maooam Documentation

Release 1

Maxime Tondeur

Sep 06, 2016

CONTENTS

Contents:

CHAPTER

ONE

INTRODUCTION

Welcome to MAAOAM python documentation

CHAPTER

TWO

MAOOAM'S PARAMETERS

This module defines the parameters for the model.

Note: The python code is available here : params2.py and on Git.

Example

```
>>> from params2 import ndim,natm,noc
>>> from params2 import oms,nboc,ams,nbatm
>>> from params2 import *
```

There are three types of parameters :

- integral parameters : simulation time (transient and effective), time step, writeout and write step time
- dimensional parameters : dimensions of the truncation of fourier for the atmosphere and the ocean
- physical parameters : they are used in the tensor for the integration

2.1 Integral parameters

Warning: Time is adimensional. If t_real is in seconds, then t_model = t_real * f_0 where f_0 is the Coriolis parameter at 45 degrees latitude (1.032e-4)

- t_trans
- t_run
- dt
- writeout
- tw

2.2 Dimensional parameters

- oms and ams
- nboc and natm
- natm and noc

- ndim

The matrices OMS and AMS gives the possible values of the modes Nx and Ny. It is computed with nboc and natm the numbers of atmospheric and oceanic blocs. natm and noc are the number of functions available. ndim is the total dimension of the system.

Example

```
>>> oms =get_modes(2,4)# ocean mode selection
>>> ams =get_modes(2,2)# atmosphere mode selection
>>> nboc,nbatm = 2*4,2*2          # number of blocks
>>> (natm,noc,ndim)=init_params(nboc,nbatm)
>>>
>>> # Oceanic blocs
>>> #( x block number accounts for half-integer wavenumber e.g 1 => 1/2 , 2 => 1, etc.
↪..)
>>> OMS[0,:] = 1,1
>>> OMS[1,:] = 1,2
>>> OMS[2,:] = 1,3
>>> OMS[3,:] = 1,4
>>> OMS[4,:] = 2,1
>>> OMS[5,:] = 2,2
>>> OMS[6,:] = 2,3
>>> OMS[7,:] = 2,4
>>> #Atmospheric blocs
>>> AMS[0,:] = 1,1
>>> AMS[1,:] = 1,2
>>> AMS[2,:] = 2,1
>>> AMS[3,:] = 2,2
```

2.3 Physical parameters

Scale parameters, ocean's and atmosphere parameters, temperature-related ocean's and atmosphere's parameters other constants, coupling parameters

2.4 Dependancies

```
>>> import numpy as np
```

2.5 Fonctions

Here are the functions to generate the parameters.

params2.**get_modes**(*nxmax*, *nymax*)
: Computes the matrix oms and ams with nxmax and nymax

params2.**init_params**(*nboc*, *nbatm*)
: Computes the dimensions of the system

CHAPTER

THREE

INITIAL CONDITIONS GENERATOR MODULE

This module generates initial conditions for the model if it doesn't exist with the good dimensions.

It is possible to change the dimensions of the system in params.py the parameters file. Then delete ic.py and ic_def.py will regenerates it.

Note: The python code is available here : ic_def.py and on Git.

Example

```
>>> from ic_def import load_IC
>>> load_IC()
```

3.1 Global file

the file ic.py in the same directory.

3.2 Dependancies

uses the modules to know the dimensions :

```
>>> from params2 import natm,noc,ndim,t_trans,t_run
>>> from inprod_analytic import awavenum,owavenum,init_inprod
```

3.3 Functions

Here is the function in the module :

ic_def.**load_IC**()

Check if ic.py exists, if not creates it with good dimensions and zero initial conditions

CHAPTER FOUR

INITIAL CONDITIONS MODULE

This file defines the initial conditions of the model. To be deleted if the dimensions are changed.

Note: The Code is available on Git.

Example

```
>>> from ic import X1
>>> from ic import X0
```

4.1 Global variables (state vectors)

- X1 ic computed with python after 1000 years with step time of 0.1 and GRL set of parameters
- X0 random (non-null) initial conditions.

4.2 Dependencies

```
>>> import numpy as np
```

CHAPTER

FIVE

INNER PRODUCTS MODULE

Inner products between the truncated set of basis functions for the ocean and atmosphere streamfunction fields.

Note: These are calculated using the analytical expressions from De Cruz, L., Demaeyer, J. and Vannitsem, S.: A modular arbitrary-order ocean-atmosphere model: MAOOAM v1.0, Geosci. Model Dev. Discuss. And the Fortran Code

Note: The python code is available here : inprod_analytic.py and on Git.

Example

```
>>> from inprod_analytic import init_inprod
>>> init_inprod()
```

5.1 Global variables

```
>>> awavenum=np.empty(natm, dtype=object)
>>> owavenum=np.empty(noc, dtype=object)
>>> atmos=atm_tensors(natm)
>>> ocean=ocean_tensors(noc)
```

5.2 Dependancies

it uses the modules :

```
>>> from param2 import nbatm, nboc, natm,noc,n,oms,ams,pi
```

5.3 Classes

- atm_wavenum(typ,P,N,H,Nx,Ny)
- ocean_wavenum(P,H,Nx,Ny)
- atm_tensors(natm)

- ocean_tensors(noc)

class `inprod_analytic.`**`atm_wavenum`**(*typ*, *P*, *M*, *H*, *Nx*, *Ny*)

Class to define atmosphere wavenumbers.

Attributes :

- typ (char) = 'A','K' or 'L'.
- M (int)
- P (int)
- H (int)
- Nx (int)
- Ny (int)

class `inprod_analytic.`**`ocean_wavenum`**(*P*, *H*, *Nx*, *Ny*)

Class to define ocean wavenumbers

Attributes :

- P (int)
- H (int)
- Nx (int)
- Ny (int)

class `inprod_analytic.`**`atm_tensors`**(*natm*)

Class which contains all the coefficients a,c,d,s,b,g needed for the tensor computation :

Attributes :

- $a_{i,j}$
- $c_{i,j}$
- $d_{i,j}$
- $s_{i,j}$
- $b_{i,j,k}$
- $g_{i,j,k}$

Return :

- The object will be name *atmos*.

`calculate_a`()

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

`calculate_b`()

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

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

calculate_c_atm()

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

Note: Beta term for the atmosphere Strict function !! Only accepts KL type. For any other combination, it will not calculate anything.

calculate_d(*ocean*)

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

Note: Forcing of the ocean on the atmosphere. Atmospheric s tensor and oceanic M tensor must be computed before calling this routine !

calculate_g()

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

Note: This is a strict function: it only accepts AKL KKL and LLL types. For any other combination, it will not calculate anything.

calculate_s()

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

Note: Forcing (thermal) of the ocean on the atmosphere.

class inprod_analytic.**ocean_tensors**(*noc*)

Class which contains all the coefficients k,m,n,w,o,c needed for the tensor computation :

Attributes :

- $K_{i,j}$
- $M_{i,j}$
- $N_{i,j}$
- $W_{i,j}$
- $O_{i,j,k}$
- $C_{i,j,k}$

Return :

- The object will be name ocean

calculate_K(*atmos*)
Forcing of the atmosphere on the ocean.

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

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

calculate_M()
Forcing of the ocean fields on the ocean.

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

calculate_N()
Beta term for the ocean

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

calculate_O()
Temperature advection term (passive scalar)

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

calculate_C_oc()

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

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

calculate_W(*atmos*)
Short-wave radiative forcing of the ocean.

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

Note: atmospheric s tensor must be computed before calling this function !

inprod_analytic.**init_inprod**()
creates and computes the inner products.

CHAPTER

SIX

TENSOR COMPUTATION MODULE

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.

Note: These are calculated using the analytical expressions from De Cruz, L., Demaeyer, J. and Vannitsem, S.: A modular arbitrary-order ocean-atmosphere model: MAOOAM v1.0, Geosci. Model Dev. Discuss. And the Fortran Code

Note: The python code is available here : aotensor.py and on Git.

Example

```
>>> aotensor=aotensor.init_aotensor()
```

6.1 Help Functions

There are ndim coordinates that correspond to 4 physical quantities. These functions help to have the i-th coordinates of the quantity.

- psi(i) -> i
- theta(i) -> i + natm
- A(i) -> i + 2*natm
- T(i) -> i + 2*natm + noc
- kdelta(i,j) -> (i==j)

6.2 Global variables

- real_eps = 2.2204460492503131e-16
- t=np.zeros(((ndim+1),(ndim+1),(ndim+1)),dtype=float)

6.3 Dependancies

```
>>> from params2 import *
>>> from inprod_analytic import *
>>> from scipy.sparse import dok_matrix
>>> from scipy.sparse import csr_matrix
>>> import os
```

6.4 Functions

- compute_aotensor
- coeff(i,j,k,v)
- simplify
- init_aotensor

`aotensor.`**`compute_aotensor`**`()`

Computes the three-dimensional tensor t

This takes the inner products of inprod_analytic and computes the tensor

Parameters **t** (*array((37,37,37),float)*) – tensor t is a global variable of aotensor

Returns change the global tensor

Return type void

Example

```
>>> compute_aotensor()
```

Warning: This needs the global variable aotensor and the global inner products to be initialized.

Todo

Correct the line with sc (no impact for now)

`aotensor.`**`coeff`**`(i, j, k, v)`

Affects v for $t_{i,j,k}$ making that tensor[i] upper triangular. Used in compute_aotensor.

Parameters

- **i** (*int in [1,37]*) – first coordinates
- **j** (*int in [1,37]*) – second coodinates
- **k** (*int in [1,37]*) – third coordinates
- **v** (*float*) – value

Returns change the global tensor

Return type void

Example

```
>>> coeff(i,j,k,v)
```

aotensor.**simplify**()

Make sure that tensor[i] is upper triangular. To do after compute_aotensor().

Parameters **t** (*array((37,37,37),float)*) – tensor t is a global variable of aotensor

Returns change the global tensor

Return type void

Example

```
>>> simplify()
```

aotensor.**change_structure**()

Take the 3-dimensional heavy tensor t and return aotensor a list of (i,j,k,v).

Parameters **t** (*array((37,37,37),float)*) – tensor t is a global variable of aotensor

Returns change the global tensor

Return type void

Example

```
>>> change_structure()
```

CHAPTER

SEVEN

INTEGRATION MODULE

Module with the integration This module actually contains the Heun algorithm routines.

Note: The python code is available here : https://github.com/nansencenter/DAPPER/tree/max1/mods/MAOOAM

Example

```
>>> from integrator import step
>>> step(y,t,dt)
```

7.1 Global variable

aotensor

7.2 Dependencies

```
>>> from mods.MAOOAM.params2 import ndim
>>> import mods.MAOOAM.aotensor as aotensor
>>> import numpy as np
>>> from scipy.sparse import dok_matrix
>>> from scipy.sparse import csr_matrix
>>> import time
```

7.3 Functions

- sparse_mul3
- tendencies
- step

integrator.**sparse_mul3**(*arr*)
: Calculate for each i the sums on j,k of the product tensor(i,j,k)* arr(j) * arr(k)

integrator.**tendencies**(*y*)
: Calculate the tendencies thanks to the product of the tensor and the vector y

`integrator.`**`step`**`(`*y*, *t*, *dt*`)`

Heun method integration

CHAPTER

EIGHT

PRINCIPAL MODULE

Python 3.5 implementation of the modular arbitrary-order ocean-atmosphere model MAOOAM This module actually contains the Heun algorithm routines.

Note: The python code is available here : https://github.com/nansencenter/DAPPER/tree/max1/mods/MAOOAM

Example

>>>from maooam import *

8.1 Global variable

ic.X0 ic.X1 X t t_trans,t_run,tw,dt T

8.2 Dependencies

import numpy as np import params2 from params2 import ndim,tw,t_run,t_trans,dt import aotensor import integrator from plot import * import time import ic_def import ic

CHAPTER NINE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

i

m

p

A

C

G

I

L

M

O

P

S

T