INSTITUT FÜR INFORMATIK

Temporal Coarsening of Transport Matrices with Metos3D

Piwonski J., Slawig T.

Bericht Nr. 1608
November 2016
ISSN 2192-6247

CHRISTIAN-ALBRECHTS-UNIVERSITÄT

ZU KIEL

Institut für Informatik der
Christian-Albrechts-Universität zu Kiel
Olshausenstr. 40
D – 24098 Kiel

Temporal Coarsening of Transport Matrices with Metos3D

Piwonski J., Slawig T.

Bericht Nr. 1608
November 2016
ISSN 2192-6247

e-mail: {jpi,ts}@informatik.uni-kiel.de

This report describes the technique of temporal coarsening of transport matrices used by Metos3D (`https://github.com/metos3d`).

Temporal Coarsening of Transport Matrices with Metos3D

J. Piwonski and T. Slawig
{jpi,ts}@informatik.uni-kiel.de

November 8, 2016

Abstract

Transport matrices represent pre-computed ocean circulation that can be used to efficiently approximate the transport of passive tracers in ocean waters. Metos3D (`https://github.com/metos3d`) is a software toolkit used for the computation of periodic steady states of biogeochemical and marine ecosystem models. Metos3D makes use of transport matrices.

One major disadvantage of transport matrices is the fact that the used time step is hardcoded into each matrix. However, there is a possibility to alter the time step afterwards to some extent. We built in such a conversion technique into the `metos3d` script using SciPy (`https://www.scipy.org/`). This technical report explains the details of the `matrix` command and its usage.

1 Introduction

In the field of climate research simulations of marine ecosystem models are used to investigate the carbon uptake and storage of earth's oceans. The aim is to identify those processes that play a role in the global carbon cycle (cf. Fasham, 2003; Sarmiento and Gruber, 2006). However, descriptions of biogeochemical or ecological sinks and sources still contain uncertainties with regard to the number of components and to parameterization (cf. Kriest et al., 2010).

To improve this situation a wide range of marine ecosystem models need to be validated, i.e. assessed as to their ability to reproduce real world data. This involves a thorough discussion of simulation results and, before this, an estimation of optimal model parameters for preferably standardized data sets (cf. Fennel et al., 2001; Schartau and Oschlies, 2003). As a rule hundreds of model evaluations are required for optimization.

For any fully coupled simulation, i.e. simultaneous and interdependent computations of ocean circulation, tracer transport and the biogeochemical sources and sinks in three spatial dimensions, very high computational efforts are needed even at low resolution. Computational complexity increases still more if annual cycles are investigated, since each model evaluation then involves long-time integration (the so-called spin-up) until an equilibrium state is reached under given forcing (cf. Bernsen et al., 2008).

Off-line simulation affords fundamentally reduced computational costs combined with an acceptable loss of accuracy. The principle is to precompute transport data for passive tracers. This approach was adopted by Khatiwala et al. (2005) when introducing the so-called transport matrix method. The authors used matrices to store the results of a general circulation model, which were then applied to biogeochemical tracer variables.

The Marine Ecosystem Toolkit for Optimization and Simulation (Piwonski and Slawig, 2016, Metos3D) makes use of transport matrices. It offers a default (matrix) data set with a fixed, hard coded time step of 3 hours, see `https://github.com/metos3d/data`. Until now, this prescribed time step was the only one available for ocean circulation. In this work we present the implementation of a technique that allows temporal coarsening of transport matrices to some extent. Moreover, we show how to use the `metos3d` script to convert given transport matrices to coarser time steps. Our approach basically follows the description in (Khatiwala, 2007).

In Section 2 and 3 we shortly recapitulate the underlying transport equations and the used transport matrices, respectively. In Section 4 we explain the temporal coarsening. Details of the implementation are discussed in Section 5. Finally, the usage of the `matrix` command is shown in Section 6.

2 Transport equations

We consider the following tracer transport model, which is defined by a system of semilinear parabolic partial differential equations (PDEs) of the

form

$$\frac{\partial y_i}{\partial t} = \nabla \cdot (\kappa \nabla y_i) - \nabla \cdot (v\, y_i) + q_i(y, u, b, d),\ i = 1, \ldots, n_y, \tag{1}$$

on a time interval $I := [0, T]$ and a spatial domain $\Omega \subset \mathbb{R}^3$ with boundary $\Gamma = \partial\Omega$. $y_i : I \times \Omega \to \mathbb{R}$ denotes a single tracer concentration, and $y = (y_i)_{i=1}^{n_y}$ is the vector of all tracers.

The transport of tracers in marine waters is determined by diffusion and advection, which are reflected in the first two linear terms on the right-hand side of (1). Diffusion mixing coefficient $\kappa : I \times \Omega \to \mathbb{R}$ and advection velocity field $v : I \times \Omega \to \mathbb{R}^3$ may either be regarded as given data, or else have to be simulated by an ocean model along with (1). Molecular diffusion of tracers is regarded as negligible compared to turbulent mixing diffusion. Thus κ and both transport terms are the same for all y_i.

Biogeochemical processes within the ecosystem are represented by the last term on the right-hand side of (1), i.e.

$$q_i(y, u, b, d) = q_i(y_1, \ldots, y_n, u, b, d), \quad i = 1, \ldots, n_y.$$

The functions represented by q_i will often be nonlinear and depend on several tracers, thereby coupling the system. However, since biogeochemical and marine ecosystem models are not the focus of this report they are not discussed any further here. We refer the reader to Piwonski and Slawig (2016) for more details.

3 Transport matrices

We introduce the following notation: Let the domain Ω be discretized by a grid $(\mathbf{x}_k)_{k=1}^{n_x} \subset \mathbb{R}^3$ and one year in time by $0 = t_0 < \ldots < t_j < t_j + \Delta t_j =: t_{j+1} < \ldots < t_{n_t} = 1$. This means that there are n_t time steps per year. For time instant t_j, $\mathbf{y}_j = (\mathbf{y}_{ji})_{i=1}^{n_y} \in \mathbb{R}^{n_y n_x}$ denotes a vector of the values of *all* tracers at all grid points, appropriately concatenated. We use analogous notations $\mathbf{b}_j, \mathbf{d}_j$, and $\mathbf{q}_j$ for boundary and domain data and for the biogeochemical terms at the j-th time step.

The transport matrix method (Khatiwala et al., 2005) allows fast simulation of tracer transport, assuming that forcing data diffusivity κ and advection velocity v are given. It approximates the discretized counterpart of

(1) by

$$\begin{aligned}\mathbf{y}_{j+1} &= \mathbf{A}'_{imp,j}(\mathbf{A}'_{exp,j}\mathbf{y}_j + \Delta t_j \mathbf{q}_j(\mathbf{y}_j, \mathbf{u}, \mathbf{b}_j, \mathbf{d}_j)) \\ &=: \varphi_j(\mathbf{y}_j, \mathbf{u}, \mathbf{b}_j, \mathbf{d}_j), \quad j = 0, \ldots, n_t - 1.\end{aligned} \tag{2}$$

Since transport affects each tracer individually and is identical for all of them, both $\mathbf{A}'_{exp,j}, \mathbf{A}'_{imp,j}$ are block-diagonal matrices with n_y identical blocks $\mathbf{A}_{exp,j}, \mathbf{A}_{imp,j} \in \mathbb{R}^{n_x \times n_x}$, respectively.

The distinction between implicit and explicit matrices originates in an operator splitting scheme used in the ocean model. Diffusion (or some part of it) is usually discretized implicitly; in our case this applies only to vertical diffusion.

To reduce storing efforts and increase feasibility only a small number of averaged matrices are stored; in our case monthly averages are used.

4 Temporal coarsening

Formally, an *explicit* matrix represents the whole explicit time step, i.e.

$$\mathbf{A}_{exp,j} = (\mathbf{I} + \Delta t \mathbf{L}_{exp,j})\,.$$

In contrast, an *implicit* matrix is the inverse of the step back in time, i.e.

$$\mathbf{A}_{imp,j} = (\mathbf{I} - \Delta t \mathbf{L}_{imp,j})^{-1}\,.$$

Here, $\mathbf{L}_{exp,j}$ and $\mathbf{L}_{imp,j}$ represent those parts of the transport term in (1) that are discretized explicitly or implicitly w. r. t. time.

4.1 Explicit matrix

The aim is to construct an explicit matrix

$$\mathbf{A}_{exp,m} = (\mathbf{I} + m\Delta t \mathbf{L}_{exp})$$

with an m times coarser time step than a given

$$\mathbf{A}_{exp} = (\mathbf{I} + \Delta t \mathbf{L}_{exp})\,.$$

Note that we omitted the time index j for clarity. Moreover, we assume that m is an integer.

Starting with $\mathbf{A}_{exp}$, we successively rearrange the definition and end up with:

$$\begin{aligned}
\mathbf{A}_{exp} &:= \mathbf{I} + \Delta t \mathbf{L}_{exp} \\
\mathbf{A}_{exp} - \mathbf{I} &= \Delta t \mathbf{L}_{exp} && | - \mathbf{I} \\
m(\mathbf{A}_{exp} - \mathbf{I}) &= m \Delta t \mathbf{L}_{exp} && | \cdot m \\
\mathbf{I} + m(\mathbf{A}_{exp} - \mathbf{I}) &= \mathbf{I} + m \Delta t \mathbf{L}_{exp} && | + \mathbf{I} \\
\mathbf{I} + m(\mathbf{A}_{exp} - \mathbf{I}) &=: \mathbf{A}_{exp,m}
\end{aligned}$$

Hence, regarding an *explicit* matrix, we need to subtract the identity, scale the resulting matrix with the coarsening factor and add the identity back to the matrix.

4.2 Implicit matrix

Formally, we would like to construct

$$\mathbf{A}_{imp,m} = (\mathbf{I} - m \Delta t \mathbf{L}_{imp})^{-1}$$

from a given

$$\mathbf{A}_{imp} = (\mathbf{I} - \Delta t \mathbf{L}_{imp})^{-1} .$$

However, the implicit matrix in an inverse. We won't be able to disassemble it so easy. Instead, we compute the m-th power of $\mathbf{A}_{imp}$, i.e.

$$(\mathbf{A}_{imp})^m = ((\mathbf{I} - \Delta t \mathbf{L}_{imp})^{-1})^m$$

or

$$(\mathbf{A}_{imp})^m = ((\mathbf{I} - \Delta t \mathbf{L}_{imp})^m)^{-1} ,$$

since all matrices are equal. Then, the inner part can be written as

$$\begin{aligned}
(\mathbf{I} - \Delta t \mathbf{L}_{imp})^m &= \sum_{k=0}^{m} \binom{m}{k} \mathbf{I}^{m-k} (-\Delta t \mathbf{L}_{imp})^k \\
&= \binom{m}{0} \mathbf{I}^m (-\Delta t \mathbf{L}_{imp})^0 + \binom{m}{1} \mathbf{I}^{m-1} (-\Delta t \mathbf{L}_{imp})^1 + \sum_{k=2}^{m} \binom{m}{k} \mathbf{I}^{m-k} (-\Delta t \mathbf{L}_{imp})^k \\
&= \mathbf{I} - m \Delta t \mathbf{L}_{imp} + \sum_{k=2}^{m} \binom{m}{k} \mathbf{I}^{m-k} (-\Delta t \mathbf{L}_{imp})^k \\
&= (\mathbf{I} - m \Delta t \mathbf{L}_{imp}) + \mathcal{O}(\Delta t^2) .
\end{aligned}$$

The approximation is asymptotically correct. All other terms except for $(\mathbf{I} - m\Delta t\mathbf{L}_{imp})$ are of quadratic or higher order in Δt, and thus tend much faster towards zero than Δt. Hence,

$$\mathbf{A}_{imp,m} \approx (\mathbf{A}_{imp})^m .$$

> **Note:** Regarding the standard matrix data set of Metos3D (`https://github.com/metos3d/data`), both explicit and implicit matrices will retain their *sparse structure* after coarsening.
> In general, this won't be the case. A matrix-matrix-multiplication, as used to coarsen the implicit matrix, usually blows up the number of zeros of a sparse matrix. Most importantly, the script does not support a change in sparse structure (yet).

5 Implementation

We assume that all matrix files are stored in the default PETSc CSR matrix format, "also called the Yale sparse matrix format or compressed sparse row format, CSR" (Balay et al., 2012, PETSc User Manual, Section 3.1.1). Note that PESTc always stores binary data in *big-endian* byte order. The format reads as follows[1]:

```
int          MAT_FILE_CLASSID
int          number of rows
int          number of columns
int          total number of nonzeros
int*         number nonzeros in each row
int*         column indices of all nonzeros (starting index is zero)
PetscScalar* values of all nonzeros
```

Here, `PetscScalar` is a `double`, i.e. a 64 bit floating point number and `int` is a 32 bit integer. `MAT_FILE_CLASSID` is a PETSc internal ID for matrices that we won't use.

[1] `http://www.mcs.anl.gov/petsc/petsc-current/docs/manualpages/Mat/MatLoad.html`

We use the SciPy CSR sparse matrix type[2] to perform the matrix operations shown in Section 4. Additionally, we use NumPy, which is part of SciPy, for array handling. Hence, we start with:

```
import numpy as np
from scipy.sparse import csr_matrix, eye
```

We will open each matrix file twice. The first time we read only the first 4 integers, i.e. ID, number of rows, number of columns and number of non-zeros:

```
[id, nrow, ncol, nnz] = np.fromfile(filepathin, dtype = '>i4', count = 4)
```

Assume the path to the matrix file is placed in `filepathin`. The data type is described by `'>i4'`. The `>` denotes big-endian and `i4` denotes a 32 bit (4 byte) integer. We can now construct the whole PETSc matrix data type, i.e.

```
petscaij = '>i4, >i4, >i4, >i4, %d>i4, %d>i4, %d>f8' % (nrow, nnz, nnz)
```

Then, we can read the whole file as an NumPy array:

```
matrixarray = np.fromfile(filepathin, dtype = petscaij)
```

The `fromfile()` routine returns a tuple, at which the first entry is the desired array and the second entry is the data type description.

```
[id, nrow, ncol, nnz, nnzrow, indices, data] = matrixarray[0]
```

Although, both formats are called CSR, there is a difference in how the indices are stored. Whereas PETSc stores the number of non-zeros per row, SciPy uses index pointers, which are the cumulative sum of the former, prepended with a zero. We compute this as follows:

```
indptr = np.insert(np.cumsum(nnzrow), 0, 0)
```

We then construct the matrix, set the coarsening factor and finally, depending on the matrix type, perform the coarsening as described in Section 4:

[2] `https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.sparse.csr_matrix.html`

```
A = csr_matrix((data, indices, indptr), shape=(nrow, ncol))
m = factor
if matrixtype == 'exp':
    I = eye(nrow, format = 'csr')
    Am = I + m * (A - I)
elif matrixtype == 'imp':
    Am = A**m
```

At last, we exchange the computed indices and data in the loaded array and store the whole array to a file with a given `filepathout`, i.e.

```
matrixarray[0][5] = Am.indices
matrixarray[0][6] = Am.data
matrixarray.tofile(filepathout)
```

6 Usage

> **Note:** You must have SciPy (`https://www.scipy.org/`) installed.

The format of the `matrix` subcommand is:

```
metos3d [-v] matrix [exp|imp] [count] [factor] [file-format-in] [file-format-out]
```

A matrix *type* must be specified after the `matrix` command, which is either `exp` or `imp`. Then, a matrix `count` as well as a coarsening `factor` must follow. At last, the user must provide an input and output file format, respectively. In general, as for the other commands, the `-v` switch can be used to force `metos3d` to provide additional debug information while processing the request.

Example:
Let's assume we are using the *default* Metos3D data set with its *default* directory structure at the *default* location. We change into the matrix kind folder, i.e.

```
cd ~/.metos3d/data/data/TMM/2.8/Transport/Matrix5_4/
```

Let's say, we want to create matrices with $m = 2, 4, 8$ to carry out some comparison experiments. We successively type:

```
# create 2dt matrices
mkdir 2dt
metos3d matrix exp 12 2 1dt/Ae_%02d.petsc 2dt/Ae_%02d.petsc
metos3d matrix imp 12 2 1dt/Ai_%02d.petsc 2dt/Ai_%02d.petsc

# create 4dt matrices
mkdir 4dt
metos3d matrix exp 12 4 1dt/Ae_%02d.petsc 4dt/Ae_%02d.petsc
metos3d matrix imp 12 4 1dt/Ai_%02d.petsc 4dt/Ai_%02d.petsc

# create 8dt matrices
mkdir 8dt
metos3d matrix exp 12 8 1dt/Ae_%02d.petsc 8dt/Ae_%02d.petsc
metos3d matrix imp 12 8 1dt/Ai_%02d.petsc 8dt/Ai_%02d.petsc
```

That's it. Have fun.

Note: The *option files* must match the used time step accordingly. Here, the input directory for matrices, the time step and the time step count must be adapted, i.e.

```
# 1dt
-Metos3DTimeStepCount 2880
-Metos3DTimeStep 0.0003472222222222 # 1.0/2880
-Metos3DMatrixInputDirectory data/TMM/.../1dt/

# 2dt
-Metos3DTimeStepCount 1440
-Metos3DTimeStep 0.0006944444444444 # 1.0/1440
-Metos3DMatrixInputDirectory data/TMM/.../2dt/

# 4dt
-Metos3DTimeStepCount 720
-Metos3DTimeStep 0.0013888888888889 # 1.0/720
-Metos3DMatrixInputDirectory data/TMM/.../4dt/

# 8dt
-Metos3DTimeStepCount 360
-Metos3DTimeStep 0.0027777777777778 # 1.0/360
-Metos3DMatrixInputDirectory data/TMM/.../8dt/
```

References

Balay, S., Brown, J., , Buschelman, K., Eijkhout, V., Gropp, W. D., Kaushik, D., Knepley, M. G., McInnes, L. C., Smith, B. F., and Zhang, H.: PETSc Users Manual, Tech. Rep. ANL-95/11 - Revision 3.3, Argonne National Laboratory, Lemont, 2012.

Bernsen, E., Dijkstra, H. A., and Wubs, F. W.: A method to reduce the spin-up time of ocean models, Ocean Modelling, 20, 380 – 392, doi:10.1016/j.ocemod.2007.10.008, 2008.

Fasham, M. J. R., ed.: Ocean Biogeochemistry. The Role of the Ocean Carbon

Cycle in Global Change., Global Change – The IGBP Series, Springer, Berlin et al., 2003.

Fennel, K., Losch, M., Schröter, J., and Wenzel, M.: Testing a marine ecosystem model: sensitivity analysis and parameter optimization, Journal of Marine Systems, 28, 45–63, doi:10.1016/S0924-7963(00)00083-X, 2001.

Khatiwala, S.: A computational framework for simulation of biogeochemical tracers in the ocean, Global Biogeochemical Cycles, 21, doi:10.1029/2007GB002923, 2007.

Khatiwala, S., Visbeck, M., and Cane, M.: Accelerated simulation of passive tracers in ocean circulation models, Ocean Modelling, 9, 51–69, 2005.

Kriest, I., Khatiwala, S., and Oschlies, A.: Towards an assessment of simple global marine biogeochemical models of different complexity, Progress In Oceanography, 86, 337–360, doi:10.1016/j.pocean.2010.05.002, 2010.

Piwonski, J. and Slawig, T.: Metos3D: the Marine Ecosystem Toolkit for Optimization and Simulation in 3-D – Part 1: Simulation Package v0.3.2, Geoscientific Model Development, 9, 3729–3750, doi:10.5194/gmd-9-3729-2016, URL `http://www.geosci-model-dev.net/9/3729/2016/`, 2016.

Sarmiento, J. L. and Gruber, N.: Ocean Biogeochemical Dynamics, Princeton University Press, Princeton et al., 2006.

Schartau, M. and Oschlies, A.: Simultaneous data-based optimization of a 1d-ecosystem model at three locations in the north Atlantic: Part I - method and parameter estimates, Journal of Marine Research 61, pp. 765–793, 2003.