Reference Manual

Generated by Doxygen 1.8.3.1

Thu Mar 24 2016 14:02:17

Contents

Chapter 1

Main Page

About

This software is provided as supplementary material with:

- De Cruz, L., Demaeyer, J. and Vannitsem, S.: A modular arbitrary-order ocean-atmosphere model: MAOOAM, Geosci. Model Dev. Discuss., doi:xx/xxx, 2016.

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

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

Consult the MAOOAM code repository at doi:yy/yyy for updates, and our website for additional resources.

A pdf version of this manual is available here.

Installation

The code should be compiled with gfortran 4.7+ (allows for allocatable arrays in namelists). Unpack the archive in a folder, and run: make

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

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

Description of the files

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

- maooam.f90 : Main program.
- aotensor_def.f90 : Tensor aotensor computation module.
- IC_def.f90 : A module which loads the user specified initial condition.
- inprod_analytic.f90 : Inner products computation module.
- integrator.f90 : A module which contains the integrator for the model equations.

- Makefile : The Makefile.
- params.f90 : The model parameters module.
- maooam_tl_ad.f90 : Tangent Linear (TL) and Adjoint (AD) model tensors definition module
- tl_ad_integrator.f90 : 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.
- stat.f90 : A module for statistic accumulation.
- params.nml : A namelist to specify the model parameters.
- int_params.nml : A namelist to specify the integration parameters.
- modeselection.nml : A namelist to specify which spectral decomposition will be used.

Usage

The user first has to fill the params.nml and int_params.nml namelist files according to their needs.

The modeselection.nml namelist can then be filled :

- NBOC and NBATM specify the number of blocks that will be used in respectively the ocean and the atmosphere. Each block corresponds to a given x and y wavenumber.
- The OMS and AMS arrays are integer arrays which specify which wavenumbers of the spectral decomposition will be used in respectively the ocean and the atmosphere. Their shapes are OMS(NBOC,2) and AMS(NBATM,2).
- The first dimension specifies the number attributed by the user to the block and the second dimension specifies the x and the y wavenumbers.
- The VDDG model, described in Vannitsem et al. (2015) is given as an example in the archive.
- Note that the variables of the model are numbered according to the chosen order of the blocks.

Model and integration parameters can be specified in the params.nml namelist file.

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 maooam_tl_ad and tl_ad_integrator module. It is documented here.

Implementation notes

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

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

with $y_0 = 1$.

The tensor aotensor_def::aotensor is the tensor $\mathscr{T}$ that encodes the differential equations is composed so that:

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

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

The tensor for this model is composed in aotensor_def and uses the inner products defined in inprod_analytic .

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 $\mathscr{T}_{i,j,k}^{TD}$ is defined as:

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

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

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

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

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

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

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

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

Implementation :

The two tensors are implemented in the module maooam_tl_ad and must be initialized (after calling params::init_params and aotensor_def::aotensor) by calling maooam_tl_ad::init_tltensor() and maooam_tl_ad::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 tl_ad_maooam. An example on how to use it can be found in the test file test_tl_ad.f90

Chapter 3

Data Type Index

3.1 Data Types List

Here are the data types with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Data Type Documentation

5.1 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.

Public Member Functions

- subroutine, public init_aotensor

  Subroutine to initialise the aotensor tensor.

Public Attributes

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

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

Private Member Functions

- 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 $\mathscr{T}_{i,j,k}$ *structure.*
- subroutine add_count (i, j, k, v)

  Subroutine to count the elements of the aotensor $\mathscr{T}_{i,j,k}$. *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.

Private Attributes

- 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.

5.1.1 Detailed Description

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

Copyright

2015 Lesley De Cruz & Jonathan Demaeyer. See LICENSE.txt for license information.

Remarks

Generated Fortran90/95 code from ../../lua/aomodel_extended/aotensor.lua

Definition at line 21 of file aotensor_def.f90.

5.1.2 Member Function/Subroutine Documentation

5.1.2.1 integer function aotensor_def::a (integer *i*) [private]

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

Definition at line 75 of file aotensor_def.f90.

5.1.2.2 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 $\mathscr{T}_{i,j,k}$. 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 123 of file aotensor_def.f90.

5.1.2.3 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 $\mathscr{T}_{i,j,k}$ structure.

Parameters

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

Definition at line 98 of file aotensor_def.f90.

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

Subroutine to compute the tensor aotensor.

Parameters

func	External function to be used

Definition at line 131 of file aotensor_def.f90.

5.1.2.5 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 201 of file aotensor_def.f90.

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

Kronecker delta function.

Definition at line 87 of file aotensor_def.f90.

5.1.2.7 integer function aotensor_def::psi (integer *i*) [private]

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

Definition at line 63 of file aotensor_def.f90.

5.1.2.8 integer function aotensor_def::t (integer *i*) [private]

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

Definition at line 81 of file aotensor_def.f90.

5.1.2.9 integer function aotensor_def::theta (integer *i*) [private]

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

Definition at line 69 of file aotensor_def.f90.

5.1.3 Member Data Documentation

5.1.3.1 type(coolist), dimension(:), allocatable, public aotensor_def::aotensor

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

Definition at line 45 of file aotensor_def.f90.

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

Vector used to count the tensor elements.

Definition at line 37 of file aotensor_def.f90.

5.1.3.3 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.

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

- aotensor_def.f90

5.2 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

5.2.1 Detailed Description

Type holding the atmospheric inner products tensors.

Definition at line 52 of file inprod_analytic.f90.

5.2.2 Member Data Documentation

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

Definition at line 53 of file inprod_analytic.f90.

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

Definition at line 54 of file inprod_analytic.f90.

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

Definition at line 53 of file inprod_analytic.f90.

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

Definition at line 53 of file inprod_analytic.f90.

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

Definition at line 54 of file inprod_analytic.f90.

5.2.2.6 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

5.3 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.

5.3.1 Detailed Description

Atmospheric bloc specification type.

Definition at line 39 of file inprod_analytic.f90.

5.3.2 Member Data Documentation

5.3.2.1 integer inprod_analytic::atm_wavenum::h =0 [private]

Definition at line 41 of file inprod_analytic.f90.

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

Definition at line 41 of file inprod_analytic.f90.

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

Definition at line 42 of file inprod_analytic.f90.

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

Definition at line 42 of file inprod_analytic.f90.

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

Definition at line 41 of file inprod_analytic.f90.

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

Definition at line 40 of file inprod_analytic.f90.

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

- inprod_analytic.f90

5.4 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.

5.4.1 Detailed Description

Coordinate list. Type used to represent the sparse tensor.

Definition at line 27 of file tensor.f90.

5.4.2 Member Data Documentation

5.4.2.1 type(coolist_elem), dimension(:), allocatable tensor::coolist::elems

Lists of elements tensor::coolist_elem.

Definition at line 28 of file tensor.f90.

5.4.2.2 integer tensor::coolist::nelems = 0

Number of elements in the list.

Definition at line 29 of file tensor.f90.

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

- tensor.f90

5.5 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.

5.5.1 Detailed Description

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

Definition at line 20 of file tensor.f90.

5.5.2 Member Data Documentation

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

Index j of the element.

Definition at line 21 of file tensor.f90.

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

Index k of the element.

Definition at line 22 of file tensor.f90.

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

Value of the element.

Definition at line 23 of file tensor.f90.

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

- tensor.f90

5.6 ic_def Module Reference

Module to load the initial condition.

Public Member Functions

- 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.

Public Attributes

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

  Initial condition vector.

Private Attributes

- logical exists

  Boolean to test for file existence.

5.6.1 Detailed Description

Module to load the initial condition.

Copyright

2015 Lesley De Cruz & Jonathan Demaeyer. See LICENSE.txt for license information.

Definition at line 12 of file ic_def.f90.

5.6.2 Member Function/Subroutine Documentation

5.6.2.1 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 31 of file ic_def.f90.

5.6.3 Member Data Documentation

5.6.3.1 logical ic_def::exists [private]

Boolean to test for file existence.

Definition at line 21 of file ic_def.f90.

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

Initial condition vector.

Definition at line 23 of file ic_def.f90.

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

- ic_def.f90

5.7 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.

Public Member Functions

- subroutine, public init_inprod

  Initialisation of the inner product.
- subroutine, public deallocate_inprod

  Deallocation of the inner products.

Public Attributes

- 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.

Private Member Functions

- 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.

5.7.1 Detailed Description

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

Copyright

2015 Lesley De Cruz & Jonathan Demaeyer. See LICENSE.txt for license information.

Remarks

Generated Fortran90/95 code from ../../lua/aomodel_extended/inprod_analytic.lua

Definition at line 23 of file inprod_analytic.f90.

5.7.2 Member Function/Subroutine Documentation

5.7.2.1 REAL(KIND=8) function inprod_analytic::b1 (integer *Pi*, integer *Pj*, integer *Pk*) [private]

Cehelsky & Tung Helper functions.

Definition at line 90 of file inprod_analytic.f90.

5.7.2.2 REAL(KIND=8) function inprod_analytic::b2 (integer *Pi*, integer *Pj*, integer *Pk*) [private]

Cehelsky & Tung Helper functions.

Definition at line 96 of file inprod_analytic.f90.

5.7.2.3 subroutine inprod_analytic::calculate_a ( ) [private]

Eigenvalues of the Laplacian (atmospheric)

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

Definition at line 154 of file inprod_analytic.f90.

5.7.2.4 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 181 of file inprod_analytic.f90.

5.7.2.5 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 215 of file inprod_analytic.f90.

5.7.2.6 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 567 of file inprod_analytic.f90.

5.7.2.7 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 254 of file inprod_analytic.f90.

5.7.2.8 subroutine inprod_analytic::calculate_g ( ) [private]

Temperature advection terms (atmospheric)

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

Definition at line 287 of file inprod_analytic.f90.

5.7.2.9 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 433 of file inprod_analytic.f90.

5.7.2.10 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 463 of file inprod_analytic.f90.

5.7.2.11 subroutine inprod_analytic::calculate_n ( ) [private]

Beta term for the ocean.

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

Definition at line 486 of file inprod_analytic.f90.

5.7.2.12 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 515 of file inprod_analytic.f90.

5.7.2.13 subroutine inprod_analytic::calculate_s ( ) [private]

Forcing (thermal) of the ocean on the atmosphere.

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

Definition at line 379 of file inprod_analytic.f90.

5.7.2.14 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 604 of file inprod_analytic.f90.

5.7.2.15 subroutine, public inprod_analytic::deallocate_inprod ( )

Deallocation of the inner products.

Definition at line 721 of file inprod_analytic.f90.

5.7.2.16 REAL(KIND=8) function inprod_analytic::delta (integer *r*) [private]

Integer Dirac delta function.

Definition at line 102 of file inprod_analytic.f90.

5.7.2.17 REAL(KIND=8) function inprod_analytic::flambda (integer *r*) [private]

"Odd or even" function

Definition at line 112 of file inprod_analytic.f90.

5.7.2.18 subroutine, public inprod_analytic::init_inprod ( )

Initialisation of the inner product.

Definition at line 638 of file inprod_analytic.f90.

5.7.2.19 REAL(KIND=8) function inprod_analytic::s1 (integer *Pj,* integer *Pk,* integer *Mj,* integer *Hk*) [private]

Cehelsky & Tung Helper functions.

Definition at line 122 of file inprod_analytic.f90.

5.7.2.20 REAL(KIND=8) function inprod_analytic::s2 (integer *Pj,* integer *Pk,* integer *Mj,* integer *Hk*) [private]

Cehelsky & Tung Helper functions.

Definition at line 128 of file inprod_analytic.f90.

5.7.2.21 REAL(KIND=8) function inprod_analytic::s3 (integer *Pj,* integer *Pk,* integer *Hj,* integer *Hk*) [private]

Cehelsky & Tung Helper functions.

Definition at line 134 of file inprod_analytic.f90.

5.7.2.22 REAL(KIND=8) function inprod_analytic::s4 (integer *Pj,* integer *Pk,* integer *Hj,* integer *Hk*) [private]

Cehelsky & Tung Helper functions.

Definition at line 140 of file inprod_analytic.f90.

5.7.3 Member Data Documentation

5.7.3.1 type(atm_tensors), public inprod_analytic::atmos

Atmospheric tensors.

Definition at line 69 of file inprod_analytic.f90.

5.7.3.2 type(atm_wavenum), dimension(:), allocatable, public inprod_analytic::awavenum

Atmospheric blocs specification.

Definition at line 64 of file inprod_analytic.f90.

5.7.3.3 type(ocean_tensors), public inprod_analytic::ocean

Oceanic tensors.

Definition at line 71 of file inprod_analytic.f90.

5.7.3.4 type(ocean_wavenum), dimension(:), allocatable, public inprod_analytic::owavenum

Oceanic blocs specification.

Definition at line 66 of file inprod_analytic.f90.

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

- inprod_analytic.f90

5.8 integrator Module Reference

Module with the integration routines.

Public Member Functions

- subroutine, public init_integrator

  Routine to initialise the integration buffers.
- subroutine, public step (y, t, dt, res)

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

Private Member Functions

- subroutine tendencies (t, y, res)

  Routine computing the tendencies of the model.

Private Attributes

- 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.

5.8.1 Detailed Description

Module with the integration routines.

Copyright

2015 Lesley De Cruz & Jonathan Demaeyer. See LICENSE.txt for license information.

Remarks

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

Definition at line 19 of file integrator.f90.

5.8.2 Member Function/Subroutine Documentation

5.8.2.1 subroutine, public integrator::init_integrator ( )

Routine to initialise the integration buffers.

Definition at line 36 of file integrator.f90.

5.8.2.2 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.

Parameters

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

Definition at line 60 of file integrator.f90.

5.8.2.3 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 48 of file integrator.f90.

5.8.3 Member Data Documentation

5.8.3.1 real(kind=8), dimension(:), allocatable integrator::buf_f0 [private]

Buffer to hold tendencies at the initial position.

Definition at line 28 of file integrator.f90.

5.8.3.2 real(kind=8), dimension(:), allocatable integrator::buf_f1 [private]

Buffer to hold tendencies at the intermediate position.

Definition at line 29 of file integrator.f90.

5.8.3.3 real(kind=8), dimension(:), allocatable integrator::buf_y1 [private]

Buffer to hold the intermediate position (Heun algorithm)

Definition at line 27 of file integrator.f90.

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

- integrator.f90

5.9 maooam_tl_ad Module Reference

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

Public Member Functions

- subroutine, public init_tltensor
  Routine to initialize the TL tensor.
- subroutine, public init_adtensor
  Routine to initialize the AD tensor.
- subroutine, public init_adtensor_ref
  Alternate method to initialize the AD tensor 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.

Public Attributes

- 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.

Private Member Functions

- type(coolist) function,
  dimension(ndim) jacobian (ystar)

  Compute the Jacobian of MAOOAM in point ystar.
- real(kind=8) function,
  dimension(ndim, ndim) jacobian_mat (ystar)

  Compute the Jacobian of MAOOAM in point ystar.
- subroutine 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 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 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.

Private Attributes

- 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.

5.9.1 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 !

Definition at line 19 of file maooam_tl_ad.f90.

5.9.2 Member Function/Subroutine Documentation

5.9.2.1 subroutine, public maooam_tl_ad::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 390 of file maooam_tl_ad.f90.

5.9.2.2 subroutine maooam_tl_ad::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 249 of file maooam_tl_ad.f90.

5.9.2.3 subroutine maooam_tl_ad::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 352 of file maooam_tl_ad.f90.

5.9.2.4 subroutine maooam_tl_ad::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 263 of file maooam_tl_ad.f90.

5.9.2.5 subroutine maooam_tl_ad::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 364 of file maooam_tl_ad.f90.

5.9.2.6 subroutine maooam_tl_ad::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 223 of file maooam_tl_ad.f90.

5.9.2.7 subroutine maooam_tl_ad::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 324 of file maooam_tl_ad.f90.

5.9.2.8 subroutine maooam_tl_ad::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 127 of file maooam_tl_ad.f90.

5.9.2.9 subroutine, public maooam_tl_ad::init_adtensor ( )

Routine to initialize the AD tensor.

Definition at line 199 of file maooam_tl_ad.f90.

5.9.2.10 subroutine, public maooam_tl_ad::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 300 of file maooam_tl_ad.f90.

5.9.2.11 subroutine, public maooam_tl_ad::init_tltensor ( )

Routine to initialize the TL tensor.

Definition at line 103 of file maooam_tl_ad.f90.

5.9.2.12 type(coolist) function, dimension(ndim) maooam_tl_ad::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 81 of file maooam_tl_ad.f90.

5.9.2.13 real(kind=8) function, dimension(ndim,ndim) maooam_tl_ad::jacobian_mat (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 matrix form

Definition at line 90 of file maooam_tl_ad.f90.

5.9.2.14 subroutine, public maooam_tl_ad::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 402 of file maooam_tl_ad.f90.

5.9.2.15 subroutine maooam_tl_ad::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 153 of file maooam_tl_ad.f90.

5.9.2.16 subroutine maooam_tl_ad::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 167 of file maooam_tl_ad.f90.

5.9.3 Member Data Documentation

5.9.3.1 type(coolist), dimension(:), allocatable, public maooam_tl_ad::adtensor

Tensor representation of the Adjoint tendencies.

Definition at line 51 of file maooam_tl_ad.f90.

5.9.3.2 integer, dimension(:), allocatable maooam_tl_ad::count_elems [private]

Vector used to count the tensor elements.

Definition at line 45 of file maooam_tl_ad.f90.

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

Epsilon to test equality with 0.

Definition at line 42 of file maooam_tl_ad.f90.

5.9.3.4 type(coolist), dimension(:), allocatable, public maooam_tl_ad::tltensor

Tensor representation of the Tangent Linear tendencies.

Definition at line 48 of file maooam_tl_ad.f90.

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

- maooam_tl_ad.f90

5.10 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

5.10.1 Detailed Description

Type holding the oceanic inner products tensors.

Definition at line 58 of file inprod_analytic.f90.

5.10.2 Member Data Documentation

5.10.2.1 real(kind=8), dimension(:,:,:), allocatable inprod_analytic::ocean_tensors::c [private]

Definition at line 60 of file inprod_analytic.f90.

5.10.2.2 real(kind=8), dimension(:,:), allocatable inprod_analytic::ocean_tensors::k [private]

Definition at line 59 of file inprod_analytic.f90.

5.10.2.3 real(kind=8), dimension(:,:), allocatable inprod_analytic::ocean_tensors::m [private]

Definition at line 59 of file inprod_analytic.f90.

5.10.2.4 real(kind=8), dimension(:,:), allocatable inprod_analytic::ocean_tensors::n [private]

Definition at line 59 of file inprod_analytic.f90.

5.10.2.5 real(kind=8), dimension(:,:,:), allocatable inprod_analytic::ocean_tensors::o [private]

Definition at line 60 of file inprod_analytic.f90.

5.10.2.6 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

5.11 inprod_analytic::ocean_wavenum Type Reference

Oceanic bloc specification type.

Private Attributes

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

5.11.1 Detailed Description

Oceanic bloc specification type.

Definition at line 46 of file inprod_analytic.f90.

5.11.2 Member Data Documentation

5.11.2.1 integer inprod_analytic::ocean_wavenum::h [private]

Definition at line 47 of file inprod_analytic.f90.

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

Definition at line 48 of file inprod_analytic.f90.

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

Definition at line 48 of file inprod_analytic.f90.

5.11.2.4 integer inprod_analytic::ocean_wavenum::p [private]

Definition at line 47 of file inprod_analytic.f90.

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

- inprod_analytic.f90

5.12 params Module Reference

The model parameters module.

Public Member Functions

- subroutine init_params

  Parameters initialisation routine.

Public Attributes

- 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
- 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.

Private Member Functions

- subroutine, private init_nml

Read the basic parameters and mode selection from the namelist.

5.12.1 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

Definition at line 18 of file params.f90.

5.12.2 Member Function/Subroutine Documentation

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

Read the basic parameters and mode selection from the namelist.

Definition at line 90 of file params.f90.

5.12.2.2 subroutine params::init_params ( )

Parameters initialisation routine.

Definition at line 134 of file params.f90.

5.12.3 Member Data Documentation

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

Atmospheric mode selection array.

Definition at line 81 of file params.f90.

5.12.3.2 real(kind=8) params::betp

Definition at line 67 of file params.f90.

5.12.3.3 real(kind=8) params::ca

C_a - Constant short-wave radiation of the atmosphere.

Definition at line 40 of file params.f90.

5.12.3.4 real(kind=8) params::co

C_a - Constant short-wave radiation of the ocean.

Definition at line 38 of file params.f90.

5.12.3.5 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.

5.12.3.6 real(kind=8) params::cpo

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

Definition at line 56 of file params.f90.

5.12.3.7 real(kind=8) params::d

Merchanical coupling parameter between the ocean and the atmosphere.

Definition at line 31 of file params.f90.

5.12.3.8 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.

5.12.3.9 real(kind=8) params::dt

Integration time step.

Definition at line 71 of file params.f90.

5.12.3.10 real(kind=8) params::epsa

ε_a - Emissivity coefficient for the grey-body atmosphere.

Definition at line 43 of file params.f90.

5.12.3.11 real(kind=8) params::f0

f_0 - Coriolis parameter

Definition at line 32 of file params.f90.

5.12.3.12 real(kind=8) params::g

γ

Definition at line 50 of file params.f90.

5.12.3.13 real(kind=8) params::ga

γ_a - Specific heat capacity of the atmosphere.

Definition at line 44 of file params.f90.

5.12.3.14 real(kind=8) params::go

γ_o - Specific heat capacity of the ocean.

Definition at line 39 of file params.f90.

5.12.3.15 real(kind=8) params::gp

g'Reduced gravity

Definition at line 33 of file params.f90.

5.12.3.16 real(kind=8) params::h

Depth of the active water layer of the ocean.

Definition at line 34 of file params.f90.

5.12.3.17 real(kind=8) params::k

Bottom atmospheric friction coefficient.

Definition at line 28 of file params.f90.

5.12.3.18 real(kind=8) params::kd

k_d - Non-dimensional bottom atmospheric friction coefficient.

Definition at line 53 of file params.f90.

5.12.3.19 real(kind=8) params::kdp

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

Definition at line 54 of file params.f90.

5.12.3.20 real(kind=8) params::kp

k' - Internal atmospheric friction coefficient.

Definition at line 29 of file params.f90.

5.12.3.21 real(kind=8) params::l

L - Domain length scale

Definition at line 64 of file params.f90.

5.12.3.22 real(kind=8) params::lambda

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

Definition at line 37 of file params.f90.

5.12.3.23 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.

5.12.3.24 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.

5.12.3.25 real(kind=8) params::lr

L_R - Rossby deformation radius

Definition at line 49 of file params.f90.

5.12.3.26 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.

5.12.3.27 real(kind=8) params::lsbpo

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

Definition at line 62 of file params.f90.

5.12.3.28 real(kind=8) params::n

$n = 2L_y/L_x$ - Aspect ratio

Definition at line 24 of file params.f90.

5.12.3.29 integer params::natm =0

Number of atmospheric basis functions.

Definition at line 77 of file params.f90.

5.12.3.30 integer params::nbatm

Number of oceanic blocks.

Definition at line 76 of file params.f90.

5.12.3.31 integer params::nboc

Number of atmospheric blocks.

Definition at line 75 of file params.f90.

5.12.3.32 integer params::ndim

Number of variables (dimension of the model)

Definition at line 79 of file params.f90.

5.12.3.33 integer params::noc =0

Number of oceanic basis functions.

Definition at line 78 of file params.f90.

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

Ocean mode selection array.

Definition at line 80 of file params.f90.

5.12.3.35 real(kind=8) params::phi0

Latitude in radian.

Definition at line 25 of file params.f90.

5.12.3.36 real(kind=8) params::phi0_npi

Latitude exprimed in fraction of pi.

Definition at line 35 of file params.f90.

5.12.3.37 real(kind=8) params::pi

π

Definition at line 48 of file params.f90.

5.12.3.38 real(kind=8) params::r

Frictional coefficient at the bottom of the ocean.

Definition at line 30 of file params.f90.

5.12.3.39 real(kind=8) params::rp

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

Definition at line 51 of file params.f90.

5.12.3.40 real(kind=8) params::rr

R - Gas constant of dry air

Definition at line 45 of file params.f90.

5.12.3.41 real(kind=8) params::rra

Earth radius.

Definition at line 26 of file params.f90.

5.12.3.42 real(kind=8) params::sb

Stefan–Boltzmann constant.

Definition at line 66 of file params.f90.

5.12.3.43 real(kind=8) params::sbpa

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

Definition at line 61 of file params.f90.

5.12.3.44 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.

5.12.3.45 real(kind=8) params::sc

Ratio of surface to atmosphere temperature.

Definition at line 65 of file params.f90.

5.12.3.46 real(kind=8) params::scale

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

Definition at line 47 of file params.f90.

5.12.3.47 real(kind=8) params::sig0

σ_0 - Non-dimensional static stability of the atmosphere.

Definition at line 27 of file params.f90.

5.12.3.48 real(kind=8) params::t_run

Effective intergration time (length of the generated trajectory)

Definition at line 70 of file params.f90.

5.12.3.49 real(kind=8) params::t_trans

Transient time period.

Definition at line 69 of file params.f90.

5.12.3.50 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.

5.12.3.51 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.

5.12.3.52 real(kind=8) params::tw

Write all variables every tw time units.

Definition at line 72 of file params.f90.

5.12.3.53 logical params::writeout

Write to file boolean.

Definition at line 73 of file params.f90.

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

- params.f90

5.13 stat Module Reference

Statistics accumulators.

Public Member Functions

- 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.

Private Attributes

- 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

5.13.1 Detailed Description

Statistics accumulators.

Copyright

Definition at line 14 of file stat.f90.

5.13.2 Member Function/Subroutine Documentation

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

Accumulate one state.

Definition at line 47 of file stat.f90.

5.13.2.2 subroutine, public stat::init_stat ( )

Initialise the accumulators.

Definition at line 34 of file stat.f90.

5.13.2.3 integer function, public stat::iter ( )

Function returning the number of data accumulated.

Definition at line 71 of file stat.f90.

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

Function returning the mean.

Definition at line 59 of file stat.f90.

5.13.2.5 subroutine, public stat::reset ( )

Routine resetting the accumulators.

Definition at line 77 of file stat.f90.

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

Function returning the variance.

Definition at line 65 of file stat.f90.

5.13.3 Member Data Documentation

5.13.3.1 integer stat::i =0 [private]

Number of stats accumulated.

Definition at line 20 of file stat.f90.

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

Vector storing the inline mean.

Definition at line 23 of file stat.f90.

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

Previous mean vector.

Definition at line 24 of file stat.f90.

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

Definition at line 26 of file stat.f90.

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

Vector storing the inline variance.

Definition at line 25 of file stat.f90.

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

- stat.f90

5.14 tensor Module Reference

Tensor utility module.

Data Types

- type coolist

  Coordinate list. Type used to represent the sparse tensor.
- type coolist_elem

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

Public Member Functions

- subroutine, public copy_tensor (src, dst)

  Routine to copy a tensor.
- subroutine, public mat_to_coo (src, dst)

  Routine to convert a matrix to a tensor.
- subroutine, public sparse_mul3 (coolist_ijk, arr_j, arr_k, res)

  Sparse multiplication of a tensor with two vectors: $\sum_{j,k=0}^{ndim} \mathcal{T}_{i,j,k}\, a_j\, b_k$.
- subroutine, public jsparse_mul (coolist_ijk, arr_j, jcoo_ij)

  Sparse multiplication of two tensors to determine the Jacobian:

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

  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 (coolist_ij, arr_j, res)

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

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

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

  .

Public Attributes

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

  Parameter to test the equality with zero.

5.14.1 Detailed Description

Tensor utility module.

Copyright

Definition at line 13 of file tensor.f90.

5.14.2 Member Function/Subroutine Documentation

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

Routine to copy a tensor.

Parameters

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 43 of file tensor.f90.

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

Sparse multiplication of two tensors to determine the Jacobian:

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

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

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

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 122 of file tensor.f90.

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

Sparse multiplication of two tensors to determine the Jacobian:

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

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 165 of file tensor.f90.

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

Routine to convert a matrix to a tensor.

Parameters

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.

Definition at line 65 of file tensor.f90.

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

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

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

.

Parameters

tensor	a coordinate list (sparse tensor) which will be simplified.

Definition at line 207 of file tensor.f90.

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

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

Parameters

coolist_ij	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 190 of file tensor.f90.

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

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

Parameters

coolist_ijk	a coordinate list (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 98 of file tensor.f90.

5.14.3 Member Data Documentation

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

Parameter to test the equality with zero.

Definition at line 33 of file tensor.f90.

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

- tensor.f90

5.15 tl_ad_integrator Module Reference

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

Public Member Functions

- subroutine, public init_ad_integrator

  Routine to initialise the adjoint model 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 init_tl_integrator

  Routine to initialise the tangent linear model integration buffers.
- 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.

Private Attributes

- real(kind=8), dimension(:),
  allocatable ad_buf_y1

  Buffer to hold the intermediate position (Heun algorithm) of the adjoint model.
- real(kind=8), dimension(:),
  allocatable ad_buf_f0

  Buffer to hold tendencies at the initial position of the adjoint model.
- real(kind=8), dimension(:),
  allocatable ad_buf_f1

  Buffer to hold tendencies at the intermediate position of the adjoint model.
- real(kind=8), dimension(:),
  allocatable tl_buf_y1

  Buffer to hold the intermediate position (Heun algorithm) of the tangent linear model.
- real(kind=8), dimension(:),
  allocatable tl_buf_f0

  Buffer to hold tendencies at the initial position of the tangent linear model.
- real(kind=8), dimension(:),
  allocatable tl_buf_f1

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

5.15.1 Detailed Description

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

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.

Definition at line 22 of file tl_ad_integrator.f90.

5.15.2 Member Function/Subroutine Documentation

5.15.2.1 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.

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 62 of file tl_ad_integrator.f90.

5.15.2.2 subroutine, public tl_ad_integrator::init_ad_integrator ( )

Routine to initialise the adjoint model integration buffers.

Definition at line 50 of file tl_ad_integrator.f90.

5.15.2.3 subroutine, public tl_ad_integrator::init_tl_integrator ( )

Routine to initialise the tangent linear model integration buffers.

Definition at line 82 of file tl_ad_integrator.f90.

5.15.2.4 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.

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 94 of file tl_ad_integrator.f90.

5.15.3 Member Data Documentation

5.15.3.1 real(kind=8), dimension(:), allocatable tl_ad_integrator::ad_buf_f0 [private]

Buffer to hold tendencies at the initial position of the adjoint model.

Definition at line 31 of file tl_ad_integrator.f90.

5.15.3.2 real(kind=8), dimension(:), allocatable tl_ad_integrator::ad_buf_f1 [private]

Buffer to hold tendencies at the intermediate position of the adjoint model.

Definition at line 32 of file tl_ad_integrator.f90.

5.15.3.3 real(kind=8), dimension(:), allocatable tl_ad_integrator::ad_buf_y1 [private]

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

Definition at line 30 of file tl_ad_integrator.f90.

5.15.3.4 real(kind=8), dimension(:), allocatable tl_ad_integrator::tl_buf_f0 [private]

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

Definition at line 35 of file tl_ad_integrator.f90.

5.15.3.5 real(kind=8), dimension(:), allocatable tl_ad_integrator::tl_buf_f1 [private]

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

Definition at line 36 of file tl_ad_integrator.f90.

5.15.3.6 real(kind=8), dimension(:), allocatable tl_ad_integrator::tl_buf_y1 [private]

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

Definition at line 34 of file tl_ad_integrator.f90.

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

- tl_ad_integrator.f90

5.16 util Module Reference

Utility module.

Public Member Functions

- 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.

5.16.1 Detailed Description

Utility module.

Copyright

2015 Lesley De Cruz & Jonathan Demaeyer. See LICENSE.txt for license information.

Definition at line 13 of file util.f90.

5.16.2 Member Function/Subroutine Documentation

5.16.2.1 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 35 of file util.f90.

5.16.2.2 CHARACTER(len=20) function, public util::str (integer, intent(in) *k*)

Convert an integer to string.

Definition at line 28 of file util.f90.

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

- util.f90

Chapter 6

File Documentation

6.1 aotensor_def.f90 File Reference

Data Types

- 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.

6.2 doc/gen_doc.md File Reference

6.3 doc/tl_ad_doc.md File Reference

6.4 ic_def.f90 File Reference

Data Types

- module ic_def

  Module to load the initial condition.

6.5 inprod_analytic.f90 File Reference

Data Types

- 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.
- 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.

6.6 integrator.f90 File Reference

Data Types

- module integrator

  Module with the integration routines.

6.7 LICENSE.txt File Reference

Functions

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

Variables

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

obtaining a copy of this software and associated documentation to deal in the Software without including without limitation the rights to merge

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

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

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

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

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

- The MIT free of to any person

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

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

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

- The MIT free of to any person

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

6.7.1 Function Documentation

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

6.7.1.2 The MIT License (MIT)

6.7.2 Variable Documentation

6.7.2.1 The MIT free of charge

Definition at line 6 of file LICENSE.txt.

6.7.2.2 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.

6.7.2.3 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.

6.7.2.4 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.

6.7.2.5 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.

6.7.2.6 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.

6.7.2.7 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.

6.7.2.8 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.

6.7.2.9 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.

6.7.2.10 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.

6.7.2.11 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.

6.7.2.12 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.

6.7.2.13 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.

6.7.2.14 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.

6.7.2.15 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.

6.7.2.16 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.

6.7.2.17 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.

6.7.2.18 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.

6.7.2.19 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.

6.7.2.20 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.

6.8 maooam.f90 File Reference

Functions/Subroutines

- program maooam

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

6.8.1 Function/Subroutine Documentation

6.8.1.1 program maooam ( )

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

Copyright

Definition at line 13 of file maooam.f90.

6.9 maooam_tl_ad.f90 File Reference

Data Types

- module maooam_tl_ad

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

6.10 params.f90 File Reference

Data Types

- module params

  The model parameters module.

6.11 stat.f90 File Reference

Data Types

- module stat

  Statistics accumulators.

6.12 tensor.f90 File Reference

Data Types

- module tensor

  Tensor utility module.
- type tensor::coolist_elem

  Coordinate list element type. Elementary elements of the sparse tensors.
- type tensor::coolist

  Coordinate list. Type used to represent the sparse tensor.

6.13 test_aotensor.f90 File Reference

Functions/Subroutines

- program test_aotensor

  Small program to print the inner products.

6.13.1 Function/Subroutine Documentation

6.13.1.1 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.

6.14 test_inprod_analytic.f90 File Reference

Functions/Subroutines

- program inprod_analytic_test

  Small program to print the inner products.

6.14.1 Function/Subroutine Documentation

6.14.1.1 program inprod_analytic_test ( )

Small program to print the inner products.

Copyright

2015 Lesley De Cruz & Jonathan Demaeyer. See LICENSE.txt for license information.

Definition at line 13 of file test_inprod_analytic.f90.

6.15 test_tl_ad.f90 File Reference

Functions/Subroutines

- program test_tl_ad

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

6.15.1 Function/Subroutine Documentation

6.15.1.1 real(kind=8) function gasdev (integer *idum*)

Definition at line 149 of file test_tl_ad.f90.

6.15.1.2 real(kind=8) function ran2 (integer *idum*)

Definition at line 174 of file test_tl_ad.f90.

6.15.1.3 program test_tl_ad ( )

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

Copyright

Definition at line 14 of file test_tl_ad.f90.

6.16 tl_ad_integrator.f90 File Reference

Data Types

- module tl_ad_integrator

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

6.17 util.f90 File Reference

Data Types

- module util

  Utility module.

Index