Reference Manual

Generated by Doxygen 1.8.11

Contents

Chapter 1

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

About

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, 2016.

for the MAOOAM original code, and with

- for the stochastic part.

Please cite both articles 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.decruz@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.

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.

The following files are part of the MAOOAM model alone:

- 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.

with the addition of the files:

- maooam_stoch.f90 : Stochastic implementation of MAOOAM.
- maooam_MTV.f90 : Main program - MTV implementation for MAOOAM.
- maooam_WL.f90 : Main program - WL implementation for MAOOAM.
- corrmod.f90 : Unresolved variables correlation matrix initialization module.
- corr_tensor.f90 : Correlations and derivatives for the memory term of the WL parameterization.
- dec_tensor.f90 : Tensor resolved-unresolved components decomposition module.

- int_comp.f90 : Utility module containing the routines to perform the integration of functions.
- int_corr.f90 : Module to compute or load the integrals of the correlation matrices.
- MAR.f90 : Multidimensional AutoRegressive (MAR) module to generate the correlation for the WL parameterization.
- memory.f90 : WL parameterization memory term M_3 computation module.
- MTV_int_tensor.f90 : MTV tensors computation module.
- MTV_sigma_tensor.f90 : MTV noise sigma matrices computation module.
- WL_tensor.f90 : WL tensors computation module.
- rk2_stoch_integrator.f90 : Stochastic RK2 integration routines module.
- rk2_ss_integrator.f90 : Stochastic uncoupled resolved nonlinear and tangent linear RK2 dynamics integration module.
- rk2_MTV_integrator.f90 : MTV RK2 integration routines module.
- rk2_WL_integrator.f90 : WL RK2 integration routines module.
- sf_def.f90 : Module to select the resolved-unresolved components.
- SF.nml : A namelist to select the resolved-unresolved components.
- sqrt_mod.f90 : Utility module with various routine to compute matrix square root.
- stoch_mod.f90 : Utility module containing the stochastic related routines.
- stoch_params.f90 : Stochastic models parameters module.
- stoch_params.nml : A namelist to specify the stochastic models parameters.

which belong specifically to the stochastic implementation.

MAOOAM 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(NB↩ATM,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 :

```
./maooam
```

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 :

```
./maooam
```

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`.

Stochastic code usage

The user first has to fill the MAOOAM model namelist files according to their needs (see the previous section). Additional namelist files for the fine tuning of the parameterization must then be filled, and some "definition" files (with the extension .def) must be provided. An example is provided with the code.

Full details over the parameterization options and definition files are available `here`.

The program "maooam_stoch" will generate the evolution of the full stochastic dynamics with the command:

```
./maooam_stoch
```

or any other dynamics if specified as an argument (see the header of `maooam_stoch.f90`). It will generate two files :

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

The program "maooam_MTV" will generate the evolution of the MTV parameterization evolution, with the command:

```
./maooam_MTV
```

It will generate three files :

- evol_MTV.dat : the recorded time evolution of the variables.
- ptend_MTV.dat : the recorded time evolution of the tendencies (used for debugging).
- mean_field_MTV.dat : the mean field (the climatology)

The program "maooam_WL" will generate the evolution of the MTV parameterization evolution, with the command:

```
./maooam_WL
```

It will generate three files :

- evol_WL.dat : the recorded time evolution of the variables.
- ptend_WL.dat : the recorded time evolution of the tendencies (used for debugging).
- mean_field_WL.dat : the mean field (the climatology)

MAOOAM Implementation notes

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

$$\frac{dz_i}{dt} = \sum_{j,k=0}^{ndim} \mathcal{T}_{i,j,k} \, z_k \, z_j$$

with $z_0 = 1$.

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 dz_i/dt proportional to $z_j \, z_k$.
- Furthermore, z_0 is always equal to 1, so that $\mathcal{T}_{i,0,0}$ is the constant contribution to dz_i/dt
- $\mathcal{T}_{i,j,0} + \mathcal{T}_{i,0,j}$ is the contribution to dz_i/dt which is linear in z_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.

Stochastic code implementation notes

A stochastic version of MAOOAM and two stochastic parameterization methods (MTV and WL) are provided with this code.

The stochastic version of MAOOAM is given by

$$\frac{d\boldsymbol{z}}{dt} = f(\boldsymbol{z}) + \boldsymbol{q} \cdot \boldsymbol{dW}(t)$$

where $\boldsymbol{dW}$ is a vector of standard Gaussian White noise and where several choice for $f(\boldsymbol{z})$ are available. For instance, the default choice is to use the full dynamics:

$$f(\boldsymbol{z}) = \sum_{j,k=0}^{ndim} \mathcal{T}_{i,j,k} \, z_k \, z_j.$$

The implementation uses the tensorial framework described above and add some noise to it. This stochastic version is further detailed `here`.

The MTV parameterization for MAOOAM is given by

$$\frac{d\boldsymbol{x}}{dt} = F_x(\boldsymbol{x}) + \frac{1}{\delta} R(\boldsymbol{x}) + G(\boldsymbol{x}) + \sqrt{2} \, \boldsymbol{\sigma}(\boldsymbol{x}) \cdot \boldsymbol{dW}$$

where $\boldsymbol{x}$ is the set of resolved variables and $\boldsymbol{dW}$ is a vector of standard Gaussian White noise. F_x is the set of tendencies of resolved system alone and δ is the timescale separation parameter.

The WL parameterizations for MAOOAM is given by

$$\frac{d\boldsymbol{x}}{dt} = F_x(\boldsymbol{x}) + \varepsilon \, M_1(\boldsymbol{x}) + \varepsilon^2 \, M_2(\boldsymbol{x}, t) + \varepsilon^2 \, M_3(\boldsymbol{x}, t)$$

where ε is the resolved-unresolved components coupling strenght and where the different terms M_i account for different effect.

The implementation for these two approaches uses the tensorial framework described above, with the addition of new tensors to account for the terms $R, G, \sigma, M_1, M_2, M_3$. They are detailed more completely `here`.

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

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

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

where $\boldsymbol{y}^*$ is the point where the Tangent model is defined (with $z_0^* = 1$).

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_tltensor() 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

Modular arbitrary-order ocean-atmosphere model: The MTV and WL parameterizations

The stochastic version of MAOOAM

The stochastic version of MAOOAM is given by

$$\frac{d\boldsymbol{z}}{dt} = f(\boldsymbol{z}) + \boldsymbol{q} \cdot \boldsymbol{dW}(t)$$

where $\boldsymbol{dW}$ is a vector of standard Gaussian White noise and where several choice for $f(\boldsymbol{z})$ are available. For instance, the default choice is to use the full dynamics:

$$f(\boldsymbol{z}) = \sum_{j,k=0}^{ndim} \mathcal{T}_{i,j,k} \, z_k \; z_j.$$

The implementation uses thus the tensorial framework of MAOOAM and add some noise to it. To study parameterization methods in MAOOAM, the models variables $\boldsymbol{z}$ is divised in two components: the resolved component $\boldsymbol{x}$ and the unresolved component $\boldsymbol{y}$ (see below for more details).

Since MAOOAM is a ocean-atmosphere model, it can be decomposed further into oceanic and atmospheric components:

$$\boldsymbol{z} = \{\boldsymbol{x}_{\mathsf{a}}, \boldsymbol{x}_{\mathsf{o}}, \boldsymbol{y}_{\mathsf{a}}, \boldsymbol{y}_{\mathsf{o}}\}$$

and in the present implementation, the noise amplitude can be set in each component:

$$\frac{d\boldsymbol{x}_{\mathsf{a}}}{dt} = f_{x,\mathsf{a}}(\boldsymbol{z}) + \boldsymbol{q}_{x,\mathsf{a}} \cdot \boldsymbol{dW}_{x,\mathsf{a}}(t)$$

$$\frac{d\boldsymbol{x}_{\mathsf{o}}}{dt} = f_{x,\mathsf{o}}(\boldsymbol{z}) + \boldsymbol{q}_{x,\mathsf{o}} \cdot \boldsymbol{dW}_{x,\mathsf{o}}(t)$$

$$\frac{d\boldsymbol{y}_{\mathsf{a}}}{dt} = f_{y,\mathsf{a}}(\boldsymbol{z}) + \boldsymbol{q}_{y,\mathsf{a}} \cdot \boldsymbol{dW}_{y,\mathsf{a}}(t)$$

$$\frac{d\boldsymbol{y}_{\mathsf{o}}}{dt} = f_{y,\mathsf{o}}(\boldsymbol{z}) + \boldsymbol{q}_{y,\mathsf{o}} \cdot \boldsymbol{dW}_{y,\mathsf{o}}(t)$$

through the parameters stoch_params::q_ar, stoch_params::q_au, stoch_params::q_or and stoch_params::q_ou.

The resolved-unresolved components

Due to the decomposition into resoved variables $\boldsymbol{x}$ and unresolved variables $\boldsymbol{y}$, the equation of the MAOOAM model can be rewritten:

$$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{H}^x + \boldsymbol{L}^{xx} \cdot \boldsymbol{x} + \boldsymbol{L}^{xy} \cdot \boldsymbol{y} + \boldsymbol{B}^{xxx} : \boldsymbol{x} \otimes \boldsymbol{x} + \boldsymbol{B}^{xxy} : \boldsymbol{x} \otimes \boldsymbol{y} + \boldsymbol{B}^{xyy} : \boldsymbol{y} \otimes \boldsymbol{y} + \boldsymbol{q}_x \cdot \boldsymbol{dW}_x$$
$$\frac{d\boldsymbol{y}}{dt} = \boldsymbol{H}^y + \boldsymbol{L}^{yx} \cdot \boldsymbol{x} + \boldsymbol{L}^{yy} \cdot \boldsymbol{y} + \boldsymbol{B}^{yxx} : \boldsymbol{x} \otimes \boldsymbol{x} + \boldsymbol{B}^{yxy} : \boldsymbol{x} \otimes \boldsymbol{y} + \boldsymbol{B}^{yyy} : \boldsymbol{y} \otimes \boldsymbol{y} + \boldsymbol{q}_y \cdot \boldsymbol{dW}_y$$

where $\boldsymbol{q}_x = \{\boldsymbol{q}_{x,\mathrm{a}}, \boldsymbol{q}_{x,\mathrm{o}}\}$ and $\boldsymbol{q}_y = \{\boldsymbol{q}_{y,\mathrm{a}}, \boldsymbol{q}_{y,\mathrm{o}}\}$. We have thus also $\boldsymbol{dW}_x = \{\boldsymbol{dW}_{x,\mathrm{a}}, \boldsymbol{dW}_{x,\mathrm{o}}\}$ and $\boldsymbol{dW}_y = \{\boldsymbol{dW}_{y,\mathrm{a}}, \boldsymbol{dW}_{y,\mathrm{o}}\}$. The various terms of the equations above are accessible in the dec_tensor module. To specify which variables belong to the resolved (unresolved) component, the user must fill the SF.nml namelist file by setting the component of the vector sf_def::sf to 0 (1). This file must be filled before starting any of the stochastic and parameterization codes. If this file is not present, launch one of the programs. It will generate a new SF.nml file and then abort.

The purpose of the parameterization is to reduce the $\boldsymbol{x}$ equation by closing it while keeping the statisical properies of the full system. To apply the parameterizations proposed in this implementation, we consider a modified version of the equation above:

$$\frac{d\boldsymbol{x}}{dt} = F_x(\boldsymbol{x}) + \boldsymbol{q}_x \cdot \boldsymbol{dW}_x + \frac{\varepsilon}{\delta}\, \Psi_x(\boldsymbol{x}, \boldsymbol{y})$$
$$\frac{d\boldsymbol{y}}{dt} = \frac{1}{\delta^2}\left(F_y(\boldsymbol{y}) + \delta\, \boldsymbol{q}_y \cdot \boldsymbol{dW}_y\right) + \frac{\varepsilon}{\delta}\, \Psi_y(\boldsymbol{x}, \boldsymbol{y})$$

where ε is the resolved-unresolved components coupling strength given by the parameter stoch_params::eps_pert. δ is the timescale separation parameter given by the parameter stoch_params::tdelta. By setting those to 1, one recover the first equations above.

The function Ψ_x includes all the $\boldsymbol{x}$ terms, and thus F_x and Ψ_x are unequivocally defined. On the other hand, depending on the value of the parameter stoch_params::mode, the terms regrouped in the function F_y can be different. Indeed, if stoch_params::mode is set to

- 'qfst', then:
  $$F_y(\boldsymbol{y}) = \boldsymbol{B}^{yyy} : \boldsymbol{y} \otimes \boldsymbol{y}$$
- 'ures', then:
  $$F_y(\boldsymbol{y}) = \boldsymbol{H}^y + \boldsymbol{L}^{yy} \cdot \boldsymbol{y} + \boldsymbol{B}^{yyy} : \boldsymbol{y} \otimes \boldsymbol{y}$$
  However, for the WL parameterization, this parameter must be set to 'ures' by definition. See the article accompagnying this code for more details.

The MTV parameterization

This parameterization is also called homogenization. Its acronym comes from the names of the authors that proposed this approach for climate modes: Majda, Timofeyev and Vanden Eijnden (Majda et al., 2001). It is given by

$$\frac{d\boldsymbol{x}}{dt} = F_X(\boldsymbol{x}) + \frac{1}{\delta} R(\boldsymbol{x}) + G(\boldsymbol{x}) + \sqrt{2}\, \boldsymbol{\sigma}(\boldsymbol{x}) \cdot \boldsymbol{dW}$$

where $\boldsymbol{x}$ is the set of resolved variables and $\boldsymbol{dW}$ is a vector of standard Gaussian White noise. F_x is the set of tendencies of resolved system alone and δ is the timescale separation parameter.

Correlations specification

The ingredients needed to compute the terms $R, G, \boldsymbol{\sigma}$ of this parametrization are the unresolved variables covariance matrix and the integrated correlation matrices. The unresolved variables covariance matrix is given by

$$\boldsymbol{\sigma}_y = \langle \boldsymbol{y} \otimes \boldsymbol{y} \rangle$$

and is present in the implementation through the matrices corrmod::corr_i and corrmod::corr_i_full. Their inverses are also available through corrmod::inv_corr_i and corrmod::inv_corr_i_full . The integrated correlation matrices are given by

$$\boldsymbol{\Sigma} = \int_0^\infty ds \langle \boldsymbol{y} \otimes \boldsymbol{y}^s \rangle$$

$$\boldsymbol{\Sigma}_2 = \int_0^\infty ds \ \left(\langle \boldsymbol{y} \otimes \boldsymbol{y}^s \rangle \otimes \langle \boldsymbol{y} \otimes \boldsymbol{y}^s \rangle \right)$$

and is present in the implementation through the matrices int_corr::corrint and int_corr::corr2int .

These matrices are computed from the correlation matrix $\langle \boldsymbol{y} \otimes \boldsymbol{y}^s \rangle$ which is accessible through the function corrmod::corrcomp. For instance, the covariance matrix $\boldsymbol{\sigma}_y$ is then simply the correlation matrix at the lagtime 0, and $\boldsymbol{\Sigma}$ and $\boldsymbol{\Sigma}_2$ can be computed via integration over the lagtime.

There exists three different ways to load the correlation matrix, specified by the value of the parameters stoch_↩ params::load_mode and stoch_params::int_corr_mode . The stoch_params::load_mode specify how the correlation matrix is loaded can take three different values:

- 'defi': from an analytical definition encoded in the corrmod module function corrmod::corrcomp_from_def .
- 'spli': from a spline definition file 'corrspline.def' .
- 'expo': from a fit with exponentials definition file 'correxpo.def'

The stoch_params::int_corr_mode specify how the correlation are integrated and can take two different values:

- 'file': Integration results provided by files 'corrint.def' and 'corr2int.def'
- 'prog': Integration computed directly by the program with the correlation matrix. Write 'corrint.def' and 'corr2int.def' files to be reused later.

These parameters can be set up in the namelist file stoch_params.nml . Examples of the ".def" files specifying the integrals are provided with the code.

Other MTV setup parameters

Some additional parameters complete the options possible for the MTV parameters :

- stoch_params::mnuti : Multiplicative noise update time interval – Time interval over which the matrix $\boldsymbol{\sigma}(\boldsymbol{x})$ is updated.
- stoch_params::t_trans_stoch : Transient period of the stochastic model.
- stoch_params::maxint : Specify the upper limit of the numerical integration if stoch_params::int_corr_mode is set to 'prog'.

Definition files

The following definition files are needed by the parameterization, depending on the value of the parameters described above. Examples of those files are joined to the code. The files include:

- 'correxpo.def': Coefficients a_i of the fit of the correlations with the function
$$a_4 + a_0 \, \exp\left(-\frac{t}{a_1}\right) \, \cos(a_2 \, t + a_3)$$
where t is the lag-time and τ is the decorrelation time. Used if stoch_params::load_mode is set to 'expo'.
- 'corrspline.def': Coefficients b_i of the spline used to model the correlation functions. Used if stoch_params↩ ::load_mode is set to 'spli'.
- 'corrint.def': File holding the matrix $\mathbf{\Sigma}$. Used if stoch_params::int_corr_mode is set to 'file'.
- 'corr2int.def': File holding the matrix $\mathbf{\Sigma}_2$.

The WL parameterization

This parameterization is based on the Ruelle response theory. Its acronym comes from the names of the authors that proposed this approach: Wouters and Lucarini (Wouters and Lucarini, 2012). It is given by

$$\frac{d\boldsymbol{x}}{dt} = F_x(\boldsymbol{x}) + \varepsilon \, M_1(\boldsymbol{x}) + \varepsilon^2 \, M_2(\boldsymbol{x},t) + \varepsilon^2 \, M_3(\boldsymbol{x},t)$$

where ε is the resolved-unresolved components coupling strength and where the different terms M_i account for average, correlation and memory effects.

Correlations specification

The ingredients needed to compute the M_i terms of this parametrization are the unresolved variable covariance matrix $\langle \boldsymbol{y} \otimes \boldsymbol{y} \rangle$ and correlation matrix $\langle \boldsymbol{y} \otimes \boldsymbol{y}^s \rangle$. The unresolved variables covariance matrix is given by

$$\boldsymbol{\sigma}_y = \langle \boldsymbol{y} \otimes \boldsymbol{y} \rangle$$

and is present in the implementation through the matrices corrmod::corr_i and corrmod::corr_i_full. Their inverses are also available through corrmod::inv_corr_i and corrmod::inv_corr_i_full.

The correlation matrix $\langle \boldsymbol{y} \otimes \boldsymbol{y}^s \rangle$ is accessible through the function corrmod::corrcomp.

As for the MTV case, there exists three different ways to load the correlation matrix, specified by the value of the parameters stoch_params::load_mode and stoch_params::int_corr_mode . The stoch_params::load_mode specify how the correlation matrix is loaded can take three different values:

- 'defi': from an analytical definition encoded in the corrmod module function corrmod::corrcomp_from_def .
- 'spli': from a spline definition file 'corrspline.def' .
- 'expo': from a fit with exponentials definition file 'correxpo.def'

The correlation term M_2 is emulated by an order m multidimensional AutoRegressive (MAR) process:

$$\boldsymbol{u}_n = \sum_{i=1}^{m} \boldsymbol{u}_{n-i} \cdot \boldsymbol{W}_i + \boldsymbol{Q} \cdot \boldsymbol{\xi}_n$$

of which the $\boldsymbol{W}_i$ and $\boldsymbol{Q}$ matrices are also needed (the $\boldsymbol{\xi}_n$ are vectors of standard Gaussian white noise). It is implemented in the MAR module.

Other WL setup parameters

Some additional parameters complete the options possible for the WL parameters :

- stoch_params::muti : Memory term M_3 update time interval.
- stoch_params::t_trans_stoch : Transient period of the stochastic model.
- stoch_params::meml : Time over which the memory kernel is numericaly integrated.
- stoch_params::t_trans_mem : Transient period of the stochastic model to initialize the memory term.
- stoch_params::dts : Intrisic resolved dynamics time step.
- stoch_params::x_int_mode : Integration mode for the resolved component (not used for the moment – must be set to 'reso').

Note that the stoch_params::mode must absolutely be set to 'ures', by definition.

Definition files

The following definition files are needed by the parameterization, depending on the value of the parameters described above. Examples of those files are joined to the code. The files include:

- 'correxpo.def': Coefficients a_i of the fit of the correlations with the function
$$a_4 + a_0 \,\exp\left(-\frac{t}{a_1}\right)\, \cos(a_2\, t + a_3)$$
where t is the lag-time and τ is the decorrelation time. Used if stoch_params::load_mode is set to 'expo'.
- 'corrspline.def': Coefficients b_i of the spline used to model the correlation functions. Used if stoch_params↩ ::load_mode is set to 'spli'.
- 'MAR_R_params.def': File specifying the $\boldsymbol{R} = \boldsymbol{Q}^2$ matrix for the MAR.
- 'MAR_W_params.def': File specifying the $\boldsymbol{W}_i$ matrices for the MAR.

The various terms are then constructed according to these definition files.

References

- Vannitsem, S., Demaeyer, J., De Cruz, L., and Ghil, M.: Low-frequency variability and heat transport in a loworder nonlinear coupled ocean-atmosphere model, Physica D: Nonlinear Phenomena, 309, 71-85, 2015.
- De Cruz, L., Demaeyer, J., & Vannitsem, S.: The Modular Arbitrary-Order Ocean-Atmosphere Model: MA↩ OOAM v1.0, Geoscientific Model Development, 9(8), 2793-2808, 2016.
- Majda, A. J., Timofeyev, I., & Vanden Eijnden, E.: A mathematical framework for stochastic climate models, Communications on Pure and Applied Mathematics, 54(8), 891-974, 2001.
- Franzke, C., Majda, A. J., & Vanden-Eijnden, E.: Low-order stochastic mode reduction for a realistic barotropic model climate, Journal of the atmospheric sciences, 62(6), 1722-1745, 2005.
- Wouters, J., & Lucarini, V.: Disentangling multi-level systems: averaging, correlations and memory. Journal of Statistical Mechanics: Theory and Experiment, 2012(03), P03003, 2012.
- Demaeyer, J., & Vannitsem, S.: Stochastic parametrization of subgrid-scale processes in coupled ocean–atmosphere systems: benefits and limitations of response theory, Quarterly Journal of the Royal Meteorological Society, 143(703), 881-896, 2017.

Please see the main article for the full list of references.

Chapter 4

Modules Index

4.1 Modules List

Here is a list of all modules with brief descriptions:

Chapter 5

Data Type Index

5.1 Data Types List

Here are the data types with brief descriptions:

Chapter 6

File Index

6.1 File List

Here is a list of all files with brief descriptions:

Chapter 7

Module Documentation

7.1 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.*

7.1.1 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

Remarks

Generated Fortran90/95 code from aotensor.lua

7.1.2 Function/Subroutine Documentation

7.1.2.1 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
```

7.1.2.2 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
```

7.1.2.3 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)%nelems)+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)%nelems=n
115      END IF
```

7.1.2.4 subroutine aotensor_def::compute_aotensor (external *func*) [private]

Subroutine to compute the tensor aotensor.

Parameters

func	External function to be used

Definition at line 132 of file aotensor_def.f90.

7.1.2.5 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 ! ***"
```

```
count_elems=0

CALL compute_aotensor(add_count)

DO i=1,ndim
   ALLOCATE(aotensor(i)%elems(count_elems(i)), stat=allocstat)
   IF (allocstat /= 0) stop "*** Not enough memory ! ***"
END DO

DEALLOCATE(count_elems, stat=allocstat)
IF (allocstat /= 0) stop "*** Deallocation problem ! ***"

CALL compute_aotensor(coeff)

CALL simplify(aotensor)

CALL deallocate_inprod ! Clean the inner product tensors

```

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

Kronecker delta function.

Definition at line 88 of file aotensor_def.f90.

```
INTEGER :: i,j,kdelta
kdelta=0
IF (i == j) kdelta = 1
```

7.1.2.7 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.

```
INTEGER :: i,psi
psi = i
```

7.1.2.8 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.

```
INTEGER :: i,t
t = i + 2 * natm + noc
```

7.1.2.9 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.

```
INTEGER :: i,theta
theta = i + natm
```

7.1.3 Variable Documentation

7.1.3.1 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
```

7.1.3.2 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
```

7.1.3.3 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
```

7.2 corr_tensor Module Reference

Module to compute the correlations and derivatives used to compute the memory term of the WL parameterization.

Functions/Subroutines

- subroutine, public init_corr_tensor

    Subroutine to initialise the correlations tensors.

Variables

- type(coolist), dimension(:,:), allocatable, public yy

    Coolist holding the $\langle Y \otimes Y^s \rangle$ terms.
- type(coolist), dimension(:,:), allocatable, public dy

    Coolist holding the $\langle \partial_Y \otimes Y^s \rangle$ terms.
- type(coolist), dimension(:,:), allocatable, public ydy

    Coolist holding the $\langle Y \otimes \partial_Y \otimes Y^s \rangle$ terms.
- type(coolist), dimension(:,:), allocatable, public dyy

    Coolist holding the $\langle \partial_Y \otimes Y^s \otimes Y^s \rangle$ terms.
- type(coolist4), dimension(:,:), allocatable, public ydyy

    Coolist holding the $\langle Y \otimes \partial_Y \otimes Y^s \otimes Y^s \rangle$ terms.
- real(kind=8), dimension(:), allocatable dumb_vec

    Dumb vector to be used in the calculation.
- real(kind=8), dimension(:,:), allocatable dumb_mat1

    Dumb matrix to be used in the calculation.
- real(kind=8), dimension(:,:), allocatable dumb_mat2

    Dumb matrix to be used in the calculation.
- real(kind=8), dimension(:,:), allocatable expm

    Matrix holding the product inv_corr_i$*$corr_ij at time s.

7.2.1 Detailed Description

Module to compute the correlations and derivatives used to compute the memory term of the WL parameterization.

Copyright

Remarks

7.2.2 Function/Subroutine Documentation

7.2.2.1 subroutine, public corr_tensor::init_corr_tensor ( )

Subroutine to initialise the correlations tensors.

Definition at line 45 of file corr_tensor.f90.

```
45      INTEGER :: i,j,m,allocstat
46
47      CALL init_corr
48
49      print*, 'Computing the time correlation tensors...'
50
51      ALLOCATE(yy(ndim,mems),dy(ndim,mems), dyy(ndim,mems), stat=allocstat)
52      IF (allocstat /= 0) stop "*** Not enough memory ! ***"
53
54      ALLOCATE(ydy(ndim,mems), ydyy(ndim,mems), stat=allocstat)
55      IF (allocstat /= 0) stop "*** Not enough memory ! ***"
56
57      ALLOCATE(dumb_vec(ndim), stat=allocstat)
```

```
IF (allocstat /= 0) stop "*** Not enough memory ! ***"

ALLOCATE(dumb_mat1(ndim,ndim), dumb_mat2(ndim,ndim), stat=allocstat)
IF (allocstat /= 0) stop "*** Not enough memory ! ***"

ALLOCATE(expm(n_unres,n_unres), stat=allocstat)
IF (allocstat /= 0) stop "*** Not enough memory ! ***"

DO m=1,mems
   CALL corrcomp((m-1)*muti)

   ! YY
   CALL ireduce(dumb_mat2,corr_ij,n_unres,ind,rind)
   CALL matc_to_coo(dumb_mat2,yy(:,m))

   ! dY
   expm=matmul(inv_corr_i,corr_ij)
   CALL ireduce(dumb_mat2,expm,n_unres,ind,rind)
   CALL matc_to_coo(dumb_mat2,dy(:,m))

   ! YdY
   DO i=1,n_unres
      CALL ireduce(dumb_mat2,mean(i)*expm,n_unres,ind,rind)
      CALL add_matc_to_tensor(ind(i),dumb_mat2,ydy(:,m))
   ENDDO

   ! dYY
   dumb_vec(1:n_unres)=matmul(mean,expm)
   DO i=1,n_unres
      CALL vector_outer(expm(i,:),dumb_vec(1:n_unres),dumb_mat2(1:n_unres,1:n_unres))
      CALL ireduce(dumb_mat1,dumb_mat2+transpose(dumb_mat2),n_unres,ind,rind)
      CALL add_matc_to_tensor(ind(i),dumb_mat1,dyy(:,m))
   ENDDO

   ! YdYY
   DO i=1,n_unres
      DO j=1,n_unres
         CALL vector_outer(corr_ij(i,:),expm(j,:),dumb_mat2(1:n_unres,1:n_unres))
         CALL ireduce(dumb_mat1,dumb_mat2+transpose(dumb_mat2),n_unres,ind,rind)
         CALL add_matc_to_tensor4(ind(i),ind(j),dumb_mat1,ydyy(:,m))
      ENDDO
   ENDDO
ENDDO

DEALLOCATE(dumb_mat1, dumb_mat2, stat=allocstat)
IF (allocstat /= 0)  stop "*** Problem to deallocate ! ***"

DEALLOCATE(dumb_vec, stat=allocstat)
IF (allocstat /= 0)  stop "*** Problem to deallocate ! ***"
```

7.2.3 Variable Documentation

7.2.3.1 real(kind=8), dimension(:,:), allocatable corr_tensor::dumb_mat1 [private]

Dumb matrix to be used in the calculation.

Definition at line 37 of file corr_tensor.f90.

```
37    REAL(KIND=8), DIMENSION(:,:), ALLOCATABLE :: dumb_mat1 !< Dumb matrix to be used in the calculation
```

7.2.3.2 real(kind=8), dimension(:,:), allocatable corr_tensor::dumb_mat2 [private]

Dumb matrix to be used in the calculation.

Definition at line 38 of file corr_tensor.f90.

```
38    REAL(KIND=8), DIMENSION(:,:), ALLOCATABLE :: dumb_mat2 !< Dumb matrix to be used in the calculation
```

7.2.3.3 real(kind=8), dimension(:), allocatable corr_tensor::dumb_vec [private]

Dumb vector to be used in the calculation.

Definition at line 36 of file corr_tensor.f90.

```
36   REAL(KIND=8), DIMENSION(:), ALLOCATABLE :: dumb_vec !< Dumb vector to be used in the calculation
```

7.2.3.4 type(coolist), dimension(:,:), allocatable, public corr_tensor::dy

Coolist holding the $\langle \partial_Y \otimes Y^s \rangle$ terms.

Definition at line 31 of file corr_tensor.f90.

```
31   TYPE(coolist), DIMENSION(:,:), ALLOCATABLE, PUBLIC :: dy !< Coolist holding the  \f$\langle \partial_Y
       \otimes Y^s \rangle\f$ terms
```

7.2.3.5 type(coolist), dimension(:,:), allocatable, public corr_tensor::dyy

Coolist holding the $\langle \partial_Y \otimes Y^s \otimes Y^s \rangle$ terms.

Definition at line 33 of file corr_tensor.f90.

```
33   TYPE(coolist), DIMENSION(:,:), ALLOCATABLE, PUBLIC :: dyy !< Coolist holding the  \f$\langle \partial_Y
       \otimes Y^s \otimes Y^s \rangle\f$ terms
```

7.2.3.6 real(kind=8), dimension(:,:), allocatable corr_tensor::expm [private]

Matrix holding the product inv_corr_i∗corr_ij at time s.

Definition at line 39 of file corr_tensor.f90.

```
39   REAL(KIND=8), DIMENSION(:,:), ALLOCATABLE :: expm !< Matrix holding the product inv_corr_i*corr_ij at
       time \f$s\f$
```

7.2.3.7 type(coolist), dimension(:,:), allocatable, public corr_tensor::ydy

Coolist holding the $\langle Y \otimes \partial_Y \otimes Y^s \rangle$ terms.

Definition at line 32 of file corr_tensor.f90.

```
32   TYPE(coolist), DIMENSION(:,:), ALLOCATABLE, PUBLIC :: ydy !< Coolist holding the  \f$\langle Y \otimes
       \partial_Y \otimes Y^s \rangle\f$ terms
```

7.2.3.8 type(coolist4), dimension(:,:), allocatable, public corr_tensor::ydyy

Coolist holding the $\langle Y \otimes \partial_Y \otimes Y^s \otimes Y^s \rangle$ terms.

Definition at line 34 of file corr_tensor.f90.

```
34   TYPE(coolist4), DIMENSION(:,:), ALLOCATABLE, PUBLIC :: ydyy !< Coolist holding the  \f$\langle Y \otimes
       \partial_Y \otimes Y^s \otimes Y^s \rangle\f$ terms
```

7.2.3.9 type(coolist), dimension(:,:), allocatable, public corr_tensor::yy

Coolist holding the $\langle Y \otimes Y^s \rangle$ terms.

Definition at line 30 of file corr_tensor.f90.

```
30   TYPE(coolist), DIMENSION(:,:), ALLOCATABLE, PUBLIC :: yy !< Coolist holding the  \f$\langle Y \otimes Y^s
       \rangle\f$ terms
```

7.3 corrmod Module Reference

Module to initialize the correlation matrix of the unresolved variables.

Functions/Subroutines

- subroutine, public init_corr

  Subroutine to initialise the computation of the correlation.
- subroutine corrcomp_from_def (s)

  Subroutine to compute the correlation of the unresolved variables $\langle Y \otimes Y^s \rangle$ at time s from the definition given inside the module.
- subroutine corrcomp_from_spline (s)

  Subroutine to compute the correlation of the unresolved variables $\langle Y \otimes Y^s \rangle$ at time s from the spline representation.
- subroutine splint (xa, ya, y2a, n, x, y)

  Routine to compute the spline representation parameters.
- real(kind=8) function fs (s, p)

  Exponential fit function.
- subroutine corrcomp_from_fit (s)

  Subroutine to compute the correlation of the unresolved variables $\langle Y \otimes Y^s \rangle$ at time s from the exponential representation.

Variables

- real(kind=8), dimension(:), allocatable, public mean

  Vector holding the mean of the unresolved dynamics (reduced version)
- real(kind=8), dimension(:), allocatable, public mean_full

  Vector holding the mean of the unresolved dynamics (full version)
- real(kind=8), dimension(:,:), allocatable, public corr_i_full

  Covariance matrix of the unresolved variables (full version)
- real(kind=8), dimension(:,:), allocatable, public inv_corr_i_full

  Inverse of the covariance matrix of the unresolved variables (full version)
- real(kind=8), dimension(:,:), allocatable, public corr_i

  Covariance matrix of the unresolved variables (reduced version)
- real(kind=8), dimension(:,:), allocatable, public inv_corr_i

  Inverse of the covariance matrix of the unresolved variables (reduced version)
- real(kind=8), dimension(:,:), allocatable, public corr_ij

  Matrix holding the correlation matrix at a given time.
- real(kind=8), dimension(:,:,:), allocatable y2

  Vector holding coefficient of the spline and exponential correlation representation.
- real(kind=8), dimension(:,:,:), allocatable ya

  Vector holding coefficient of the spline and exponential correlation representation.
- real(kind=8), dimension(:), allocatable xa

  Vector holding coefficient of the spline and exponential correlation representation.
- integer nspl

  Integers needed by the spline representation of the correlation.
- integer klo
- integer khi
- procedure(corrcomp_from_spline), pointer, public corrcomp

  Pointer to the correlation computation routine.

7.3.1 Detailed Description

Module to initialize the correlation matrix of the unresolved variables.

Copyright

Remarks

7.3.2 Function/Subroutine Documentation

7.3.2.1 subroutine corrmod::corrcomp_from_def (real(kind=8), intent(in) *s*) [private]

Subroutine to compute the correlation of the unresolved variables $\langle Y \otimes Y^s \rangle$ at time s from the definition given inside the module.

Parameters

s	time s at which the correlation is computed

Definition at line 148 of file corrmod.f90.

```
148     REAL(KIND=8), INTENT(IN) :: s
149     REAL(KIND=8) :: y
150     INTEGER :: i,j
151 
152     ! Definition of the corr_ij matrix as a function of time
153 
154     corr_ij(10,10)=((7.66977252307669*cos(0.039597160512071454*s))/exp(0.06567483898489704*s) + (1.02409061
      73830213*cos(&
155          &0.07283568782600224*s))/exp(0.017262015588746404*s) - (0.6434985372062336*sin(0.03959716051207145
      4*s&
156          &))/exp(0.06567483898489704*s) + (0.6434985372062335*sin(0.07283568782600224*s))/exp(0.01726201558
      8746404*s))
157     corr_ij(10,9)=((0.6434985372062321*cos(0.039597160512071454*s))/exp(0.06567483898489704*s) - (0.6434985
      372062324*co&
158          &s(0.07283568782600224*s))/exp(0.017262015588746404*s) + (7.669772523076694*sin(0.0395971605120714
      54*&
159          &s))/exp(0.06567483898489704*s) + (1.024090617383021*sin(0.07283568782600224*s))/exp(0.01726201558
      8746404*s))
160     corr_ij(10,8)=0
161     corr_ij(10,7)=0
162     corr_ij(10,6)=0
163     corr_ij(10,5)=((-2.236364132659011*cos(0.039597160512071454*s))/exp(0.06567483898489704*s) + (6.9528041
      48086198*cos&
164          &(0.07283568782600224*s))/exp(0.017262015588746404*s) - (1.4494534432272481*sin(0.0395971605120714
      54*&
165          &s))/exp(0.06567483898489704*s) - (0.6818177416446283*sin(0.07283568782600224*s))/exp(0.0172620155
      88746404*s))
166     corr_ij(10,4)=((1.4494534432272483*cos(0.039597160512071454*s))/exp(0.06567483898489704*s) + (0.6818177
      416446293*co&
167          &s(0.07283568782600224*s))/exp(0.017262015588746404*s) - (2.2363641326590127*sin(0.039597160512071
      454&
168          &*s))/exp(0.06567483898489704*s) + (6.952804148086195*sin(0.07283568782600224*s))/exp(0.0172620155
      88746404*s))
169     corr_ij(10,3)=0
170     corr_ij(10,2)=0
171     corr_ij(10,1)=0
172     corr_ij(9,10)=((-0.6434985372062344*cos(0.039597160512071454*s))/exp(0.06567483898489704*s) + (0.643498
      537206234*co&
173          &s(0.07283568782600224*s))/exp(0.017262015588746404*s) - (7.669772523076689*sin(0.0395971605120714
      54*&
174          &s))/exp(0.06567483898489704*s) - (1.0240906173830204*sin(0.07283568782600224*s))/exp(0.0172620155
      88746404*s))
175     corr_ij(9,9)=((7.66977252307669*cos(0.039597160512071454*s))/exp(0.06567483898489704*s) + (1.0240906173
      830204*cos(&
176          &0.07283568782600224*s))/exp(0.017262015588746404*s) - (0.643498537206233*sin(0.039597160512071454
      *s)&
177          &)/exp(0.06567483898489704*s) + (0.6434985372062327*sin(0.07283568782600224*s))/exp(0.017262015588
      746404*s))
178     corr_ij(9,8)=0
179     corr_ij(9,7)=0
180     corr_ij(9,6)=0
181     corr_ij(9,5)=((-1.4494534432272477*cos(0.039597160512071454*s))/exp(0.06567483898489704*s) - (0.6818177
      416446249*c&
182          &os(0.07283568782600224*s))/exp(0.017262015588746404*s) + (2.2363641326590105*sin(0.03959716051207
      145&
183          &4*s))/exp(0.06567483898489704*s) - (6.952804148086195*sin(0.07283568782600224*s))/exp(0.017262015
      588746404*s))
184     corr_ij(9,4)=((-2.2363641326590127*cos(0.039597160512071454*s))/exp(0.06567483898489704*s) + (6.9528041
      48086194*co&
185          &s(0.07283568782600224*s))/exp(0.017262015588746404*s) - (1.4494534432272486*sin(0.039597160512071
      454&
186          &*s))/exp(0.06567483898489704*s) - (0.6818177416446249*sin(0.07283568782600224*s))/exp(0.017262015
      588746404*s))
187     corr_ij(9,3)=0
188     corr_ij(9,2)=0
189     corr_ij(9,1)=0
190     corr_ij(8,10)=0
191 
192     corr_ij(8,9)=0
193     corr_ij(8,8)=(9.135647293470983/exp(0.05076718239027029*s) + 2.2233889637758932/exp(0.016285467000648854
      4*s))
194     corr_ij(8,7)=0
195     corr_ij(8,6)=0
196     corr_ij(8,5)=0
197     corr_ij(8,4)=0
198     corr_ij(8,3)=(-5.938566084855411/exp(0.05076718239027029*s) + 11.97129741027622/exp(0.016285467000648854
```

```
4*s))
  corr_ij(8,2)=0
  corr_ij(8,1)=0
  corr_ij(7,10)=0

  corr_ij(7,9)=0
  corr_ij(7,8)=0
  corr_ij(7,7)=((11.518026982819887*cos(0.029341409726871926*s))/exp(0.04435489221745234*s) + (0.05351107
793747755*c&
      &os(0.11425932545092894*s))/exp(0.019700737327669783*s) - (0.14054811601869432*sin(0.0293414097268
719&
      &26*s))/exp(0.04435489221745234*s) + (0.14054811601869702*sin(0.11425932545092894*s))/exp(0.019700
737327669783*s))
  corr_ij(7,6)=((0.14054811601869532*cos(0.029341409726871926*s))/exp(0.04435489221745234*s) - (0.1405481
1601869702*&
      &cos(0.11425932545092894*s))/exp(0.019700737327669783*s) + (11.518026982819887*sin(0.0293414097268
719&
      &26*s))/exp(0.04435489221745234*s) + (0.0535110779374777*sin(0.11425932545092894*s))/exp(0.0197007
37327669783*s))
  corr_ij(7,5)=0
  corr_ij(7,4)=0
  corr_ij(7,3)=0
  corr_ij(7,2)=((-0.732907009016115*cos(0.029341409726871926*s))/exp(0.04435489221745234*s) + (2.72884503
1386875*cos&
      &(0.11425932545092894*s))/exp(0.019700737327669783*s) - (2.4717920234033532*sin(0.0293414097268719
26*&
      &s))/exp(0.04435489221745234*s) - (0.24003801347124257*sin(0.11425932545092894*s))/exp(0.019700737
327669783*s))
  corr_ij(7,1)=((2.4717920234033532*cos(0.029341409726871926*s))/exp(0.04435489221745234*s) + (0.24003801
34712426*co&
      &s(0.11425932545092894*s))/exp(0.019700737327669783*s) - (0.7329070090161153*sin(0.029341409726871
926&
      &*s))/exp(0.04435489221745234*s) + (2.728845031386876*sin(0.11425932545092894*s))/exp(0.0197007373
27669783*s))
  corr_ij(6,10)=0

  corr_ij(6,9)=0
  corr_ij(6,8)=0
  corr_ij(6,7)=((-0.1405481160186977*cos(0.029341409726871926*s))/exp(0.04435489221745234*s) + (0.1405481
1601869713*&
      &cos(0.11425932545092894*s))/exp(0.019700737327669783*s) - (11.518026982819885*sin(0.0293414097268
719&
      &26*s))/exp(0.04435489221745234*s) - (0.05351107793747755*sin(0.11425932545092894*s))/exp(0.019700
737327669783*s))
  corr_ij(6,6)=((11.518026982819885*cos(0.029341409726871926*s))/exp(0.04435489221745234*s) + (0.05351107
793747768*c&
      &os(0.11425932545092894*s))/exp(0.019700737327669783*s) - (0.14054811601869832*sin(0.0293414097268
719&
      &26*s))/exp(0.04435489221745234*s) + (0.14054811601869707*sin(0.11425932545092894*s))/exp(0.019700
737327669783*s))
  corr_ij(6,5)=0
  corr_ij(6,4)=0
  corr_ij(6,3)=0
  corr_ij(6,2)=((-2.471792023403353*cos(0.029341409726871926*s))/exp(0.04435489221745234*s) - (0.24003801
34712425*co&
      &s(0.11425932545092894*s))/exp(0.019700737327669783*s) + (0.7329070090161155*sin(0.029341409726871
926&
      &*s))/exp(0.04435489221745234*s) - (2.7288450313868755*sin(0.11425932545092894*s))/exp(0.019700737
327669783*s))
  corr_ij(6,1)=((-0.7329070090161154*cos(0.029341409726871926*s))/exp(0.04435489221745234*s) + (2.7288450
31386876*co&
      &s(0.11425932545092894*s))/exp(0.019700737327669783*s) - (2.4717920234033524*sin(0.029341409726871
926&
      &*s))/exp(0.04435489221745234*s) - (0.24003801347124343*sin(0.11425932545092894*s))/exp(0.01970073
7327669783*s))
  corr_ij(5,10)=((0.5794534449999711*cos(0.039597160512071454*s))/exp(0.06567483898489704*s) + (4.1369865
70427212*cos&
      &(0.07283568782600224*s))/exp(0.017262015588746404*s) - (1.0360597341248128*sin(0.0395971605120714
54*&
      &s))/exp(0.06567483898489704*s) + (3.167330918996692*sin(0.07283568782600224*s))/exp(0.01726201558
8746404*s))
  corr_ij(5,9)=((1.0360597341248134*cos(0.039597160512071454*s))/exp(0.06567483898489704*s) - (3.16733091
89966856*co&
      &s(0.07283568782600224*s))/exp(0.017262015588746404*s) + (0.5794534449999746*sin(0.039597160512071
454&
      &*s))/exp(0.06567483898489704*s) + (4.1369865704272115*sin(0.07283568782600224*s))/exp(0.017262015
588746404*s))
  corr_ij(5,8)=0
  corr_ij(5,7)=0
  corr_ij(5,6)=0
  corr_ij(5,5)=((-0.37825091063447547*cos(0.039597160512071454*s))/exp(0.06567483898489704*s) + (30.09469
0926061638*&
      &cos(0.07283568782600224*s))/exp(0.017262015588746404*s) + (0.16085380971100194*sin(0.039597160512
071&
      &454*s))/exp(0.06567483898489704*s) - (0.1608538097109995*sin(0.07283568782600224*s))/exp(0.017262
015588746404*s))
  corr_ij(5,4)=((-0.16085380971100238*cos(0.039597160512071454*s))/exp(0.06567483898489704*s) + (0.160853
```

```
80971100127&
       &*cos(0.07283568782600224*s))/exp(0.017262015588746404*s) - (0.37825091063447586*sin(0.03959716051
207&
       &1454*s))/exp(0.06567483898489704*s) + (30.09469092606163*sin(0.07283568782600224*s))/exp(0.017262
015588746404*s))
  corr_ij(5,3)=0
  corr_ij(5,2)=0
  corr_ij(5,1)=0
  corr_ij(4,10)=((-1.0360597341248106*cos(0.039597160512071454*s))/exp(0.06567483898489704*s) + (3.167330
918996689*co&
       &s(0.07283568782600224*s))/exp(0.017262015588746404*s) - (0.5794534449999716*sin(0.039597160512071
454&
       &*s))/exp(0.06567483898489704*s) - (4.1369865704272115*sin(0.07283568782600224*s))/exp(0.017262015
588746404*s))
  corr_ij(4,9)=((0.5794534449999711*cos(0.039597160512071454*s))/exp(0.06567483898489704*s) + (4.13698657
04272115*co&
       &s(0.07283568782600224*s))/exp(0.017262015588746404*s) - (1.0360597341248114*sin(0.039597160512071
454&
       &*s))/exp(0.06567483898489704*s) + (3.1673309189966843*sin(0.07283568782600224*s))/exp(0.017262015
588746404*s))
  corr_ij(4,8)=0
  corr_ij(4,7)=0
  corr_ij(4,6)=0
  corr_ij(4,5)=((0.16085380971100194*cos(0.039597160512071454*s))/exp(0.06567483898489704*s) - (0.1608538
0971100371*&
       &cos(0.07283568782600224*s))/exp(0.017262015588746404*s) + (0.37825091063447497*sin(0.039597160512
071&
       &454*s))/exp(0.06567483898489704*s) - (30.094690926061617*sin(0.07283568782600224*s))/exp(0.017262
015588746404*s))
  corr_ij(4,4)=((-0.37825091063447536*cos(0.039597160512071454*s))/exp(0.06567483898489704*s) + (30.09469
0926061617*&
       &cos(0.07283568782600224*s))/exp(0.017262015588746404*s) + (0.16085380971100172*sin(0.039597160512
071&
       &454*s))/exp(0.06567483898489704*s) - (0.16085380971100616*sin(0.07283568782600224*s))/exp(0.01726
2015588746404*s))
  corr_ij(4,3)=0
  corr_ij(4,2)=0
  corr_ij(4,1)=0
  corr_ij(3,10)=0

  corr_ij(3,9)=0
  corr_ij(3,8)=(0.24013456462471527/exp(0.05076718239027029*s) + 5.792596760796093/exp(0.0162854670006488
54*s))
  corr_ij(3,7)=0
  corr_ij(3,6)=0
  corr_ij(3,5)=0
  corr_ij(3,4)=0
  corr_ij(3,3)=(-0.15609785880208227/exp(0.05076718239027029*s) + 31.18882918422289/exp(0.016285467000648
854*s))
  corr_ij(3,2)=0
  corr_ij(3,1)=0
  corr_ij(2,10)=0

  corr_ij(2,9)=0
  corr_ij(2,8)=0
  corr_ij(2,7)=((1.6172201305728584*cos(0.029341409726871926*s))/exp(0.04435489221745234*s) + (0.37871789
179790255*c&
       &os(0.11425932545092894*s))/exp(0.019700737327669783*s) + (1.2889451151208258*sin(0.02934140972687
192&
       &6*s))/exp(0.04435489221745234*s) + (1.4228849217537705*sin(0.11425932545092894*s))/exp(0.01970073
7327669783*s))
  corr_ij(2,6)=((-1.2889451151208255*cos(0.029341409726871926*s))/exp(0.04435489221745234*s) - (1.4228849
217537702*c&
       &os(0.11425932545092894*s))/exp(0.019700737327669783*s) + (1.6172201305728586*sin(0.02934140972687
192&
       &6*s))/exp(0.04435489221745234*s) + (0.3787178917979035*sin(0.11425932545092894*s))/exp(0.01970073
7327669783*s))
  corr_ij(2,5)=0
  corr_ij(2,4)=0
  corr_ij(2,3)=0
  corr_ij(2,2)=((0.1789135645266575*cos(0.029341409726871926*s))/exp(0.04435489221745234*s) + (26.8170244
57844113*co&
       &s(0.11425932545092894*s))/exp(0.019700737327669783*s) - (0.4268927977731004*sin(0.029341409726871
926&
       &*s))/exp(0.04435489221745234*s) + (0.4268927977730982*sin(0.11425932545092894*s))/exp(0.019700737
327669783*s))
  corr_ij(2,1)=((0.4268927977731007*cos(0.029341409726871926*s))/exp(0.04435489221745234*s) - (0.42689279
777309963*c&
       &os(0.11425932545092894*s))/exp(0.019700737327669783*s) + (0.17891356452665746*sin(0.0293414097268
719&
       &26*s))/exp(0.04435489221745234*s) + (26.81702445784412*sin(0.11425932545092894*s))/exp(0.01970073
7327669783*s))
  corr_ij(1,10)=0

  corr_ij(1,9)=0
  corr_ij(1,8)=0
  corr_ij(1,7)=((1.288945115120824*cos(0.029341409726871926*s))/exp(0.04435489221745234*s) + (1.422884921
```

```
7537711*cos&
      &(0.11425932545092894*s))/exp(0.019700737327669783*s) - (1.617220130572856*sin(0.02934140972687192
6*s&
      &))/exp(0.04435489221745234*s) - (0.3787178917979028*sin(0.11425932545092894*s))/exp(0.01970073732
7669783*s))
  corr_ij(1,6)=((1.6172201305728564*cos(0.029341409726871926*s))/exp(0.04435489221745234*s) + (0.37871789
179790377*c&
      &os(0.11425932545092894*s))/exp(0.019700737327669783*s) + (1.2889451151208242*sin(0.02934140972687
192&
      &6*s))/exp(0.04435489221745234*s) + (1.4228849217537711*sin(0.11425932545092894*s))/exp(0.01970073
7327669783*s))
  corr_ij(1,5)=0
  corr_ij(1,4)=0
  corr_ij(1,3)=0
  corr_ij(1,2)=((-0.4268927977731002*cos(0.029341409726871926*s))/exp(0.04435489221745234*s) + (0.4268927
977730981*c&
      &os(0.11425932545092894*s))/exp(0.019700737327669783*s) - (0.1789135645266573*sin(0.02934140972687
192&
      &6*s))/exp(0.04435489221745234*s) - (26.81702445784412*sin(0.11425932545092894*s))/exp(0.019700737
327669783*s))
  corr_ij(1,1)=((0.1789135645266574*cos(0.029341409726871926*s))/exp(0.04435489221745234*s) + (26.8170244
57844113*co&
      &s(0.11425932545092894*s))/exp(0.019700737327669783*s) - (0.42689279777310024*sin(0.02934140972687
192&
      &6*s))/exp(0.04435489221745234*s) + (0.4268927977730997*sin(0.11425932545092894*s))/exp(0.01970073
7327669783*s))

  corr_ij=q_au**2*corr_ij

```

7.3.2.2 subroutine corrmod::corrcomp_from_fit (real(kind=8), intent(in) *s*) [private]

Subroutine to compute the correlation of the unresolved variables $\langle Y \otimes Y^s \rangle$ at time s from the exponential representation.

Parameters

s	time s at which the correlation is computed

Definition at line 399 of file corrmod.f90.

```
399   REAL(KIND=8), INTENT(IN) :: s
400   REAL(KIND=8) :: y
401   INTEGER :: i,j
402
403   corr_ij=0.d0
404   DO i=1,n_unres
405      DO j=1,n_unres
406         corr_ij(i,j)=fs(s,ya(i,j,:))
407      END DO
408   END DO
```

7.3.2.3 subroutine corrmod::corrcomp_from_spline (real(kind=8), intent(in) *s*) [private]

Subroutine to compute the correlation of the unresolved variables $\langle Y \otimes Y^s \rangle$ at time s from the spline representation.

Parameters

s	time s at which the correlation is computed

Definition at line 333 of file corrmod.f90.

```
333   REAL(KIND=8), INTENT(IN) :: s
```

```
   REAL(KIND=8) :: y
   INTEGER :: i,j
   corr_ij=0.d0
   DO i=1,n_unres
      DO j=1,n_unres
         CALL splint(xa,ya(i,j,:),y2(i,j,:),nspl,s,y)
         corr_ij(i,j)=y
      END DO
   END DO
```

7.3.2.4 real(kind=8) function corrmod::fs (real(kind=8), intent(in) *s*, real(kind=8), dimension(5), intent(in) *p*) [private]

Exponential fit function.

Parameters

s	time s at which the function is evaluated
p	vector holding the coefficients of the fit function

Definition at line 388 of file corrmod.f90.

```
388    REAL(KIND=8), INTENT(IN) :: s
389    REAL(KIND=8), DIMENSION(5), INTENT(IN) :: p
390    REAL(KIND=8) :: fs
391    fs=p(1)*exp(-s/p(2))*cos(p(3)*s+p(4))
392    RETURN
```

7.3.2.5 subroutine, public corrmod::init_corr ( )

Subroutine to initialise the computation of the correlation.

Definition at line 46 of file corrmod.f90.

```
46     INTEGER :: allocstat,i,j,k,nf
47     REAL(KIND=8), DIMENSION(5) :: dumb
48     LOGICAL :: ex
49
50     ! Selection of the loading mode
51     SELECT CASE (load_mode)
52     CASE ('defi')
53        corrcomp => corrcomp_from_def
54     CASE ('spli')
55        INQUIRE(file='corrspline.def',exist=ex)
56        IF (.not.ex) stop "*** File corrspline.def not found ! ***"
57        OPEN(20,file='corrspline.def',status='old')
58        READ(20,*) nf,nspl
59        IF (nf /= n_unres) stop "*** Dimension in files corrspline.def and sf.nml do not correspond ! ***"
60        ALLOCATE(xa(nspl), ya(n_unres,n_unres,nspl), y2(n_unres,n_unres,nspl),
   stat=allocstat)
61        IF (allocstat /= 0) stop "*** Not enough memory ! ***"
62        READ(20,*) xa
63        maxint=xa(nspl)/2
64        DO k=1,n_unres*n_unres
65           READ(20,*) i,j
66           READ(20,*) ya(i,j,:)
67           READ(20,*) y2(i,j,:)
68        ENDDO
69        CLOSE(20)
70        corrcomp => corrcomp_from_spline
71        klo=1
72        khi=nspl
73     CASE ('expo')
74        INQUIRE(file='correxpo.def',exist=ex)
75        IF (.not.ex) stop "*** File correxpo.def not found ! ***"
76        OPEN(20,file='correxpo.def',status='old')
```

```
      READ(20,*) nf,maxint
      IF (nf /= n_unres) stop "*** Dimension in files correxpo.def and sf.nml do not correspond ! ***"
      ALLOCATE(ya(n_unres,n_unres,5), stat=allocstat)
      IF (allocstat /= 0) stop "*** Not enough memory ! ***"
      DO k=1,n_unres*n_unres
         READ(20,*) i,j,dumb
         ya(i,j,:)=dumb
      ENDDO
      CLOSE(20)
      corrcomp => corrcomp_from_fit
   CASE DEFAULT
      stop '*** LOAD_MODE variable not properly defined in corrmod.nml ***'
   END SELECT

   ALLOCATE(mean(n_unres),mean_full(0:ndim), stat=allocstat)
   IF (allocstat /= 0) stop "*** Not enough memory ! ***"

   ALLOCATE(inv_corr_i(n_unres,n_unres), stat=allocstat)
   IF (allocstat /= 0) stop "*** Not enough memory ! ***"

   ALLOCATE(corr_i(n_unres,n_unres), stat=allocstat)
   IF (allocstat /= 0) stop "*** Not enough memory ! ***"

    ALLOCATE(corr_ij(n_unres,n_unres), stat=allocstat)
    IF (allocstat /= 0) stop "*** Not enough memory ! ***"

    ALLOCATE(corr_i_full(ndim,ndim), stat=allocstat)
    IF (allocstat /= 0) stop "*** Not enough memory ! ***"

    ALLOCATE(inv_corr_i_full(ndim,ndim), stat=allocstat)
    IF (allocstat /= 0) stop "*** Not enough memory ! ***"

    corr_ij=0.d0

    CALL corrcomp(0.d0)
    corr_i=corr_ij
    inv_corr_i=invmat(corr_i)

    corr_i_full=0.d0
    DO i=1,n_unres
       DO j=1,n_unres
          corr_i_full(ind(i),ind(j))=corr_i(i,j)
       ENDDO
    ENDDO

    inv_corr_i_full=0.d0
    DO i=1,n_unres
       DO j=1,n_unres
          inv_corr_i_full(ind(i),ind(j))=inv_corr_i(i,j)
       ENDDO
    ENDDO

    mean=0.d0
    INQUIRE(file='mean.def',exist=ex)
    IF (ex) THEN
       OPEN(20,file='mean.def',status='old')
       READ(20,*) mean
       CLOSE(20)
    ENDIF

    mean_full=0.d0
    DO i=1,n_unres
       mean_full(ind(i))=mean(i)
    ENDDO

```

7.3.2.6 subroutine corrmod::splint (real(kind=8), dimension(n), intent(in) *xa*, real(kind=8), dimension(n), intent(in) *ya*, real(kind=8), dimension(n), intent(in) *y2a*, integer, intent(in) *n*, real(kind=8), intent(in) *x*, real(kind=8), intent(out) *y*) [private]

Routine to compute the spline representation parameters.

Definition at line 347 of file corrmod.f90.

```
    INTEGER, INTENT(IN) :: n
    REAL(KIND=8), INTENT(IN), DIMENSION(n) :: xa,y2a,ya
    REAL(KIND=8), INTENT(IN) :: x
    REAL(KIND=8), INTENT(OUT) :: y
```

```
   INTEGER :: k
   REAL(KIND=8) :: a,b,h
   if ((khi-klo.gt.1).or.(xa(klo).gt.x).or.(xa(khi).lt.x)) then
      if ((khi-klo.eq.1).and.(xa(klo).lt.x)) then
         khi=klo
         DO WHILE (xa(khi).lt.x)
            khi=khi+1
         END DO
         klo=khi-1
      else
         khi=n
         klo=1
         DO WHILE (khi-klo.gt.1)
            k=(khi+klo)/2
            if(xa(k).gt.x)then
               khi=k
            else
               klo=k
            endif
         END DO
      end if
   !    print*, "search",x,khi-klo,xa(klo),xa(khi)
   ! else
   !    print*, "ok",x,khi-klo,xa(klo),xa(khi)
   endif
   h=xa(khi)-xa(klo)
   if (h.eq.0.) stop 'bad xa input in splint'
   a=(xa(khi)-x)/h
   b=(x-xa(klo))/h
   y=a*ya(klo)+b*ya(khi)+((a**3-a)*y2a(klo)+(b**3-b)*y2a(khi))*(h**2)/6.
   return
```

7.3.3 Variable Documentation

7.3.3.1 real(kind=8), dimension(:,:), allocatable, public corrmod::corr_i

Covariance matrix of the unresolved variables (reduced version)

Definition at line 30 of file corrmod.f90.

```
30   REAL(KIND=8), DIMENSION(:,:), ALLOCATABLE, PUBLIC :: corr_i !< Covariance matrix of the unresolved
       variables (reduced version)
```

7.3.3.2 real(kind=8), dimension(:,:), allocatable, public corrmod::corr_i_full

Covariance matrix of the unresolved variables (full version)

Definition at line 28 of file corrmod.f90.

```
28   REAL(KIND=8), DIMENSION(:,:), ALLOCATABLE, PUBLIC :: corr_i_full !< Covariance matrix of the unresolved
       variables (full version)
```

7.3.3.3 real(kind=8), dimension(:,:), allocatable, public corrmod::corr_ij

Matrix holding the correlation matrix at a given time.

Definition at line 32 of file corrmod.f90.

```
32   REAL(KIND=8), DIMENSION(:,:), ALLOCATABLE, PUBLIC :: corr_ij !< Matrix holding the correlation matrix at
       a given time
```

7.3.3.4 procedure(corrcomp_from_spline), pointer, public corrmod::corrcomp

Pointer to the correlation computation routine.

Definition at line 41 of file corrmod.f90.

```
PROCEDURE(corrcomp_from_spline), POINTER, PUBLIC :: corrcomp
```

7.3.3.5 real(kind=8), dimension(:,:), allocatable, public corrmod::inv_corr_i

Inverse of the covariance matrix of the unresolved variables (reduced version)

Definition at line 31 of file corrmod.f90.

```
REAL(KIND=8), DIMENSION(:,:), ALLOCATABLE, PUBLIC :: inv_corr_i !< Inverse of the covariance matrix of
  the unresolved variables (reduced version)
```

7.3.3.6 real(kind=8), dimension(:,:), allocatable, public corrmod::inv_corr_i_full

Inverse of the covariance matrix of the unresolved variables (full version)

Definition at line 29 of file corrmod.f90.

```
REAL(KIND=8), DIMENSION(:,:), ALLOCATABLE, PUBLIC :: inv_corr_i_full !< Inverse of the covariance matrix
  of the unresolved variables (full version)
```

7.3.3.7 integer corrmod::khi [private]

Definition at line 38 of file corrmod.f90.

7.3.3.8 integer corrmod::klo [private]

Definition at line 38 of file corrmod.f90.

7.3.3.9 real(kind=8), dimension(:), allocatable, public corrmod::mean

Vector holding the mean of the unresolved dynamics (reduced version)

Definition at line 26 of file corrmod.f90.

```
REAL(KIND=8), DIMENSION(:), ALLOCATABLE, PUBLIC :: mean !< Vector holding the mean of the unresolved
  dynamics (reduced version)
```

7.3.3.10 real(kind=8), dimension(:), allocatable, public corrmod::mean_full

Vector holding the mean of the unresolved dynamics (full version)

Definition at line 27 of file corrmod.f90.

```
27    REAL(KIND=8), DIMENSION(:), ALLOCATABLE, PUBLIC :: mean_full !< Vector holding the mean of the unresolved
        dynamics (full version)
```

7.3.3.11 integer corrmod::nspl [private]

Integers needed by the spline representation of the correlation.

Definition at line 38 of file corrmod.f90.

```
38    INTEGER :: nspl,klo,khi
```

7.3.3.12 real(kind=8), dimension(:), allocatable corrmod::xa [private]

Vector holding coefficient of the spline and exponential correlation representation.

Definition at line 35 of file corrmod.f90.

```
35    REAL(KIND=8), DIMENSION(:), ALLOCATABLE :: xa !< Vector holding coefficient of the spline and exponential
        correlation representation
```

7.3.3.13 real(kind=8), dimension(:,:,:), allocatable corrmod::y2 [private]

Vector holding coefficient of the spline and exponential correlation representation.

Definition at line 33 of file corrmod.f90.

```
33    REAL(KIND=8), DIMENSION(:,:,:), ALLOCATABLE :: y2 !< Vector holding coefficient of the spline and
        exponential correlation representation
```

7.3.3.14 real(kind=8), dimension(:,:,:), allocatable corrmod::ya [private]

Vector holding coefficient of the spline and exponential correlation representation.

Definition at line 34 of file corrmod.f90.

```
34    REAL(KIND=8), DIMENSION(:,:,:), ALLOCATABLE :: ya !< Vector holding coefficient of the spline and
        exponential correlation representation
```

7.4 dec_tensor Module Reference

The resolved-unresolved components decomposition of the tensor.

Functions/Subroutines

- subroutine suppress_and (t, cst, v1, v2)

  Subroutine to suppress from the tensor t_{ijk} components satisfying SF(j)=v1 and SF(k)=v2.
- subroutine suppress_or (t, cst, v1, v2)

  Subroutine to suppress from the tensor t_{ijk} components satisfying SF(j)=v1 or SF(k)=v2.
- subroutine reorder (t, cst, v)

  Subroutine to reorder the tensor t_{ijk} components : if SF(j)=v then it return t_{ikj}.
- subroutine init_sub_tensor (t, cst, v)

  Subroutine that suppress all the components of a tensor t_{ijk} where if SF(i)=v.
- subroutine, public init_dec_tensor

  Subroutine that initialize and compute the decomposed tensors.

Variables

- type(coolist), dimension(:), allocatable, public ff_tensor

  Tensor holding the part of the unresolved tensor involving only unresolved variables.
- type(coolist), dimension(:), allocatable, public sf_tensor

  Tensor holding the part of the resolved tensor involving unresolved variables.
- type(coolist), dimension(:), allocatable, public ss_tensor

  Tensor holding the part of the resolved tensor involving only resolved variables.
- type(coolist), dimension(:), allocatable, public fs_tensor

  Tensor holding the part of the unresolved tensor involving resolved variables.
- type(coolist), dimension(:), allocatable, public hx

  Tensor holding the constant part of the resolved tendencies.
- type(coolist), dimension(:), allocatable, public lxx

  Tensor holding the linear part of the resolved tendencies involving the resolved variables.
- type(coolist), dimension(:), allocatable, public lxy

  Tensor holding the linear part of the resolved tendencies involving the unresolved variables.
- type(coolist), dimension(:), allocatable, public bxxx

  Tensor holding the quadratic part of the resolved tendencies involving resolved variables.
- type(coolist), dimension(:), allocatable, public bxxy

  Tensor holding the quadratic part of the resolved tendencies involving both resolved and unresolved variables.
- type(coolist), dimension(:), allocatable, public bxyy

  Tensor holding the quadratic part of the resolved tendencies involving unresolved variables.
- type(coolist), dimension(:), allocatable, public hy

  Tensor holding the constant part of the unresolved tendencies.
- type(coolist), dimension(:), allocatable, public lyx

  Tensor holding the linear part of the unresolved tendencies involving the resolved variables.
- type(coolist), dimension(:), allocatable, public lyy

  Tensor holding the linear part of the unresolved tendencies involving the unresolved variables.
- type(coolist), dimension(:), allocatable, public byxx

  Tensor holding the quadratic part of the unresolved tendencies involving resolved variables.
- type(coolist), dimension(:), allocatable, public byxy

  Tensor holding the quadratic part of the unresolved tendencies involving both resolved and unresolved variables.

- type(coolist), dimension(:), allocatable, public byyy

  Tensor holding the quadratic part of the unresolved tendencies involving unresolved variables.
- type(coolist), dimension(:), allocatable, public ss_tl_tensor

  Tensor of the tangent linear model tendencies of the resolved component alone.
- type(coolist), dimension(:), allocatable dumb

  Dumb coolist to make the computations.

7.4.1 Detailed Description

The resolved-unresolved components decomposition of the tensor.

Copyright

2017 Jonathan Demaeyer. See LICENSE.txt for license information.

Remarks

7.4.2 Function/Subroutine Documentation

7.4.2.1 subroutine, public dec_tensor::init_dec_tensor ( )

Subroutine that initialize and compute the decomposed tensors.

Definition at line 195 of file dec_tensor.f90.

```
195   USE params, only:ndim
196   USE aotensor_def, only:aotensor
197   USE sf_def, only: load_sf
198   USE tensor, only:copy_tensor,add_to_tensor, scal_mul_coo
199   USE tl_ad_tensor, only: init_tltensor,tltensor
200   USE stoch_params, only: init_stoch_params,mode, tdelta,eps_pert
201   INTEGER :: allocstat
202
203   CALL init_stoch_params
204
205   CALL init_tltensor  ! and tl tensor
206
207   CALL load_sf  ! Load the resolved-unresolved decomposition
208
209   ! Allocating the returned arrays
210
211   ALLOCATE(ff_tensor(ndim),fs_tensor(ndim),sf_tensor(ndim),ss_tensor( ndim), stat=allocstat)
212   IF (allocstat /= 0) stop "*** Not enough memory ! ***"
213
214   ALLOCATE(ss_tl_tensor(ndim), stat=allocstat)
215   IF (allocstat /= 0) stop "*** Not enough memory ! ***"
216
217   ALLOCATE(hx(ndim),lxx(ndim),lxy(ndim),bxxx(ndim),bxxy(ndim),bxyy( ndim), stat=allocstat)
218   IF (allocstat /= 0) stop "*** Not enough memory ! ***"
219
220   ALLOCATE(hy(ndim),lyx(ndim),lyy(ndim),byxx(ndim),byxy(ndim),byyy( ndim), stat=allocstat)
221   IF (allocstat /= 0) stop "*** Not enough memory ! ***"
222
223   ! General decomposition
224   ! ff tensor
225   ALLOCATE(dumb(ndim), stat=allocstat)
226   IF (allocstat /= 0) stop "*** Not enough memory ! ***"
```

```

    IF (mode.ne.'qfst') THEN
        CALL copy_tensor(aotensor,dumb) !Copy the tensors
        CALL init_sub_tensor(dumb,0,0)
        CALL suppress_or(dumb,1,0,0) ! Clear entries with resolved variables
        CALL copy_tensor(dumb,ff_tensor)
    ELSE
        CALL copy_tensor(aotensor,dumb) !Copy the tensors
        CALL init_sub_tensor(dumb,0,0)
        CALL suppress_or(dumb,0,0,0) ! Clear entries with resolved variables and linear and constant terms
        CALL copy_tensor(dumb,ff_tensor)
    ENDIF

    allocstat=0
    DEALLOCATE(dumb, stat=allocstat)
    IF (allocstat /= 0)  stop "*** Problem to deallocate ! ***"

    ! fs tensor
    ALLOCATE(dumb(ndim), stat=allocstat)
    IF (allocstat /= 0) stop "*** Not enough memory ! ***"

    IF (mode.ne.'qfst') THEN
        CALL copy_tensor(aotensor,dumb) !Copy the tensors
        CALL init_sub_tensor(dumb,0,0)
        CALL suppress_and(dumb,1,1,1) ! Clear entries with only unresolved variables and constant
        CALL copy_tensor(dumb,fs_tensor)
    ELSE
        CALL copy_tensor(aotensor,dumb) !Copy the tensors
        CALL init_sub_tensor(dumb,0,0)
        CALL suppress_and(dumb,0,1,1) ! Clear entries with only quadratic unresolved variables
        CALL copy_tensor(dumb,fs_tensor)
    ENDIF

    allocstat=0
    DEALLOCATE(dumb, stat=allocstat)
    IF (allocstat /= 0)  stop "*** Problem to deallocate ! ***"

    ! sf tensor
    ALLOCATE(dumb(ndim), stat=allocstat)
    IF (allocstat /= 0) stop "*** Not enough memory ! ***"


    CALL copy_tensor(aotensor,dumb) !Copy the tensors
    CALL init_sub_tensor(dumb,1,1)
    CALL suppress_and(dumb,0,0,0) ! Clear entries with only unresolved variables and constant
    CALL copy_tensor(dumb,sf_tensor)

    allocstat=0
    DEALLOCATE(dumb, stat=allocstat)
    IF (allocstat /= 0)  stop "*** Problem to deallocate ! ***"

    ! ss tensor
    ALLOCATE(dumb(ndim), stat=allocstat)
    IF (allocstat /= 0) stop "*** Not enough memory ! ***"


    CALL copy_tensor(aotensor,dumb) !Copy the tensors
    CALL init_sub_tensor(dumb,1,1)
    CALL suppress_or(dumb,0,1,1) ! Clear entries with only unresolved variables and constant
    CALL copy_tensor(dumb,ss_tensor)

    allocstat=0
    DEALLOCATE(dumb, stat=allocstat)
    IF (allocstat /= 0)  stop "*** Problem to deallocate ! ***"

    ! ss tangent linear tensor

    ALLOCATE(dumb(ndim), stat=allocstat)
    IF (allocstat /= 0) stop "*** Not enough memory ! ***"

    CALL copy_tensor(tltensor,dumb) !Copy the tensors
    CALL init_sub_tensor(dumb,1,1)
    CALL suppress_or(dumb,0,1,1) ! Clear entries with only unresolved variables and constant
    CALL copy_tensor(dumb,ss_tl_tensor)

    allocstat=0
    DEALLOCATE(dumb, stat=allocstat)
    IF (allocstat /= 0)  stop "*** Problem to deallocate ! ***"

    ! Multiply the aotensor part that need to be by the perturbation and time
    ! separation parameter

    ALLOCATE(dumb(ndim), stat=allocstat)
    IF (allocstat /= 0) stop "*** Not enough memory ! ***"

    CALL copy_tensor(ss_tensor,dumb)
    CALL scal_mul_coo(1.d0/tdelta**2,ff_tensor)
```

```
    CALL scal_mul_coo(eps_pert/tdelta,fs_tensor)
    CALL add_to_tensor(ff_tensor,dumb)
    CALL add_to_tensor(fs_tensor,dumb)
    CALL scal_mul_coo(eps_pert/tdelta,sf_tensor)
    CALL add_to_tensor(sf_tensor,dumb)

    allocstat=0
    DEALLOCATE(aotensor, stat=allocstat)
    IF (allocstat /= 0)  stop "*** Problem to deallocate ! ***"

    ALLOCATE(aotensor(ndim), stat=allocstat)
    IF (allocstat /= 0) stop "*** Not enough memory ! ***"

    CALL copy_tensor(dumb,aotensor)

    allocstat=0
    DEALLOCATE(dumb, stat=allocstat)
    IF (allocstat /= 0)  stop "*** Problem to deallocate ! ***"

    ! MTV decomposition
    ! Unresolved tensors

    ! Hy tensor
    ALLOCATE(dumb(ndim), stat=allocstat)
    IF (allocstat /= 0) stop "*** Not enough memory ! ***"

    CALL copy_tensor(aotensor,dumb) !Copy the tensors
    CALL init_sub_tensor(dumb,0,0)
    CALL suppress_or(dumb,0,1,1) ! Clear entries with unresolved variables
    CALL suppress_or(dumb,1,0,0) ! Suppress linear and nonlinear resolved terms
    CALL copy_tensor(dumb,hy)

    allocstat=0
    DEALLOCATE(dumb, stat=allocstat)
    IF (allocstat /= 0)  stop "*** Problem to deallocate ! ***"

    ! Lyx tensor
    ALLOCATE(dumb(ndim), stat=allocstat)
    IF (allocstat /= 0) stop "*** Not enough memory ! ***"

    CALL copy_tensor(aotensor,dumb) !Copy the tensors
    CALL init_sub_tensor(dumb,0,0)
    CALL suppress_or(dumb,0,1,1) ! Clear entries with unresolved variables
    CALL suppress_and(dumb,1,1,1) ! Clear constant entries
    CALL suppress_and(dumb,1,0,0) ! Clear entries with nonlinear resolved terms
    CALL reorder(dumb,1,0) ! Resolved variables must be the third (k) index
    CALL copy_tensor(dumb,lyx)

    allocstat=0
    DEALLOCATE(dumb, stat=allocstat)
    IF (allocstat /= 0)  stop "*** Problem to deallocate ! ***"

    ! Lyy tensor
    ALLOCATE(dumb(ndim), stat=allocstat)
    IF (allocstat /= 0) stop "*** Not enough memory ! ***"

    CALL copy_tensor(aotensor,dumb) !Copy the tensors
    CALL init_sub_tensor(dumb,0,0)
    CALL suppress_or(dumb,1,0,0) ! Clear entries with resolved variables
    CALL suppress_and(dumb,0,1,1) ! Clear entries with nonlinear unresolved terms
    CALL suppress_and(dumb,0,0,0) ! Clear constant entries
    CALL reorder(dumb,0,1) ! Unresolved variables must be the third (k) index
    CALL copy_tensor(dumb,lyy)

    allocstat=0
    DEALLOCATE(dumb, stat=allocstat)
    IF (allocstat /= 0)  stop "*** Problem to deallocate ! ***"

    ! Byxy tensor
    ALLOCATE(dumb(ndim), stat=allocstat)
    IF (allocstat /= 0) stop "*** Not enough memory ! ***"

    CALL copy_tensor(aotensor,dumb) !Copy the tensors
    CALL init_sub_tensor(dumb,0,0)
    CALL suppress_and(dumb,1,1,1) ! Clear constant or linear terms and nonlinear unresolved only entries
    CALL suppress_and(dumb,0,0,0) ! Clear entries with only resolved variables and constant
    CALL reorder(dumb,0,1) ! Unresolved variables must be the third (k) index
    CALL copy_tensor(dumb,byxy)

    allocstat=0
    DEALLOCATE(dumb, stat=allocstat)
    IF (allocstat /= 0)  stop "*** Problem to deallocate ! ***"

    ! Byyy tensor
    ALLOCATE(dumb(ndim), stat=allocstat)
    IF (allocstat /= 0) stop "*** Not enough memory ! ***"
```

```
    CALL copy_tensor(aotensor,dumb) !Copy the tensors
    CALL init_sub_tensor(dumb,0,0)
    CALL suppress_or(dumb,0,0,0) ! Clear entries with resolved variables and linear and constant terms
    CALL copy_tensor(dumb,byyy)

    allocstat=0
    DEALLOCATE(dumb, stat=allocstat)
    IF (allocstat /= 0)  stop "*** Problem to deallocate ! ***"

    ! Byxx tensor
    ALLOCATE(dumb(ndim), stat=allocstat)
    IF (allocstat /= 0) stop "*** Not enough memory ! ***"

    CALL copy_tensor(aotensor,dumb) !Copy the tensors
    CALL init_sub_tensor(dumb,0,0)
    CALL suppress_or(dumb,1,1,1) ! Clear entries with unresolved variables and linear and constant terms
    CALL copy_tensor(dumb,byxx)

    allocstat=0
    DEALLOCATE(dumb, stat=allocstat)
    IF (allocstat /= 0)  stop "*** Problem to deallocate ! ***"

    ! Resolved tensors

    ! Hx tensor
    ALLOCATE(dumb(ndim), stat=allocstat)
    IF (allocstat /= 0) stop "*** Not enough memory ! ***"

    CALL copy_tensor(aotensor,dumb) !Copy the tensors
    CALL init_sub_tensor(dumb,1,1)
    CALL suppress_or(dumb,1,0,0) ! Clear entries with resolved variables
    CALL suppress_or(dumb,0,1,1) ! Suppress linear and nonlinear unresolved terms
    CALL copy_tensor(dumb,hx)

    allocstat=0
    DEALLOCATE(dumb, stat=allocstat)
    IF (allocstat /= 0)  stop "*** Problem to deallocate ! ***"

    ! Lxy tensor
    ALLOCATE(dumb(ndim), stat=allocstat)
    IF (allocstat /= 0) stop "*** Not enough memory ! ***"

    CALL copy_tensor(aotensor,dumb) !Copy the tensors
    CALL init_sub_tensor(dumb,1,1)
    CALL suppress_or(dumb,1,0,0) ! Clear entries with resolved variables
    CALL suppress_and(dumb,0,0,0) ! Clear constant entries
    CALL suppress_and(dumb,0,1,1) ! Clear entries with nonlinear unresolved terms
    CALL reorder(dumb,0,1) ! Resolved variables must be the third (k) index
    CALL copy_tensor(dumb,lxy)

    allocstat=0
    DEALLOCATE(dumb, stat=allocstat)
    IF (allocstat /= 0)  stop "*** Problem to deallocate ! ***"

    ! Lxx tensor
    ALLOCATE(dumb(ndim), stat=allocstat)
    IF (allocstat /= 0) stop "*** Not enough memory ! ***"

    CALL copy_tensor(aotensor,dumb) !Copy the tensors
    CALL init_sub_tensor(dumb,1,1)
    CALL suppress_or(dumb,0,1,1) ! Clear entries with unresolved variables
    CALL suppress_and(dumb,1,0,0) ! Clear entries with nonlinear resolved terms
    CALL suppress_and(dumb,1,1,1) ! Clear constant entries
    CALL reorder(dumb,1,0) ! Resolved variables must be the third (k) index
    CALL copy_tensor(dumb,lxx)

    allocstat=0
    DEALLOCATE(dumb, stat=allocstat)
    IF (allocstat /= 0)  stop "*** Problem to deallocate ! ***"

    ! Bxxy tensor
    ALLOCATE(dumb(ndim), stat=allocstat)
    IF (allocstat /= 0) stop "*** Not enough memory ! ***"

    CALL copy_tensor(aotensor,dumb) !Copy the tensors
    CALL init_sub_tensor(dumb,1,1)
    CALL suppress_and(dumb,1,1,1) ! Clear constant or linear terms and nonlinear unresolved only entries
    CALL suppress_and(dumb,0,0,0) ! Clear entries with only resolved variables and constant
    CALL reorder(dumb,0,1) ! Unresolved variables must be the third (k) index
    CALL copy_tensor(dumb,bxxy)

    allocstat=0
    DEALLOCATE(dumb, stat=allocstat)
    IF (allocstat /= 0)  stop "*** Problem to deallocate ! ***"

    ! Bxxx tensor
    ALLOCATE(dumb(ndim), stat=allocstat)
```

```
    IF (allocstat /= 0) stop "*** Not enough memory ! ***"

    CALL copy_tensor(aotensor,dumb) !Copy the tensors
    CALL init_sub_tensor(dumb,1,1)
    CALL suppress_or(dumb,1,1,1) ! Clear entries with unresolved variables and linear and constant terms
    CALL copy_tensor(dumb,bxxx)

    allocstat=0
    DEALLOCATE(dumb, stat=allocstat)
    IF (allocstat /= 0)  stop "*** Problem to deallocate ! ***"

    ! Bxyy tensor
    ALLOCATE(dumb(ndim), stat=allocstat)
    IF (allocstat /= 0) stop "*** Not enough memory ! ***"

    CALL copy_tensor(aotensor,dumb) !Copy the tensors
    CALL init_sub_tensor(dumb,1,1)
    CALL suppress_or(dumb,0,0,0) ! Clear entries with resolved variables and linear and constant terms
    CALL copy_tensor(dumb,bxyy)

    allocstat=0
    DEALLOCATE(dumb, stat=allocstat)
    IF (allocstat /= 0)  stop "*** Problem to deallocate ! ***"


    ! ! Droping the unneeded part of the tensors to define them
    ! ! Resolved tensors :
    ! ! ss tensor
    ! CALL suppress_or(ss_tensor,0,1,1) ! Clear entries with unresolved variables
    ! ! sf tensor
    ! CALL suppress_and(sf_tensor,0,0,0) ! Clear entries with only resolved variables and constant
    ! ! Hx tensor
    ! CALL suppress_or(Hx,0,1,1) ! Clear entries with unresolved variables
    ! CALL suppress_or(Hx,1,0,0) ! Suppress linear and nonlinear resolved terms
    ! ! Lxx tensor
    ! CALL suppress_or(Lxx,0,1,1) ! Clear entries with unresolved variables
    ! CALL suppress_and(Lxx,1,0,0) ! Clear entries with nonlinear resolved terms
    ! CALL suppress_and(Lxx,1,1,1) ! Clear constant entries
    ! CALL reorder(Lxx,1,0) ! Resolved variables must be the third (k) index
    ! ! Lxy tensor
    ! CALL suppress_or(Lxy,1,0,0) ! Clear entries with resolved variables
    ! CALL suppress_and(Lxy,0,0,0) ! Clear constant entries
    ! CALL suppress_and(Lxy,0,1,1) ! Clear entries with nonlinear unresolved terms
    ! CALL reorder(Lxy,0,1) ! Unresolved variables must be the third (k) index
    ! ! Bxxy tensor
    ! CALL suppress_and(Bxxy,1,1,1) ! Clear constant or linear terms and nonlinear unresolved only entries
    ! CALL suppress_and(Bxxy,0,0,0) ! Clear entries with only resolved variables and constant
    ! CALL reorder(Bxxy,0,1) ! Unresolved variables must be the third (k) index
    ! ! Bxxx tensor
    ! CALL suppress_or(Bxxx,1,1,1) ! Clear entries with unresolved variables and linear and constant terms
    ! ! Bxyy
    ! CALL suppress_or(Bxyy,0,0,0) ! Clear entries with resolved variables and linear and constant terms

    ! ! Unresolved tensors :

    ! ! Hy tensor
    ! ! Lyy tensor
    ! ! Lyx tensor
    ! ! Byxy tensor
    ! ! Byyy tensor
    ! CALL suppress_or(Byyy,0,0,0) ! Clear entries with resolved variables and linear and constant terms
    ! ! Byxx
    ! CALL suppress_or(Byxx,1,1,1) ! Clear entries with unresolved variables and linear and constant terms
```

7.4.2.2 subroutine dec_tensor::init_sub_tensor (type(coolist), dimension(ndim), intent(inout) *t*, integer, intent(in) *cst*, integer, intent(in) *v*)

Subroutine that suppress all the components of a tensor t_{ijk} where if SF(i)=v.

Parameters

t	tensor over which the routine acts
cst	constant which controls if the 0 index is taken as a unresolved or a resolved one
v	constant of the conditional (0 to suppress resolved, 1 for unresolved)

Definition at line 174 of file dec_tensor.f90.

```
USE params, only:ndim
USE sf_def, only: sf
TYPE(coolist), DIMENSION(ndim), INTENT(INOUT) :: t
INTEGER, INTENT(IN) :: cst,v
INTEGER :: i

sf(0)=cst ! control wether 0 index is considered unresolved or not
DO i=1,ndim
   IF (sf(i)==v) t(i)%nelems=0
ENDDO
```

7.4.2.3 subroutine dec_tensor::reorder (type(coolist), dimension(ndim), intent(inout) *t*, integer, intent(in) *cst*, integer, intent(in) *v*)

Subroutine to reorder the tensor t_{ijk} components : if SF(j)=v then it return t_{ikj}.

Parameters

t	tensor over which the routine acts
cst	constant which controls if the 0 index is taken as a unresolved or a resolved one
v	constant of the conditional (0 to invert resolved, 1 for unresolved)

Definition at line 148 of file dec_tensor.f90.

```
USE params, only:ndim
USE sf_def, only: sf
TYPE(coolist), DIMENSION(ndim), INTENT(INOUT) :: t
INTEGER, INTENT(IN) :: cst,v
INTEGER :: i,n,li,liii

sf(0)=cst ! control wether 0 index is considered unresolved or not
DO i=1,ndim

   n=t(i)%nelems
   DO li=1,n
      IF (sf(t(i)%elems(li)%j)==v) THEN
         liii=t(i)%elems(li)%j
         t(i)%elems(li)%j=t(i)%elems(li)%k
         t(i)%elems(li)%k=liii
      ENDIF
   ENDDO
ENDDO
```

7.4.2.4 subroutine dec_tensor::suppress_and (type(coolist), dimension(ndim), intent(inout) *t*, integer, intent(in) *cst*, integer, intent(in) *v1*, integer, intent(in) *v2*) [private]

Subroutine to suppress from the tensor t_{ijk} components satisfying SF(j)=v1 and SF(k)=v2.

Parameters

t	tensor over which the routine acts
cst	constant which controls if the 0 index is taken as a unresolved or a resolved one
v1	first constant of the conditional (0 to suppress resolved, 1 for unresolved)
v2	second constant of the conditional (0 to suppress resolved, 1 for unresolved)

Definition at line 77 of file dec_tensor.f90.

```
77      USE params, only:ndim
78      USE sf_def, only: sf
79      TYPE(coolist), DIMENSION(ndim), INTENT(INOUT) :: t
80      INTEGER, INTENT(IN) :: cst,v1,v2
81      INTEGER :: i,n,li,liii
82
83      sf(0)=cst ! control wether 0 index is considered unresolved or not
84      DO i=1,ndim
85         n=t(i)%nelems
86         DO li=1,n
87            ! Clear entries with only resolved variables and shift rest of the items one place down.
88            ! Make sure not to skip any entries while shifting!
89
90            DO WHILE ((sf(t(i)%elems(li)%j)==v1).and.(sf(t(i)%elems(li)%k)==v2))
91               !print*, i,li,t(i)%nelems,n
92               DO liii=li+1,n
93                  t(i)%elems(liii-1)%j=t(i)%elems(liii)%j
94                  t(i)%elems(liii-1)%k=t(i)%elems(liii)%k
95                  t(i)%elems(liii-1)%v=t(i)%elems(liii)%v
96               ENDDO
97               t(i)%nelems=t(i)%nelems-1
98               IF (li>t(i)%nelems) exit
99            ENDDO
100            IF (li>t(i)%nelems) exit
101         ENDDO
102      ENDDO
103
```

7.4.2.5 subroutine dec_tensor::suppress_or (type(coolist), dimension(ndim), intent(inout) *t*, integer, intent(in) *cst*, integer, intent(in) *v1*, integer, intent(in) *v2*)

Subroutine to suppress from the tensor t_{ijk} components satisfying SF(j)=v1 or SF(k)=v2.

Parameters

t	tensor over which the routine acts
cst	constant which controls if the 0 index is taken as a unresolved or a resolved one
v1	first constant of the conditional (0 to suppress resolved, 1 for unresolved)
v2	second constant of the conditional (0 to suppress resolved, 1 for unresolved)

Definition at line 113 of file dec_tensor.f90.

```
113      USE params, only:ndim
114      USE sf_def, only: sf
115      TYPE(coolist), DIMENSION(ndim), INTENT(INOUT) :: t
116      INTEGER, INTENT(IN) :: cst,v1,v2
117      INTEGER :: i,n,li,liii
118
119      sf(0)=cst ! control wether 0 index is considered unresolved or not
120      DO i=1,ndim
121         n=t(i)%nelems
122         DO li=1,n
123            ! Clear entries with only resolved variables and shift rest of the items one place down.
124            ! Make sure not to skip any entries while shifting!
125
126            DO WHILE ((sf(t(i)%elems(li)%j)==v1).or.(sf(t(i)%elems(li)%k)==v2))
127               !print*, i,li,t(i)%nelems,n
128               DO liii=li+1,n
129                  t(i)%elems(liii-1)%j=t(i)%elems(liii)%j
130                  t(i)%elems(liii-1)%k=t(i)%elems(liii)%k
131                  t(i)%elems(liii-1)%v=t(i)%elems(liii)%v
132               ENDDO
133               t(i)%nelems=t(i)%nelems-1
134               IF (li>t(i)%nelems) exit
135            ENDDO
136            IF (li>t(i)%nelems) exit
137         ENDDO
138      ENDDO
139
```

7.4.3 Variable Documentation

7.4.3.1 type(coolist), dimension(:), allocatable, public dec_tensor::bxxx

Tensor holding the quadratic part of the resolved tendencies involving resolved variables.

Definition at line 39 of file dec_tensor.f90.

```
39   TYPE(coolist), DIMENSION(:), ALLOCATABLE, PUBLIC :: bxxx !< Tensor holding the quadratic part of
       the resolved tendencies involving resolved variables
```

7.4.3.2 type(coolist), dimension(:), allocatable, public dec_tensor::bxxy

Tensor holding the quadratic part of the resolved tendencies involving both resolved and unresolved variables.

Definition at line 40 of file dec_tensor.f90.

```
40   TYPE(coolist), DIMENSION(:), ALLOCATABLE, PUBLIC :: bxxy !< Tensor holding the quadratic part of
       the resolved tendencies involving both resolved and unresolved variables
```

7.4.3.3 type(coolist), dimension(:), allocatable, public dec_tensor::bxyy

Tensor holding the quadratic part of the resolved tendencies involving unresolved variables.

Definition at line 41 of file dec_tensor.f90.

```
41   TYPE(coolist), DIMENSION(:), ALLOCATABLE, PUBLIC :: bxyy !< Tensor holding the quadratic part of
       the resolved tendencies involving unresolved variables
```

7.4.3.4 type(coolist), dimension(:), allocatable, public dec_tensor::byxx

Tensor holding the quadratic part of the unresolved tendencies involving resolved variables.

Definition at line 46 of file dec_tensor.f90.

```
46   TYPE(coolist), DIMENSION(:), ALLOCATABLE, PUBLIC :: byxx !< Tensor holding the quadratic part of
       the unresolved tendencies involving resolved variables
```

7.4.3.5 type(coolist), dimension(:), allocatable, public dec_tensor::byxy

Tensor holding the quadratic part of the unresolved tendencies involving both resolved and unresolved variables.

Definition at line 47 of file dec_tensor.f90.

```
47   TYPE(coolist), DIMENSION(:), ALLOCATABLE, PUBLIC :: byxy !< Tensor holding the quadratic part of
       the unresolved tendencies involving both resolved and unresolved variables
```

7.4.3.6 type(coolist), dimension(:), allocatable, public dec_tensor::byyy

Tensor holding the quadratic part of the unresolved tendencies involving unresolved variables.

Definition at line 48 of file dec_tensor.f90.

```
48   TYPE(coolist), DIMENSION(:), ALLOCATABLE, PUBLIC :: byyy !< Tensor holding the quadratic part of
       the unresolved tendencies involving unresolved variables
```

7.4.3.7 type(coolist), dimension(:), allocatable dec_tensor::dumb [private]

Dumb coolist to make the computations.

Definition at line 53 of file dec_tensor.f90.

```
53   TYPE(coolist), DIMENSION(:), ALLOCATABLE :: dumb !< Dumb coolist to make the computations
```

7.4.3.8 type(coolist), dimension(:), allocatable, public dec_tensor::ff_tensor

Tensor holding the part of the unresolved tensor involving only unresolved variables.

Definition at line 31 of file dec_tensor.f90.

```
31   TYPE(coolist), DIMENSION(:), ALLOCATABLE, PUBLIC :: ff_tensor !< Tensor holding the part of the
       unresolved tensor involving only unresolved variables
```

7.4.3.9 type(coolist), dimension(:), allocatable, public dec_tensor::fs_tensor

Tensor holding the part of the unresolved tensor involving resolved variables.

Definition at line 34 of file dec_tensor.f90.

```
34   TYPE(coolist), DIMENSION(:), ALLOCATABLE, PUBLIC :: fs_tensor !< Tensor holding the part of the
       unresolved tensor involving resolved variables
```

7.4.3.10 type(coolist), dimension(:), allocatable, public dec_tensor::hx

Tensor holding the constant part of the resolved tendencies.

Definition at line 36 of file dec_tensor.f90.

```
36   TYPE(coolist), DIMENSION(:), ALLOCATABLE, PUBLIC :: hx !< Tensor holding the constant part of the
       resolved tendencies
```

7.4.3.11 type(coolist), dimension(:), allocatable, public dec_tensor::hy

Tensor holding the constant part of the unresolved tendencies.

Definition at line 43 of file dec_tensor.f90.

```
43   TYPE(coolist), DIMENSION(:), ALLOCATABLE, PUBLIC :: hy !< Tensor holding the constant part of the
       unresolved tendencies
```

7.4.3.12 type(coolist), dimension(:), allocatable, public dec_tensor::lxx

Tensor holding the linear part of the resolved tendencies involving the resolved variables.

Definition at line 37 of file dec_tensor.f90.

```
37   TYPE(coolist), DIMENSION(:), ALLOCATABLE, PUBLIC :: lxx !< Tensor holding the linear part of the
       resolved tendencies involving the resolved variables
```

7.4.3.13 type(coolist), dimension(:), allocatable, public dec_tensor::lxy

Tensor holding the linear part of the resolved tendencies involving the unresolved variables.

Definition at line 38 of file dec_tensor.f90.

```
38   TYPE(coolist), DIMENSION(:), ALLOCATABLE, PUBLIC :: lxy !< Tensor holding the linear part of the
       resolved tendencies involving the unresolved variables
```

7.4.3.14 type(coolist), dimension(:), allocatable, public dec_tensor::lyx

Tensor holding the linear part of the unresolved tendencies involving the resolved variables.

Definition at line 44 of file dec_tensor.f90.

```
44   TYPE(coolist), DIMENSION(:), ALLOCATABLE, PUBLIC :: lyx !< Tensor holding the linear part of the
       unresolved tendencies involving the resolved variables
```

7.4.3.15 type(coolist), dimension(:), allocatable, public dec_tensor::lyy

Tensor holding the linear part of the unresolved tendencies involving the unresolved variables.

Definition at line 45 of file dec_tensor.f90.

```
45   TYPE(coolist), DIMENSION(:), ALLOCATABLE, PUBLIC :: lyy !< Tensor holding the linear part of the
       unresolved tendencies involving the unresolved variables
```

7.4.3.16 type(coolist), dimension(:), allocatable, public dec_tensor::sf_tensor

Tensor holding the part of the resolved tensor involving unresolved variables.

Definition at line 32 of file dec_tensor.f90.

```
32   TYPE(coolist), DIMENSION(:), ALLOCATABLE, PUBLIC :: sf_tensor !< Tensor holding the part of the
       resolved tensor involving unresolved variables
```

7.4.3.17 type(coolist), dimension(:), allocatable, public dec_tensor::ss_tensor

Tensor holding the part of the resolved tensor involving only resolved variables.

Definition at line 33 of file dec_tensor.f90.

```
33   TYPE(coolist), DIMENSION(:), ALLOCATABLE, PUBLIC :: ss_tensor !< Tensor holding the part of the
       resolved tensor involving only resolved variables
```

7.4.3.18 type(coolist), dimension(:), allocatable, public dec_tensor::ss_tl_tensor

Tensor of the tangent linear model tendencies of the resolved component alone.

Definition at line 50 of file dec_tensor.f90.

```
50   TYPE(coolist), DIMENSION(:), ALLOCATABLE, PUBLIC :: ss_tl_tensor !< Tensor of the tangent linear
       model tendencies of the resolved component alone
```

7.5 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.

7.5.1 Detailed Description

Module to load the initial condition.

Copyright

2016 Lesley De Cruz, Jonathan Demaeyer & Sebastian Schubert See LICENSE.txt for license information.

7.5.2 Function/Subroutine Documentation

7.5.2.1 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             ic(0)=1.0d0
82             ! except IC(0), nothing has to be done IC has already the right values
83             WRITE(6,*) "*** IC.nml namelist written. Starting with initial condition in IC.nml !***"
84         END SELECT
85      ELSE
86         CALL init_random_seed()
87         CALL random_seed(get=seed)
88         ic=0
89         ic(0)=1.0d0
```

```
        init_type="zero"
        size_of_random_noise=0.d0
        WRITE(6,*) "*** IC.nml namelist written. Starting with 0 as initial condition !***"
     END IF
     OPEN(8, file="IC.nml", status='REPLACE')
     WRITE(8,'(a)') "!------------------------------------------------------------------------------!"
     WRITE(8,'(a)') "! Namelist file :                                                              !"
     WRITE(8,'(a)') "! Initial condition.                                                           !"
     WRITE(8,'(a)') "!------------------------------------------------------------------------------!"
     WRITE(8,*) ""
      WRITE(8,'(a)') "&ICLIST"
      WRITE(8,*) " ! psi variables"
      DO i=1,natm
         WRITE(8,*) " IC("//trim(str(i))//") = ",ic(i),"   ! typ= "&
              &//awavenum(i)%typ//", Nx= "//trim(rstr(awavenum(i)&
              &%Nx,fm))//", Ny= "//trim(rstr(awavenum(i)%Ny,fm))
      END DO
      WRITE(8,*) " ! theta variables"
      DO i=1,natm
         WRITE(8,*) " IC("//trim(str(i+natm))//") = ",ic(i+natm),"   ! typ= "&
              &//awavenum(i)%typ//", Nx= "//trim(rstr(awavenum(i)&
              &%Nx,fm))//", Ny= "//trim(rstr(awavenum(i)%Ny,fm))
      END DO

      WRITE(8,*) " ! A variables"
      DO i=1,noc
         WRITE(8,*) " IC("//trim(str(i+2*natm))//") = ",ic(i+2*natm),"   ! Nx&
              &= "//trim(rstr(owavenum(i)%Nx,fm))//", Ny= "&
              &//trim(rstr(owavenum(i)%Ny,fm))
      END DO
      WRITE(8,*) " ! T variables"
      DO i=1,noc
         WRITE(8,*) " IC("//trim(str(i+noc+2*natm))//") = ",ic(i+2*natm+noc),"   &
              &! Nx= "//trim(rstr(owavenum(i)%Nx,fm))//", Ny= "&
              &//trim(rstr(owavenum(i)%Ny,fm))
      END DO

      WRITE(8,'(a)') "&END"
      WRITE(8,*) ""
      WRITE(8,'(a)') "!------------------------------------------------------------------------------!"
      WRITE(8,'(a)') "! Initialisation type.                                                         !"
      WRITE(8,'(a)') "!------------------------------------------------------------------------------!"
      WRITE(8,'(a)') "! type = 'read': use IC above (will generate a new seed);"
      WRITE(8,'(a)') "!        'rand': random state (will generate a new seed);"
      WRITE(8,'(a)') "!        'zero': zero IC (will generate a new seed);"
      WRITE(8,'(a)') "!        'seed': use the seed below (generate the same IC)"
      WRITE(8,*) ""
      WRITE(8,'(a)') "&RAND"
      WRITE(8,'(a)') "  init_type= '"//init_type//"'"
      WRITE(8,'(a,d15.7)') "  size_of_random_noise = ",size_of_random_noise
      DO i=1,j
         WRITE(8,*) " seed("//trim(str(i))//") = ",seed(i)
      END DO
      WRITE(8,'(a)') "&END"
      WRITE(8,*) ""
      CLOSE(8)

```

7.5.3 Variable Documentation

7.5.3.1 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.
```

7.5.3.2 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
```

7.6 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.

7.6.1 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

Remarks

Generated Fortran90/95 code from inprod_analytic.lua

7.6.2 Function/Subroutine Documentation

7.6.2.1 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)
```

7.6.2.2 real(kind=8) function inprod_analytic::b2 (integer *Pi*, integer *Pj*, integer *Pk*) [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)
```

7.6.2.3 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
```

7.6.2.4 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
```

7.6.2.5 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
```

7.6.2.6 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
```

```
        ALLOCATE(ocean%C(noc,noc,noc), stat=allocstat)
        IF (allocstat /= 0) stop "*** Not enough memory ! ***"
     END IF
  END IF
  ocean%C=0.d0
  val=0.d0

  DO i=1,noc
     DO j=1,noc
        DO k=1,noc
           val = ocean%M(k,k) * ocean%O(i,j,k)
           IF (val /= 0.d0) ocean%C(i,j,k) = val
        END DO
     END DO
  END DO
```

7.6.2.7 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
```

7.6.2.8 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.

```
    INTEGER :: i,j,k
    TYPE(atm_wavenum) :: ti, tj, tk
    REAL(KIND=8) :: val,vb1, vb2, vs1, vs2, vs3, vs4
    INTEGER :: allocstat

    IF (natm == 0 ) THEN
       stop "*** Problem with calculate_g : natm==0 ! ***"
    ELSE
       IF (.NOT. ALLOCATED(atmos%g)) THEN
          ALLOCATE(atmos%g(natm,natm,natm), stat=allocstat)
          IF (allocstat /= 0) stop "*** Not enough memory ! ***"
       END IF
    END IF
    atmos%g=0.d0

    DO i=1,natm
       DO j=1,natm
          DO k=1,natm
             ti = awavenum(i)
             tj = awavenum(j)
             tk = awavenum(k)
             val=0.d0
             IF ((ti%typ == "A") .AND. (tj%typ == "K") .AND. (tk%typ == "L")) THEN
                vb1 = b1(ti%P,tj%P,tk%P)
                vb2 = b2(ti%P,tj%P,tk%P)
                val = -2 * sqrt(2.) / pi * tj%M * delta(tj%M - tk%H) * flambda(ti%P + tj%P + tk%P)
                IF (val /= 0.d0) val = val * (vb1**2 / (vb1**2 - 1) - vb2**2 / (vb2**2 - 1))
             ELSEIF ((ti%typ == "K") .AND. (tj%typ == "K") .AND. (tk%typ == "L")) THEN
                vs1 = s1(tj%P,tk%P,tj%M,tk%H)
                vs2 = s2(tj%P,tk%P,tj%M,tk%H)
                val = vs1 * (delta(ti%M - tk%H - tj%M) * delta(ti%P -&
                     & tk%P + tj%P) - delta(ti%M- tk%H - tj%M) *&
                     & delta(ti%P + tk%P - tj%P) + (delta(tk%H - tj%M&
                     & + ti%M) + delta(tk%H - tj%M - ti%M)) *&
                     & delta(tk%P + tj%P - ti%P)) + vs2 * (delta(ti%M&
                     & - tk%H - tj%M) * delta(ti%P - tk%P - tj%P) +&
                     & (delta(tk%H - tj%M - ti%M) + delta(ti%M + tk%H&
                     & - tj%M)) * (delta(ti%P - tk%P + tj%P) -&
                     & delta(tk%P - tj%P + ti%P)))
             END IF
             val=val*n
             IF (val /= 0.d0) THEN
                atmos%g(i,j,k) = val
                atmos%g(j,k,i) = val
                atmos%g(k,i,j) = val
                atmos%g(i,k,j) = -val
                atmos%g(j,i,k) = -val
                atmos%g(k,j,i) = -val
             ENDIF
          END DO
       END DO
    END DO

    DO i=1,natm
       DO j=i,natm
          DO k=j,natm
             ti = awavenum(i)
             tj = awavenum(j)
             tk = awavenum(k)
             val=0.d0

             IF ((ti%typ == "L") .AND. (tj%typ == "L") .AND. (tk%typ == "L")) THEN
                vs3 = s3(tj%P,tk%P,tj%H,tk%H)
                vs4 = s4(tj%P,tk%P,tj%H,tk%H)
                val = vs3 * ((delta(tk%H - tj%H - ti%H) - delta(tk%H &
                     &- tj%H + ti%H)) * delta(tk%P + tj%P - ti%P) +&
                     & delta(tk%H + tj%H - ti%H) * (delta(tk%P - tj%P&
                     & + ti%P) - delta(tk%P - tj%P - ti%P))) + vs4 *&
                     & ((delta(tk%H + tj%H - ti%H) * delta(tk%P - tj&
                     &%P - ti%P)) + (delta(tk%H - tj%H + ti%H) -&
                     & delta(tk%H - tj%H - ti%H)) * (delta(tk%P - tj&
                     &%P - ti%P) - delta(tk%P - tj%P + ti%P)))
             ENDIF
             val=val*n
             IF (val /= 0.d0) THEN
                atmos%g(i,j,k) = val
                atmos%g(j,k,i) = val
                atmos%g(k,i,j) = val
                atmos%g(i,k,j) = -val
                atmos%g(j,i,k) = -val
                atmos%g(k,j,i) = -val
             ENDIF
          ENDDO
       ENDDO
    ENDDO

```

7.6.2.9 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.

```
INTEGER :: i,j
INTEGER :: allocstat

IF ((.NOT. ALLOCATED(atmos%a)) .OR. (.NOT. ALLOCATED(atmos%s))) THEN
   stop "*** atmos%a and atmos%s must be defined before calling calculate_K ! ***"
END IF

IF (noc == 0 ) THEN
   stop "*** Problem with calculate_K : noc==0 ! ***"
ELSEIF (natm == 0 ) THEN
   stop "*** Problem with calculate_K : natm==0 ! ***"
ELSE
   IF (.NOT. ALLOCATED(ocean%K)) THEN
      ALLOCATE(ocean%K(noc,natm), stat=allocstat)
      IF (allocstat /= 0) stop "*** Not enough memory ! ***"
   END IF
END IF
ocean%K=0.d0

DO i=1,noc
   DO j=1,natm
      ocean%K(i,j) = atmos%s(j,i) * atmos%a(j,j)
   END DO
END DO
```

7.6.2.10 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.

```
INTEGER :: i
TYPE(ocean_wavenum) :: di
INTEGER :: allocstat
IF (noc == 0 ) THEN
   stop "*** Problem with calculate_M : noc==0 ! ***"
ELSE
   IF (.NOT. ALLOCATED(ocean%M)) THEN
      ALLOCATE(ocean%M(noc,noc), stat=allocstat)
      IF (allocstat /= 0) stop "*** Not enough memory ! ***"
   END IF
END IF
ocean%M=0.d0

DO i=1,noc
   di = owavenum(i)
   ocean%M(i,i) = -(n**2) * di%Nx**2 - di%Ny**2
END DO
```

7.6.2.11 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
```

7.6.2.12 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
```

```
                ocean%O(i,j,k) = val
                ocean%O(j,k,i) = val
                ocean%O(k,i,j) = val
                ocean%O(i,k,j) = -val
                ocean%O(j,i,k) = -val
                ocean%O(k,j,i) = -val
             END IF
          END DO
       END DO
    END DO
```

7.6.2.13 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
```

7.6.2.14 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.

```
    INTEGER :: i,j
    INTEGER :: allocstat

    IF (.NOT. ALLOCATED(atmos%s)) THEN
       stop "*** atmos%s must be defined before calling calculate_W ! ***"
    END IF

    IF (noc == 0 ) THEN
       stop "*** Problem with calculate_W : noc==0 ! ***"
    ELSEIF (natm == 0 ) THEN
       stop "*** Problem with calculate_W : natm==0 ! ***"
    ELSE
       IF (.NOT. ALLOCATED(ocean%W)) THEN
          ALLOCATE(ocean%W(noc,natm), stat=allocstat)
          IF (allocstat /= 0) stop "*** Not enough memory ! ***"
       END IF
    END IF
    ocean%W=0.d0

    DO i=1,noc
       DO j=1,natm
          ocean%W(i,j) = atmos%s(j,i)
       END DO
    END DO
```

7.6.2.15 subroutine, public inprod_analytic::deallocate_inprod ( )

Deallocation of the inner products.

Definition at line 722 of file inprod_analytic.f90.

```
    INTEGER :: allocstat

    ! Deallocation of atmospheric inprod
    allocstat=0
    IF (ALLOCATED(atmos%a)) DEALLOCATE(atmos%a, stat=allocstat)
    IF (allocstat /= 0)  stop "*** Problem to deallocate ! ***"

    allocstat=0
    IF (ALLOCATED(atmos%c)) DEALLOCATE(atmos%c, stat=allocstat)
    IF (allocstat /= 0)  stop "*** Problem to deallocate ! ***"

    allocstat=0
    IF (ALLOCATED(atmos%d)) DEALLOCATE(atmos%d, stat=allocstat)
    IF (allocstat /= 0)  stop "*** Problem to deallocate ! ***"

    allocstat=0
    IF (ALLOCATED(atmos%s)) DEALLOCATE(atmos%s, stat=allocstat)
    IF (allocstat /= 0)  stop "*** Problem to deallocate ! ***"

    allocstat=0
    IF (ALLOCATED(atmos%g)) DEALLOCATE(atmos%g, stat=allocstat)
    IF (allocstat /= 0)  stop "*** Problem to deallocate ! ***"

    allocstat=0
    IF (ALLOCATED(atmos%b)) DEALLOCATE(atmos%b, stat=allocstat)
    IF (allocstat /= 0)  stop "*** Problem to deallocate ! ***"

    ! Deallocation of oceanic inprod
    allocstat=0
    IF (ALLOCATED(ocean%K)) DEALLOCATE(ocean%K, stat=allocstat)
    IF (allocstat /= 0)  stop "*** Problem to deallocate ! ***"

    allocstat=0
    IF (ALLOCATED(ocean%M)) DEALLOCATE(ocean%M, stat=allocstat)
    IF (allocstat /= 0)  stop "*** Problem to deallocate ! ***"

    allocstat=0
    IF (ALLOCATED(ocean%N)) DEALLOCATE(ocean%N, stat=allocstat)
```

```
    IF (allocstat /= 0)  stop "*** Problem to deallocate ! ***"

    allocstat=0
    IF (ALLOCATED(ocean%W)) DEALLOCATE(ocean%W, stat=allocstat)
    IF (allocstat /= 0)  stop "*** Problem to deallocate ! ***"

    allocstat=0
    IF (ALLOCATED(ocean%O)) DEALLOCATE(ocean%O, stat=allocstat)
    IF (allocstat /= 0)  stop "*** Problem to deallocate ! ***"

    allocstat=0
    IF (ALLOCATED(ocean%C)) DEALLOCATE(ocean%C, stat=allocstat)
    IF (allocstat /= 0)  stop "*** Problem to deallocate ! ***"
```

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

Integer Dirac delta function.

Definition at line 103 of file inprod_analytic.f90.

```
    INTEGER :: r
    IF (r==0) THEN
       delta = 1.d0
    ELSE
       delta = 0.d0
    ENDIF
```

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

"Odd or even" function

Definition at line 113 of file inprod_analytic.f90.

```
    INTEGER :: r
    IF (mod(r,2)==0) THEN
       flambda = 0.d0
    ELSE
       flambda = 1.d0
    ENDIF
```

7.6.2.18 subroutine, public inprod_analytic::init_inprod ( )

Initialisation of the inner product.

Definition at line 639 of file inprod_analytic.f90.

```
    INTEGER :: i,j
    INTEGER :: allocstat

    ! Definition of the types and wave numbers tables

    ALLOCATE(owavenum(noc),awavenum(natm), stat=allocstat)
    IF (allocstat /= 0) stop "*** Not enough memory ! ***"

    j=0
    DO i=1,nbatm
       IF (ams(i,1)==1) THEN
          awavenum(j+1)%typ='A'
          awavenum(j+2)%typ='K'
          awavenum(j+3)%typ='L'

          awavenum(j+1)%P=ams(i,2)
```

```
         awavenum(j+2)%M=ams(i,1)
         awavenum(j+2)%P=ams(i,2)
         awavenum(j+3)%H=ams(i,1)
         awavenum(j+3)%P=ams(i,2)

         awavenum(j+1)%Ny=REAL(ams(i,2))
         awavenum(j+2)%Nx=REAL(ams(i,1))
         awavenum(j+2)%Ny=REAL(ams(i,2))
         awavenum(j+3)%Nx=REAL(ams(i,1))
         awavenum(j+3)%Ny=REAL(ams(i,2))

         j=j+3
      ELSE
         awavenum(j+1)%typ='K'
         awavenum(j+2)%typ='L'

         awavenum(j+1)%M=ams(i,1)
         awavenum(j+1)%P=ams(i,2)
         awavenum(j+2)%H=ams(i,1)
         awavenum(j+2)%P=ams(i,2)

         awavenum(j+1)%Nx=REAL(ams(i,1))
         awavenum(j+1)%Ny=REAL(ams(i,2))
         awavenum(j+2)%Nx=REAL(ams(i,1))
         awavenum(j+2)%Ny=REAL(ams(i,2))

         j=j+2

      ENDIF
   ENDDO

   DO i=1,noc
      owavenum(i)%H=oms(i,1)
      owavenum(i)%P=oms(i,2)

      owavenum(i)%Nx=oms(i,1)/2.d0
      owavenum(i)%Ny=oms(i,2)

   ENDDO

   ! Computation of the atmospheric inner products tensors

   CALL calculate_a
   CALL calculate_g
   CALL calculate_s
   CALL calculate_b
   CALL calculate_c_atm

   ! Computation of the oceanic inner products tensors

   CALL calculate_m
   CALL calculate_n
   CALL calculate_o
   CALL calculate_c_oc
   CALL calculate_w
   CALL calculate_k

   ! A last atmospheric one that needs ocean%M

   CALL calculate_d
```

7.6.2.19 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.

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

7.6.2.20 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
```

7.6.2.21 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
```

7.6.2.22 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
```

7.6.3 Variable Documentation

7.6.3.1 type(atm_tensors), public inprod_analytic::atmos

Atmospheric tensors.

Definition at line 69 of file inprod_analytic.f90.

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

7.6.3.2 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
```

7.6.3.3 type(ocean_tensors), public inprod_analytic::ocean

Oceanic tensors.

Definition at line 71 of file inprod_analytic.f90.

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

7.6.3.4 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
```

7.7 int_comp Module Reference

Utility module containing the routines to perform the integration of functions.

Functions/Subroutines

- subroutine, public integrate (func, ss)

  Routine to compute integrals of function from O to #maxint.
- subroutine qromb (func, a, b, ss)

  Romberg integration routine.
- subroutine qromo (func, a, b, ss, choose)

  Romberg integration routine on an open interval.
- subroutine polint (xa, ya, n, x, y, dy)

  Polynomial interpolation routine.
- subroutine trapzd (func, a, b, s, n)

  Trapezoidal rule integration routine.
- subroutine midpnt (func, a, b, s, n)

  Midpoint rule integration routine.
- subroutine midexp (funk, aa, bb, s, n)

  Midpoint routine for bb infinite with funk decreasing infinitely rapidly at infinity.

7.7.1 Detailed Description

Utility module containing the routines to perform the integration of functions.

Copyright

2017 Jonathan Demaeyer. See LICENSE.txt for license information.

Remarks

Most are taken from the Numerical Recipes

7.7.2 Function/Subroutine Documentation

7.7.2.1 subroutine, public int_comp::integrate (external *func*, real(kind=8) *ss*)

Routine to compute integrals of function from O to #maxint.

Parameters

func	function to integrate
ss	result of the integration

Definition at line 30 of file int_comp.f90.

```
REAL(KIND=8) :: ss,func,b
EXTERNAL func
b=maxint
! CALL qromo(func,0.D0,1.D0,ss,midexp)
CALL qromb(func,0.d0,b,ss)
```

7.7.2.2 subroutine int_comp::midexp (external *funk*, real(kind=8) *aa*, real(kind=8) *bb*, real(kind=8) *s*, integer *n*) [private]

Midpoint routine for bb infinite with funk decreasing infinitely rapidly at infinity.

Parameters

funk	function to integrate
aa	lower limit of the integral
bb	higher limit of the integral
s	result of the integration
n	higher stage of the rule to be computed

Definition at line 200 of file int_comp.f90.

```
INTEGER :: n
REAL(KIND=8) :: aa,bb,s,funk
EXTERNAL funk
INTEGER :: it,j
REAL(KIND=8) :: ddel,del,sum,tnm,x,func,a,b
func(x)=funk(-log(x))/x
b=exp(-aa)
a=0.
if (n.eq.1) then
   s=(b-a)*func(0.5*(a+b))
else
   it=3**(n-2)
   tnm=it
   del=(b-a)/(3.*tnm)
   ddel=del+del
   x=a+0.5*del
   sum=0.
   do j=1,it
      sum=sum+func(x)
      x=x+ddel
      sum=sum+func(x)
      x=x+del
   end do
   s=(s+(b-a)*sum/tnm)/3.
endif
return
```

7.7.2.3 subroutine int_comp::midpnt (external *func*, real(kind=8) *a*, real(kind=8) *b*, real(kind=8) *s*, integer *n*) [private]

Midpoint rule integration routine.

Parameters

func	function to integrate
a	lower limit of the integral
b	higher limit of the integral
s	result of the integration
n	higher stage of the rule to be computed

Definition at line 167 of file int_comp.f90.

```
167      INTEGER :: n
168      REAL(KIND=8) :: a,b,s,func
169      EXTERNAL func
170      INTEGER :: it,j
171      REAL(KIND=8) :: ddel,del,sum,tnm,x
172      if (n.eq.1) then
173         s=(b-a)*func(0.5*(a+b))
174      else
175         it=3**(n-2)
176         tnm=it
177         del=(b-a)/(3.*tnm)
178         ddel=del+del
179         x=a+0.5*del
180         sum=0.
181         do j=1,it
182            sum=sum+func(x)
183            x=x+ddel
184            sum=sum+func(x)
185            x=x+del
186         end do
187         s=(s+(b-a)*sum/tnm)/3.
188      endif
189      return
```

7.7.2.4 subroutine int_comp::polint (real(kind=8), dimension(n) *xa*, real(kind=8), dimension(n) *ya*, integer *n*, real(kind=8) *x*, real(kind=8) *y*, real(kind=8) *dy*) [private]

Polynomial interpolation routine.

Definition at line 91 of file int_comp.f90.

```
91      INTEGER :: n,nmax
92      REAL(KIND=8) :: dy,x,y,xa(n),ya(n)
93      parameter(nmax=10)
94      INTEGER :: i,m,ns
95      REAL(KIND=8) :: den,dif,dift,ho,hp,w,c(nmax),d(nmax)
96      ns=1
97      dif=abs(x-xa(1))
98      do i=1,n
99         dift=abs(x-xa(i))
100         if (dift.lt.dif) then
101            ns=i
102            dif=dift
103         endif
104         c(i)=ya(i)
105         d(i)=ya(i)
106      end do
107      y=ya(ns)
108      ns=ns-1
109      do m=1,n-1
110         do i=1,n-m
111            ho=xa(i)-x
112            hp=xa(i+m)-x
113            w=c(i+1)-d(i)
114            den=ho-hp
115            if(den.eq.0.)stop 'failure in polint'
116            den=w/den
117            d(i)=hp*den
118            c(i)=ho*den
```

```
119          end do
120          if (2*ns.lt.n-m)then
121             dy=c(ns+1)
122          else
123             dy=d(ns)
124             ns=ns-1
125          endif
126          y=y+dy
127       end do
128       return
```

7.7.2.5 subroutine int_comp::qromb (external *func*, real(kind=8) *a*, real(kind=8) *b*, real(kind=8) *ss*) [private]

Romberg integration routine.

Parameters

func	function to integrate
a	lower limit of the integral
b	higher limit of the integral
func	function to integrate
ss	result of the integration

Definition at line 44 of file int_comp.f90.

```
44      INTEGER :: jmax,jmaxp,k,km
45      REAL(KIND=8) :: a,b,func,ss,eps
46      EXTERNAL func
47      parameter(eps=1.d-6, jmax=20, jmaxp=jmax+1, k=5, km=k-1)
48      INTEGER j
49      REAL(KIND=8) :: dss,h(jmaxp),s(jmaxp)
50      h(1)=1.
51      DO j=1,jmax
52         CALL trapzd(func,a,b,s(j),j)
53         IF (j.ge.k) THEN
54            CALL polint(h(j-km),s(j-km),k,0.d0,ss,dss)
55            IF (abs(dss).le.eps*abs(ss)) RETURN
56         ENDIF
57         s(j+1)=s(j)
58         h(j+1)=0.25*h(j)
59      ENDDO
60      stop 'too many steps in qromb'
```

7.7.2.6 subroutine int_comp::qromo (external *func*, real(kind=8) *a*, real(kind=8) *b*, real(kind=8) *ss*, external *choose*) [private]

Romberg integration routine on an open interval.

Parameters

a	lower limit of the integral
b	higher limit of the integral
func	function to integrate
ss	result of the integration
chose	routine to perform the integration

Definition at line 70 of file int_comp.f90.

```
INTEGER :: jmax,jmaxp,k,km
REAL(KIND=8) :: a,b,func,ss,eps
EXTERNAL func,choose
parameter(eps=1.e-6, jmax=14, jmaxp=jmax+1, k=5, km=k-1)
INTEGER :: j
REAL(KIND=8) :: dss,h(jmaxp),s(jmaxp)
h(1)=1.
DO j=1,jmax
   CALL choose(func,a,b,s(j),j)
   IF (j.ge.k) THEN
      call polint(h(j-km),s(j-km),k,0.d0,ss,dss)
      if (abs(dss).le.eps*abs(ss)) return
   ENDIF
   s(j+1)=s(j)
   h(j+1)=h(j)/9.
ENDDO
stop 'too many steps in qromo'
```

7.7.2.7 subroutine int_comp::trapzd (external *func*, real(kind=8) *a*, real(kind=8) *b*, real(kind=8) *s*, integer *n*) [private]

Trapezoidal rule integration routine.

Parameters

func	function to integrate
a	lower limit of the integral
b	higher limit of the integral
s	result of the integration
n	higher stage of the rule to be computed

Definition at line 138 of file int_comp.f90.

```
INTEGER :: n
REAL(KIND=8) :: a,b,s,func
EXTERNAL func
INTEGER :: it,j
REAL(KIND=8) :: del,sum,tnm,x
if (n.eq.1) then
   s=0.5*(b-a)*(func(a)+func(b))
else
   it=2**(n-2)
   tnm=it
   del=(b-a)/tnm
   x=a+0.5*del
   sum=0.
   do j=1,it
      sum=sum+func(x)
      x=x+del
   end do
   s=0.5*(s+(b-a)*sum/tnm)
endif
return
```

7.8 int_corr Module Reference

Module to compute or load the integrals of the correlation matrices.

Functions/Subroutines

- subroutine, public init_corrint

Subroutine to initialise the integrated matrices and tensors.

- real(kind=8) function func_ij (s)

  Function that returns the component oi and oj of the correlation matrix at time s.
- real(kind=8) function func_ijkl (s)

  Function that returns the component oi,oj,ok and ol of the outer product of the correlation matrix with itself at time s.
- subroutine, public comp_corrint

  Routine that actually compute or load the integrals.

Variables

- integer oi
- integer oj
- integer ok
- integer ol

  Integers that specify the matrices and tensor component considered as a function of time.
- real(kind=8), parameter real_eps = 2.2204460492503131e-16

  Small epsilon constant to determine equality with zero.
- real(kind=8), dimension(:,:), allocatable, public corrint

  Matrix holding the integral of the correlation matrix.
- type(coolist4), dimension(:), allocatable, public corr2int

  Tensor holding the integral of the correlation outer product with itself.

7.8.1 Detailed Description

Module to compute or load the integrals of the correlation matrices.

Copyright

2017 Jonathan Demaeyer. See LICENSE.txt for license information.

Remarks

7.8.2 Function/Subroutine Documentation

7.8.2.1 subroutine, public int_corr::comp_corrint ( )

Routine that actually compute or load the integrals.

Definition at line 75 of file int_corr.f90.

```
IMPLICIT NONE
INTEGER :: i,j,k,l,n,allocstat
REAL(KIND=8) :: ss
LOGICAL :: ex

INQUIRE(file='corrint.def',exist=ex)
SELECT CASE (int_corr_mode)
CASE ('file')
   IF (ex) THEN
      OPEN(30,file='corrint.def',status='old')
      READ(30,*) corrint
      CLOSE(30)
   ELSE
      stop "*** File corrint.def not found ! ***"
   END IF
CASE ('prog')
   DO i = 1,n_unres
      DO j= 1,n_unres
         oi=i
         oj=j
         !   print*, oi,oj
         CALL integrate(func_ij,ss)
         corrint(ind(i),ind(j))=ss
      END DO
   END DO

    OPEN(30,file='corrint.def')
    WRITE(30,*) corrint
    CLOSE(30)
 END SELECT


 INQUIRE(file='corr2int.def',exist=ex)
 SELECT CASE (int_corr_mode)
 CASE ('file')
    IF (ex) THEN
       CALL load_tensor4_from_file("corr2int.def",corr2int)
    ELSE
       stop "*** File corr2int.def not found ! ***"
    END IF
 CASE ('prog')
    DO i = 1,n_unres
       n=0
       DO j= 1,n_unres
          DO k= 1,n_unres
             DO l = 1,n_unres
                oi=i
                oj=j
                ok=k
                ol=l

                CALL integrate(func_ijkl,ss)
                IF (abs(ss)>real_eps) n=n+1
             ENDDO
          ENDDO
       ENDDO
       IF (n/=0) THEN
          ALLOCATE(corr2int(ind(i))%elems(n), stat=allocstat)
          IF (allocstat /= 0) stop "*** Not enough memory ! ***"

          n=0
          DO j= 1,n_unres
             DO k= 1,n_unres
                DO l = 1,n_unres
                   oi=i
                   oj=j
                   ok=k
                   ol=l

                   CALL integrate(func_ijkl,ss)
                   IF (abs(ss)>real_eps) THEN
                      n=n+1
                      corr2int(ind(i))%elems(n)%j=ind(j)
                      corr2int(ind(i))%elems(n)%k=ind(k)
                      corr2int(ind(i))%elems(n)%l=ind(l)
                      corr2int(ind(i))%elems(n)%v=ss
                   END IF
                ENDDO
             ENDDO
          ENDDO
          corr2int(ind(i))%nelems=n
       END IF
    ENDDO

    CALL write_tensor4_to_file("corr2int.def",corr2int)
 CASE DEFAULT
    stop '*** INT_CORR_MODE variable not properly defined in corrmod.nml ***'
```

```
162     END SELECT
163
```

7.8.2.2 real(kind=8) function int_corr::func_ij (real(kind=8) *s*) [private]

Function that returns the component oi and oj of the correlation matrix at time s.

Parameters

s	time at which the function is evaluated

Definition at line 55 of file int_corr.f90.

```
55      IMPLICIT NONE
56      REAL(KIND=8) :: s,func_ij
57      CALL corrcomp(s)
58      func_ij=corr_ij(oi,oj)
59      RETURN
```

7.8.2.3 real(kind=8) function int_corr::func_ijkl (real(kind=8) *s*) [private]

Function that returns the component oi,oj,ok and ol of the outer product of the correlation matrix with itself at time s.

Parameters

s	time at which the function is evaluated

Definition at line 66 of file int_corr.f90.

```
66      IMPLICIT NONE
67      REAL(KIND=8) :: s,func_ijkl
68      CALL corrcomp(s)
69      func_ijkl=corr_ij(oi,oj)*corr_ij(ok,ol)
70      RETURN
```

7.8.2.4 subroutine, public int_corr::init_corrint ( )

Subroutine to initialise the integrated matrices and tensors.

Definition at line 38 of file int_corr.f90.

```
38      INTEGER :: allocstat
39
40      ALLOCATE(corrint(ndim,ndim), stat=allocstat)
41      IF (allocstat /= 0) stop "*** Not enough memory ! ***"
42
43      ALLOCATE(corr2int(ndim), stat=allocstat)
44      IF (allocstat /= 0) stop "*** Not enough memory ! ***"
45
46      CALL init_corr ! Initialize the correlation matrix function
47
48      corrint=0.d0
49
```

7.8.3 Variable Documentation

7.8.3.1 type(coolist4), dimension(:), allocatable, public int_corr::corr2int

Tensor holding the integral of the correlation outer product with itself.

Definition at line 30 of file int_corr.f90.

```
30   TYPE(coolist4), DIMENSION(:), ALLOCATABLE, PUBLIC :: corr2int !< Tensor holding the integral of
       the correlation outer product with itself
```

7.8.3.2 real(kind=8), dimension(:,:), allocatable, public int_corr::corrint

Matrix holding the integral of the correlation matrix.

Definition at line 29 of file int_corr.f90.

```
29   REAL(KIND=8), DIMENSION(:,:), ALLOCATABLE, PUBLIC :: corrint !< Matrix holding the integral of the
       correlation matrix
```

7.8.3.3 integer int_corr::oi [private]

Definition at line 26 of file int_corr.f90.

```
26   INTEGER :: oi,oj,ok,ol !< Integers that specify the matrices and tensor component considered as a
       function of time
```

7.8.3.4 integer int_corr::oj [private]

Definition at line 26 of file int_corr.f90.

7.8.3.5 integer int_corr::ok [private]

Definition at line 26 of file int_corr.f90.

7.8.3.6 integer int_corr::ol [private]

Integers that specify the matrices and tensor component considered as a function of time.

Definition at line 26 of file int_corr.f90.

7.8.3.7 real(kind=8), parameter int_corr::real_eps = 2.2204460492503131e-16 [private]

Small epsilon constant to determine equality with zero.

Definition at line 27 of file int_corr.f90.

```
27    REAL(KIND=8), PARAMETER :: real_eps = 2.2204460492503131e-16 !< Small epsilon constant to determine
        equality with zero
```

7.9 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.
- real(kind=8), dimension(:), allocatable buf_ka

  Buffer A to hold tendencies.
- real(kind=8), dimension(:), allocatable buf_kb

  Buffer B to hold tendencies.

7.9.1 Detailed Description

Module with the integration routines.

Module with the RK4 integration routines.

Copyright

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.

Copyright

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.

7.9.2 Function/Subroutine Documentation

7.9.2.1 subroutine public 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 ! ***"
```

7.9.2.2 subroutine public 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.

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

Parameters

y	Initial point.
t	Actual integration time
dt	Integration timestep.
res	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
```

7.9.2.3 subroutine 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

t	Time at which the tendencies have to be computed. Actually not needed for autonomous systems.
y	Point at which the tendencies have to be computed.
res	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)
```

7.9.3 Variable Documentation

7.9.3.1 real(kind=8), dimension(:), allocatable 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
```

7.9.3.2 real(kind=8), dimension(:), allocatable 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
```

7.9.3.3 real(kind=8), dimension(:), allocatable 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
```

7.9.3.4 real(kind=8), dimension(:), allocatable 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
```

7.9.3.5 real(kind=8), dimension(:), allocatable 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)
```

7.10 mar Module Reference

Multidimensional Autoregressive module to generate the correlation for the WL parameterization.

Functions/Subroutines

- subroutine, public init_mar
  Subroutine to initialise the MAR.
- subroutine, public mar_step (x)
  Routine to generate one step of the MAR.
- subroutine, public mar_step_red (xred)
  Routine to generate one step of the reduce MAR.
- subroutine stoch_vec (dW)

Variables

- real(kind=8), dimension(:,:), allocatable, public q
  Square root of the noise covariance matrix.
- real(kind=8), dimension(:,:), allocatable, public qred
  Reduce version of Q.
- real(kind=8), dimension(:,:), allocatable, public rred
  Covariance matrix of the noise.
- real(kind=8), dimension(:,:,:), allocatable, public w
  W_i matrix.
- real(kind=8), dimension(:,:,:), allocatable, public wred
  Reduce W_i matrix.
- real(kind=8), dimension(:), allocatable buf_y
- real(kind=8), dimension(:), allocatable dw
- integer, public ms
  order of the MAR

7.10.1 Detailed Description

Multidimensional Autoregressive module to generate the correlation for the WL parameterization.

Copyright

Remarks

Based on the equation $y_n = \sum_{i=1}^{m} y_{n-i} \cdot W_i + Q \cdot \xi_n foranorder$

7.10.2 Function/Subroutine Documentation

7.10.2.1 subroutine, public mar::init_mar ( )

Subroutine to initialise the MAR.

Definition at line 45 of file MAR.f90.

```
45      INTEGER :: allocstat,nf,i,info,info2
46      INTEGER, DIMENSION(3) :: s
47
48      print*, 'Initializing the MAR integrator...'
49
50      print*, 'Loading the MAR config from files...'
51
52      OPEN(20,file='MAR_R_params.def',status='old')
53      READ(20,*) nf,ms
54      IF (nf /= n_unres) stop "*** Dimension in files MAR_R_params.def and sf.nml do not correspond ! ***"
55      ALLOCATE(qred(n_unres,n_unres),rred(n_unres,n_unres),wred(ms,n_unres,n_unres),
       stat=allocstat)
56      IF (allocstat /= 0) stop "*** Not enough memory ! ***"
57      ALLOCATE(q(ndim,ndim),w(ms,ndim,ndim), stat=allocstat)
58      IF (allocstat /= 0) stop "*** Not enough memory ! ***"
59      ALLOCATE(buf_y(0:ndim), dw(ndim), stat=allocstat)
60      IF (allocstat /= 0) stop "*** Not enough memory ! ***"
61      READ(20,*) rred
62      CLOSE(20)
63
64      OPEN(20,file='MAR_W_params.def',status='old')
65      READ(20,*) nf,ms
66      s=shape(wred)
67      IF (nf /= n_unres) stop "*** Dimension in files MAR_W_params.def and sf.nml do not correspond ! ***"
68      IF (s(1) /= ms) stop "*** MAR order in files MAR_R_params.def and MAR_W_params.def do not correspond !
        ***"
69      DO i=1,ms
70         READ(20,*) wred(i,:,:)
71      ENDDO
72      CLOSE(20)
73
74      CALL init_sqrt
75      CALL sqrtm(rred,qred,info,info2)
76      CALL ireduce(q,qred,n_unres,ind,rind)
77
78      DO i=1,ms
79         CALL ireduce(w(i,:,:),wred(i,:,:),n_unres,ind,rind)
80      ENDDO
81
82      ! Kept for internal testing - Uncomment if not needed
83      ! DEALLOCATE(Wred,Rred,Qred, STAT=AllocStat)
84      ! IF (AllocStat /= 0) STOP "*** Deallocation problem ! ***"
85
86      print*, 'MAR of order',ms,'found!'
87
```

7.10.2.2 subroutine, public mar::mar_step (real(kind=8), dimension(0:ndim,ms), intent(inout) *x*)

Routine to generate one step of the MAR.

Parameters

x	State vector of the MAR (store the y_i)

Definition at line 93 of file MAR.f90.

```
93      REAL(KIND=8), DIMENSION(0:ndim,ms), INTENT(INOUT) :: x
94      INTEGER :: j
95
```

```
    CALL stoch_vec(dw)
    buf_y=0.d0
    buf_y(1:ndim)=matmul(q,dw)
    DO j=1,ms
        buf_y(1:ndim)=buf_y(1:ndim)+matmul(x(1:ndim,j),w(j,:,:))
     ENDDO
     x=eoshift(x,shift=-1,boundary=buf_y,dim=2)
```

7.10.2.3 subroutine, public mar::mar_step_red (real(kind=8), dimension(0:ndim,ms), intent(inout) *xred*)

Routine to generate one step of the reduce MAR.

Parameters

xred	State vector of the MAR (store the y_i)

Remarks

For debugging purpose only

Definition at line 110 of file MAR.f90.

```
    REAL(KIND=8), DIMENSION(0:ndim,ms), INTENT(INOUT) :: xred
    INTEGER :: j

    CALL stoch_vec(dw)
    buf_y=0.d0
    buf_y(1:n_unres)=matmul(qred,dw(1:n_unres))
    DO j=1,ms
       buf_y(1:n_unres)=buf_y(1:n_unres)+matmul(xred(1:n_unres,j),wred(j,:,:))
    ENDDO
    xred=eoshift(xred,shift=-1,boundary=buf_y,dim=2)
```

7.10.2.4 subroutine mar::stoch_vec (real(kind=8), dimension(ndim), intent(inout) *dW*) [private]

Definition at line 125 of file MAR.f90.

```
    REAL(KIND=8), DIMENSION(ndim), INTENT(INOUT) :: dw
    INTEGER :: i
    DO i=1,ndim
       dw(i)=gasdev()
    ENDDO
```

7.10.3 Variable Documentation

7.10.3.1 real(kind=8), dimension(:), allocatable mar::buf_y [private]

Definition at line 34 of file MAR.f90.

```
REAL(KIND=8), DIMENSION(:), ALLOCATABLE :: buf_y,dw
```

7.10.3.2 real(kind=8), dimension(:), allocatable mar::dw [private]

Definition at line 34 of file MAR.f90.

7.10.3.3 integer, public mar::ms

order of the MAR

Definition at line 36 of file MAR.f90.

```
36 INTEGER :: ms !< order of the MAR
```

7.10.3.4 real(kind=8), dimension(:,:), allocatable, public mar::q

Square root of the noise covariance matrix.

Definition at line 29 of file MAR.f90.

```
29 REAL(KIND=8), DIMENSION(:,:), ALLOCATABLE :: q !< Square root of the noise covariance matrix
```

7.10.3.5 real(kind=8), dimension(:,:), allocatable, public mar::qred

Reduce version of Q.

Definition at line 30 of file MAR.f90.

```
30 REAL(KIND=8), DIMENSION(:,:), ALLOCATABLE :: qred !< Reduce version of Q
```

7.10.3.6 real(kind=8), dimension(:,:), allocatable, public mar::rred

Covariance matrix of the noise.

Definition at line 31 of file MAR.f90.

```
31 REAL(KIND=8), DIMENSION(:,:), ALLOCATABLE :: rred !< Covariance matrix of the noise
```

7.10.3.7 real(kind=8), dimension(:,:,:), allocatable, public mar::w

W_i matrix.

Definition at line 32 of file MAR.f90.

```
32 REAL(KIND=8), DIMENSION(:,:,:), ALLOCATABLE :: w !< W_i matrix
```

7.10.3.8 real(kind=8), dimension(:,:,:), allocatable, public mar::wred

Reduce W_i matrix.

Definition at line 33 of file MAR.f90.

```
33   REAL(KIND=8), DIMENSION(:,:,:), ALLOCATABLE :: wred !< Reduce W_i matrix
```

7.11 memory Module Reference

Module that compute the memory term M_3 of the WL parameterization.

Functions/Subroutines

- subroutine, public init_memory
  Subroutine to initialise the memory.
- subroutine, public compute_m3 (y, dt, dtn, savey, save_ev, evolve, inter, h_int)
  Compute the integrand of M_3 at each time in the past and integrate to get the memory term.
- subroutine, public test_m3 (y, dt, dtn, h_int)
  Routine to test the #compute_M3 routine.

Variables

- real(kind=8), dimension(:,:), allocatable x
  Array storing the previous state of the system.
- real(kind=8), dimension(:,:), allocatable xs
  Array storing the resolved time evolution of the previous state of the system.
- real(kind=8), dimension(:,:), allocatable zs
  Dummy array to replace Xs in case where the evolution is not stored.
- real(kind=8), dimension(:), allocatable buf_m
  Dummy vector.
- real(kind=8), dimension(:), allocatable buf_m3
  Dummy vector to store the M_3 integrand.
- integer t_index
  Integer storing the time index (current position in the arrays)
- procedure(ss_step), pointer step
  Procedural pointer pointing on the resolved dynamics step routine.

7.11.1 Detailed Description

Module that compute the memory term M_3 of the WL parameterization.

Copyright

Remarks

7.11.2 Function/Subroutine Documentation

7.11.2.1 subroutine, public memory::compute_m3 (real(kind=8), dimension(0:ndim), intent(in) *y*, real(kind=8), intent(in) *dt*, real(kind=8), intent(in) *dtn*, logical, intent(in) *savey*, logical, intent(in) *save_ev*, logical, intent(in) *evolve*, real(kind=8), intent(in) *inter*, real(kind=8), dimension(0:ndim), intent(out) *h_int*)

Compute the integrand of M_3 at each time in the past and integrate to get the memory term.

Parameters

y	current state
dt	timestep
dtn	stochastic timestep
savey	set if the state is stored in X at the end
save_ev	set if the result of the resolved time evolution is stored in Xs at the end
evolve	set if the resolved time evolution is performed
inter	set over which time interval the resolved time evolution must be computed
h_int	result of the integration - give the memory term

Definition at line 86 of file memory.f90.

```
86     REAL(KIND=8), DIMENSION(0:ndim), INTENT(IN) :: y
87     REAL(KIND=8), INTENT(IN) :: dt,dtn
88     LOGICAL, INTENT(IN) :: savey,save_ev,evolve
89     REAL(KIND=8), INTENT(IN) :: inter
90     REAL(KIND=8), DIMENSION(0:ndim), INTENT(OUT) :: h_int
91     REAL(KIND=8) :: t
92     INTEGER :: i,j
93
94     x(:,t_index)=y
95     IF (b23def) THEN
96        xs(:,t_index)=y
97        zs(:,t_index)=y
98        DO i=1,mems-1
99           j=modulo(t_index+i-1,mems)+1
100           zs(:,j)=xs(:,j)
101           IF (evolve) THEN
102              IF (dt.lt.inter) THEN
103                 t=0.d0
104                 DO WHILE (t+dt<inter)
105                    CALL step(zs(:,j),y,t,dt,dtn,zs(:,j))
106                 ENDDO
107                 CALL step(zs(:,j),y,t,inter-t,sqrt(inter-t),zs(:,j))
108              ELSE
109                 CALL step(zs(:,j),y,t,inter,sqrt(inter),zs(:,j))
110              ENDIF
111           ENDIF
112           IF (save_ev) xs(:,j)=zs(:,j)
113        ENDDO
114     ENDIF
115
116
117     ! Computing the integral
118     h_int=0.d0
119
120     DO i=1,mems
121        j=modulo(t_index+i-2,mems)+1
122        buf_m3=0.d0
123        IF (ldef) THEN
124           CALL sparse_mul3(ltot(:,i),y,x(:,j),buf_m)
125           buf_m3=buf_m3+buf_m
126        ENDIF
127        IF (b14def) THEN
128           CALL sparse_mul3(b14(:,i),x(:,j),x(:,j),buf_m)
129           buf_m3=buf_m3+buf_m
130        ENDIF
131        IF (b23def) THEN
132           CALL sparse_mul3(b23(:,i),x(:,j),zs(:,j),buf_m)
133           buf_m3=buf_m3+buf_m
134        ENDIF
```

```
      IF (mdef) THEN
         CALL sparse_mul4(mtot(:,i),x(:,j),x(:,j),zs(:,j),buf_m)
         buf_m3=buf_m3+buf_m
      ENDIF
      IF ((i.eq.1).or.(i.eq.mems)) THEN
         h_int=h_int+0.5*buf_m3
      ELSE
         h_int=h_int+buf_m3
      ENDIF
   ENDDO

   h_int=muti*h_int
   IF (savey) THEN
      t_index=t_index-1
      IF (t_index.eq.0) t_index=mems
   ENDIF
```

7.11.2.2 subroutine, public memory::init_memory ( )

Subroutine to initialise the memory.

Definition at line 45 of file memory.f90.

```
    INTEGER :: allocstat

    t_index=mems

    ALLOCATE(x(0:ndim,mems), stat=allocstat)
    IF (allocstat /= 0) stop "*** Not enough memory ! ***"

    x=0.d0

    IF (b23def) THEN
       ALLOCATE(xs(0:ndim,mems), zs(0:ndim,mems), stat=allocstat)
       IF (allocstat /= 0) stop "*** Not enough memory ! ***"

       xs=0.d0
    ENDIF

    ALLOCATE(buf_m3(0:ndim), buf_m(0:ndim), stat=allocstat)
    IF (allocstat /= 0) stop "*** Not enough memory ! ***"

    SELECT CASE (x_int_mode)
    CASE('reso')
       step => ss_step
    CASE('tang')
       step => ss_tl_step
    CASE DEFAULT
       stop '*** X_INT_MODE variable not properly defined in stoch_params.nml ***'
    END SELECT

```

7.11.2.3 subroutine, public memory::test_m3 (real(kind=8), dimension(0:ndim), intent(in) *y*, real(kind=8), intent(in) *dt*, real(kind=8), intent(in) *dtn*, real(kind=8), dimension(0:ndim), intent(out) *h_int*)

Routine to test the #compute_M3 routine.

Parameters

y	current state
dt	timestep
dtn	stochastic timestep
h_int	result of the integration - give the memory term

Definition at line 159 of file memory.f90.

```
    REAL(KIND=8), DIMENSION(0:ndim), INTENT(IN) :: y
    REAL(KIND=8), INTENT(IN) :: dt,dtn
    REAL(KIND=8), DIMENSION(0:ndim), INTENT(OUT) :: h_int
    INTEGER :: i,j

    CALL compute_m3(y,dt,dtn,.true.,.true.,.true.,muti,h_int)
    print*, t_index
    print*, 'X'
    DO i=1,mems
       j=modulo(t_index+i-1,mems)+1
       print*, i,j,x(1,j)
    ENDDO

    IF (b23def) THEN
       print*, 'Xs'
       DO i=1,mems
          j=modulo(t_index+i-1,mems)+1
          print*, i,j,xs(1,j)
       ENDDO
    ENDIF
    print*, 'h_int',h_int
```

7.11.3 Variable Documentation

7.11.3.1 real(kind=8), dimension(:), allocatable memory::buf_m [private]

Dummy vector.

Definition at line 31 of file memory.f90.

```
31 REAL(KIND=8), DIMENSION(:), ALLOCATABLE :: buf_m !< Dummy vector
```

7.11.3.2 real(kind=8), dimension(:), allocatable memory::buf_m3 [private]

Dummy vector to store the M_3 integrand.

Definition at line 32 of file memory.f90.

```
32 REAL(KIND=8), DIMENSION(:), ALLOCATABLE :: buf_m3 !< Dummy vector to store the \f$M_3\f$ integrand
```

7.11.3.3 procedure(ss_step), pointer memory::step [private]

Procedural pointer pointing on the resolved dynamics step routine.

Definition at line 36 of file memory.f90.

```
36 PROCEDURE(ss_step), POINTER :: step !< Procedural pointer pointing on the resolved dynamics step routine
```

7.11.3.4 integer memory::t_index [private]

Integer storing the time index (current position in the arrays)

Definition at line 34 of file memory.f90.

```
34 INTEGER :: t_index !< Integer storing the time index (current position in the arrays)
```

7.11.3.5 real(kind=8), dimension(:,:), allocatable memory::x [private]

Array storing the previous state of the system.

Definition at line 28 of file memory.f90.

```
28    REAL(KIND=8), DIMENSION(:,:), ALLOCATABLE :: x !< Array storing the previous state of the system
```

7.11.3.6 real(kind=8), dimension(:,:), allocatable memory::xs [private]

Array storing the resolved time evolution of the previous state of the system.

Definition at line 29 of file memory.f90.

```
29    REAL(KIND=8), DIMENSION(:,:), ALLOCATABLE :: xs !< Array storing the resolved time evolution of the
       previous state of the system
```

7.11.3.7 real(kind=8), dimension(:,:), allocatable memory::zs [private]

Dummy array to replace Xs in case where the evolution is not stored.

Definition at line 30 of file memory.f90.

```
30    REAL(KIND=8), DIMENSION(:,:), ALLOCATABLE :: zs !< Dummy array to replace Xs in case where the evolution
       is not stored
```

7.12 mtv_int_tensor Module Reference

The MTV tensors used to integrate the MTV model.

Functions/Subroutines

- subroutine, public init_mtv_int_tensor

    Subroutine to initialise the MTV tensor.

Variables

- real(kind=8), dimension(:), allocatable, public h1

    First constant vector.
- real(kind=8), dimension(:), allocatable, public h2

    Second constant vector.
- real(kind=8), dimension(:), allocatable, public h3

    Third constant vector.
- real(kind=8), dimension(:), allocatable, public htot

    Total constant vector.
- type(coolist), dimension(:), allocatable, public l1

    First linear tensor.
- type(coolist), dimension(:), allocatable, public l2

    Second linear tensor.
- type(coolist), dimension(:), allocatable, public l3

    Third linear tensor.
- type(coolist), dimension(:), allocatable, public ltot

    Total linear tensor.
- type(coolist), dimension(:), allocatable, public b1

    First quadratic tensor.
- type(coolist), dimension(:), allocatable, public b2

    Second quadratic tensor.
- type(coolist), dimension(:), allocatable, public btot

    Total quadratic tensor.
- type(coolist4), dimension(:), allocatable, public mtot

    Tensor for the cubic terms.
- real(kind=8), dimension(:,:), allocatable, public q1

    Constant terms for the state-dependent noise covariance matrix.
- real(kind=8), dimension(:,:), allocatable, public q2

    Constant terms for the state-independent noise covariance matrix.
- type(coolist), dimension(:), allocatable, public utot

    Linear terms for the state-dependent noise covariance matrix.
- type(coolist4), dimension(:), allocatable, public vtot

    Quadratic terms for the state-dependent noise covariance matrix.
- real(kind=8), dimension(:), allocatable dumb_vec

    Dummy vector.
- real(kind=8), dimension(:,:), allocatable dumb_mat1

    Dummy matrix.
- real(kind=8), dimension(:,:), allocatable dumb_mat2

    Dummy matrix.
- real(kind=8), dimension(:,:), allocatable dumb_mat3

    Dummy matrix.
- real(kind=8), dimension(:,:), allocatable dumb_mat4

    Dummy matrix.

7.12.1 Detailed Description

The MTV tensors used to integrate the MTV model.

Copyright

Remarks

See : Franzke, C., Majda, A. J., & Vanden-Eijnden, E. (2005). Low-order stochastic mode reduction for a realistic barotropic model climate. Journal of the atmospheric sciences, 62(6), 1722-1745.

7.12.2 Function/Subroutine Documentation

7.12.2.1 subroutine, public mtv_int_tensor::init_mtv_int_tensor ( )

Subroutine to initialise the MTV tensor.

Definition at line 89 of file MTV_int_tensor.f90.

```
89      INTEGER :: allocstat,i,j,k,l
90
91      print*, 'Initializing the decomposition tensors...'
92      CALL init_dec_tensor
93      print*, "Initializing the correlation matrices and tensors..."
94      CALL init_corrint
95      print*, "Computing the correlation integrated matrices and tensors..."
96      CALL comp_corrint
97
98      !H part
99      print*, "Computing the H term..."
100
101      ALLOCATE(h1(0:ndim), h2(0:ndim), h3(0:ndim), htot(0:ndim),
     stat=allocstat)
102      IF (allocstat /= 0) stop "*** Not enough memory ! ***"
103
104      ALLOCATE(dumb_mat1(ndim,ndim), dumb_mat2(ndim,ndim), stat=allocstat)
105      IF (allocstat /= 0) stop "*** Not enough memory ! ***"
106
107      ALLOCATE(dumb_mat3(ndim,ndim), dumb_mat4(ndim,ndim), stat=allocstat)
108      IF (allocstat /= 0) stop "*** Not enough memory ! ***"
109
110      !H1
111      CALL coo_to_mat_ik(lxy,dumb_mat1)
112      dumb_mat2=matmul(dumb_mat1,corrint)
113      CALL sparse_mul3_with_mat(bxxy,dumb_mat2,h1)
114
115      ! H2
116      h2=0.d0
117      IF (mode.ne.'ures') THEN
118         CALL coo_to_mat_ik(lyy,dumb_mat1)
119         dumb_mat1=matmul(inv_corr_i_full,dumb_mat1)
120
121         DO i=1,ndim
122            CALL coo_to_mat_i(i,bxyy,dumb_mat2)
123            CALL sparse_mul4_with_mat_jl(corr2int,dumb_mat2,dumb_mat3)
124            CALL sparse_mul4_with_mat_jl(corr2int,transpose(dumb_mat2),dumb_mat4)
125            dumb_mat3=dumb_mat3+dumb_mat4
126            h2(i)=mat_contract(dumb_mat1,dumb_mat3)
127         ENDDO
128      ENDIF
129
130      !H3
131      h3=0.d0
132      CALL sparse_mul3_with_mat(bxyy,corr_i_full,h3)
133
134      !Htot
135      htot=0.d0
136      htot=h1+h2+h3
```

```
    print*, "Computing the L terms..."
    ALLOCATE(l1(ndim), l2(ndim), l3(ndim), stat=allocstat)
    IF (allocstat /= 0) stop "*** Not enough memory ! ***"

    !L1
    CALL coo_to_mat_ik(lyx,dumb_mat1)
    CALL coo_to_mat_ik(lxy,dumb_mat2)
    dumb_mat3=matmul(inv_corr_i_full,corrint)
    dumb_mat4=matmul(dumb_mat2,matmul(transpose(dumb_mat3),dumb_mat1))
    CALL matc_to_coo(dumb_mat4,l1)

    !L2
    dumb_mat4=0.d0
    DO i=1,ndim
       DO j=1,ndim
          CALL coo_to_mat_i(i,bxyy,dumb_mat1)
          CALL sparse_mul4_with_mat_jl(corr2int,dumb_mat1+transpose(dumb_mat1),dumb_mat2)

          CALL coo_to_mat_j(j,byxy,dumb_mat1)
          dumb_mat1=matmul(inv_corr_i_full,dumb_mat1)
          dumb_mat4(i,j)=mat_contract(dumb_mat1,dumb_mat2)
       END DO
    END DO
    CALL matc_to_coo(dumb_mat4,l2)

    !L3
    dumb_mat4=0.d0
    DO i=1,ndim
       DO j=1,ndim
          CALL coo_to_mat_j(j,bxxy,dumb_mat1)
          CALL coo_to_mat_i(i,bxxy,dumb_mat2)
          dumb_mat4(i,j)=mat_trace(matmul(dumb_mat1,matmul(corrint,transpose(dumb_mat2))))
       ENDDO
    END DO
    CALL matc_to_coo(dumb_mat4,l3)

    !Ltot

    ALLOCATE(ltot(ndim), stat=allocstat)
    IF (allocstat /= 0) stop "*** Not enough memory ! ***"

    CALL add_to_tensor(l1,ltot)
    CALL add_to_tensor(l2,ltot)
    CALL add_to_tensor(l3,ltot)

    print*, "Computing the B terms..."
    ALLOCATE(b1(ndim), b2(ndim), btot(ndim), stat=allocstat)
    IF (allocstat /= 0) stop "*** Not enough memory ! ***"
    ALLOCATE(dumb_vec(ndim), stat=allocstat)
    IF (allocstat /= 0) stop "*** Not enough memory ! ***"

    ! B1
    CALL coo_to_mat_ik(lxy,dumb_mat1)
    dumb_mat2=matmul(inv_corr_i_full,corrint)

    dumb_mat3=matmul(dumb_mat1,transpose(dumb_mat2))
    DO j=1,ndim
       DO k=1,ndim
          CALL coo_to_vec_jk(j,k,byxx,dumb_vec)
          dumb_vec=matmul(dumb_mat3,dumb_vec)
          CALL add_vec_jk_to_tensor(j,k,dumb_vec,b1)
       ENDDO
    END DO

    ! B2
    CALL coo_to_mat_ik(lyx,dumb_mat3)
    dumb_mat2=matmul(inv_corr_i_full,corrint)

    dumb_mat4=matmul(transpose(dumb_mat2),dumb_mat3)
    DO i=1,ndim
       CALL coo_to_mat_i(i,bxxy,dumb_mat1)
       dumb_mat2=matmul(dumb_mat1,dumb_mat4)
       CALL add_matc_to_tensor(i,dumb_mat2,b2)
    ENDDO

    CALL add_to_tensor(b1,btot)
    CALL add_to_tensor(b2,btot)

    !M

    print*, "Computing the M term..."

    ALLOCATE(mtot(ndim), stat=allocstat)
    IF (allocstat /= 0) stop "*** Not enough memory ! ***"

    dumb_mat2=matmul(inv_corr_i_full,corrint)
```

```

    DO i=1,ndim
       CALL coo_to_mat_i(i,bxxy,dumb_mat1)
       dumb_mat3=matmul(dumb_mat1,transpose(dumb_mat2))
       DO k=1,ndim
          DO l=1,ndim
             CALL coo_to_vec_jk(k,l,byxx,dumb_vec)
             dumb_vec=matmul(dumb_mat3,dumb_vec)
             CALL add_vec_ikl_to_tensor4(i,k,l,dumb_vec,mtot)
          ENDDO
       END DO
    END DO

    !Q

    print*, "Computing the Q terms..."
    ALLOCATE(q1(ndim,ndim), q2(ndim,ndim), stat=allocstat)
    IF (allocstat /= 0) stop "*** Not enough memory ! ***"

    !Q1

    CALL coo_to_mat_ik(lxy,dumb_mat1)
    q1=matmul(dumb_mat1,matmul(corrint,transpose(dumb_mat1)))

    !Q2

    DO i=1,ndim
       DO j=1,ndim
          CALL coo_to_mat_i(i,bxyy,dumb_mat1)
          CALL coo_to_mat_i(j,bxyy,dumb_mat2)
          CALL sparse_mul4_with_mat_jl(corr2int,dumb_mat2,dumb_mat3)
          CALL sparse_mul4_with_mat_jl(corr2int,transpose(dumb_mat2),dumb_mat4)
          dumb_mat2=dumb_mat3+dumb_mat4
          q2(i,j)=mat_contract(dumb_mat1,dumb_mat2)
       END DO
    END DO

    !U

    ALLOCATE(utot(ndim), stat=allocstat)
    IF (allocstat /= 0) stop "*** Not enough memory ! ***"

    CALL coo_to_mat_ik(lxy,dumb_mat1)
    DO i=1,ndim
       CALL coo_to_mat_i(i,bxxy,dumb_mat2)
       dumb_mat3=matmul(dumb_mat1,matmul(corrint,transpose(dumb_mat2)))
       CALL add_matc_to_tensor(i,dumb_mat3,utot)
    ENDDO

    DO j=1,ndim
       CALL coo_to_mat_i(j,bxxy,dumb_mat2)
       dumb_mat3=matmul(dumb_mat1,matmul(corrint,transpose(dumb_mat2)))
       DO k=1,ndim
          CALL add_vec_jk_to_tensor(j,k,dumb_mat3(:,k),utot)
       ENDDO
    ENDDO

    !V

    ALLOCATE(vtot(ndim), stat=allocstat)
    IF (allocstat /= 0) stop "*** Not enough memory ! ***"

    DO i=1,ndim
       DO j=1,ndim
          CALL coo_to_mat_i(i,bxxy,dumb_mat1)
          CALL coo_to_mat_i(j,bxxy,dumb_mat2)
          dumb_mat3=matmul(dumb_mat1,matmul(corrint,transpose(dumb_mat2)))
          CALL add_matc_to_tensor4(j,i,dumb_mat3,vtot)
       ENDDO
    ENDDO

    DEALLOCATE(dumb_mat1, dumb_mat2, stat=allocstat)
    IF (allocstat /= 0)  stop "*** Problem to deallocate ! ***"

    DEALLOCATE(dumb_mat3, dumb_mat4, stat=allocstat)
    IF (allocstat /= 0)  stop "*** Problem to deallocate ! ***"

    DEALLOCATE(dumb_vec, stat=allocstat)
    IF (allocstat /= 0)  stop "*** Problem to deallocate ! ***"


```

7.12.3 Variable Documentation

7.12.3.1 type(coolist), dimension(:), allocatable, public mtv_int_tensor::b1

First quadratic tensor.

Definition at line 54 of file MTV_int_tensor.f90.

```
54 TYPE(coolist), DIMENSION(:), ALLOCATABLE, PUBLIC :: b1 !< First quadratic tensor
```

7.12.3.2 type(coolist), dimension(:), allocatable, public mtv_int_tensor::b2

Second quadratic tensor.

Definition at line 55 of file MTV_int_tensor.f90.

```
55 TYPE(coolist), DIMENSION(:), ALLOCATABLE, PUBLIC :: b2 !< Second quadratic tensor
```

7.12.3.3 type(coolist), dimension(:), allocatable, public mtv_int_tensor::btot

Total quadratic tensor.

Definition at line 56 of file MTV_int_tensor.f90.

```
56 TYPE(coolist), DIMENSION(:), ALLOCATABLE, PUBLIC :: btot !< Total quadratic tensor
```

7.12.3.4 real(kind=8), dimension(:,:), allocatable mtv_int_tensor::dumb_mat1 [private]

Dummy matrix.

Definition at line 67 of file MTV_int_tensor.f90.

```
67 REAL(KIND=8), DIMENSION(:,:), ALLOCATABLE :: dumb_mat1 !< Dummy matrix
```

7.12.3.5 real(kind=8), dimension(:,:), allocatable mtv_int_tensor::dumb_mat2 [private]

Dummy matrix.

Definition at line 68 of file MTV_int_tensor.f90.

```
68 REAL(KIND=8), DIMENSION(:,:), ALLOCATABLE :: dumb_mat2 !< Dummy matrix
```

7.12.3.6 real(kind=8), dimension(:,:), allocatable mtv_int_tensor::dumb_mat3 [private]

Dummy matrix.

Definition at line 69 of file MTV_int_tensor.f90.

```
69    REAL(KIND=8), DIMENSION(:,:), ALLOCATABLE :: dumb_mat3 !< Dummy matrix
```

7.12.3.7 real(kind=8), dimension(:,:), allocatable mtv_int_tensor::dumb_mat4 [private]

Dummy matrix.

Definition at line 70 of file MTV_int_tensor.f90.

```
70    REAL(KIND=8), DIMENSION(:,:), ALLOCATABLE :: dumb_mat4 !< Dummy matrix
```

7.12.3.8 real(kind=8), dimension(:), allocatable mtv_int_tensor::dumb_vec [private]

Dummy vector.

Definition at line 66 of file MTV_int_tensor.f90.

```
66    REAL(KIND=8), DIMENSION(:), ALLOCATABLE :: dumb_vec !< Dummy vector
```

7.12.3.9 real(kind=8), dimension(:), allocatable, public mtv_int_tensor::h1

First constant vector.

Definition at line 42 of file MTV_int_tensor.f90.

```
42    REAL(KIND=8), DIMENSION(:), ALLOCATABLE, PUBLIC :: h1   !< First constant vector
```

7.12.3.10 real(kind=8), dimension(:), allocatable, public mtv_int_tensor::h2

Second constant vector.

Definition at line 43 of file MTV_int_tensor.f90.

```
43    REAL(KIND=8), DIMENSION(:), ALLOCATABLE, PUBLIC :: h2   !< Second constant vector
```

7.12.3.11 real(kind=8), dimension(:), allocatable, public mtv_int_tensor::h3

Third constant vector.

Definition at line 44 of file MTV_int_tensor.f90.

```
44    REAL(KIND=8), DIMENSION(:), ALLOCATABLE, PUBLIC :: h3   !< Third constant vector
```

7.12.3.12 real(kind=8), dimension(:), allocatable, public mtv_int_tensor::htot

Total constant vector.

Definition at line 45 of file MTV_int_tensor.f90.

```
45    REAL(KIND=8), DIMENSION(:), ALLOCATABLE, PUBLIC :: htot !< Total constant vector
```

7.12.3.13 type(coolist), dimension(:), allocatable, public mtv_int_tensor::l1

First linear tensor.

Definition at line 48 of file MTV_int_tensor.f90.

```
48    TYPE(coolist), DIMENSION(:), ALLOCATABLE, PUBLIC :: l1   !< First linear tensor
```

7.12.3.14 type(coolist), dimension(:), allocatable, public mtv_int_tensor::l2

Second linear tensor.

Definition at line 49 of file MTV_int_tensor.f90.

```
49    TYPE(coolist), DIMENSION(:), ALLOCATABLE, PUBLIC :: l2   !< Second linear tensor
```

7.12.3.15 type(coolist), dimension(:), allocatable, public mtv_int_tensor::l3

Third linear tensor.

Definition at line 50 of file MTV_int_tensor.f90.

```
50    TYPE(coolist), DIMENSION(:), ALLOCATABLE, PUBLIC :: l3   !< Third linear tensor
```

7.12.3.16 type(coolist), dimension(:), allocatable, public mtv_int_tensor::ltot

Total linear tensor.

Definition at line 51 of file MTV_int_tensor.f90.

```
51    TYPE(coolist), DIMENSION(:), ALLOCATABLE, PUBLIC :: ltot !< Total linear tensor
```

7.12.3.17 type(coolist4), dimension(:), allocatable, public mtv_int_tensor::mtot

Tensor for the cubic terms.

Definition at line 58 of file MTV_int_tensor.f90.

```
58    TYPE(coolist4), DIMENSION(:), ALLOCATABLE, PUBLIC :: mtot !< Tensor for the cubic terms
```

7.12.3.18 real(kind=8), dimension(:,:), allocatable, public mtv_int_tensor::q1

Constant terms for the state-dependent noise covariance matrix.

Definition at line 61 of file MTV_int_tensor.f90.

```
61    REAL(KIND=8), DIMENSION(:,:), ALLOCATABLE, PUBLIC :: q1 !< Constant terms for the state-dependent noise
        covariance matrix
```

7.12.3.19 real(kind=8), dimension(:,:), allocatable, public mtv_int_tensor::q2

Constant terms for the state-independent noise covariance matrix.

Definition at line 62 of file MTV_int_tensor.f90.

```
62    REAL(KIND=8), DIMENSION(:,:), ALLOCATABLE, PUBLIC :: q2 !< Constant terms for the state-independent noise
        covariance matrix
```

7.12.3.20 type(coolist), dimension(:), allocatable, public mtv_int_tensor::utot

Linear terms for the state-dependent noise covariance matrix.

Definition at line 63 of file MTV_int_tensor.f90.

```
63    TYPE(coolist), DIMENSION(:), ALLOCATABLE, PUBLIC :: utot !< Linear terms for the state-dependent
        noise covariance matrix
```

7.12.3.21 type(coolist4), dimension(:), allocatable, public mtv_int_tensor::vtot

Quadratic terms for the state-dependent noise covariance matrix.

Definition at line 64 of file MTV_int_tensor.f90.

```
64   TYPE(coolist4), DIMENSION(:), ALLOCATABLE, PUBLIC :: vtot !< Quadratic terms for the
       state-dependent noise covariance matrix
```

7.13 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
  Merchanical 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
  Depth of the active water layer of the ocean.
- real(kind=8) phi0_npi
  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

    β' *- Non-dimensional beta parameter*
- 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.

7.13.1 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

7.13.2 Function/Subroutine Documentation

7.13.2.1 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      OPEN(8, file="int_params.nml", status='OLD', recl=80, delim='APOSTROPHE')
126      READ(8,nml=int_params)
127
128
```

7.13.2.2 subroutine params::init_params ( )

Parameters initialisation routine.

Definition at line 133 of file params.f90.

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

```
    noc=s(1)

    ndim=2*natm+2*noc

    !-----------------------------------------------------------!
    !                                                           !
    ! Some general parameters (Domain, beta, gamma, coupling)  !
    !                                                           !
    !-----------------------------------------------------------!

    pi=dacos(-1.d0)
    l=scale/pi
    phi0=phi0_npi*pi
    lr=sqrt(gp*h)/f0
    g=-l**2/lr**2
    betp=l/rra*cos(phi0)/sin(phi0)
    rp=r/f0
    dp=d/f0
    kd=k*2
    kdp=kp

    !-----------------------------------------------------!
    !                                                     !
    ! DERIVED QUANTITIES                                  !
    !                                                     !
    !-----------------------------------------------------!

    cpo=co/(go*f0) * rr/(f0**2*l**2)
    lpo=lambda/(go*f0)
    cpa=ca/(ga*f0) * rr/(f0**2*l**2)/2 ! Cpa acts on psi1-psi3, not on theta
    lpa=lambda/(ga*f0)
    sbpo=4*sb*to0**3/(go*f0) ! long wave radiation lost by ocean to atmosphere space
    sbpa=8*epsa*sb*ta0**3/(go*f0) ! long wave radiation from atmosphere absorbed by ocean
    lsbpo=2*epsa*sb*to0**3/(ga*f0) ! long wave radiation from ocean absorbed by atmosphere
    lsbpa=8*epsa*sb*ta0**3/(ga*f0) ! long wave radiation lost by atmosphere to space & ocea
```

7.13.3 Variable Documentation

7.13.3.1 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
```

7.13.3.2 real(kind=8) params::betp

β' - Non-dimensional beta parameter

Definition at line 67 of file params.f90.

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

7.13.3.3 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$C_a\f$ - Constant short-wave radiation of the atmosphere.
```

7.13.3.4 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.
```

7.13.3.5 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.
```

7.13.3.6 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.
```

7.13.3.7 real(kind=8) params::d

Merchanical coupling parameter between the ocean and the atmosphere.

Definition at line 31 of file params.f90.

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

7.13.3.8 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.
```

7.13.3.9 real(kind=8) params::dt

Integration time step.

Definition at line 71 of file params.f90.

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

7.13.3.10 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.
```

7.13.3.11 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
```

7.13.3.12 real(kind=8) params::g

γ

Definition at line 50 of file params.f90.

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

7.13.3.13 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.
```

7.13.3.14 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.
```

7.13.3.15 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
```

7.13.3.16 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.
```

7.13.3.17 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.
```

7.13.3.18 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.
```

7.13.3.19 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.
```

7.13.3.20 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.
```

7.13.3.21 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
```

7.13.3.22 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.
```

7.13.3.23 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.
```

7.13.3.24 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.
```

7.13.3.25 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
```

7.13.3.26 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.
```

7.13.3.27 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.
```

7.13.3.28 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
```

7.13.3.29 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
```

7.13.3.30 integer params::nbatm

Number of oceanic blocks.

Definition at line 76 of file params.f90.

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

7.13.3.31 integer params::nboc

Number of atmospheric blocks.

Definition at line 75 of file params.f90.

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

7.13.3.32 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)
```

7.13.3.33 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
```

7.13.3.34 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
```

7.13.3.35 real(kind=8) params::phi0

Latitude in radian.

Definition at line 25 of file params.f90.

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

7.13.3.36 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.
```

7.13.3.37 real(kind=8) params::pi

π

Definition at line 48 of file params.f90.

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

7.13.3.38 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.
```

7.13.3.39 real(kind=8) params::rp

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

Definition at line 51 of file params.f90.

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

7.13.3.40 real(kind=8) params::rr

R - Gas constant of dry air

Definition at line 45 of file params.f90.

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

7.13.3.41 real(kind=8) params::rra

Earth radius.

Definition at line 26 of file params.f90.

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

7.13.3.42 real(kind=8) params::sb

Stefan–Boltzmann constant.

Definition at line 66 of file params.f90.

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

7.13.3.43 real(kind=8) params::sbpa

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

Definition at line 61 of file params.f90.

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

7.13.3.44 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.
```

7.13.3.45 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.
```

7.13.3.46 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.
```

7.13.3.47 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.
```

7.13.3.48 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)
```

7.13.3.49 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
```

7.13.3.50 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.
```

7.13.3.51 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.
```

7.13.3.52 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
```

7.13.3.53 logical params::writeout

Write to file boolean.

Definition at line 73 of file params.f90.

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

7.14 rk2_mtv_integrator Module Reference

Module with the MTV rk2 integration routines.

Functions/Subroutines

- subroutine, public init_integrator

  Subroutine to initialize the MTV rk2 integrator.
- subroutine init_noise

  Routine to initialize the noise vectors and buffers.
- subroutine init_g

  Routine to initialize the G term.
- subroutine compg (y)

  Routine to actualize the G term based on the state y of the MTV system.
- subroutine, public step (y, t, dt, dtn, res, tend)

  Routine to perform an integration step (Heun algorithm) of the MTV system. The incremented time is returned.
- subroutine, public full_step (y, t, dt, dtn, res)

  Routine to perform an integration step (Heun algorithm) of the full stochastic system. The incremented time is returned.

Variables

- real(kind=8), dimension(:), allocatable buf_y1
- real(kind=8), dimension(:), allocatable buf_f0
- real(kind=8), dimension(:), allocatable buf_f1

  Integration buffers.
- real(kind=8), dimension(:), allocatable dw
- real(kind=8), dimension(:), allocatable dwmult

  Standard gaussian noise buffers.
- real(kind=8), dimension(:), allocatable dwar
- real(kind=8), dimension(:), allocatable dwau
- real(kind=8), dimension(:), allocatable dwor
- real(kind=8), dimension(:), allocatable dwou

  Standard gaussian noise buffers.
- real(kind=8), dimension(:), allocatable anoise
- real(kind=8), dimension(:), allocatable noise

  Additive noise term.
- real(kind=8), dimension(:), allocatable noisemult

  Multiplicative noise term.
- real(kind=8), dimension(:), allocatable g

  G term of the MTV tendencies.
- real(kind=8), dimension(:), allocatable buf_g

  Buffer for the G term computation.
- logical mult

  Logical indicating if the sigma1 matrix must be computed for every state change.
- logical q1fill

  Logical indicating if the matrix Q1 is non-zero.
- logical compute_mult

  Logical indicating if the Gaussian noise for the multiplicative noise must be computed.
- real(kind=8), parameter sq2 = sqrt(2.D0)

  Hard coded square root of 2.

7.14.1 Detailed Description

Module with the MTV rk2 integration routines.

Copyright

2017 Jonathan Demaeyer. See LICENSE.txt for license information.

Remarks

This module actually contains the Heun algorithm routines.

7.14.2 Function/Subroutine Documentation

7.14.2.1 subroutine rk2_mtv_integrator::compg (real(kind=8), dimension(0:ndim), intent(in) *y*) [private]

Routine to actualize the G term based on the state y of the MTV system.

Parameters

y	State of the MTV system

Definition at line 105 of file rk2_MTV_integrator.f90.

```
105     REAL(KIND=8), DIMENSION(0:ndim), INTENT(IN) :: y
106
107     g=htot
108     CALL sparse_mul2_k(ltot,y,buf_g)
109     g=g+buf_g
110     CALL sparse_mul3(btot,y,y,buf_g)
111     g=g+buf_g
112     CALL sparse_mul4(mtot,y,y,y,buf_g)
113     g=g+buf_g
```

7.14.2.2 subroutine, public rk2_mtv_integrator::full_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), intent(in) *dtn*, real(kind=8), dimension(0:ndim), intent(out) *res*)

Routine to perform an integration step (Heun algorithm) of the full stochastic system. The incremented time is returned.

Parameters

y	Initial point.
t	Actual integration time
dt	Integration timestep.
dtn	Stochastoc integration timestep (normally square-root of dt).
res	Final point after the step.

Definition at line 170 of file rk2_MTV_integrator.f90.

```
REAL(KIND=8), DIMENSION(0:ndim), INTENT(IN) :: y
REAL(KIND=8), INTENT(INOUT) :: t
REAL(KIND=8), INTENT(IN) :: dt,dtn
REAL(KIND=8), DIMENSION(0:ndim), INTENT(OUT) :: res
CALL stoch_atm_res_vec(dwar)
CALL stoch_atm_unres_vec(dwau)
CALL stoch_oc_res_vec(dwor)
CALL stoch_oc_unres_vec(dwou)
anoise=(q_ar*dwar+q_au*dwau+q_or*dwor+q_ou*dwou)*dtn
CALL sparse_mul3(aotensor,y,y,buf_f0)
buf_y1 = y+dt*buf_f0+anoise
CALL sparse_mul3(aotensor,buf_y1,buf_y1,buf_f1)
res=y+0.5*(buf_f0+buf_f1)*dt+anoise
t=t+dt
```

7.14.2.3 subroutine rk2_mtv_integrator::init_g ( ) [private]

Routine to initialize the G term.

Definition at line 97 of file rk2_MTV_integrator.f90.

```
INTEGER :: allocstat
ALLOCATE(g(0:ndim), buf_g(0:ndim), stat=allocstat)
IF (allocstat /= 0) stop "*** Not enough memory ! ***"
```

7.14.2.4 subroutine, public rk2_mtv_integrator::init_integrator ( )

Subroutine to initialize the MTV rk2 integrator.

Definition at line 50 of file rk2_MTV_integrator.f90.

```
INTEGER :: allocstat

CALL init_ss_integrator ! Initialize the uncoupled resolved dynamics

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

buf_y1=0.d0
buf_f1=0.d0
buf_f0=0.d0

print*, 'Initializing the integrator ...'
CALL init_sigma(mult,q1fill)
CALL init_noise
CALL init_g
```

7.14.2.5 subroutine rk2_mtv_integrator::init_noise ( ) [private]

Routine to initialize the noise vectors and buffers.

Definition at line 69 of file rk2_MTV_integrator.f90.

```
INTEGER :: allocstat
ALLOCATE(dw(0:ndim), dwmult(0:ndim), stat=allocstat)
IF (allocstat /= 0) stop "*** Not enough memory ! ***"

ALLOCATE(dwar(0:ndim),dwau(0:ndim),dwor(0:ndim),dwou(0:ndim),
stat=allocstat)
IF (allocstat /= 0) stop "*** Not enough memory ! ***"

ALLOCATE(anoise(0:ndim), noise(0:ndim), noisemult(0:ndim), stat=allocstat)
IF (allocstat /= 0) stop "*** Not enough memory ! ***"

dw=0.d0
dwmult=0.d0

dwar=0.d0
dwor=0.d0
dwau=0.d0
dwou=0.d0

anoise=0.d0
noise=0.d0
noisemult=0.d0

compute_mult=((q1fill).OR.(mult))

```

7.14.2.6 subroutine, public rk2_mtv_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), intent(in) *dtn*, real(kind=8), dimension(0:ndim), intent(out) *res*, real(kind=8), dimension(0:ndim), intent(out) *tend*)

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

Parameters

y	Initial point.
t	Actual integration time
dt	Integration timestep.
dtn	Stochastic integration timestep (normally square-root of dt).
res	Final point after the step.
tend	Partial or full tendencies used to perform the step (used for debugging).

Definition at line 124 of file rk2_MTV_integrator.f90.

```
124   REAL(KIND=8), DIMENSION(0:ndim), INTENT(IN) :: y
125   REAL(KIND=8), INTENT(INOUT) :: t
126   REAL(KIND=8), INTENT(IN) :: dt,dtn
127   REAL(KIND=8), DIMENSION(0:ndim), INTENT(OUT) :: res,tend
128
129   CALL compg(y)
130
131   CALL stoch_atm_res_vec(dwar)
132   CALL stoch_oc_res_vec(dwor)
133   anoise=q_ar*dwar+q_or*dwor
134   CALL stoch_vec(dw)
135   IF (compute_mult) CALL stoch_vec(dwmult)
136   noise(1:ndim)=matmul(sig2,dw(1:ndim))
137   IF ((mult).and.(mod(t,mnuti)<dt)) CALL compute_mult_sigma(y)
138   IF (compute_mult) noisemult(1:ndim)=matmul(sig1,dwmult(1:ndim))
139
140   CALL tendencies(t,y,buf_f0)
141   buf_y1 = y+dt*(buf_f0+g)+(anoise+sq2*(noise+noisemult))*dtn
142
143   buf_f1=g
144   CALL compg(buf_y1)
145   g=0.5*(g+buf_f1)
146
147   IF ((mult).and.(mod(t,mnuti)<dt)) CALL compute_mult_sigma(buf_y1)
148   IF (compute_mult) THEN
149      buf_f1(1:ndim)=matmul(sig1,dwmult(1:ndim))
```

```
        noisemult(1:ndim)=0.5*(noisemult(1:ndim)+buf_f1(1:ndim))
     ENDIF


     CALL tendencies(t,buf_y1,buf_f1)
     buf_f0=0.5*(buf_f0+buf_f1)
     res=y+dt*(buf_f0+g)+(anoise+sq2*(noise+noisemult))*dtn
     ! tend=G+sq2*(noise+noisemult)/dtn
     tend=sq2*noisemult/dtn
     t=t+dt

```

7.14.3 Variable Documentation

7.14.3.1 real(kind=8), dimension(:), allocatable rk2_mtv_integrator::anoise [private]

Definition at line 33 of file rk2_MTV_integrator.f90.

```
33    REAL(KIND=8), DIMENSION(:), ALLOCATABLE :: anoise,noise          !< Additive noise term
```

7.14.3.2 real(kind=8), dimension(:), allocatable rk2_mtv_integrator::buf_f0 [private]

Definition at line 30 of file rk2_MTV_integrator.f90.

7.14.3.3 real(kind=8), dimension(:), allocatable rk2_mtv_integrator::buf_f1 [private]

Integration buffers.

Definition at line 30 of file rk2_MTV_integrator.f90.

7.14.3.4 real(kind=8), dimension(:), allocatable rk2_mtv_integrator::buf_g [private]

Buffer for the G term computation.

Definition at line 36 of file rk2_MTV_integrator.f90.

```
36    REAL(KIND=8), DIMENSION(:), ALLOCATABLE :: buf_g                 !< Buffer for the G term computation
```

7.14.3.5 real(kind=8), dimension(:), allocatable rk2_mtv_integrator::buf_y1 [private]

Definition at line 30 of file rk2_MTV_integrator.f90.

```
30    REAL(KIND=8), DIMENSION(:), ALLOCATABLE :: buf_y1,buf_f0,buf_f1 !< Integration buffers
```

7.14.3.6 logical rk2_mtv_integrator::compute_mult [private]

Logical indicating if the Gaussian noise for the multiplicative noise must be computed.

Definition at line 40 of file rk2_MTV_integrator.f90.

```
LOGICAL :: compute_mult                                   !< Logical indicating if the Gaussian
  noise for the multiplicative noise must be computed
```

7.14.3.7 real(kind=8), dimension(:), allocatable rk2_mtv_integrator::dw [private]

Definition at line 31 of file rk2_MTV_integrator.f90.

```
REAL(KIND=8), DIMENSION(:), ALLOCATABLE :: dw, dwmult          !< Standard gaussian noise buffers
```

7.14.3.8 real(kind=8), dimension(:), allocatable rk2_mtv_integrator::dwar [private]

Definition at line 32 of file rk2_MTV_integrator.f90.

```
REAL(KIND=8), DIMENSION(:), ALLOCATABLE :: dwar,dwau,dwor,dwou  !< Standard gaussian noise buffers
```

7.14.3.9 real(kind=8), dimension(:), allocatable rk2_mtv_integrator::dwau [private]

Definition at line 32 of file rk2_MTV_integrator.f90.

7.14.3.10 real(kind=8), dimension(:), allocatable rk2_mtv_integrator::dwmult [private]

Standard gaussian noise buffers.

Definition at line 31 of file rk2_MTV_integrator.f90.

7.14.3.11 real(kind=8), dimension(:), allocatable rk2_mtv_integrator::dwor [private]

Definition at line 32 of file rk2_MTV_integrator.f90.

7.14.3.12 real(kind=8), dimension(:), allocatable rk2_mtv_integrator::dwou [private]

Standard gaussian noise buffers.

Definition at line 32 of file rk2_MTV_integrator.f90.

7.14.3.13 real(kind=8), dimension(:), allocatable rk2_mtv_integrator::g [private]

G term of the MTV tendencies.

Definition at line 35 of file rk2_MTV_integrator.f90.

```
35    REAL(KIND=8), DIMENSION(:), ALLOCATABLE :: g                    !< G term of the MTV tendencies
```

7.14.3.14 logical rk2_mtv_integrator::mult [private]

Logical indicating if the sigma1 matrix must be computed for every state change.

Definition at line 38 of file rk2_MTV_integrator.f90.

```
38    LOGICAL :: mult                                                 !< Logical indicating if the sigma1
       matrix must be computed for every state change
```

7.14.3.15 real(kind=8), dimension(:), allocatable rk2_mtv_integrator::noise [private]

Additive noise term.

Definition at line 33 of file rk2_MTV_integrator.f90.

7.14.3.16 real(kind=8), dimension(:), allocatable rk2_mtv_integrator::noisemult [private]

Multiplicative noise term.

Definition at line 34 of file rk2_MTV_integrator.f90.

```
34    REAL(KIND=8), DIMENSION(:), ALLOCATABLE :: noisemult            !< Multiplicative noise term
```

7.14.3.17 logical rk2_mtv_integrator::q1fill [private]

Logical indicating if the matrix Q1 is non-zero.

Definition at line 39 of file rk2_MTV_integrator.f90.

```
39    LOGICAL :: q1fill                                               !< Logical indicating if the matrix Q1 is
       non-zero
```

7.14.3.18 real(kind=8), parameter rk2_mtv_integrator::sq2 = sqrt(2.D0) [private]

Hard coded square root of 2.

Definition at line 42 of file rk2_MTV_integrator.f90.

```
42    REAL(KIND=8), PARAMETER :: sq2 = sqrt(2.d0)                     !< Hard coded square root of 2
```

7.15 rk2_ss_integrator Module Reference

Module with the stochastic uncoupled resolved nonlinear and tangent linear rk2 dynamics integration routines.

Functions/Subroutines

- subroutine, public init_ss_integrator

  Subroutine to initialize the uncoupled resolved rk2 integrator.
- subroutine, public tendencies (t, y, res)

  Routine computing the tendencies of the uncoupled resolved model.
- subroutine, public tl_tendencies (t, y, ys, res)

  Tendencies for the tangent linear model of the uncoupled resolved dynamics in point ystar for perturbation deltay.
- subroutine, public ss_step (y, ys, t, dt, dtn, res)

  Routine to perform a stochastic integration step of the unresolved uncoupled dynamics (Heun algorithm). The incremented time is returned.
- subroutine, public ss_tl_step (y, ys, t, dt, dtn, res)

  Routine to perform a stochastic integration step of the unresolved uncoupled tangent linear dynamics (Heun algorithm). The incremented time is returned.

Variables

- real(kind=8), dimension(:), allocatable dwar
- real(kind=8), dimension(:), allocatable dwor

  Standard gaussian noise buffers.
- real(kind=8), dimension(:), allocatable anoise

  Additive noise term.
- real(kind=8), dimension(:), allocatable buf_y1
- real(kind=8), dimension(:), allocatable buf_f0
- real(kind=8), dimension(:), allocatable buf_f1

  Integration buffers.

7.15.1 Detailed Description

Module with the stochastic uncoupled resolved nonlinear and tangent linear rk2 dynamics integration routines.

Copyright

Remarks

This module actually contains the Heun algorithm routines.

7.15.2 Function/Subroutine Documentation

7.15.2.1 subroutine, public rk2_ss_integrator::init_ss_integrator ( )

Subroutine to initialize the uncoupled resolved rk2 integrator.

Definition at line 40 of file rk2_ss_integrator.f90.

```
40      INTEGER :: allocstat
41
42      ALLOCATE(buf_y1(0:ndim),buf_f0(0:ndim),buf_f1(0:ndim),stat=allocstat)
43      IF (allocstat /= 0) stop "*** Not enough memory ! ***"
44
45      ALLOCATE(anoise(0:ndim),stat=allocstat)
46      IF (allocstat /= 0) stop "*** Not enough memory ! ***"
47
48      ALLOCATE(dwar(0:ndim),dwor(0:ndim),stat=allocstat)
49      IF (allocstat /= 0) stop "*** Not enough memory ! ***"
50
51      dwar=0.d0
52      dwor=0.d0
53
```

7.15.2.2 subroutine, public rk2_ss_integrator::ss_step (real(kind=8), dimension(0:ndim), intent(in) *y*, real(kind=8), dimension(0:ndim), intent(in) *ys*, real(kind=8), intent(inout) *t*, real(kind=8), intent(in) *dt*, real(kind=8), intent(in) *dtn*, real(kind=8), dimension(0:ndim), intent(out) *res*)

Routine to perform a stochastic integration step of the unresolved uncoupled dynamics (Heun algorithm). The incremented time is returned.

Parameters

y	Initial point.
ys	Dummy argument for compatibility.
t	Actual integration time
dt	Integration timestep.
dtn	Stochastic integration timestep (normally square-root of dt).
res	Final point after the step.

Definition at line 92 of file rk2_ss_integrator.f90.

```
92      REAL(KIND=8), DIMENSION(0:ndim), INTENT(IN) :: y,ys
93      REAL(KIND=8), INTENT(INOUT) :: t
94      REAL(KIND=8), INTENT(IN) :: dt,dtn
95      REAL(KIND=8), DIMENSION(0:ndim), INTENT(OUT) :: res
96
97      CALL stoch_atm_res_vec(dwar)
98      CALL stoch_oc_res_vec(dwor)
99      anoise=(q_ar*dwar+q_or*dwor)*dtn
100      CALL tendencies(t,y,buf_f0)
101      buf_y1 = y+dt*buf_f0+anoise
102      CALL tendencies(t,buf_y1,buf_f1)
103      res=y+0.5*(buf_f0+buf_f1)*dt+anoise
104      t=t+dt
```

7.15.2.3 subroutine, public rk2_ss_integrator::ss_tl_step (real(kind=8), dimension(0:ndim), intent(in) *y*, real(kind=8), dimension(0:ndim), intent(in) *ys*, real(kind=8), intent(inout) *t*, real(kind=8), intent(in) *dt*, real(kind=8), intent(in) *dtn*, real(kind=8), dimension(0:ndim), intent(out) *res*)

Routine to perform a stochastic integration step of the unresolved uncoupled tangent linear dynamics (Heun algorithm). The incremented time is returned.

Parameters

y	Initial point.
ys	point in trajectory to which the tangent space belongs.
t	Actual integration time
dt	Integration timestep.
dtn	Stochastic integration timestep (normally square-root of dt).
res	Final point after the step.

Definition at line 117 of file rk2_ss_integrator.f90.

```
117     REAL(KIND=8), DIMENSION(0:ndim), INTENT(IN) :: y,ys
118     REAL(KIND=8), INTENT(INOUT) :: t
119     REAL(KIND=8), INTENT(IN) :: dt,dtn
120     REAL(KIND=8), DIMENSION(0:ndim), INTENT(OUT) :: res
121
122     CALL stoch_atm_res_vec(dwar)
123     CALL stoch_oc_res_vec(dwor)
124     anoise=(q_ar*dwar+q_or*dwor)*dtn
125     CALL tl_tendencies(t,y,ys,buf_f0)
126     buf_y1 = y+dt*buf_f0+anoise
127     CALL tl_tendencies(t,buf_y1,ys,buf_f1)
128     res=y+0.5*(buf_f0+buf_f1)*dt+anoise
129     t=t+dt
```

7.15.2.4 subroutine, public rk2_ss_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*)

Routine computing the tendencies of the uncoupled resolved model.

Parameters

t	Time at which the tendencies have to be computed. Actually not needed for autonomous systems.
y	Point at which the tendencies have to be computed.
res	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 63 of file rk2_ss_integrator.f90.

```
63     REAL(KIND=8), INTENT(IN) :: t
64     REAL(KIND=8), DIMENSION(0:ndim), INTENT(IN) :: y
65     REAL(KIND=8), DIMENSION(0:ndim), INTENT(OUT) :: res
66     CALL sparse_mul3(ss_tensor, y, y, res)
```

7.15.2.5 subroutine, public rk2_ss_integrator::tl_tendencies (real(kind=8), intent(in) *t*, real(kind=8), dimension(0:ndim), intent(in) *y*, real(kind=8), dimension(0:ndim), intent(in) *ys*, real(kind=8), dimension(0:ndim), intent(out) *res*)

Tendencies for the tangent linear model of the uncoupled resolved dynamics in point ystar for perturbation deltay.

Parameters

t	time
y	point of the tangent space at which the tendencies have to be computed.
ys	point in trajectory to which the tangent space belongs.
res	vector to store the result.

Definition at line 76 of file rk2_ss_integrator.f90.

```
76      REAL(KIND=8), INTENT(IN) :: t
77      REAL(KIND=8), DIMENSION(0:ndim), INTENT(IN) :: y,ys
78      REAL(KIND=8), DIMENSION(0:ndim), INTENT(OUT) :: res
79      CALL sparse_mul3(ss_tl_tensor, y, ys, res)
```

7.15.3 Variable Documentation

7.15.3.1 real(kind=8), dimension(:), allocatable rk2_ss_integrator::anoise [private]

Additive noise term.

Definition at line 30 of file rk2_ss_integrator.f90.

```
30    REAL(KIND=8), DIMENSION(:), ALLOCATABLE :: anoise !< Additive noise term
```

7.15.3.2 real(kind=8), dimension(:), allocatable rk2_ss_integrator::buf_f0 [private]

Definition at line 32 of file rk2_ss_integrator.f90.

7.15.3.3 real(kind=8), dimension(:), allocatable rk2_ss_integrator::buf_f1 [private]

Integration buffers.

Definition at line 32 of file rk2_ss_integrator.f90.

7.15.3.4 real(kind=8), dimension(:), allocatable rk2_ss_integrator::buf_y1 [private]

Definition at line 32 of file rk2_ss_integrator.f90.

```
32    REAL(KIND=8), DIMENSION(:), ALLOCATABLE :: buf_y1,buf_f0,buf_f1 !< Integration buffers
```

7.15.3.5 real(kind=8), dimension(:), allocatable rk2_ss_integrator::dwar [private]

Definition at line 28 of file rk2_ss_integrator.f90.

```
28  REAL(KIND=8), DIMENSION(:), ALLOCATABLE :: dwar,dwor !< Standard gaussian noise buffers
```

7.15.3.6 real(kind=8), dimension(:), allocatable rk2_ss_integrator::dwor [private]

Standard gaussian noise buffers.

Definition at line 28 of file rk2_ss_integrator.f90.

7.16 rk2_stoch_integrator Module Reference

Module with the stochastic rk2 integration routines.

Functions/Subroutines

- subroutine, public init_integrator (force)
  Subroutine to initialize the integrator.
- subroutine tendencies (t, y, res)
  Routine computing the tendencies of the selected model.
- subroutine, public step (y, t, dt, dtn, res, tend)
  Routine to perform a stochastic step of the selected dynamics (Heun algorithm). The incremented time is returned.

Variables

- real(kind=8), dimension(:), allocatable dwar
- real(kind=8), dimension(:), allocatable dwau
- real(kind=8), dimension(:), allocatable dwor
- real(kind=8), dimension(:), allocatable dwou
  Standard gaussian noise buffers.
- real(kind=8), dimension(:), allocatable buf_y1
- real(kind=8), dimension(:), allocatable buf_f0
- real(kind=8), dimension(:), allocatable buf_f1
  Integration buffers.
- real(kind=8), dimension(:), allocatable anoise
  Additive noise term.
- type(coolist), dimension(:), allocatable int_tensor
  Dummy tensor that will hold the tendencies tensor.

7.16.1 Detailed Description

Module with the stochastic rk2 integration routines.

Copyright

2017 Jonathan Demaeyer. See LICENSE.txt for license information.

Remarks

This module actually contains the Heun algorithm routines. There are four modes for this integrator:

- full: use the full dynamics
- ures: use the intrinsic unresolved dynamics
- qfst: use the quadratic terms of the unresolved tendencies
- reso: use the resolved dynamics alone

7.16.2 Function/Subroutine Documentation

7.16.2.1 subroutine, public rk2_stoch_integrator::init_integrator (character∗4, intent(in), optional *force*)

Subroutine to initialize the integrator.

Parameters

force	Parameter to force the mode of the integrator

Definition at line 48 of file rk2_stoch_integrator.f90.

```
48      INTEGER :: allocstat
49      CHARACTER*4, INTENT(IN), OPTIONAL :: force
50      CHARACTER*4 :: test
51
52      ALLOCATE(buf_y1(0:ndim),buf_f0(0:ndim),buf_f1(0:ndim),stat=allocstat)
53      IF (allocstat /= 0) stop "*** Not enough memory ! ***"
54
55      ALLOCATE(anoise(0:ndim),stat=allocstat)
56      IF (allocstat /= 0) stop "*** Not enough memory ! ***"
57
58      ALLOCATE(dwar(0:ndim),dwau(0:ndim),dwor(0:ndim),dwou(0:ndim),
       stat=allocstat)
59      IF (allocstat /= 0) stop "*** Not enough memory ! ***"
60
61      ALLOCATE(int_tensor(ndim),stat=allocstat)
62      IF (allocstat /= 0) stop "*** Not enough memory ! ***"
63
64      dwar=0.d0
65      dwor=0.d0
66      dwau=0.d0
67      dwou=0.d0
68
69      IF (PRESENT(force)) THEN
70         test=force
71      ELSE
72         test=mode
73      ENDIF
74
75      SELECT CASE (test)
76      CASE('full')
77         CALL copy_tensor(aotensor,int_tensor)
78      CASE('ures')
79         CALL copy_tensor(ff_tensor,int_tensor)
```

```
    CASE('qfst')
       CALL copy_tensor(byyy,int_tensor)
    CASE('reso')
       CALL copy_tensor(ss_tensor,int_tensor)
    CASE DEFAULT
       stop '*** MODE variable not properly defined ***'
    END SELECT

```

7.16.2.2 subroutine, public rk2_stoch_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), intent(in) *dtn*, real(kind=8), dimension(0:ndim), intent(out) *res*, real(kind=8), dimension(0:ndim), intent(out) *tend*)

Routine to perform a stochastic step of the selected dynamics (Heun algorithm). The incremented time is returned.

Parameters

y	Initial point.
t	Actual integration time
dt	Integration timestep.
dtn	Stochastic integration timestep (normally square-root of dt).
res	Final point after the step.
tend	Partial or full tendencies used to perform the step (used for debugging).

Definition at line 112 of file rk2_stoch_integrator.f90.

```
112     REAL(KIND=8), DIMENSION(0:ndim), INTENT(IN) :: y
113     REAL(KIND=8), INTENT(INOUT) :: t
114     REAL(KIND=8), INTENT(IN) :: dt,dtn
115     REAL(KIND=8), DIMENSION(0:ndim), INTENT(OUT) :: res,tend
116
117     CALL stoch_atm_res_vec(dwar)
118     CALL stoch_atm_unres_vec(dwau)
119     CALL stoch_oc_res_vec(dwor)
120     CALL stoch_oc_unres_vec(dwou)
121     anoise=(q_ar*dwar+q_au*dwau+q_or*dwor+q_ou*dwou)*dtn
122     CALL tendencies(t,y,buf_f0)
123     CALL sparse_mul3(int_tensor,y,y,tend)
124     buf_y1 = y+dt*buf_f0+anoise
125     CALL sparse_mul3(int_tensor,buf_y1,buf_y1,buf_f1)
126     tend=0.5*(tend+buf_f1)
127     CALL tendencies(t,buf_y1,buf_f1)
128     res=y+0.5*(buf_f0+buf_f1)*dt+anoise
129     t=t+dt
```

7.16.2.3 subroutine rk2_stoch_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 selected model.

Parameters

t	Time at which the tendencies have to be computed. Actually not needed for autonomous systems.
y	Point at which the tendencies have to be computed.
res	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 97 of file rk2_stoch_integrator.f90.

```
97      REAL(KIND=8), INTENT(IN) :: t
98      REAL(KIND=8), DIMENSION(0:ndim), INTENT(IN) :: y
99      REAL(KIND=8), DIMENSION(0:ndim), INTENT(OUT) :: res
100      CALL sparse_mul3(int_tensor, y, y, res)
```

7.16.3 Variable Documentation

7.16.3.1 real(kind=8), dimension(:), allocatable rk2_stoch_integrator::anoise [private]

Additive noise term.

Definition at line 37 of file rk2_stoch_integrator.f90.

```
37    REAL(KIND=8), DIMENSION(:), ALLOCATABLE :: anoise !< Additive noise term
```

7.16.3.2 real(kind=8), dimension(:), allocatable rk2_stoch_integrator::buf_f0 [private]

Definition at line 35 of file rk2_stoch_integrator.f90.

7.16.3.3 real(kind=8), dimension(:), allocatable rk2_stoch_integrator::buf_f1 [private]

Integration buffers.

Definition at line 35 of file rk2_stoch_integrator.f90.

7.16.3.4 real(kind=8), dimension(:), allocatable rk2_stoch_integrator::buf_y1 [private]

Definition at line 35 of file rk2_stoch_integrator.f90.

```
35    REAL(KIND=8), DIMENSION(:), ALLOCATABLE :: buf_y1,buf_f0,buf_f1 !< Integration buffers
```

7.16.3.5 real(kind=8), dimension(:), allocatable rk2_stoch_integrator::dwar [private]

Definition at line 33 of file rk2_stoch_integrator.f90.

```
33    REAL(KIND=8), DIMENSION(:), ALLOCATABLE :: dwar,dwau,dwor,dwou !< Standard gaussian noise buffers
```

7.16.3.6 real(kind=8), dimension(:), allocatable rk2_stoch_integrator::dwau [private]

Definition at line 33 of file rk2_stoch_integrator.f90.

7.16.3.7 real(kind=8), dimension(:), allocatable rk2_stoch_integrator::dwor [private]

Definition at line 33 of file rk2_stoch_integrator.f90.

7.16.3.8 real(kind=8), dimension(:), allocatable rk2_stoch_integrator::dwou [private]

Standard gaussian noise buffers.

Definition at line 33 of file rk2_stoch_integrator.f90.

7.16.3.9 type(coolist), dimension(:), allocatable rk2_stoch_integrator::int_tensor [private]

Dummy tensor that will hold the tendencies tensor.

Definition at line 39 of file rk2_stoch_integrator.f90.

```
39   TYPE(coolist), DIMENSION(:), ALLOCATABLE :: int_tensor !< Dummy tensor that will hold the
       tendencies tensor
```

7.17 rk2_wl_integrator Module Reference

Module with the WL rk2 integration routines.

Functions/Subroutines

- subroutine, public init_integrator

  Subroutine that initialize the MARs, the memory unit and the integration buffers.
- subroutine compute_m1 (y)

  Routine to compute the M_1 term.
- subroutine compute_m2 (y)

  Routine to compute the M_2 term.
- subroutine, public step (y, t, dt, dtn, res, tend)

  Routine to perform an integration step (Heun algorithm) of the WL system. The incremented time is returned.
- subroutine, public full_step (y, t, dt, dtn, res)

  Routine to perform an integration step (Heun algorithm) of the full stochastic system. The incremented time is returned.

Variables

- real(kind=8), dimension(:), allocatable buf_y1
- real(kind=8), dimension(:), allocatable buf_f0
- real(kind=8), dimension(:), allocatable buf_f1

  Integration buffers.
- real(kind=8), dimension(:), allocatable buf_m2
- real(kind=8), dimension(:), allocatable buf_m1
- real(kind=8), dimension(:), allocatable buf_m3
- real(kind=8), dimension(:), allocatable buf_m
- real(kind=8), dimension(:), allocatable buf_m3s

  Dummy buffers holding the terms /f$M_i.
- real(kind=8), dimension(:), allocatable anoise

  Additive noise term.
- real(kind=8), dimension(:), allocatable dwar
- real(kind=8), dimension(:), allocatable dwau
- real(kind=8), dimension(:), allocatable dwor
- real(kind=8), dimension(:), allocatable dwou

  Standard gaussian noise buffers.
- real(kind=8), dimension(:,:), allocatable x1

  Buffer holding the subsequent states of the first MAR.
- real(kind=8), dimension(:,:), allocatable x2

  Buffer holding the subsequent states of the second MAR.

7.17.1 Detailed Description

Module with the WL rk2 integration routines.

Copyright

2017 Jonathan Demaeyer. See LICENSE.txt for license information.

Remarks

This module actually contains the Heun algorithm routines.

7.17.2 Function/Subroutine Documentation

7.17.2.1 subroutine rk2_wl_integrator::compute_m1 (real(kind=8), dimension(0:ndim), intent(in) *y*) [private]

Routine to compute the M_1 term.

Parameters

y	Present state of the WL system

Definition at line 106 of file rk2_WL_integrator.f90.

```
106     REAL(KIND=8), DIMENSION(0:ndim), INTENT(IN) :: y
```

```
107     buf_m1=0.d0
108     IF (m12def) CALL sparse_mul2_k(m12, y, buf_m1)
109     buf_m1=buf_m1+m1tot
```

7.17.2.2 subroutine rk2_wl_integrator::compute_m2 (real(kind=8), dimension(0:ndim), intent(in) *y*) [private]

Routine to compute the M_2 term.

Parameters

y	Present state of the WL system

Definition at line 115 of file rk2_WL_integrator.f90.

```
115     REAL(KIND=8), DIMENSION(0:ndim), INTENT(IN) :: y
116     buf_m=0.d0
117     buf_m2=0.d0
118     IF (m21def) CALL sparse_mul3(m21, y, x1(0:ndim,1), buf_m)
119     IF (m22def) CALL sparse_mul3(m22, x2(0:ndim,1), x2(0:ndim,1), buf_m2)
120     buf_m2=buf_m2+buf_m
```

7.17.2.3 subroutine, public rk2_wl_integrator::full_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), intent(in) *dtn*, real(kind=8), dimension(0:ndim), intent(out) *res*)

Routine to perform an integration step (Heun algorithm) of the full stochastic system. The incremented time is returned.

Parameters

y	Initial point.
t	Actual integration time
dt	Integration timestep.
dtn	Stochastoc integration timestep (normally square-root of dt).
res	Final point after the step.

Definition at line 185 of file rk2_WL_integrator.f90.

```
185     REAL(KIND=8), DIMENSION(0:ndim), INTENT(IN) :: y
186     REAL(KIND=8), INTENT(INOUT) :: t
187     REAL(KIND=8), INTENT(IN) :: dt,dtn
188     REAL(KIND=8), DIMENSION(0:ndim), INTENT(OUT) :: res
189     CALL stoch_atm_res_vec(dwar)
190     CALL stoch_atm_unres_vec(dwau)
191     CALL stoch_oc_res_vec(dwor)
192     CALL stoch_oc_unres_vec(dwou)
193     anoise=(q_ar*dwar+q_au*dwau+q_or*dwor+q_ou*dwou)*dtn
194     CALL sparse_mul3(aotensor,y,y,buf_f0)
195     buf_y1 = y+dt*buf_f0+anoise
196     CALL sparse_mul3(aotensor,buf_y1,buf_y1,buf_f1)
197     res=y+0.5*(buf_f0+buf_f1)*dt+anoise
198     t=t+dt
```

7.17.2.4 subroutine, public rk2_wl_integrator::init_integrator ( )

Subroutine that initialize the MARs, the memory unit and the integration buffers.

Definition at line 44 of file rk2_WL_integrator.f90.

```
44      INTEGER :: allocstat,i
45
46      CALL init_ss_integrator
47
48      print*, 'Initializing the integrator ...'
49
50      IF (mode.ne.'ures') THEN
51         print*, '*** Mode set to ',mode,' in stoch_params.nml ***'
52         print*, '*** WL configuration only support unresolved mode ***'
53         stop "*** Please change to 'ures' and perform the configuration again ! ***"
54      ENDIF
55
56      ALLOCATE(buf_y1(0:ndim),buf_f0(0:ndim),buf_f1(0:ndim),stat=allocstat)
57      IF (allocstat /= 0) stop "*** Not enough memory ! ***"
58
59      ALLOCATE(buf_m1(0:ndim), buf_m2(0:ndim), buf_m3(0:ndim), buf_m(0:
       ndim), buf_m3s(0:ndim), stat=allocstat)
60      IF (allocstat /= 0) stop "*** Not enough memory ! ***"
61
62      ALLOCATE(dwar(0:ndim),dwau(0:ndim),dwor(0:ndim),dwou(0:ndim),
       stat=allocstat)
63      IF (allocstat /= 0) stop "*** Not enough memory ! ***"
64
65      ALLOCATE(anoise(0:ndim), stat=allocstat)
66      IF (allocstat /= 0) stop "*** Not enough memory ! ***"
67
68      buf_y1=0.d0
69      buf_f1=0.d0
70      buf_f0=0.d0
71
72      dwar=0.d0
73      dwor=0.d0
74      dwau=0.d0
75      dwou=0.d0
76
77      buf_m1=0.d0
78      buf_m2=0.d0
79      buf_m3=0.d0
80      buf_m3s=0.d0
81      buf_m=0.d0
82
83      print*, 'Initializing the MARs ...'
84
85      CALL init_mar
86
87      ALLOCATE(x1(0:ndim,ms), x2(0:ndim,ms), stat=allocstat)
88
89      x1=0.d0
90      DO i=1,50000
91         CALL mar_step(x1)
92      ENDDO
93
94      x2=0.d0
95      DO i=1,50000
96         CALL mar_step(x2)
97      ENDDO
98
99      CALL init_memory
100
```

7.17.2.5 subroutine, public rk2_wl_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), intent(in) *dtn*, real(kind=8), dimension(0:ndim), intent(out) *res*, real(kind=8), dimension(0:ndim), intent(out) *tend*)

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

Parameters

y	Initial point.
t	Actual integration time
dt	Integration timestep.
dtn	Stochastic integration timestep (normally square-root of dt).
res	Final point after the step.
tend	Partial or full tendencies used to perform the step (used for debugging).

Definition at line 132 of file rk2_WL_integrator.f90.

```
132       REAL(KIND=8), DIMENSION(0:ndim), INTENT(IN) :: y
133       REAL(KIND=8), INTENT(INOUT) :: t
134       REAL(KIND=8), INTENT(IN) :: dt,dtn
135       REAL(KIND=8), DIMENSION(0:ndim), INTENT(OUT) :: res,tend
136       INTEGER :: i
137 
138       IF (mod(t,muti)<dt) THEN
139          CALL compute_m3(y,dts,dtsn,.true.,.true.,.true.,muti/2,buf_m3s)
140          buf_m3=buf_m3s
141          DO i=1,1
142             CALL compute_m3(y,dts,dtsn,.false.,.true.,.true.,muti/2,buf_m3s)
143             buf_m3=buf_m3+buf_m3s
144          ENDDO
145          !DO i=1,2
146          !   CALL compute_M3(y,dts,dtsn,.false.,.true.,.true.,muti/2,buf_M3s)
147          !   buf_M3=buf_M3+buf_M3s
148          !ENDDO
149          buf_m3=buf_m3/2
150       ENDIF
151 
152 
153       CALL stoch_atm_res_vec(dwar)
154       CALL stoch_oc_res_vec(dwor)
155       anoise=(q_ar*dwar+q_or*dwor)*dtn
156 
157       CALL tendencies(t,y,buf_f0)
158       CALL mar_step(x1)
159       CALL mar_step(x2)
160       CALL compute_m1(y)
161       CALL compute_m2(y)
162       buf_f0= buf_f0+buf_m1+buf_m2+buf_m3
163       buf_y1 = y+dt*buf_f0+anoise
164 
165       CALL tendencies(t+dt,buf_y1,buf_f1)
166       CALL compute_m1(buf_y1)
167       CALL compute_m2(buf_y1)
168       !IF (mod(t,muti)<dt) CALL compute_M3(buf_y1,dts,dtsn,.false.,.true.,buf_M3)
169 
170       buf_f0=0.5*(buf_f0+buf_f1+buf_m1+buf_m2+buf_m3)
171       res=y+dt*buf_f0+anoise
172 
173       tend=buf_m3
174       t=t+dt
175 
```

7.17.3 Variable Documentation

7.17.3.1 real(kind=8), dimension(:), allocatable rk2_wl_integrator::anoise [private]

Additive noise term.

Definition at line 33 of file rk2_WL_integrator.f90.

```
33    REAL(KIND=8), DIMENSION(:), ALLOCATABLE :: anoise          !< Additive noise term
```

7.17.3.2 real(kind=8), dimension(:), allocatable rk2_wl_integrator::buf_f0 [private]

Definition at line 31 of file rk2_WL_integrator.f90.

7.17.3.3 real(kind=8), dimension(:), allocatable rk2_wl_integrator::buf_f1 [private]

Integration buffers.

Definition at line 31 of file rk2_WL_integrator.f90.

7.17.3.4 real(kind=8), dimension(:), allocatable rk2_wl_integrator::buf_m [private]

Definition at line 32 of file rk2_WL_integrator.f90.

7.17.3.5 real(kind=8), dimension(:), allocatable rk2_wl_integrator::buf_m1 [private]

Definition at line 32 of file rk2_WL_integrator.f90.

7.17.3.6 real(kind=8), dimension(:), allocatable rk2_wl_integrator::buf_m2 [private]

Definition at line 32 of file rk2_WL_integrator.f90.

```
32   REAL(KIND=8), DIMENSION(:), ALLOCATABLE :: buf_m2,buf_m1,buf_m3,buf_m,buf_m3s !< Dummy buffers holding
       the terms /f$M_i\f$ of the parameterization
```

7.17.3.7 real(kind=8), dimension(:), allocatable rk2_wl_integrator::buf_m3 [private]

Definition at line 32 of file rk2_WL_integrator.f90.

7.17.3.8 real(kind=8), dimension(:), allocatable rk2_wl_integrator::buf_m3s [private]

Dummy buffers holding the terms /f$M_i.

Definition at line 32 of file rk2_WL_integrator.f90.

7.17.3.9 real(kind=8), dimension(:), allocatable rk2_wl_integrator::buf_y1 [private]

Definition at line 31 of file rk2_WL_integrator.f90.

```
31   REAL(KIND=8), DIMENSION(:), ALLOCATABLE :: buf_y1,buf_f0,buf_f1 !< Integration buffers
```

7.17.3.10 real(kind=8), dimension(:), allocatable rk2_wl_integrator::dwar [private]

Definition at line 34 of file rk2_WL_integrator.f90.

```
34    REAL(KIND=8), DIMENSION(:), ALLOCATABLE :: dwar,dwau,dwor,dwou  !< Standard gaussian noise buffers
```

7.17.3.11 real(kind=8), dimension(:), allocatable rk2_wl_integrator::dwau [private]

Definition at line 34 of file rk2_WL_integrator.f90.

7.17.3.12 real(kind=8), dimension(:), allocatable rk2_wl_integrator::dwor [private]

Definition at line 34 of file rk2_WL_integrator.f90.

7.17.3.13 real(kind=8), dimension(:), allocatable rk2_wl_integrator::dwou [private]

Standard gaussian noise buffers.

Definition at line 34 of file rk2_WL_integrator.f90.

7.17.3.14 real(kind=8), dimension(:,:), allocatable rk2_wl_integrator::x1 [private]

Buffer holding the subsequent states of the first MAR.

Definition at line 36 of file rk2_WL_integrator.f90.

```
36    REAL(KIND=8), DIMENSION(:,:), ALLOCATABLE :: x1 !< Buffer holding the subsequent states of the first MAR
```

7.17.3.15 real(kind=8), dimension(:,:), allocatable rk2_wl_integrator::x2 [private]

Buffer holding the subsequent states of the second MAR.

Definition at line 37 of file rk2_WL_integrator.f90.

```
37    REAL(KIND=8), DIMENSION(:,:), ALLOCATABLE :: x2 !< Buffer holding the subsequent states of the second MAR
```

7.18 sf_def Module Reference

Module to select the resolved-unresolved components.

Functions/Subroutines

- subroutine, public load_sf

    Subroutine to load the unresolved variable defintion vector SF from SF.nml if it exists. If it does not, then write SF.nml with no unresolved variables specified (null vector).

Variables

- logical exists

    Boolean to test for file existence.
- integer, dimension(:), allocatable, public sf

    Unresolved variable definition vector.
- integer, dimension(:), allocatable, public ind
- integer, dimension(:), allocatable, public rind

    Unresolved reduction indices.
- integer, dimension(:), allocatable, public sl_ind
- integer, dimension(:), allocatable, public sl_rind

    Resolved reduction indices.
- integer, public n_unres

    Number of unresolved variables.
- integer, public n_res

    Number of resolved variables.
- integer, dimension(:,:), allocatable, public bar
- integer, dimension(:,:), allocatable, public bau
- integer, dimension(:,:), allocatable, public bor
- integer, dimension(:,:), allocatable, public bou

    Filter matrices.

7.18.1 Detailed Description

Module to select the resolved-unresolved components.

Copyright

2017 Jonathan Demaeyer See LICENSE.txt for license information.

7.18.2 Function/Subroutine Documentation

7.18.2.1 subroutine, public sf_def::load_sf ( )

Subroutine to load the unresolved variable defintion vector SF from SF.nml if it exists. If it does not, then write SF.nml with no unresolved variables specified (null vector).

Definition at line 37 of file sf_def.f90.

```
    INTEGER :: i,allocstat,n,ns
    CHARACTER(len=20) :: fm

    namelist /sflist/ sf

    fm(1:6)='(F3.1)'

    IF (ndim == 0) stop "*** Number of dimensions is 0! ***"
    ALLOCATE(sf(0:ndim), stat=allocstat)
    IF (allocstat /= 0) stop "*** Not enough memory ! ***"

    INQUIRE(file='./SF.nml',exist=exists)

    IF (exists) THEN
       OPEN(8, file="SF.nml", status='OLD', recl=80, delim='APOSTROPHE')
       READ(8,nml=sflist)
       CLOSE(8)
       n_unres=0
       DO i=1,ndim ! Computing the number of unresolved variables
          IF (sf(i)==1) n_unres=n_unres+1
       ENDDO
       IF (n_unres==0) stop "*** No unresolved variable specified! ***"
       n_res=ndim-n_unres
       ALLOCATE(ind(n_unres), rind(0:ndim), sl_ind(n_res), sl_rind(0:ndim),
      stat=allocstat)
       IF (allocstat /= 0) stop "*** Not enough memory ! ***"
       ALLOCATE(bar(0:ndim,0:ndim), bau(0:ndim,0:ndim), bor(0:
      ndim,0:ndim), bou(0:ndim,0:ndim), stat=allocstat)
       IF (allocstat /= 0) stop "*** Not enough memory ! ***"
       rind=0
       n=1
       ns=1
       DO i=1,ndim
          IF (sf(i)==1) THEN
             ind(n)=i
             rind(i)=n
             n=n+1
          ELSE
             sl_ind(ns)=i
             sl_rind(i)=ns
             ns=ns+1
          ENDIF
       ENDDO
       bar=0
       bau=0
       bor=0
       bou=0
       DO i=1,2*natm
          IF (sf(i)==1) THEN
             bau(i,i)=1
          ELSE
             bar(i,i)=1
          ENDIF
       ENDDO
       DO i=2*natm+1,ndim
          IF (sf(i)==1) THEN
             bou(i,i)=1
          ELSE
             bor(i,i)=1
          ENDIF
       ENDDO
    ELSE
       OPEN(8, file="SF.nml", status='NEW')
       WRITE(8,'(a)') "!------------------------------------------------------------------------------!"
       WRITE(8,'(a)') "! Namelist file :                                                              !"
        WRITE(8,'(a)') "! Unresolved variables specification (1 -> unresolved, 0 -> resolved)
             !"
        WRITE(8,'(a)') "!------------------------------------------------------------------------------!"
        WRITE(8,*) ""
        WRITE(8,'(a)') "&SFLIST"
        WRITE(8,*) " ! psi variables"
        DO i=1,natm
           WRITE(8,*) " SF("//trim(str(i))//") = 0"//"   ! typ= "&
                &//awavenum(i)%typ//", Nx= "//trim(rstr(awavenum(i)&
                &%Nx,fm))//", Ny= "//trim(rstr(awavenum(i)%Ny,fm))
        END DO
        WRITE(8,*) " ! theta variables"
        DO i=1,natm
           WRITE(8,*) " SF("//trim(str(i+natm))//") = 0"//"   ! typ= "&
                &//awavenum(i)%typ//", Nx= "//trim(rstr(awavenum(i)&
                &%Nx,fm))//", Ny= "//trim(rstr(awavenum(i)%Ny,fm))
        END DO

        WRITE(8,*) " ! A variables"
        DO i=1,noc
           WRITE(8,*) " SF("//trim(str(i+2*natm))//") = 0"//"   ! Nx&
                &= "//trim(rstr(owavenum(i)%Nx,fm))//", Ny= "&
```

```
                &//trim(rstr(owavenum(i)%Ny,fm))
         END DO
         WRITE(8,*) " ! T variables"
         DO i=1,noc
            WRITE(8,*) " SF("//trim(str(i+noc+2*natm))//") = 0"//"   &
                &! Nx= "//trim(rstr(owavenum(i)%Nx,fm))//", Ny= "&
                &//trim(rstr(owavenum(i)%Ny,fm))
         END DO

         WRITE(8,'(a)') "&END"
         WRITE(8,*) ""
         CLOSE(8)
         stop "*** SF.nml namelist written. Fill in the file and rerun !***"
      ENDIF
```

7.18.3 Variable Documentation

7.18.3.1 integer, dimension(:,:), allocatable, public sf_def::bar

Definition at line 28 of file sf_def.f90.

```
28    INTEGER, DIMENSION(:,:), ALLOCATABLE, PUBLIC :: bar,bau,bor,bou  !< Filter matrices
```

7.18.3.2 integer, dimension(:,:), allocatable, public sf_def::bau

Definition at line 28 of file sf_def.f90.

7.18.3.3 integer, dimension(:,:), allocatable, public sf_def::bor

Definition at line 28 of file sf_def.f90.

7.18.3.4 integer, dimension(:,:), allocatable, public sf_def::bou

Filter matrices.

Definition at line 28 of file sf_def.f90.

7.18.3.5 logical sf_def::exists [private]

Boolean to test for file existence.

Definition at line 21 of file sf_def.f90.

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

7.18.3.6 integer, dimension(:), allocatable, public sf_def::ind

Definition at line 24 of file sf_def.f90.

```
24    INTEGER, DIMENSION(:), ALLOCATABLE, PUBLIC :: ind,rind        !< Unresolved reduction indices
```

7.18.3.7 integer, public sf_def::n_res

Number of resolved variables.

Definition at line 27 of file sf_def.f90.

```
27   INTEGER, PUBLIC :: n_res !< Number of resolved variables
```

7.18.3.8 integer, public sf_def::n_unres

Number of unresolved variables.

Definition at line 26 of file sf_def.f90.

```
26   INTEGER, PUBLIC :: n_unres !< Number of unresolved variables
```

7.18.3.9 integer, dimension(:), allocatable, public sf_def::rind

Unresolved reduction indices.

Definition at line 24 of file sf_def.f90.

7.18.3.10 integer, dimension(:), allocatable, public sf_def::sf

Unresolved variable definition vector.

Definition at line 23 of file sf_def.f90.

```
23   INTEGER, DIMENSION(:), ALLOCATABLE, PUBLIC :: sf             !< Unresolved variable definition vector
```

7.18.3.11 integer, dimension(:), allocatable, public sf_def::sl_ind

Definition at line 25 of file sf_def.f90.

```
25   INTEGER, DIMENSION(:), ALLOCATABLE, PUBLIC :: sl_ind,sl_rind  !< Resolved reduction indices
```

7.18.3.12 integer, dimension(:), allocatable, public sf_def::sl_rind

Resolved reduction indices.

Definition at line 25 of file sf_def.f90.

7.19 sigma Module Reference

The MTV noise sigma matrices used to integrate the MTV model.

Functions/Subroutines

- subroutine, public init_sigma (mult, Q1fill)
  Subroutine to initialize the sigma matices.
- subroutine, public compute_mult_sigma (y)
  Routine to actualize the matrix σ_1 based on the state y of the MTV system.

Variables

- real(kind=8), dimension(:,:), allocatable, public sig1
  $\sigma_1(X)$ state-dependent noise matrix
- real(kind=8), dimension(:,:), allocatable, public sig2
  σ_2 state-independent noise matrix
- real(kind=8), dimension(:,:), allocatable, public sig1r
  Reduced $\sigma_1(X)$ state-dependent noise matrix.
- real(kind=8), dimension(:,:), allocatable dumb_mat1
  Dummy matrix.
- real(kind=8), dimension(:,:), allocatable dumb_mat2
  Dummy matrix.
- real(kind=8), dimension(:,:), allocatable dumb_mat3
  Dummy matrix.
- real(kind=8), dimension(:,:), allocatable dumb_mat4
  Dummy matrix.
- integer, dimension(:), allocatable ind1
- integer, dimension(:), allocatable rind1
- integer, dimension(:), allocatable ind2
- integer, dimension(:), allocatable rind2
  Reduction indices.
- integer n1
- integer n2

7.19.1 Detailed Description

The MTV noise sigma matrices used to integrate the MTV model.

Copyright

Remarks

See : Franzke, C., Majda, A. J., & Vanden-Eijnden, E. (2005). Low-order stochastic mode reduction for a realistic barotropic model climate. Journal of the atmospheric sciences, 62(6), 1722-1745.

7.19.2 Function/Subroutine Documentation

7.19.2.1 subroutine, public sigma::compute_mult_sigma (real(kind=8), dimension(0:ndim), intent(in) *y*)

Routine to actualize the matrix σ_1 based on the state y of the MTV system.

Parameters

y	State of the MTV system

Definition at line 93 of file MTV_sigma_tensor.f90.

```
93      REAL(KIND=8), DIMENSION(0:ndim), INTENT(IN) :: y
94      INTEGER :: info,info2
95      CALL sparse_mul3_mat(utot,y,dumb_mat1)
96      CALL sparse_mul4_mat(vtot,y,y,dumb_mat2)
97      dumb_mat3=dumb_mat1+dumb_mat2+q1
98      CALL reduce(dumb_mat3,dumb_mat1,n1,ind1,rind1)
99      IF (n1 /= 0) THEN
100         CALL sqrtm_svd(dumb_mat1(1:n1,1:n1),dumb_mat2(1:n1,1:n1),info,info2,min(max(n1/2,2),64))
101         ! dumb_mat2=0.D0
102         ! CALL chol(0.5*(dumb_mat1(1:n1,1:n1)+transpose(dumb_mat1(1:n1,1:n1))),dumb_mat2(1:n1,1:n1),info)
103         IF ((.not.any(isnan(dumb_mat2))).and.(info.eq.0).and.(.not.any(dumb_mat2>huge(0.d0)))) THEN
104            CALL ireduce(sig1,dumb_mat2,n1,ind1,rind1)
105         ELSE
106            sig1=sig1r
107         ENDIF
108      ELSE
109         sig1=sig1r
110      ENDIF
```

7.19.2.2 subroutine, public sigma::init_sigma (logical, intent(out) *mult,* logical, intent(out) *Q1fill*)

Subroutine to initialize the sigma matices.

Definition at line 48 of file MTV_sigma_tensor.f90.

```
48      LOGICAL, INTENT(OUT) :: mult,q1fill
49      INTEGER :: allocstat,info1,info2
50
51      CALL init_sqrt
52
53      ALLOCATE(sig1(ndim,ndim), sig2(ndim,ndim), sig1r(ndim,
       ndim),stat=allocstat)
54      IF (allocstat /= 0) stop "*** Not enough memory ! ***"
55
56      ALLOCATE(ind1(ndim), rind1(ndim), ind2(ndim), rind2(ndim),
       stat=allocstat)
57      IF (allocstat /= 0) stop "*** Not enough memory ! ***"
58
59      ALLOCATE(dumb_mat1(ndim,ndim), dumb_mat2(ndim,ndim), stat=allocstat)
60      IF (allocstat /= 0) stop "*** Not enough memory ! ***"
61
62      ALLOCATE(dumb_mat3(ndim,ndim), dumb_mat4(ndim,ndim), stat=allocstat)
63      IF (allocstat /= 0) stop "*** Not enough memory ! ***"
64
65      print*, "Initializing the sigma matrices"
66
67      CALL reduce(q2,dumb_mat1,n2,ind2,rind2)
68      IF (n2 /= 0) THEN
69         CALL sqrtm_svd(dumb_mat1(1:n2,1:n2),dumb_mat2(1:n2,1:n2),info1,info2,min(max(n2/2,2),64))
70         CALL ireduce(sig2,dumb_mat2,n2,ind2,rind2)
71      ELSE
72         sig2=0.d0
73      ENDIF
74
75      mult=(.not.((tensor_empty(utot)).and.(tensor4_empty(vtot))))
76      q1fill=.true.
77      CALL reduce(q1,dumb_mat1,n1,ind1,rind1)
78      IF (n1 /= 0) THEN
79
80         CALL sqrtm_svd(dumb_mat1(1:n1,1:n1),dumb_mat2(1:n1,1:n1),info1,info2,min(max(n1/2,2),64))
81         CALL ireduce(sig1,dumb_mat2,n1,ind1,rind1)
82      ELSE
83         q1fill=.false.
84         sig1=0.d0
85      ENDIF
86      sig1r=sig1
87
```

7.19.3 Variable Documentation

7.19.3.1 real(kind=8), dimension(:,:), allocatable sigma::dumb_mat1 [private]

Dummy matrix.

Definition at line 35 of file MTV_sigma_tensor.f90.

```
REAL(KIND=8), DIMENSION(:,:), ALLOCATABLE :: dumb_mat1 !< Dummy matrix
```

7.19.3.2 real(kind=8), dimension(:,:), allocatable sigma::dumb_mat2 [private]

Dummy matrix.

Definition at line 36 of file MTV_sigma_tensor.f90.

```
REAL(KIND=8), DIMENSION(:,:), ALLOCATABLE :: dumb_mat2 !< Dummy matrix
```

7.19.3.3 real(kind=8), dimension(:,:), allocatable sigma::dumb_mat3 [private]

Dummy matrix.

Definition at line 37 of file MTV_sigma_tensor.f90.

```
REAL(KIND=8), DIMENSION(:,:), ALLOCATABLE :: dumb_mat3 !< Dummy matrix
```

7.19.3.4 real(kind=8), dimension(:,:), allocatable sigma::dumb_mat4 [private]

Dummy matrix.

Definition at line 38 of file MTV_sigma_tensor.f90.

```
REAL(KIND=8), DIMENSION(:,:), ALLOCATABLE :: dumb_mat4 !< Dummy matrix
```

7.19.3.5 integer, dimension(:), allocatable sigma::ind1 [private]

Definition at line 39 of file MTV_sigma_tensor.f90.

```
INTEGER, DIMENSION(:), ALLOCATABLE :: ind1,rind1,ind2,rind2 !< Reduction indices
```

7.19.3.6 integer, dimension(:), allocatable sigma::ind2 [private]

Definition at line 39 of file MTV_sigma_tensor.f90.

7.19.3.7 integer sigma::n1 [private]

Definition at line 41 of file MTV_sigma_tensor.f90.

```
41    INTEGER :: n1,n2
```

7.19.3.8 integer sigma::n2 [private]

Definition at line 41 of file MTV_sigma_tensor.f90.

7.19.3.9 integer, dimension(:), allocatable sigma::rind1 [private]

Definition at line 39 of file MTV_sigma_tensor.f90.

7.19.3.10 integer, dimension(:), allocatable sigma::rind2 [private]

Reduction indices.

Definition at line 39 of file MTV_sigma_tensor.f90.

7.19.3.11 real(kind=8), dimension(:,:), allocatable, public sigma::sig1

$\sigma_1(X)$ state-dependent noise matrix

Definition at line 31 of file MTV_sigma_tensor.f90.

```
31    REAL(KIND=8), DIMENSION(:,:), ALLOCATABLE, PUBLIC :: sig1 !< \f$\sigma_1(X)\f$ state-dependent noise matrix
```

7.19.3.12 real(kind=8), dimension(:,:), allocatable, public sigma::sig1r

Reduced $\sigma_1(X)$ state-dependent noise matrix.

Definition at line 33 of file MTV_sigma_tensor.f90.

```
33    REAL(KIND=8), DIMENSION(:,:), ALLOCATABLE, PUBLIC :: sig1r !< Reduced \f$\sigma_1(X)\f$ state-dependent noise matrix
```

7.19.3.13 real(kind=8), dimension(:,:), allocatable, public sigma::sig2

σ_2 state-independent noise matrix

Definition at line 32 of file MTV_sigma_tensor.f90.

```
32    REAL(KIND=8), DIMENSION(:,:), ALLOCATABLE, PUBLIC :: sig2 !< \f$\sigma_2\f$ state-independent noise matrix
```

7.20 sqrt_mod Module Reference

Utility module with various routine to compute matrix square root.

Functions/Subroutines

- subroutine, public init_sqrt
- subroutine, public sqrtm (A, sqA, info, info_triu, bs)

  Routine to compute a real square-root of a matrix.
- logical function selectev (a, b)
- subroutine sqrtm_triu (A, sqA, info, bs)
- subroutine csqrtm_triu (A, sqA, info, bs)
- subroutine rsf2csf (T, Z, Tz, Zz)
- subroutine, public chol (A, sqA, info)

  Routine to perform a Cholesky decomposition.
- subroutine, public sqrtm_svd (A, sqA, info, info_triu, bs)

  Routine to compute a real square-root of a matrix via a SVD decomposition.

Variables

- real(kind=8), dimension(:), allocatable work
- integer lwork
- real(kind=8), parameter real_eps = 2.2204460492503131e-16

7.20.1 Detailed Description

Utility module with various routine to compute matrix square root.

Copyright

2017 Jonathan Demaeyer. See LICENSE.txt for license information.

Remarks

Mainly based on the numerical recipes and from: Edvin Deadman, Nicholas J. Higham, Rui Ralha (2013) "↩
Blocked Schur Algorithms for Computing the Matrix Square Root", Lecture Notes in Computer Science, 7782. pp. 171-182.

7.20.2 Function/Subroutine Documentation

7.20.2.1 subroutine, public sqrt_mod::chol (real(kind=8), dimension(:,:), intent(in) *A*, real(kind=8), dimension(:,:), intent(out) *sqA*, integer, intent(out) *info*)

Routine to perform a Cholesky decomposition.

Parameters

A	Matrix whose decomposition is evaluated.
sqA	Cholesky decomposition of A.
info	Information code returned by the Lapack routines.

Definition at line 386 of file sqrt_mod.f90.

```
REAL(KIND=8), DIMENSION(:,:), INTENT(IN) :: a
REAL(KIND=8), DIMENSION(:,:), INTENT(OUT) :: sqa
INTEGER, INTENT(OUT) :: info

sqa=a
CALL dpotrf('L',SIZE(sqa,1),sqa,SIZE(sqa,1),info)
```

7.20.2.2 subroutine sqrt_mod::csqrtm_triu (complex(kind=16), dimension(:,:), intent(in) *A*, complex(kind=16), dimension(:,:), intent(out) *sqA*, integer, intent(out) *info*, integer, intent(in), optional *bs*) [private]

Definition at line 235 of file sqrt_mod.f90.

```
COMPLEX(KIND=16), DIMENSION(:,:), INTENT(IN) :: a
INTEGER, INTENT(IN), OPTIONAL :: bs
COMPLEX(KIND=16), DIMENSION(:,:), INTENT(OUT) :: sqa
INTEGER, INTENT(OUT) :: info
COMPLEX(KIND=16), DIMENSION(SIZE(A,1)) :: a_diag
COMPLEX(KIND=16), DIMENSION(SIZE(A,1),SIZE(A,1)) :: r,sm,rii,rjj
INTEGER, DIMENSION(2*SIZE(A,1),2) :: start_stop_pairs
COMPLEX(KIND=16) :: s,denom,scale
INTEGER :: i,j,k,start,n,sstop,m
INTEGER :: istart,istop,jstart,jstop
INTEGER :: nblocks,blocksize
INTEGER :: bsmall,blarge,nlarge,nsmall

blocksize=64
IF (PRESENT(bs)) blocksize=bs
n=SIZE(a,1)
! print*,  blocksize

CALL cdiag(a,a_diag)
r=0.d0
DO i=1,n
   r(i,i)=sqrt(a_diag(i))
ENDDO


nblocks=max(floordiv(n,blocksize),1)
bsmall=floordiv(n,nblocks)
nlarge=mod(n,nblocks)
blarge=bsmall+1
nsmall=nblocks-nlarge
IF (nsmall*bsmall + nlarge*blarge /= n) stop 'Sqrtm: Internal inconsistency'

! print*, nblocks,bsmall,nsmall,blarge,nlarge

start=1
DO i=1,nsmall
   start_stop_pairs(i,1)=start
   start_stop_pairs(i,2)=start+bsmall-1
   start=start+bsmall
ENDDO
DO i=nsmall+1,nsmall+nlarge
   start_stop_pairs(i,1)=start
   start_stop_pairs(i,2)=start+blarge-1
   start=start+blarge
ENDDO

! DO i=1,SIZE(start_stop_pairs,1)
!    print*, i
!    print*, start_stop_pairs(i,1),start_stop_pairs(i,2)
! END DO

DO k=1,nsmall+nlarge
```

```
        start=start_stop_pairs(k,1)
        sstop=start_stop_pairs(k,2)
        DO j=start,sstop
           DO i=j-1,start,-1
              s=0.d0
              IF (j-i>1) s= dot_product(r(i,i+1:j-1),r(i+1:j-1,j))
              denom= r(i,i)+r(j,j)
              IF (denom==0.d0) stop 'Sqrtm: Failed to find the matrix square root'
              r(i,j)=(a(i,j)-s)/denom
           END DO
        END DO
     END DO

     ! print*, 'R'
     ! CALL printmat(R)

     DO j=1,nblocks
        jstart=start_stop_pairs(j,1)
        jstop=start_stop_pairs(j,2)
        DO i=j-1,1,-1
           istart=start_stop_pairs(i,1)
           istop=start_stop_pairs(i,2)
           sm=0.d0
           sm(istart:istop,jstart:jstop)=a(istart:istop,jstart:jstop)
           IF (j-i>1) sm(istart:istop,jstart:jstop) = sm(istart:istop&
                &,jstart:jstop) - matmul(r(istart:istop,istop:jstart)&
                &,r(istop:jstart,jstart:jstop))
           rii=0.d0
           rii = r(istart:istop, istart:istop)
           rjj=0.d0
           rjj = r(jstart:jstop, jstart:jstop)
           m=istop-istart+1
           n=jstop-jstart+1
           k=1
           ! print*, m,n
           ! print*, istart,istop
           ! print*, jstart,jstop

           ! print*, 'Rii',Rii(istart:istop, istart:istop)
           ! print*, 'Rjj',Rjj(jstart:jstop,jstart:jstop)
           ! print*, 'Sm',Sm(istart:istop,jstart:jstop)

           CALL ztrsyl('N','N',k,m,n,rii(istart:istop, istart:istop),m&
                &,rjj(jstart:jstop,jstart:jstop),n,sm(istart:istop&
                &,jstart:jstop),m,scale,info)
           r(istart:istop,jstart:jstop)=sm(istart:istop,jstart:jstop)*scale
        ENDDO
     ENDDO
     sqa=r
```

7.20.2.3 subroutine, public sqrt_mod::init_sqrt ( )

Definition at line 39 of file sqrt_mod.f90.

```
   INTEGER :: allocstat
   lwork=10
   lwork=ndim*lwork

   ! print*, lwork

   IF (ALLOCATED(work)) THEN
      DEALLOCATE(work, stat=allocstat)
      IF (allocstat /= 0) stop "*** Deallocation problem ! ***"
   ENDIF
   ALLOCATE(work(lwork), stat=allocstat)
   IF (allocstat /= 0) stop "*** Not enough memory ! ***"

   ! ALLOCATE(zwork(lwork), STAT=AllocStat)
   ! IF (AllocStat /= 0) STOP "*** Not enough memory ! ***"
```

7.20.2.4 subroutine sqrt_mod::rsf2csf (real(kind=8), dimension(:,:), intent(in) *T*, real(kind=8), dimension(:,:), intent(in) *Z*, complex(kind=16), dimension(:,:), intent(out) *Tz*, complex(kind=16), dimension(:,:), intent(out) *Zz*) [private]

Definition at line 339 of file sqrt_mod.f90.

```
    REAL(KIND=8), DIMENSION(:,:), INTENT(IN) :: t,z
    COMPLEX(KIND=16), DIMENSION(:,:), INTENT(OUT) :: tz,zz
    INTEGER, PARAMETER :: nb=2
    COMPLEX(KIND=16), DIMENSION(nb) :: w
    !COMPLEX(KIND=16), DIMENSION(nb,nb) :: vl,vr
    COMPLEX(KIND=16) :: r,c,s,mu
    COMPLEX(KIND=16), DIMENSION(nb,nb) :: g,gc
    INTEGER :: n,m!,info
    !REAL(KIND=8), DIMENSION(2*nb) :: rwork
    !REAL(KIND=8), DIMENSION(2*nb) :: ztwork

    ! print*, lwork
    tz=cmplx(t,kind=16)
    zz=cmplx(z,kind=16)
    n=SIZE(t,1)
    DO m=n,2,-1
       IF (abs(tz(m,m-1)) > real_eps*(abs(tz(m-1,m-1)) + abs(tz(m,m)))) THEN
          g=tz(m-1:m,m-1:m)
          ! CALL printmat(dble(G))
          ! CALL zgeev('N','N',nb,G,nb,w,vl,nb,vr,nb,ztwork,2*nb,rwork,info)
          ! CALL cprintmat(G)
          ! print*, m,w,info
          s=g(1,1)+g(2,2)
          c=g(1,1)*g(2,2)-g(1,2)*g(2,1)
          w(1)=s/2+sqrt(s**2/4-c)
          mu=w(1)-tz(m,m)
          r=sqrt(mu*conjg(mu)+tz(m,m-1)*conjg(tz(m,m-1)))
          c=mu/r
          s=tz(m,m-1)/r
          g(1,1)=conjg(c)
          g(1,2)=s
          g(2,1)=-s
          g(2,2)=c
          gc=conjg(transpose(g))
          tz(m-1:m,m-1:n)=matmul(g,tz(m-1:m,m-1:n))
          tz(1:m,m-1:m)=matmul(tz(1:m,m-1:m),gc)
          zz(:,m-1:m)=matmul(zz(:,m-1:m),gc)
       END IF
       tz(m,m-1)=cmplx(0.d0,kind=16)
    END DO
```

7.20.2.5 logical function sqrt_mod::selectev (real(kind=8) *a*, real(kind=8) *b*) [private]

Definition at line 122 of file sqrt_mod.f90.

```
122     REAL(KIND=8) :: a,b
123     LOGICAL selectev
124     selectev=.false.
125     ! IF (a>b) selectev=.true.
126     RETURN
```

7.20.2.6 subroutine, public sqrt_mod::sqrtm (real(kind=8), dimension(:,:), intent(in) *A*, real(kind=8), dimension(:,:), intent(out) *sqA*, integer, intent(out) *info*, integer, intent(out) *info_triu*, integer, intent(in), optional *bs*)

Routine to compute a real square-root of a matrix.

Parameters

A	Matrix whose square root to evaluate.
sqA	Square root of `A`.
info	Information code returned by the Lapack routines.
info_triu	Information code returned by the triangular matrix Lapack routines.
bs	Optional blocksize specification variable.

Definition at line 63 of file sqrt_mod.f90.

```
    REAL(KIND=8), DIMENSION(:,:), INTENT(IN) :: a
    REAL(KIND=8), DIMENSION(:,:), INTENT(OUT) :: sqa
    INTEGER, INTENT(IN), OPTIONAL :: bs
    INTEGER, INTENT(OUT) :: info,info_triu
    REAL(KIND=8), DIMENSION(SIZE(A,1),SIZE(A,1)) :: t,z,r
    COMPLEX(KIND=16), DIMENSION(SIZE(A,1),SIZE(A,1)) :: tz,zz,rz
    REAL(KIND=8), DIMENSION(SIZE(A,1)) :: wr,wi
    LOGICAL, DIMENSION(SIZE(A,1)) :: bwork
    LOGICAL :: selectev
    INTEGER :: n
    INTEGER :: sdim=0
    n=SIZE(a,1)
    t=a
    ! print*, n,size(work,1)
    CALL dgees('v','n',selectev,n,t,n,sdim,wr,wi,z,n,work,lwork,bwork,info)
    ! print*, 'Z'
    ! CALL printmat(Z)
    ! print*, 'T'
    ! CALL printmat(T)
    ! CALL DGEES('V','N',SIZE(T,1),T,SIZE(T,1),0,wr,wi,Z,SIZE(Z,1),work,lwork,info)
    ! print*, info
    CALL triu(t,r)
    IF (any(t /= r)) THEN
       ! print*, 'T'
       ! CALL printmat(T)
       ! print*, 'Z'
       ! CALL printmat(Z)
       CALL rsf2csf(t,z,tz,zz)
       ! print*, 'Tz'
       ! CALL printmat(dble(Tz))
       ! print*, 'iTz'
       ! CALL printmat(dble(aimag(Tz)))
       ! print*, 'Zz'
       ! CALL printmat(dble(Zz))
       ! print*, 'iZz'
       ! CALL printmat(dble(aimag(Zz)))
       IF (PRESENT(bs)) THEN
           CALL csqrtm_triu(tz,rz,info_triu,bs)
        ELSE
           CALL csqrtm_triu(tz,rz,info_triu)
        END IF
        rz=matmul(zz,matmul(rz,conjg(transpose(zz))))
        ! print*, 'sqAz'
        ! CALL printmat(dble(Rz))
        ! print*, 'isqAz'
        ! CALL printmat(dble(aimag(Rz)))
        sqa=dble(rz)
     ELSE
        IF (PRESENT(bs)) THEN
           CALL sqrtm_triu(t,r,info_triu,bs)
        ELSE
           CALL sqrtm_triu(t,r,info_triu)
        END IF
        sqa=matmul(z,matmul(r,transpose(z)))
     ENDIF

```

7.20.2.7 subroutine, public sqrt_mod::sqrtm_svd (real(kind=8), dimension(:,:), intent(in) *A*, real(kind=8), dimension(:,:), intent(out) *sqA*, integer, intent(out) *info*, integer, intent(out) *info_triu*, integer, intent(in), optional *bs*)

Routine to compute a real square-root of a matrix via a SVD decomposition.

Parameters

A	Matrix whose square root to evaluate.
sqA	Square root of `A`.
info	Information code returned by the Lapack routines.
info_triu	Not used (present for compatibility).
bs	Not used (present for compatibility).

Definition at line 401 of file sqrt_mod.f90.

```
REAL(KIND=8), DIMENSION(:,:), INTENT(IN) :: a
REAL(KIND=8), DIMENSION(:,:), INTENT(OUT) :: sqa
INTEGER, INTENT(IN), OPTIONAL :: bs
INTEGER, INTENT(OUT) :: info,info_triu
REAL(KIND=8), DIMENSION(SIZE(A,1)) :: s
REAL(KIND=8), DIMENSION(SIZE(A,1),SIZE(A,1)) :: sq,u,vt
INTEGER :: i,n

sqa=a
n=SIZE(sqa,1)
CALL dgesvd('A','A',n,n,sqa,n,s,u,n,vt,n,work,lwork,info)
sq=0.d0
DO i=1,n
   sq(i,i)=sqrt(s(i))
ENDDO
sqa=matmul(u,matmul(sq,vt))
```

7.20.2.8 subroutine sqrt_mod::sqrtm_triu (real(kind=8), dimension(:,:), intent(in) *A*, real(kind=8), dimension(:,:), intent(out) *sqA*, integer, intent(out) *info*, integer, intent(in), optional *bs*) [private]

Definition at line 131 of file sqrt_mod.f90.

```
REAL(KIND=8), DIMENSION(:,:), INTENT(IN) :: a
INTEGER, INTENT(IN), OPTIONAL :: bs
REAL(KIND=8), DIMENSION(:,:), INTENT(OUT) :: sqa
INTEGER, INTENT(OUT) :: info
REAL(KIND=8), DIMENSION(SIZE(A,1)) :: a_diag
REAL(KIND=8), DIMENSION(SIZE(A,1),SIZE(A,1)) :: r,sm,rii,rjj
INTEGER, DIMENSION(2*SIZE(A,1),2) :: start_stop_pairs
REAL(KIND=8) :: s,denom,scale
INTEGER :: i,j,k,start,n,sstop,m
INTEGER :: istart,istop,jstart,jstop
INTEGER :: nblocks,blocksize
INTEGER :: bsmall,blarge,nlarge,nsmall

blocksize=64
IF (PRESENT(bs)) blocksize=bs
n=SIZE(a,1)
! print*,  blocksize

CALL diag(a,a_diag)
r=0.d0
DO i=1,n
   r(i,i)=sqrt(a_diag(i))
ENDDO


nblocks=max(floordiv(n,blocksize),1)
bsmall=floordiv(n,nblocks)
nlarge=mod(n,nblocks)
blarge=bsmall+1
nsmall=nblocks-nlarge
IF (nsmall*bsmall + nlarge*blarge /= n) stop 'Sqrtm: Internal inconsistency'

! print*, nblocks,bsmall,nsmall,blarge,nlarge

start=1
DO i=1,nsmall
   start_stop_pairs(i,1)=start
   start_stop_pairs(i,2)=start+bsmall-1
   start=start+bsmall
ENDDO
DO i=nsmall+1,nsmall+nlarge
   start_stop_pairs(i,1)=start
   start_stop_pairs(i,2)=start+blarge-1
   start=start+blarge
ENDDO

! DO i=1,SIZE(start_stop_pairs,1)
!    print*, i
!    print*, start_stop_pairs(i,1),start_stop_pairs(i,2)
! END DO

DO k=1,nsmall+nlarge
   start=start_stop_pairs(k,1)
   sstop=start_stop_pairs(k,2)
   DO j=start,sstop
      DO i=j-1,start,-1
         s=0.d0
```

```
            IF (j-i>1) s= dot_product(r(i,i+1:j-1),r(i+1:j-1,j))
            denom= r(i,i)+r(j,j)
            IF (denom==0.d0) stop 'Sqrtm: Failed to find the matrix square root'
            r(i,j)=(a(i,j)-s)/denom
         END DO
      END DO
   END DO

   ! print*, 'R'
   ! CALL printmat(R)

   DO j=1,nblocks
      jstart=start_stop_pairs(j,1)
      jstop=start_stop_pairs(j,2)
      DO i=j-1,1,-1
         istart=start_stop_pairs(i,1)
         istop=start_stop_pairs(i,2)
         sm=0.d0
         sm(istart:istop,jstart:jstop)=a(istart:istop,jstart:jstop)
         IF (j-i>1) sm(istart:istop,jstart:jstop) = sm(istart:istop&
              &,jstart:jstop) - matmul(r(istart:istop,istop:jstart)&
              &,r(istop:jstart,jstart:jstop))
         rii=0.d0
         rii = r(istart:istop, istart:istop)
         rjj=0.d0
         rjj = r(jstart:jstop, jstart:jstop)
         m=istop-istart+1
         n=jstop-jstart+1
         k=1
         ! print*, m,n
         ! print*, istart,istop
         ! print*, jstart,jstop

         ! print*, 'Rii',Rii(istart:istop, istart:istop)
         ! print*, 'Rjj',Rjj(jstart:jstop,jstart:jstop)
         ! print*, 'Sm',Sm(istart:istop,jstart:jstop)

         CALL dtrsyl('N','N',k,m,n,rii(istart:istop, istart:istop),m&
              &,rjj(jstart:jstop,jstart:jstop),n,sm(istart:istop&
              &,jstart:jstop),m,scale,info)
         r(istart:istop,jstart:jstop)=sm(istart:istop,jstart:jstop)*scale
      ENDDO
   ENDDO
   sqa=r
```

7.20.3 Variable Documentation

7.20.3.1 integer sqrt_mod::lwork [private]

Definition at line 30 of file sqrt_mod.f90.

```
30 INTEGER :: lwork
```

7.20.3.2 real(kind=8), parameter sqrt_mod::real_eps = 2.2204460492503131e-16 [private]

Definition at line 32 of file sqrt_mod.f90.

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

7.20.3.3 real(kind=8), dimension(:), allocatable sqrt_mod::work [private]

Definition at line 27 of file sqrt_mod.f90.

```
27 REAL(KIND=8), DIMENSION(:), ALLOCATABLE :: work
```

7.21 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

7.21.1 Detailed Description

Statistics accumulators.

Copyright

2015 Lesley De Cruz & Jonathan Demaeyer. See LICENSE.txt for license information.

7.21.2 Function/Subroutine Documentation

7.21.2.1 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)
```

7.21.2.2 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
```

7.21.2.3 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
```

7.21.2.4 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
```

7.21.2.5 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
```

7.21.2.6 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)
```

7.21.3 Variable Documentation

7.21.3.1 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
```

7.21.3.2 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
```

7.21.3.3 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
```

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

Definition at line 26 of file stat.f90.

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

7.21.3.5 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
```

7.22 stoch_mod Module Reference

Utility module containing the stochastic related routines.

Functions/Subroutines

- real(kind=8) function, public gasdev ()
- subroutine, public stoch_vec (dW)

    Routine to fill a vector with standard Gaussian noise process values.
- subroutine, public stoch_atm_vec (dW)

    routine to fill the atmospheric component of a vector with standard gaussian noise process values
- subroutine, public stoch_atm_res_vec (dW)

    routine to fill the resolved atmospheric component of a vector with standard gaussian noise process values
- subroutine, public stoch_atm_unres_vec (dW)

    routine to fill the unresolved atmospheric component of a vector with standard gaussian noise process values
- subroutine, public stoch_oc_vec (dW)

    routine to fill the oceanic component of a vector with standard gaussian noise process values
- subroutine, public stoch_oc_res_vec (dW)

    routine to fill the resolved oceanic component of a vector with standard gaussian noise process values
- subroutine, public stoch_oc_unres_vec (dW)

    routine to fill the unresolved oceanic component of a vector with standard gaussian noise process values

Variables

- integer iset =0
- real(kind=8) gset

7.22.1 Detailed Description

Utility module containing the stochastic related routines.

Copyright

Remarks

7.22.2 Function/Subroutine Documentation

7.22.2.1 real(kind=8) function, public stoch_mod::gasdev ( )

Definition at line 32 of file stoch_mod.f90.

```
REAL(KIND=8) :: gasdev
REAL(KIND=8) :: fac,rsq,v1,v2,r
IF (iset.eq.0) THEN
   DO
      CALL random_number(r)
      v1=2.d0*r-1.
      CALL random_number(r)
      v2=2.d0*r-1.
      rsq=v1**2+v2**2
      IF (rsq.lt.1.d0.and.rsq.ne.0.d0) EXIT
   ENDDO
   fac=sqrt(-2.*log(rsq)/rsq)
   gset=v1*fac
   gasdev=v2*fac
   iset=1
ELSE
   gasdev=gset
   iset=0
ENDIF
RETURN
```

7.22.2.2 subroutine, public stoch_mod::stoch_atm_res_vec (real(kind=8), dimension(0:ndim), intent(inout) *dW*)

routine to fill the resolved atmospheric component of a vector with standard gaussian noise process values

Parameters

dW	vector to fill

Definition at line 77 of file stoch_mod.f90.

```
77      real(kind=8), dimension(0:ndim), intent(inout) :: dw
78      integer :: i
79      dw=0.d0
80      do i=1,2*natm
81         IF (sf(i)==0) dw(i)=gasdev()
82      enddo
```

7.22.2.3 subroutine, public stoch_mod::stoch_atm_unres_vec (real(kind=8), dimension(0:ndim), intent(inout) *dW*)

routine to fill the unresolved atmospheric component of a vector with standard gaussian noise process values

Parameters

dW	vector to fill

Definition at line 88 of file stoch_mod.f90.

```
88      real(kind=8), dimension(0:ndim), intent(inout) :: dw
89      integer :: i
90      dw=0.d0
91      do i=1,2*natm
92         IF (sf(i)==1) dw(i)=gasdev()
93      enddo
```

7.22.2.4 subroutine, public stoch_mod::stoch_atm_vec (real(kind=8), dimension(0:ndim), intent(inout) *dW*)

routine to fill the atmospheric component of a vector with standard gaussian noise process values

Parameters

dW	vector to fill

Definition at line 67 of file stoch_mod.f90.

```
67      real(kind=8), dimension(0:ndim), intent(inout) :: dw
68      integer :: i
69      do i=1,2*natm
70         dw(i)=gasdev()
71      enddo
```

7.22.2.5 subroutine, public stoch_mod::stoch_oc_res_vec (real(kind=8), dimension(0:ndim), intent(inout) *dW*)

routine to fill the resolved oceanic component of a vector with standard gaussian noise process values

Parameters

dW	vector to fill

Definition at line 109 of file stoch_mod.f90.

```
real(kind=8), dimension(0:ndim), intent(inout) :: dw
integer :: i
dw=0.d0
do i=2*natm+1,ndim
   IF (sf(i)==0) dw(i)=gasdev()
enddo
```

7.22.2.6 subroutine, public stoch_mod::stoch_oc_unres_vec (real(kind=8), dimension(0:ndim), intent(inout) *dW*)

routine to fill the unresolved oceanic component of a vector with standard gaussian noise process values

Parameters

dW	vector to fill

Definition at line 120 of file stoch_mod.f90.

```
real(kind=8), dimension(0:ndim), intent(inout) :: dw
integer :: i
dw=0.d0
do i=2*natm+1,ndim
   IF (sf(i)==1) dw(i)=gasdev()
enddo
```

7.22.2.7 subroutine, public stoch_mod::stoch_oc_vec (real(kind=8), dimension(0:ndim), intent(inout) *dW*)

routine to fill the oceanic component of a vector with standard gaussian noise process values

Parameters

dW	vector to fill

Definition at line 99 of file stoch_mod.f90.

```
real(kind=8), dimension(0:ndim), intent(inout) :: dw
integer :: i
do i=2*natm+1,ndim
   dw(i)=gasdev()
enddo
```

7.22.2.8 subroutine, public stoch_mod::stoch_vec (real(kind=8), dimension(0:ndim), intent(inout) *dW*)

Routine to fill a vector with standard Gaussian noise process values.

Parameters

dW	Vector to fill

Definition at line 57 of file stoch_mod.f90.

```
REAL(KIND=8), DIMENSION(0:ndim), INTENT(INOUT) :: dw
INTEGER :: i
DO i=1,ndim
   dw(i)=gasdev()
ENDDO
```

7.22.3 Variable Documentation

7.22.3.1 real(kind=8) stoch_mod::gset [private]

Definition at line 24 of file stoch_mod.f90.

```
REAL(KIND=8) :: gset
```

7.22.3.2 integer stoch_mod::iset =0 [private]

Definition at line 23 of file stoch_mod.f90.

```
INTEGER :: iset=0
```

7.23 stoch_params Module Reference

The stochastic models parameters module.

Functions/Subroutines

- subroutine init_stoch_params

    Stochastic parameters initialization routine.

Variables

- real(kind=8) mnuti

  Multiplicative noise update time interval.
- real(kind=8) t_trans_stoch

  Transient time period of the stochastic model evolution.
- real(kind=8) q_ar

  Atmospheric resolved component noise amplitude.
- real(kind=8) q_au

  Atmospheric unresolved component noise amplitude.
- real(kind=8) q_or

  Oceanic resolved component noise amplitude.
- real(kind=8) q_ou

  Oceanic unresolved component noise amplitude.
- real(kind=8) dtn

  Square root of the timestep.
- real(kind=8) eps_pert

  Perturbation parameter for the coupling.
- real(kind=8) tdelta

  Time separation parameter.
- real(kind=8) muti

  Memory update time interval.
- real(kind=8) meml

  Time over which the memory kernel is integrated.
- real(kind=8) t_trans_mem

  Transient time period to initialize the memory term.
- character(len=4) x_int_mode

  Integration mode for the resolved component.
- real(kind=8) dts

  Intrisic resolved dynamics time step.
- integer mems

  Number of steps in the memory kernel integral.
- real(kind=8) dtsn

  Square root of the intrisic resolved dynamics time step.
- real(kind=8) maxint

  Upper integration limit of the correlations.
- character(len=4) load_mode

  Loading mode for the correlations.
- character(len=4) int_corr_mode

  Correlation integration mode.
- character(len=4) mode

  Stochastic mode parameter.

7.23.1 Detailed Description

The stochastic models parameters module.

Copyright

Remarks

7.23.2 Function/Subroutine Documentation

7.23.2.1 subroutine stoch_params::init_stoch_params ( )

Stochastic parameters initialization routine.

Definition at line 58 of file stoch_params.f90.

```
58
59      namelist /mtvparams/ mnuti
60      namelist /stparams/ q_ar,q_au,q_or,q_ou,eps_pert,tdelta,t_trans_stoch
61      namelist /wlparams/ muti,meml,x_int_mode,dts,t_trans_mem
62      namelist /corr_init_mode/ load_mode,int_corr_mode,maxint
63      namelist /stoch_int_params/ mode
64
65
66      OPEN(8, file="stoch_params.nml", status='OLD', recl=80, delim='APOSTROPHE')
67      READ(8,nml=mtvparams)
68      READ(8,nml=wlparams)
69      READ(8,nml=stparams)
70      READ(8,nml=stoch_int_params)
71      READ(8,nml=corr_init_mode)
72      CLOSE(8)
73
74      dtn=sqrt(dt)
75      dtsn=sqrt(dts)
76      mems=ceiling(meml/muti)
77
78      q_au=q_au/tdelta
79      q_ou=q_ou/tdelta
80
```

7.23.3 Variable Documentation

7.23.3.1 real(kind=8) stoch_params::dtn

Square root of the timestep.

Definition at line 32 of file stoch_params.f90.

```
32    REAL(KIND=8) :: dtn                !< Square root of the timestep
```

7.23.3.2 real(kind=8) stoch_params::dts

Intrisic resolved dynamics time step.

Definition at line 40 of file stoch_params.f90.

```
40    REAL(KIND=8) :: dts                !< Intrisic resolved dynamics time step
```

7.23.3.3 real(kind=8) stoch_params::dtsn

Square root of the intrisic resolved dynamics time step.

Definition at line 43 of file stoch_params.f90.

```
43    REAL(KIND=8) :: dtsn               !< Square root of the intrisic resolved dynamics time step
```

7.23.3.4 real(kind=8) stoch_params::eps_pert

Perturbation parameter for the coupling.

Definition at line 33 of file stoch_params.f90.

```
33  REAL(KIND=8) :: eps_pert          !< Perturbation parameter for the coupling
```

7.23.3.5 character(len=4) stoch_params::int_corr_mode

Correlation integration mode.

Definition at line 47 of file stoch_params.f90.

```
47  CHARACTER(LEN=4) :: int_corr_mode  !< Correlation integration mode
```

7.23.3.6 character(len=4) stoch_params::load_mode

Loading mode for the correlations.

Definition at line 46 of file stoch_params.f90.

```
46  CHARACTER(LEN=4) :: load_mode      !< Loading mode for the correlations
```

7.23.3.7 real(kind=8) stoch_params::maxint

Upper integration limit of the correlations.

Definition at line 45 of file stoch_params.f90.

```
45  REAL(KIND=8) :: maxint             !< Upper integration limit of the correlations
```

7.23.3.8 real(kind=8) stoch_params::meml

Time over which the memory kernel is integrated.

Definition at line 37 of file stoch_params.f90.

```
37  REAL(KIND=8) :: meml               !< Time over which the memory kernel is integrated
```

7.23.3.9 integer stoch_params::mems

Number of steps in the memory kernel integral.

Definition at line 42 of file stoch_params.f90.

```
42    INTEGER :: mems                    !< Number of steps in the memory kernel integral
```

7.23.3.10 real(kind=8) stoch_params::mnuti

Multiplicative noise update time interval.

Definition at line 25 of file stoch_params.f90.

```
25    REAL(KIND=8) :: mnuti              !< Multiplicative noise update time interval
```

7.23.3.11 character(len=4) stoch_params::mode

Stochastic mode parameter.

Definition at line 49 of file stoch_params.f90.

```
49    CHARACTER(len=4) :: mode           !< Stochastic mode parameter
```

7.23.3.12 real(kind=8) stoch_params::muti

Memory update time interval.

Definition at line 36 of file stoch_params.f90.

```
36    REAL(KIND=8) :: muti                !< Memory update time interval
```

7.23.3.13 real(kind=8) stoch_params::q_ar

Atmospheric resolved component noise amplitude.

Definition at line 28 of file stoch_params.f90.

```
28    REAL(KIND=8) :: q_ar               !< Atmospheric resolved component noise amplitude
```

7.23.3.14 real(kind=8) stoch_params::q_au

Atmospheric unresolved component noise amplitude.

Definition at line 29 of file stoch_params.f90.

```
29   REAL(KIND=8) :: q_au               !< Atmospheric unresolved component noise amplitude
```

7.23.3.15 real(kind=8) stoch_params::q_or

Oceanic resolved component noise amplitude.

Definition at line 30 of file stoch_params.f90.

```
30   REAL(KIND=8) :: q_or               !< Oceanic resolved component noise amplitude
```

7.23.3.16 real(kind=8) stoch_params::q_ou

Oceanic unresolved component noise amplitude.

Definition at line 31 of file stoch_params.f90.

```
31   REAL(KIND=8) :: q_ou               !< Oceanic unresolved component noise amplitude
```

7.23.3.17 real(kind=8) stoch_params::t_trans_mem

Transient time period to initialize the memory term.

Definition at line 38 of file stoch_params.f90.

```
38   REAL(KIND=8) :: t_trans_mem        !< Transient time period to initialize the memory term
```

7.23.3.18 real(kind=8) stoch_params::t_trans_stoch

Transient time period of the stochastic model evolution.

Definition at line 27 of file stoch_params.f90.

```
27   REAL(KIND=8) :: t_trans_stoch      !< Transient time period of the stochastic model evolution
```

7.23.3.19 real(kind=8) stoch_params::tdelta

Time separation parameter.

Definition at line 34 of file stoch_params.f90.

```
34    REAL(KIND=8) :: tdelta             !< Time separation parameter
```

7.23.3.20 character(len=4) stoch_params::x_int_mode

Integration mode for the resolved component.

Definition at line 39 of file stoch_params.f90.

```
39    CHARACTER(len=4) :: x_int_mode     !< Integration mode for the resolved component
```

7.24 tensor Module Reference

Tensor utility module.

Data Types

- type coolist

  Coordinate list. Type used to represent the sparse tensor.
- type coolist4

  4d coordinate list. Type used to represent the rank-4 sparse tensor.
- type coolist_elem

  Coordinate list element type. Elementary elements of the sparse tensors.
- type coolist_elem4

  4d coordinate list element type. Elementary elements of the 4d sparse tensors.

Functions/Subroutines

- subroutine, public copy_tensor (src, dst)

  Routine to copy a rank-3 tensor.
- subroutine, public add_to_tensor (src, dst)

  Routine to add a rank-3 tensor to another one.
- subroutine, public add_matc_to_tensor (i, src, dst)

  Routine to add a matrix to a rank-3 tensor.
- subroutine, public add_matc_to_tensor4 (i, j, src, dst)

  Routine to add a matrix to a rank-4 tensor.
- subroutine, public add_vec_jk_to_tensor (j, k, src, dst)

  Routine to add a vector to a rank-3 tensor.
- subroutine, public add_vec_ikl_to_tensor4_perm (i, k, l, src, dst)

  Routine to add a vector to a rank-4 tensor plus permutation.
- subroutine, public add_vec_ikl_to_tensor4 (i, k, l, src, dst)

Routine to add a vector to a rank-4 tensor.

- subroutine, public add_vec_ijk_to_tensor4 (i, j, k, src, dst)

  Routine to add a vector to a rank-4 tensor.

- subroutine, public mat_to_coo (src, dst)

  Routine to convert a matrix to a rank-3 tensor.

- subroutine, public tensor_to_coo (src, dst)

  Routine to convert a rank-3 tensor from matrix to coolist representation.

- subroutine, public tensor4_to_coo4 (src, dst)

  Routine to convert a rank-4 tensor from matrix to coolist representation.

- subroutine, public print_tensor (t)

  Routine to print a rank 3 tensor coolist.

- subroutine, public print_tensor4 (t)

  Routine to print a rank-4 tensor coolist.

- subroutine, public sparse_mul3 (coolist_ijk, arr_j, arr_k, res)

  Sparse multiplication of a rank-3 tensor coolist with two vectors: $\sum_{j,k=0}^{ndim} \mathcal{T}_{i,j,k}\, a_j\, b_k$.

- subroutine, public sparse_mul3_mat (coolist_ijk, arr_k, res)

  Sparse multiplication of a rank-3 tensor coolist with a vector: $\sum_{k=0}^{ndim} \mathcal{T}_{i,j,k}\, b_k$. *Its output is a matrix.*

- subroutine, public sparse_mul4 (coolist_ijkl, arr_j, arr_k, arr_l, res)

  Sparse multiplication of a rank-4 tensor coolist with three vectors: $\sum_{j,k,l=0}^{ndim} \mathcal{T}_{i,j,k,l}\, a_j\, b_k\, c_l$.

- subroutine, public sparse_mul4_mat (coolist_ijkl, arr_k, arr_l, res)

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

- subroutine, public jsparse_mul (coolist_ijk, arr_j, jcoo_ij)

  Sparse multiplication of two tensors to determine the Jacobian:

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

  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:

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

  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 sparse_mul2_j (coolist_ijk, arr_j, res)

  Sparse multiplication of a 3d sparse tensor with a vectors: $\sum_{j=0}^{ndim} \mathcal{T}_{i,j,k}\, a_j$.

- subroutine, public sparse_mul2_k (coolist_ijk, arr_k, res)

  Sparse multiplication of a rank-3 sparse tensor coolist with a vector: $\sum_{k=0}^{ndim} \mathcal{T}_{i,j,k}\, a_k$.

- subroutine, public simplify (tensor)

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

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

.

- subroutine, public coo_to_mat_ik (src, dst)

  Routine to convert a rank-3 tensor coolist component into a matrix with i and k indices.
- subroutine, public coo_to_mat_ij (src, dst)

  Routine to convert a rank-3 tensor coolist component into a matrix with i and j indices.
- subroutine, public coo_to_mat_i (i, src, dst)

  Routine to convert a rank-3 tensor coolist component into a matrix.
- subroutine, public coo_to_vec_jk (j, k, src, dst)

  Routine to convert a rank-3 tensor coolist component into a vector.
- subroutine, public coo_to_mat_j (j, src, dst)

  Routine to convert a rank-3 tensor coolist component into a matrix.
- subroutine, public sparse_mul4_with_mat_jl (coolist_ijkl, mat_jl, res)

  Sparse multiplication of a rank-4 tensor coolist with a matrix : $\sum_{j,l=0}^{ndim} \mathcal{T}_{i,j,k,l} \, m_{j,l}.$
- subroutine, public sparse_mul4_with_mat_kl (coolist_ijkl, mat_kl, res)

  Sparse multiplication of a rank-4 tensor coolist with a matrix : $\sum_{j,l=0}^{ndim} \mathcal{T}_{i,j,k,l} \, m_{k,l}.$
- subroutine, public sparse_mul3_with_mat (coolist_ijk, mat_jk, res)

  Sparse multiplication of a rank-3 tensor coolist with a matrix: $\sum_{j,k=0}^{ndim} \mathcal{T}_{i,j,k} \, m_{j,k}.$
- subroutine, public matc_to_coo (src, dst)

  Routine to convert a matrix to a rank-3 tensor.
- subroutine, public scal_mul_coo (s, t)

  Routine to multiply a rank-3 tensor by a scalar.
- logical function, public tensor_empty (t)

  Test if a rank-3 tensor coolist is empty.
- logical function, public tensor4_empty (t)

  Test if a rank-4 tensor coolist is empty.
- subroutine, public load_tensor4_from_file (s, t)

  Load a rank-4 tensor coolist from a file definition.
- subroutine, public write_tensor4_to_file (s, t)

  Load a rank-4 tensor coolist from a file definition.

Variables

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

  Parameter to test the equality with zero.

7.24.1 Detailed Description

Tensor utility module.

Copyright

Remarks

coolist is a type and also means "coordinate list"

7.24.2 Function/Subroutine Documentation

7.24.2.1 subroutine, public tensor::add_matc_to_tensor (integer, intent(in) *i*, real(kind=8), dimension(ndim,ndim), intent(in) *src*, type(coolist), dimension(ndim), intent(inout) *dst*)

Routine to add a matrix to a rank-3 tensor.

Parameters

i	Add to tensor component i
src	Matrix to add
dst	Destination tensor

Definition at line 144 of file tensor.f90.

```
144     INTEGER, INTENT(IN) :: i
145     REAL(KIND=8), DIMENSION(ndim,ndim), INTENT(IN) :: src
146     TYPE(coolist), DIMENSION(ndim), INTENT(INOUT) :: dst
147     TYPE(coolist_elem), DIMENSION(:), ALLOCATABLE :: celems
148     INTEGER :: j,k,r,n,nsrc,allocstat
149
150     nsrc=0
151     DO j=1,ndim
152        DO k=1,ndim
153           IF (abs(src(j,k))>real_eps) nsrc=nsrc+1
154        END DO
155     END DO
156
157     IF (dst(i)%nelems==0) THEN
158        IF (ALLOCATED(dst(i)%elems)) THEN
159           DEALLOCATE(dst(i)%elems, stat=allocstat)
160           IF (allocstat /= 0) stop "*** Deallocation problem ! ***"
161        ENDIF
162        ALLOCATE(dst(i)%elems(nsrc), stat=allocstat)
163        IF (allocstat /= 0) stop "*** Not enough memory ! ***"
164        n=0
165     ELSE
166        n=dst(i)%nelems
167        ALLOCATE(celems(n), stat=allocstat)
168        DO j=1,n
169           celems(j)%j=dst(i)%elems(j)%j
170           celems(j)%k=dst(i)%elems(j)%k
171           celems(j)%v=dst(i)%elems(j)%v
172        ENDDO
173        DEALLOCATE(dst(i)%elems, stat=allocstat)
174        IF (allocstat /= 0) stop "*** Deallocation problem ! ***"
175        ALLOCATE(dst(i)%elems(nsrc+n), stat=allocstat)
176        IF (allocstat /= 0) stop "*** Not enough memory ! ***"
177        DO j=1,n
178           dst(i)%elems(j)%j=celems(j)%j
179           dst(i)%elems(j)%k=celems(j)%k
180           dst(i)%elems(j)%v=celems(j)%v
181        ENDDO
182        DEALLOCATE(celems, stat=allocstat)
183        IF (allocstat /= 0) stop "*** Deallocation problem ! ***"
184     ENDIF
185     r=0
186     DO j=1,ndim
187        DO k=1,ndim
188           IF (abs(src(j,k))>real_eps) THEN
189              r=r+1
190              dst(i)%elems(n+r)%j=j
191              dst(i)%elems(n+r)%k=k
192              dst(i)%elems(n+r)%v=src(j,k)
193           ENDIF
194        ENDDO
195     END DO
196     dst(i)%nelems=nsrc+n
197
```

7.24.2.2 subroutine, public tensor::add_matc_to_tensor4 (integer, intent(in) *i*, integer, intent(in) *j*, real(kind=8), dimension(ndim,ndim), intent(in) *src*, type(coolist4), dimension(ndim), intent(inout) *dst*)

Routine to add a matrix to a rank-4 tensor.

Parameters

i	Add to tensor component i,j
j	Add to tensor component i,j
src	Matrix to add
dst	Destination tensor

Definition at line 206 of file tensor.f90.

```
206     INTEGER, INTENT(IN) :: i,j
207     REAL(KIND=8), DIMENSION(ndim,ndim), INTENT(IN) :: src
208     TYPE(coolist4), DIMENSION(ndim), INTENT(INOUT) :: dst
209     TYPE(coolist_elem4), DIMENSION(:), ALLOCATABLE :: celems
210     INTEGER :: k,l,r,n,nsrc,allocstat
211
212     nsrc=0
213     DO k=1,ndim
214        DO l=1,ndim
215           IF (abs(src(k,l))>real_eps) nsrc=nsrc+1
216        END DO
217     END DO
218
219     IF (dst(i)%nelems==0) THEN
220        IF (ALLOCATED(dst(i)%elems)) THEN
221           DEALLOCATE(dst(i)%elems, stat=allocstat)
222           IF (allocstat /= 0) stop "*** Deallocation problem ! ***"
223        ENDIF
224        ALLOCATE(dst(i)%elems(nsrc), stat=allocstat)
225        IF (allocstat /= 0) stop "*** Not enough memory ! ***"
226        n=0
227     ELSE
228        n=dst(i)%nelems
229        ALLOCATE(celems(n), stat=allocstat)
230        DO k=1,n
231           celems(k)%j=dst(i)%elems(k)%j
232           celems(k)%k=dst(i)%elems(k)%k
233           celems(k)%l=dst(i)%elems(k)%l
234           celems(k)%v=dst(i)%elems(k)%v
235        ENDDO
236        DEALLOCATE(dst(i)%elems, stat=allocstat)
237        IF (allocstat /= 0) stop "*** Deallocation problem ! ***"
238        ALLOCATE(dst(i)%elems(nsrc+n), stat=allocstat)
239        IF (allocstat /= 0) stop "*** Not enough memory ! ***"
240        DO k=1,n
241           dst(i)%elems(k)%j=celems(k)%j
242           dst(i)%elems(k)%k=celems(k)%k
243           dst(i)%elems(k)%l=celems(k)%l
244           dst(i)%elems(k)%v=celems(k)%v
245        ENDDO
246        DEALLOCATE(celems, stat=allocstat)
247        IF (allocstat /= 0) stop "*** Deallocation problem ! ***"
248     ENDIF
249     r=0
250     DO k=1,ndim
251        DO l=1,ndim
252           IF (abs(src(k,l))>real_eps) THEN
253              r=r+1
254              dst(i)%elems(n+r)%j=j
255              dst(i)%elems(n+r)%k=k
256              dst(i)%elems(n+r)%l=l
257              dst(i)%elems(n+r)%v=src(k,l)
258           ENDIF
259        ENDDO
260     END DO
261     dst(i)%nelems=nsrc+n
262
```

7.24.2.3 subroutine, public tensor::add_to_tensor (type(coolist), dimension(ndim), intent(in) *src*, type(coolist), dimension(ndim), intent(inout) *dst*)

Routine to add a rank-3 tensor to another one.

Parameters

src	Tensor to add
dst	Destination tensor

Definition at line 92 of file tensor.f90.

```
92     TYPE(coolist), DIMENSION(ndim), INTENT(IN) :: src
93     TYPE(coolist), DIMENSION(ndim), INTENT(INOUT) :: dst
94     TYPE(coolist_elem), DIMENSION(:), ALLOCATABLE :: celems
95     INTEGER :: i,j,n,allocstat
96
97     DO i=1,ndim
98        IF (src(i)%nelems/=0) THEN
99           IF (dst(i)%nelems==0) THEN
100             IF (ALLOCATED(dst(i)%elems)) THEN
101                DEALLOCATE(dst(i)%elems, stat=allocstat)
102                IF (allocstat /= 0) stop "*** Deallocation problem ! ***"
103             ENDIF
104             ALLOCATE(dst(i)%elems(src(i)%nelems), stat=allocstat)
105             IF (allocstat /= 0) stop "*** Not enough memory ! ***"
106             n=0
107          ELSE
108             n=dst(i)%nelems
109             ALLOCATE(celems(n), stat=allocstat)
110             DO j=1,n
111                celems(j)%j=dst(i)%elems(j)%j
112                celems(j)%k=dst(i)%elems(j)%k
113                celems(j)%v=dst(i)%elems(j)%v
114             ENDDO
115             DEALLOCATE(dst(i)%elems, stat=allocstat)
116             IF (allocstat /= 0) stop "*** Deallocation problem ! ***"
117             ALLOCATE(dst(i)%elems(src(i)%nelems+n), stat=allocstat)
118             IF (allocstat /= 0) stop "*** Not enough memory ! ***"
119             DO j=1,n
120                dst(i)%elems(j)%j=celems(j)%j
121                dst(i)%elems(j)%k=celems(j)%k
122                dst(i)%elems(j)%v=celems(j)%v
123             ENDDO
124             DEALLOCATE(celems, stat=allocstat)
125             IF (allocstat /= 0) stop "*** Deallocation problem ! ***"
126          ENDIF
127          DO j=1,src(i)%nelems
128             dst(i)%elems(n+j)%j=src(i)%elems(j)%j
129             dst(i)%elems(n+j)%k=src(i)%elems(j)%k
130             dst(i)%elems(n+j)%v=src(i)%elems(j)%v
131          ENDDO
132          dst(i)%nelems=src(i)%nelems+n
133       ENDIF
134    ENDDO
135
```

7.24.2.4 subroutine, public tensor::add_vec_ijk_to_tensor4 (integer, intent(in) *i*, integer, intent(in) *j*, integer, intent(in) *k*, real(kind=8), dimension(ndim), intent(in) *src*, type(coolist4), dimension(ndim), intent(inout) *dst*)

Routine to add a vector to a rank-4 tensor.

Parameters

i,j,k	Add to tensor component i,j and k
src	Vector to add
dst	Destination tensor

Definition at line 454 of file tensor.f90.

```
454      INTEGER, INTENT(IN) :: i,j,k
```

```
    REAL(KIND=8), DIMENSION(ndim), INTENT(IN) :: src
    TYPE(coolist4), DIMENSION(ndim), INTENT(INOUT) :: dst
    TYPE(coolist_elem4), DIMENSION(:), ALLOCATABLE :: celems
    INTEGER :: l,ne,r,n,nsrc,allocstat

    nsrc=0
    DO l=1,ndim
       IF (abs(src(l))>real_eps) nsrc=nsrc+1
    ENDDO

    IF (dst(i)%nelems==0) THEN
       IF (ALLOCATED(dst(i)%elems)) THEN
          DEALLOCATE(dst(i)%elems, stat=allocstat)
          IF (allocstat /= 0) stop "*** Deallocation problem ! ***"
       ENDIF
       ALLOCATE(dst(i)%elems(nsrc), stat=allocstat)
       IF (allocstat /= 0) stop "*** Not enough memory ! ***"
       n=0
    ELSE
       n=dst(i)%nelems
       ALLOCATE(celems(n), stat=allocstat)
       DO ne=1,n
          celems(ne)%j=dst(i)%elems(ne)%j
          celems(ne)%k=dst(i)%elems(ne)%k
          celems(ne)%l=dst(i)%elems(ne)%l
          celems(ne)%v=dst(i)%elems(ne)%v
       ENDDO
       DEALLOCATE(dst(i)%elems, stat=allocstat)
       IF (allocstat /= 0) stop "*** Deallocation problem ! ***"
       ALLOCATE(dst(i)%elems(nsrc+n), stat=allocstat)
       IF (allocstat /= 0) stop "*** Not enough memory ! ***"
       DO ne=1,n
          dst(i)%elems(ne)%j=celems(ne)%j
          dst(i)%elems(ne)%k=celems(ne)%k
          dst(i)%elems(ne)%l=celems(ne)%l
          dst(i)%elems(ne)%v=celems(ne)%v
       ENDDO
       DEALLOCATE(celems, stat=allocstat)
       IF (allocstat /= 0) stop "*** Deallocation problem ! ***"
    ENDIF
    r=0
    DO l=1,ndim
       IF (abs(src(l))>real_eps) THEN
          r=r+1
          dst(i)%elems(n+r)%j=j
          dst(i)%elems(n+r)%k=k
          dst(i)%elems(n+r)%l=l
          dst(i)%elems(n+r)%v=src(l)
       ENDIF
    ENDDO
    dst(i)%nelems=nsrc+n
```

7.24.2.5 subroutine, public tensor::add_vec_ikl_to_tensor4 (integer, intent(in) *i*, integer, intent(in) *k*, integer, intent(in) *l*, real(kind=8), dimension(ndim), intent(in) *src*, type(coolist4), dimension(ndim), intent(inout) *dst*)

Routine to add a vector to a rank-4 tensor.

Parameters

i,k,l	Add to tensor component i,k and l
src	Vector to add
dst	Destination tensor

Definition at line 395 of file tensor.f90.

```
    INTEGER, INTENT(IN) :: i,k,l
    REAL(KIND=8), DIMENSION(ndim), INTENT(IN) :: src
    TYPE(coolist4), DIMENSION(ndim), INTENT(INOUT) :: dst
    TYPE(coolist_elem4), DIMENSION(:), ALLOCATABLE :: celems
    INTEGER :: j,ne,r,n,nsrc,allocstat

    nsrc=0
    DO j=1,ndim
```

```
403          IF (abs(src(j))>real_eps) nsrc=nsrc+1
404       ENDDO
405
406       IF (dst(i)%nelems==0) THEN
407          IF (ALLOCATED(dst(i)%elems)) THEN
408             DEALLOCATE(dst(i)%elems, stat=allocstat)
409             IF (allocstat /= 0) stop "*** Deallocation problem ! ***"
410          ENDIF
411          ALLOCATE(dst(i)%elems(nsrc), stat=allocstat)
412          IF (allocstat /= 0) stop "*** Not enough memory ! ***"
413          n=0
414       ELSE
415          n=dst(i)%nelems
416          ALLOCATE(celems(n), stat=allocstat)
417          DO ne=1,n
418             celems(ne)%j=dst(i)%elems(ne)%j
419             celems(ne)%k=dst(i)%elems(ne)%k
420             celems(ne)%l=dst(i)%elems(ne)%l
421             celems(ne)%v=dst(i)%elems(ne)%v
422          ENDDO
423          DEALLOCATE(dst(i)%elems, stat=allocstat)
424          IF (allocstat /= 0) stop "*** Deallocation problem ! ***"
425          ALLOCATE(dst(i)%elems(nsrc+n), stat=allocstat)
426          IF (allocstat /= 0) stop "*** Not enough memory ! ***"
427          DO ne=1,n
428             dst(i)%elems(ne)%j=celems(ne)%j
429             dst(i)%elems(ne)%k=celems(ne)%k
430             dst(i)%elems(ne)%l=celems(ne)%l
431             dst(i)%elems(ne)%v=celems(ne)%v
432          ENDDO
433          DEALLOCATE(celems, stat=allocstat)
434          IF (allocstat /= 0) stop "*** Deallocation problem ! ***"
435       ENDIF
436       r=0
437       DO j=1,ndim
438          IF (abs(src(j))>real_eps) THEN
439             r=r+1
440             dst(i)%elems(n+r)%j=j
441             dst(i)%elems(n+r)%k=k
442             dst(i)%elems(n+r)%l=l
443             dst(i)%elems(n+r)%v=src(j)
444          ENDIF
445       ENDDO
446       dst(i)%nelems=nsrc+n
```

7.24.2.6 subroutine, public tensor::add_vec_ikl_to_tensor4_perm (integer, intent(in) *i*, integer, intent(in) *k*, integer, intent(in) *l*, real(kind=8), dimension(ndim), intent(in) *src*, type(coolist4), dimension(ndim), intent(inout) *dst*)

Routine to add a vector to a rank-4 tensor plus permutation.

Parameters

i,k,l	Add to tensor component i,k and l
src	Vector to add
dst	Destination tensor

Definition at line 326 of file tensor.f90.

```
326       INTEGER, INTENT(IN) :: i,k,l
327       REAL(KIND=8), DIMENSION(ndim), INTENT(IN) :: src
328       TYPE(coolist4), DIMENSION(ndim), INTENT(INOUT) :: dst
329       TYPE(coolist_elem4), DIMENSION(:), ALLOCATABLE :: celems
330       INTEGER :: j,ne,r,n,nsrc,allocstat
331
332       nsrc=0
333       DO j=1,ndim
334          IF (abs(src(j))>real_eps) nsrc=nsrc+1
335       ENDDO
336       nsrc=nsrc*3
337       IF (dst(i)%nelems==0) THEN
338          IF (ALLOCATED(dst(i)%elems)) THEN
339             DEALLOCATE(dst(i)%elems, stat=allocstat)
340             IF (allocstat /= 0) stop "*** Deallocation problem ! ***"
```

```
            ENDIF
            ALLOCATE(dst(i)%elems(nsrc), stat=allocstat)
            IF (allocstat /= 0) stop "*** Not enough memory ! ***"
            n=0
         ELSE
            n=dst(i)%nelems
            ALLOCATE(celems(n), stat=allocstat)
            DO ne=1,n
               celems(ne)%j=dst(i)%elems(ne)%j
               celems(ne)%k=dst(i)%elems(ne)%k
               celems(ne)%l=dst(i)%elems(ne)%l
               celems(ne)%v=dst(i)%elems(ne)%v
            ENDDO
            DEALLOCATE(dst(i)%elems, stat=allocstat)
            IF (allocstat /= 0) stop "*** Deallocation problem ! ***"
            ALLOCATE(dst(i)%elems(nsrc+n), stat=allocstat)
            IF (allocstat /= 0) stop "*** Not enough memory ! ***"
            DO ne=1,n
               dst(i)%elems(ne)%j=celems(ne)%j
               dst(i)%elems(ne)%k=celems(ne)%k
               dst(i)%elems(ne)%l=celems(ne)%l
               dst(i)%elems(ne)%v=celems(ne)%v
            ENDDO
            DEALLOCATE(celems, stat=allocstat)
            IF (allocstat /= 0) stop "*** Deallocation problem ! ***"
         ENDIF
         r=0
         DO j=1,ndim
            IF (abs(src(j))>real_eps) THEN
               r=r+1
               dst(i)%elems(n+r)%j=j
               dst(i)%elems(n+r)%k=k
               dst(i)%elems(n+r)%l=l
               dst(i)%elems(n+r)%v=src(j)
               r=r+1
               dst(i)%elems(n+r)%j=k
               dst(i)%elems(n+r)%k=l
               dst(i)%elems(n+r)%l=j
               dst(i)%elems(n+r)%v=src(j)
               r=r+1
               dst(i)%elems(n+r)%j=l
               dst(i)%elems(n+r)%k=j
               dst(i)%elems(n+r)%l=k
               dst(i)%elems(n+r)%v=src(j)
            ENDIF
         ENDDO
         dst(i)%nelems=nsrc+n
```

7.24.2.7 subroutine, public tensor::add_vec_jk_to_tensor (integer, intent(in) *j*, integer, intent(in) *k*, real(kind=8), dimension(ndim), intent(in) *src*, type(coolist), dimension(ndim), intent(inout) *dst*)

Routine to add a vector to a rank-3 tensor.

Parameters

j,k	Add to tensor component j and k
src	Vector to add
dst	Destination tensor

Definition at line 271 of file tensor.f90.

```
         INTEGER, INTENT(IN) :: j,k
         REAL(KIND=8), DIMENSION(ndim), INTENT(IN) :: src
         TYPE(coolist), DIMENSION(ndim), INTENT(INOUT) :: dst
         TYPE(coolist_elem), DIMENSION(:), ALLOCATABLE :: celems
         INTEGER :: i,l,r,n,nsrc,allocstat

         DO i=1,ndim
            nsrc=0
            IF (abs(src(i))>real_eps) nsrc=1
            IF (dst(i)%nelems==0) THEN
               IF (ALLOCATED(dst(i)%elems)) THEN
                  DEALLOCATE(dst(i)%elems, stat=allocstat)
```

```
            IF (allocstat /= 0) stop "*** Deallocation problem ! ***"
         ENDIF
         ALLOCATE(dst(i)%elems(nsrc), stat=allocstat)
         IF (allocstat /= 0) stop "*** Not enough memory ! ***"
         n=0
      ELSE
         n=dst(i)%nelems
         ALLOCATE(celems(n), stat=allocstat)
         DO l=1,n
            celems(l)%j=dst(i)%elems(l)%j
            celems(l)%k=dst(i)%elems(l)%k
            celems(l)%v=dst(i)%elems(l)%v
         ENDDO
         DEALLOCATE(dst(i)%elems, stat=allocstat)
         IF (allocstat /= 0) stop "*** Deallocation problem ! ***"
         ALLOCATE(dst(i)%elems(nsrc+n), stat=allocstat)
         IF (allocstat /= 0) stop "*** Not enough memory ! ***"
         DO l=1,n
            dst(i)%elems(l)%j=celems(l)%j
            dst(i)%elems(l)%k=celems(l)%k
            dst(i)%elems(l)%v=celems(l)%v
         ENDDO
         DEALLOCATE(celems, stat=allocstat)
         IF (allocstat /= 0) stop "*** Deallocation problem ! ***"
      ENDIF
      r=0
      IF (abs(src(i))>real_eps) THEN
         r=r+1
         dst(i)%elems(n+r)%j=j
         dst(i)%elems(n+r)%k=k
         dst(i)%elems(n+r)%v=src(i)
      ENDIF
      dst(i)%nelems=nsrc+n
   END DO
```

7.24.2.8 subroutine, public tensor::coo_to_mat_i (integer, intent(in) *i*, type(coolist), dimension(ndim), intent(in) *src*, real(kind=8), dimension(ndim,ndim), intent(out) *dst*)

Routine to convert a rank-3 tensor coolist component into a matrix.

Parameters

i	Component to convert
src	Source tensor
dst	Destination matrix

Definition at line 971 of file tensor.f90.

```
   INTEGER, INTENT(IN) :: i
   TYPE(coolist), DIMENSION(ndim), INTENT(IN) :: src
   REAL(KIND=8), DIMENSION(ndim,ndim), INTENT(OUT) :: dst
   INTEGER :: n

   dst=0.d0
   DO n=1,src(i)%nelems
      dst(src(i)%elems(n)%j,src(i)%elems(n)%k)=src(i)%elems(n)%v
   ENDDO
```

7.24.2.9 subroutine, public tensor::coo_to_mat_ij (type(coolist), dimension(ndim), intent(in) *src*, real(kind=8), dimension(ndim,ndim), intent(out) *dst*)

Routine to convert a rank-3 tensor coolist component into a matrix with i and j indices.

Parameters

src	Source tensor
dst	Destination matrix

Definition at line 938 of file tensor.f90.

```
TYPE(coolist), DIMENSION(ndim), INTENT(IN) :: src
REAL(KIND=8), DIMENSION(ndim,ndim), INTENT(OUT) :: dst
INTEGER :: i,n

dst=0.d0
DO i=1,ndim
   DO n=1,src(i)%nelems
      dst(i,src(i)%elems(n)%j)=src(i)%elems(n)%v
   ENDDO
ENDDO
```

7.24.2.10 subroutine, public tensor::coo_to_mat_ik (type(coolist), dimension(ndim), intent(in) *src*, real(kind=8), dimension(ndim,ndim), intent(out) *dst*)

Routine to convert a rank-3 tensor coolist component into a matrix with i and k indices.

Parameters

src	Source tensor
dst	Destination matrix

Definition at line 922 of file tensor.f90.

```
TYPE(coolist), DIMENSION(ndim), INTENT(IN) :: src
REAL(KIND=8), DIMENSION(ndim,ndim), INTENT(OUT) :: dst
INTEGER :: i,n

dst=0.d0
DO i=1,ndim
   DO n=1,src(i)%nelems
      dst(i,src(i)%elems(n)%k)=src(i)%elems(n)%v
   ENDDO
ENDDO
```

7.24.2.11 subroutine, public tensor::coo_to_mat_j (integer, intent(in) *j*, type(coolist), dimension(ndim), intent(in) *src*, real(kind=8), dimension(ndim,ndim), intent(out) *dst*)

Routine to convert a rank-3 tensor coolist component into a matrix.

Parameters

j	Component to convert
src	Source tensor
dst	Destination matrix

Definition at line 1007 of file tensor.f90.

```
1007    INTEGER, INTENT(IN) :: j
1008    TYPE(coolist), DIMENSION(ndim), INTENT(IN) :: src
1009    REAL(KIND=8), DIMENSION(ndim,ndim), INTENT(OUT) :: dst
1010    INTEGER :: i,n
1011
1012    dst=0.d0
1013    DO i=1,ndim
1014       DO n=1,src(i)%nelems
1015          IF (src(i)%elems(n)%j==j) dst(i,src(i)%elems(n)%k)=src(i)%elems(n)%v
1016       ENDDO
1017    END DO
```

7.24.2.12 subroutine, public tensor::coo_to_vec_jk (integer, intent(in) *j*, integer, intent(in) *k*, type(coolist), dimension(ndim), intent(in) *src*, real(kind=8), dimension(ndim), intent(out) *dst*)

Routine to convert a rank-3 tensor coolist component into a vector.

Parameters

j	Component j,k to convert
k	Component j,k to convert
src	Source tensor
dst	Destination vector

Definition at line 988 of file tensor.f90.

```
988    INTEGER, INTENT(IN) :: j,k
989    TYPE(coolist), DIMENSION(ndim), INTENT(IN) :: src
990    REAL(KIND=8), DIMENSION(ndim), INTENT(OUT) :: dst
991    INTEGER :: i,n
992
993    dst=0.d0
994    DO i=1,ndim
995       DO n=1,src(i)%nelems
996          IF ((src(i)%elems(n)%j==j).and.(src(i)%elems(n)%k==k)) dst(i)=src(i)%elems(n)%v
997       END DO
998    ENDDO
```

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

Routine to copy a rank-3 tensor.

Parameters

src	Source tensor
dst	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 71 of file tensor.f90.

```
71    TYPE(coolist), DIMENSION(ndim), INTENT(IN) :: src
```

```fortran
      TYPE(coolist), DIMENSION(ndim), INTENT(OUT) :: dst
      INTEGER :: i,j,allocstat

      DO i=1,ndim
         IF (dst(i)%nelems/=0) stop "*** copy_tensor : Destination coolist not empty ! ***"
         ALLOCATE(dst(i)%elems(src(i)%nelems), stat=allocstat)
         IF (allocstat /= 0) stop "*** Not enough memory ! ***"
         DO j=1,src(i)%nelems
            dst(i)%elems(j)%j=src(i)%elems(j)%j
            dst(i)%elems(j)%k=src(i)%elems(j)%k
            dst(i)%elems(j)%v=src(i)%elems(j)%v
         ENDDO
         dst(i)%nelems=src(i)%nelems
      ENDDO
```

7.24.2.14 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:

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

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

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

Definition at line 767 of file tensor.f90.

```fortran
767       TYPE(coolist), DIMENSION(ndim), INTENT(IN):: coolist_ijk
768       TYPE(coolist), DIMENSION(ndim), INTENT(OUT):: jcoo_ij
769       REAL(KIND=8), DIMENSION(0:ndim), INTENT(IN)  :: arr_j
770       REAL(KIND=8) :: v
771       INTEGER :: i,j,k,n,nj,allocstat
772       DO i=1,ndim
773          IF (jcoo_ij(i)%nelems/=0) stop "*** jsparse_mul : Destination coolist not empty ! ***"
774          nj=2*coolist_ijk(i)%nelems
775          ALLOCATE(jcoo_ij(i)%elems(nj), stat=allocstat)
776          IF (allocstat /= 0) stop "*** Not enough memory ! ***"
777          nj=0
778          DO n=1,coolist_ijk(i)%nelems
779             j=coolist_ijk(i)%elems(n)%j
780             k=coolist_ijk(i)%elems(n)%k
781             v=coolist_ijk(i)%elems(n)%v
782             IF (j /=0) THEN
783                nj=nj+1
784                jcoo_ij(i)%elems(nj)%j=j
785                jcoo_ij(i)%elems(nj)%k=0
786                jcoo_ij(i)%elems(nj)%v=v*arr_j(k)
787             END IF
788
789             IF (k /=0) THEN
790                nj=nj+1
791                jcoo_ij(i)%elems(nj)%j=k
792                jcoo_ij(i)%elems(nj)%k=0
793                jcoo_ij(i)%elems(nj)%v=v*arr_j(j)
794             END IF
795          END DO
796          jcoo_ij(i)%nelems=nj
797       END DO
```

7.24.2.15 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:

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

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

coolist←_ijk	a coordinate list (sparse tensor) of which index 2 or 3 will be contracted.
arr_j	the vector to be contracted with index 2 and then index 3 of ffi_coo_ijk
jcoo_ij	a matrix to store the result of the contraction

Definition at line 810 of file tensor.f90.

```
810     TYPE(coolist), DIMENSION(ndim), INTENT(IN):: coolist_ijk
811     REAL(KIND=8), DIMENSION(ndim,ndim), INTENT(OUT):: jcoo_ij
812     REAL(KIND=8), DIMENSION(0:ndim), INTENT(IN)  :: arr_j
813     REAL(KIND=8) :: v
814     INTEGER :: i,j,k,n
815     jcoo_ij=0.d0
816     DO i=1,ndim
817        DO n=1,coolist_ijk(i)%nelems
818           j=coolist_ijk(i)%elems(n)%j
819           k=coolist_ijk(i)%elems(n)%k
820           v=coolist_ijk(i)%elems(n)%v
821           IF (j /=0) jcoo_ij(i,j)=jcoo_ij(i,j)+v*arr_j(k)
822           IF (k /=0) jcoo_ij(i,k)=jcoo_ij(i,k)+v*arr_j(j)
823        END DO
824     END DO
```

7.24.2.16 subroutine, public tensor::load_tensor4_from_file (character (len=*), intent(in) *s*, type(coolist4), dimension(ndim), intent(out) *t*)

Load a rank-4 tensor coolist from a file definition.

Parameters

s	Filename of the tensor definition file
t	The loaded coolist

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 1181 of file tensor.f90.

```
    CHARACTER (LEN=*), INTENT(IN) :: s
    TYPE(coolist4), DIMENSION(ndim), INTENT(OUT) :: t
    INTEGER :: i,ir,j,k,l,n,allocstat
    REAL(KIND=8) :: v
    OPEN(30,file=s,status='old')
    DO i=1,ndim
       READ(30,*) ir,n
       IF (n /= 0) THEN
          ALLOCATE(t(i)%elems(n), stat=allocstat)
          IF (allocstat /= 0) stop "*** Not enough memory ! ***"
          t(i)%nelems=n
       ENDIF
       DO n=1,t(i)%nelems
          READ(30,*) ir,j,k,l,v
          t(i)%elems(n)%j=j
          t(i)%elems(n)%k=k
          t(i)%elems(n)%l=l
          t(i)%elems(n)%v=v
       ENDDO
    END DO
    CLOSE(30)
```

7.24.2.17 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 rank-3 tensor.

Parameters

src	Source matrix
dst	Destination tensor

Remarks

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

Definition at line 515 of file tensor.f90.

```
515    REAL(KIND=8), DIMENSION(0:ndim,0:ndim), INTENT(IN) :: src
516    TYPE(coolist), DIMENSION(ndim), INTENT(OUT) :: dst
517    INTEGER :: i,j,n,allocstat
518    DO i=1,ndim
519       n=0
520       DO j=1,ndim
521          IF (abs(src(i,j))>real_eps) n=n+1
522       ENDDO
523       IF (n/=0) THEN
524          IF (dst(i)%nelems/=0) stop "*** mat_to_coo : Destination coolist not empty ! ***"
525          ALLOCATE(dst(i)%elems(n), stat=allocstat)
526          IF (allocstat /= 0) stop "*** Not enough memory ! ***"
527          n=0
528          DO j=1,ndim
529             IF (abs(src(i,j))>real_eps) THEN
530                n=n+1
531                dst(i)%elems(n)%j=j
532                dst(i)%elems(n)%k=0
533                dst(i)%elems(n)%v=src(i,j)
534             ENDIF
535          ENDDO
536       ENDIF
537       dst(i)%nelems=n
538    ENDDO
```

7.24.2.18 subroutine, public tensor::matc_to_coo (real(kind=8), dimension(ndim,ndim), intent(in) *src*, type(coolist), dimension(ndim), intent(out) *dst*)

Routine to convert a matrix to a rank-3 tensor.

Parameters

src	Source matrix
dst	Destination tensor

Remarks

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

Definition at line 1103 of file tensor.f90.

```
REAL(KIND=8), DIMENSION(ndim,ndim), INTENT(IN) :: src
TYPE(coolist), DIMENSION(ndim), INTENT(OUT) :: dst
INTEGER :: i,j,n,allocstat
DO i=1,ndim
   n=0
   DO j=1,ndim
      IF (abs(src(i,j))>real_eps) n=n+1
   ENDDO
   IF (n/=0) THEN
      IF (dst(i)%nelems/=0) stop "*** mat_to_coo : Destination coolist not empty ! ***"
      ALLOCATE(dst(i)%elems(n), stat=allocstat)
      IF (allocstat /= 0) stop "*** Not enough memory ! ***"
      n=0
      DO j=1,ndim
         IF (abs(src(i,j))>real_eps) THEN
            n=n+1
            dst(i)%elems(n)%j=0
            dst(i)%elems(n)%k=j
            dst(i)%elems(n)%v=src(i,j)
         ENDIF
      ENDDO
   ENDIF
   dst(i)%nelems=n
ENDDO
```

7.24.2.19 subroutine, public tensor::print_tensor (type(coolist), dimension(ndim), intent(in) *t*)

Routine to print a rank 3 tensor coolist.

Parameters

t	coolist to print

Definition at line 622 of file tensor.f90.

```
USE util, only: str
TYPE(coolist), DIMENSION(ndim), INTENT(IN) :: t
INTEGER :: i,n,j,k
DO i=1,ndim
   DO n=1,t(i)%nelems
      j=t(i)%elems(n)%j
      k=t(i)%elems(n)%k
      IF( abs(t(i)%elems(n)%v) .GE. real_eps) THEN
         write(*,"(A,ES12.5)") "tensor["//trim(str(i))//"]["//trim(str(j)) &
              &//"]["//trim(str(k))//"] = ",t(i)%elems(n)%v
      END IF
   END DO
END DO
```

7.24.2.20 subroutine, public tensor::print_tensor4 (type(coolist4), dimension(ndim), intent(in) *t*)

Routine to print a rank-4 tensor coolist.

Parameters

t	coolist to print

Definition at line 640 of file tensor.f90.

```
640     USE util, only: str
641     TYPE(coolist4), DIMENSION(ndim), INTENT(IN) :: t
642     INTEGER :: i,n,j,k,l
643     DO i=1,ndim
644        DO n=1,t(i)%nelems
645           j=t(i)%elems(n)%j
646           k=t(i)%elems(n)%k
647           l=t(i)%elems(n)%l
648           IF( abs(t(i)%elems(n)%v) .GE. real_eps) THEN
649              write(*,"(A,ES12.5)") "tensor["//trim(str(i))//"]["//trim(str(j)) &
650                 &//"]["//trim(str(k))//"]["//trim(str(l))//"] = ",t(i)%elems(n)%v
651           END IF
652        END DO
653     END DO
```

7.24.2.21 subroutine, public tensor::scal_mul_coo (real(kind=8), intent(in) *s*, type(coolist), dimension(ndim), intent(inout) *t*)

Routine to multiply a rank-3 tensor by a scalar.

Parameters

s	The scalar
t	The tensor

Definition at line 1133 of file tensor.f90.

```
1133    REAL(KIND=8), INTENT(IN) :: s
1134    TYPE(coolist), DIMENSION(ndim), INTENT(INOUT) :: t
1135    INTEGER :: i,li,n
1136    DO i=1,ndim
1137       n=t(i)%nelems
1138       DO li=1,n
1139         t(i)%elems(li)%v=s*t(i)%elems(li)%v
1140       ENDDO
1141    ENDDO
```

7.24.2.22 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

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

.

Parameters

tensor	a coordinate list (sparse tensor) which will be simplified.

Definition at line 874 of file tensor.f90.

```
874     TYPE(coolist), DIMENSION(ndim), INTENT(INOUT):: tensor
875     INTEGER :: i,j,k
876     INTEGER :: li,lii,liii,n
877     DO i= 1,ndim
878        n=tensor(i)%nelems
879        DO li=n,2,-1
880           j=tensor(i)%elems(li)%j
881           k=tensor(i)%elems(li)%k
882           DO lii=li-1,1,-1
883              IF ((j==tensor(i)%elems(lii)%j).AND.(k==tensor(i)%elems(lii)%k)) THEN
884                 ! Found another entry with the same i,j,k: merge both into
885                 ! the one listed first (of those two).
886                 tensor(i)%elems(lii)%v=tensor(i)%elems(lii)%v+tensor(i)%elems(li)%v
887                 ! Shift the rest of the items one place down.
888                 DO liii=li+1,n
889                    tensor(i)%elems(liii-1)%j=tensor(i)%elems(liii)%j
890                    tensor(i)%elems(liii-1)%k=tensor(i)%elems(liii)%k
891                    tensor(i)%elems(liii-1)%v=tensor(i)%elems(liii)%v
892                 END DO
893                 tensor(i)%nelems=tensor(i)%nelems-1
894                 ! Here we should stop because the li no longer points to the
895                 ! original i,j,k element
896                 EXIT
897              ENDIF
898           ENDDO
899        ENDDO
900        n=tensor(i)%nelems
901        DO li=1,n
902           ! Clear new "almost" zero entries and shift rest of the items one place down.
903           ! Make sure not to skip any entries while shifting!
904           DO WHILE (abs(tensor(i)%elems(li)%v) < real_eps)
905              DO liii=li+1,n
906                 tensor(i)%elems(liii-1)%j=tensor(i)%elems(liii)%j
907                 tensor(i)%elems(liii-1)%k=tensor(i)%elems(liii)%k
908                 tensor(i)%elems(liii-1)%v=tensor(i)%elems(liii)%v
909              ENDDO
910              tensor(i)%nelems=tensor(i)%nelems-1
911           ENDDO
912        ENDDO
913
914     ENDDO
```

7.24.2.23 subroutine, public tensor::sparse_mul2_j (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(out) *res*)

Sparse multiplication of a 3d sparse tensor with a vectors: $\sum_{j=0}^{ndim} \mathcal{T}_{i,j,k} \, a_j$.

Parameters

coolist←_ijk	a coordinate list (sparse tensor) of which index 2 will be contracted.
arr_j	the vector to be contracted with index 2 of coolist_ijk
res	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 835 of file tensor.f90.

```
835     TYPE(coolist), DIMENSION(ndim), INTENT(IN):: coolist_ijk
836     REAL(KIND=8), DIMENSION(0:ndim), INTENT(IN)  :: arr_j
837     REAL(KIND=8), DIMENSION(0:ndim), INTENT(OUT) :: res
838     INTEGER :: i,j,n
839     res=0.d0
840     DO i=1,ndim
841        DO n=1,coolist_ijk(i)%nelems
842          j=coolist_ijk(i)%elems(n)%j
843          res(i) = res(i) + coolist_ijk(i)%elems(n)%v * arr_j(j)
844       END DO
845    END DO
```

7.24.2.24 subroutine, public tensor::sparse_mul2_k (type(coolist), dimension(ndim), intent(in) *coolist_ijk*, real(kind=8), dimension(0:ndim), intent(in) *arr_k*, real(kind=8), dimension(0:ndim), intent(out) *res*)

Sparse multiplication of a rank-3 sparse tensor coolist with a vector: $\sum_{k=0}^{ndim} \mathcal{T}_{i,j,k} \, a_k$.

Parameters

coolist_ijk	a coordinate list (sparse tensor) of which index k will be contracted.
arr_k	the vector to be contracted with index k of coolist_ijk
res	vector (buffer) to store the result of the contraction

Remarks

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

Definition at line 856 of file tensor.f90.

```
856     TYPE(coolist), DIMENSION(ndim), INTENT(IN):: coolist_ijk
857     REAL(KIND=8), DIMENSION(0:ndim), INTENT(IN)  :: arr_k
858     REAL(KIND=8), DIMENSION(0:ndim), INTENT(OUT) :: res
859     INTEGER :: i,k,n
860     res=0.d0
861     DO i=1,ndim
862        DO n=1,coolist_ijk(i)%nelems
863          k=coolist_ijk(i)%elems(n)%k
864          res(i) = res(i) + coolist_ijk(i)%elems(n)%v * arr_k(k)
865       END DO
866    END DO
```

7.24.2.25 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 rank-3 tensor coolist with two vectors: $\sum_{j,k=0}^{ndim} \mathcal{T}_{i,j,k} \, a_j \, b_k$.

Parameters

coolist_ijk	a coolist (sparse tensor) of which index 2 and 3 will be contracted.
arr_j	the vector to be contracted with index 2 of coolist_ijk
arr_k	the vector to be contracted with index 3 of coolist_ijk
res	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 666 of file tensor.f90.

```
666     TYPE(coolist), DIMENSION(ndim), INTENT(IN):: coolist_ijk
667     REAL(KIND=8), DIMENSION(0:ndim), INTENT(IN)  :: arr_j, arr_k
668     REAL(KIND=8), DIMENSION(0:ndim), INTENT(OUT) :: res
669     INTEGER :: i,j,k,n
670     res=0.d0
671     DO i=1,ndim
672        DO n=1,coolist_ijk(i)%nelems
673          j=coolist_ijk(i)%elems(n)%j
674          k=coolist_ijk(i)%elems(n)%k
675          res(i) = res(i) + coolist_ijk(i)%elems(n)%v * arr_j(j)*arr_k(k)
676       END DO
677    END DO
```

7.24.2.26 subroutine, public tensor::sparse_mul3_mat (type(coolist), dimension(ndim), intent(in) *coolist_ijk*, real(kind=8), dimension(0:ndim), intent(in) *arr_k*, real(kind=8), dimension(ndim,ndim), intent(out) *res*)

Sparse multiplication of a rank-3 tensor coolist with a vector: $\sum_{k=0}^{ndim} \mathcal{T}_{i,j,k} \, b_k$. Its output is a matrix.

Parameters

coolist_ijk	a coolist (sparse tensor) of which index k will be contracted.
arr_k	the vector to be contracted with index k of coolist_ijk
res	matrix (buffer) to store the result of the contraction

Remarks

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

Definition at line 689 of file tensor.f90.

```
689     TYPE(coolist), DIMENSION(ndim), INTENT(IN):: coolist_ijk
690     REAL(KIND=8), DIMENSION(0:ndim), INTENT(IN)  :: arr_k
691     REAL(KIND=8), DIMENSION(ndim,ndim), INTENT(OUT) :: res
692     INTEGER :: i,j,k,n
693     res=0.d0
694     DO i=1,ndim
695        DO n=1,coolist_ijk(i)%nelems
696          j=coolist_ijk(i)%elems(n)%j
697          IF (j /= 0) THEN
```

```
            k=coolist_ijk(i)%elems(n)%k
            res(i,j) = res(i,j) + coolist_ijk(i)%elems(n)%v * arr_k(k)
         ENDIF
      END DO
   END DO
```

7.24.2.27 subroutine, public tensor::sparse_mul3_with_mat (type(coolist), dimension(ndim), intent(in) *coolist_ijk*, real(kind=8), dimension(ndim,ndim), intent(in) *mat_jk*, real(kind=8), dimension(0:ndim), intent(out) *res*)

Sparse multiplication of a rank-3 tensor coolist with a matrix: $\sum_{j,k=0}^{ndim} \mathcal{T}_{i,j,k} \, m_{j,k}$.

Parameters

coolist←_ijk	a coolist (sparse tensor) of which index j and k will be contracted.
mat_jk	the matrix to be contracted with index j and k of coolist_ijk
res	vector (buffer) to store the result of the contraction

Remarks

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

Definition at line 1079 of file tensor.f90.

```
   TYPE(coolist), DIMENSION(ndim), INTENT(IN):: coolist_ijk
   REAL(KIND=8), DIMENSION(ndim,ndim), INTENT(IN)  :: mat_jk
   REAL(KIND=8), DIMENSION(0:ndim), INTENT(OUT) :: res
   INTEGER i,j,k,n

   res=0.d0
   DO i=1,ndim
      DO n=1,coolist_ijk(i)%nelems
         j=coolist_ijk(i)%elems(n)%j
         k=coolist_ijk(i)%elems(n)%k

         res(i) = res(i) + coolist_ijk(i)%elems(n)%v * mat_jk(j,k)
      ENDDO
   END DO

```

7.24.2.28 subroutine, public tensor::sparse_mul4 (type(coolist4), dimension(ndim), intent(in) *coolist_ijkl*, 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(in) *arr_l*, real(kind=8), dimension(0:ndim), intent(out) *res*)

Sparse multiplication of a rank-4 tensor coolist with three vectors: $\sum_{j,k,l=0}^{ndim} \mathcal{T}_{i,j,k,l} \, a_j \, b_k \, c_l$.

Parameters

coolist_ijkl	a coolist (sparse tensor) of which index j, k and l will be contracted.
arr_j	the vector to be contracted with index j of coolist_ijkl
arr_k	the vector to be contracted with index k of coolist_ijkl
arr_l	the vector to be contracted with index l of coolist_ijkl
res	vector (buffer) to store the result of the contraction

Remarks

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

Definition at line 715 of file tensor.f90.

```
TYPE(coolist4), DIMENSION(ndim), INTENT(IN):: coolist_ijkl
REAL(KIND=8), DIMENSION(0:ndim), INTENT(IN)  :: arr_j, arr_k, arr_l
REAL(KIND=8), DIMENSION(0:ndim), INTENT(OUT) :: res
INTEGER :: i,j,k,n,l
res=0.d0
DO i=1,ndim
   DO n=1,coolist_ijkl(i)%nelems
     j=coolist_ijkl(i)%elems(n)%j
     k=coolist_ijkl(i)%elems(n)%k
     l=coolist_ijkl(i)%elems(n)%l
     res(i) = res(i) + coolist_ijkl(i)%elems(n)%v * arr_j(j)*arr_k(k)*arr_l(l)
  END DO
END DO
```

7.24.2.29 subroutine, public tensor::sparse_mul4_mat (type(coolist4), dimension(ndim), intent(in) *coolist_ijkl,* real(kind=8), dimension(0:ndim), intent(in) *arr_k,* real(kind=8), dimension(0:ndim), intent(in) *arr_l,* real(kind=8), dimension(ndim,ndim), intent(out) *res*)

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

Parameters

coolist_ijkl	a coordinate list (sparse tensor) of which index 3 and 4 will be contracted.
arr_k	the vector to be contracted with index 3 of coolist_ijkl
arr_l	the vector to be contracted with index 4 of coolist_ijkl
res	matrix (buffer) to store the result of the contraction

Remarks

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

Definition at line 739 of file tensor.f90.

```
TYPE(coolist4), DIMENSION(ndim), INTENT(IN):: coolist_ijkl
REAL(KIND=8), DIMENSION(0:ndim), INTENT(IN)  :: arr_k, arr_l
REAL(KIND=8), DIMENSION(ndim,ndim), INTENT(OUT) :: res
INTEGER :: i,j,k,n,l
res=0.d0
DO i=1,ndim
   DO n=1,coolist_ijkl(i)%nelems
     j=coolist_ijkl(i)%elems(n)%j
     IF (j /= 0) THEN
        k=coolist_ijkl(i)%elems(n)%k
        l=coolist_ijkl(i)%elems(n)%l
        res(i,j) = res(i,j) + coolist_ijkl(i)%elems(n)%v * arr_k(k) * arr_l(l)
     ENDIF
  END DO
END DO
```

7.24.2.30 subroutine, public tensor::sparse_mul4_with_mat_jl (type(coolist4), dimension(ndim), intent(in) *coolist_ijkl,* real(kind=8), dimension(ndim,ndim), intent(in) *mat_jl,* real(kind=8), dimension(ndim,ndim), intent(out) *res*)

Sparse multiplication of a rank-4 tensor coolist with a matrix : $\sum_{j,l=0}^{ndim} \mathcal{T}_{i,j,k,l} \, m_{j,l}$.

Parameters

coolist_ijkl	a coolist (sparse tensor) of which index j and l will be contracted.
mat_jl	the matrix to be contracted with indices j and l of coolist_ijkl
res	matrix (buffer) to store the result of the contraction

Remarks

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

Definition at line 1028 of file tensor.f90.

```
1028     TYPE(coolist4), DIMENSION(ndim), INTENT(IN):: coolist_ijkl
1029     REAL(KIND=8), DIMENSION(ndim,ndim), INTENT(IN)   :: mat_jl
1030     REAL(KIND=8), DIMENSION(ndim,ndim), INTENT(OUT)  :: res
1031     INTEGER i,j,k,l,n
1032 
1033     res=0.d0
1034     DO i=1,ndim
1035        DO n=1,coolist_ijkl(i)%nelems
1036           j=coolist_ijkl(i)%elems(n)%j
1037           k=coolist_ijkl(i)%elems(n)%k
1038           l=coolist_ijkl(i)%elems(n)%l
1039 
1040           res(i,k) = res(i,k) + coolist_ijkl(i)%elems(n)%v * mat_jl(j,l)
1041        ENDDO
1042     END DO
1043 
```

7.24.2.31 subroutine, public tensor::sparse_mul4_with_mat_kl (type(coolist4), dimension(ndim), intent(in) *coolist_ijkl,* real(kind=8), dimension(ndim,ndim), intent(in) *mat_kl,* real(kind=8), dimension(ndim,ndim), intent(out) *res*)

Sparse multiplication of a rank-4 tensor coolist with a matrix : $\sum_{j,l=0}^{ndim} \mathcal{T}_{i,j,k,l} \, m_{k,l}$.

Parameters

coolist_ijkl	a coolist (sparse tensor) of which index k and l will be contracted.
mat_kl	the matrix to be contracted with indices k and l of coolist_ijkl
res	matrix (buffer) to store the result of the contraction

Remarks

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

Definition at line 1053 of file tensor.f90.

```
TYPE(coolist4), DIMENSION(ndim), INTENT(IN):: coolist_ijkl
REAL(KIND=8), DIMENSION(ndim,ndim), INTENT(IN)  :: mat_kl
REAL(KIND=8), DIMENSION(ndim,ndim), INTENT(OUT) :: res
INTEGER i,j,k,l,n

res=0.d0
DO i=1,ndim
   DO n=1,coolist_ijkl(i)%nelems
      j=coolist_ijkl(i)%elems(n)%j
      k=coolist_ijkl(i)%elems(n)%k
      l=coolist_ijkl(i)%elems(n)%l

      res(i,j) = res(i,j) + coolist_ijkl(i)%elems(n)%v * mat_kl(k,l)
   ENDDO
END DO

```

7.24.2.32 logical function, public tensor::tensor4_empty (type(coolist4), dimension(ndim), intent(in) *t*)

Test if a rank-4 tensor coolist is empty.

Parameters

t	rank-4 tensor coolist to be tested

Definition at line 1163 of file tensor.f90.

```
1163     TYPE(coolist4), DIMENSION(ndim), INTENT(IN) :: t
1164     LOGICAL :: tensor4_empty
1165     INTEGER :: i
1166     tensor4_empty=.true.
1167     DO i=1,ndim
1168        IF (t(i)%nelems /= 0) THEN
1169           tensor4_empty=.false.
1170           RETURN
1171        ENDIF
1172     END DO
1173     RETURN
```

7.24.2.33 subroutine, public tensor::tensor4_to_coo4 (real(kind=8), dimension(ndim,0:ndim,0:ndim,0:ndim), intent(in) *src*, type(coolist4), dimension(ndim), intent(out) *dst*)

Routine to convert a rank-4 tensor from matrix to coolist representation.

Parameters

src	Source matrix
dst	Destination coolist

Remarks

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

Definition at line 583 of file tensor.f90.

```
583     REAL(KIND=8), DIMENSION(ndim,0:ndim,0:ndim,0:ndim), INTENT(IN) :: src
584     TYPE(coolist4), DIMENSION(ndim), INTENT(OUT) :: dst
```

```
    INTEGER :: i,j,k,l,n,allocstat

    DO i=1,ndim
       n=0
       DO j=0,ndim
          DO k=0,ndim
             DO l=0,ndim
                IF (abs(src(i,j,k,l))>real_eps) n=n+1
             ENDDO
          ENDDO
       ENDDO
       IF (n/=0) THEN
          IF (dst(i)%nelems/=0) stop "*** tensor_to_coo : Destination coolist not empty ! ***"
          ALLOCATE(dst(i)%elems(n), stat=allocstat)
          IF (allocstat /= 0) stop "*** Not enough memory ! ***"
          n=0
          DO j=0,ndim
             DO k=0,ndim
                DO l=0,ndim
                   IF (abs(src(i,j,k,l))>real_eps) THEN
                      n=n+1
                      dst(i)%elems(n)%j=j
                      dst(i)%elems(n)%k=k
                      dst(i)%elems(n)%l=l
                      dst(i)%elems(n)%v=src(i,j,k,l)
                   ENDIF
                ENDDO
             ENDDO
          ENDDO
       ENDIF
       dst(i)%nelems=n
    ENDDO
```

7.24.2.34 logical function, public tensor::tensor_empty (type(coolist), dimension(ndim), intent(in) *t*)

Test if a rank-3 tensor coolist is empty.

Parameters

t	rank-3 tensor coolist to be tested

Definition at line 1147 of file tensor.f90.

```
    TYPE(coolist), DIMENSION(ndim), INTENT(IN) :: t
    LOGICAL :: tensor_empty
    INTEGER :: i
    tensor_empty=.true.
    DO i=1,ndim
       IF (t(i)%nelems /= 0) THEN
          tensor_empty=.false.
          RETURN
       ENDIF
    END DO
    RETURN
```

7.24.2.35 subroutine, public tensor::tensor_to_coo (real(kind=8), dimension(ndim,0:ndim,0:ndim), intent(in) *src*, type(coolist), dimension(ndim), intent(out) *dst*)

Routine to convert a rank-3 tensor from matrix to coolist representation.

Parameters

src	Source matrix
dst	Destination coolist

Remarks

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

Definition at line 547 of file tensor.f90.

```
547       REAL(KIND=8), DIMENSION(ndim,0:ndim,0:ndim), INTENT(IN) :: src
548       TYPE(coolist), DIMENSION(ndim), INTENT(OUT) :: dst
549       INTEGER :: i,j,k,n,allocstat
550
551       DO i=1,ndim
552          n=0
553          DO j=0,ndim
554             DO k=0,ndim
555                IF (abs(src(i,j,k))>real_eps) n=n+1
556             ENDDO
557          ENDDO
558          IF (n/=0) THEN
559             IF (dst(i)%nelems/=0) stop "*** tensor_to_coo : Destination coolist not empty ! ***"
560             ALLOCATE(dst(i)%elems(n), stat=allocstat)
561             IF (allocstat /= 0) stop "*** Not enough memory ! ***"
562             n=0
563             DO j=0,ndim
564                DO k=0,ndim
565                   IF (abs(src(i,j,k))>real_eps) THEN
566                      n=n+1
567                      dst(i)%elems(n)%j=j
568                      dst(i)%elems(n)%k=k
569                      dst(i)%elems(n)%v=src(i,j,k)
570                   ENDIF
571                ENDDO
572             ENDDO
573          ENDIF
574          dst(i)%nelems=n
575       ENDDO
```

7.24.2.36 subroutine, public tensor::write_tensor4_to_file (character (len=*), intent(in) *s*, type(coolist4), dimension(ndim), intent(in) *t*)

Load a rank-4 tensor coolist from a file definition.

Parameters

s	Destination filename
t	The coolist to write

Definition at line 1208 of file tensor.f90.

```
1208      CHARACTER (LEN=*), INTENT(IN) :: s
1209      TYPE(coolist4), DIMENSION(ndim), INTENT(IN) :: t
1210      INTEGER :: i,j,k,l,n
1211      OPEN(30,file=s)
1212      DO i=1,ndim
1213         WRITE(30,*) i,t(i)%nelems
1214         DO n=1,t(i)%nelems
1215            j=t(i)%elems(n)%j
1216            k=t(i)%elems(n)%k
1217            l=t(i)%elems(n)%l
1218            WRITE(30,*) i,j,k,l,t(i)%elems(n)%v
1219         END DO
1220      END DO
1221      CLOSE(30)
```

7.24.3 Variable Documentation

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

Parameter to test the equality with zero.

Definition at line 50 of file tensor.f90.

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

7.25 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.
- 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.

7.25.1 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.

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 bufers will probably have to be defined.

7.25.2 Function/Subroutine Documentation

7.25.2.1 subroutine public 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.

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

Parameters

y	Initial point.
ystar	Adjoint model at the point ystar.
t	Actual integration time
dt	Integration timestep.
res	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
```

7.25.2.2 subroutine public tl_ad_integrator::init_tl_ad_integrator ( )

Routine to initialise the integration buffers.

Routine to initialise the TL-AD integration bufers.

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 ! ***"
```

7.25.2.3 subroutine public 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.

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

Parameters

y	Initial point.
ystar	Adjoint model at the point ystar.
t	Actual integration time
dt	Integration timestep.
res	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 tl(t,ystar,y,buf_f0)
92      buf_y1 = y+dt*buf_f0
93      CALL tl(t+dt,ystar,buf_y1,buf_f1)
94      res=y+0.5*(buf_f0+buf_f1)*dt
95      t=t+dt
```

7.25.3 Variable Documentation

7.25.3.1 real(kind=8), dimension(:), allocatable 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
```

7.25.3.2 real(kind=8), dimension(:), allocatable 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
```

7.25.3.3 real(kind=8), dimension(:), allocatable 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
```

7.25.3.4 real(kind=8), dimension(:), allocatable 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
```

7.25.3.5 real(kind=8), dimension(:), allocatable tl_ad_integrator::buf_y1 [private]

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

Buffer to hold the intermediate position 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
```

7.26 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), public 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, deltay, buf)
    Tendencies for the AD of MAOOAM in point ystar for perturbation deltay.
- subroutine, public tl (t, ystar, deltay, buf)
    Tendencies for the TL of MAOOAM in point ystar for perturbation deltay.

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.

7.26.1 Detailed Description

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

Copyright

Remarks

The routines of this module should be called only after params::init_params() and aotensor_def::init_↩ aotensor() have been called !

7.26.2 Function/Subroutine Documentation

7.26.2.1 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) *deltay*, real(kind=8), dimension(0:ndim), intent(out) *buf*)

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

Parameters

t	time
ystar	vector with the variables (current point in trajectory)
deltay	vector with the perturbation of the variables at time t
buf	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)
```

7.26.2.2 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	tensor i index
j	tensor j index
k	tensor k index
v	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
```

7.26.2.3 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	tensor i index
j	tensor j index
k	tensor k index
v	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
```

7.26.2.4 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	tensor i index
j	tensor j index
k	tensor k index
v	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
```

```
    IF (.NOT. ALLOCATED(adtensor)) stop "*** ad_coeff routine : tensor not yet allocated ***"
    IF ((abs(v) .ge. real_eps).AND.(i /=0)) THEN
       IF (k /=0) THEN
          IF (.NOT. ALLOCATED(adtensor(k)%elems)) stop "*** ad_coeff routine : tensor not yet allocated
     ***"
          n=(adtensor(k)%nelems)+1
          adtensor(k)%elems(n)%j=i
          adtensor(k)%elems(n)%k=j
          adtensor(k)%elems(n)%v=v
          adtensor(k)%nelems=n
       END IF
       IF (j /=0) THEN
          IF (.NOT. ALLOCATED(adtensor(j)%elems)) stop "*** ad_coeff routine : tensor not yet allocated
     ***"
          n=(adtensor(j)%nelems)+1
          adtensor(j)%elems(n)%j=i
          adtensor(j)%elems(n)%k=k
          adtensor(j)%elems(n)%v=v
          adtensor(j)%nelems=n
       END IF
    END IF
```

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

Definition at line 358 of file tl_ad_tensor.f90.

```
    INTEGER, INTENT(IN) :: i,j,k
    REAL(KIND=8), INTENT(IN) :: v
    INTEGER :: n
    IF (.NOT. ALLOCATED(adtensor)) stop "*** ad_coeff_ref routine : tensor not yet allocated ***"
    IF ((abs(v) .ge. real_eps).AND.(j /=0)) THEN
       IF (.NOT. ALLOCATED(adtensor(j)%elems)) stop "*** ad_coeff_ref routine : tensor not yet allocated
     ***"
       n=(adtensor(j)%nelems)+1
       adtensor(j)%elems(n)%j=i
       adtensor(j)%elems(n)%k=k
       adtensor(j)%elems(n)%v=v
       adtensor(j)%nelems=n
    END IF
```

7.26.2.6 subroutine tl_ad_tensor::compute_adtensor (external *func*) [private]

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

Parameters

func	subroutine used to do the computation

Definition at line 217 of file tl_ad_tensor.f90.

7.26.2.7 subroutine tl_ad_tensor::compute_adtensor_ref (external *func*) [private]

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

Parameters

func	subroutine used to do the computation

Definition at line 318 of file tl_ad_tensor.f90.

7.26.2.8 subroutine tl_ad_tensor::compute_tltensor (external *func*) [private]

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

Parameters

func	subroutine used to do the computation

Definition at line 121 of file tl_ad_tensor.f90.

7.26.2.9 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
```

7.26.2.10 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.

```
    INTEGER :: i
    INTEGER :: allocstat
    ALLOCATE(adtensor(ndim),count_elems(ndim), stat=allocstat)
    IF (allocstat /= 0) stop "*** Not enough memory ! ***"
    count_elems=0
    CALL compute_adtensor_ref(ad_add_count_ref)

    DO i=1,ndim
       ALLOCATE(adtensor(i)%elems(count_elems(i)), stat=allocstat)
       IF (allocstat /= 0) stop "*** Not enough memory ! ***"
    END DO

    DEALLOCATE(count_elems, stat=allocstat)
    IF (allocstat /= 0) stop "*** Deallocation problem ! ***"

    CALL compute_adtensor_ref(ad_coeff_ref)

    CALL simplify(adtensor)

```

7.26.2.11 subroutine, public tl_ad_tensor::init_tltensor ( )

Routine to initialize the TL tensor.

Definition at line 97 of file tl_ad_tensor.f90.

```
   INTEGER :: i
   INTEGER :: allocstat
   ALLOCATE(tltensor(ndim),count_elems(ndim), stat=allocstat)
    IF (allocstat /= 0) stop "*** Not enough memory ! ***"
    count_elems=0
    CALL compute_tltensor(tl_add_count)

    DO i=1,ndim
       ALLOCATE(tltensor(i)%elems(count_elems(i)), stat=allocstat)
       IF (allocstat /= 0) stop "*** Not enough memory ! ***"
    END DO

    DEALLOCATE(count_elems, stat=allocstat)
    IF (allocstat /= 0) stop "*** Deallocation problem ! ***"

    CALL compute_tltensor(tl_coeff)

    CALL simplify(tltensor)

```

7.26.2.12 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

ystar	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)
```

7.26.2.13 real(kind=8) function, dimension(ndim,ndim), public tl_ad_tensor::jacobian_mat (real(kind=8), dimension(0:ndim), intent(in) *ystar*)

Compute the Jacobian of MAOOAM in point ystar.

Parameters

ystar	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)
```

7.26.2.14 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) *deltay,* real(kind=8), dimension(0:ndim), intent(out) *buf*)

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

Parameters

t	time
ystar	vector with the variables (current point in trajectory)
deltay	vector with the perturbation of the variables at time t
buf	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)
```

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

Definition at line 147 of file tl_ad_tensor.f90.

```
    INTEGER, INTENT(IN) :: i,j,k
    REAL(KIND=8), INTENT(IN)  :: v
    IF (abs(v) .ge. real_eps) THEN
       IF (j /= 0) count_elems(i)=count_elems(i)+1
       IF (k /= 0) count_elems(i)=count_elems(i)+1
    ENDIF
```

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

Definition at line 161 of file tl_ad_tensor.f90.

```
    INTEGER, INTENT(IN) :: i,j,k
    REAL(KIND=8), INTENT(IN) :: v
    INTEGER :: n
    IF (.NOT. ALLOCATED(tltensor)) stop "*** tl_coeff routine : tensor not yet allocated ***"
    IF (.NOT. ALLOCATED(tltensor(i)%elems)) stop "*** tl_coeff routine : tensor not yet allocated
   ***"
    IF (abs(v) .ge. real_eps) THEN
       IF (j /=0) THEN
          n=(tltensor(i)%nelems)+1
          tltensor(i)%elems(n)%j=j
          tltensor(i)%elems(n)%k=k
          tltensor(i)%elems(n)%v=v
          tltensor(i)%nelems=n
       END IF
       IF (k /=0) THEN
          n=(tltensor(i)%nelems)+1
          tltensor(i)%elems(n)%j=k
          tltensor(i)%elems(n)%k=j
          tltensor(i)%elems(n)%v=v
          tltensor(i)%nelems=n
       END IF
    END IF
```

7.26.3 Variable Documentation

7.26.3.1 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
```

7.26.3.2 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
```

7.26.3.3 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
```

7.26.3.4 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
```

7.27 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.
- real(kind=8) function, public mat_trace (A)
- real(kind=8) function, public mat_contract (A, B)
- subroutine, public choldc (a, p)
- subroutine, public printmat (A)
- subroutine, public cprintmat (A)
- real(kind=8) function, dimension(size(a, 1), size(a, 2)), public invmat (A)
- subroutine, public triu (A, T)
- subroutine, public diag (A, d)
- subroutine, public cdiag (A, d)
- integer function, public floordiv (i, j)
- subroutine, public reduce (A, Ared, n, ind, rind)
- subroutine, public ireduce (A, Ared, n, ind, rind)
- subroutine, public vector_outer (u, v, A)

7.27.1 Detailed Description

Utility module.

Copyright

2015 Lesley De Cruz & Jonathan Demaeyer. See LICENSE.txt for license information.

7.27.2 Function/Subroutine Documentation

7.27.2.1 subroutine, public util::cdiag (complex(kind=16), dimension(:,:), intent(in) *A*, complex(kind=16), dimension(:), intent(out) *d*)

Definition at line 230 of file util.f90.

```
230     COMPLEX(KIND=16), DIMENSION(:,:), INTENT(IN) :: a
231     COMPLEX(KIND=16), DIMENSION(:), INTENT(OUT) :: d
232     INTEGER :: i
233
234     DO i=1,SIZE(a,1)
235        d(i)=a(i,i)
236     END DO
```

7.27.2.2 subroutine, public util::choldc (real(kind=8), dimension(:,:) *a*, real(kind=8), dimension(:) *p*)

Definition at line 137 of file util.f90.

```
REAL(KIND=8), DIMENSION(:,:) :: a
REAL(KIND=8), DIMENSION(:) :: p
INTEGER :: n
INTEGER :: i,j,k
REAL(KIND=8) :: sum
n=size(a,1)
DO i=1,n
   DO j=i,n
      sum=a(i,j)
      DO k=i-1,1,-1
         sum=sum-a(i,k)*a(j,k)
      END DO
      IF (i.eq.j) THEN
         IF (sum.le.0.) stop 'choldc failed'
         p(i)=sqrt(sum)
      ELSE
         a(j,i)=sum/p(i)
      ENDIF
   END DO
END DO
RETURN
```

7.27.2.3 subroutine, public util::cprintmat (complex(kind=16), dimension(:,:), intent(in) *A*)

Definition at line 170 of file util.f90.

```
COMPLEX(KIND=16), DIMENSION(:,:), INTENT(IN) :: a
INTEGER :: i

DO i=1,SIZE(a,1)
   print*, a(i,:)
END DO
```

7.27.2.4 subroutine, public util::diag (real(kind=8), dimension(:,:), intent(in) *A*, real(kind=8), dimension(:), intent(out) *d*)

Definition at line 220 of file util.f90.

```
REAL(KIND=8), DIMENSION(:,:), INTENT(IN) :: a
REAL(KIND=8), DIMENSION(:), INTENT(OUT) :: d
INTEGER :: i

DO i=1,SIZE(a,1)
   d(i)=a(i,i)
END DO
```

7.27.2.5 integer function, public util::floordiv (integer *i*, integer *j*)

Definition at line 241 of file util.f90.

```
INTEGER :: i,j,floordiv
floordiv=int(floor(real(i)/real(j)))
RETURN
```

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

Initialize a square matrix A as a unit matrix.

Definition at line 100 of file util.f90.

```
100     REAL(KIND=8), DIMENSION(:,:),INTENT(INOUT) :: a
101     INTEGER :: i,n
102     n=size(a,1)
103     a=0.0d0
104     DO i=1,n
105       a(i,i)=1.0d0
106     END DO
107
```

7.27.2.7 subroutine, public util::init_random_seed ( )

Random generator initialization routine.

Definition at line 45 of file util.f90.

7.27.2.8 real(kind=8) function, dimension(size(a,1),size(a,2)), public util::invmat (real(kind=8), dimension(:,:), intent(in) *A*)

Definition at line 179 of file util.f90.

```
179     REAL(KIND=8), DIMENSION(:,:), INTENT(IN) :: a
180     REAL(KIND=8), DIMENSION(SIZE(A,1),SIZE(A,2)) :: ainv
181
182     REAL(KIND=8), DIMENSION(SIZE(A,1)) :: work  ! work array for LAPACK
183     INTEGER, DIMENSION(SIZE(A,1)) :: ipiv   ! pivot indices
184     INTEGER :: n, info
185
186     ! Store A in Ainv to prevent it from being overwritten by LAPACK
187     ainv = a
188     n = size(a,1)
189
190     ! DGETRF computes an LU factorization of a general M-by-N matrix A
191     ! using partial pivoting with row interchanges.
192     CALL dgetrf(n, n, ainv, n, ipiv, info)
193
194     IF (info /= 0) THEN
195        stop 'Matrix is numerically singular!'
196     ENDIF
197
198     ! DGETRI computes the inverse of a matrix using the LU factorization
199     ! computed by DGETRF.
200     CALL dgetri(n, ainv, n, ipiv, work, n, info)
201
202     IF (info /= 0) THEN
203        stop 'Matrix inversion failed!'
204     ENDIF
```

7.27.2.9 subroutine, public util::ireduce (real(kind=8), dimension(:,:), intent(out) *A,* real(kind=8), dimension(:,:), intent(in) *Ared,* integer, intent(in) *n,* integer, dimension(:), intent(in) *ind,* integer, dimension(:), intent(in) *rind*)

Definition at line 275 of file util.f90.

```
275     REAL(KIND=8), DIMENSION(:,:), INTENT(OUT) :: a
276     REAL(KIND=8), DIMENSION(:,:), INTENT(IN) :: ared
277     INTEGER, INTENT(IN) :: n
278     INTEGER, DIMENSION(:), INTENT(IN) :: ind,rind
279     INTEGER :: i,j
280     a=0.d0
281     DO i=1,n
282        DO j=1,n
283           a(ind(i),ind(j))=ared(i,j)
284        END DO
285     END DO
```

7.27.2.10 real(kind=8) function, public util::mat_contract (real(kind=8), dimension(:,:) *A*, real(kind=8), dimension(:,:) *B*)

Definition at line 123 of file util.f90.

```
123     REAL(KIND=8), DIMENSION(:,:) :: a,b
124     REAL(KIND=8) :: mat_contract
125     INTEGER :: i,j,n
126     n=size(a,1)
127     mat_contract=0.d0
128     DO i=1,n
129        DO j=1,n
130           mat_contract=mat_contract+a(i,j)*b(i,j)
131        END DO
132     ENDDO
133     RETURN
```

7.27.2.11 real(kind=8) function, public util::mat_trace (real(kind=8), dimension(:,:) *A*)

Definition at line 111 of file util.f90.

```
111     REAL(KIND=8), DIMENSION(:,:) :: a
112     REAL(KIND=8) :: mat_trace
113     INTEGER :: i,n
114     n=size(a,1)
115     mat_trace=0.d0
116     DO i=1,n
117        mat_trace=mat_trace+a(i,i)
118     END DO
119     RETURN
```

7.27.2.12 subroutine, public util::printmat (real(kind=8), dimension(:,:), intent(in) *A*)

Definition at line 161 of file util.f90.

```
161     REAL(KIND=8), DIMENSION(:,:), INTENT(IN) :: a
162     INTEGER :: i
163
164     DO i=1,SIZE(a,1)
165        print*, a(i,:)
166     END DO
```

7.27.2.13 subroutine, public util::reduce (real(kind=8), dimension(:,:), intent(in) *A*, real(kind=8), dimension(:,:), intent(out) *Ared*, integer, intent(out) *n*, integer, dimension(:), intent(out) *ind*, integer, dimension(:), intent(out) *rind*)

Definition at line 247 of file util.f90.

```
247     REAL(KIND=8), DIMENSION(:,:), INTENT(IN) :: a
248     REAL(KIND=8), DIMENSION(:,:), INTENT(OUT) :: ared
249     INTEGER, INTENT(OUT) :: n
250     INTEGER, DIMENSION(:), INTENT(OUT) :: ind,rind
251     LOGICAL, DIMENSION(SIZE(A,1)) :: sel
252     INTEGER :: i,j
253
254     ind=0
255     rind=0
256     sel=.false.
257     n=0
258     DO i=1,SIZE(a,1)
259        IF (any(a(i,:)/=0)) THEN
260           n=n+1
261           sel(i)=.true.
262           ind(n)=i
263           rind(i)=n
264        ENDIF
265     END DO
266     ared=0.d0
267     DO i=1,SIZE(a,1)
268        DO j=1,SIZE(a,1)
269           IF (sel(i).and.sel(j)) ared(rind(i),rind(j))=a(i,j)
270        ENDDO
271     ENDDO
```

7.27.2.14 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 37 of file util.f90.

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

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

Convert an integer to string.

Definition at line 30 of file util.f90.

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

7.27.2.16 subroutine, public util::triu (real(kind=8), dimension(:,:), intent(in) *A*, real(kind=8), dimension(:,:), intent(out) *T*)

Definition at line 208 of file util.f90.

```
208      REAL(KIND=8), DIMENSION(:,:), INTENT(IN) :: a
209      REAL(KIND=8), DIMENSION(:,:), INTENT(OUT) :: t
210      INTEGER i,j
211      t=0.d0
212      DO i=1,SIZE(a,1)
213         DO j=i,SIZE(a,1)
214            t(i,j)=a(i,j)
215         END DO
216      END DO
```

7.27.2.17 subroutine, public util::vector_outer (real(kind=8), dimension(:), intent(in) *u*, real(kind=8), dimension(:), intent(in) *v*, real(kind=8), dimension(:,:), intent(out) *A*)

Definition at line 289 of file util.f90.

```
289      REAL(KIND=8), DIMENSION(:), INTENT(IN) :: u,v
290      REAL(KIND=8), DIMENSION(:,:), INTENT(OUT) :: a
291      INTEGER :: i,j
292
293      a=0.d0
294      DO i=1,SIZE(u)
295         DO j=1,SIZE(v)
296            a(i,j)=u(i)*v(j)
297         ENDDO
298      ENDDO
```

7.28 wl_tensor Module Reference

The WL tensors used to integrate the model.

Functions/Subroutines

- subroutine, public init_wl_tensor

    Subroutine to initialise the WL tensor.

Variables

- real(kind=8), dimension(:), allocatable, public m11

    First component of the M1 term.
- type(coolist), dimension(:), allocatable, public m12

    Second component of the M1 term.
- real(kind=8), dimension(:), allocatable, public m13

    Third component of the M1 term.
- real(kind=8), dimension(:), allocatable, public m1tot

    Total M_1 vector.
- type(coolist), dimension(:), allocatable, public m21

    First tensor of the M2 term.
- type(coolist), dimension(:), allocatable, public m22

    Second tensor of the M2 term.
- type(coolist), dimension(:,:), allocatable, public l1

    First linear tensor.
- type(coolist), dimension(:,:), allocatable, public l2

    Second linear tensor.
- type(coolist), dimension(:,:), allocatable, public l4

    Fourth linear tensor.
- type(coolist), dimension(:,:), allocatable, public l5

    Fifth linear tensor.
- type(coolist), dimension(:,:), allocatable, public ltot

    Total linear tensor.
- type(coolist), dimension(:,:), allocatable, public b1

    First quadratic tensor.
- type(coolist), dimension(:,:), allocatable, public b2

    Second quadratic tensor.
- type(coolist), dimension(:,:), allocatable, public b3

    Third quadratic tensor.
- type(coolist), dimension(:,:), allocatable, public b4

    Fourth quadratic tensor.
- type(coolist), dimension(:,:), allocatable, public b14

    Joint 1st and 4th tensors.
- type(coolist), dimension(:,:), allocatable, public b23

    Joint 2nd and 3rd tensors.
- type(coolist4), dimension(:,:), allocatable, public mtot

    Tensor for the cubic terms.
- real(kind=8), dimension(:), allocatable dumb_vec

    Dummy vector.
- real(kind=8), dimension(:,:), allocatable dumb_mat1

    Dummy matrix.
- real(kind=8), dimension(:,:), allocatable dumb_mat2

    Dummy matrix.
- real(kind=8), dimension(:,:), allocatable dumb_mat3

Dummy matrix.

- real(kind=8), dimension(:,:), allocatable dumb_mat4

  Dummy matrix.

- logical, public m12def
- logical, public m21def
- logical, public m22def
- logical, public ldef
- logical, public b14def
- logical, public b23def
- logical, public mdef

  Boolean to (de)activate the computation of the terms.

7.28.1 Detailed Description

The WL tensors used to integrate the model.

Copyright

Remarks

7.28.2 Function/Subroutine Documentation

7.28.2.1 subroutine, public wl_tensor::init_wl_tensor ( )

Subroutine to initialise the WL tensor.

Definition at line 94 of file WL_tensor.f90.

```
94      INTEGER :: allocstat,i,j,k,m
95
96      print*, 'Initializing the decompostion tensors...'
97      CALL init_dec_tensor
98      print*, "Initializing the correlation matrices and tensors..."
99      CALL init_corr_tensor
100
101      !M1 part
102      print*, "Computing the M1 terms..."
103
104      ALLOCATE(m11(0:ndim), m12(ndim), m13(0:ndim), m1tot(0:ndim),
     stat=allocstat)
105      IF (allocstat /= 0) stop "*** Not enough memory ! ***"
106
107      ALLOCATE(dumb_mat1(ndim,ndim), dumb_mat2(ndim,ndim), stat=allocstat)
108      IF (allocstat /= 0) stop "*** Not enough memory ! ***"
109
110      ALLOCATE(dumb_mat3(ndim,ndim), dumb_mat4(ndim,ndim), stat=allocstat)
111      IF (allocstat /= 0) stop "*** Not enough memory ! ***"
112
113      ALLOCATE(dumb_vec(ndim), stat=allocstat)
114      IF (allocstat /= 0) stop "*** Not enough memory ! ***"
115
116      !M11
117      m11=0.d0
118      ! CALL coo_to_mat_ik(Lxy,dumb_mat1)
119      ! M11(1:ndim)=matmul(dumb_mat1,mean_full(1:ndim))
120
121      !M12
```

```
    ! dumb_mat2=0.D0
    ! DO i=1,ndim
    !    CALL coo_to_mat_i(i,Bxxy,dumb_mat1)
    !    dumb_mat2(i,:)=matmul(dumb_mat1,mean_full(1:ndim))
    ! ENDDO
    ! CALL matc_to_coo(dumb_mat2,M12)

    m12def=.not.tensor_empty(m12)

    !M13
    m13=0.d0
    CALL sparse_mul3_with_mat(bxyy,corr_i_full,m13)

    !M1tot
    m1tot=0.d0
    m1tot=m11+m13

    print*, "Computing the M2 terms..."
    ALLOCATE(m21(ndim), m22(ndim), stat=allocstat)
    IF (allocstat /= 0) stop "*** Not enough memory ! ***"

    !M21
    CALL copy_tensor(lxy,m21)
    CALL add_to_tensor(bxxy,m21)

    m21def=.not.tensor_empty(m21)

    !M22
    CALL copy_tensor(bxyy,m22)

    m22def=.not.tensor_empty(m22)

    !M3 tensor
    print*, "Computing the M3 terms..."
    ! Linear terms
    print*, "Computing the L subterms..."
    ALLOCATE(l1(ndim,mems), l2(ndim,mems), l4(ndim,mems), l5(ndim,mems),
  stat=allocstat)
    IF (allocstat /= 0) stop "*** Not enough memory ! ***"

    !L1
    CALL coo_to_mat_ik(lyx,dumb_mat1)
    CALL coo_to_mat_ik(lxy,dumb_mat2)
    DO m=1,mems
       CALL coo_to_mat_ik(dy(:,m),dumb_mat3)
       dumb_mat4=matmul(dumb_mat2,matmul(transpose(dumb_mat3),dumb_mat1))
       CALL matc_to_coo(dumb_mat4,l1(:,m))
    ENDDO

    !L2
    DO m=1,mems
       dumb_mat4=0.d0
       DO i=1,ndim
          CALL coo_to_mat_i(i,bxyy,dumb_mat1)
          CALL sparse_mul4_with_mat_kl(ydyy(:,m),dumb_mat1,dumb_mat2)
          DO j=1,ndim
             CALL coo_to_mat_j(j,byxy,dumb_mat1)
             dumb_mat4(i,j)=mat_trace(matmul(dumb_mat1,dumb_mat2))
          ENDDO
       END DO
       CALL matc_to_coo(dumb_mat4,l2(:,m))
    ENDDO

    !L4
    ! DO m=1,mems
    !    dumb_mat4=0.D0
    !    DO i=1,ndim
    !       CALL coo_to_mat_i(i,Bxyy,dumb_mat1)
    !       CALL sparse_mul3_with_mat(dYY(:,m),dumb_mat1,dumb_vec) ! Bxyy*dYY
    !       CALL coo_to_mat_ik(Lyx,dumb_mat1)
    !       dumb_mat4(i,:)=matmul(transpose(dumb_mat1),dumb_vec)
    !    ENDDO
    !    CALL matc_to_coo(dumb_mat4,L4(:,m))
    ! ENDDO

    !L5

    ! CALL coo_to_mat_ik(Lxy,dumb_mat1)
    ! DO m=1,mems
    !    dumb_mat4=0.D0
    !    DO i=1,ndim
    !       CALL sparse_mul3_mat(YdY(:,m),dumb_mat1(i,:),dumb_mat2)
    !       DO j=1,ndim
    !          CALL coo_to_mat_j(j,Byxy,dumb_mat3)
    !          dumb_mat4(i,j)=mat_trace(matmul(dumb_mat3,dumb_mat2))
    !       ENDDO
    !    END DO
```

```
   !    CALL matc_to_coo(dumb_mat4,L5(:,m))
   ! ENDDO

   !Ltot

   ALLOCATE(ltot(ndim,mems), stat=allocstat)
   IF (allocstat /= 0) stop "*** Not enough memory ! ***"

   DO m=1,mems
      CALL add_to_tensor(l1(:,m),ltot(:,m))
      CALL add_to_tensor(l2(:,m),ltot(:,m))
      CALL add_to_tensor(l4(:,m),ltot(:,m))
      CALL add_to_tensor(l5(:,m),ltot(:,m))
   ENDDO

   ldef=.not.tensor_empty(ltot)

   print*, "Computing the B terms..."
   ALLOCATE(b1(ndim,mems), b2(ndim,mems), b3(ndim,mems), b4(ndim,mems),
 stat=allocstat)
   IF (allocstat /= 0) stop "*** Not enough memory ! ***"

   ! B1
   CALL coo_to_mat_ik(lxy,dumb_mat1)
   dumb_mat1=transpose(dumb_mat1)
   DO m=1,mems
      CALL coo_to_mat_ik(dy(:,m),dumb_mat2)
      dumb_mat2=matmul(dumb_mat2,dumb_mat1)
      DO j=1,ndim
         DO k=1,ndim
            CALL coo_to_vec_jk(j,k,byxx,dumb_vec)
            dumb_vec=matmul(dumb_vec,dumb_mat2)
            CALL add_vec_jk_to_tensor(j,k,dumb_vec,b1(:,m))
         ENDDO
      ENDDO
   ENDDO

   ! B2
   CALL coo_to_mat_ik(lyx,dumb_mat3)
   dumb_mat3=transpose(dumb_mat3)
   DO m=1,mems
      DO i=1,ndim
         CALL coo_to_mat_i(i,bxxy,dumb_mat1)
         CALL coo_to_mat_ik(dy(:,m),dumb_mat2)
         dumb_mat1=matmul(dumb_mat2,transpose(dumb_mat1))
         dumb_mat1=matmul(dumb_mat3,dumb_mat1)
         CALL add_matc_to_tensor(i,dumb_mat1,b2(:,m))
      ENDDO
   ENDDO

   ! B3
   ! DO m=1,mems
   !    DO i=1,ndim
   !       CALL coo_to_mat_i(i,Bxxy,dumb_mat1)
   !       dumb_mat4=0.D0
   !       DO j=1,ndim
   !          CALL coo_to_mat_j(j,YdY(:,m),dumb_mat2)
   !          CALL coo_to_mat_i(j,Byxy,dumb_mat3)
   !          dumb_mat2=matmul(dumb_mat3,dumb_mat2)
   !          dumb_mat4=dumb_mat4+dumb_mat2
   !       ENDDO
   !       dumb_mat4=matmul(dumb_mat4,transpose(dumb_mat1))
   !       CALL add_matc_to_tensor(i,dumb_mat4,B3(:,m))
   !    ENDDO
   ! ENDDO

   ! B4
   ! DO m=1,mems
   !    DO i=1,ndim
   !       CALL coo_to_mat_i(i,Bxyy,dumb_mat1)
   !       CALL sparse_mul3_with_mat(dYY(:,m),dumb_mat1,dumb_vec) ! Bxyy*dYY
   !       DO j=1,ndim
   !          CALL coo_to_mat_j(j,Byxx,dumb_mat1)
   !          dumb_mat4(j,:)=matmul(transpose(dumb_mat1),dumb_vec)
   !       ENDDO
   !       CALL add_matc_to_tensor(i,dumb_mat4,B4(:,m))
   !    ENDDO
   ! ENDDO

   ALLOCATE(b14(ndim,mems), b23(ndim,mems), stat=allocstat)
   IF (allocstat /= 0) stop "*** Not enough memory ! ***"

   DO m=1,mems
      CALL add_to_tensor(b1(:,m),b14(:,m))
      CALL add_to_tensor(b2(:,m),b23(:,m))
      CALL add_to_tensor(b4(:,m),b14(:,m))
      CALL add_to_tensor(b3(:,m),b23(:,m))
```

```
    ENDDO

    b14def=.not.tensor_empty(b14)
    b23def=.not.tensor_empty(b23)

    !M

    print*, "Computing the M term..."

    ALLOCATE(mtot(ndim,mems), stat=allocstat)
    IF (allocstat /= 0) stop "*** Not enough memory ! ***"

    DO m=1,mems
       DO i=1,ndim
          CALL coo_to_mat_i(i,bxxy,dumb_mat1)
          CALL coo_to_mat_ik(dy(:,m),dumb_mat2)
          dumb_mat1=matmul(dumb_mat2,transpose(dumb_mat1))
          DO j=1,ndim
             DO k=1,ndim
                CALL coo_to_vec_jk(j,k,byxx,dumb_vec)
                dumb_vec=matmul(dumb_vec,dumb_mat1)
                CALL add_vec_ijk_to_tensor4(i,j,k,dumb_vec,mtot(:,m))
             ENDDO
          END DO
       END DO
    END DO

    mdef=.not.tensor4_empty(mtot)


    DEALLOCATE(dumb_mat1, dumb_mat2, stat=allocstat)
    IF (allocstat /= 0)  stop "*** Problem to deallocate ! ***"

    DEALLOCATE(dumb_mat3, dumb_mat4, stat=allocstat)
    IF (allocstat /= 0)  stop "*** Problem to deallocate ! ***"

    DEALLOCATE(dumb_vec, stat=allocstat)
    IF (allocstat /= 0)  stop "*** Problem to deallocate ! ***"
```

7.28.3 Variable Documentation

7.28.3.1 type(coolist), dimension(:,:), allocatable, public wl_tensor::b1

First quadratic tensor.

Definition at line 60 of file WL_tensor.f90.

```
TYPE(coolist), DIMENSION(:,:), ALLOCATABLE, PUBLIC :: b1    !< First quadratic tensor
```

7.28.3.2 type(coolist), dimension(:,:), allocatable, public wl_tensor::b14

Joint 1st and 4th tensors.

Definition at line 64 of file WL_tensor.f90.

```
TYPE(coolist), DIMENSION(:,:), ALLOCATABLE, PUBLIC :: b14   !< Joint 1st and 4th tensors
```

7.28.3.3 logical, public wl_tensor::b14def

Definition at line 75 of file WL_tensor.f90.

7.28.3.4 type(coolist), dimension(:,:), allocatable, public wl_tensor::b2

Second quadratic tensor.

Definition at line 61 of file WL_tensor.f90.

```
61   TYPE(coolist), DIMENSION(:,:), ALLOCATABLE, PUBLIC :: b2    !< Second quadratic tensor
```

7.28.3.5 type(coolist), dimension(:,:), allocatable, public wl_tensor::b23

Joint 2nd and 3rd tensors.

Definition at line 65 of file WL_tensor.f90.

```
65   TYPE(coolist), DIMENSION(:,:), ALLOCATABLE, PUBLIC :: b23   !< Joint 2nd and 3rd tensors
```

7.28.3.6 logical, public wl_tensor::b23def

Definition at line 75 of file WL_tensor.f90.

7.28.3.7 type(coolist), dimension(:,:), allocatable, public wl_tensor::b3

Third quadratic tensor.

Definition at line 62 of file WL_tensor.f90.

```
62   TYPE(coolist), DIMENSION(:,:), ALLOCATABLE, PUBLIC :: b3    !< Third quadratic tensor
```

7.28.3.8 type(coolist), dimension(:,:), allocatable, public wl_tensor::b4

Fourth quadratic tensor.

Definition at line 63 of file WL_tensor.f90.

```
63   TYPE(coolist), DIMENSION(:,:), ALLOCATABLE, PUBLIC :: b4    !< Fourth quadratic tensor
```

7.28.3.9 real(kind=8), dimension(:,:), allocatable wl_tensor::dumb_mat1 [private]

Dummy matrix.

Definition at line 70 of file WL_tensor.f90.

```
70   REAL(KIND=8), DIMENSION(:,:), ALLOCATABLE :: dumb_mat1 !< Dummy matrix
```

7.28.3.10 real(kind=8), dimension(:,:), allocatable wl_tensor::dumb_mat2 [private]

Dummy matrix.

Definition at line 71 of file WL_tensor.f90.

```
71 REAL(KIND=8), DIMENSION(:,:), ALLOCATABLE :: dumb_mat2 !< Dummy matrix
```

7.28.3.11 real(kind=8), dimension(:,:), allocatable wl_tensor::dumb_mat3 [private]

Dummy matrix.

Definition at line 72 of file WL_tensor.f90.

```
72 REAL(KIND=8), DIMENSION(:,:), ALLOCATABLE :: dumb_mat3 !< Dummy matrix
```

7.28.3.12 real(kind=8), dimension(:,:), allocatable wl_tensor::dumb_mat4 [private]

Dummy matrix.

Definition at line 73 of file WL_tensor.f90.

```
73 REAL(KIND=8), DIMENSION(:,:), ALLOCATABLE :: dumb_mat4 !< Dummy matrix
```

7.28.3.13 real(kind=8), dimension(:), allocatable wl_tensor::dumb_vec [private]

Dummy vector.

Definition at line 69 of file WL_tensor.f90.

```
69 REAL(KIND=8), DIMENSION(:), ALLOCATABLE :: dumb_vec !< Dummy vector
```

7.28.3.14 type(coolist), dimension(:,:), allocatable, public wl_tensor::l1

First linear tensor.

Definition at line 53 of file WL_tensor.f90.

```
53 TYPE(coolist), DIMENSION(:,:), ALLOCATABLE, PUBLIC :: l1    !< First linear tensor
```

7.28.3.15 type(coolist), dimension(:,:), allocatable, public wl_tensor::l2

Second linear tensor.

Definition at line 54 of file WL_tensor.f90.

```
54   TYPE(coolist), DIMENSION(:,:), ALLOCATABLE, PUBLIC :: l2    !< Second linear tensor
```

7.28.3.16 type(coolist), dimension(:,:), allocatable, public wl_tensor::l4

Fourth linear tensor.

Definition at line 55 of file WL_tensor.f90.

```
55   TYPE(coolist), DIMENSION(:,:), ALLOCATABLE, PUBLIC :: l4    !< Fourth linear tensor
```

7.28.3.17 type(coolist), dimension(:,:), allocatable, public wl_tensor::l5

Fifth linear tensor.

Definition at line 56 of file WL_tensor.f90.

```
56   TYPE(coolist), DIMENSION(:,:), ALLOCATABLE, PUBLIC :: l5    !< Fifth linear tensor
```

7.28.3.18 logical, public wl_tensor::ldef

Definition at line 75 of file WL_tensor.f90.

7.28.3.19 type(coolist), dimension(:,:), allocatable, public wl_tensor::ltot

Total linear tensor.

Definition at line 57 of file WL_tensor.f90.

```
57   TYPE(coolist), DIMENSION(:,:), ALLOCATABLE, PUBLIC :: ltot  !< Total linear tensor
```

7.28.3.20 real(kind=8), dimension(:), allocatable, public wl_tensor::m11

First component of the M1 term.

Definition at line 42 of file WL_tensor.f90.

```
42   REAL(KIND=8), DIMENSION(:), ALLOCATABLE, PUBLIC :: m11    !< First component of the M1 term
```

7.28.3.21 type(coolist), dimension(:), allocatable, public wl_tensor::m12

Second component of the M1 term.

Definition at line 43 of file WL_tensor.f90.

```
43    TYPE(coolist), DIMENSION(:), ALLOCATABLE, PUBLIC :: m12   !< Second component of the M1 term
```

7.28.3.22 logical, public wl_tensor::m12def

Definition at line 75 of file WL_tensor.f90.

```
75    LOGICAL, PUBLIC :: m12def,m21def,m22def,ldef,b14def,b23def,mdef !< Boolean to (de)activate the
        computation of the terms
```

7.28.3.23 real(kind=8), dimension(:), allocatable, public wl_tensor::m13

Third component of the M1 term.

Definition at line 44 of file WL_tensor.f90.

```
44    REAL(KIND=8), DIMENSION(:), ALLOCATABLE, PUBLIC :: m13    !< Third component of the M1 term
```

7.28.3.24 real(kind=8), dimension(:), allocatable, public wl_tensor::m1tot

Total M_1 vector.

Definition at line 45 of file WL_tensor.f90.

```
45    REAL(KIND=8), DIMENSION(:), ALLOCATABLE, PUBLIC :: m1tot  !< Total \f$M_1\f$ vector
```

7.28.3.25 type(coolist), dimension(:), allocatable, public wl_tensor::m21

First tensor of the M2 term.

Definition at line 48 of file WL_tensor.f90.

```
48    TYPE(coolist), DIMENSION(:), ALLOCATABLE, PUBLIC :: m21   !< First tensor of the M2 term
```

7.28.3.26 logical, public wl_tensor::m21def

Definition at line 75 of file WL_tensor.f90.

7.28.3.27 type(coolist), dimension(:), allocatable, public wl_tensor::m22

Second tensor of the M2 term.

Definition at line 49 of file WL_tensor.f90.

```
49    TYPE(coolist), DIMENSION(:), ALLOCATABLE, PUBLIC :: m22   !< Second tensor of the M2 term
```

7.28.3.28 logical, public wl_tensor::m22def

Definition at line 75 of file WL_tensor.f90.

7.28.3.29 logical, public wl_tensor::mdef

Boolean to (de)activate the computation of the terms.

Definition at line 75 of file WL_tensor.f90.

7.28.3.30 type(coolist4), dimension(:,:), allocatable, public wl_tensor::mtot

Tensor for the cubic terms.

Definition at line 67 of file WL_tensor.f90.

```
67    TYPE(coolist4), DIMENSION(:,:), ALLOCATABLE, PUBLIC :: mtot !< Tensor for the cubic terms
```

Chapter 8

Data Type Documentation

8.1 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

8.1.1 Detailed Description

Type holding the atmospheric inner products tensors.

Definition at line 52 of file inprod_analytic.f90.

8.1.2 Member Data Documentation

8.1.2.1 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
```

8.1.2.2 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
```

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

Definition at line 53 of file inprod_analytic.f90.

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

Definition at line 53 of file inprod_analytic.f90.

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

Definition at line 54 of file inprod_analytic.f90.

8.1.2.6 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

8.2 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.

8.2.1 Detailed Description

Atmospheric bloc specification type.

Definition at line 39 of file inprod_analytic.f90.

8.2.2 Member Data Documentation

8.2.2.1 integer inprod_analytic::atm_wavenum::h =0 [private]

Definition at line 41 of file inprod_analytic.f90.

8.2.2.2 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
```

8.2.2.3 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.
```

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

Definition at line 42 of file inprod_analytic.f90.

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

Definition at line 41 of file inprod_analytic.f90.

8.2.2.6 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

8.3 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.

8.3.1 Detailed Description

Coordinate list. Type used to represent the sparse tensor.

Definition at line 38 of file tensor.f90.

8.3.2 Member Data Documentation

8.3.2.1 type(coolist_elem), dimension(:), allocatable tensor::coolist::elems

Lists of elements tensor::coolist_elem.

Definition at line 39 of file tensor.f90.

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

8.3.2.2 integer tensor::coolist::nelems = 0

Number of elements in the list.

Definition at line 40 of file tensor.f90.

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

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

- tensor.f90

8.4 tensor::coolist4 Type Reference

4d coordinate list. Type used to represent the rank-4 sparse tensor.

Public Attributes

- type(coolist_elem4), dimension(:), allocatable elems
- integer nelems = 0

8.4.1 Detailed Description

4d coordinate list. Type used to represent the rank-4 sparse tensor.

Definition at line 44 of file tensor.f90.

8.4.2 Member Data Documentation

8.4.2.1 type(coolist_elem4), dimension(:), allocatable tensor::coolist4::elems

Definition at line 45 of file tensor.f90.

```
45      TYPE(coolist_elem4), DIMENSION(:), ALLOCATABLE :: elems
```

8.4.2.2 integer tensor::coolist4::nelems = 0

Definition at line 46 of file tensor.f90.

```
46      INTEGER :: nelems = 0
```

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

- tensor.f90

8.5 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.

8.5.1 Detailed Description

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

Definition at line 25 of file tensor.f90.

8.5.2 Member Data Documentation

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

Index j of the element.

Definition at line 26 of file tensor.f90.

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

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

Index k of the element.

Definition at line 27 of file tensor.f90.

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

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

Value of the element.

Definition at line 28 of file tensor.f90.

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

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

- tensor.f90

8.6 tensor::coolist_elem4 Type Reference

4d coordinate list element type. Elementary elements of the 4d sparse tensors.

Private Attributes

- integer j
- integer k
- integer l
- real(kind=8) v

8.6.1 Detailed Description

4d coordinate list element type. Elementary elements of the 4d sparse tensors.

Definition at line 32 of file tensor.f90.

8.6.2 Member Data Documentation

8.6.2.1 integer tensor::coolist_elem4::j [private]

Definition at line 33 of file tensor.f90.

```
33      INTEGER :: j,k,l
```

8.6.2.2 integer tensor::coolist_elem4::k [private]

Definition at line 33 of file tensor.f90.

8.6.2.3 integer tensor::coolist_elem4::l [private]

Definition at line 33 of file tensor.f90.

8.6.2.4 real(kind=8) tensor::coolist_elem4::v [private]

Definition at line 34 of file tensor.f90.

```
34      REAL(KIND=8) :: v
```

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

- tensor.f90

8.7 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

8.7.1 Detailed Description

Type holding the oceanic inner products tensors.

Definition at line 58 of file inprod_analytic.f90.

8.7.2 Member Data Documentation

8.7.2.1 real(kind=8), dimension(:,:,:), allocatable inprod_analytic::ocean_tensors::c [private]

Definition at line 60 of file inprod_analytic.f90.

8.7.2.2 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
```

8.7.2.3 real(kind=8), dimension(:,:), allocatable inprod_analytic::ocean_tensors::m [private]

Definition at line 59 of file inprod_analytic.f90.

8.7.2.4 real(kind=8), dimension(:,:), allocatable inprod_analytic::ocean_tensors::n [private]

Definition at line 59 of file inprod_analytic.f90.

8.7.2.5 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
```

8.7.2.6 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

8.8 inprod_analytic::ocean_wavenum Type Reference

Oceanic bloc specification type.

Private Attributes

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

8.8.1 Detailed Description

Oceanic bloc specification type.

Definition at line 46 of file inprod_analytic.f90.

8.8.2 Member Data Documentation

8.8.2.1 integer inprod_analytic::ocean_wavenum::h [private]

Definition at line 47 of file inprod_analytic.f90.

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

Definition at line 48 of file inprod_analytic.f90.

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

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

Definition at line 48 of file inprod_analytic.f90.

8.8.2.4 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 9

File Documentation

9.1 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.*

9.2 corr_tensor.f90 File Reference

Modules

- module corr_tensor

  Module to compute the correlations and derivatives used to compute the memory term of the WL parameterization.

Functions/Subroutines

- subroutine, public corr_tensor::init_corr_tensor

  Subroutine to initialise the correlations tensors.

Variables

- type(coolist), dimension(:,:), allocatable, public corr_tensor::yy

  Coolist holding the $\langle Y \otimes Y^s \rangle$ *terms.*
- type(coolist), dimension(:,:), allocatable, public corr_tensor::dy

  Coolist holding the $\langle \partial_Y \otimes Y^s \rangle$ *terms.*
- type(coolist), dimension(:,:), allocatable, public corr_tensor::ydy

  Coolist holding the $\langle Y \otimes \partial_Y \otimes Y^s \rangle$ *terms.*
- type(coolist), dimension(:,:), allocatable, public corr_tensor::dyy

  Coolist holding the $\langle \partial_Y \otimes Y^s \otimes Y^s \rangle$ *terms.*
- type(coolist4), dimension(:,:), allocatable, public corr_tensor::ydyy

  Coolist holding the $\langle Y \otimes \partial_Y \otimes Y^s \otimes Y^s \rangle$ *terms.*
- real(kind=8), dimension(:), allocatable corr_tensor::dumb_vec

  Dumb vector to be used in the calculation.
- real(kind=8), dimension(:,:), allocatable corr_tensor::dumb_mat1

  Dumb matrix to be used in the calculation.
- real(kind=8), dimension(:,:), allocatable corr_tensor::dumb_mat2

  Dumb matrix to be used in the calculation.
- real(kind=8), dimension(:,:), allocatable corr_tensor::expm

  *Matrix holding the product inv_corr_i*corr_ij at time* s*.*

9.3 corrmod.f90 File Reference

Modules

- module corrmod

  Module to initialize the correlation matrix of the unresolved variables.

Functions/Subroutines

- subroutine, public corrmod::init_corr

  Subroutine to initialise the computation of the correlation.
- subroutine corrmod::corrcomp_from_def (s)

  Subroutine to compute the correlation of the unresolved variables $\langle Y \otimes Y^s \rangle$ *at time* s *from the definition given inside the module.*
- subroutine corrmod::corrcomp_from_spline (s)

  Subroutine to compute the correlation of the unresolved variables $\langle Y \otimes Y^s \rangle$ *at time* s *from the spline representation.*
- subroutine corrmod::splint (xa, ya, y2a, n, x, y)

  Routine to compute the spline representation parameters.
- real(kind=8) function corrmod::fs (s, p)

  Exponential fit function.
- subroutine corrmod::corrcomp_from_fit (s)

  Subroutine to compute the correlation of the unresolved variables $\langle Y \otimes Y^s \rangle$ *at time* s *from the exponential representation.*

Variables

- real(kind=8), dimension(:), allocatable, public corrmod::mean

  Vector holding the mean of the unresolved dynamics (reduced version)
- real(kind=8), dimension(:), allocatable, public corrmod::mean_full

  Vector holding the mean of the unresolved dynamics (full version)
- real(kind=8), dimension(:,:), allocatable, public corrmod::corr_i_full

  Covariance matrix of the unresolved variables (full version)
- real(kind=8), dimension(:,:), allocatable, public corrmod::inv_corr_i_full

  Inverse of the covariance matrix of the unresolved variables (full version)
- real(kind=8), dimension(:,:), allocatable, public corrmod::corr_i

  Covariance matrix of the unresolved variables (reduced version)
- real(kind=8), dimension(:,:), allocatable, public corrmod::inv_corr_i

  Inverse of the covariance matrix of the unresolved variables (reduced version)
- real(kind=8), dimension(:,:), allocatable, public corrmod::corr_ij

  Matrix holding the correlation matrix at a given time.
- real(kind=8), dimension(:,:,:), allocatable corrmod::y2

  Vector holding coefficient of the spline and exponential correlation representation.
- real(kind=8), dimension(:,:,:), allocatable corrmod::ya

  Vector holding coefficient of the spline and exponential correlation representation.
- real(kind=8), dimension(:), allocatable corrmod::xa

  Vector holding coefficient of the spline and exponential correlation representation.
- integer corrmod::nspl

  Integers needed by the spline representation of the correlation.
- integer corrmod::klo
- integer corrmod::khi
- procedure(corrcomp_from_spline), pointer, public corrmod::corrcomp

  Pointer to the correlation computation routine.

9.4 dec_tensor.f90 File Reference

Modules

- module dec_tensor

  The resolved-unresolved components decomposition of the tensor.

Functions/Subroutines

- subroutine dec_tensor::suppress_and (t, cst, v1, v2)

  Subroutine to suppress from the tensor t_{ijk} components satisfying SF(j)=v1 and SF(k)=v2.
- subroutine dec_tensor::suppress_or (t, cst, v1, v2)

  Subroutine to suppress from the tensor t_{ijk} components satisfying SF(j)=v1 or SF(k)=v2.
- subroutine dec_tensor::reorder (t, cst, v)

  Subroutine to reorder the tensor t_{ijk} components : if SF(j)=v then it return t_{ikj}.
- subroutine dec_tensor::init_sub_tensor (t, cst, v)

  Subroutine that suppress all the components of a tensor t_{ijk} where if SF(i)=v.
- subroutine, public dec_tensor::init_dec_tensor

  Subroutine that initialize and compute the decomposed tensors.

Variables

- type(coolist), dimension(:), allocatable, public dec_tensor::ff_tensor

  Tensor holding the part of the unresolved tensor involving only unresolved variables.
- type(coolist), dimension(:), allocatable, public dec_tensor::sf_tensor

  Tensor holding the part of the resolved tensor involving unresolved variables.
- type(coolist), dimension(:), allocatable, public dec_tensor::ss_tensor

  Tensor holding the part of the resolved tensor involving only resolved variables.
- type(coolist), dimension(:), allocatable, public dec_tensor::fs_tensor

  Tensor holding the part of the unresolved tensor involving resolved variables.
- type(coolist), dimension(:), allocatable, public dec_tensor::hx

  Tensor holding the constant part of the resolved tendencies.
- type(coolist), dimension(:), allocatable, public dec_tensor::lxx

  Tensor holding the linear part of the resolved tendencies involving the resolved variables.
- type(coolist), dimension(:), allocatable, public dec_tensor::lxy

  Tensor holding the linear part of the resolved tendencies involving the unresolved variables.
- type(coolist), dimension(:), allocatable, public dec_tensor::bxxx

  Tensor holding the quadratic part of the resolved tendencies involving resolved variables.
- type(coolist), dimension(:), allocatable, public dec_tensor::bxxy

  Tensor holding the quadratic part of the resolved tendencies involving both resolved and unresolved variables.
- type(coolist), dimension(:), allocatable, public dec_tensor::bxyy

  Tensor holding the quadratic part of the resolved tendencies involving unresolved variables.
- type(coolist), dimension(:), allocatable, public dec_tensor::hy

  Tensor holding the constant part of the unresolved tendencies.
- type(coolist), dimension(:), allocatable, public dec_tensor::lyx

  Tensor holding the linear part of the unresolved tendencies involving the resolved variables.
- type(coolist), dimension(:), allocatable, public dec_tensor::lyy

  Tensor holding the linear part of the unresolved tendencies involving the unresolved variables.
- type(coolist), dimension(:), allocatable, public dec_tensor::byxx

  Tensor holding the quadratic part of the unresolved tendencies involving resolved variables.
- type(coolist), dimension(:), allocatable, public dec_tensor::byxy

  Tensor holding the quadratic part of the unresolved tendencies involving both resolved and unresolved variables.
- type(coolist), dimension(:), allocatable, public dec_tensor::byyy

  Tensor holding the quadratic part of the unresolved tendencies involving unresolved variables.
- type(coolist), dimension(:), allocatable, public dec_tensor::ss_tl_tensor

  Tensor of the tangent linear model tendencies of the resolved component alone.
- type(coolist), dimension(:), allocatable dec_tensor::dumb

  Dumb coolist to make the computations.

9.5 doc/gen_doc.md File Reference

9.6 doc/sto_doc.md File Reference

9.7 doc/tl_ad_doc.md File Reference

9.8 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.

9.9 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.

9.10 int_comp.f90 File Reference

Modules

- module int_comp

  Utility module containing the routines to perform the integration of functions.

Functions/Subroutines

- subroutine, public int_comp::integrate (func, ss)

  Routine to compute integrals of function from O to #maxint.
- subroutine int_comp::qromb (func, a, b, ss)

  Romberg integration routine.
- subroutine int_comp::qromo (func, a, b, ss, choose)

  Romberg integration routine on an open interval.
- subroutine int_comp::polint (xa, ya, n, x, y, dy)

  Polynomial interpolation routine.
- subroutine int_comp::trapzd (func, a, b, s, n)

  Trapezoidal rule integration routine.
- subroutine int_comp::midpnt (func, a, b, s, n)

  Midpoint rule integration routine.
- subroutine int_comp::midexp (funk, aa, bb, s, n)

  Midpoint routine for bb infinite with funk decreasing infinitely rapidly at infinity.

9.11 int_corr.f90 File Reference

Modules

- module int_corr

  Module to compute or load the integrals of the correlation matrices.

Functions/Subroutines

- subroutine, public int_corr::init_corrint
  Subroutine to initialise the integrated matrices and tensors.
- real(kind=8) function int_corr::func_ij (s)
  Function that returns the component oi and oj of the correlation matrix at time s.
- real(kind=8) function int_corr::func_ijkl (s)
  Function that returns the component oi,oj,ok and ol of the outer product of the correlation matrix with itself at time s.
- subroutine, public int_corr::comp_corrint
  Routine that actually compute or load the integrals.

Variables

- integer int_corr::oi
- integer int_corr::oj
- integer int_corr::ok
- integer int_corr::ol
  Integers that specify the matrices and tensor component considered as a function of time.
- real(kind=8), parameter int_corr::real_eps = 2.2204460492503131e-16
  Small epsilon constant to determine equality with zero.
- real(kind=8), dimension(:,:), allocatable, public int_corr::corrint
  Matrix holding the integral of the correlation matrix.
- type(coolist4), dimension(:), allocatable, public int_corr::corr2int
  Tensor holding the integral of the correlation outer product with itself.

9.12 LICENSE.txt File Reference

Functions

- The MIT License (MIT) Copyright(c) 2015-2017 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

9.12.1 Function Documentation

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

9.12.1.2 The MIT License (MIT)

9.12.2 Variable Documentation

9.12.2.1 The MIT free of charge

Definition at line 6 of file LICENSE.txt.

9.12.2.2 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.

9.12.2.3 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.

9.12.2.4 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.

9.12.2.5 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.

9.12.2.6 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.

9.12.2.7 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.

9.12.2.8 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.

9.12.2.9 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.

9.12.2.10 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.

9.12.2.11 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.

9.12.2.12 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.

9.12.2.13 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.

9.12.2.14 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.

9.12.2.15 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.

9.12.2.16 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.

9.12.2.17 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.

9.12.2.18 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.

9.12.2.19 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.

9.12.2.20 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.

9.13 maooam.f90 File Reference

Functions/Subroutines

- program maooam

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

9.13.1 Function/Subroutine Documentation

9.13.1.1 program maooam ( )

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 maooam.f90.

9.14 maooam_MTV.f90 File Reference

Functions/Subroutines

- program maooam_mtv

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

9.14.1 Function/Subroutine Documentation

9.14.1.1 program maooam_mtv ( )

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

Copyright

2017 Jonathan Demaeyer. See LICENSE.txt for license information.

Definition at line 12 of file maooam_MTV.f90.

9.15 maooam_stoch.f90 File Reference

Functions/Subroutines

- program maooam_stoch

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

9.15.1 Function/Subroutine Documentation

9.15.1.1 program maooam_stoch ( )

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

Copyright

2017 Jonathan Demaeyer. See LICENSE.txt for license information.

Remarks

There are four dynamics modes:

- full: generate the full dynamics
- unres: generate the intrinsic unresolved dynamics
- qfst: generate dynamics given by the quadratic terms of the unresolved tendencies
- reso: use the resolved dynamics alone

Definition at line 24 of file maooam_stoch.f90.

9.16 maooam_WL.f90 File Reference

Functions/Subroutines

- program maooam_wl

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

9.16.1 Function/Subroutine Documentation

9.16.1.1 program maooam_wl ( )

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

Copyright

2017 Jonathan Demaeyer. See LICENSE.txt for license information.

Definition at line 12 of file maooam_WL.f90.

9.17 MAR.f90 File Reference

Modules

- module mar

  Multidimensional Autoregressive module to generate the correlation for the WL parameterization.

Functions/Subroutines

- subroutine, public mar::init_mar
  Subroutine to initialise the MAR.
- subroutine, public mar::mar_step (x)
  Routine to generate one step of the MAR.
- subroutine, public mar::mar_step_red (xred)
  Routine to generate one step of the reduce MAR.
- subroutine mar::stoch_vec (dW)

Variables

- real(kind=8), dimension(:,:), allocatable, public mar::q
  Square root of the noise covariance matrix.
- real(kind=8), dimension(:,:), allocatable, public mar::qred
  Reduce version of Q.
- real(kind=8), dimension(:,:), allocatable, public mar::rred
  Covariance matrix of the noise.
- real(kind=8), dimension(:,:,:), allocatable, public mar::w
  W_i matrix.
- real(kind=8), dimension(:,:,:), allocatable, public mar::wred
  Reduce W_i matrix.
- real(kind=8), dimension(:), allocatable mar::buf_y
- real(kind=8), dimension(:), allocatable mar::dw
- integer, public mar::ms
  order of the MAR

9.18 memory.f90 File Reference

Modules

- module memory
  Module that compute the memory term M_3 of the WL parameterization.

Functions/Subroutines

- subroutine, public memory::init_memory
  Subroutine to initialise the memory.
- subroutine, public memory::compute_m3 (y, dt, dtn, savey, save_ev, evolve, inter, h_int)
  Compute the integrand of M_3 at each time in the past and integrate to get the memory term.
- subroutine, public memory::test_m3 (y, dt, dtn, h_int)
  Routine to test the #compute_M3 routine.

Variables

- real(kind=8), dimension(:,:), allocatable memory::x

  Array storing the previous state of the system.
- real(kind=8), dimension(:,:), allocatable memory::xs

  Array storing the resolved time evolution of the previous state of the system.
- real(kind=8), dimension(:,:), allocatable memory::zs

  Dummy array to replace Xs in case where the evolution is not stored.
- real(kind=8), dimension(:), allocatable memory::buf_m

  Dummy vector.
- real(kind=8), dimension(:), allocatable memory::buf_m3

  Dummy vector to store the M_3 integrand.
- integer memory::t_index

  Integer storing the time index (current position in the arrays)
- procedure(ss_step), pointer memory::step

  Procedural pointer pointing on the resolved dynamics step routine.

9.19 MTV_int_tensor.f90 File Reference

Modules

- module mtv_int_tensor

  The MTV tensors used to integrate the MTV model.

Functions/Subroutines

- subroutine, public mtv_int_tensor::init_mtv_int_tensor

  Subroutine to initialise the MTV tensor.

Variables

- real(kind=8), dimension(:), allocatable, public mtv_int_tensor::h1

  First constant vector.
- real(kind=8), dimension(:), allocatable, public mtv_int_tensor::h2

  Second constant vector.
- real(kind=8), dimension(:), allocatable, public mtv_int_tensor::h3

  Third constant vector.
- real(kind=8), dimension(:), allocatable, public mtv_int_tensor::htot

  Total constant vector.
- type(coolist), dimension(:), allocatable, public mtv_int_tensor::l1

  First linear tensor.
- type(coolist), dimension(:), allocatable, public mtv_int_tensor::l2

  Second linear tensor.
- type(coolist), dimension(:), allocatable, public mtv_int_tensor::l3

  Third linear tensor.
- type(coolist), dimension(:), allocatable, public mtv_int_tensor::ltot

  Total linear tensor.
- type(coolist), dimension(:), allocatable, public mtv_int_tensor::b1

First quadratic tensor.

- type(coolist), dimension(:), allocatable, public mtv_int_tensor::b2

  Second quadratic tensor.

- type(coolist), dimension(:), allocatable, public mtv_int_tensor::btot

  Total quadratic tensor.

- type(coolist4), dimension(:), allocatable, public mtv_int_tensor::mtot

  Tensor for the cubic terms.

- real(kind=8), dimension(:,:), allocatable, public mtv_int_tensor::q1

  Constant terms for the state-dependent noise covariance matrix.

- real(kind=8), dimension(:,:), allocatable, public mtv_int_tensor::q2

  Constant terms for the state-independent noise covariance matrix.

- type(coolist), dimension(:), allocatable, public mtv_int_tensor::utot

  Linear terms for the state-dependent noise covariance matrix.

- type(coolist4), dimension(:), allocatable, public mtv_int_tensor::vtot

  Quadratic terms for the state-dependent noise covariance matrix.

- real(kind=8), dimension(:), allocatable mtv_int_tensor::dumb_vec

  Dummy vector.

- real(kind=8), dimension(:,:), allocatable mtv_int_tensor::dumb_mat1

  Dummy matrix.

- real(kind=8), dimension(:,:), allocatable mtv_int_tensor::dumb_mat2

  Dummy matrix.

- real(kind=8), dimension(:,:), allocatable mtv_int_tensor::dumb_mat3

  Dummy matrix.

- real(kind=8), dimension(:,:), allocatable mtv_int_tensor::dumb_mat4

  Dummy matrix.

9.20 MTV_sigma_tensor.f90 File Reference

Modules

- module sigma

  The MTV noise sigma matrices used to integrate the MTV model.

Functions/Subroutines

- subroutine, public sigma::init_sigma (mult, Q1fill)

  Subroutine to initialize the sigma matices.

- subroutine, public sigma::compute_mult_sigma (y)

  Routine to actualize the matrix σ_1 based on the state y of the MTV system.

Variables

- real(kind=8), dimension(:,:), allocatable, public sigma::sig1

  $\sigma_1(X)$ *state-dependent noise matrix*
- real(kind=8), dimension(:,:), allocatable, public sigma::sig2

  σ_2 *state-independent noise matrix*
- real(kind=8), dimension(:,:), allocatable, public sigma::sig1r

  Reduced $\sigma_1(X)$ *state-dependent noise matrix.*
- real(kind=8), dimension(:,:), allocatable sigma::dumb_mat1

  Dummy matrix.
- real(kind=8), dimension(:,:), allocatable sigma::dumb_mat2

  Dummy matrix.
- real(kind=8), dimension(:,:), allocatable sigma::dumb_mat3

  Dummy matrix.
- real(kind=8), dimension(:,:), allocatable sigma::dumb_mat4

  Dummy matrix.
- integer, dimension(:), allocatable sigma::ind1
- integer, dimension(:), allocatable sigma::rind1
- integer, dimension(:), allocatable sigma::ind2
- integer, dimension(:), allocatable sigma::rind2

  Reduction indices.
- integer sigma::n1
- integer sigma::n2

9.21 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

  Merchanical 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

  β' - *Non-dimensional beta parameter*
- 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.

9.22 rk2_integrator.f90 File Reference

Modules

- module integrator

  Module with the integration routines.

Functions/Subroutines

- subroutine, public integrator::init_integrator

  Routine to initialise the integration buffers.
- subroutine integrator::tendencies (t, y, res)

  Routine computing the tendencies of the model.
- subroutine, public 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 integrator::buf_y1

  Buffer to hold the intermediate position (Heun algorithm)
- real(kind=8), dimension(:), allocatable integrator::buf_f0

  Buffer to hold tendencies at the initial position.
- real(kind=8), dimension(:), allocatable integrator::buf_f1

  Buffer to hold tendencies at the intermediate position.

9.23 rk2_MTV_integrator.f90 File Reference

Modules

- module rk2_mtv_integrator

  Module with the MTV rk2 integration routines.

Functions/Subroutines

- subroutine, public rk2_mtv_integrator::init_integrator

  Subroutine to initialize the MTV rk2 integrator.
- subroutine rk2_mtv_integrator::init_noise

  Routine to initialize the noise vectors and buffers.
- subroutine rk2_mtv_integrator::init_g

  Routine to initialize the G term.
- subroutine rk2_mtv_integrator::compg (y)

  Routine to actualize the G term based on the state y of the MTV system.
- subroutine, public rk2_mtv_integrator::step (y, t, dt, dtn, res, tend)

  Routine to perform an integration step (Heun algorithm) of the MTV system. The incremented time is returned.
- subroutine, public rk2_mtv_integrator::full_step (y, t, dt, dtn, res)

  Routine to perform an integration step (Heun algorithm) of the full stochastic system. The incremented time is returned.

Variables

- real(kind=8), dimension(:), allocatable rk2_mtv_integrator::buf_y1
- real(kind=8), dimension(:), allocatable rk2_mtv_integrator::buf_f0
- real(kind=8), dimension(:), allocatable rk2_mtv_integrator::buf_f1

  Integration buffers.
- real(kind=8), dimension(:), allocatable rk2_mtv_integrator::dw
- real(kind=8), dimension(:), allocatable rk2_mtv_integrator::dwmult

  Standard gaussian noise buffers.
- real(kind=8), dimension(:), allocatable rk2_mtv_integrator::dwar
- real(kind=8), dimension(:), allocatable rk2_mtv_integrator::dwau
- real(kind=8), dimension(:), allocatable rk2_mtv_integrator::dwor
- real(kind=8), dimension(:), allocatable rk2_mtv_integrator::dwou

  Standard gaussian noise buffers.
- real(kind=8), dimension(:), allocatable rk2_mtv_integrator::anoise
- real(kind=8), dimension(:), allocatable rk2_mtv_integrator::noise

  Additive noise term.
- real(kind=8), dimension(:), allocatable rk2_mtv_integrator::noisemult

  Multiplicative noise term.
- real(kind=8), dimension(:), allocatable rk2_mtv_integrator::g

  G term of the MTV tendencies.
- real(kind=8), dimension(:), allocatable rk2_mtv_integrator::buf_g

  Buffer for the G term computation.
- logical rk2_mtv_integrator::mult

  Logical indicating if the sigma1 matrix must be computed for every state change.
- logical rk2_mtv_integrator::q1fill

  Logical indicating if the matrix Q1 is non-zero.
- logical rk2_mtv_integrator::compute_mult

  Logical indicating if the Gaussian noise for the multiplicative noise must be computed.
- real(kind=8), parameter rk2_mtv_integrator::sq2 = sqrt(2.D0)

  Hard coded square root of 2.

9.24 rk2_ss_integrator.f90 File Reference

Modules

- module rk2_ss_integrator

  Module with the stochastic uncoupled resolved nonlinear and tangent linear rk2 dynamics integration routines.

Functions/Subroutines

- subroutine, public rk2_ss_integrator::init_ss_integrator

  Subroutine to initialize the uncoupled resolved rk2 integrator.
- subroutine, public rk2_ss_integrator::tendencies (t, y, res)

  Routine computing the tendencies of the uncoupled resolved model.
- subroutine, public rk2_ss_integrator::tl_tendencies (t, y, ys, res)

  Tendencies for the tangent linear model of the uncoupled resolved dynamics in point ystar for perturbation deltay.
- subroutine, public rk2_ss_integrator::ss_step (y, ys, t, dt, dtn, res)

  Routine to perform a stochastic integration step of the unresolved uncoupled dynamics (Heun algorithm). The incremented time is returned.
- subroutine, public rk2_ss_integrator::ss_tl_step (y, ys, t, dt, dtn, res)

  Routine to perform a stochastic integration step of the unresolved uncoupled tangent linear dynamics (Heun algorithm). The incremented time is returned.

Variables

- real(kind=8), dimension(:), allocatable rk2_ss_integrator::dwar
- real(kind=8), dimension(:), allocatable rk2_ss_integrator::dwor

  Standard gaussian noise buffers.
- real(kind=8), dimension(:), allocatable rk2_ss_integrator::anoise

  Additive noise term.
- real(kind=8), dimension(:), allocatable rk2_ss_integrator::buf_y1
- real(kind=8), dimension(:), allocatable rk2_ss_integrator::buf_f0
- real(kind=8), dimension(:), allocatable rk2_ss_integrator::buf_f1

  Integration buffers.

9.25 rk2_stoch_integrator.f90 File Reference

Modules

- module rk2_stoch_integrator

  Module with the stochastic rk2 integration routines.

Functions/Subroutines

- subroutine, public rk2_stoch_integrator::init_integrator (force)

  Subroutine to initialize the integrator.
- subroutine rk2_stoch_integrator::tendencies (t, y, res)

  Routine computing the tendencies of the selected model.
- subroutine, public rk2_stoch_integrator::step (y, t, dt, dtn, res, tend)

  Routine to perform a stochastic step of the selected dynamics (Heun algorithm). The incremented time is returned.

Variables

- real(kind=8), dimension(:), allocatable rk2_stoch_integrator::dwar
- real(kind=8), dimension(:), allocatable rk2_stoch_integrator::dwau
- real(kind=8), dimension(:), allocatable rk2_stoch_integrator::dwor
- real(kind=8), dimension(:), allocatable rk2_stoch_integrator::dwou

  Standard gaussian noise buffers.
- real(kind=8), dimension(:), allocatable rk2_stoch_integrator::buf_y1
- real(kind=8), dimension(:), allocatable rk2_stoch_integrator::buf_f0
- real(kind=8), dimension(:), allocatable rk2_stoch_integrator::buf_f1

  Integration buffers.
- real(kind=8), dimension(:), allocatable rk2_stoch_integrator::anoise

  Additive noise term.
- type(coolist), dimension(:), allocatable rk2_stoch_integrator::int_tensor

  Dummy tensor that will hold the tendencies tensor.

9.26 rk2_tl_ad_integrator.f90 File Reference

Modules

- module tl_ad_integrator

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

Functions/Subroutines

- subroutine, public tl_ad_integrator::init_tl_ad_integrator

  Routine to initialise the integration buffers.
- subroutine, public 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 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 tl_ad_integrator::buf_y1

  Buffer to hold the intermediate position (Heun algorithm) of the tangent linear model.
- real(kind=8), dimension(:), allocatable tl_ad_integrator::buf_f0

  Buffer to hold tendencies at the initial position of the tangent linear model.
- real(kind=8), dimension(:), allocatable tl_ad_integrator::buf_f1

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

9.27 rk2_WL_integrator.f90 File Reference

Modules

- module rk2_wl_integrator

  Module with the WL rk2 integration routines.

Functions/Subroutines

- subroutine, public rk2_wl_integrator::init_integrator

  Subroutine that initialize the MARs, the memory unit and the integration buffers.
- subroutine rk2_wl_integrator::compute_m1 (y)

  Routine to compute the M_1 term.
- subroutine rk2_wl_integrator::compute_m2 (y)

  Routine to compute the M_2 term.
- subroutine, public rk2_wl_integrator::step (y, t, dt, dtn, res, tend)

  Routine to perform an integration step (Heun algorithm) of the WL system. The incremented time is returned.
- subroutine, public rk2_wl_integrator::full_step (y, t, dt, dtn, res)

  Routine to perform an integration step (Heun algorithm) of the full stochastic system. The incremented time is returned.

Variables

- real(kind=8), dimension(:), allocatable rk2_wl_integrator::buf_y1
- real(kind=8), dimension(:), allocatable rk2_wl_integrator::buf_f0
- real(kind=8), dimension(:), allocatable rk2_wl_integrator::buf_f1

  Integration buffers.
- real(kind=8), dimension(:), allocatable rk2_wl_integrator::buf_m2
- real(kind=8), dimension(:), allocatable rk2_wl_integrator::buf_m1
- real(kind=8), dimension(:), allocatable rk2_wl_integrator::buf_m3
- real(kind=8), dimension(:), allocatable rk2_wl_integrator::buf_m
- real(kind=8), dimension(:), allocatable rk2_wl_integrator::buf_m3s

  Dummy buffers holding the terms /f$M_i.
- real(kind=8), dimension(:), allocatable rk2_wl_integrator::anoise

  Additive noise term.
- real(kind=8), dimension(:), allocatable rk2_wl_integrator::dwar
- real(kind=8), dimension(:), allocatable rk2_wl_integrator::dwau
- real(kind=8), dimension(:), allocatable rk2_wl_integrator::dwor
- real(kind=8), dimension(:), allocatable rk2_wl_integrator::dwou

  Standard gaussian noise buffers.
- real(kind=8), dimension(:,:), allocatable rk2_wl_integrator::x1

  Buffer holding the subsequent states of the first MAR.
- real(kind=8), dimension(:,:), allocatable rk2_wl_integrator::x2

  Buffer holding the subsequent states of the second MAR.

9.28 rk4_integrator.f90 File Reference

Modules

- module integrator

  Module with the integration routines.

Functions/Subroutines

- subroutine, public integrator::init_integrator

  Routine to initialise the integration buffers.
- subroutine integrator::tendencies (t, y, res)

  Routine computing the tendencies of the model.
- subroutine, public 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 integrator::buf_ka

  Buffer A to hold tendencies.
- real(kind=8), dimension(:), allocatable integrator::buf_kb

  Buffer B to hold tendencies.

9.29 rk4_tl_ad_integrator.f90 File Reference

Modules

- module tl_ad_integrator

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

Functions/Subroutines

- subroutine, public tl_ad_integrator::init_tl_ad_integrator

  Routine to initialise the integration buffers.
- subroutine, public 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 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 tl_ad_integrator::buf_ka

  Buffer to hold tendencies in the RK4 scheme for the tangent linear model.
- real(kind=8), dimension(:), allocatable tl_ad_integrator::buf_kb

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

9.30 sf_def.f90 File Reference

Modules

- module sf_def

  Module to select the resolved-unresolved components.

Functions/Subroutines

- subroutine, public sf_def::load_sf

  Subroutine to load the unresolved variable defintion vector SF from SF.nml if it exists. If it does not, then write SF.nml with no unresolved variables specified (null vector).

Variables

- logical sf_def::exists

    Boolean to test for file existence.
- integer, dimension(:), allocatable, public sf_def::sf

    Unresolved variable definition vector.
- integer, dimension(:), allocatable, public sf_def::ind
- integer, dimension(:), allocatable, public sf_def::rind

    Unresolved reduction indices.
- integer, dimension(:), allocatable, public sf_def::sl_ind
- integer, dimension(:), allocatable, public sf_def::sl_rind

    Resolved reduction indices.
- integer, public sf_def::n_unres

    Number of unresolved variables.
- integer, public sf_def::n_res

    Number of resolved variables.
- integer, dimension(:,:), allocatable, public sf_def::bar
- integer, dimension(:,:), allocatable, public sf_def::bau
- integer, dimension(:,:), allocatable, public sf_def::bor
- integer, dimension(:,:), allocatable, public sf_def::bou

    Filter matrices.

9.31 sqrt_mod.f90 File Reference

Modules

- module sqrt_mod

    Utility module with various routine to compute matrix square root.

Functions/Subroutines

- subroutine, public sqrt_mod::init_sqrt
- subroutine, public sqrt_mod::sqrtm (A, sqA, info, info_triu, bs)

    Routine to compute a real square-root of a matrix.
- logical function sqrt_mod::selectev (a, b)
- subroutine sqrt_mod::sqrtm_triu (A, sqA, info, bs)
- subroutine sqrt_mod::csqrtm_triu (A, sqA, info, bs)
- subroutine sqrt_mod::rsf2csf (T, Z, Tz, Zz)
- subroutine, public sqrt_mod::chol (A, sqA, info)

    Routine to perform a Cholesky decomposition.
- subroutine, public sqrt_mod::sqrtm_svd (A, sqA, info, info_triu, bs)

    Routine to compute a real square-root of a matrix via a SVD decomposition.

Variables

- real(kind=8), dimension(:), allocatable sqrt_mod::work
- integer sqrt_mod::lwork
- real(kind=8), parameter sqrt_mod::real_eps = 2.2204460492503131e-16

9.32 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

9.33 stoch_mod.f90 File Reference

Modules

- module stoch_mod

    Utility module containing the stochastic related routines.

Functions/Subroutines

- real(kind=8) function, public stoch_mod::gasdev ()
- subroutine, public stoch_mod::stoch_vec (dW)
  Routine to fill a vector with standard Gaussian noise process values.
- subroutine, public stoch_mod::stoch_atm_vec (dW)
  routine to fill the atmospheric component of a vector with standard gaussian noise process values
- subroutine, public stoch_mod::stoch_atm_res_vec (dW)
  routine to fill the resolved atmospheric component of a vector with standard gaussian noise process values
- subroutine, public stoch_mod::stoch_atm_unres_vec (dW)
  routine to fill the unresolved atmospheric component of a vector with standard gaussian noise process values
- subroutine, public stoch_mod::stoch_oc_vec (dW)
  routine to fill the oceanic component of a vector with standard gaussian noise process values
- subroutine, public stoch_mod::stoch_oc_res_vec (dW)
  routine to fill the resolved oceanic component of a vector with standard gaussian noise process values
- subroutine, public stoch_mod::stoch_oc_unres_vec (dW)
  routine to fill the unresolved oceanic component of a vector with standard gaussian noise process values

Variables

- integer stoch_mod::iset =0
- real(kind=8) stoch_mod::gset

9.34 stoch_params.f90 File Reference

Modules

- module stoch_params
  The stochastic models parameters module.

Functions/Subroutines

- subroutine stoch_params::init_stoch_params
  Stochastic parameters initialization routine.

Variables

- real(kind=8) stoch_params::mnuti
  Multiplicative noise update time interval.
- real(kind=8) stoch_params::t_trans_stoch
  Transient time period of the stochastic model evolution.
- real(kind=8) stoch_params::q_ar
  Atmospheric resolved component noise amplitude.
- real(kind=8) stoch_params::q_au
  Atmospheric unresolved component noise amplitude.
- real(kind=8) stoch_params::q_or
  Oceanic resolved component noise amplitude.

- real(kind=8) stoch_params::q_ou

  Oceanic unresolved component noise amplitude.
- real(kind=8) stoch_params::dtn

  Square root of the timestep.
- real(kind=8) stoch_params::eps_pert

  Perturbation parameter for the coupling.
- real(kind=8) stoch_params::tdelta

  Time separation parameter.
- real(kind=8) stoch_params::muti

  Memory update time interval.
- real(kind=8) stoch_params::meml

  Time over which the memory kernel is integrated.
- real(kind=8) stoch_params::t_trans_mem

  Transient time period to initialize the memory term.
- character(len=4) stoch_params::x_int_mode

  Integration mode for the resolved component.
- real(kind=8) stoch_params::dts

  Intrisic resolved dynamics time step.
- integer stoch_params::mems

  Number of steps in the memory kernel integral.
- real(kind=8) stoch_params::dtsn

  Square root of the intrisic resolved dynamics time step.
- real(kind=8) stoch_params::maxint

  Upper integration limit of the correlations.
- character(len=4) stoch_params::load_mode

  Loading mode for the correlations.
- character(len=4) stoch_params::int_corr_mode

  Correlation integration mode.
- character(len=4) stoch_params::mode

  Stochastic mode parameter.

9.35 tensor.f90 File Reference

Data Types

- type tensor::coolist_elem

  Coordinate list element type. Elementary elements of the sparse tensors.
- type tensor::coolist_elem4

  4d coordinate list element type. Elementary elements of the 4d sparse tensors.
- type tensor::coolist

  Coordinate list. Type used to represent the sparse tensor.
- type tensor::coolist4

  4d coordinate list. Type used to represent the rank-4 sparse tensor.

Modules

- module tensor

  Tensor utility module.

Functions/Subroutines

- subroutine, public tensor::copy_tensor (src, dst)

  Routine to copy a rank-3 tensor.
- subroutine, public tensor::add_to_tensor (src, dst)

  Routine to add a rank-3 tensor to another one.
- subroutine, public tensor::add_matc_to_tensor (i, src, dst)

  Routine to add a matrix to a rank-3 tensor.
- subroutine, public tensor::add_matc_to_tensor4 (i, j, src, dst)

  Routine to add a matrix to a rank-4 tensor.
- subroutine, public tensor::add_vec_jk_to_tensor (j, k, src, dst)

  Routine to add a vector to a rank-3 tensor.
- subroutine, public tensor::add_vec_ikl_to_tensor4_perm (i, k, l, src, dst)

  Routine to add a vector to a rank-4 tensor plus permutation.
- subroutine, public tensor::add_vec_ikl_to_tensor4 (i, k, l, src, dst)

  Routine to add a vector to a rank-4 tensor.
- subroutine, public tensor::add_vec_ijk_to_tensor4 (i, j, k, src, dst)

  Routine to add a vector to a rank-4 tensor.
- subroutine, public tensor::mat_to_coo (src, dst)

  Routine to convert a matrix to a rank-3 tensor.
- subroutine, public tensor::tensor_to_coo (src, dst)

  Routine to convert a rank-3 tensor from matrix to coolist representation.
- subroutine, public tensor::tensor4_to_coo4 (src, dst)

  Routine to convert a rank-4 tensor from matrix to coolist representation.
- subroutine, public tensor::print_tensor (t)

  Routine to print a rank 3 tensor coolist.
- subroutine, public tensor::print_tensor4 (t)

  Routine to print a rank-4 tensor coolist.
- subroutine, public tensor::sparse_mul3 (coolist_ijk, arr_j, arr_k, res)

  Sparse multiplication of a rank-3 tensor coolist with two vectors: $\sum_{j,k=0}^{ndim} \mathcal{T}_{i,j,k}\, a_j\, b_k$.
- subroutine, public tensor::sparse_mul3_mat (coolist_ijk, arr_k, res)

  Sparse multiplication of a rank-3 tensor coolist with a vector: $\sum_{k=0}^{ndim} \mathcal{T}_{i,j,k}\, b_k$. *Its output is a matrix.*
- subroutine, public tensor::sparse_mul4 (coolist_ijkl, arr_j, arr_k, arr_l, res)

  Sparse multiplication of a rank-4 tensor coolist with three vectors: $\sum_{j,k,l=0}^{ndim} \mathcal{T}_{i,j,k,l}\, a_j\, b_k\, c_l$.
- subroutine, public tensor::sparse_mul4_mat (coolist_ijkl, arr_k, arr_l, res)

  Sparse multiplication of a tensor with two vectors: $\sum_{k,l=0}^{ndim} \mathcal{T}_{i,j,k,l}\, b_k\, c_l$.
- subroutine, public tensor::jsparse_mul (coolist_ijk, arr_j, jcoo_ij)

  Sparse multiplication of two tensors to determine the Jacobian:

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

  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 tensor::jsparse_mul_mat (coolist_ijk, arr_j, jcoo_ij)

Sparse multiplication of two tensors to determine the Jacobian:

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

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_j (coolist_ijk, arr_j, res)

  Sparse multiplication of a 3d sparse tensor with a vectors: $\sum_{j=0}^{ndim} \mathcal{T}_{i,j,k}\, a_j.$
- subroutine, public tensor::sparse_mul2_k (coolist_ijk, arr_k, res)

  Sparse multiplication of a rank-3 sparse tensor coolist with a vector: $\sum_{k=0}^{ndim} \mathcal{T}_{i,j,k}\, a_k.$
- subroutine, public tensor::simplify (tensor)

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

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

  .
- subroutine, public tensor::coo_to_mat_ik (src, dst)

  Routine to convert a rank-3 tensor coolist component into a matrix with i and k indices.
- subroutine, public tensor::coo_to_mat_ij (src, dst)

  Routine to convert a rank-3 tensor coolist component into a matrix with i and j indices.
- subroutine, public tensor::coo_to_mat_i (i, src, dst)

  Routine to convert a rank-3 tensor coolist component into a matrix.
- subroutine, public tensor::coo_to_vec_jk (j, k, src, dst)

  Routine to convert a rank-3 tensor coolist component into a vector.
- subroutine, public tensor::coo_to_mat_j (j, src, dst)

  Routine to convert a rank-3 tensor coolist component into a matrix.
- subroutine, public tensor::sparse_mul4_with_mat_jl (coolist_ijkl, mat_jl, res)

  Sparse multiplication of a rank-4 tensor coolist with a matrix : $\sum_{j,l=0}^{ndim} \mathcal{T}_{i,j,k,l}\, m_{j,l}.$
- subroutine, public tensor::sparse_mul4_with_mat_kl (coolist_ijkl, mat_kl, res)

  Sparse multiplication of a rank-4 tensor coolist with a matrix : $\sum_{j,l=0}^{ndim} \mathcal{T}_{i,j,k,l}\, m_{k,l}.$
- subroutine, public tensor::sparse_mul3_with_mat (coolist_ijk, mat_jk, res)

  Sparse multiplication of a rank-3 tensor coolist with a matrix: $\sum_{j,k=0}^{ndim} \mathcal{T}_{i,j,k}\, m_{j,k}.$
- subroutine, public tensor::matc_to_coo (src, dst)

  Routine to convert a matrix to a rank-3 tensor.
- subroutine, public tensor::scal_mul_coo (s, t)

  Routine to multiply a rank-3 tensor by a scalar.
- logical function, public tensor::tensor_empty (t)

  Test if a rank-3 tensor coolist is empty.
- logical function, public tensor::tensor4_empty (t)

  Test if a rank-4 tensor coolist is empty.
- subroutine, public tensor::load_tensor4_from_file (s, t)

  Load a rank-4 tensor coolist from a file definition.
- subroutine, public tensor::write_tensor4_to_file (s, t)

  Load a rank-4 tensor coolist from a file definition.

Variables

- real(kind=8), parameter tensor::real_eps = 2.2204460492503131e-16
  Parameter to test the equality with zero.

9.36 test_aotensor.f90 File Reference

Functions/Subroutines

- program test_aotensor
  Small program to print the inner products.

9.36.1 Function/Subroutine Documentation

9.36.1.1 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.

9.37 test_corr.f90 File Reference

Functions/Subroutines

- program test_corr
  Small program to print the correlation and covariance matrices.

9.37.1 Function/Subroutine Documentation

9.37.1.1 program test_corr ( )

Small program to print the correlation and covariance matrices.

Copyright

2017 Jonathan Demaeyer. See LICENSE.txt for license information.

Definition at line 12 of file test_corr.f90.

9.38 test_corr_tensor.f90 File Reference

Functions/Subroutines

- program test_corr_tensor
  Small program to print the time correlations tensors.

9.38.1 Function/Subroutine Documentation

9.38.1.1 program test_corr_tensor ( )

Small program to print the time correlations tensors.

Copyright

2017 Jonathan Demaeyer. See LICENSE.txt for license information.

Definition at line 12 of file test_corr_tensor.f90.

9.39 test_dec_tensor.f90 File Reference

Functions/Subroutines

- program test_dec_tensor
  Small program to print the decomposed tensors.

9.39.1 Function/Subroutine Documentation

9.39.1.1 program test_dec_tensor ( )

Small program to print the decomposed tensors.

Copyright

2017 Jonathan Demaeyer. See LICENSE.txt for license information.

Definition at line 12 of file test_dec_tensor.f90.

9.40 test_inprod_analytic.f90 File Reference

Functions/Subroutines

- program inprod_analytic_test
  Small program to print the inner products.

9.40.1 Function/Subroutine Documentation

9.40.1.1 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.

9.41 test_MAR.f90 File Reference

Functions/Subroutines

- program test_mar

  Small program to test the Multivariate AutoRegressive model.

9.41.1 Function/Subroutine Documentation

9.41.1.1 program test_mar ( )

Small program to test the Multivariate AutoRegressive model.

Copyright

2017 Jonathan Demaeyer. See LICENSE.txt for license information.

Definition at line 12 of file test_MAR.f90.

9.42 test_memory.f90 File Reference

Functions/Subroutines

- program test_memory

  Small program to test the WL memory module.

9.42.1 Function/Subroutine Documentation

9.42.1.1 program test_memory ( )

Small program to test the WL memory module.

Copyright

2017 Jonathan Demaeyer. See LICENSE.txt for license information.

Definition at line 12 of file test_memory.f90.

9.43 test_MTV_int_tensor.f90 File Reference

Functions/Subroutines

- program test_mtv_int_tensor

  Small program to print the MTV integrated tensors.

9.43.1 Function/Subroutine Documentation

9.43.1.1 program test_mtv_int_tensor ( )

Small program to print the MTV integrated tensors.

Copyright

2017 Jonathan Demaeyer. See LICENSE.txt for license information.

Definition at line 12 of file test_MTV_int_tensor.f90.

9.44 test_MTV_sigma_tensor.f90 File Reference

Functions/Subroutines

- program test_sigma

  Small program to test the MTV noise sigma matrices.

9.44.1 Function/Subroutine Documentation

9.44.1.1 program test_sigma ( )

Small program to test the MTV noise sigma matrices.

Copyright

2017 Jonathan Demaeyer. See LICENSE.txt for license information.

Definition at line 12 of file test_MTV_sigma_tensor.f90.

9.45 test_sqrtm.f90 File Reference

Functions/Subroutines

- program test_sqrtm

    Small program to test the matrix square-root module.

9.45.1 Function/Subroutine Documentation

9.45.1.1 program test_sqrtm ( )

Small program to test the matrix square-root module.

Copyright

2017 Jonathan Demaeyer. See LICENSE.txt for license information.

Definition at line 12 of file test_sqrtm.f90.

9.46 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.

9.46.1 Function/Subroutine Documentation

9.46.1.1 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.

9.47 test_WL_tensor.f90 File Reference

Functions/Subroutines

- program test_wl_tensor

    Small program to print the WL tensors.

9.47.1 Function/Subroutine Documentation

9.47.1.1 program test_wl_tensor ( )

Small program to print the WL tensors.

Copyright

2017 Jonathan Demaeyer. See LICENSE.txt for license information.

Definition at line 11 of file test_WL_tensor.f90.

9.48 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), public 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, deltay, buf)

    Tendencies for the AD of MAOOAM in point ystar for perturbation deltay.
- subroutine, public tl_ad_tensor::tl (t, ystar, deltay, buf)

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

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.

9.49 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.
- real(kind=8) function, public util::mat_trace (A)
- real(kind=8) function, public util::mat_contract (A, B)
- subroutine, public util::choldc (a, p)
- subroutine, public util::printmat (A)
- subroutine, public util::cprintmat (A)
- real(kind=8) function, dimension(size(a, 1), size(a, 2)), public util::invmat (A)
- subroutine, public util::triu (A, T)
- subroutine, public util::diag (A, d)
- subroutine, public util::cdiag (A, d)
- integer function, public util::floordiv (i, j)
- subroutine, public util::reduce (A, Ared, n, ind, rind)
- subroutine, public util::ireduce (A, Ared, n, ind, rind)
- subroutine, public util::vector_outer (u, v, A)

9.49.1 Function/Subroutine Documentation

9.49.1.1 integer function init_random_seed::lcg (integer(int64) *s*)

Definition at line 85 of file util.f90.

```
integer :: lcg
integer(int64) :: s
IF (s == 0) THEN
   s = 104729
ELSE
   s = mod(s, 4294967296_int64)
END IF
s = mod(s * 279470273_int64, 4294967291_int64)
lcg = int(mod(s, int(huge(0), int64)), kind(0))
END FUNCTION lcg
```

9.50 WL_tensor.f90 File Reference

Modules

- module wl_tensor

    The WL tensors used to integrate the model.

Functions/Subroutines

- subroutine, public wl_tensor::init_wl_tensor

    Subroutine to initialise the WL tensor.

Variables

- real(kind=8), dimension(:), allocatable, public wl_tensor::m11

    First component of the M1 term.
- type(coolist), dimension(:), allocatable, public wl_tensor::m12

    Second component of the M1 term.
- real(kind=8), dimension(:), allocatable, public wl_tensor::m13

    Third component of the M1 term.
- real(kind=8), dimension(:), allocatable, public wl_tensor::m1tot

    Total M_1 vector.
- type(coolist), dimension(:), allocatable, public wl_tensor::m21

    First tensor of the M2 term.
- type(coolist), dimension(:), allocatable, public wl_tensor::m22

    Second tensor of the M2 term.
- type(coolist), dimension(:,:), allocatable, public wl_tensor::l1

    First linear tensor.
- type(coolist), dimension(:,:), allocatable, public wl_tensor::l2

    Second linear tensor.
- type(coolist), dimension(:,:), allocatable, public wl_tensor::l4

    Fourth linear tensor.
- type(coolist), dimension(:,:), allocatable, public wl_tensor::l5

    Fifth linear tensor.

- type(coolist), dimension(:,:), allocatable, public wl_tensor::ltot

  Total linear tensor.
- type(coolist), dimension(:,:), allocatable, public wl_tensor::b1

  First quadratic tensor.
- type(coolist), dimension(:,:), allocatable, public wl_tensor::b2

  Second quadratic tensor.
- type(coolist), dimension(:,:), allocatable, public wl_tensor::b3

  Third quadratic tensor.
- type(coolist), dimension(:,:), allocatable, public wl_tensor::b4

  Fourth quadratic tensor.
- type(coolist), dimension(:,:), allocatable, public wl_tensor::b14

  Joint 1st and 4th tensors.
- type(coolist), dimension(:,:), allocatable, public wl_tensor::b23

  Joint 2nd and 3rd tensors.
- type(coolist4), dimension(:,:), allocatable, public wl_tensor::mtot

  Tensor for the cubic terms.
- real(kind=8), dimension(:), allocatable wl_tensor::dumb_vec

  Dummy vector.
- real(kind=8), dimension(:,:), allocatable wl_tensor::dumb_mat1

  Dummy matrix.
- real(kind=8), dimension(:,:), allocatable wl_tensor::dumb_mat2

  Dummy matrix.
- real(kind=8), dimension(:,:), allocatable wl_tensor::dumb_mat3

  Dummy matrix.
- real(kind=8), dimension(:,:), allocatable wl_tensor::dumb_mat4

  Dummy matrix.
- logical, public wl_tensor::m12def
- logical, public wl_tensor::m21def
- logical, public wl_tensor::m22def
- logical, public wl_tensor::ldef
- logical, public wl_tensor::b14def
- logical, public wl_tensor::b23def
- logical, public wl_tensor::mdef

  Boolean to (de)activate the computation of the terms.

Index