allocstat)
221     IF (allocstat /= 0) stop "*** Deallocation problem ! ***"
222
223     CALL compute_aotensor(coeff)
224
225     CALL simplify(aotensor)
226
227     CALL deallocate_inprod ! Clean the inner product tensors
228

```

6.1.2.6 integer function aotensor_def::kdelta (integer i, integer j) [private]

Kronecker delta function.

Definition at line 88 of file aotensor_def.f90.

```

88     INTEGER :: i,j,kdelta
89     kdelta=0
90     IF (i == j) kdelta = 1

```

6.1.2.7 integer function aotensor_def::psi (integer i) [private]

Translate the $\psi_{a,i}$ coefficients into effective coordinates.

Definition at line 64 of file aotensor_def.f90.

```

64     INTEGER :: i,psi
65     psi = i

```

6.1.2.8 integer function aotensor_def::t (integer i) [private]

Translate the $\delta T_{o,i}$ coefficients into effective coordinates.

Definition at line 82 of file aotensor_def.f90.

```

82     INTEGER :: i,t
83     t = i + 2 * natm + noc

```

6.1.2.9 integer function aotensor_def::theta (integer i) [private]

Translate the $\theta_{a,i}$ coefficients into effective coordinates.

Definition at line 70 of file aotensor_def.f90.

```

70     INTEGER :: i,theta
71     theta = i + natm

```

6.1.3 Variable Documentation

6.1.3.1 `type(coolist), dimension(:), allocatable, public aotensor_def::aotensor`

$\mathcal{T}_{i,j,k}$ - Tensor representation of the tendencies.

Definition at line 45 of file aotensor_def.f90.

```
45  TYPE(coolist), DIMENSION(:), ALLOCATABLE, PUBLIC :: aotensor
```

6.1.3.2 `integer, dimension(:), allocatable aotensor_def::count_elems [private]`

Vector used to count the tensor elements.

Definition at line 37 of file aotensor_def.f90.

```
37  INTEGER, DIMENSION(:), ALLOCATABLE :: count_elems
```

6.1.3.3 `real(kind=8), parameter aotensor_def::real_eps = 2.2204460492503131e-16 [private]`

Epsilon to test equality with 0.

Definition at line 40 of file aotensor_def.f90.

```
40  REAL(KIND=8), PARAMETER :: real_eps = 2.2204460492503131e-16
```

6.2 ic_def Module Reference

Module to load the initial condition.

Functions/Subroutines

- subroutine, public [load_ic](#)

Subroutine to load the initial condition if IC.nml exists. If it does not, then write IC.nml with 0 as initial condition.

Variables

- logical [exists](#)
Boolean to test for file existence.
- `real(kind=8), dimension(:), allocatable, public ic`
Initial condition vector.

6.2.1 Detailed Description

Module to load the initial condition.

Copyright

2016 Lesley De Cruz, Jonathan Demaeyer & Sebastian Schubert See [LICENSE.txt](#) for license information.

6.2.2 Function/Subroutine Documentation

6.2.2.1 subroutine, public ic_def::load_ic ()

Subroutine to load the initial condition if IC.nml exists. If it does not, then write IC.nml with 0 as initial condition.

Definition at line 32 of file ic_def.f90.

```

32     INTEGER :: i,allocstat,j
33     CHARACTER(len=20) :: fm
34     REAL(KIND=8) :: size_of_random_noise
35     INTEGER, DIMENSION(:), ALLOCATABLE :: seed
36     CHARACTER(LEN=4) :: init_type
37     namelist /iclist/ ic
38     namelist /rand/ init_type,size_of_random_noise,seed
39
40
41     fm(1:6)=' (F3.1)'
42
43     CALL random_seed(size=j)
44
45     IF (ndim == 0) stop "*** Number of dimensions is 0! ***"
46     ALLOCATE(ic(0:ndim),seed(j), stat=allocstat)
47     IF (allocstat /= 0) stop "*** Not enough memory ! ***"
48
49     INQUIRE(file='./IC.nml',exist=exists)
50
51     IF (exists) THEN
52         OPEN(8, file="IC.nml", status='OLD', recl=80, delim='APOSTROPHE')
53         READ(8,nml=iclist)
54         READ(8,nml=rand)
55         CLOSE(8)
56         SELECT CASE (init_type)
57             CASE ('seed')
58                 CALL random_seed(put=seed)
59                 CALL random_number(ic)
60                 ic=2*(ic-0.5)
61                 ic=ic*size_of_random_noise*10.d0
62                 ic(0)=1.0d0
63                 WRITE(6,*) "*** IC.nml namelist written. Starting with 'seeded' random initial condition !***"
64             CASE ('rand')
65                 CALL init_random_seed()
66                 CALL random_seed(get=seed)
67                 CALL random_number(ic)
68                 ic=2*(ic-0.5)
69                 ic=ic*size_of_random_noise*10.d0
70                 ic(0)=1.0d0
71                 WRITE(6,*) "*** IC.nml namelist written. Starting with random initial condition !***"
72             CASE ('zero')
73                 CALL init_random_seed()
74                 CALL random_seed(get=seed)
75                 ic=0
76                 ic(0)=1.0d0
77                 WRITE(6,*) "*** IC.nml namelist written. Starting with initial condition in IC.nml !***"
78             CASE ('read')
79                 CALL init_random_seed()
80                 CALL random_seed(get=seed)
81                 !nothing has to be done IC has already the right values
82                 WRITE(6,*) "*** IC.nml namelist written. Starting with initial condition in IC.nml !***"
83         END SELECT
84     ELSE
85         CALL init_random_seed()
86         CALL random_seed(get=seed)
87         ic=0
88         ic(0)=1.0d0
89         init_type="zero"

```



```

90      size_of_random_noise=0.d0
91      WRITE(6,*) "*** IC.nml namelist written. Starting with 0 as initial condition !***"
92  END IF
93  OPEN(8, file="IC.nml", status='REPLACE')
94  WRITE(8,'(a)') " !-----!"
95  WRITE(8,'(a)') " ! Namelist file : !"
96  WRITE(8,'(a)') " ! Initial condition. !"
97  WRITE(8,'(a)') " !-----!"
98  WRITE(8,*) ""
99  WRITE(8,'(a)') "&ICLIST"
100  WRITE(8,*) " ! psi variables"
101  DO i=1,natm
102      WRITE(8,*) " IC("//trim(str(i))//") = ",ic(i)," ! typ= "&
103      & //awavenum(i)%typ//", Nx= "//trim(rstr(awavenum(i)&
104      & %Nx, fm))//", Ny= "//trim(rstr(awavenum(i)%Ny, fm))
105  END DO
106  WRITE(8,*) " ! theta variables"
107  DO i=1,natm
108      WRITE(8,*) " IC("//trim(str(i+natm))//") = ",ic(i+natm)," ! typ= "&
109      & //awavenum(i)%typ//", Nx= "//trim(rstr(awavenum(i)&
110      & %Nx, fm))//", Ny= "//trim(rstr(awavenum(i)%Ny, fm))
111  END DO
112
113  WRITE(8,*) " ! A variables"
114  DO i=1,noc
115      WRITE(8,*) " IC("//trim(str(i+2*natm))//") = ",ic(i+2*natm)," ! Nx&
116      & = "//trim(rstr(owavenum(i)%Nx, fm))//", Ny= "&
117      & //trim(rstr(owavenum(i)%Ny, fm))
118  END DO
119  WRITE(8,*) " ! T variables"
120  DO i=1,noc
121      WRITE(8,*) " IC("//trim(str(i+noc+2*natm))//") = ",ic(i+2*natm+noc)," &
122      & ! Nx= "//trim(rstr(owavenum(i)%Nx, fm))//", Ny= "&
123      & //trim(rstr(owavenum(i)%Ny, fm))
124  END DO
125
126  WRITE(8,'(a)') "&END"
127  WRITE(8,*) ""
128  WRITE(8,'(a)') " !-----!"
129  WRITE(8,'(a)') " ! Initialisation type. !"
130  WRITE(8,'(a)') " !-----!"
131  WRITE(8,'(a)') " ! type = 'read': use IC above (will generate a new seed);"
132  WRITE(8,'(a)') " ! 'rand': random state (will generate a new seed);"
133  WRITE(8,'(a)') " ! 'zero': zero IC (will generate a new seed);"
134  WRITE(8,'(a)') " ! 'seed': use the seed below (generate the same IC)"
135  WRITE(8,*) ""
136  WRITE(8,'(a)') "&RAND"
137  WRITE(8,'(a)') " init_type= "//init_type//""
138  WRITE(8,'(a,d15.7)') " size_of_random_noise = ",size_of_random_noise
139  DO i=1,j
140      WRITE(8,*) " seed("//trim(str(i))//") = ",seed(i)
141  END DO
142  WRITE(8,'(a)') "&END"
143  WRITE(8,*) ""
144  CLOSE(8)
145

```

6.2.3 Variable Documentation

6.2.3.1 logical ic_def::exists [private]

Boolean to test for file existence.

Definition at line 21 of file ic_def.f90.

```
21  LOGICAL :: exists !< Boolean to test for file existence.
```

6.2.3.2 real(kind=8), dimension(:), allocatable, public ic_def::ic

Initial condition vector.

Definition at line 23 of file ic_def.f90.

```
23  REAL(KIND=8), DIMENSION(:), ALLOCATABLE, PUBLIC :: ic !< Initial condition vector
```

6.3 inprod_analytic Module Reference

Inner products between the truncated set of basis functions for the ocean and atmosphere streamfunction fields. These are partly calculated using the analytical expressions from Cehelsky, P., & Tung, K. K. : Theories of multiple equilibria and weather regimes-A critical reexamination. Part II: Baroclinic two-layer models. Journal of the atmospheric sciences, 44(21), 3282-3303, 1987.

Data Types

- type [atm_tensors](#)
Type holding the atmospheric inner products tensors.
- type [atm_wavenum](#)
Atmospheric bloc specification type.
- type [ocean_tensors](#)
Type holding the oceanic inner products tensors.
- type [ocean_wavenum](#)
Oceanic bloc specification type.

Functions/Subroutines

- real(kind=8) function [b1](#) (Pi, Pj, Pk)
Cehelsky & Tung Helper functions.
- real(kind=8) function [b2](#) (Pi, Pj, Pk)
Cehelsky & Tung Helper functions.
- real(kind=8) function [delta](#) (r)
Integer Dirac delta function.
- real(kind=8) function [flambda](#) (r)
"Odd or even" function
- real(kind=8) function [s1](#) (Pj, Pk, Mj, Hk)
Cehelsky & Tung Helper functions.
- real(kind=8) function [s2](#) (Pj, Pk, Mj, Hk)
Cehelsky & Tung Helper functions.
- real(kind=8) function [s3](#) (Pj, Pk, Hj, Hk)
Cehelsky & Tung Helper functions.
- real(kind=8) function [s4](#) (Pj, Pk, Hj, Hk)
Cehelsky & Tung Helper functions.
- subroutine [calculate_a](#)
Eigenvalues of the Laplacian (atmospheric)
- subroutine [calculate_b](#)
Streamfunction advection terms (atmospheric)
- subroutine [calculate_c_atm](#)
Beta term for the atmosphere.
- subroutine [calculate_d](#)
Forcing of the ocean on the atmosphere.
- subroutine [calculate_g](#)
Temperature advection terms (atmospheric)
- subroutine [calculate_s](#)
Forcing (thermal) of the ocean on the atmosphere.
- subroutine [calculate_k](#)

- *Forcing of the atmosphere on the ocean.*
subroutine [calculate_m](#)
- *Forcing of the ocean fields on the ocean.*
subroutine [calculate_n](#)
- *Beta term for the ocean.*
subroutine [calculate_o](#)
- *Temperature advection term (passive scalar)*
subroutine [calculate_c_oc](#)
- *Streamfunction advection terms (oceanic)*
subroutine [calculate_w](#)
- *Short-wave radiative forcing of the ocean.*
subroutine, public [init_inprod](#)
- *Initialisation of the inner product.*
subroutine, public [deallocate_inprod](#)
- *Deallocation of the inner products.*

Variables

- type([atm_wavenum](#)), dimension(:), allocatable, public [awavenum](#)
Atmospheric blocs specification.
- type([ocean_wavenum](#)), dimension(:), allocatable, public [owavenum](#)
Oceanic blocs specification.
- type([atm_tensors](#)), public [atmos](#)
Atmospheric tensors.
- type([ocean_tensors](#)), public [ocean](#)
Oceanic tensors.

6.3.1 Detailed Description

Inner products between the truncated set of basis functions for the ocean and atmosphere streamfunction fields. These are partly calculated using the analytical expressions from Cehelsky, P., & Tung, K. K. : Theories of multiple equilibria and weather regimes-A critical reexamination. Part II: Baroclinic two-layer models. Journal of the atmospheric sciences, 44(21), 3282-3303, 1987.

Copyright

2015 Lesley De Cruz & Jonathan Demaeyer. See [LICENSE.txt](#) for license information.

Remarks

Generated Fortran90/95 code from inprod_analytic.lua

6.3.2 Function/Subroutine Documentation

6.3.2.1 `real(kind=8) function inprod_analytic::b1 (integer Pi, integer Pj, integer Pk) [private]`

Cehelsky & Tung Helper functions.

Definition at line 91 of file inprod_analytic.f90.

```
91      INTEGER :: pi,pj,pk
92      b1 = (pk + pj) / REAL(pi)
```

6.3.2.2 real(kind=8) function inprod_analytic::b2 (integer P_i , integer P_j , integer P_k) [private]

Cehelsky & Tung Helper functions.

Definition at line 97 of file inprod_analytic.f90.

```

97      INTEGER :: pi,pj,pk
98      b2 = (pk - pj) / REAL(pi)

```

6.3.2.3 subroutine inprod_analytic::calculate_a () [private]

Eigenvalues of the Laplacian (atmospheric)

$$a_{i,j} = (F_i, \nabla^2 F_j) .$$

Definition at line 155 of file inprod_analytic.f90.

```

155      INTEGER :: i
156      TYPE(atm_wavenum) :: ti
157      INTEGER :: allocstat
158      IF (natm == 0 ) THEN
159          stop "*** Problem with calculate_a : natm==0 ! ***"
160      ELSE
161          IF (.NOT. ALLOCATED(atmos%a)) THEN
162              ALLOCATE(atmos%a(natm,natm), stat=allocstat)
163              IF (allocstat /= 0) stop "*** Not enough memory ! ***"
164          END IF
165      END IF
166      atmos%a=0.d0
167
168      DO i=1,natm
169          ti = awavenum(i)
170          atmos%a(i,i) = -(n**2) * ti%Nx**2 - ti%Ny**2
171      ENDDO

```

6.3.2.4 subroutine inprod_analytic::calculate_b () [private]

Streamfunction advection terms (atmospheric)

$$b_{i,j,k} = (F_i, J(F_j, \nabla^2 F_k)) .$$

Remarks

Atmospheric g and a tensors must be computed before calling this routine

Definition at line 182 of file inprod_analytic.f90.

```

182      INTEGER :: i,j,k
183      INTEGER :: allocstat
184
185      IF ((.NOT. ALLOCATED(atmos%a)) .OR. (.NOT. ALLOCATED(atmos%g))) THEN
186          stop "*** atmos%a and atmos%g must be defined before calling calculate_b ! ***"
187      END IF
188
189      IF (natm == 0 ) THEN
190          stop "*** Problem with calculate_b : natm==0 ! ***"
191      ELSE
192          IF (.NOT. ALLOCATED(atmos%b)) THEN
193              ALLOCATE(atmos%b(natm,natm,natm), stat=allocstat)
194              IF (allocstat /= 0) stop "*** Not enough memory ! ***"
195          END IF
196      END IF
197      atmos%b=0.d0
198
199      DO i=1,natm
200          DO j=1,natm
201              DO k=1,natm
202                  atmos%b(i,j,k) = atmos%a(k,k) * atmos%g(i,j,k)
203              END DO
204          END DO
205      END DO

```

6.3.2.5 subroutine inprod_analytic::calculate_c_atm () [private]

Beta term for the atmosphere.

$$c_{i,j} = (F_i, \partial_x F_j) .$$

Remarks

Strict function !! Only accepts KL type. For any other combination, it will not calculate anything

Definition at line 216 of file inprod_analytic.f90.

```

216     INTEGER :: i,j
217     TYPE(atm_wavenum) :: ti, tj
218     REAL(KIND=8) :: val
219     INTEGER :: allocstat
220
221     IF (natm == 0 ) THEN
222         stop "*** Problem with calculate_c_atm : natm==0 ! ***"
223     ELSE
224         IF (.NOT. ALLOCATED(atmos%c)) THEN
225             ALLOCATE(atmos%c(natm,natm), stat=allocstat)
226             IF (allocstat /= 0) stop "*** Not enough memory ! ***"
227         END IF
228     END IF
229     atmos%c=0.d0
230
231     DO i=1,natm
232         DO j=1,natm
233             ti = awavenum(i)
234             tj = awavenum(j)
235             val = 0.d0
236             IF ((ti%typ == "K") .AND. (tj%typ == "L")) THEN
237                 val = n * ti%M * delta(ti%M - tj%H) * delta(ti%P - tj%P)
238             END IF
239             IF (val /= 0.d0) THEN
240                 atmos%c(i,j)=val
241                 atmos%c(j,i) = - val
242             ENDIF
243         END DO
244     END DO

```

6.3.2.6 subroutine inprod_analytic::calculate_c_oc () [private]

Streamfunction advection terms (oceanic)

$$C_{i,j,k} = (\eta_i, J(\eta_j, \nabla^2 \eta_k)) .$$

Remarks

Requires $O_{\{i,j,k\}}$ and $M_{\{i,j\}}$ to be calculated beforehand.

Definition at line 568 of file inprod_analytic.f90.

```

568     INTEGER :: i,j,k
569     REAL(KIND=8) :: val
570     INTEGER :: allocstat
571
572     IF ((.NOT. ALLOCATED(ocean%O)) .OR. (.NOT. ALLOCATED(ocean%M))) THEN
573         stop "*** ocean%O and ocean%M must be defined before calling calculate_C ! ***"
574     END IF
575
576     IF (noc == 0 ) THEN
577         stop "*** Problem with calculate_C : noc==0 ! ***"
578     ELSE
579         IF (.NOT. ALLOCATED(ocean%C)) THEN

```

```

580         ALLOCATE(ocean%C(noc,noc,noc), stat=allocstat)
581         IF (allocstat /= 0) stop "*** Not enough memory ! ***"
582     END IF
583 END IF
584 ocean%C=0.d0
585 val=0.d0
586
587 DO i=1,noc
588     DO j=1,noc
589         DO k=1,noc
590             val = ocean%M(k,k) * ocean%O(i,j,k)
591             IF (val /= 0.d0) ocean%C(i,j,k) = val
592         END DO
593     END DO
594 END DO

```

6.3.2.7 subroutine inprod_analytic::calculate_d () [private]

Forcing of the ocean on the atmosphere.

$$d_{i,j} = (F_i, \nabla^2 \eta_j) .$$

Remarks

Atmospheric s tensor and oceanic M tensor must be computed before calling this routine !

Definition at line 255 of file inprod_analytic.f90.

```

255     INTEGER :: i,j
256     INTEGER :: allocstat
257
258     IF ((.NOT. ALLOCATED(atmos%s)) .OR. (.NOT. ALLOCATED(ocean%M))) THEN
259         stop "*** atmos%s and ocean%M must be defined before calling calculate_d ! ***"
260     END IF
261
262
263     IF (natm == 0 ) THEN
264         stop "*** Problem with calculate_d : natm==0 ! ***"
265     ELSE
266         IF (.NOT. ALLOCATED(atmos%d)) THEN
267             ALLOCATE(atmos%d(natm,noc), stat=allocstat)
268             IF (allocstat /= 0) stop "*** Not enough memory ! ***"
269         END IF
270     END IF
271     atmos%d=0.d0
272
273     DO i=1,natm
274         DO j=1,noc
275             atmos%d(i,j)=atmos%s(i,j) * ocean%M(j,j)
276         END DO
277     END DO

```

6.3.2.8 subroutine inprod_analytic::calculate_g () [private]

Temperature advection terms (atmospheric)

$$g_{i,j,k} = (F_i, J(F_j, F_k)) .$$

Definition at line 288 of file inprod_analytic.f90.

```

288     INTEGER :: i,j,k
289     TYPE(atm_wavenum) :: ti, tj, tk
290     REAL(KIND=8) :: val,vb1, vb2, vs1, vs2, vs3, vs4
291     INTEGER :: allocstat
292
293     IF (natm == 0 ) THEN
294         stop "*** Problem with calculate_g : natm==0 ! ***"
295     ELSE
296         IF (.NOT. ALLOCATED(atmos%g)) THEN
297             ALLOCATE(atmos%g(natm,natm,natm), stat=allocstat)
298             IF (allocstat /= 0) stop "*** Not enough memory ! ***"
299         END IF
300     END IF
301     atmos%g=0.d0
302
303     DO i=1,natm
304         DO j=1,natm
305             DO k=1,natm
306                 ti = awavenum(i)
307                 tj = awavenum(j)
308                 tk = awavenum(k)
309                 val=0.d0
310                 IF ((ti%typ == "A").AND. (tj%typ == "K").AND. (tk%typ == "L")) THEN
311                     vb1 = b1(ti%P,tj%P,tk%P)
312                     vb2 = b2(ti%P,tj%P,tk%P)
313                     val = -2 * sqrt(2.) / pi * tj%M * delta(tj%M - tk%H) * flambda(ti%P + tj%P + tk%P)
314                     IF (val /= 0.d0) val = val * (vb1**2 / (vb1**2 - 1) - vb2**2 / (vb2**2 - 1))
315                 ELSEIF ((ti%typ == "K").AND. (tj%typ == "K").AND. (tk%typ == "L")) THEN
316                     vs1 = s1(tj%P,tk%P,tj%M,tk%H)
317                     vs2 = s2(tj%P,tk%P,tj%M,tk%H)
318                     val = vs1 * (delta(ti%M - tk%H - tj%M) * delta(ti%P -&
319                         & tk%P + tj%P) - delta(ti%M - tk%H - tj%M) *&
320                         & delta(ti%P + tk%P - tj%P) + (delta(tk%H - tj%M&
321                         & + ti%M) + delta(tk%H - tj%M - ti%M)) *&
322                         & delta(tk%P + tj%P - ti%P)) + vs2 * (delta(ti%M&
323                         & - tk%H - tj%M) * delta(ti%P - tk%P - tj%P) +&
324                         & (delta(tk%H - tj%M - ti%M) + delta(ti%M + tk%H&
325                         & - tj%M)) * (delta(ti%P - tk%P + tj%P) -&
326                         & delta(tk%P - tj%P + ti%P)))
327                 END IF
328                 val=val*n
329                 IF (val /= 0.d0) THEN
330                     atmos%g(i,j,k) = val
331                     atmos%g(j,k,i) = val
332                     atmos%g(k,i,j) = val
333                     atmos%g(i,k,j) = -val
334                     atmos%g(j,i,k) = -val
335                     atmos%g(k,j,i) = -val
336                 ENDIF
337             END DO
338         END DO
339     END DO
340
341     DO i=1,natm
342         DO j=i,natm
343             DO k=j,natm
344                 ti = awavenum(i)
345                 tj = awavenum(j)
346                 tk = awavenum(k)
347                 val=0.d0
348
349                 IF ((ti%typ == "L").AND. (tj%typ == "L").AND. (tk%typ == "L")) THEN
350                     vs3 = s3(tj%P,tk%P,tj%H,tk%H)
351                     vs4 = s4(tj%P,tk%P,tj%H,tk%H)
352                     val = vs3 * ((delta(tk%H - tj%H - ti%H) - delta(tk%H &
353                         & - tj%H + ti%H)) * delta(tk%P + tj%P - ti%P) +&
354                         & delta(tk%H + tj%H - ti%H) * (delta(tk%P - tj%P&
355                         & + ti%P) - delta(tk%P - tj%P - ti%P))) + vs4 *&
356                         & ((delta(tk%H + tj%H - ti%H) * delta(tk%P - tj&
357                         & %P - ti%P)) + (delta(tk%H - tj%H + ti%H) -&
358                         & delta(tk%H - tj%H - ti%H)) * (delta(tk%P - tj&
359                         & %P - ti%P) - delta(tk%P - tj%P + ti%P)))
360                 ENDIF
361                 val=val*n
362                 IF (val /= 0.d0) THEN
363                     atmos%g(i,j,k) = val
364                     atmos%g(j,k,i) = val
365                     atmos%g(k,i,j) = val
366                     atmos%g(i,k,j) = -val
367                     atmos%g(j,i,k) = -val
368                     atmos%g(k,j,i) = -val
369                 ENDIF
370             ENDDO
371         ENDDO
372     ENDDO
373

```

6.3.2.9 subroutine inprod_analytic::calculate_k () [private]

Forcing of the atmosphere on the ocean.

$$K_{i,j} = (\eta_i, \nabla^2 F_j) .$$

Remarks

atmospheric a and s tensors must be computed before calling this function !

Definition at line 434 of file inprod_analytic.f90.

```

434     INTEGER :: i,j
435     INTEGER :: allocstat
436
437     IF ((.NOT. ALLOCATED(atmos%a)) .OR. (.NOT. ALLOCATED(atmos%s))) THEN
438         stop "*** atmos%a and atmos%s must be defined before calling calculate_K ! ***"
439     END IF
440
441     IF (noc == 0 ) THEN
442         stop "*** Problem with calculate_K : noc==0 ! ***"
443     ELSEIF (natm == 0 ) THEN
444         stop "*** Problem with calculate_K : natm==0 ! ***"
445     ELSE
446         IF (.NOT. ALLOCATED(ocean%K)) THEN
447             ALLOCATE(ocean%K(noc,natm), stat=allocstat)
448             IF (allocstat /= 0) stop "*** Not enough memory ! ***"
449         END IF
450     END IF
451     ocean%K=0.d0
452
453     DO i=1,noc
454         DO j=1,natm
455             ocean%K(i,j) = atmos%s(j,i) * atmos%a(j,j)
456         END DO
457     END DO

```

6.3.2.10 subroutine inprod_analytic::calculate_m () [private]

Forcing of the ocean fields on the ocean.

$$M_{i,j} = (eta_i, \nabla^2 \eta_j) .$$

Definition at line 464 of file inprod_analytic.f90.

```

464     INTEGER :: i
465     TYPE(ocean_wavenum) :: di
466     INTEGER :: allocstat
467     IF (noc == 0 ) THEN
468         stop "*** Problem with calculate_M : noc==0 ! ***"
469     ELSE
470         IF (.NOT. ALLOCATED(ocean%M)) THEN
471             ALLOCATE(ocean%M(noc,noc), stat=allocstat)
472             IF (allocstat /= 0) stop "*** Not enough memory ! ***"
473         END IF
474     END IF
475     ocean%M=0.d0
476
477     DO i=1,noc
478         di = owavenum(i)
479         ocean%M(i,i) = -(n**2) * di%Nx**2 - di%Ny**2
480     END DO

```


6.3.2.11 subroutine inprod_analytic::calculate_n() [private]

Beta term for the ocean.

$$N_{i,j} = (\eta_i, \partial_x \eta_j).$$

Definition at line 487 of file inprod_analytic.f90.

```

487     INTEGER :: i,j
488     TYPE(ocean_wavenum) :: di,dj
489     REAL(KIND=8) :: val
490     INTEGER :: allocstat
491     IF (noc == 0 ) THEN
492         stop "*** Problem with calculate_N : noc==0 ! ***"
493     ELSE
494         IF (.NOT. ALLOCATED(ocean%N)) THEN
495             ALLOCATE(ocean%N(noc,noc), stat=allocstat)
496             IF (allocstat /= 0) stop "*** Not enough memory ! ***"
497         END IF
498     END IF
499     ocean%N=0.d0
500     val=0.d0
501
502     DO i=1,noc
503         DO j=1,noc
504             di = owavenum(i)
505             dj = owavenum(j)
506             val = delta(di%P - dj%P) * flambda(di%H + dj%H)
507             IF (val /= 0.d0) ocean%N(i,j) = val * (-2) * dj%H * di%H * n / ((dj%H**2 - di%H**2) * pi)
508         END DO
509     END DO

```

6.3.2.12 subroutine inprod_analytic::calculate_o() [private]

Temperature advection term (passive scalar)

$$O_{i,j,k} = (\eta_i, J(\eta_j, \eta_k)) .$$

Definition at line 516 of file inprod_analytic.f90.

```

516     INTEGER :: i,j,k
517     REAL(KIND=8) :: vs3,vs4,val
518     TYPE(ocean_wavenum) :: di,dj,dk
519     INTEGER :: allocstat
520     IF (noc == 0 ) THEN
521         stop "*** Problem with calculate_O : noc==0 ! ***"
522     ELSE
523         IF (.NOT. ALLOCATED(ocean%O)) THEN
524             ALLOCATE(ocean%O(noc,noc,noc), stat=allocstat)
525             IF (allocstat /= 0) stop "*** Not enough memory ! ***"
526         END IF
527     END IF
528     ocean%O=0.d0
529     val=0.d0
530
531     DO i=1,noc
532         DO j=i,noc
533             DO k=j,noc
534                 di = owavenum(i)
535                 dj = owavenum(j)
536                 dk = owavenum(k)
537                 vs3 = s3(dj%P,dk%P,dj%H,dk%H)
538                 vs4 = s4(dj%P,dk%P,dj%H,dk%H)
539                 val = vs3*((delta(dk%H - dj%H - di%H) - delta(dk%H - dj%
540                     &%H + di%H)) * delta(dk%P + dj%P - di%P) + delta(dk%
541                     &%H + dj%H - di%H) * (delta(dk%P - dj%P + di%P) -&
542                     & delta(dk%P - dj%P - di%P))) + vs4 * ((delta(dk%H &
543                     &+ dj%H - di%H) * delta(dk%P - dj%P - di%P)) +&
544                     & (delta(dk%H - dj%H + di%H) - delta(dk%H - dj%H -&
545                     & di%H)) * (delta(dk%P - dj%P - di%P) - delta(dk%P &
546                     &- dj%P + di%P)))
547                 val = val * n / 2
548             IF (val /= 0.d0) THEN

```

```

549             ocean%O(i,j,k) = val
550             ocean%O(j,k,i) = val
551             ocean%O(k,i,j) = val
552             ocean%O(i,k,j) = -val
553             ocean%O(j,i,k) = -val
554             ocean%O(k,j,i) = -val
555         END IF
556     END DO
557 END DO
558 END DO

```

6.3.2.13 subroutine inprod_analytic::calculate_s () [private]

Forcing (thermal) of the ocean on the atmosphere.

$$s_{i,j} = (F_i, \eta_j).$$

Definition at line 380 of file inprod_analytic.f90.

```

380     INTEGER :: i,j
381     TYPE(atm_wavenum) :: ti
382     TYPE(ocean_wavenum) :: dj
383     REAL(KIND=8) :: val
384     INTEGER :: allocstat
385     IF (natm == 0 ) THEN
386         stop "*** Problem with calculate_s : natm==0 ! ***"
387     ELSEIF (noc == 0) then
388         stop "*** Problem with calculate_s : noc==0 ! ***"
389     ELSE
390         IF (.NOT. ALLOCATED(atmos%s)) THEN
391             ALLOCATE(atmos%s(natm,noc), stat=allocstat)
392             IF (allocstat /= 0) stop "*** Not enough memory ! ***"
393         END IF
394     END IF
395     atmos%s=0.d0
396
397     DO i=1,natm
398         DO j=1,noc
399             ti = awavenum(i)
400             dj = owavenum(j)
401             val=0.d0
402             IF (ti%typ == "A") THEN
403                 val = flambda(dj%H) * flambda(dj%P + ti%P)
404                 IF (val /= 0.d0) THEN
405                     val = val*8*sqrt(2.)*dj%P/(pi**2 * (dj%P**2 - ti%P**2) * dj%H)
406                 END IF
407             ELSEIF (ti%typ == "K") THEN
408                 val = flambda(2 * ti%M + dj%H) * delta(dj%P - ti%P)
409                 IF (val /= 0.d0) THEN
410                     val = val*4*dj%H/(pi * (-4 * ti%M**2 + dj%H**2))
411                 END IF
412             ELSEIF (ti%typ == "L") THEN
413                 val = delta(dj%P - ti%P) * delta(2 * ti%H - dj%H)
414             END IF
415             IF (val /= 0.d0) THEN
416                 atmos%s(i,j)=val
417             ENDIF
418         END DO
419     END DO

```

6.3.2.14 subroutine inprod_analytic::calculate_w () [private]

Short-wave radiative forcing of the ocean.

$$W_{i,j} = (\eta_i, F_j).$$

Remarks

atmospheric s tensor must be computed before calling this function !

Definition at line 605 of file inprod_analytic.f90.

```

605     INTEGER :: i,j
606     INTEGER :: allocstat
607
608     IF (.NOT. ALLOCATED(atmos%s)) THEN
609         stop "*** atmos%s must be defined before calling calculate_W ! ***"
610     END IF
611
612     IF (noc == 0 ) THEN
613         stop "*** Problem with calculate_W : noc==0 ! ***"
614     ELSEIF (natm == 0 ) THEN
615         stop "*** Problem with calculate_W : natm==0 ! ***"
616     ELSE
617         IF (.NOT. ALLOCATED(ocean%W)) THEN
618             ALLOCATE(ocean%W(noc,natm), stat=allocstat)
619             IF (allocstat /= 0) stop "*** Not enough memory ! ***"
620         END IF
621     END IF
622     ocean%W=0.d0
623
624     DO i=1,noc
625         DO j=1,natm
626             ocean%W(i,j) = atmos%s(j,i)
627         END DO
628     END DO

```

6.3.2.15 subroutine, public inprod_analytic::deallocate_inprod ()

Deallocation of the inner products.

Definition at line 722 of file inprod_analytic.f90.

```

722     INTEGER :: allocstat
723
724     ! Deallocation of atmospheric inprod
725     allocstat=0
726     IF (ALLOCATED(atmos%a)) DEALLOCATE(atmos%a, stat=allocstat)
727     IF (allocstat /= 0) stop "*** Problem to deallocate ! ***"
728
729     allocstat=0
730     IF (ALLOCATED(atmos%c)) DEALLOCATE(atmos%c, stat=allocstat)
731     IF (allocstat /= 0) stop "*** Problem to deallocate ! ***"
732
733     allocstat=0
734     IF (ALLOCATED(atmos%d)) DEALLOCATE(atmos%d, stat=allocstat)
735     IF (allocstat /= 0) stop "*** Problem to deallocate ! ***"
736
737     allocstat=0
738     IF (ALLOCATED(atmos%s)) DEALLOCATE(atmos%s, stat=allocstat)
739     IF (allocstat /= 0) stop "*** Problem to deallocate ! ***"
740
741     allocstat=0
742     IF (ALLOCATED(atmos%g)) DEALLOCATE(atmos%g, stat=allocstat)
743     IF (allocstat /= 0) stop "*** Problem to deallocate ! ***"
744
745     allocstat=0
746     IF (ALLOCATED(atmos%b)) DEALLOCATE(atmos%b, stat=allocstat)
747     IF (allocstat /= 0) stop "*** Problem to deallocate ! ***"
748
749     ! Deallocation of oceanic inprod
750     allocstat=0
751     IF (ALLOCATED(ocean%K)) DEALLOCATE(ocean%K, stat=allocstat)
752     IF (allocstat /= 0) stop "*** Problem to deallocate ! ***"
753
754     allocstat=0
755     IF (ALLOCATED(ocean%M)) DEALLOCATE(ocean%M, stat=allocstat)
756     IF (allocstat /= 0) stop "*** Problem to deallocate ! ***"
757
758     allocstat=0
759     IF (ALLOCATED(ocean%N)) DEALLOCATE(ocean%N, stat=allocstat)

```

```

760   IF (allocstat /= 0) stop "*** Problem to deallocate ! ***"
761
762   allocstat=0
763   IF (ALLOCATED(ocean%W)) DEALLOCATE(ocean%W, stat=allocstat)
764   IF (allocstat /= 0) stop "*** Problem to deallocate ! ***"
765
766   allocstat=0
767   IF (ALLOCATED(ocean%O)) DEALLOCATE(ocean%O, stat=allocstat)
768   IF (allocstat /= 0) stop "*** Problem to deallocate ! ***"
769
770   allocstat=0
771   IF (ALLOCATED(ocean%C)) DEALLOCATE(ocean%C, stat=allocstat)
772   IF (allocstat /= 0) stop "*** Problem to deallocate ! ***"

```

6.3.2.16 real(kind=8) function inprod_analytic::delta (integer r) [private]

Integer Dirac delta function.

Definition at line 103 of file inprod_analytic.f90.

```

103   INTEGER :: r
104   IF (r==0) THEN
105     delta = 1.d0
106   ELSE
107     delta = 0.d0
108   ENDIF

```

6.3.2.17 real(kind=8) function inprod_analytic::flambda (integer r) [private]

"Odd or even" function

Definition at line 113 of file inprod_analytic.f90.

```

113   INTEGER :: r
114   IF (mod(r,2)==0) THEN
115     flambda = 0.d0
116   ELSE
117     flambda = 1.d0
118   ENDIF

```

6.3.2.18 subroutine, public inprod_analytic::init_inprod ()

Initialisation of the inner product.

Definition at line 639 of file inprod_analytic.f90.

```

639   INTEGER :: i,j
640   INTEGER :: allocstat
641
642   ! Definition of the types and wave numbers tables
643
644   ALLOCATE(owavenum(noc),awavenum(natm), stat=allocstat)
645   IF (allocstat /= 0) stop "*** Not enough memory ! ***"
646
647   j=0
648   DO i=1,nbatm
649     IF (ams(i,1)==1) THEN
650       awavenum(j+1)%typ='A'
651       awavenum(j+2)%typ='K'
652       awavenum(j+3)%typ='L'
653
654       awavenum(j+1)%P=ams(i,2)

```

```

655         awavenum(j+2)%M=ams(i,1)
656         awavenum(j+2)%P=ams(i,2)
657         awavenum(j+3)%H=ams(i,1)
658         awavenum(j+3)%P=ams(i,2)
659
660         awavenum(j+1)%Ny=REAL(ams(i,2))
661         awavenum(j+2)%Nx=REAL(ams(i,1))
662         awavenum(j+2)%Ny=REAL(ams(i,2))
663         awavenum(j+3)%Nx=REAL(ams(i,1))
664         awavenum(j+3)%Ny=REAL(ams(i,2))
665
666         j=j+3
667     ELSE
668         awavenum(j+1)%typ='K'
669         awavenum(j+2)%typ='L'
670
671         awavenum(j+1)%M=ams(i,1)
672         awavenum(j+1)%P=ams(i,2)
673         awavenum(j+2)%H=ams(i,1)
674         awavenum(j+2)%P=ams(i,2)
675
676         awavenum(j+1)%Nx=REAL(ams(i,1))
677         awavenum(j+1)%Ny=REAL(ams(i,2))
678         awavenum(j+2)%Nx=REAL(ams(i,1))
679         awavenum(j+2)%Ny=REAL(ams(i,2))
680
681         j=j+2
682
683     ENDIF
684 ENDDO
685
686 DO i=1,noc
687     owavenum(i)%H=oms(i,1)
688     owavenum(i)%P=oms(i,2)
689
690     owavenum(i)%Nx=oms(i,1)/2.d0
691     owavenum(i)%Ny=oms(i,2)
692
693 ENDDO
694
695 ! Computation of the atmospheric inner products tensors
696
697 CALL calculate_a
698 CALL calculate_g
699 CALL calculate_s
700 CALL calculate_b
701 CALL calculate_c_atm
702
703 ! Computation of the oceanic inner products tensors
704
705 CALL calculate_m
706 CALL calculate_n
707 CALL calculate_o
708 CALL calculate_c_oc
709 CALL calculate_w
710 CALL calculate_k
711
712 ! A last atmospheric one that needs ocean%M
713
714 CALL calculate_d
715
716
717

```

6.3.2.19 real(kind=8) function inprod_analytic::s1 (integer *Pj*, integer *Pk*, integer *Mj*, integer *Hk*) [private]

Cehelsky & Tung Helper functions.

Definition at line 123 of file inprod_analytic.f90.

```

123     INTEGER :: pk,pj,mj,hk
124     s1 = -((pk * mj + pj * hk)) / 2.d0

```

6.3.2.20 `real(kind=8) function inprod_analytic::s2 (integer Pj, integer Pk, integer Mj, integer Hk) [private]`

Cehelsky & Tung Helper functions.

Definition at line 129 of file `inprod_analytic.f90`.

```
129    INTEGER :: pk,pj,mj,hk
130    s2 = (pk * mj - pj * hk) / 2.d0
```

6.3.2.21 `real(kind=8) function inprod_analytic::s3 (integer Pj, integer Pk, integer Hj, integer Hk) [private]`

Cehelsky & Tung Helper functions.

Definition at line 135 of file `inprod_analytic.f90`.

```
135    INTEGER :: pj,pk,hj,hk
136    s3 = (pk * hj + pj * hk) / 2.d0
```

6.3.2.22 `real(kind=8) function inprod_analytic::s4 (integer Pj, integer Pk, integer Hj, integer Hk) [private]`

Cehelsky & Tung Helper functions.

Definition at line 141 of file `inprod_analytic.f90`.

```
141    INTEGER :: pj,pk,hj,hk
142    s4 = (pk * hj - pj * hk) / 2.d0
```

6.3.3 Variable Documentation

6.3.3.1 `type(atm_tensors), public inprod_analytic::atmos`

Atmospheric tensors.

Definition at line 69 of file `inprod_analytic.f90`.

```
69    TYPE(atm_tensors), PUBLIC :: atmos
```

6.3.3.2 `type(atm_wavenum), dimension(:), allocatable, public inprod_analytic::awavenum`

Atmospheric blocs specification.

Definition at line 64 of file `inprod_analytic.f90`.

```
64    TYPE(atm_wavenum), DIMENSION(:), ALLOCATABLE, PUBLIC :: awavenum
```

6.3.3.3 `type(ocean_tensors), public inprod_analytic::ocean`

Oceanic tensors.

Definition at line 71 of file `inprod_analytic.f90`.

```
71  TYPE(ocean_tensors), PUBLIC :: ocean
```

6.3.3.4 `type(ocean_wavenum), dimension(:), allocatable, public inprod_analytic::owavenum`

Oceanic blocs specification.

Definition at line 66 of file `inprod_analytic.f90`.

```
66  TYPE(ocean_wavenum), DIMENSION(:), ALLOCATABLE, PUBLIC :: owavenum
```

6.4 params Module Reference

The model parameters module.

Functions/Subroutines

- subroutine, private `init_nml`
Read the basic parameters and mode selection from the namelist.
- subroutine `init_params`
Parameters initialisation routine.

Variables

- real(kind=8) `n`
 $n = 2L_y/L_x$ - Aspect ratio
- real(kind=8) `phi0`
Latitude in radian.
- real(kind=8) `rra`
Earth radius.
- real(kind=8) `sig0`
 σ_0 - Non-dimensional static stability of the atmosphere.
- real(kind=8) `k`
Bottom atmospheric friction coefficient.
- real(kind=8) `kp`
 k' - Internal atmospheric friction coefficient.
- real(kind=8) `r`
Frictional coefficient at the bottom of the ocean.
- real(kind=8) `d`
Mechanical coupling parameter between the ocean and the atmosphere.
- real(kind=8) `f0`

- f_0 - Coriolis parameter
 - real(kind=8) [gp](#)
- g' Reduced gravity
 - real(kind=8) [h](#)
- h - Depth of the active water layer of the ocean.
 - real(kind=8) [phi0_npi](#)
- ϕ_0 Latitude exprimed in fraction of pi.
 - real(kind=8) [lambda](#)
- λ - Sensible + turbulent heat exchange between the ocean and the atmosphere.
 - real(kind=8) [co](#)
- C_a - Constant short-wave radiation of the ocean.
 - real(kind=8) [go](#)
- γ_o - Specific heat capacity of the ocean.
 - real(kind=8) [ca](#)
- C_a - Constant short-wave radiation of the atmosphere.
 - real(kind=8) [to0](#)
- T_o^0 - Stationary solution for the 0-th order ocean temperature.
 - real(kind=8) [ta0](#)
- T_a^0 - Stationary solution for the 0-th order atmospheric temperature.
 - real(kind=8) [epsa](#)
- ϵ_a - Emissivity coefficient for the grey-body atmosphere.
 - real(kind=8) [ga](#)
- γ_a - Specific heat capacity of the atmosphere.
 - real(kind=8) [rr](#)
- R - Gas constant of dry air
 - real(kind=8) [scale](#)
- $L_y = L \pi$ - The characteristic space scale.
 - real(kind=8) [pi](#)
- π
 - real(kind=8) [lr](#)
- L_R - Rossby deformation radius
 - real(kind=8) [g](#)
- γ
 - real(kind=8) [rp](#)
- r' - Frictional coefficient at the bottom of the ocean.
 - real(kind=8) [dp](#)
- d' - Non-dimensional mechanical coupling parameter between the ocean and the atmosphere.
 - real(kind=8) [kd](#)
- k_d - Non-dimensional bottom atmospheric friction coefficient.
 - real(kind=8) [kdp](#)
- k'_d - Non-dimensional internal atmospheric friction coefficient.
 - real(kind=8) [cpo](#)
- C'_a - Non-dimensional constant short-wave radiation of the ocean.
 - real(kind=8) [lpo](#)
- λ'_o - Non-dimensional sensible + turbulent heat exchange from ocean to atmosphere.
 - real(kind=8) [cpa](#)
- C'_a - Non-dimensional constant short-wave radiation of the atmosphere.
 - real(kind=8) [lpa](#)
- λ'_a - Non-dimensional sensible + turbulent heat exchange from atmosphere to ocean.
 - real(kind=8) [sbpo](#)
- $\sigma'_{B,o}$ - Long wave radiation lost by ocean to atmosphere & space.
 -

- real(kind=8) [sbpa](#)
 $\sigma'_{B,a}$ - Long wave radiation from atmosphere absorbed by ocean.
- real(kind=8) [lsbpo](#)
 $S'_{B,o}$ - Long wave radiation from ocean absorbed by atmosphere.
- real(kind=8) [lsbpa](#)
 $S'_{B,a}$ - Long wave radiation lost by atmosphere to space & ocean.
- real(kind=8) [l](#)
 L - Domain length scale
- real(kind=8) [sc](#)
Ratio of surface to atmosphere temperature.
- real(kind=8) [sb](#)
Stefan–Boltzmann constant.
- real(kind=8) [betp](#)
- real(kind=8) [t_trans](#)
Transient time period.
- real(kind=8) [t_run](#)
Effective intergration time (length of the generated trajectory)
- real(kind=8) [dt](#)
Integration time step.
- real(kind=8) [tw](#)
Write all variables every tw time units.
- logical [writeout](#)
Write to file boolean.
- integer [nboc](#)
Number of atmospheric blocks.
- integer [nbatm](#)
Number of oceanic blocks.
- integer [natm](#) =0
Number of atmospheric basis functions.
- integer [noc](#) =0
Number of oceanic basis functions.
- integer [ndim](#)
Number of variables (dimension of the model)
- integer, dimension(:,:), allocatable [oms](#)
Ocean mode selection array.
- integer, dimension(:,:), allocatable [ams](#)
Atmospheric mode selection array.

6.4.1 Detailed Description

The model parameters module.

Copyright

2015 Lesley De Cruz & Jonathan Demaeyer. See [LICENSE.txt](#) for license information.

Remarks

Once the [init_params\(\)](#) subroutine is called, the parameters are loaded globally in the main program and its subroutines and function

6.4.2 Function/Subroutine Documentation

6.4.2.1 subroutine, private params::init_nml () [private]

Read the basic parameters and mode selection from the namelist.

Definition at line 91 of file params.f90.

```

91      INTEGER :: allocstat
92
93      namelist /aoscale/  scale,f0,n,rra,phi0_npi
94      namelist /oparams/  gp,r,h,d
95      namelist /aparams/  k,kp,sig0
96      namelist /toparams/ go,co,to0
97      namelist /taparams/ ga,ca,epsa,ta0
98      namelist /otparams/ sc,lambda,rr,sb
99
100     namelist /modeselection/ oms,ams
101     namelist /numblocs/  nboc,nbatm
102
103     namelist /int_params/ t_trans,t_run,dt,tw,writeout
104
105     OPEN(8, file="params.nml", status='OLD', recl=80, delim='APOSTROPHE')
106
107     READ(8,nml=aoscale)
108     READ(8,nml=oparams)
109     READ(8,nml=aparams)
110     READ(8,nml=toparams)
111     READ(8,nml=taparams)
112     READ(8,nml=otparams)
113
114     CLOSE(8)
115
116     OPEN(8, file="modeselection.nml", status='OLD', recl=80, delim='APOSTROPHE')
117     READ(8,nml=numblocs)
118
119     ALLOCATE(oms(nboc,2),ams(nbatm,2), stat=allocstat)
120     IF (allocstat /= 0) stop "*** Not enough memory ! ***"
121
122     READ(8,nml=modeselection)
123     CLOSE(8)
124
125     ! Must still sort modeselection here
126
127     OPEN(8, file="int_params.nml", status='OLD', recl=80, delim='APOSTROPHE')
128     READ(8,nml=int_params)
129
130
```

6.4.2.2 subroutine params::init_params ()

Parameters initialisation routine.

Definition at line 135 of file params.f90.

```

135     INTEGER, DIMENSION(2) :: s
136     INTEGER :: i
137     CALL init_nml
138
139     !-----!
140     !
141     ! Computation of the dimension of the atmospheric
142     ! and oceanic components
143     !
144     !-----!
145
146     natm=0
147     DO i=1,nbatm
148         IF (ams(i,1)==1) THEN
149             natm=natm+3
150         ELSE
151             natm=natm+2
152         ENDIF
153     END DO

```

```

153      ENDDO
154      s=shape(oms)
155      noc=s(1)
156
157      ndim=2*natm+2*noc
158
159      !-----!
160      !                                     !
161      ! Some general parameters (Domain, beta, gamma, coupling) !
162      !                                     !
163      !-----!
164
165      pi=dacos(-1.d0)
166      l=scale/pi
167      phi0=phi0_npi*pi
168      lr=sqrt(gp*h)/f0
169      g=-1**2/lr**2
170      betp=1/rra*cos(phi0)/sin(phi0)
171      rp=r/f0
172      dp=d/f0
173      kd=k*2
174      kdp=kp
175
176      !-----!
177      !                                     !
178      ! DERIVED QUANTITIES                                     !
179      !                                     !
180      !-----!
181
182      cpo=co/(go*f0) * rr/(f0**2*1**2)
183      lpo=lambda/(go*f0)
184      cpa=ca/(ga*f0) * rr/(f0**2*1**2)/2 ! Cpa acts on psil-psi3, not on theta
185      lpa=lambda/(ga*f0)
186      sbpo=4*sb*to0**3/(go*f0) ! long wave radiation lost by ocean to atmosphere space
187      sbpa=8*epsa*sb*ta0**3/(go*f0) ! long wave radiation from atmosphere absorbed by ocean
188      lsbo=2*epsa*sb*to0**3/(ga*f0) ! long wave radiation from ocean absorbed by atmosphere
189      lsba=8*epsa*sb*ta0**3/(ga*f0) ! long wave radiation lost by atmosphere to space & ocea
190
191

```

6.4.3 Variable Documentation

6.4.3.1 integer, dimension(:, :), allocatable params::ams

Atmospheric mode selection array.

Definition at line 81 of file params.f90.

```

81      INTEGER, DIMENSION(:, :), ALLOCATABLE :: ams      !< Atmospheric mode selection array

```

6.4.3.2 real(kind=8) params::betp

Definition at line 67 of file params.f90.

```

67      REAL(KIND=8) :: betp      ! \f$\beta$ - Non-dimensional beta parameter

```

6.4.3.3 real(kind=8) params::ca

C_a - Constant short-wave radiation of the atmosphere.

Definition at line 40 of file params.f90.

```

40      REAL(KIND=8) :: ca      !< \f$C_a$ - Constant short-wave radiation of the atmosphere.

```

6.4.3.4 `real(kind=8) params::co`

C_a - Constant short-wave radiation of the ocean.

Definition at line 38 of file params.f90.

```
38  REAL(KIND=8) :: co      !< \f$C_a\f$ - Constant short-wave radiation of the ocean.
```

6.4.3.5 `real(kind=8) params::cpa`

C'_a - Non-dimensional constant short-wave radiation of the atmosphere.

Remarks

Cpa acts on psi1-psi3, not on theta.

Definition at line 58 of file params.f90.

```
58  REAL(KIND=8) :: cpa      !< \f$C'_a\f$ - Non-dimensional constant short-wave radiation of the
    atmosphere. @remark Cpa acts on psi1-psi3, not on theta.
```

6.4.3.6 `real(kind=8) params::cpo`

C'_a - Non-dimensional constant short-wave radiation of the ocean.

Definition at line 56 of file params.f90.

```
56  REAL(KIND=8) :: cpo      !< \f$C'_a\f$ - Non-dimensional constant short-wave radiation of the ocean.
```

6.4.3.7 `real(kind=8) params::d`

Mechanical coupling parameter between the ocean and the atmosphere.

Definition at line 31 of file params.f90.

```
31  REAL(KIND=8) :: d      !< Mechanical coupling parameter between the ocean and the atmosphere.
```

6.4.3.8 `real(kind=8) params::dp`

d' - Non-dimensional mechanical coupling parameter between the ocean and the atmosphere.

Definition at line 52 of file params.f90.

```
52  REAL(KIND=8) :: dp      !< \f$d'\f$ - Non-dimensional mechanical coupling parameter between the ocean
    and the atmosphere.
```

6.4.3.9 real(kind=8) params::dt

Integration time step.

Definition at line 71 of file params.f90.

```
71  REAL(KIND=8) :: dt           !< Integration time step
```

6.4.3.10 real(kind=8) params::epsa

ϵ_a - Emissivity coefficient for the grey-body atmosphere.

Definition at line 43 of file params.f90.

```
43  REAL(KIND=8) :: epsa       !< \f$\epsilon_a\f$ - Emissivity coefficient for the grey-body atmosphere.
```

6.4.3.11 real(kind=8) params::f0

f_0 - Coriolis parameter

Definition at line 32 of file params.f90.

```
32  REAL(KIND=8) :: f0         !< \f$f_0\f$ - Coriolis parameter
```

6.4.3.12 real(kind=8) params::g

γ

Definition at line 50 of file params.f90.

```
50  REAL(KIND=8) :: g         !< \f$\gamma\f$
```

6.4.3.13 real(kind=8) params::ga

γ_a - Specific heat capacity of the atmosphere.

Definition at line 44 of file params.f90.

```
44  REAL(KIND=8) :: ga       !< \f$\gamma_a\f$ - Specific heat capacity of the atmosphere.
```

6.4.3.14 `real(kind=8) params::go`

γ_o - Specific heat capacity of the ocean.

Definition at line 39 of file `params.f90`.

```
39  REAL(KIND=8) :: go           !< \f$\gamma_o\f$ - Specific heat capacity of the ocean.
```

6.4.3.15 `real(kind=8) params::gp`

g' Reduced gravity

Definition at line 33 of file `params.f90`.

```
33  REAL(KIND=8) :: gp           !< \f$g'\f$Reduced gravity
```

6.4.3.16 `real(kind=8) params::h`

Depth of the active water layer of the ocean.

Definition at line 34 of file `params.f90`.

```
34  REAL(KIND=8) :: h           !< Depth of the active water layer of the ocean.
```

6.4.3.17 `real(kind=8) params::k`

Bottom atmospheric friction coefficient.

Definition at line 28 of file `params.f90`.

```
28  REAL(KIND=8) :: k           !< Bottom atmospheric friction coefficient.
```

6.4.3.18 `real(kind=8) params::kd`

k_d - Non-dimensional bottom atmospheric friction coefficient.

Definition at line 53 of file `params.f90`.

```
53  REAL(KIND=8) :: kd           !< \f$k_d\f$ - Non-dimensional bottom atmospheric friction coefficient.
```

6.4.3.19 real(kind=8) params::kdp

k'_d - Non-dimensional internal atmospheric friction coefficient.

Definition at line 54 of file params.f90.

```
54  REAL(KIND=8) :: kdp          !< \f$k'_d\f$ - Non-dimensional internal atmospheric friction coefficient.
```

6.4.3.20 real(kind=8) params::kp

k' - Internal atmospheric friction coefficient.

Definition at line 29 of file params.f90.

```
29  REAL(KIND=8) :: kp          !< \f$k'\f$ - Internal atmospheric friction coefficient.
```

6.4.3.21 real(kind=8) params::l

L - Domain length scale

Definition at line 64 of file params.f90.

```
64  REAL(KIND=8) :: l          !< \f$L\f$ - Domain length scale
```

6.4.3.22 real(kind=8) params::lambda

λ - Sensible + turbulent heat exchange between the ocean and the atmosphere.

Definition at line 37 of file params.f90.

```
37  REAL(KIND=8) :: lambda      !< \f$\lambda\f$ - Sensible + turbulent heat exchange between the ocean and the
    atmosphere.
```

6.4.3.23 real(kind=8) params::lpa

λ'_a - Non-dimensional sensible + turbulent heat exchange from atmosphere to ocean.

Definition at line 59 of file params.f90.

```
59  REAL(KIND=8) :: lpa          !< \f$\lambda'_a\f$ - Non-dimensional sensible + turbulent heat exchange from
    atmosphere to ocean.
```

6.4.3.24 real(kind=8) params::lpo

λ'_o - Non-dimensional sensible + turbulent heat exchange from ocean to atmosphere.

Definition at line 57 of file params.f90.

```
57  REAL(KIND=8) :: lpo      !< \f$\lambda'_o\f$ - Non-dimensional sensible + turbulent heat exchange from
    ocean to atmosphere.
```

6.4.3.25 real(kind=8) params::lr

L_R - Rossby deformation radius

Definition at line 49 of file params.f90.

```
49  REAL(KIND=8) :: lr      !< \f$L_R\f$ - Rossby deformation radius
```

6.4.3.26 real(kind=8) params::lsbpa

$S'_{B,a}$ - Long wave radiation lost by atmosphere to space & ocean.

Definition at line 63 of file params.f90.

```
63  REAL(KIND=8) :: lsbpa   !< \f$S'_{B,a}\f$ - Long wave radiation lost by atmosphere to space & ocean.
```

6.4.3.27 real(kind=8) params::lsbpo

$S'_{B,o}$ - Long wave radiation from ocean absorbed by atmosphere.

Definition at line 62 of file params.f90.

```
62  REAL(KIND=8) :: lsbpo   !< \f$S'_{B,o}\f$ - Long wave radiation from ocean absorbed by atmosphere.
```

6.4.3.28 real(kind=8) params::n

$n = 2L_y/L_x$ - Aspect ratio

Definition at line 24 of file params.f90.

```
24  REAL(KIND=8) :: n      !< \f$n = 2 L_y / L_x\f$ - Aspect ratio
```


6.4.3.29 integer params::natm =0

Number of atmospheric basis functions.

Definition at line 77 of file params.f90.

```
77  INTEGER :: natm=0 !< Number of atmospheric basis functions
```

6.4.3.30 integer params::nbatm

Number of oceanic blocks.

Definition at line 76 of file params.f90.

```
76  INTEGER :: nbatm !< Number of oceanic blocks
```

6.4.3.31 integer params::nboc

Number of atmospheric blocks.

Definition at line 75 of file params.f90.

```
75  INTEGER :: nboc !< Number of atmospheric blocks
```

6.4.3.32 integer params::ndim

Number of variables (dimension of the model)

Definition at line 79 of file params.f90.

```
79  INTEGER :: ndim !< Number of variables (dimension of the model)
```

6.4.3.33 integer params::noc =0

Number of oceanic basis functions.

Definition at line 78 of file params.f90.

```
78  INTEGER :: noc=0 !< Number of oceanic basis functions
```

6.4.3.34 integer, dimension(:, :), allocatable params::oms

Ocean mode selection array.

Definition at line 80 of file params.f90.

```
80  INTEGER, DIMENSION(:, :), ALLOCATABLE :: oms      !< Ocean mode selection array
```

6.4.3.35 real(kind=8) params::phi0

Latitude in radian.

Definition at line 25 of file params.f90.

```
25  REAL(KIND=8) :: phi0      !< Latitude in radian
```

6.4.3.36 real(kind=8) params::phi0_npi

Latitude exprimed in fraction of pi.

Definition at line 35 of file params.f90.

```
35  REAL(KIND=8) :: phi0_npi !< Latitude exprimed in fraction of pi.
```

6.4.3.37 real(kind=8) params::pi

π

Definition at line 48 of file params.f90.

```
48  REAL(KIND=8) :: pi      !< \f$\pi\f$
```

6.4.3.38 real(kind=8) params::r

Frictional coefficient at the bottom of the ocean.

Definition at line 30 of file params.f90.

```
30  REAL(KIND=8) :: r      !< Frictional coefficient at the bottom of the ocean.
```

6.4.3.39 real(kind=8) params::rp

r' - Frictional coefficient at the bottom of the ocean.

Definition at line 51 of file params.f90.

```
51  REAL(KIND=8) :: rp          !< \f$r'\f$ - Frictional coefficient at the bottom of the ocean.
```

6.4.3.40 real(kind=8) params::rr

R - Gas constant of dry air

Definition at line 45 of file params.f90.

```
45  REAL(KIND=8) :: rr          !< \f$R\f$ - Gas constant of dry air
```

6.4.3.41 real(kind=8) params::rra

Earth radius.

Definition at line 26 of file params.f90.

```
26  REAL(KIND=8) :: rra          !< Earth radius
```

6.4.3.42 real(kind=8) params::sb

Stefan–Boltzmann constant.

Definition at line 66 of file params.f90.

```
66  REAL(KIND=8) :: sb          !< Stefan-Boltzmann constant
```

6.4.3.43 real(kind=8) params::sbpa

$\sigma'_{B,a}$ - Long wave radiation from atmosphere absorbed by ocean.

Definition at line 61 of file params.f90.

```
61  REAL(KIND=8) :: sbpa        !< \f$\sigma'_{B,a}\f$ - Long wave radiation from atmosphere absorbed by ocean.
```

6.4.3.44 `real(kind=8) params::sbpo`

$\sigma'_{B,o}$ - Long wave radiation lost by ocean to atmosphere & space.

Definition at line 60 of file params.f90.

```
60  REAL(KIND=8) :: sbpo      !< \f$\sigma'_{B,o}\f$ - Long wave radiation lost by ocean to atmosphere &
    space.
```

6.4.3.45 `real(kind=8) params::sc`

Ratio of surface to atmosphere temperature.

Definition at line 65 of file params.f90.

```
65  REAL(KIND=8) :: sc      !< Ratio of surface to atmosphere temperature.
```

6.4.3.46 `real(kind=8) params::scale`

$L_y = L \pi$ - The characteristic space scale.

Definition at line 47 of file params.f90.

```
47  REAL(KIND=8) :: scale      !< \f$L_y = L \, \pi\f$ - The characteristic space scale.
```

6.4.3.47 `real(kind=8) params::sig0`

σ_0 - Non-dimensional static stability of the atmosphere.

Definition at line 27 of file params.f90.

```
27  REAL(KIND=8) :: sig0      !< \f$\sigma_0\f$ - Non-dimensional static stability of the atmosphere.
```

6.4.3.48 `real(kind=8) params::t_run`

Effective intergration time (length of the generated trajectory)

Definition at line 70 of file params.f90.

```
70  REAL(KIND=8) :: t_run      !< Effective intergration time (length of the generated trajectory)
```

6.4.3.49 real(kind=8) params::t_trans

Transient time period.

Definition at line 69 of file params.f90.

```
69  REAL(KIND=8) :: t_trans      !< Transient time period
```

6.4.3.50 real(kind=8) params::ta0

T_a^0 - Stationary solution for the 0-th order atmospheric temperature.

Definition at line 42 of file params.f90.

```
42  REAL(KIND=8) :: ta0          !< \f$T_a^0\f$ - Stationary solution for the 0-th order atmospheric
    temperature.
```

6.4.3.51 real(kind=8) params::to0

T_o^0 - Stationary solution for the 0-th order ocean temperature.

Definition at line 41 of file params.f90.

```
41  REAL(KIND=8) :: to0          !< \f$T_o^0\f$ - Stationary solution for the 0-th order ocean temperature.
```

6.4.3.52 real(kind=8) params::tw

Write all variables every tw time units.

Definition at line 72 of file params.f90.

```
72  REAL(KIND=8) :: tw          !< Write all variables every tw time units
```

6.4.3.53 logical params::writeout

Write to file boolean.

Definition at line 73 of file params.f90.

```
73  LOGICAL :: writeout         !< Write to file boolean
```

6.5 rk2_integrator Module Reference

Module with the integration routines.

Functions/Subroutines

- subroutine, public `init_integrator`
Routine to initialise the integration buffers.
- subroutine `tendencies` (t, y, res)
Routine computing the tendencies of the model.
- subroutine, public `step` (y, t, dt, res)
Routine to perform an integration step (Heun algorithm). The incremented time is returned.

Variables

- real(kind=8), dimension(:), allocatable `buf_y1`
Buffer to hold the intermediate position (Heun algorithm)
- real(kind=8), dimension(:), allocatable `buf_f0`
Buffer to hold tendencies at the initial position.
- real(kind=8), dimension(:), allocatable `buf_f1`
Buffer to hold tendencies at the intermediate position.

6.5.1 Detailed Description

Module with the integration routines.

Copyright

2015 Lesley De Cruz & Jonathan Demaeyer. See [LICENSE.txt](#) for license information.

Remarks

This module actually contains the Heun algorithm routines. The user can modify it according to its preferred integration scheme. For higher-order schemes, additional buffers will probably have to be defined.

6.5.2 Function/Subroutine Documentation

6.5.2.1 subroutine, public `rk2_integrator::init_integrator` ()

Routine to initialise the integration buffers.

Definition at line 37 of file `rk2_integrator.f90`.

```
37  INTEGER :: allocstat
38  ALLOCATE(buf_y1(0:ndim),buf_f0(0:ndim),buf_f1(0:ndim) ,stat=allocstat)
39  IF (allocstat /= 0) stop "*** Not enough memory ! ***"
```

6.5.2.2 subroutine, public `rk2_integrator::step` (real(kind=8), dimension(0:ndim), intent(in) y, real(kind=8), intent(inout) t, real(kind=8), intent(in) dt, real(kind=8), dimension(0:ndim), intent(out) res)

Routine to perform an integration step (Heun algorithm). The incremented time is returned.

Parameters

<i>y</i>	Initial point.
<i>t</i>	Actual integration time
<i>dt</i>	Integration timestep.
<i>res</i>	Final point after the step.

Definition at line 61 of file rk2_integrator.f90.

```

61     REAL(KIND=8), DIMENSION(0:ndim), INTENT(IN) :: y
62     REAL(KIND=8), INTENT(INOUT) :: t
63     REAL(KIND=8), INTENT(IN) :: dt
64     REAL(KIND=8), DIMENSION(0:ndim), INTENT(OUT) :: res
65
66     CALL tendencies(t,y,buf_f0)
67     buf_y1 = y+dt*buf_f0
68     CALL tendencies(t+dt,buf_y1,buf_f1)
69     res=y+0.5*(buf_f0+buf_f1)*dt
70     t=t+dt

```

6.5.2.3 subroutine rk2_integrator::tendencies (real(kind=8), intent(in) *t*, real(kind=8), dimension(0:ndim), intent(in) *y*,
real(kind=8), dimension(0:ndim), intent(out) *res*) [private]

Routine computing the tendencies of the model.

Parameters

<i>t</i>	Time at which the tendencies have to be computed. Actually not needed for autonomous systems.
<i>y</i>	Point at which the tendencies have to be computed.
<i>res</i>	vector to store the result.

Remarks

Note that it is NOT safe to pass *y* as a result buffer, as this operation does multiple passes.

Definition at line 49 of file rk2_integrator.f90.

```

49     REAL(KIND=8), INTENT(IN) :: t
50     REAL(KIND=8), DIMENSION(0:ndim), INTENT(IN) :: y
51     REAL(KIND=8), DIMENSION(0:ndim), INTENT(OUT) :: res
52     CALL sparse_mul3(aotensor, y, y, res)

```

6.5.3 Variable Documentation

6.5.3.1 real(kind=8), dimension(:), allocatable rk2_integrator::buf_f0 [private]

Buffer to hold tendencies at the initial position.

Definition at line 28 of file rk2_integrator.f90.

```

28     REAL(KIND=8), DIMENSION(:), ALLOCATABLE :: buf_f0 !< Buffer to hold tendencies at the initial position

```

6.5.3.2 `real(kind=8), dimension(:), allocatable rk2_integrator::buf_f1` [private]

Buffer to hold tendencies at the intermediate position.

Definition at line 29 of file `rk2_integrator.f90`.

```
29  REAL(KIND=8), DIMENSION(:), ALLOCATABLE :: buf_f1 !< Buffer to hold tendencies at the intermediate
    position
```

6.5.3.3 `real(kind=8), dimension(:), allocatable rk2_integrator::buf_y1` [private]

Buffer to hold the intermediate position (Heun algorithm)

Definition at line 27 of file `rk2_integrator.f90`.

```
27  REAL(KIND=8), DIMENSION(:), ALLOCATABLE :: buf_y1 !< Buffer to hold the intermediate position (Heun
    algorithm)
```

6.6 `rk2_tl_ad_integrator` Module Reference

Tangent Linear (TL) and Adjoint (AD) model versions of MAOOAM. Integrators module.

Functions/Subroutines

- subroutine, public `init_tl_ad_integrator`
Routine to initialise the integration buffers.
- subroutine, public `ad_step` (`y`, `ystar`, `t`, `dt`, `res`)
Routine to perform an integration step (Heun algorithm) of the adjoint model. The incremented time is returned.
- subroutine, public `tl_step` (`y`, `ystar`, `t`, `dt`, `res`)
Routine to perform an integration step (Heun algorithm) of the tangent linear model. The incremented time is returned.

Variables

- `real(kind=8), dimension(:), allocatable buf_y1`
Buffer to hold the intermediate position (Heun algorithm) of the tangent linear model.
- `real(kind=8), dimension(:), allocatable buf_f0`
Buffer to hold tendencies at the initial position of the tangent linear model.
- `real(kind=8), dimension(:), allocatable buf_f1`
Buffer to hold tendencies at the intermediate position of the tangent linear model.

6.6.1 Detailed Description

Tangent Linear (TL) and Adjoint (AD) model versions of MAOOAM. Integrators module.

Copyright

2016 Lesley De Cruz & Jonathan Demaeyer. See [LICENSE.txt](#) for license information.

Remarks

This module actually contains the Heun algorithm routines. The user can modify it according to its preferred integration scheme. For higher-order schemes, additional buffers will probably have to be defined.

6.6.2 Function/Subroutine Documentation

6.6.2.1 subroutine, public rk2_tl_ad_integrator::ad_step (real(kind=8), dimension(0:ndim), intent(in) *y*, real(kind=8), dimension(0:ndim), intent(in) *ystar*, real(kind=8), intent(inout) *t*, real(kind=8), intent(in) *dt*, real(kind=8), dimension(0:ndim), intent(out) *res*)

Routine to perform an integration step (Heun algorithm) of the adjoint model. The incremented time is returned.

Parameters

<i>y</i>	Initial point.
<i>ystar</i>	Adjoint model at the point ystar.
<i>t</i>	Actual integration time
<i>dt</i>	Integration timestep.
<i>res</i>	Final point after the step.

Definition at line 61 of file rk2_tl_ad_integrator.f90.

```

61      REAL(KIND=8), DIMENSION(0:ndim), INTENT(IN) :: y, ystar
62      REAL(KIND=8), INTENT(INOUT) :: t
63      REAL(KIND=8), INTENT(IN) :: dt
64      REAL(KIND=8), DIMENSION(0:ndim), INTENT(OUT) :: res
65
66      CALL ad(t, ystar, y, buf_f0)
67      buf_y1 = y + dt * buf_f0
68      CALL ad(t + dt, ystar, buf_y1, buf_f1)
69      res = y + 0.5 * (buf_f0 + buf_f1) * dt
70      t = t + dt

```

6.6.2.2 subroutine, public rk2_tl_ad_integrator::init_tl_ad_integrator ()

Routine to initialise the integration buffers.

Definition at line 41 of file rk2_tl_ad_integrator.f90.

```

41      INTEGER :: allocstat
42      ALLOCATE(buf_y1(0:ndim), buf_f0(0:ndim), buf_f1(0:ndim), stat=allocstat)
43      IF (allocstat /= 0) stop "*** Not enough memory ! ***"

```

6.6.2.3 subroutine, public rk2_tl_ad_integrator::tl_step (real(kind=8), dimension(0:ndim), intent(in) *y*, real(kind=8), dimension(0:ndim), intent(in) *ystar*, real(kind=8), intent(inout) *t*, real(kind=8), intent(in) *dt*, real(kind=8), dimension(0:ndim), intent(out) *res*)

Routine to perform an integration step (Heun algorithm) of the tangent linear model. The incremented time is returned.

Parameters

<i>y</i>	Initial point.
<i>ystar</i>	Adjoint model at the point ystar.
<i>t</i>	Actual integration time
<i>dt</i>	Integration timestep.
<i>res</i>	Final point after the step.

Definition at line 86 of file rk2_tl_ad_integrator.f90.

```

86     REAL(KIND=8), DIMENSION(0:ndim), INTENT(IN) :: y, ystar
87     REAL(KIND=8), INTENT(INOUT) :: t
88     REAL(KIND=8), INTENT(IN) :: dt
89     REAL(KIND=8), DIMENSION(0:ndim), INTENT(OUT) :: res
90
91     CALL t1(t, ystar, y, buf_f0)
92     buf_y1 = y+dt*buf_f0
93     CALL t1(t+dt, ystar, buf_y1, buf_f1)
94     res=y+0.5*(buf_f0+buf_f1)*dt
95     t=t+dt

```

6.6.3 Variable Documentation

6.6.3.1 `real(kind=8), dimension(:), allocatable rk2_tl_ad_integrator::buf_f0` [private]

Buffer to hold tendencies at the initial position of the tangent linear model.

Definition at line 31 of file rk2_tl_ad_integrator.f90.

```

31     REAL(KIND=8), DIMENSION(:), ALLOCATABLE :: buf_f0 !< Buffer to hold tendencies at the initial position of
        the tangent linear model

```

6.6.3.2 `real(kind=8), dimension(:), allocatable rk2_tl_ad_integrator::buf_f1` [private]

Buffer to hold tendencies at the intermediate position of the tangent linear model.

Definition at line 32 of file rk2_tl_ad_integrator.f90.

```

32     REAL(KIND=8), DIMENSION(:), ALLOCATABLE :: buf_f1 !< Buffer to hold tendencies at the intermediate
        position of the tangent linear model

```

6.6.3.3 `real(kind=8), dimension(:), allocatable rk2_tl_ad_integrator::buf_y1` [private]

Buffer to hold the intermediate position (Heun algorithm) of the tangent linear model.

Definition at line 30 of file rk2_tl_ad_integrator.f90.

```

30     REAL(KIND=8), DIMENSION(:), ALLOCATABLE :: buf_y1 !< Buffer to hold the intermediate position (Heun
        algorithm) of the tangent linear model

```

6.7 rk4_integrator Module Reference

Module with the RK4 integration routines.

Functions/Subroutines

- subroutine, public [init_integrator](#)
Routine to initialise the integration buffers.
- subroutine [tendencies](#) (t, y, res)
Routine computing the tendencies of the model.
- subroutine, public [step](#) (y, t, dt, res)
Routine to perform an integration step (RK4 algorithm). The incremented time is returned.

Variables

- real(kind=8), dimension(:), allocatable [buf_y1](#)
Buffer to hold the intermediate position (Heun algorithm)
- real(kind=8), dimension(:), allocatable [buf_ka](#)
Buffer A to hold tendencies.
- real(kind=8), dimension(:), allocatable [buf_kb](#)
Buffer B to hold tendencies.

6.7.1 Detailed Description

Module with the RK4 integration routines.

Copyright

2015 Lesley De Cruz & Jonathan Demaeyer. See [LICENSE.txt](#) for license information.

Remarks

This module actually contains the RK4 algorithm routines. The user can modify it according to its preferred integration scheme. For higher-order schemes, additional buffers will probably have to be defined.

6.7.2 Function/Subroutine Documentation

6.7.2.1 subroutine, public rk4_integrator::init_integrator ()

Routine to initialise the integration buffers.

Definition at line 37 of file rk4_integrator.f90.

```
37  INTEGER :: allocstat
38  ALLOCATE(buf_y1(0:ndim),buf_ka(0:ndim),buf_kb(0:ndim) ,stat=allocstat)
39  IF (allocstat /= 0) stop "*** Not enough memory ! ***"
```

6.7.2.2 subroutine, public rk4_integrator::step (real(kind=8), dimension(0:ndim), intent(in) y, real(kind=8), intent(inout) t, real(kind=8), intent(in) dt, real(kind=8), dimension(0:ndim), intent(out) res)

Routine to perform an integration step (RK4 algorithm). The incremented time is returned.

Parameters

<i>y</i>	Initial point.
<i>t</i>	Actual integration time
<i>dt</i>	Integration timestep.
<i>res</i>	Final point after the step.

Definition at line 61 of file rk4_integrator.f90.

```

61     REAL(KIND=8), DIMENSION(0:ndim), INTENT(IN) :: y
62     REAL(KIND=8), INTENT(INOUT) :: t
63     REAL(KIND=8), INTENT(IN) :: dt
64     REAL(KIND=8), DIMENSION(0:ndim), INTENT(OUT) :: res
65
66     CALL tendencies(t,y,buf_ka)
67     buf_y1 = y + 0.5*dt*buf_ka
68
69     CALL tendencies(t+0.5*dt,buf_y1,buf_kb)
70     buf_y1 = y + 0.5*dt*buf_kb
71     buf_ka = buf_ka + 2*buf_kb
72
73     CALL tendencies(t+0.5*dt,buf_y1,buf_kb)
74     buf_y1 = y + dt*buf_kb
75     buf_ka = buf_ka + 2*buf_kb
76
77     CALL tendencies(t+dt,buf_y1,buf_kb)
78     buf_ka = buf_ka + buf_kb
79
80     t=t+dt
81     res=y+buf_ka*dt/6

```

6.7.2.3 subroutine rk4_integrator::tendencies (real(kind=8), intent(in) *t*, real(kind=8), dimension(0:ndim), intent(in) *y*,
real(kind=8), dimension(0:ndim), intent(out) *res*) [private]

Routine computing the tendencies of the model.

Parameters

<i>t</i>	Time at which the tendencies have to be computed. Actually not needed for autonomous systems.
<i>y</i>	Point at which the tendencies have to be computed.
<i>res</i>	vector to store the result.

Remarks

Note that it is NOT safe to pass *y* as a result buffer, as this operation does multiple passes.

Definition at line 49 of file rk4_integrator.f90.

```

49     REAL(KIND=8), INTENT(IN) :: t
50     REAL(KIND=8), DIMENSION(0:ndim), INTENT(IN) :: y
51     REAL(KIND=8), DIMENSION(0:ndim), INTENT(OUT) :: res
52     CALL sparse_mul3(aotensor, y, y, res)

```

6.7.3 Variable Documentation

6.7.3.1 real(kind=8), dimension(:), allocatable rk4_integrator::buf_ka [private]

Buffer A to hold tendencies.

Definition at line 28 of file rk4_integrator.f90.

```
28  REAL(KIND=8), DIMENSION(:), ALLOCATABLE :: buf_ka !< Buffer A to hold tendencies
```

6.7.3.2 `real(kind=8), dimension(:), allocatable rk4_integrator::buf_kb` [private]

Buffer B to hold tendencies.

Definition at line 29 of file rk4_integrator.f90.

```
29  REAL(KIND=8), DIMENSION(:), ALLOCATABLE :: buf_kb !< Buffer B to hold tendencies
```

6.7.3.3 `real(kind=8), dimension(:), allocatable rk4_integrator::buf_y1` [private]

Buffer to hold the intermediate position (Heun algorithm)

Definition at line 27 of file rk4_integrator.f90.

```
27  REAL(KIND=8), DIMENSION(:), ALLOCATABLE :: buf_y1 !< Buffer to hold the intermediate position (Heun
    algorithm)
```

6.8 rk4_tl_ad_integrator Module Reference

Tangent Linear (TL) and Adjoint (AD) model versions of MAOOAM. Integrators module.

Functions/Subroutines

- subroutine, public `init_tl_ad_integrator`
Routine to initialise the TL-AD integration buffers.
- subroutine, public `ad_step` (y, ystar, t, dt, res)
Routine to perform an integration step (RK4 algorithm) of the adjoint model. The incremented time is returned.
- subroutine, public `tl_step` (y, ystar, t, dt, res)
Routine to perform an integration step (RK4 algorithm) of the tangent linear model. The incremented time is returned.

Variables

- `real(kind=8), dimension(:), allocatable buf_y1`
Buffer to hold the intermediate position of the tangent linear model.
- `real(kind=8), dimension(:), allocatable buf_ka`
Buffer to hold tendencies in the RK4 scheme for the tangent linear model.
- `real(kind=8), dimension(:), allocatable buf_kb`
Buffer to hold tendencies in the RK4 scheme for the tangent linear model.

6.8.1 Detailed Description

Tangent Linear (TL) and Adjoint (AD) model versions of MAOOAM. Integrators module.

Copyright

2016 Lesley De Cruz, Jonathan Demaeyer & Sebastian Schubert. See [LICENSE.txt](#) for license information.

Remarks

This module actually contains the RK4 algorithm routines. The user can modify it according to its preferred integration scheme. For higher-order schemes, additional buffers will probably have to be defined.

6.8.2 Function/Subroutine Documentation

6.8.2.1 subroutine, public `rk4_tl_ad_integrator::ad_step` (`real(kind=8)`, `dimension(0:ndim)`, `intent(in)` `y`, `real(kind=8)`, `dimension(0:ndim)`, `intent(in)` `ystar`, `real(kind=8)`, `intent(inout)` `t`, `real(kind=8)`, `intent(in)` `dt`, `real(kind=8)`, `dimension(0:ndim)`, `intent(out)` `res`)

Routine to perform an integration step (RK4 algorithm) of the adjoint model. The incremented time is returned.

Parameters

<code>y</code>	Initial point.
<code>ystar</code>	Adjoint model at the point ystar.
<code>t</code>	Actual integration time
<code>dt</code>	Integration timestep.
<code>res</code>	Final point after the step.

Definition at line 65 of file `rk4_tl_ad_integrator.f90`.

```

65     REAL(KIND=8), DIMENSION(0:ndim), INTENT(IN) :: y, ystar
66     REAL(KIND=8), INTENT(INOUT) :: t
67     REAL(KIND=8), INTENT(IN) :: dt
68     REAL(KIND=8), DIMENSION(0:ndim), INTENT(OUT) :: res
69
70     CALL ad(t, ystar, y, buf_ka)
71     buf_y1 = y + 0.5*dt*buf_ka
72     CALL ad(t+0.5*dt, ystar, buf_y1, buf_kb)
73     buf_y1 = y + 0.5*dt*buf_kb
74     buf_ka = buf_ka + 2*buf_kb
75     CALL ad(t+0.5*dt, ystar, buf_y1, buf_kb)
76     buf_y1 = y + 0.5*dt*buf_kb
77     buf_ka = buf_ka + 2*buf_kb
78     CALL ad(t+dt, ystar, buf_y1, buf_kb)
79     buf_ka = buf_ka + buf_kb
80     res = y + buf_ka*dt/6
81     t = t + dt

```

6.8.2.2 subroutine, public `rk4_tl_ad_integrator::init_tl_ad_integrator` ()

Routine to initialise the TL-AD integration buffers.

Definition at line 43 of file `rk4_tl_ad_integrator.f90`.

```

43     INTEGER :: allocstat
44     ALLOCATE(buf_y1(0:ndim), buf_ka(0:ndim), buf_kb(0:ndim), stat=allocstat)
45     IF (allocstat /= 0) stop "*** Not enough memory ! ***"

```

6.8.2.3 subroutine, public rk4_tl_ad_integrator::tl_step (real(kind=8), dimension(0:ndim), intent(in) y, real(kind=8), dimension(0:ndim), intent(in) ystar, real(kind=8), intent(inout) t, real(kind=8), intent(in) dt, real(kind=8), dimension(0:ndim), intent(out) res)

Routine to perform an integration step (RK4 algorithm) of the tangent linear model. The incremented time is returned.

Parameters

<i>y</i>	Initial point.
<i>ystar</i>	Adjoint model at the point ystar.
<i>t</i>	Actual integration time
<i>dt</i>	Integration timestep.
<i>res</i>	Final point after the step.

Definition at line 98 of file rk4_tl_ad_integrator.f90.

```

98      REAL(KIND=8), DIMENSION(0:ndim), INTENT(IN) :: y, ystar
99      REAL(KIND=8), INTENT(INOUT) :: t
100     REAL(KIND=8), INTENT(IN) :: dt
101     REAL(KIND=8), DIMENSION(0:ndim), INTENT(OUT) :: res
102
103     CALL tl(t, ystar, y, buf_ka)
104     buf_y1 = y + 0.5*dt*buf_ka
105     CALL tl(t+0.5*dt, ystar, buf_y1, buf_kb)
106     buf_y1 = y + 0.5*dt*buf_kb
107     buf_ka = buf_ka + 2*buf_kb
108     CALL tl(t+0.5*dt, ystar, buf_y1, buf_kb)
109     buf_y1 = y + 0.5*dt*buf_kb
110     buf_ka = buf_ka + 2*buf_kb
111     CALL tl(t+dt, ystar, buf_y1, buf_kb)
112     buf_ka = buf_ka + buf_kb
113     res = y + buf_ka*dt/6
114     t = t + dt

```

6.8.3 Variable Documentation

6.8.3.1 real(kind=8), dimension(:), allocatable rk4_tl_ad_integrator::buf_ka [private]

Buffer to hold tendencies in the RK4 scheme for the tangent linear model.

Definition at line 33 of file rk4_tl_ad_integrator.f90.

```

33      REAL(KIND=8), DIMENSION(:), ALLOCATABLE :: buf_ka !< Buffer to hold tendencies in the RK4 scheme for the
      tangent linear model

```

6.8.3.2 real(kind=8), dimension(:), allocatable rk4_tl_ad_integrator::buf_kb [private]

Buffer to hold tendencies in the RK4 scheme for the tangent linear model.

Definition at line 34 of file rk4_tl_ad_integrator.f90.

```

34      REAL(KIND=8), DIMENSION(:), ALLOCATABLE :: buf_kb !< Buffer to hold tendencies in the RK4 scheme for the
      tangent linear model

```

6.8.3.3 `real(kind=8), dimension(:), allocatable rk4_tl_ad_integrator::buf_y1` [private]

Buffer to hold the intermediate position of the tangent linear model.

Definition at line 32 of file `rk4_tl_ad_integrator.f90`.

```
32  REAL(KIND=8), DIMENSION(:), ALLOCATABLE :: buf_y1 !< Buffer to hold the intermediate position of the
    tangent linear model
```

6.9 stat Module Reference

Statistics accumulators.

Functions/Subroutines

- subroutine, public `init_stat`
Initialise the accumulators.
- subroutine, public `acc` (x)
Accumulate one state.
- `real(kind=8)` function, `dimension(0:ndim)`, public `mean` ()
Function returning the mean.
- `real(kind=8)` function, `dimension(0:ndim)`, public `var` ()
Function returning the variance.
- integer function, public `iter` ()
Function returning the number of data accumulated.
- subroutine, public `reset`
Routine resetting the accumulators.

Variables

- integer `i` =0
Number of stats accumulated.
- `real(kind=8)`, `dimension(:)`, allocatable `m`
Vector storing the inline mean.
- `real(kind=8)`, `dimension(:)`, allocatable `mprev`
Previous mean vector.
- `real(kind=8)`, `dimension(:)`, allocatable `v`
Vector storing the inline variance.
- `real(kind=8)`, `dimension(:)`, allocatable `mtmp`

6.9.1 Detailed Description

Statistics accumulators.

Copyright

2015 Lesley De Cruz & Jonathan Demaeyer. See [LICENSE.txt](#) for license information.

6.9.2 Function/Subroutine Documentation

6.9.2.1 subroutine, public stat::acc (real(kind=8), dimension(0:ndim), intent(in) x)

Accumulate one state.

Definition at line 48 of file stat.f90.

```

48      IMPLICIT NONE
49      REAL(KIND=8), DIMENSION(0:ndim), INTENT(IN) :: x
50      i=i+1
51      mprev=m+(x-m)/i
52      mtmp=mprev
53      mprev=m
54      m=mtmp
55      v=v+(x-mprev)*(x-m)

```

6.9.2.2 subroutine, public stat::init_stat ()

Initialise the accumulators.

Definition at line 35 of file stat.f90.

```

35      INTEGER :: allocstat
36
37      ALLOCATE(m(0:ndim),mprev(0:ndim),v(0:ndim),mtmp(0:ndim), stat=allocstat)
38      IF (allocstat /= 0) stop '*** Not enough memory ***'
39      m=0.d0
40      mprev=0.d0
41      v=0.d0
42      mtmp=0.d0
43

```

6.9.2.3 integer function, public stat::iter ()

Function returning the number of data accumulated.

Definition at line 72 of file stat.f90.

```

72      INTEGER :: iter
73      iter=i

```

6.9.2.4 real(kind=8) function, dimension(0:ndim), public stat::mean ()

Function returning the mean.

Definition at line 60 of file stat.f90.

```

60      REAL(KIND=8), DIMENSION(0:ndim) :: mean
61      mean=m

```

6.9.2.5 subroutine, public stat::reset ()

Routine resetting the accumulators.

Definition at line 78 of file stat.f90.

```
78      m=0.d0
79      mprev=0.d0
80      v=0.d0
81      i=0
```

6.9.2.6 real(kind=8) function, dimension(0:ndim), public stat::var ()

Function returning the variance.

Definition at line 66 of file stat.f90.

```
66      REAL(KIND=8), DIMENSION(0:ndim) :: var
67      var=v/(i-1)
```

6.9.3 Variable Documentation

6.9.3.1 integer stat::i =0 [private]

Number of stats accumulated.

Definition at line 20 of file stat.f90.

```
20  INTEGER :: i=0 !< Number of stats accumulated
```

6.9.3.2 real(kind=8), dimension(:), allocatable stat::m [private]

Vector storing the inline mean.

Definition at line 23 of file stat.f90.

```
23  REAL(KIND=8), DIMENSION(:), ALLOCATABLE :: m !< Vector storing the inline mean
```

6.9.3.3 real(kind=8), dimension(:), allocatable stat::mprev [private]

Previous mean vector.

Definition at line 24 of file stat.f90.

```
24  REAL(KIND=8), DIMENSION(:), ALLOCATABLE :: mprev !< Previous mean vector
```

6.9.3.4 `real(kind=8), dimension(:), allocatable stat::mtmp` [private]

Definition at line 26 of file stat.f90.

```
26  REAL(KIND=8), DIMENSION(:), ALLOCATABLE :: mtmp
```

6.9.3.5 `real(kind=8), dimension(:), allocatable stat::v` [private]

Vector storing the inline variance.

Definition at line 25 of file stat.f90.

```
25  REAL(KIND=8), DIMENSION(:), ALLOCATABLE :: v      !< Vector storing the inline variance
```

6.10 tensor Module Reference

Tensor utility module.

Data Types

- type `coolist`
Coordinate list. Type used to represent the sparse tensor.
- type `coolist_elem`
Coordinate list element type. Elementary elements of the sparse tensors.

Functions/Subroutines

- subroutine, public `copy_tensor` (src, dst)
Routine to copy a tensor.
- subroutine, public `mat_to_coo` (src, dst)
Routine to convert a matrix to a tensor.
- subroutine, public `sparse_mul3` (coolist_ijk, arr_j, arr_k, res)
Sparse multiplication of a tensor with two vectors: $\sum_{j,k=0}^{ndim} \mathcal{T}_{i,j,k} a_j b_k$.
- subroutine, public `jsparse_mul` (coolist_ijk, arr_j, jcoo_ij)
Sparse multiplication of two tensors to determine the Jacobian:

$$J_{i,j} = \sum_{k=0}^{ndim} (\mathcal{T}_{i,j,k} + \mathcal{T}_{i,k,j}) a_k.$$

It's implemented slightly differently: for every $\mathcal{T}_{i,j,k}$, we add to $J_{i,j}$ as follows:

$$J_{i,j} = J_{i,j} + \mathcal{T}_{i,j,k} a_k J_{i,k} = J_{i,k} + \mathcal{T}_{i,j,k} a_j$$

This version return a coolist (sparse tensor).

- subroutine, public `jsparse_mul_mat` (coolist_ijk, arr_j, jcoo_ij)

Sparse multiplication of two tensors to determine the Jacobian:

$$J_{i,j} = \sum_{k=0}^{ndim} (\mathcal{T}_{i,j,k} + \mathcal{T}_{i,k,j}) a_k.$$

It's implemented slightly differently: for every $\mathcal{T}_{i,j,k}$, we add to $J_{i,j}$ as follows:

$$J_{i,j} = J_{i,j} + \mathcal{T}_{i,j,k} a_k \quad J_{i,k} = J_{i,k} + \mathcal{T}_{i,j,k} a_j$$

This version return a matrix.

- subroutine, public `sparse_mul2` (coolist_ij, arr_j, res)

Sparse multiplication of a 2d sparse tensor with a vectors: $\sum_{j=0}^{ndim} \mathcal{T}_{i,j,k} a_j b_k$.

- subroutine, public `simplify` (tensor)

Routine to simplify a coolist (sparse tensor). For each index i , it upper triangularize the matrix

$$\mathcal{T}_{i,j,k} \quad 0 \leq j, k \leq ndim.$$

Variables

- real(kind=8), parameter `real_eps` = 2.2204460492503131e-16

Parameter to test the equality with zero.

6.10.1 Detailed Description

Tensor utility module.

Copyright

2015 Lesley De Cruz & Jonathan Demaeyer. See [LICENSE.txt](#) for license information.

6.10.2 Function/Subroutine Documentation

- 6.10.2.1 subroutine, public `tensor::copy_tensor` (type(coolist), dimension(ndim), intent(in) *src*, type(coolist), dimension(ndim), intent(out) *dst*)

Routine to copy a tensor.

Parameters

<i>src</i>	Source tensor
<i>dst</i>	Destination tensor

Remarks

The destination tensor have to be an empty tensor, i.e. with unallocated list of elements and nelems set to 0.

Definition at line 44 of file tensor.f90.

```

44     TYPE(coolist), DIMENSION(ndim), INTENT(IN) :: src
45     TYPE(coolist), DIMENSION(ndim), INTENT(OUT) :: dst
46     INTEGER :: i,j,allocstat
47
48     DO i=1,ndim
49         IF (dst(i)%nelems/=0) stop "*** copy_tensor : Destination coolist not empty ! ***"
50         ALLOCATE(dst(i)%elems(src(i)%nelems), stat=allocstat)
51         IF (allocstat /= 0) stop "*** Not enough memory ! ***"
52         DO j=1,src(i)%nelems
53             dst(i)%elems(j)%j=src(i)%elems(j)%j
54             dst(i)%elems(j)%k=src(i)%elems(j)%k
55             dst(i)%elems(j)%v=src(i)%elems(j)%v
56         ENDDO
57         dst(i)%nelems=src(i)%nelems
58     ENDDO

```

6.10.2.2 subroutine, public tensor::jsparse_mul (type(coolist), dimension(ndim), intent(in) coolist_ijk, real(kind=8), dimension(0:ndim), intent(in) arr_j, type(coolist), dimension(ndim), intent(out) jcoo_ij)

Sparse multiplication of two tensors to determine the Jacobian:

$$J_{i,j} = \sum_{k=0}^{ndim} (\mathcal{T}_{i,j,k} + \mathcal{T}_{i,k,j}) a_k.$$

It's implemented slightly differently: for every $\mathcal{T}_{i,j,k}$, we add to $J_{i,j}$ as follows:

$$J_{i,j} = J_{i,j} + \mathcal{T}_{i,j,k} a_k J_{i,k} = J_{i,k} + \mathcal{T}_{i,j,k} a_j$$

This version return a coolist (sparse tensor).

Parameters

<i>coolist_ijk</i>	a coordinate list (sparse tensor) of which index 2 or 3 will be contracted.
<i>arr_j</i>	the vector to be contracted with index 2 and then index 3 of ffi_coo_ijk
<i>jcoo_ij</i>	a coolist (sparse tensor) to store the result of the contraction

Definition at line 123 of file tensor.f90.

```

123     TYPE(coolist), DIMENSION(ndim), INTENT(IN) :: coolist_ijk
124     TYPE(coolist), DIMENSION(ndim), INTENT(OUT) :: jcoo_ij
125     REAL(KIND=8), DIMENSION(0:ndim), INTENT(IN) :: arr_j
126     REAL(KIND=8) :: v
127     INTEGER :: i,j,k,n,nj,allocstat
128
129     DO i=1,ndim
130         IF (jcoo_ij(i)%nelems/=0) stop "*** jsparse_mul : Destination coolist not empty ! ***"
131         nj=2*coolist_ijk(i)%nelems
132         ALLOCATE(jcoo_ij(i)%elems(nj), stat=allocstat)
133         IF (allocstat /= 0) stop "*** Not enough memory ! ***"
134         nj=0
135         DO n=1,coolist_ijk(i)%nelems
136             j=coolist_ijk(i)%elems(n)%j
137             k=coolist_ijk(i)%elems(n)%k
138             v=coolist_ijk(i)%elems(n)%v
139             IF (j /=0) THEN
140                 nj=nj+1
141                 jcoo_ij(i)%elems(nj)%j=j
142                 jcoo_ij(i)%elems(nj)%k=0
143                 jcoo_ij(i)%elems(nj)%v=v*arr_j(k)
144             END IF
145             IF (k /=0) THEN
146                 nj=nj+1
147                 jcoo_ij(i)%elems(nj)%j=k
148                 jcoo_ij(i)%elems(nj)%k=0
149                 jcoo_ij(i)%elems(nj)%v=v*arr_j(j)
150             END IF
151         END DO
152         jcoo_ij(i)%nelems=nj
153     ENDDO

```

6.10.2.3 subroutine, public `tensor::jsparse_mul_mat (type(coolist), dimension(ndim), intent(in) coolist_ijk, real(kind=8), dimension(0:ndim), intent(in) arr_j, real(kind=8), dimension(ndim,ndim), intent(out) jcoo_ij)`

Sparse multiplication of two tensors to determine the Jacobian:

$$J_{i,j} = \sum_{k=0}^{ndim} (\mathcal{T}_{i,j,k} + \mathcal{T}_{i,k,j}) a_k.$$

It's implemented slightly differently: for every $\mathcal{T}_{i,j,k}$, we add to $J_{i,j}$ as follows:

$$J_{i,j} = J_{i,j} + \mathcal{T}_{i,j,k} a_k J_{i,k} = J_{i,k} + \mathcal{T}_{i,j,k} a_j$$

This version return a matrix.

Parameters

<i>coolist_ijk</i>	a coordinate list (sparse tensor) of which index 2 or 3 will be contracted.
<i>arr_j</i>	the vector to be contracted with index 2 and then index 3 of <i>ffi_coo_ijk</i>
<i>jcoo_ij</i>	a matrix to store the result of the contraction

Definition at line 166 of file `tensor.f90`.

```

166  TYPE(coolist), DIMENSION(ndim), INTENT(IN) :: coolist_ijk
167  REAL(KIND=8), DIMENSION(ndim,ndim), INTENT(OUT) :: jcoo_ij
168  REAL(KIND=8), DIMENSION(0:ndim), INTENT(IN) :: arr_j
169  REAL(KIND=8) :: v
170  INTEGER :: i,j,k,n
171  jcoo_ij=0.d0
172  DO i=1,ndim
173      DO n=1,coolist_ijk(i)%elems
174          j=coolist_ijk(i)%elems(n)%j
175          k=coolist_ijk(i)%elems(n)%k
176          v=coolist_ijk(i)%elems(n)%v
177          IF (j /=0) jcoo_ij(i,j)=jcoo_ij(i,j)+v*arr_j(k)
178          IF (k /=0) jcoo_ij(i,k)=jcoo_ij(i,k)+v*arr_j(j)
179      END DO
180  END DO

```

6.10.2.4 subroutine, public `tensor::mat_to_coo (real(kind=8), dimension(0:ndim,0:ndim), intent(in) src, type(coolist), dimension(ndim), intent(out) dst)`

Routine to convert a matrix to a tensor.

Parameters

<i>src</i>	Source matrix
<i>dst</i>	Destination tensor

Remarks

The destination tensor have to be an empty tensor, i.e. with unallocated list of elements and *nelems* set to 0.

Definition at line 66 of file `tensor.f90`.

```

66     REAL(KIND=8), DIMENSION(0:ndim,0:ndim), INTENT(IN) :: src
67     TYPE(coolist), DIMENSION(ndim), INTENT(OUT) :: dst
68     INTEGER :: i,j,n,allocstat
69     DO i=1,ndim
70         n=0
71         DO j=1,ndim
72             IF (abs(src(i,j))>real_eps) n=n+1
73         ENDDO
74         IF (dst(i)%elems/=0) stop "*** mat_to_coo : Destination coolist not empty ! ***"
75         ALLOCATE(dst(i)%elems(n), stat=allocstat)
76         IF (allocstat /= 0) stop "*** Not enough memory ! ***"
77         n=0
78         DO j=1,ndim
79             IF (abs(src(i,j))>real_eps) THEN
80                 n=n+1
81                 dst(i)%elems(n)%j=j
82                 dst(i)%elems(n)%k=0
83                 dst(i)%elems(n)%v=src(i,j)
84             ENDIF
85         ENDDO
86         dst(i)%elems=n
87     ENDDO

```

6.10.2.5 subroutine, public tensor::simplify (type(coolist), dimension(ndim), intent(inout) tensor)

Routine to simplify a coolist (sparse tensor). For each index i , it upper triangularize the matrix

$$\mathcal{T}_{i,j,k} \quad 0 \leq j,k \leq ndim.$$

.

Parameters

<i>tensor</i>	a coordinate list (sparse tensor) which will be simplified.
---------------	---

Definition at line 208 of file tensor.f90.

```

208     TYPE(coolist), DIMENSION(ndim), INTENT(INOUT) :: tensor
209     INTEGER :: i,j,k
210     INTEGER :: li,lii,liii,n
211     DO i= 1,ndim
212         n=tensor(i)%elems
213         DO li=n,2,-1
214             j=tensor(i)%elems(li)%j
215             k=tensor(i)%elems(li)%k
216             DO lii=li-1,1,-1
217                 IF ((j==tensor(i)%elems(lii)%j).AND.(k==tensor(i)%elems(lii)%k)) THEN
218                     ! Found another entry with the same i,j,k: merge both into
219                     ! the one listed first (of those two).
220                     tensor(i)%elems(lii)%v=tensor(i)%elems(lii)%v+tensor(i)%elems(li)%v
221                     ! Shift the rest of the items one place down.
222                     DO liii=li+1,n
223                         tensor(i)%elems(liii-1)%j=tensor(i)%elems(lii)%j
224                         tensor(i)%elems(liii-1)%k=tensor(i)%elems(lii)%k
225                         tensor(i)%elems(liii-1)%v=tensor(i)%elems(lii)%v
226                     END DO
227                     tensor(i)%elems=tensor(i)%elems-1
228                     ! Here we should stop because the li no longer points to the
229                     ! original i,j,k element
230                     EXIT
231                 ENDIF
232             ENDDO
233         ENDDO
234         n=tensor(i)%elems
235         DO li=1,n
236             ! Clear new "almost" zero entries and shift rest of the items one place down.
237             ! Make sure not to skip any entries while shifting!
238             DO WHILE (abs(tensor(i)%elems(li)%v) < real_eps)
239                 DO liii=li+1,n
240                     tensor(i)%elems(liii-1)%j=tensor(i)%elems(lii)%j
241                     tensor(i)%elems(liii-1)%k=tensor(i)%elems(lii)%k
242                     tensor(i)%elems(liii-1)%v=tensor(i)%elems(lii)%v
243                 ENDDO

```

```

244         tensor(i)%nelems=tensor(i)%nelems-1
245     ENDDO
246 ENDDO
247
248 ENDDO

```

6.10.2.6 subroutine, public `tensor::sparse_mul2 (type(coolist), dimension(ndim), intent(in) coolist_ij, real(kind=8), dimension(0:ndim), intent(in) arr_j, real(kind=8), dimension(0:ndim), intent(out) res)`

Sparse multiplication of a 2d sparse tensor with a vectors: $\sum_{j=0}^{ndim} \mathcal{T}_{i,j,k} a_j b_k$.

Parameters

<i>coolist_ij</i>	a coordinate list (sparse tensor) of which index 2 will be contracted.
<i>arr_j</i>	the vector to be contracted with index 2 of coolist_ijk
<i>res</i>	vector (buffer) to store the result of the contraction

Remarks

Note that it is NOT safe to pass `arr_j` as a result buffer, as this operation does multiple passes.

Definition at line 191 of file tensor.f90.

```

191  TYPE(coolist), DIMENSION(ndim), INTENT(IN):: coolist_ij
192  REAL(KIND=8), DIMENSION(0:ndim), INTENT(IN) :: arr_j
193  REAL(KIND=8), DIMENSION(0:ndim), INTENT(OUT) :: res
194  INTEGER :: i,j,n
195  res=0.d0
196  DO i=1,ndim
197      DO n=1,coolist_ij(i)%nelems
198          j=coolist_ij(i)%elems(n)%j
199          res(i) = res(i) + coolist_ij(i)%elems(n)%v * arr_j(j)
200      END DO
201  END DO

```

6.10.2.7 subroutine, public `tensor::sparse_mul3 (type(coolist), dimension(ndim), intent(in) coolist_ijk, real(kind=8), dimension(0:ndim), intent(in) arr_j, real(kind=8), dimension(0:ndim), intent(in) arr_k, real(kind=8), dimension(0:ndim), intent(out) res)`

Sparse multiplication of a tensor with two vectors: $\sum_{j,k=0}^{ndim} \mathcal{T}_{i,j,k} a_j b_k$.

Parameters

<i>coolist_ijk</i>	a coordinate list (sparse tensor) of which index 2 and 3 will be contracted.
<i>arr_j</i>	the vector to be contracted with index 2 of coolist_ijk
<i>arr_k</i>	the vector to be contracted with index 3 of coolist_ijk
<i>res</i>	vector (buffer) to store the result of the contraction

Remarks

Note that it is NOT safe to pass `arr_j/arr_k` as a result buffer, as this operation does multiple passes.

Definition at line 99 of file `tensor.f90`.

```

99      TYPE(coolist), DIMENSION(ndim), INTENT(IN):: coolist_ijk
100     REAL(KIND=8), DIMENSION(0:ndim), INTENT(IN)  :: arr_j, arr_k
101     REAL(KIND=8), DIMENSION(0:ndim), INTENT(OUT) :: res
102     INTEGER :: i,j,k,n
103     res=0.d0
104     DO i=1,ndim
105         DO n=1,coolist_ijk(i)%elems
106             j=coolist_ijk(i)%elems(n)%j
107             k=coolist_ijk(i)%elems(n)%k
108             res(i) = res(i) + coolist_ijk(i)%elems(n)%v * arr_j(j)*arr_k(k)
109         END DO
110     END DO

```

6.10.3 Variable Documentation

6.10.3.1 `real(kind=8), parameter tensor::real_eps = 2.2204460492503131e-16`

Parameter to test the equality with zero.

Definition at line 33 of file `tensor.f90`.

```

33     REAL(KIND=8), PARAMETER :: real_eps = 2.2204460492503131e-16

```

6.11 tl_ad_tensor Module Reference

Tangent Linear (TL) and Adjoint (AD) model versions of MAOOAM. Tensors definition module.

Functions/Subroutines

- type([coolist](#)) function, dimension(ndim) [jacobian](#) (ystar)
Compute the Jacobian of MAOOAM in point ystar.
- real(kind=8) function, dimension(ndim, ndim) [jacobian_mat](#) (ystar)
Compute the Jacobian of MAOOAM in point ystar.
- subroutine, public [init_tltensor](#)
Routine to initialize the TL tensor.
- subroutine [compute_tltensor](#) (func)
Routine to compute the TL tensor from the original MAOOAM one.
- subroutine [tl_add_count](#) (i, j, k, v)
Subroutine used to count the number of TL tensor entries.
- subroutine [tl_coeff](#) (i, j, k, v)
Subroutine used to compute the TL tensor entries.
- subroutine, public [init_adtensor](#)
Routine to initialize the AD tensor.
- subroutine [compute_adtensor](#) (func)
Subroutine to compute the AD tensor from the original MAOOAM one.
- subroutine [ad_add_count](#) (i, j, k, v)

- Subroutine used to count the number of AD tensor entries.*
- subroutine [ad_coeff](#) (i, j, k, v)
- subroutine, public [init_adtensor_ref](#)
 - Alternate method to initialize the AD tensor from the TL tensor.*
- subroutine [compute_adtensor_ref](#) (func)
 - Alternate subroutine to compute the AD tensor from the TL one.*
- subroutine [ad_add_count_ref](#) (i, j, k, v)
 - Alternate subroutine used to count the number of AD tensor entries from the TL tensor.*
- subroutine [ad_coeff_ref](#) (i, j, k, v)
 - Alternate subroutine used to compute the AD tensor entries from the TL tensor.*
- subroutine, public [ad](#) (t, ystar, deltax, buf)
 - Tendencies for the AD of MAOOAM in point ystar for perturbation deltax.*
- subroutine, public [tl](#) (t, ystar, deltax, buf)
 - Tendencies for the TL of MAOOAM in point ystar for perturbation deltax.*

Variables

- real(kind=8), parameter [real_eps](#) = 2.2204460492503131e-16
 - Epsilon to test equality with 0.*
- integer, dimension(:), allocatable [count_elems](#)
 - Vector used to count the tensor elements.*
- type([coolist](#)), dimension(:), allocatable, public [tltensor](#)
 - Tensor representation of the Tangent Linear tendencies.*
- type([coolist](#)), dimension(:), allocatable, public [adtensor](#)
 - Tensor representation of the Adjoint tendencies.*

6.11.1 Detailed Description

Tangent Linear (TL) and Adjoint (AD) model versions of MAOOAM. Tensors definition module.

Copyright

2016 Lesley De Cruz & Jonathan Demaeyer. See [LICENSE.txt](#) for license information.

Remarks

The routines of this module should be called only after [params::init_params\(\)](#) and [adtensor_def::init_↔adtensor\(\)](#) have been called !

6.11.2 Function/Subroutine Documentation

- 6.11.2.1 subroutine, public [tl_ad_tensor::ad](#) (real(kind=8), intent(in) *t*, real(kind=8), dimension(0:ndim), intent(in) *ystar*, real(kind=8), dimension(0:ndim), intent(in) *deltax*, real(kind=8), dimension(0:ndim), intent(out) *buf*)

Tendencies for the AD of MAOOAM in point ystar for perturbation deltax.

Parameters

<i>t</i>	time
<i>ystar</i>	vector with the variables (current point in trajectory)
<i>deltay</i>	vector with the perturbation of the variables at time <i>t</i>
<i>buf</i>	vector (buffer) to store derivatives.

Definition at line 384 of file tl_ad_tensor.f90.

```

384     REAL(KIND=8), INTENT(IN) :: t
385     REAL(KIND=8), DIMENSION(0:ndim), INTENT(IN) :: ystar,deltay
386     REAL(KIND=8), DIMENSION(0:ndim), INTENT(OUT) :: buf
387     CALL sparse_mul3(adtensor,deltay,ystar,buf)

```

6.11.2.2 subroutine tl_ad_tensor::ad_add_count (integer, intent(in) *i*, integer, intent(in) *j*, integer, intent(in) *k*, real(kind=8), intent(in) *v*) [private]

Subroutine used to count the number of AD tensor entries.

Parameters

<i>i</i>	tensor <i>i</i> index
<i>j</i>	tensor <i>j</i> index
<i>k</i>	tensor <i>k</i> index
<i>v</i>	value that will be added

Definition at line 243 of file tl_ad_tensor.f90.

```

243     INTEGER, INTENT(IN) :: i,j,k
244     REAL(KIND=8), INTENT(IN) :: v
245     IF ((abs(v) .ge. real_eps).AND.(i /= 0)) THEN
246         IF (k /= 0) count_elems(k)=count_elems(k)+1
247         IF (j /= 0) count_elems(j)=count_elems(j)+1
248     ENDIF

```

6.11.2.3 subroutine tl_ad_tensor::ad_add_count_ref (integer, intent(in) *i*, integer, intent(in) *j*, integer, intent(in) *k*, real(kind=8), intent(in) *v*) [private]

Alternate subroutine used to count the number of AD tensor entries from the TL tensor.

Parameters

<i>i</i>	tensor <i>i</i> index
<i>j</i>	tensor <i>j</i> index
<i>k</i>	tensor <i>k</i> index
<i>v</i>	value that will be added

Definition at line 346 of file tl_ad_tensor.f90.

```

346     INTEGER, INTENT(IN) :: i,j,k

```

```

347     REAL(KIND=8), INTENT(IN) :: v
348     IF ((abs(v) .ge. real_eps).AND.(j /= 0)) count_elems(j)=count_elems(j)+1

```

6.11.2.4 subroutine tl_ad_tensor::ad_coeff (integer, intent(in) *i*, integer, intent(in) *j*, integer, intent(in) *k*, real(kind=8), intent(in) *v*) [private]

Parameters

<i>i</i>	tensor <i>i</i> index
<i>j</i>	tensor <i>j</i> index
<i>k</i>	tensor <i>k</i> index
<i>v</i>	value to add

Definition at line 257 of file tl_ad_tensor.f90.

```

257     INTEGER, INTENT(IN) :: i,j,k
258     REAL(KIND=8), INTENT(IN) :: v
259     INTEGER :: n
260     IF (.NOT. ALLOCATED(adtensor)) stop "*** ad_coeff routine : tensor not yet allocated ***"
261     IF ((abs(v) .ge. real_eps).AND.(i /=0)) THEN
262         IF (k /=0) THEN
263             IF (.NOT. ALLOCATED(adtensor(k)%elems)) stop "*** ad_coeff routine : tensor not yet allocated
264             ***"
265             n=(adtensor(k)%elems)+1
266             adtensor(k)%elems(n)%j=i
267             adtensor(k)%elems(n)%k=j
268             adtensor(k)%elems(n)%v=v
269             adtensor(k)%elems=n
270         END IF
271         IF (j /=0) THEN
272             IF (.NOT. ALLOCATED(adtensor(j)%elems)) stop "*** ad_coeff routine : tensor not yet allocated
273             ***"
274             n=(adtensor(j)%elems)+1
275             adtensor(j)%elems(n)%j=i
276             adtensor(j)%elems(n)%k=k
277             adtensor(j)%elems(n)%v=v
278             adtensor(j)%elems=n
279         END IF
280     END IF

```

6.11.2.5 subroutine tl_ad_tensor::ad_coeff_ref (integer, intent(in) *i*, integer, intent(in) *j*, integer, intent(in) *k*, real(kind=8), intent(in) *v*) [private]

Alternate subroutine used to compute the AD tensor entries from the TL tensor.

Parameters

<i>i</i>	tensor <i>i</i> index
<i>j</i>	tensor <i>j</i> index
<i>k</i>	tensor <i>k</i> index
<i>v</i>	value to add

Definition at line 358 of file tl_ad_tensor.f90.

```

358     INTEGER, INTENT(IN) :: i,j,k
359     REAL(KIND=8), INTENT(IN) :: v
360     INTEGER :: n

```

```

361     IF (.NOT. ALLOCATED(adtensor)) stop "*** ad_coeff_ref routine : tensor not yet allocated ***"
362     IF ((abs(v) .ge. real_eps).AND.(j /=0)) THEN
363         IF (.NOT. ALLOCATED(adtensor(j)%elems)) stop "*** ad_coeff_ref routine : tensor not yet allocated
***"
364         n=(adtensor(j)%elems)+1
365         adtensor(j)%elems(n)%j=i
366         adtensor(j)%elems(n)%k=k
367         adtensor(j)%elems(n)%v=v
368         adtensor(j)%elems=n
369     END IF

```

6.11.2.6 subroutine tl_ad_tensor::compute_adtensor (external func) [private]

Subroutine to compute the AD tensor from the original MAOOAM one.

Parameters

<i>func</i>	subroutine used to do the computation
-------------	---------------------------------------

Definition at line 217 of file tl_ad_tensor.f90.

6.11.2.7 subroutine tl_ad_tensor::compute_adtensor_ref (external func) [private]

Alternate subroutine to compute the AD tensor from the TL one.

Parameters

<i>func</i>	subroutine used to do the computation
-------------	---------------------------------------

Definition at line 318 of file tl_ad_tensor.f90.

6.11.2.8 subroutine tl_ad_tensor::compute_tltensor (external func) [private]

Routine to compute the TL tensor from the original MAOOAM one.

Parameters

<i>func</i>	subroutine used to do the computation
-------------	---------------------------------------

Definition at line 121 of file tl_ad_tensor.f90.

6.11.2.9 subroutine, public tl_ad_tensor::init_adtensor ()

Routine to initialize the AD tensor.

Definition at line 193 of file tl_ad_tensor.f90.

```

193     INTEGER :: i
194     INTEGER :: allocstat
195     ALLOCATE (adtensor(ndim),count_elems(ndim), stat=allocstat)
196     IF (allocstat /= 0) stop "*** Not enough memory ! ***"
197     count_elems=0
198     CALL compute_adtensor(ad_add_count)
199
200     DO i=1,ndim
201         ALLOCATE (adtensor(i)%elems(count_elems(i)), stat=allocstat)
202         IF (allocstat /= 0) stop "*** Not enough memory ! ***"
203     END DO
204
205     DEALLOCATE(count_elems, stat=allocstat)
206     IF (allocstat /= 0) stop "*** Deallocation problem ! ***"
207
208     CALL compute_adtensor(ad_coeff)
209
210     CALL simplify(adtensor)
211

```

6.11.2.10 subroutine, public tl_ad_tensor::init_adtensor_ref ()

Alternate method to initialize the AD tensor from the TL tensor.

Remarks

The [tltensor](#) must be initialised before using this method.

Definition at line 294 of file `tl_ad_tensor.f90`.

```

294     INTEGER :: i
295     INTEGER :: allocstat
296     ALLOCATE (adtensor(ndim),count_elems(ndim), stat=allocstat)
297     IF (allocstat /= 0) stop "*** Not enough memory ! ***"
298     count_elems=0
299     CALL compute_adtensor_ref(ad_add_count_ref)
300
301     DO i=1,ndim
302         ALLOCATE (adtensor(i)%elems(count_elems(i)), stat=allocstat)
303         IF (allocstat /= 0) stop "*** Not enough memory ! ***"
304     END DO
305
306     DEALLOCATE(count_elems, stat=allocstat)
307     IF (allocstat /= 0) stop "*** Deallocation problem ! ***"
308
309     CALL compute_adtensor_ref(ad_coeff_ref)
310
311     CALL simplify(adtensor)
312

```

6.11.2.11 subroutine, public tl_ad_tensor::init_tltensor ()

Routine to initialize the TL tensor.

Definition at line 97 of file `tl_ad_tensor.f90`.

```

97     INTEGER :: i
98     INTEGER :: allocstat
99     ALLOCATE (tltensor(ndim),count_elems(ndim), stat=allocstat)
100     IF (allocstat /= 0) stop "*** Not enough memory ! ***"
101     count_elems=0
102     CALL compute_tltensor(tl_add_count)
103
104     DO i=1,ndim
105         ALLOCATE (tltensor(i)%elems(count_elems(i)), stat=allocstat)
106         IF (allocstat /= 0) stop "*** Not enough memory ! ***"
107     END DO
108
109     DEALLOCATE(count_elems, stat=allocstat)
110     IF (allocstat /= 0) stop "*** Deallocation problem ! ***"
111
112     CALL compute_tltensor(tl_coeff)
113
114     CALL simplify(tltensor)
115

```

6.11.2.12 `type(coolist) function, dimension(ndim) tl_ad_tensor::jacobian (real(kind=8), dimension(0:ndim), intent(in) ystar) [private]`

Compute the Jacobian of MAOOAM in point ystar.

Parameters

<i>ystar</i>	array with variables in which the jacobian should be evaluated.
--------------	---

Returns

Jacobian in coolist-form (table of tuples {i,j,0,value})

Definition at line 75 of file `tl_ad_tensor.f90`.

```

75     REAL(KIND=8), DIMENSION(0:ndim), INTENT(IN) :: ystar
76     TYPE(coolist), DIMENSION(ndim) :: jacobian
77     CALL jsparse_mul(aotensor, ystar, jacobian)

```

6.11.2.13 `real(kind=8) function, dimension(ndim,ndim) tl_ad_tensor::jacobian_mat (real(kind=8), dimension(0:ndim), intent(in) ystar) [private]`

Compute the Jacobian of MAOOAM in point ystar.

Parameters

<i>ystar</i>	array with variables in which the jacobian should be evaluated.
--------------	---

Returns

Jacobian in matrix form

Definition at line 84 of file `tl_ad_tensor.f90`.

```

84     REAL(KIND=8), DIMENSION(0:ndim), INTENT(IN) :: ystar
85     REAL(KIND=8), DIMENSION(ndim,ndim) :: jacobian_mat
86     CALL jsparse_mul_mat(aotensor, ystar, jacobian_mat)

```

6.11.2.14 `subroutine, public tl_ad_tensor::tl (real(kind=8), intent(in) t, real(kind=8), dimension(0:ndim), intent(in) ystar, real(kind=8), dimension(0:ndim), intent(in) deltat, real(kind=8), dimension(0:ndim), intent(out) buf)`

Tendencies for the TL of MAOOAM in point ystar for perturbation deltat.

Parameters

<i>t</i>	time
<i>ystar</i>	vector with the variables (current point in trajectory)
<i>deltat</i>	vector with the perturbation of the variables at time t
<i>buf</i>	vector (buffer) to store derivatives.

Definition at line 396 of file `tl_ad_tensor.f90`.

```

396  REAL(KIND=8), INTENT(IN) :: t
397  REAL(KIND=8), DIMENSION(0:ndim), INTENT(IN) :: ystar,deltay
398  REAL(KIND=8), DIMENSION(0:ndim), INTENT(OUT) :: buf
399  CALL sparse_mul3(tltensor,deltay,ystar,buf)

```

6.11.2.15 subroutine `tl_ad_tensor::tl_add_count` (integer, intent(in) *i*, integer, intent(in) *j*, integer, intent(in) *k*, real(kind=8), intent(in) *v*) [private]

Subroutine used to count the number of TL tensor entries.

Parameters

<i>i</i>	tensor <i>i</i> index
<i>j</i>	tensor <i>j</i> index
<i>k</i>	tensor <i>k</i> index
<i>v</i>	value that will be added

Definition at line 147 of file `tl_ad_tensor.f90`.

```

147  INTEGER, INTENT(IN) :: i,j,k
148  REAL(KIND=8), INTENT(IN) :: v
149  IF (abs(v) .ge. real_eps) THEN
150      IF (j /= 0) count_elems(i)=count_elems(i)+1
151      IF (k /= 0) count_elems(i)=count_elems(i)+1
152  ENDIF

```

6.11.2.16 subroutine `tl_ad_tensor::tl_coeff` (integer, intent(in) *i*, integer, intent(in) *j*, integer, intent(in) *k*, real(kind=8), intent(in) *v*) [private]

Subroutine used to compute the TL tensor entries.

Parameters

<i>i</i>	tensor <i>i</i> index
<i>j</i>	tensor <i>j</i> index
<i>k</i>	tensor <i>k</i> index
<i>v</i>	value to add

Definition at line 161 of file `tl_ad_tensor.f90`.

```

161  INTEGER, INTENT(IN) :: i,j,k
162  REAL(KIND=8), INTENT(IN) :: v
163  INTEGER :: n
164  IF (.NOT. ALLOCATED(tltensor)) stop "*** tl_coeff routine : tensor not yet allocated ***"
165  IF (.NOT. ALLOCATED(tltensor(i)%elems)) stop "*** tl_coeff routine : tensor not yet allocated ***"
166  IF (abs(v) .ge. real_eps) THEN
167      IF (j /=0) THEN
168          n=(tltensor(i)%elems)+1
169          tltensor(i)%elems(n)%j=j
170          tltensor(i)%elems(n)%k=k
171          tltensor(i)%elems(n)%v=v
172          tltensor(i)%elems=n

```



```

173         END IF
174         IF (k /= 0) THEN
175             n=(tltensor(i)%elems)+1
176             tltensor(i)%elems(n)%j=k
177             tltensor(i)%elems(n)%k=j
178             tltensor(i)%elems(n)%v=v
179             tltensor(i)%elems=n
180         END IF
181     END IF

```

6.11.3 Variable Documentation

6.11.3.1 `type(coolist), dimension(:), allocatable, public tl_ad_tensor::adtensor`

Tensor representation of the Adjoint tendencies.

Definition at line 44 of file `tl_ad_tensor.f90`.

```

44  TYPE(coolist), DIMENSION(:), ALLOCATABLE, PUBLIC :: adtensor

```

6.11.3.2 `integer, dimension(:), allocatable tl_ad_tensor::count_elems` [private]

Vector used to count the tensor elements.

Definition at line 38 of file `tl_ad_tensor.f90`.

```

38  INTEGER, DIMENSION(:), ALLOCATABLE :: count_elems

```

6.11.3.3 `real(kind=8), parameter tl_ad_tensor::real_eps = 2.2204460492503131e-16` [private]

Epsilon to test equality with 0.

Definition at line 35 of file `tl_ad_tensor.f90`.

```

35  REAL(KIND=8), PARAMETER :: real_eps = 2.2204460492503131e-16

```

6.11.3.4 `type(coolist), dimension(:), allocatable, public tl_ad_tensor::tltensor`

Tensor representation of the Tangent Linear tendencies.

Definition at line 41 of file `tl_ad_tensor.f90`.

```

41  TYPE(coolist), DIMENSION(:), ALLOCATABLE, PUBLIC :: tltensor

```

6.12 util Module Reference

Utility module.

Functions/Subroutines

- `character(len=20)` function, public `str` (`k`)
Convert an integer to string.
- `character(len=40)` function, public `rstr` (`x`, `fm`)
Convert a real to string with a given format.
- subroutine, public `init_random_seed` ()
Random generator initialization routine.
- subroutine, public `init_one` (`A`)
Initialize a square matrix A as a unit matrix.

6.12.1 Detailed Description

Utility module.

Copyright

2015 Lesley De Cruz & Jonathan Demaeyer. See [LICENSE.txt](#) for license information.

6.12.2 Function/Subroutine Documentation

6.12.2.1 subroutine, public `util::init_one` (`real(kind=8)`, `dimension(:, :)`, `intent(inout)` `A`)

Initialize a square matrix A as a unit matrix.

Definition at line 99 of file `util.f90`.

```

99      REAL(KIND=8), DIMENSION(:, :), INTENT(INOUT) :: a
100      INTEGER :: i, n
101      n=size(a,1)
102      a=0.0d0
103      DO i=1,n
104          a(i,i)=1.0d0
105      END DO
106
```

6.12.2.2 subroutine, public `util::init_random_seed` ()

Random generator initialization routine.

Definition at line 44 of file `util.f90`.

6.12.2.3 `character(len=40)` function, public `util::rstr` (`real(kind=8)`, `intent(in)` `x`, `character(len=20)`, `intent(in)` `fm`)

Convert a real to string with a given format.

Definition at line 36 of file `util.f90`.

```

36      REAL(KIND=8), INTENT(IN) :: x
37      CHARACTER(len=20), INTENT(IN) :: fm
38      WRITE (rstr, trim(adjustl(fm))) x
39      rstr = adjustl(rstr)

```

6.12.2.4 `character(len=20)` function, public `util::str` (`integer`, `intent(in)` `k`)

Convert an integer to string.

Definition at line 29 of file `util.f90`.

```

29      INTEGER, INTENT(IN) :: k
30      WRITE (str, *) k
31      str = adjustl(str)

```

Chapter 7

Data Type Documentation

7.1 inprod_analytic::atm_tensors Type Reference

Type holding the atmospheric inner products tensors.

Private Attributes

- `real(kind=8), dimension(:, :), allocatable` [a](#)
- `real(kind=8), dimension(:, :), allocatable` [c](#)
- `real(kind=8), dimension(:, :), allocatable` [d](#)
- `real(kind=8), dimension(:, :), allocatable` [s](#)
- `real(kind=8), dimension(:, :, :), allocatable` [b](#)
- `real(kind=8), dimension(:, :, :), allocatable` [g](#)

7.1.1 Detailed Description

Type holding the atmospheric inner products tensors.

Definition at line 52 of file `inprod_analytic.f90`.

7.1.2 Member Data Documentation

7.1.2.1 `real(kind=8), dimension(:, :), allocatable inprod_analytic::atm_tensors::a` `[private]`

Definition at line 53 of file `inprod_analytic.f90`.

```
53      REAL(KIND=8), DIMENSION(:, :), ALLOCATABLE :: a, c, d, s
```

7.1.2.2 `real(kind=8), dimension(:, :, :), allocatable inprod_analytic::atm_tensors::b` `[private]`

Definition at line 54 of file `inprod_analytic.f90`.

```
54      REAL(KIND=8), DIMENSION(:, :, :), ALLOCATABLE :: b, g
```

7.1.2.3 `real(kind=8), dimension(:,,:), allocatable inprod_analytic::atm_tensors::c` [private]

Definition at line 53 of file inprod_analytic.f90.

7.1.2.4 `real(kind=8), dimension(:,,:), allocatable inprod_analytic::atm_tensors::d` [private]

Definition at line 53 of file inprod_analytic.f90.

7.1.2.5 `real(kind=8), dimension(:,,:), allocatable inprod_analytic::atm_tensors::g` [private]

Definition at line 54 of file inprod_analytic.f90.

7.1.2.6 `real(kind=8), dimension(:,,:), allocatable inprod_analytic::atm_tensors::s` [private]

Definition at line 53 of file inprod_analytic.f90.

The documentation for this type was generated from the following file:

- [inprod_analytic.f90](#)

7.2 inprod_analytic::atm_wavenum Type Reference

Atmospheric bloc specification type.

Private Attributes

- character `typ`
- integer `m` =0
- integer `p` =0
- integer `h` =0
- real(kind=8) `nx` =0.
- real(kind=8) `ny` =0.

7.2.1 Detailed Description

Atmospheric bloc specification type.

Definition at line 39 of file inprod_analytic.f90.

7.2.2 Member Data Documentation

7.2.2.1 `integer inprod_analytic::atm_wavenum::h =0` [private]

Definition at line 41 of file inprod_analytic.f90.

7.2.2.2 integer inprod_analytic::atm_wavenum::m =0 [private]

Definition at line 41 of file inprod_analytic.f90.

```
41      INTEGER :: m=0,p=0,h=0
```

7.2.2.3 real(kind=8) inprod_analytic::atm_wavenum::nx =0. [private]

Definition at line 42 of file inprod_analytic.f90.

```
42      REAL(KIND=8) :: nx=0.,ny=0.
```

7.2.2.4 real(kind=8) inprod_analytic::atm_wavenum::ny =0. [private]

Definition at line 42 of file inprod_analytic.f90.

7.2.2.5 integer inprod_analytic::atm_wavenum::p =0 [private]

Definition at line 41 of file inprod_analytic.f90.

7.2.2.6 character inprod_analytic::atm_wavenum::typ [private]

Definition at line 40 of file inprod_analytic.f90.

```
40      CHARACTER :: typ
```

The documentation for this type was generated from the following file:

- [inprod_analytic.f90](#)

7.3 tensor::coolist Type Reference

Coordinate list. Type used to represent the sparse tensor.

Public Attributes

- type([coolist_elem](#)), dimension(:), allocatable [elems](#)
Lists of elements [tensor::coolist_elem](#).
- integer [nelems](#) = 0
Number of elements in the list.

7.3.1 Detailed Description

Coordinate list. Type used to represent the sparse tensor.

Definition at line 27 of file tensor.f90.

7.3.2 Member Data Documentation

7.3.2.1 `type(coolist_elem), dimension(:), allocatable tensor::coolist::elems`

Lists of elements [tensor::coolist_elem](#).

Definition at line 28 of file tensor.f90.

```
28      TYPE(coolist_elem), DIMENSION(:), ALLOCATABLE :: elems !< Lists of elements tensor::coolist_elem
```

7.3.2.2 `integer tensor::coolist::nelems = 0`

Number of elements in the list.

Definition at line 29 of file tensor.f90.

```
29      INTEGER :: nelems = 0 !< Number of elements in the list.
```

The documentation for this type was generated from the following file:

- [tensor.f90](#)

7.4 `tensor::coolist_elem` Type Reference

Coordinate list element type. Elementary elements of the sparse tensors.

Private Attributes

- integer [j](#)
Index j of the element.
- integer [k](#)
Index k of the element.
- real(kind=8) [v](#)
Value of the element.

7.4.1 Detailed Description

Coordinate list element type. Elementary elements of the sparse tensors.

Definition at line 20 of file tensor.f90.

7.4.2 Member Data Documentation

7.4.2.1 integer tensor::coolist_elem::j [private]

Index j of the element.

Definition at line 21 of file tensor.f90.

```
21      INTEGER :: j !< Index \f$j\f$ of the element
```

7.4.2.2 integer tensor::coolist_elem::k [private]

Index k of the element.

Definition at line 22 of file tensor.f90.

```
22      INTEGER :: k !< Index \f$k\f$ of the element
```

7.4.2.3 real(kind=8) tensor::coolist_elem::v [private]

Value of the element.

Definition at line 23 of file tensor.f90.

```
23      REAL(KIND=8) :: v !< Value of the element
```

The documentation for this type was generated from the following file:

- [tensor.f90](#)

7.5 inprod_analytic::ocean_tensors Type Reference

Type holding the oceanic inner products tensors.

Private Attributes

- `real(kind=8), dimension(:, :), allocatable` [k](#)
- `real(kind=8), dimension(:, :), allocatable` [m](#)
- `real(kind=8), dimension(:, :), allocatable` [n](#)
- `real(kind=8), dimension(:, :), allocatable` [w](#)
- `real(kind=8), dimension(:, :, :), allocatable` [o](#)
- `real(kind=8), dimension(:, :, :), allocatable` [c](#)

7.5.1 Detailed Description

Type holding the oceanic inner products tensors.

Definition at line 58 of file `inprod_analytic.f90`.

7.5.2 Member Data Documentation

7.5.2.1 `real(kind=8), dimension(:, :, :), allocatable inprod_analytic::ocean_tensors::c` [\[private\]](#)

Definition at line 60 of file `inprod_analytic.f90`.

7.5.2.2 `real(kind=8), dimension(:, :), allocatable inprod_analytic::ocean_tensors::k` [\[private\]](#)

Definition at line 59 of file `inprod_analytic.f90`.

```
59      REAL(KIND=8), DIMENSION(:, :), ALLOCATABLE :: k, m, n, w
```

7.5.2.3 `real(kind=8), dimension(:, :), allocatable inprod_analytic::ocean_tensors::m` [\[private\]](#)

Definition at line 59 of file `inprod_analytic.f90`.

7.5.2.4 `real(kind=8), dimension(:, :), allocatable inprod_analytic::ocean_tensors::n` [\[private\]](#)

Definition at line 59 of file `inprod_analytic.f90`.

7.5.2.5 `real(kind=8), dimension(:, :, :), allocatable inprod_analytic::ocean_tensors::o` [\[private\]](#)

Definition at line 60 of file `inprod_analytic.f90`.

```
60      REAL(KIND=8), DIMENSION(:, :, :), ALLOCATABLE :: o, c
```


7.5.2.6 `real(kind=8), dimension(:,,:), allocatable inprod_analytic::ocean_tensors::w` [private]

Definition at line 59 of file inprod_analytic.f90.

The documentation for this type was generated from the following file:

- [inprod_analytic.f90](#)

7.6 inprod_analytic::ocean_wavenum Type Reference

Oceanic bloc specification type.

Private Attributes

- integer `p`
- integer `h`
- `real(kind=8)` `nx`
- `real(kind=8)` `ny`

7.6.1 Detailed Description

Oceanic bloc specification type.

Definition at line 46 of file inprod_analytic.f90.

7.6.2 Member Data Documentation

7.6.2.1 `integer inprod_analytic::ocean_wavenum::h` [private]

Definition at line 47 of file inprod_analytic.f90.

7.6.2.2 `real(kind=8) inprod_analytic::ocean_wavenum::nx` [private]

Definition at line 48 of file inprod_analytic.f90.

```
48      REAL (KIND=8) :: nx, ny
```

7.6.2.3 `real(kind=8) inprod_analytic::ocean_wavenum::ny` [private]

Definition at line 48 of file inprod_analytic.f90.

7.6.2.4 `integer inprod_analytic::ocean_wavenum::p` [private]

Definition at line 47 of file inprod_analytic.f90.

```
47      INTEGER :: p, h
```

The documentation for this type was generated from the following file:

- [inprod_analytic.f90](#)

Chapter 8

File Documentation

8.1 aotensor_def.f90 File Reference

Modules

- module [aotensor_def](#)

The equation tensor for the coupled ocean-atmosphere model with temperature which allows for an extensible set of modes in the ocean and in the atmosphere.

Functions/Subroutines

- integer function [aotensor_def::psi](#) (i)
Translate the $\psi_{a,i}$ coefficients into effective coordinates.
- integer function [aotensor_def::theta](#) (i)
Translate the $\theta_{a,i}$ coefficients into effective coordinates.
- integer function [aotensor_def::a](#) (i)
Translate the $\psi_{o,i}$ coefficients into effective coordinates.
- integer function [aotensor_def::t](#) (i)
Translate the $\delta T_{o,i}$ coefficients into effective coordinates.
- integer function [aotensor_def::kdelta](#) (i, j)
Kronecker delta function.
- subroutine [aotensor_def::coeff](#) (i, j, k, v)
Subroutine to add element in the [aotensor](#) $\mathcal{T}_{i,j,k}$ structure.
- subroutine [aotensor_def::add_count](#) (i, j, k, v)
Subroutine to count the elements of the [aotensor](#) $\mathcal{T}_{i,j,k}$. Add +1 to `count_elems(i)` for each value that is added to the tensor i -th component.
- subroutine [aotensor_def::compute_aotensor](#) (func)
Subroutine to compute the tensor [aotensor](#).
- subroutine, public [aotensor_def::init_aotensor](#)
Subroutine to initialise the [aotensor](#) tensor.

Variables

- integer, dimension(:), allocatable [aotensor_def::count_elems](#)
Vector used to count the tensor elements.
- real(kind=8), parameter [aotensor_def::real_eps](#) = 2.2204460492503131e-16
Epsilon to test equality with 0.
- type(coolist), dimension(:), allocatable, public [aotensor_def::aotensor](#)
 $\mathcal{T}_{i,j,k}$ - Tensor representation of the tendencies.

8.2 doc/gen_doc.md File Reference

8.3 doc/tl_ad_doc.md File Reference

8.4 ic_def.f90 File Reference

Modules

- module [ic_def](#)
Module to load the initial condition.

Functions/Subroutines

- subroutine, public [ic_def::load_ic](#)
Subroutine to load the initial condition if IC.nml exists. If it does not, then write IC.nml with 0 as initial condition.

Variables

- logical [ic_def::exists](#)
Boolean to test for file existence.
- real(kind=8), dimension(:), allocatable, public [ic_def::ic](#)
Initial condition vector.

8.5 inprod_analytic.f90 File Reference

Data Types

- type [inprod_analytic::atm_wavenum](#)
Atmospheric bloc specification type.
- type [inprod_analytic::ocean_wavenum](#)
Oceanic bloc specification type.
- type [inprod_analytic::atm_tensors](#)
Type holding the atmospheric inner products tensors.
- type [inprod_analytic::ocean_tensors](#)
Type holding the oceanic inner products tensors.

Modules

- module [inprod_analytic](#)

Inner products between the truncated set of basis functions for the ocean and atmosphere streamfunction fields. These are partly calculated using the analytical expressions from Cehelsky, P., & Tung, K. K. : Theories of multiple equilibria and weather regimes-A critical reexamination. Part II: Baroclinic two-layer models. Journal of the atmospheric sciences, 44(21), 3282-3303, 1987.

Functions/Subroutines

- real(kind=8) function [inprod_analytic::b1](#) (Pi, Pj, Pk)
Cehelsky & Tung Helper functions.
- real(kind=8) function [inprod_analytic::b2](#) (Pi, Pj, Pk)
Cehelsky & Tung Helper functions.
- real(kind=8) function [inprod_analytic::delta](#) (r)
Integer Dirac delta function.
- real(kind=8) function [inprod_analytic::flambda](#) (r)
"Odd or even" function
- real(kind=8) function [inprod_analytic::s1](#) (Pj, Pk, Mj, Hk)
Cehelsky & Tung Helper functions.
- real(kind=8) function [inprod_analytic::s2](#) (Pj, Pk, Mj, Hk)
Cehelsky & Tung Helper functions.
- real(kind=8) function [inprod_analytic::s3](#) (Pj, Pk, Hj, Hk)
Cehelsky & Tung Helper functions.
- real(kind=8) function [inprod_analytic::s4](#) (Pj, Pk, Hj, Hk)
Cehelsky & Tung Helper functions.
- subroutine [inprod_analytic::calculate_a](#)
Eigenvalues of the Laplacian (atmospheric)
- subroutine [inprod_analytic::calculate_b](#)
Streamfunction advection terms (atmospheric)
- subroutine [inprod_analytic::calculate_c_atm](#)
Beta term for the atmosphere.
- subroutine [inprod_analytic::calculate_d](#)
Forcing of the ocean on the atmosphere.
- subroutine [inprod_analytic::calculate_g](#)
Temperature advection terms (atmospheric)
- subroutine [inprod_analytic::calculate_s](#)
Forcing (thermal) of the ocean on the atmosphere.
- subroutine [inprod_analytic::calculate_k](#)
Forcing of the atmosphere on the ocean.
- subroutine [inprod_analytic::calculate_m](#)
Forcing of the ocean fields on the ocean.
- subroutine [inprod_analytic::calculate_n](#)
Beta term for the ocean.
- subroutine [inprod_analytic::calculate_o](#)
Temperature advection term (passive scalar)
- subroutine [inprod_analytic::calculate_c_oc](#)
Streamfunction advection terms (oceanic)
- subroutine [inprod_analytic::calculate_w](#)
Short-wave radiative forcing of the ocean.
- subroutine, public [inprod_analytic::init_inprod](#)
Initialisation of the inner product.
- subroutine, public [inprod_analytic::deallocate_inprod](#)
Deallocation of the inner products.

Variables

- type(atm_wavenum), dimension(:), allocatable, public [inprod_analytic::awavenum](#)
Atmospheric blocs specification.
- type(ocean_wavenum), dimension(:), allocatable, public [inprod_analytic::owavenum](#)
Oceanic blocs specification.
- type(atm_tensors), public [inprod_analytic::atmos](#)
Atmospheric tensors.
- type(ocean_tensors), public [inprod_analytic::ocean](#)
Oceanic tensors.

8.6 LICENSE.txt File Reference

Functions

- The MIT [License](#) (MIT) Copyright(c) 2015-2016 Lesley De Cruz and Jonathan Demaeyer Permission is hereby granted
- The MIT free of to any person obtaining a [copy](#) of this software and associated documentation [files](#) (the"Software")

Variables

- The MIT free of [charge](#)
- The MIT free of to any person obtaining a [copy](#) of this software and associated documentation to deal in the [Software](#) without [restriction](#)
- The MIT free of to any person obtaining a [copy](#) of this software and associated documentation to deal in the [Software](#) without including without limitation the rights to [use](#)
- The MIT free of to any person obtaining a copy of this software and associated documentation to deal in the [Software](#) without including without limitation the rights to [copy](#)
- The MIT free of to any person obtaining a [copy](#) of this software and associated documentation to deal in the [Software](#) without including without limitation the rights to [modify](#)
- The MIT free of to any person obtaining a [copy](#) of this software and associated documentation to deal in the [Software](#) without including without limitation the rights to [merge](#)
- The MIT free of to any person obtaining a [copy](#) of this software and associated documentation to deal in the [Software](#) without including without limitation the rights to [publish](#)
- The MIT free of to any person obtaining a [copy](#) of this software and associated documentation to deal in the [Software](#) without including without limitation the rights to [distribute](#)
- The MIT free of to any person obtaining a [copy](#) of this software and associated documentation to deal in the [Software](#) without including without limitation the rights to [sublicense](#)
- The MIT free of to any person obtaining a [copy](#) of this software and associated documentation to deal in the Software without including without limitation the rights to and or sell copies of the [Software](#)
- The MIT free of to any person obtaining a [copy](#) of this software and associated documentation to deal in the [Software](#) without including without limitation the rights to and or sell copies of the and to permit persons to whom the [Software](#) is furnished to do [so](#)
- The MIT free of to any person obtaining a [copy](#) of this software and associated documentation to deal in the [Software](#) without including without limitation the rights to and or sell copies of the and to permit persons to whom the [Software](#) is furnished to do subject to the following [conditions](#)
- The MIT free of to any person obtaining a [copy](#) of this software and associated documentation to deal in the [Software](#) without including without limitation the rights to and or sell copies of the and to permit persons to whom the [Software](#) is furnished to do subject to the following WITHOUT WARRANTY OF ANY [KIND](#)

- The MIT free of to any person obtaining a [copy](#) of this software and associated documentation to deal in the [Software](#) without including without limitation the rights to and or sell copies of the and to permit persons to whom the [Software](#) is furnished to do subject to the following WITHOUT WARRANTY OF ANY EXPRESS OR [IMPLIED](#)
- The MIT free of to any person obtaining a [copy](#) of this software and associated documentation to deal in the [Software](#) without including without limitation the rights to and or sell copies of the and to permit persons to whom the [Software](#) is furnished to do subject to the following WITHOUT WARRANTY OF ANY EXPRESS OR INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF [MERCHANTABILITY](#)
- The MIT free of to any person obtaining a [copy](#) of this software and associated documentation to deal in the [Software](#) without including without limitation the rights to and or sell copies of the and to permit persons to whom the [Software](#) is furnished to do subject to the following WITHOUT WARRANTY OF ANY EXPRESS OR INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY [CLAIM](#)
- The MIT free of to any person obtaining a [copy](#) of this software and associated documentation to deal in the [Software](#) without including without limitation the rights to and or sell copies of the and to permit persons to whom the [Software](#) is furnished to do subject to the following WITHOUT WARRANTY OF ANY EXPRESS OR INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY DAMAGES OR OTHER [LIABILITY](#)
- The MIT free of to any person obtaining a [copy](#) of this software and associated documentation to deal in the [Software](#) without including without limitation the rights to and or sell copies of the and to permit persons to whom the [Software](#) is furnished to do subject to the following WITHOUT WARRANTY OF ANY EXPRESS OR INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY DAMAGES OR OTHER WHETHER IN AN ACTION OF [CONTRACT](#)
- The MIT free of to any person obtaining a [copy](#) of this software and associated documentation to deal in the [Software](#) without including without limitation the rights to and or sell copies of the and to permit persons to whom the [Software](#) is furnished to do subject to the following WITHOUT WARRANTY OF ANY EXPRESS OR INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY DAMAGES OR OTHER WHETHER IN AN ACTION OF TORT OR [OTHERWISE](#)
- The MIT free of to any person obtaining a [copy](#) of this software and associated documentation to deal in the [Software](#) without including without limitation the rights to and or sell copies of the and to permit persons to whom the [Software](#) is furnished to do subject to the following WITHOUT WARRANTY OF ANY EXPRESS OR INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY DAMAGES OR OTHER WHETHER IN AN ACTION OF TORT OR ARISING [FROM](#)

8.6.1 Function Documentation

8.6.1.1 The MIT free of to any person obtaining a [copy](#) of this software and associated documentation files (the"Software")

8.6.1.2 The MIT License (MIT)

8.6.2 Variable Documentation

8.6.2.1 The MIT free of charge

Definition at line 6 of file LICENSE.txt.

8.6.2.2 The MIT free of to any person obtaining a copy of this software and associated documentation to deal in the Software without including without limitation the rights to and or sell copies of the and to permit persons to whom the Software is furnished to do subject to the following WITHOUT WARRANTY OF ANY EXPRESS OR INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM

Definition at line 8 of file LICENSE.txt.

8.6.2.3 The MIT free of to any person obtaining a copy of this software and associated documentation to deal in the Software without including without limitation the rights to and or sell copies of the and to permit persons to whom the Software is furnished to do subject to the following conditions

Definition at line 8 of file LICENSE.txt.

8.6.2.4 The MIT free of to any person obtaining a copy of this software and associated documentation to deal in the Software without including without limitation the rights to and or sell copies of the and to permit persons to whom the Software is furnished to do subject to the following WITHOUT WARRANTY OF ANY EXPRESS OR INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY DAMAGES OR OTHER WHETHER IN AN ACTION OF CONTRACT

Definition at line 8 of file LICENSE.txt.

8.6.2.5 The MIT free of to any person obtaining a copy of this software and associated documentation to deal in the Software without including without limitation the rights to copy

Definition at line 8 of file LICENSE.txt.

8.6.2.6 The MIT free of to any person obtaining a copy of this software and associated documentation to deal in the Software without including without limitation the rights to distribute

Definition at line 8 of file LICENSE.txt.

8.6.2.7 The MIT free of to any person obtaining a copy of this software and associated documentation to deal in the Software without including without limitation the rights to and or sell copies of the and to permit persons to whom the Software is furnished to do subject to the following WITHOUT WARRANTY OF ANY EXPRESS OR INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY DAMAGES OR OTHER WHETHER IN AN ACTION OF TORT OR ARISING FROM

Definition at line 8 of file LICENSE.txt.

8.6.2.8 The MIT free of to any person obtaining a copy of this software and associated documentation to deal in the Software without including without limitation the rights to and or sell copies of the and to permit persons to whom the Software is furnished to do subject to the following WITHOUT WARRANTY OF ANY EXPRESS OR IMPLIED

Definition at line 8 of file LICENSE.txt.

8.6.2.9 The MIT free of to any person obtaining a copy of this software and associated documentation to deal in the Software without including without limitation the rights to and or sell copies of the and to permit persons to whom the Software is furnished to do subject to the following WITHOUT WARRANTY OF ANY KIND

Definition at line 8 of file LICENSE.txt.

8.6.2.10 The MIT free of to any person obtaining a copy of this software and associated documentation to deal in the Software without including without limitation the rights to and or sell copies of the and to permit persons to whom the Software is furnished to do subject to the following WITHOUT WARRANTY OF ANY EXPRESS OR INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY DAMAGES OR OTHER LIABILITY

Definition at line 8 of file LICENSE.txt.

8.6.2.11 The MIT free of to any person obtaining a copy of this software and associated documentation to deal in the Software without including without limitation the rights to and or sell copies of the and to permit persons to whom the Software is furnished to do subject to the following WITHOUT WARRANTY OF ANY EXPRESS OR INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY

Definition at line 8 of file LICENSE.txt.

8.6.2.12 The MIT free of to any person obtaining a copy of this software and associated documentation to deal in the Software without including without limitation the rights to merge

Definition at line 8 of file LICENSE.txt.

8.6.2.13 The MIT free of to any person obtaining a copy of this software and associated documentation to deal in the Software without including without limitation the rights to modify

Definition at line 8 of file LICENSE.txt.

8.6.2.14 The MIT free of to any person obtaining a copy of this software and associated documentation to deal in the Software without including without limitation the rights to and or sell copies of the and to permit persons to whom the Software is furnished to do subject to the following WITHOUT WARRANTY OF ANY EXPRESS OR INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY DAMAGES OR OTHER WHETHER IN AN ACTION OF TORT OR OTHERWISE

Definition at line 8 of file LICENSE.txt.

8.6.2.15 The MIT free of to any person obtaining a copy of this software and associated documentation to deal in the Software without including without limitation the rights to publish

Definition at line 8 of file LICENSE.txt.

8.6.2.16 The MIT free of to any person obtaining a copy of this software and associated documentation to deal in the Software without restriction

Definition at line 8 of file LICENSE.txt.

8.6.2.17 The MIT free of to any person obtaining a copy of this software and associated documentation to deal in the Software without including without limitation the rights to and or sell copies of the and to permit persons to whom the Software is furnished to do so

Definition at line 8 of file LICENSE.txt.

8.6.2.18 The MIT free of to any person obtaining a copy of this software and associated documentation to deal in the Software without including without limitation the rights to and or sell copies of the Software

Definition at line 8 of file LICENSE.txt.

8.6.2.19 The MIT free of to any person obtaining a copy of this software and associated documentation to deal in the Software without including without limitation the rights to sublicense

Definition at line 8 of file LICENSE.txt.

8.6.2.20 The MIT free of to any person obtaining a copy of this software and associated documentation to deal in the Software without including without limitation the rights to use

Definition at line 8 of file LICENSE.txt.

8.7 maoam.f90 File Reference

Functions/Subroutines

- program [maoam](#)
Fortran 90 implementation of the modular arbitrary-order ocean-atmosphere model MAOOAM.

8.7.1 Function/Subroutine Documentation

8.7.1.1 program maoam ()

Fortran 90 implementation of the modular arbitrary-order ocean-atmosphere model MAOOAM.

Copyright

2015 Lesley De Cruz & Jonathan Demaeyer. See [LICENSE.txt](#) for license information.

Definition at line 13 of file maoam.f90.

8.8 params.f90 File Reference

Modules

- module [params](#)
The model parameters module.

Functions/Subroutines

- subroutine, private [params::init_nml](#)
Read the basic parameters and mode selection from the namelist.
- subroutine [params::init_params](#)
Parameters initialisation routine.

Variables

- real(kind=8) [params::n](#)
 $n = 2L_y / L_x$ - Aspect ratio
- real(kind=8) [params::phi0](#)
Latitude in radian.
- real(kind=8) [params::rra](#)
Earth radius.
- real(kind=8) [params::sig0](#)
 σ_0 - Non-dimensional static stability of the atmosphere.
- real(kind=8) [params::k](#)
Bottom atmospheric friction coefficient.
- real(kind=8) [params::kp](#)
 k' - Internal atmospheric friction coefficient.
- real(kind=8) [params::r](#)
Frictional coefficient at the bottom of the ocean.
- real(kind=8) [params::d](#)
Mechanical coupling parameter between the ocean and the atmosphere.
- real(kind=8) [params::f0](#)
 f_0 - Coriolis parameter
- real(kind=8) [params::gp](#)
 g' Reduced gravity
- real(kind=8) [params::h](#)
Depth of the active water layer of the ocean.
- real(kind=8) [params::phi0_npi](#)
Latitude exprimed in fraction of pi.
- real(kind=8) [params::lambda](#)
 λ - Sensible + turbulent heat exchange between the ocean and the atmosphere.
- real(kind=8) [params::co](#)
 C_a - Constant short-wave radiation of the ocean.
- real(kind=8) [params::go](#)
 γ_o - Specific heat capacity of the ocean.
- real(kind=8) [params::ca](#)
 C_a - Constant short-wave radiation of the atmosphere.
- real(kind=8) [params::to0](#)

- T_o^0 - Stationary solution for the 0-th order ocean temperature.
 - real(kind=8) [params::ta0](#)
- T_a^0 - Stationary solution for the 0-th order atmospheric temperature.
 - real(kind=8) [params::epsa](#)
- ϵ_a - Emissivity coefficient for the grey-body atmosphere.
 - real(kind=8) [params::ga](#)
- γ_a - Specific heat capacity of the atmosphere.
 - real(kind=8) [params::rr](#)
- R - Gas constant of dry air
 - real(kind=8) [params::scale](#)
- $L_y = L \pi$ - The characteristic space scale.
 - real(kind=8) [params::pi](#)
- π
 - real(kind=8) [params::lr](#)
- L_R - Rossby deformation radius
 - real(kind=8) [params::g](#)
- γ
 - real(kind=8) [params::rp](#)
- r' - Frictional coefficient at the bottom of the ocean.
 - real(kind=8) [params::dp](#)
- d' - Non-dimensional mechanical coupling parameter between the ocean and the atmosphere.
 - real(kind=8) [params::kd](#)
- k_d - Non-dimensional bottom atmospheric friction coefficient.
 - real(kind=8) [params::kdp](#)
- k'_d - Non-dimensional internal atmospheric friction coefficient.
 - real(kind=8) [params::cpo](#)
- C'_a - Non-dimensional constant short-wave radiation of the ocean.
 - real(kind=8) [params::lpo](#)
- λ'_o - Non-dimensional sensible + turbulent heat exchange from ocean to atmosphere.
 - real(kind=8) [params::cpa](#)
- C'_a - Non-dimensional constant short-wave radiation of the atmosphere.
 - real(kind=8) [params::lpa](#)
- λ'_a - Non-dimensional sensible + turbulent heat exchange from atmosphere to ocean.
 - real(kind=8) [params::sbpo](#)
- $\sigma'_{B,o}$ - Long wave radiation lost by ocean to atmosphere & space.
 - real(kind=8) [params::sbpa](#)
- $\sigma'_{B,a}$ - Long wave radiation from atmosphere absorbed by ocean.
 - real(kind=8) [params::lsbpo](#)
- $S'_{B,o}$ - Long wave radiation from ocean absorbed by atmosphere.
 - real(kind=8) [params::lsbpa](#)
- $S'_{B,a}$ - Long wave radiation lost by atmosphere to space & ocean.
 - real(kind=8) [params::l](#)
- L - Domain length scale
 - real(kind=8) [params::sc](#)
- Ratio of surface to atmosphere temperature.
 - real(kind=8) [params::sb](#)
- Stefan–Boltzmann constant.
 - real(kind=8) [params::betp](#)
 - real(kind=8) [params::t_trans](#)
- Transient time period.
 - real(kind=8) [params::t_run](#)

- Effective intergration time (length of the generated trajectory)*

 - real(kind=8) `params::dt`
Integration time step.
 - real(kind=8) `params::tw`
Write all variables every tw time units.
 - logical `params::writeout`
Write to file boolean.
 - integer `params::nboc`
Number of atmospheric blocks.
 - integer `params::nbatm`
Number of oceanic blocks.
 - integer `params::natm` =0
Number of atmospheric basis functions.
 - integer `params::noc` =0
Number of oceanic basis functions.
 - integer `params::ndim`
Number of variables (dimension of the model)
 - integer, dimension(:,:), allocatable `params::oms`
Ocean mode selection array.
 - integer, dimension(:,:), allocatable `params::ams`
Atmospheric mode selection array.

8.9 rk2_integrator.f90 File Reference

Modules

- module `rk2_integrator`
Module with the integration routines.

Functions/Subroutines

- subroutine, public `rk2_integrator::init_integrator`
Routine to initialise the integration buffers.
- subroutine `rk2_integrator::tendencies` (t, y, res)
Routine computing the tendencies of the model.
- subroutine, public `rk2_integrator::step` (y, t, dt, res)
Routine to perform an integration step (Heun algorithm). The incremented time is returned.

Variables

- real(kind=8), dimension(:), allocatable `rk2_integrator::buf_y1`
Buffer to hold the intermediate position (Heun algorithm)
- real(kind=8), dimension(:), allocatable `rk2_integrator::buf_f0`
Buffer to hold tendencies at the initial position.
- real(kind=8), dimension(:), allocatable `rk2_integrator::buf_f1`
Buffer to hold tendencies at the intermediate position.

8.10 rk2_tl_ad_integrator.f90 File Reference

Modules

- module [rk2_tl_ad_integrator](#)
Tangent Linear (TL) and Adjoint (AD) model versions of MAOOAM. Integrators module.

Functions/Subroutines

- subroutine, public [rk2_tl_ad_integrator::init_tl_ad_integrator](#)
Routine to initialise the integration buffers.
- subroutine, public [rk2_tl_ad_integrator::ad_step](#) (y, ystar, t, dt, res)
Routine to perform an integration step (Heun algorithm) of the adjoint model. The incremented time is returned.
- subroutine, public [rk2_tl_ad_integrator::tl_step](#) (y, ystar, t, dt, res)
Routine to perform an integration step (Heun algorithm) of the tangent linear model. The incremented time is returned.

Variables

- real(kind=8), dimension(:), allocatable [rk2_tl_ad_integrator::buf_y1](#)
Buffer to hold the intermediate position (Heun algorithm) of the tangent linear model.
- real(kind=8), dimension(:), allocatable [rk2_tl_ad_integrator::buf_f0](#)
Buffer to hold tendencies at the initial position of the tangent linear model.
- real(kind=8), dimension(:), allocatable [rk2_tl_ad_integrator::buf_f1](#)
Buffer to hold tendencies at the intermediate position of the tangent linear model.

8.11 rk4_integrator.f90 File Reference

Modules

- module [rk4_integrator](#)
Module with the RK4 integration routines.

Functions/Subroutines

- subroutine, public [rk4_integrator::init_integrator](#)
Routine to initialise the integration buffers.
- subroutine [rk4_integrator::tendencies](#) (t, y, res)
Routine computing the tendencies of the model.
- subroutine, public [rk4_integrator::step](#) (y, t, dt, res)
Routine to perform an integration step (RK4 algorithm). The incremented time is returned.

Variables

- real(kind=8), dimension(:), allocatable [rk4_integrator::buf_y1](#)
Buffer to hold the intermediate position (Heun algorithm)
- real(kind=8), dimension(:), allocatable [rk4_integrator::buf_ka](#)
Buffer A to hold tendencies.
- real(kind=8), dimension(:), allocatable [rk4_integrator::buf_kb](#)
Buffer B to hold tendencies.

8.12 rk4_tl_ad_integrator.f90 File Reference

Modules

- module [rk4_tl_ad_integrator](#)

Tangent Linear (TL) and Adjoint (AD) model versions of MAOOAM. Integrators module.

Functions/Subroutines

- subroutine, public [rk4_tl_ad_integrator::init_tl_ad_integrator](#)
Routine to initialise the TL-AD integration buffers.
- subroutine, public [rk4_tl_ad_integrator::ad_step](#) (y, ystar, t, dt, res)
Routine to perform an integration step (RK4 algorithm) of the adjoint model. The incremented time is returned.
- subroutine, public [rk4_tl_ad_integrator::tl_step](#) (y, ystar, t, dt, res)
Routine to perform an integration step (RK4 algorithm) of the tangent linear model. The incremented time is returned.

Variables

- real(kind=8), dimension(:), allocatable [rk4_tl_ad_integrator::buf_y1](#)
Buffer to hold the intermediate position of the tangent linear model.
- real(kind=8), dimension(:), allocatable [rk4_tl_ad_integrator::buf_ka](#)
Buffer to hold tendencies in the RK4 scheme for the tangent linear model.
- real(kind=8), dimension(:), allocatable [rk4_tl_ad_integrator::buf_kb](#)
Buffer to hold tendencies in the RK4 scheme for the tangent linear model.

8.13 stat.f90 File Reference

Modules

- module [stat](#)

Statistics accumulators.

Functions/Subroutines

- subroutine, public [stat::init_stat](#)
Initialise the accumulators.
- subroutine, public [stat::acc](#) (x)
Accumulate one state.
- real(kind=8) function, dimension(0:ndim), public [stat::mean](#) ()
Function returning the mean.
- real(kind=8) function, dimension(0:ndim), public [stat::var](#) ()
Function returning the variance.
- integer function, public [stat::iter](#) ()
Function returning the number of data accumulated.
- subroutine, public [stat::reset](#)
Routine resetting the accumulators.

Variables

- integer `stat::i` =0
Number of stats accumulated.
- real(kind=8), dimension(:), allocatable `stat::m`
Vector storing the inline mean.
- real(kind=8), dimension(:), allocatable `stat::mprev`
Previous mean vector.
- real(kind=8), dimension(:), allocatable `stat::v`
Vector storing the inline variance.
- real(kind=8), dimension(:), allocatable `stat::mtmp`

8.14 tensor.f90 File Reference

Data Types

- type `tensor::coolist_elem`
Coordinate list element type. Elementary elements of the sparse tensors.
- type `tensor::coolist`
Coordinate list. Type used to represent the sparse tensor.

Modules

- module `tensor`
Tensor utility module.

Functions/Subroutines

- subroutine, public `tensor::copy_tensor` (src, dst)
Routine to copy a tensor.
- subroutine, public `tensor::mat_to_coo` (src, dst)
Routine to convert a matrix to a tensor.
- subroutine, public `tensor::sparse_mul3` (coolist_ijk, arr_j, arr_k, res)

Sparse multiplication of a tensor with two vectors:
$$\sum_{j,k=0}^{ndim} \mathcal{T}_{i,j,k} a_j b_k.$$

- subroutine, public `tensor::jsparse_mul` (coolist_ijk, arr_j, jcoo_ij)
Sparse multiplication of two tensors to determine the Jacobian:

$$J_{i,j} = \sum_{k=0}^{ndim} (\mathcal{T}_{i,j,k} + \mathcal{T}_{i,k,j}) a_k.$$

It's implemented slightly differently: for every $\mathcal{T}_{i,j,k}$, we add to $J_{i,j}$ as follows:

$$J_{i,j} = J_{i,j} + \mathcal{T}_{i,j,k} a_k \quad J_{i,k} = J_{i,k} + \mathcal{T}_{i,j,k} a_j$$

This version return a coolist (sparse tensor).

- subroutine, public `tensor::jsparse_mul_mat` (coolist_ijk, arr_j, jcoo_ij)

Sparse multiplication of two tensors to determine the Jacobian:

$$J_{i,j} = \sum_{k=0}^{ndim} (\mathcal{T}_{i,j,k} + \mathcal{T}_{i,k,j}) a_k.$$

It's implemented slightly differently: for every $\mathcal{T}_{i,j,k}$, we add to $J_{i,j}$ as follows:

$$J_{i,j} = J_{i,j} + \mathcal{T}_{i,j,k} a_k J_{i,k} = J_{i,k} + \mathcal{T}_{i,j,k} a_j$$

This version return a matrix.

- subroutine, public [tensor::sparse_mul2](#) (coolist_ij, arr_j, res)

Sparse multiplication of a 2d sparse tensor with a vectors: $\sum_{j=0}^{ndim} \mathcal{T}_{i,j,k} a_j b_k$.

- subroutine, public [tensor::simplify](#) (tensor)

Routine to simplify a coolist (sparse tensor). For each index i , it upper triangularize the matrix

$$\mathcal{T}_{i,j,k} \quad 0 \leq j, k \leq ndim.$$

Variables

- real(kind=8), parameter [tensor::real_eps](#) = 2.2204460492503131e-16

Parameter to test the equality with zero.

8.15 test_aotensor.f90 File Reference

Functions/Subroutines

- program [test_aotensor](#)
Small program to print the inner products.

8.15.1 Function/Subroutine Documentation

8.15.1.1 program test_aotensor ()

Small program to print the inner products.

Copyright

2015 Lesley De Cruz & Jonathan Demaeyer. See [LICENSE.txt](#) for license information.

Definition at line 13 of file test_aotensor.f90.

8.16 test_inprod_analytic.f90 File Reference

Functions/Subroutines

- program [inprod_analytic_test](#)
Small program to print the inner products.

8.16.1 Function/Subroutine Documentation

8.16.1.1 program inprod_analytic_test ()

Small program to print the inner products.

Copyright

2015 Lesley De Cruz & Jonathan Demaeyer. See [LICENSE.txt](#) for license information.

Remarks

Print in the same order as test_inprod.lua

Definition at line 18 of file test_inprod_analytic.f90.

8.17 test_tl_ad.f90 File Reference

Functions/Subroutines

- program [test_tl_ad](#)
Tests for the Tangent Linear (TL) and Adjoint (AD) model versions of MAOOAM.
- real(kind=8) function [gasdev](#) (idum)
- real(kind=8) function [ran2](#) (idum)

8.17.1 Function/Subroutine Documentation

8.17.1.1 real(kind=8) function gasdev (integer idum)

Definition at line 149 of file test_tl_ad.f90.

```

149  INTEGER :: idum
150  REAL(KIND=8) :: gasdev, ran2
151  !   USES ran2
152  INTEGER :: iset
153  REAL(KIND=8) :: fac, gset, rsq, v1, v2
154  SAVE iset, gset
155  DATA iset/0/
156  if (idum.lt.0) iset=0
157  if (iset.eq.0) then
158  1    v1=2.d0*ran2(idum)-1.
159      v2=2.d0*ran2(idum)-1.
160      rsq=v1**2+v2**2
161      if (rsq.ge.1.d0.or.rsq.eq.0.d0) goto 1
162      fac=sqrt(-2.*log(rsq)/rsq)
163      gset=v1*fac
164      gasdev=v2*fac
165      iset=1
166  else
167      gasdev=gset
168      iset=0
169  endif
170  return

```

8.17.1.2 real(kind=8) function ran2 (integer idum)

Definition at line 174 of file test_tl_ad.f90.

```

174  INTEGER :: idum,im1,im2,imm1,ia1,ia2,iq1,iq2,ir1,ir2,ntab,ndiv
175  REAL(KIND=8) :: ran2,am,eps,rnm
176  parameter(im1=2147483563,im2=2147483399,am=1.d0/im1,imm1=im1-1&
177    &,ia1=40014,ia2=40692,iq1=53668,iq2=52774,ir1=12211,ir2&
178    &=3791,ntab=32,ndiv=1+imm1/ntab,eps=1.2d-7,rnm=1.d0-eps)
179  INTEGER :: idum2,j,k,iv(ntab),iy
180  SAVE iv,iy,idum2
181  DATA idum2/123456789/, iv/ntab*0/, iy/0/
182  if (idum.le.0) then
183    idum=max(-idum,1)
184    idum2=idum
185    do j=ntab+8,1,-1
186      k=idum/iq1
187      idum=ia1*(idum-k*iq1)-k*ir1
188      if (idum.lt.0) idum=idum+im1
189      if (j.le.ntab) iv(j)=idum
190    enddo
191    iy=iv(1)
192  endif
193  k=idum/iq1
194  idum=ia1*(idum-k*iq1)-k*ir1
195  if (idum.lt.0) idum=idum+im1
196  k=idum2/iq2
197  idum2=ia2*(idum2-k*iq2)-k*ir2
198  if (idum2.lt.0) idum2=idum2+im2
199  j=1+iy/ndiv
200  iy=iv(j)-idum2
201  iv(j)=idum
202  if (iy.lt.1) iy=iy+imm1
203  ran2=min(am*iy,rnm)
204  return

```

8.17.1.3 program test_tl_ad ()

Tests for the Tangent Linear (TL) and Adjoint (AD) model versions of MAOOAM.

Copyright

2016 Lesley De Cruz & Jonathan Demaeyer. See [LICENSE.txt](#) for license information.

Definition at line 14 of file test_tl_ad.f90.

8.18 tl_ad_tensor.f90 File Reference

Modules

- module [tl_ad_tensor](#)

Tangent Linear (TL) and Adjoint (AD) model versions of MAOOAM. Tensors definition module.

Functions/Subroutines

- type(coolist) function, dimension(ndim) [tl_ad_tensor::jacobian](#) (ystar)
Compute the Jacobian of MAOOAM in point ystar.
- real(kind=8) function, dimension(ndim, ndim) [tl_ad_tensor::jacobian_mat](#) (ystar)
Compute the Jacobian of MAOOAM in point ystar.
- subroutine, public [tl_ad_tensor::init_tltensor](#)
Routine to initialize the TL tensor.
- subroutine [tl_ad_tensor::compute_tltensor](#) (func)
Routine to compute the TL tensor from the original MAOOAM one.
- subroutine [tl_ad_tensor::tl_add_count](#) (i, j, k, v)
Subroutine used to count the number of TL tensor entries.
- subroutine [tl_ad_tensor::tl_coeff](#) (i, j, k, v)
Subroutine used to compute the TL tensor entries.
- subroutine, public [tl_ad_tensor::init_adtensor](#)
Routine to initialize the AD tensor.
- subroutine [tl_ad_tensor::compute_adtensor](#) (func)
Subroutine to compute the AD tensor from the original MAOOAM one.
- subroutine [tl_ad_tensor::ad_add_count](#) (i, j, k, v)
Subroutine used to count the number of AD tensor entries.
- subroutine [tl_ad_tensor::ad_coeff](#) (i, j, k, v)
- subroutine, public [tl_ad_tensor::init_adtensor_ref](#)
Alternate method to initialize the AD tensor from the TL tensor.
- subroutine [tl_ad_tensor::compute_adtensor_ref](#) (func)
Alternate subroutine to compute the AD tensor from the TL one.
- subroutine [tl_ad_tensor::ad_add_count_ref](#) (i, j, k, v)
Alternate subroutine used to count the number of AD tensor entries from the TL tensor.
- subroutine [tl_ad_tensor::ad_coeff_ref](#) (i, j, k, v)
Alternate subroutine used to compute the AD tensor entries from the TL tensor.
- subroutine, public [tl_ad_tensor::ad](#) (t, ystar, deltat, buf)
Tendencies for the AD of MAOOAM in point ystar for perturbation deltat.
- subroutine, public [tl_ad_tensor::tl](#) (t, ystar, deltat, buf)
Tendencies for the TL of MAOOAM in point ystar for perturbation deltat.

Variables

- real(kind=8), parameter [tl_ad_tensor::real_eps](#) = 2.2204460492503131e-16
Epsilon to test equality with 0.
- integer, dimension(:), allocatable [tl_ad_tensor::count_elems](#)
Vector used to count the tensor elements.
- type(coolist), dimension(:), allocatable, public [tl_ad_tensor::tltensor](#)
Tensor representation of the Tangent Linear tendencies.
- type(coolist), dimension(:), allocatable, public [tl_ad_tensor::adtensor](#)
Tensor representation of the Adjoint tendencies.

8.19 util.f90 File Reference

Modules

- module [util](#)
Utility module.

Functions/Subroutines

- character(len=20) function, public `util::str` (k)
Convert an integer to string.
- character(len=40) function, public `util::rstr` (x, fm)
Convert a real to string with a given format.
- subroutine, public `util::init_random_seed` ()
Random generator initialization routine.
- integer function `lcg` (s)
- subroutine, public `util::init_one` (A)
Initialize a square matrix A as a unit matrix.

8.19.1 Function/Subroutine Documentation

8.19.1.1 integer function `init_random_seed::lcg` (integer(int64) s)

Definition at line 84 of file util.f90.

```
84      integer :: lcg
85      integer(int64) :: s
86      IF (s == 0) THEN
87          s = 104729
88      ELSE
89          s = mod(s, 4294967296_int64)
90      END IF
91      s = mod(s * 279470273_int64, 4294967291_int64)
92      lcg = int(mod(s, int(huge(0), int64)), kind(0))
93      END FUNCTION lcg
```


Index

- a
 - [aotensor_def](#), [14](#)
 - [inprod_analytic::atm_tensors](#), [77](#)
- acc
 - [stat](#), [59](#)
- ad
 - [tl_ad_tensor](#), [68](#)
- ad_add_count
 - [tl_ad_tensor](#), [69](#)
- ad_add_count_ref
 - [tl_ad_tensor](#), [69](#)
- ad_coeff
 - [tl_ad_tensor](#), [70](#)
- ad_coeff_ref
 - [tl_ad_tensor](#), [70](#)
- ad_step
 - [rk2_tl_ad_integrator](#), [51](#)
 - [rk4_tl_ad_integrator](#), [56](#)
- add_count
 - [aotensor_def](#), [14](#)
- adtensor
 - [tl_ad_tensor](#), [75](#)
- ams
 - [params](#), [37](#)
- aotensor
 - [aotensor_def](#), [17](#)
- aotensor_def, [13](#)
 - [a](#), [14](#)
 - [add_count](#), [14](#)
 - [aotensor](#), [17](#)
 - [coeff](#), [14](#)
 - [compute_aotensor](#), [15](#)
 - [count_elems](#), [17](#)
 - [init_aotensor](#), [15](#)
 - [kdelta](#), [16](#)
 - [psi](#), [16](#)
 - [real_eps](#), [17](#)
 - [t](#), [16](#)
 - [theta](#), [16](#)
- [aotensor_def.f90](#), [85](#)
- atmos
 - [inprod_analytic](#), [32](#)
- awavenum
 - [inprod_analytic](#), [32](#)
- b
 - [inprod_analytic::atm_tensors](#), [77](#)
- b1
 - [inprod_analytic](#), [21](#)
- b2
 - [inprod_analytic](#), [21](#)
- betp
 - [params](#), [37](#)
- buf_f0
 - [rk2_integrator](#), [49](#)
 - [rk2_tl_ad_integrator](#), [52](#)
- buf_f1
 - [rk2_integrator](#), [49](#)
 - [rk2_tl_ad_integrator](#), [52](#)
- buf_ka
 - [rk4_integrator](#), [54](#)
 - [rk4_tl_ad_integrator](#), [57](#)
- buf_kb
 - [rk4_integrator](#), [55](#)
 - [rk4_tl_ad_integrator](#), [57](#)
- buf_y1
 - [rk2_integrator](#), [50](#)
 - [rk2_tl_ad_integrator](#), [52](#)
 - [rk4_integrator](#), [55](#)
 - [rk4_tl_ad_integrator](#), [57](#)
- c
 - [inprod_analytic::atm_tensors](#), [77](#)
 - [inprod_analytic::ocean_tensors](#), [82](#)
- CLAIM
 - [LICENSE.txt](#), [89](#)
- CONTRACT
 - [LICENSE.txt](#), [90](#)
- ca
 - [params](#), [37](#)
- calculate_a
 - [inprod_analytic](#), [22](#)
- calculate_b
 - [inprod_analytic](#), [22](#)
- calculate_c_atm
 - [inprod_analytic](#), [22](#)
- calculate_c_oc
 - [inprod_analytic](#), [23](#)
- calculate_d
 - [inprod_analytic](#), [24](#)
- calculate_g
 - [inprod_analytic](#), [24](#)
- calculate_k
 - [inprod_analytic](#), [25](#)
- calculate_m
 - [inprod_analytic](#), [26](#)
- calculate_n
 - [inprod_analytic](#), [26](#)
- calculate_o
 - [inprod_analytic](#), [27](#)

- calculate_s
 - inprod_analytic, 28
- calculate_w
 - inprod_analytic, 28
- charge
 - LICENSE.txt, 89
- co
 - params, 37
- coeff
 - aotensor_def, 14
- compute_adtensor
 - tl_ad_tensor, 71
- compute_adtensor_ref
 - tl_ad_tensor, 71
- compute_aotensor
 - aotensor_def, 15
- compute_tltensor
 - tl_ad_tensor, 71
- conditions
 - LICENSE.txt, 90
- copy
 - LICENSE.txt, 90
- copy_tensor
 - tensor, 62
- count_elems
 - aotensor_def, 17
 - tl_ad_tensor, 75
- cpa
 - params, 38
- cpo
 - params, 38
- d
 - inprod_analytic::atm_tensors, 78
 - params, 38
- deallocate_inprod
 - inprod_analytic, 29
- delta
 - inprod_analytic, 30
- distribute
 - LICENSE.txt, 90
- doc/gen_doc.md, 86
- doc/tl_ad_doc.md, 86
- dp
 - params, 38
- dt
 - params, 38
- elems
 - tensor::coolist, 80
- epsa
 - params, 39
- exists
 - ic_def, 19
- f0
 - params, 39
- FROM
 - LICENSE.txt, 90
- files
 - LICENSE.txt, 89
- flambda
 - inprod_analytic, 30
- g
 - inprod_analytic::atm_tensors, 78
 - params, 39
- ga
 - params, 39
- gasdev
 - test_tl_ad.f90, 100
- go
 - params, 39
- gp
 - params, 40
- h
 - inprod_analytic::atm_wavenum, 78
 - inprod_analytic::ocean_wavenum, 83
 - params, 40
- i
 - stat, 60
- IMPLIED
 - LICENSE.txt, 90
- ic
 - ic_def, 19
- ic_def, 17
 - exists, 19
 - ic, 19
 - load_ic, 18
- ic_def.f90, 86
- init_adtensor
 - tl_ad_tensor, 71
- init_adtensor_ref
 - tl_ad_tensor, 72
- init_aotensor
 - aotensor_def, 15
- init_inprod
 - inprod_analytic, 30
- init_integrator
 - rk2_integrator, 48
 - rk4_integrator, 53
- init_nml
 - params, 36
- init_one
 - util, 76
- init_params
 - params, 36
- init_random_seed
 - util, 76
- init_stat
 - stat, 59
- init_tl_ad_integrator
 - rk2_tl_ad_integrator, 51
 - rk4_tl_ad_integrator, 56
- init_tltensor
 - tl_ad_tensor, 72

- inprod_analytic, 20
 - atmos, 32
 - awavenum, 32
 - b1, 21
 - b2, 21
 - calculate_a, 22
 - calculate_b, 22
 - calculate_c_atm, 22
 - calculate_c_oc, 23
 - calculate_d, 24
 - calculate_g, 24
 - calculate_k, 25
 - calculate_m, 26
 - calculate_n, 26
 - calculate_o, 27
 - calculate_s, 28
 - calculate_w, 28
 - deallocate_inprod, 29
 - delta, 30
 - flambda, 30
 - init_inprod, 30
 - ocean, 32
 - owavenum, 33
 - s1, 31
 - s2, 31
 - s3, 32
 - s4, 32
- inprod_analytic.f90, 86
- inprod_analytic::atm_tensors, 77
 - a, 77
 - b, 77
 - c, 77
 - d, 78
 - g, 78
 - s, 78
- inprod_analytic::atm_wavenum, 78
 - h, 78
 - m, 78
 - nx, 79
 - ny, 79
 - p, 79
 - typ, 79
- inprod_analytic::ocean_tensors, 81
 - c, 82
 - k, 82
 - m, 82
 - n, 82
 - o, 82
 - w, 82
- inprod_analytic::ocean_wavenum, 83
 - h, 83
 - nx, 83
 - ny, 83
 - p, 83
- inprod_analytic_test
 - test_inprod_analytic.f90, 100
- iter
 - stat, 59
- j
 - tensor::coolist_elem, 81
- jacobian
 - tl_ad_tensor, 72
- jacobian_mat
 - tl_ad_tensor, 73
- jsparse_mul
 - tensor, 63
- jsparse_mul_mat
 - tensor, 63
- k
 - inprod_analytic::ocean_tensors, 82
 - params, 40
 - tensor::coolist_elem, 81
- KIND
 - LICENSE.txt, 90
- kd
 - params, 40
- kdelta
 - aotensor_def, 16
- kdp
 - params, 40
- kp
 - params, 41
- l
 - params, 41
- LIABILITY
 - LICENSE.txt, 91
- LICENSE.txt, 88
 - CLAIM, 89
 - CONTRACT, 90
 - charge, 89
 - conditions, 90
 - copy, 90
 - distribute, 90
 - FROM, 90
 - files, 89
 - IMPLIED, 90
 - KIND, 90
 - LIABILITY, 91
 - License, 89
 - MERCHANTABILITY, 91
 - merge, 91
 - modify, 91
 - OTHERWISE, 91
 - publish, 91
 - restriction, 91
 - so, 92
 - Software, 92
 - sublicense, 92
 - use, 92
- lambda
 - params, 41
- lcg
 - util.f90, 103
- License
 - LICENSE.txt, 89

- load_ic
 - ic_def, [18](#)
- lpa
 - params, [41](#)
- lpo
 - params, [41](#)
- lr
 - params, [42](#)
- lsbpa
 - params, [42](#)
- lsbpo
 - params, [42](#)
- m
 - inprod_analytic::atm_wavenum, [78](#)
 - inprod_analytic::ocean_tensors, [82](#)
 - stat, [60](#)
- MERCHANTABILITY
 - LICENSE.txt, [91](#)
- maooam
 - maooam.f90, [92](#)
- maooam.f90, [92](#)
 - maooam, [92](#)
- mat_to_coo
 - tensor, [64](#)
- mean
 - stat, [59](#)
- merge
 - LICENSE.txt, [91](#)
- modify
 - LICENSE.txt, [91](#)
- mprev
 - stat, [60](#)
- mtmp
 - stat, [60](#)
- n
 - inprod_analytic::ocean_tensors, [82](#)
 - params, [42](#)
- natm
 - params, [42](#)
- nbatm
 - params, [43](#)
- nboc
 - params, [43](#)
- ndim
 - params, [43](#)
- nelems
 - tensor::coolist, [80](#)
- noc
 - params, [43](#)
- nx
 - inprod_analytic::atm_wavenum, [79](#)
 - inprod_analytic::ocean_wavenum, [83](#)
- ny
 - inprod_analytic::atm_wavenum, [79](#)
 - inprod_analytic::ocean_wavenum, [83](#)
- o
 - inprod_analytic::ocean_tensors, [82](#)
- OTHERWISE
 - LICENSE.txt, [91](#)
- ocean
 - inprod_analytic, [32](#)
- oms
 - params, [43](#)
- owavenum
 - inprod_analytic, [33](#)
- p
 - inprod_analytic::atm_wavenum, [79](#)
 - inprod_analytic::ocean_wavenum, [83](#)
- params, [33](#)
 - ams, [37](#)
 - betp, [37](#)
 - ca, [37](#)
 - co, [37](#)
 - cpa, [38](#)
 - cpo, [38](#)
 - d, [38](#)
 - dp, [38](#)
 - dt, [38](#)
 - epsa, [39](#)
 - f0, [39](#)
 - g, [39](#)
 - ga, [39](#)
 - go, [39](#)
 - gp, [40](#)
 - h, [40](#)
 - init_nml, [36](#)
 - init_params, [36](#)
 - k, [40](#)
 - kd, [40](#)
 - kdp, [40](#)
 - kp, [41](#)
 - l, [41](#)
 - lambda, [41](#)
 - lpa, [41](#)
 - lpo, [41](#)
 - lr, [42](#)
 - lsbpa, [42](#)
 - lsbpo, [42](#)
 - n, [42](#)
 - natm, [42](#)
 - nbatm, [43](#)
 - nboc, [43](#)
 - ndim, [43](#)
 - noc, [43](#)
 - oms, [43](#)
 - phi0, [44](#)
 - phi0_npi, [44](#)
 - pi, [44](#)
 - r, [44](#)
 - rp, [44](#)
 - rr, [45](#)
 - rra, [45](#)
 - sb, [45](#)
 - sbpa, [45](#)

- sbpo, 45
- sc, 46
- scale, 46
- sig0, 46
- t_run, 46
- t_trans, 46
- ta0, 47
- to0, 47
- tw, 47
- writeout, 47
- params.f90, 93
- phi0
 - params, 44
- phi0_npi
 - params, 44
- pi
 - params, 44
- psi
 - aotensor_def, 16
- publish
 - LICENSE.txt, 91
- r
 - params, 44
- ran2
 - test_tl_ad.f90, 100
- real_eps
 - aotensor_def, 17
 - tensor, 67
 - tl_ad_tensor, 75
- reset
 - stat, 59
- restriction
 - LICENSE.txt, 91
- rk2_integrator, 47
 - buf_f0, 49
 - buf_f1, 49
 - buf_y1, 50
 - init_integrator, 48
 - step, 48
 - tendencies, 49
- rk2_integrator.f90, 95
- rk2_tl_ad_integrator, 50
 - ad_step, 51
 - buf_f0, 52
 - buf_f1, 52
 - buf_y1, 52
 - init_tl_ad_integrator, 51
 - tl_step, 51
- rk2_tl_ad_integrator.f90, 96
- rk4_integrator, 52
 - buf_ka, 54
 - buf_kb, 55
 - buf_y1, 55
 - init_integrator, 53
 - step, 53
 - tendencies, 54
- rk4_integrator.f90, 96
- rk4_tl_ad_integrator, 55
 - ad_step, 56
 - buf_ka, 57
 - buf_kb, 57
 - buf_y1, 57
 - init_tl_ad_integrator, 56
 - tl_step, 56
- rk4_tl_ad_integrator.f90, 97
- rp
 - params, 44
- rr
 - params, 45
- rra
 - params, 45
- rstr
 - util, 76
- s
 - inprod_analytic::atm_tensors, 78
- s1
 - inprod_analytic, 31
- s2
 - inprod_analytic, 31
- s3
 - inprod_analytic, 32
- s4
 - inprod_analytic, 32
- sb
 - params, 45
- sbpa
 - params, 45
- sbpo
 - params, 45
- sc
 - params, 46
- scale
 - params, 46
- sig0
 - params, 46
- simplify
 - tensor, 65
- so
 - LICENSE.txt, 92
- Software
 - LICENSE.txt, 92
- sparse_mul2
 - tensor, 66
- sparse_mul3
 - tensor, 66
- stat, 58
 - acc, 59
 - i, 60
 - init_stat, 59
 - iter, 59
 - m, 60
 - mean, 59
 - mprev, 60
 - mtmp, 60
 - reset, 59
 - v, 61

- var, 60
- stat.f90, 97
- step
 - rk2_integrator, 48
 - rk4_integrator, 53
- str
 - util, 76
- sublicense
 - LICENSE.txt, 92
- t
 - aotensor_def, 16
- t_run
 - params, 46
- t_trans
 - params, 46
- ta0
 - params, 47
- tendencies
 - rk2_integrator, 49
 - rk4_integrator, 54
- tensor, 61
 - copy_tensor, 62
 - jsparse_mul, 63
 - jsparse_mul_mat, 63
 - mat_to_coo, 64
 - real_eps, 67
 - simplify, 65
 - sparse_mul2, 66
 - sparse_mul3, 66
- tensor.f90, 98
- tensor::coolist, 79
 - elems, 80
 - nelems, 80
- tensor::coolist_elem, 80
 - j, 81
 - k, 81
 - v, 81
- test_aotensor
 - test_aotensor.f90, 99
- test_aotensor.f90, 99
 - test_aotensor, 99
- test_inprod_analytic.f90, 99
 - inprod_analytic_test, 100
- test_tl_ad
 - test_tl_ad.f90, 101
- test_tl_ad.f90, 100
 - gasdev, 100
 - ran2, 100
 - test_tl_ad, 101
- theta
 - aotensor_def, 16
- tl
 - tl_ad_tensor, 73
- tl_ad_tensor, 67
 - ad, 68
 - ad_add_count, 69
 - ad_add_count_ref, 69
 - ad_coeff, 70
 - ad_coeff_ref, 70
 - adtensor, 75
 - compute_adtensor, 71
 - compute_adtensor_ref, 71
 - compute_tltensor, 71
 - count_elems, 75
 - init_adtensor, 71
 - init_adtensor_ref, 72
 - init_tltensor, 72
 - jacobian, 72
 - jacobian_mat, 73
 - real_eps, 75
 - tl, 73
 - tl_add_count, 74
 - tl_coeff, 74
 - tltensor, 75
 - tl_ad_tensor.f90, 101
 - tl_add_count
 - tl_ad_tensor, 74
 - tl_coeff
 - tl_ad_tensor, 74
 - tl_step
 - rk2_tl_ad_integrator, 51
 - rk4_tl_ad_integrator, 56
 - tltensor
 - tl_ad_tensor, 75
 - to0
 - params, 47
 - tw
 - params, 47
 - typ
 - inprod_analytic::atm_wavenum, 79
 - use
 - LICENSE.txt, 92
 - util, 75
 - init_one, 76
 - init_random_seed, 76
 - rstr, 76
 - str, 76
 - util.f90, 102
 - lcg, 103
 - v
 - stat, 61
 - tensor::coolist_elem, 81
 - var
 - stat, 60
 - w
 - inprod_analytic::ocean_tensors, 82
 - writeout
 - params, 47