MAOOAM Documentation

Release

Maxime Tondeur, Jonathan Demayer

February 19, 2018

CONTENTS

CHAPTER

ONE

SYNOPSIS

This repository provides the code of the model MAOOAM in python. It is a low-order ocean-atmosphere model with an arbitrary expansion of the Fourier modes of temperatures and streamfunctions. The code in Python is a translation of the Fortran code available in the main Git repository : https://github.com/Climdyn/MAOOAM.

CHAPTER TWO

MOTIVATION

The code has been translated in Python to be used with the Data Assimilation module DAPPER : https://github.com/nansencenter/DAPPER .

CHAPTER

THREE

INSTALLATION

The program can be run with python 2.7 or 3.5. Please note that python 3.5 is needed by the Data Assimilation module DAPPER. Optionally, F2py is also needed to compile the optimized fortran part.

To install, unpack the archive in a folder or clone with git:

```
>>> git clone https://github.com/Climdyn/MAOOAM.git
```

and run f2py (optional):

```
>>> f2py -c sparse_mult.pyf sparse_mult.f90
```

CHAPTER

FOUR

GETTING STARTED

The user first has to fill the params_maooam.py according to their needs. See its documentation for more information about this file. Some examples related to already published articles are available in the params folder.

Finally, the ic.py 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.py file (if it exists) and run the program:

```
>>> ipython maooam.py
```

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:

```
>>> ipython maooam.py
```

The code will generate a file evol_field.dat containing the recorded time evolution of the variables.

CHAPTER

FIVE

DESCRIPTION OF THE FILES

- maooam.py : main program.
- params_maooam.py : a module for the parameters of the model (dimensional, integral and physical).
- ic.py : initial conditions file.
- ic_def.py a module that generate the initial conditions if it does not exist.
- inprod_analytic.py : a module that compute the inner product needed for the tensor computation.
- aotensor.py : a module that compute the tensor.
- integrator.py: a module that compute one step of the model and RK2 integration.
- sparse_mult.f90 and sparse_mult.pyf : fortran and f2py files to call fortran module in python.

CHAPTER

SIX

CONTENTS

Parameters module

This module defines the parameters for the model.

Note: The python code is available here : params_maooam.py .

Example

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

There are three types of parameters :

- integration 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

Integration 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** : the transient simulation time of the model to be on the attractor. The states vectors are not written on evol_field.dat.
- **t_run** : the running simulation time of the model. The states vectors are written on evol_field.dat every tw.
- **dt** : the step time.
- **writeout** : boolean value to decide if the module produces evol_field.dat.
- **tw** : the step time to write on evol_field.
- **f2py** : boolean to activate the f2py optimization.

Dimensional parameters

- **oms** and **ams** : the matrices that gives the possible values of the modes Nx and Ny.

- **nboc** and **natm** : the numbers of oceanic and atmospheric blocs.
- **natm** and **noc** : the numbers of functions available.
- **ndim** : the total dimension.

  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
```

  Typical dimensional parameters

- atmosphere 2x,2y ; ocean 2x,4y ; ndim = 36
- atmosphere 2x,2y ; ocean 4x,4y ; ndim = 52
- atmosphere 2x,4y ; ocean 2x,4y ; ndim = 56

Physical parameters

Some defaut parameters are presented below. Some parameters files related to already published article are available in the params folder.

Scale parameters

- **scale = 5.e6** : characteristic space scale, L*pi
- **f0 = 1.032e-4** : Coriolis parameter at 45 degrees latitude
- **n = 1.5e0** : aspect ratio (n = 2Ly/Lx ; Lx = 2*pi*L/n; Ly = pi*L)
- **rra = 6370.e3** : earth radius
- **phi0_npi = 0.25e0** : latitude exprimed in fraction of pi

Parameters for the ocean

- **gp = 3.1e-2** : reduced gravity
- **r = 1.e-8** : frictional coefficient at the bottom of the ocean

- **h = 5.e2** : depth of the water layer of the ocean
- **d = 1.e-8** : the coupling parameter (should be divided by f0 to be adim)

Parameters for the atmosphere

- **k = 0.02** : atmosphere bottom friction coefficient
- **kp = 0.04** : atmosphere internal friction coefficient
- **sig0 = 0.1e0** : static stability of the atmosphere

Temperature-related parameters for the ocean

- **Go = 2.e8** : Specific heat capacity of the ocean (50m layer)
- **Co = 350** : Constant short-wave radiation of the ocean
- **To0 = 285.0** : Stationary solution for the 0-th order ocean temperature

Temperature-related parameters for the atmosphere

- **Ga = 1.e7** : Specific heat capacity of the atmosphere
- **Ca = 100.e0** ; Constant short-wave radiation of the atmosphere
- **epsa = 0.76e0** : Emissivity coefficient for the grey-body atmosphere
- **Ta0 = 270.0** : Stationary solution for the 0-th order atmospheric temperature

Other temperature-related parameters/constants

- **sc = 1.** : Ratio of surface to atmosphere temperature
- **lambda = 20.00** : Sensible+turbulent heat exchange between oc and atm
- **rr = 287.e0** : Gas constant of dry air
- **sb = 5.6e-8** : Stefan-Boltzmann constant

Key values

- **k** is the friction coefficient at the bottom of the atmosphere. Typical values are 0.01 or 0.0145 for chaotic regimes.
- **kp** is the internal friction between the atmosphere layers. kp=2*k
- **d** is the friction coefficient between the ocean and the atmosphere. Typical values are 6*10^{-8} s^{-1} or 9*10^{-8} s^{-1}.
- **lambda** is the heat exchange between the ocean and the atmosphere. Typical values are 10 W m^{-2} K^{-1} or 15.06 W m ^{-2} K^{-1}.

Dependencies

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

Fonctions

Here are the functions to generate the parameters.

`params_maooam.`**`get_modes`**`(`*nxmax*, *nymax*`)`
    Computes the matrix oms and ams with nxmax and nymax

`params_maooam.`**`init_params`**`(`*nboc*, *nbatm*`)`
    Computes the dimensions of the system

Initial conditions generator module

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

The dimensions of the system can be changed in the parameters file params_maooam.py . Then delete ic.py and ic_def.py will regenerates it.

Note: The python code is available here : ic_def.py .

Example

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

Global file

The file ic.py in the same directory.

Dependencies

Uses the following modules to know the dimensions :

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

Functions

Functions in the module :

`ic_def.`**`load_IC`**`()`
    Check if ic.py exists, if not creates it with good dimensions and zero initial conditions

Initial conditions file

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

Example

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

Global variables (state vectors)

- X0 random (non-null) initial conditions.

Dependencies

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

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 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.

Note: The python code is available here : inprod_analytic.py .

Example

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

Global variables

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

Dependencies

it uses the modules :

```
>>> import numpy as np
>>> from scipy.sparse import csr_matrix
>>> from params_maooam import nbatm
>>> from params_maooam import nboc
>>> from params_maooam import natm
>>> from params_maooam import noc
>>> from params_maooam import n
>>> from params_maooam import oms
>>> from params_maooam import ams
>>> from params_maooam import pi
```

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_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``()`

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

Note: Eigenvalues of the Laplacian (atmospheric)

`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.`**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)

`inprod_analytic.`**init_inprod**()

creates and computes the inner products.

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_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_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_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 !

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

Class to define ocean wavenumbers

Attributes :

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

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 .

Example

```
>>> aotensor, Li, Lj, Lk, Lv =aotensor.init_aotensor()
```

Help Functions

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

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

Global variables

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

Dependencies

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

Functions

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

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

Computes the three-dimensional tensor t

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: Needs the global variable aotensor and the global inner products to be initialized.

`aotensor.`**`init_aotensor`**()

Initialize the tensor.

Returns aotensor, Li, Lj, Lk, Lv

Example

```
>>> aotensor, Li, Lj, Lk, Lv = init_aotensor()
```

`aotensor.simplify()`

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

Parameters **t** (*array((ndim+1,ndim+1,ndim+1),float)*) – tensor t is a global variable of aotensor

Returns change the global tensor

Return type void

Example

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

Integration module

This module actually contains the Heun algorithm routines.

Note: The python code is available here : integrator.py .

Example

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

Global variables

- **aotensor** tensor with the format (int i, int j, int k, float v) in list
- **Li** first list of index of tensor
- **Lj** second list of index of tensor
- **Lk** third list of index of tensor
- **Lv** list of tensor values

Dependencies

```
>>> import numpy as np
>>> from params_maooam import ndim,f2py
>>> import aotensor as aotensor_mod
>>> if f2py:
>>>     import sparse_mult as mult
>>>     sparse_mul3_f2py = mult.sparse_mult.sparse_mul3
```

Functions

- sparse_mul3
- tendencies
- step

`integrator.sparse_mul3(arr)`

Calculate for each i the sums on j,k of the product

$$tensor(i, j, k) * arr(j) * arr(k)$$

Note: Python-only function

`integrator.sparse_mul3_py(arr)`

Calculate for each i the sums on j,k of the product

$$tensor(i, j, k) * arr(j) * arr(k)$$

Note: Python-only function

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

RK2 method integration

`integrator.tendencies(y)`

Calculate the tendencies thanks to the product of the tensor and the vector y

Principal module

Python implementation of the Modular Arbitrary-Order Ocean-Atmosphere Model MAOOAM

Note: The python code is available here : maooam.py .

Example

```
>>> from maooam import *
```

Global variable

- **ic.X0** : initial conditions
- **X** : live step vector
- **t** : time
- **t_trans**, **t_run** : respectively transient and running time
- **dt** : step time
- **tw** : step time for writing on evol_field.dat

Dependencies

```
>>> import numpy as np
>>> import params_maooam
>>> from params_maooam import ndim,tw,t_run,t_trans,dt
>>> import aotensor
>>> import time
>>> import ic_def
>>> import ic
>>> import sys
```

class `maooam.`**`bcolors`**

to color the instructions in the console

CHAPTER

SEVEN

CONTRIBUTORS

Maxime Tondeur, Jonathan Demaeyer

CHAPTER

EIGHT

LICENSE

CHAPTER NINE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX