Running Medslik

Fabio Viola

Contents

1 Installing Medslik

1. Download and uncompress the package;

```
$ cd test1
$ wget http://www.medslik-ii.org/data/v2.01/model/MEDSLIK_II_2.01.tar.gz
```

2. Load the proper modules;

```
$ module load gcc_9.1.0/9.1.0
$ module load intel19.5/19.5.281 intel19.5/szip/2.1.1 intel19.5/hdf5/1.10.5
$ module load intel19.5/netcdf/C_4.7.2-F_4.5.2_CXX_4.3.1
```

3. Fix the paths in `MEDSLIK_II_2.01/EXE/source/compile.sh`;

4. Compile;

```
$ cd MEDSLIK_II_2.01/EXE
$ sh source/compile.sh
```

5. Download and uncompress the other files.

```
$ cd ../..
$ wget http://www.medslik-ii.org/data/v2.01/cases/paria_casestudy.tar.gz
$ wget http://www.medslik-ii.org/data/v2.01/scripts_mdk2.tar.gz
$ wget http://www.medslik-ii.org/data/v2.01/Bathy_coast.tar.gz
$ tar xvzf Bathy_coast.tar.gz
$ tar xvzf paria_casestudy.tar.gz
$ tar xvzf scripts_mdk2.tar.gz
```

2 Running Medslik without LSF

1. First of all, assuming that Medslik is already compiled, the content of the main directory is:

```
$ ls
Bathy_coast/                      log/
medslik5.inp                      MEDSLIK_II_2.01/
medslik.tmp                       paria_casestudy/
run.sh                            scripts_mdk2/
set_env.sh
```

2. Now, we load the essential modules:

```
$ module load intel19.5/19.5.281
$ module load intel19.5/netcdf/C_4.7.2-F_4.5.2_CXX_4.3.1
```

3. Copy the input files to the MEDSLIK_II_2.01 installation folder:

```
$ cp oce_files/* MEDSLIK_II_v2.01/DATA/fcst/H3k
$ cp met_files/* MEDSLIK_II_v2.01/DATA/fcst/SK1
$ cp bnc_files/* MEDSLIK_II_v2.01/EXE/data
$ cp xp_files/* MEDSLIK_II_v2.01/EXE
```

4. Run!

```
$ cd MEDSLIK_II_v2.01/EXE
$ ./RUN.sh
```

2.1 Errors

The first run, right after the "STOKE DRIFT CALCULATION", gives the following error:

```
Fortran runtime error: File 'data/medf_.map' does not exist
```

This file should probably come from the paria_casestudy. As seen in the third step of the previous section, we copied the file manually, so let's check the original paria_casestudy folder to see if the file exists or not:

```
$ find ../../paria_casestudy -name medf_.map
```

It returns no results.

Moreover, in the output of the simulation, just before this error, it is possible to read this line:

```
At line 2939 of file /work/opa/witoil-dev/MEDSLIK_II_2.01/EXE/source/medslik_II.
   for (unit = 51, file = '.85044 18.00833 7     0 Q0')
```

If we look around that line of Fortran code:

```
    if(iaencr.eq.0) then
        open(51,file='data/'//regn1//'.map',status='old')
    else
            open(51,file='data/'//regn1//'_.map',status='old')
        endif
```

⚠ ⚠ ⚠ So, probably we really miss **medf_.map** file.

After a quick chat with Svitlana, I copied medf.map to medf_.map. The execution stops now with:

```
At line 2957 of file /work/opa/witoil-dev/MEDSLIK_II_2.01/EXE/source/medslik_II.
   for (unit = 51, file = 'data/medf_.map')
Fortran runtime error: Bad integer for item 1 in list input
```

Other errors follow the one reported above. They are related to missing output files, probably not generated due to the absence of the previous file that caused the simulation to end.
Another missing file producing an error line is initial.txt. Actually, this file is created only if the variable $ContourSlick is equal to "YES", but the file medslik_inputfile.txt is instead configured with "NO".

This attempt to make MEDSLIK run has been stopped. Please follow the next section that uses Svitlana's code.

3 Installation with Svitlana's code

1. First of all, there are three compile files:

   - compile.extract.gfortran.sh
   - compile_for_jday_lat_lon.sh
   - compile_medslikII_for.sh

   I re-wrote the code for these files in a file named compile.fabio.sh:

   ```
   # load modules
   module load intel19.5/19.5.281
   module load intel19.5/netcdf/C_4.7.2-F_4.5.2_CXX_4.3.1
   module load intel19.5/hdf5/1.10.5
   module load intel19.5/szip/2.1.1
   module load curl/7.66.0

   # set folders
   wdir=/work/opa/witoil-dev/test1/meglob
   ```

```
DIR_EXE=${wdir}/EXE
DIR_SRC=${wdir}/EXE/source
NETCDF=/zeus/opt/intel19.5/netcdf

# compile
gfortran -I$NETCDF/C_4.7.2-F_4.5.2_CXX_4.3.1/include -L$NETCDF/C_4.7.2-F_4
   .5.2_CXX_4.3.1/lib  $DIR_SRC/Extract_II.for -lnetcdf -lnetcdff -o
   $DIR_EXE/Extract_II.exe
gfortran -o $DIR_EXE/jday $DIR_SRC/jday.f
gfortran -I$NETCDF/C_4.7.2-F_4.5.2_CXX_4.3.1/include -L$NETCDF/C_4.7.2-F_4
   .5.2_CXX_4.3.1/lib  $DIR_SRC/medslik_II.for -lnetcdf -lnetcdff -o
   $DIR_EXE/medslik_II.exe
gfortran -o $DIR_EXE/lat_lon.exe $DIR_SRC/lat_lon.for
```

2. I was looking for file `medslik_II_ens.sh` but is no longer present. It was replaced by `medslik_II.sh`. Then, I modified the variables containing the Medslik directory there and in `RUN.sh`.

3. I loaded the following modules:

```
$ module load intel19.5/19.5.281
$ module load intel19.5/netcdf/C_4.7.2-F_4.5.2_CXX_4.3.1
```

4. Then we are able to run the simulation! Here, part of the output is reported:

```
Directory = /work/opa/witoil-dev/test1/meglob/DATA
 Wind type =            5           5
 Writing medslik file for date 05/08/2019 01:00
 Writing medslik file for date 05/08/2019 02:00
 Writing medslik file for date 05/08/2019 03:00
 ...
PLEASE WAIT: SIMULATION IS RUNNING
 ...
 STOKE DRIFT CALCULATION
 Reading forecast currents from file relo19080912.rel
 STOKE DRIFT CALCULATION
 Simulation has now completed           96  hours
 Simulation has completed successfully
 UR NETCDF FILE S been CLOSED
 Average velocity components of surface slick (m/s):
 East   -1.6292638497366486E-002    North   -2.8533182176095288E-002
```

4 Running Medslik with LSF

To run on LSF you can use the following code, by properly tuning bsub parameters.

```
#!/bin/bash
wdir=/work/opa/witoil-dev/test1
source ${wdir}/set_env.sh
JJ=$1

# Invoke medslik
MSHome=/work/opa/witoil-dev/test1/meglob/EXE
MSCommand=./RUN.sh
bsub -e ${wdir}/log/test_%J.err -o ${wdir}/log/test_%J.out -q s_short -J ${JJ}
   $MSHome/$MSCommand
```

5 Python files

In this section I report a short explanation of the python scripts included in the project. **NOTE:** this section is still in progress!

5.1 read_oil_data.py

For the sake of brevity I don't report here the full code, but just a few snippets (if necessary) and comments.
Lines 3 and 4 are used to read the file `oilbase.txt` in which a line has the following look:

```
Abu al bu Khoosh     Abu Dhabi        0.86831.6038.0000006.700.91858.690.241
```

Then, line 16 checks the first parameter. If its value is:

- `"NAME"` – It reads the API version through the second parameter passed to the script. Then, it cycles over the lines and when the first field of a row matches the name passed as a parameter it parses the whole line. It extracts `dens_oil`, `api`, `temp`, `visc`, `res_dens_oil`, `res_perc_oil` and `vap_press`. Then, all this information is written in the file `oil_file.txt`.
- `"API"` – It reads the API version through the second parameter passed to the script. Then, it cycles from 0 to 222 (what is this hardcoded number???). In the loop it parses a line using some splits. First, it performs a split using two space characters, in order to get in the first field the name of that zone (since it could be a name with a space, e.g. West Texas sour). After that, it extracts `dens_oil`, `api`, `temp`, `visc`, `res_dens_oil`, `res_perc_oil` and `vap_press`. Then, all this information is written in the file `oil_file.txt`.

5.2 preproc_currents_mdk2.py

Three functions:

- **open_mercator_subset**: takes as input a mercator file and a set of grid corners. It's invoked in the main loop and the first parameter passed is a filename extracted from the folder `iInputFolder` exploting the date in the loop, while the second is `grid_corners`. The input file is opened using the `Dataset` class provided by `netCDF4`. From this file, it reads the a latitude and a longitude that in combination with the grid corners are used to define `xwindow` and `ywindow` to crop the original map. These two variables are numpy arrays obtained through logical and between the area defined by grid coordinates and the global area. Other components are extracted from the dataset and *cropped/sliced*, as the components of the water velocity (`iU` and `iV`) and its temperature `iT`. No need for slicing for time `iTime` (of course) and `iDepth` (why not?). It outputs `x` and `y`, `iU` and `iV`, `iT`, `iTime` and `iDepth`.
- **interp_z**: takes as input `iU` (the *u* component of the velocity). First, it gets the shape of this variable. Then, it initialises an output variable as a 4-dimensional array filled with zeros. The four dimensions are time, depth `x` and `y`. The dimension related to depth has four elements, as four are the depths (0, 10, 30, 120 m). Then it performs interpolation with some formulas (still to study in depth). Finally, it squeezes the output structure (i.e. removes single-dimensional entries from the shape of the array)
- **interp_t**: this fuction performs **temporal interpolation**. Takes as input `iOutputUd`, `iHRTimeLine`, `iLRTimeLine` and returns a matrix `iOutputU`.

A main body that is composed by a first part dedicated to the configuration that:

1. contains an hardcoded input folder for CMEMS-GLO files (variable `iInputFolder`), an hardcoded output folder for MEDSLIK-adapted NetCDF files (variable `iStorageDirectory`).
2. Defines start date and end date (variables `iStartDate` and `iEndDate`) of the pre-processing and of course depends on the input data, as well as input and output reference hours (variables `iInputRefHours` and `iOutputRefHours`).
3. Defines the geographic limits of the preprocessing through the variables `xE`, `xW`, `yS`, `yN` and `grid_corners`.

The second part loops over the dates to:

1. Perform spatial and temporal interpolation using the function above.
2. Generate NC files. So, it defines the dimensions (time, y, x and depth), then the variables. It fills the variables with the data calculated before and adds units as attributes. This process is performed for the *u* and *v* components separately, generating a file that ends with `_U.nc` and another with `_V.nc`. Of course, since we are in the loop, it generates a couple of files for every different day.

5.3 preproc_gshhs_mdk2.py

This script was developed for coastline extraction from **GSHHS**[1] full resolution. It extracts coastline files for the desired part of the globe and makes it compatible with MEDSLIK-II.
It require the os module, the `Dataset` class of the `netCDF4` module and numpy.
It has three functions:

- **oce_grid**: takes as input a filename. That file is opened as a netCDF dataset. The function reads all the values for netCDF variables `nav_lon` and `nav_lat` in the variables `x_mod` and `y_mod` respectively. These variables are then returned by the function.
- **getheader**: is invoked with a file descriptor as input. The first 8 items of the related file are read through `numpy.fromfile` in the variable `A`. The first 8 fields are in fact the headers that can be described as:
  - `id`: Unique polygon id number, starting at 0. Polygon 0 is Eurasia, 1 is Africa, 2 North America, 3 South America, 4 Antarctica and 5 Australia.
  - `n`: Number of points in this polygon
  - `flag`: level + version ≪ 8 + greenwich ≪ 16 + source ≪ 24 + river ≪ 25. The five items contained in the flag are:
    - `level` = flag & 255. Values: 1=land, 2=lake, 3=island in lake, 4=pond in island in lake;
    - `version`: = (flag ≫ 8) & 255. Values: Should be 12 for GSHHG release 12 (i.e., version 2.2);
    - `greenwich`: (flag ≫ 16) & 1. Values: Greenwich is 1 if Greenwich is crossed;
    - `source` = (flag ≫ 24) & 1: Values: 0 = CIA WDBII, 1 = WVS;
    - `river` = (flag ≫ 25) & 1. Values: 0 = not set, 1 = river-lake and level = 2
  - `west`, `east`, `south`, `north`: min/max extent in micro-degrees
  - area: Area of polygon in 1/10 km^2
  - *area_full*: Area of original full-resolution polygon in 1/10 km^2
  - `container`: Id of container polygon that encloses this polygon (-1 if none)
  - `ancestor`: Id of ancestor polygon in the full resolution set that was the source of this polygon (-1 if none);

  The size of this structure is put in the variable `cnt`. If `cnt` is less than 8, the function returns nothing, otherwise (after a few manipulations) it returns `A` and `cnt`. Since this function is invoked by the following one specifying `input_file` as a parameter, `A` and `cnt` will refer to the input GSHHS file.
- **extract_coastline**: takes as input `x_mod` and `y_mod`, the input and the output files. The function defines the limits and prepares matrices, then opens the input file (read only) and extracts the headers using `getheader`. In the middle it performs a lot of tricky transformations (still to fully understand – I have tests in progress...). Finally, it closes the output file.

In the main body we find:

- an hardcoded configuration original gebco netCDF file (variable `gshhs_filename`) and the output directory (variable `output_dir`). It also has a variable (`oce_dir`) to state the files for the hydrometric grid. This is the part defined as*user inputs* ...
- After having defined the input of the script, the program opens an ocean forceast file and invokes the above-mentioned `oce_grid` function. Finally, the function `extract_coastline` is invoked.

[1] Global Self-consistent, Hierarchical, High-resolution Geography Database

5.3.1 GSHHS

GSHHG stands for *Global Self-consistent, Hierarchical, High-resolution Geography Database*. It is a high-resolution geography data set, amalgamated from two databases: **WVS** (*World Vector Shorelines*) and **WDBII** (*CIA World Data Bank II*). GSHHG data have undergone extensive processing and should be free of internal inconsistencies such as erratic points and crossing segments. The shorelines are constructed entirely from hierarchically arranged closed polygons. GSHHG is released under the GNU Lesser General Public license.
I downloaded the main files provided by **NOAA** (*National Oceanic and Atmospheric Administration* from **here**.
I'm currently focusing on binary input files (those in `gshhs-bin-2.3.7.zip`). Before going into the details of my experiments it is worth analysing the content of the file.
The last letter of each file indicates the resolution: `c` stands for *crude*, `l` for *low*, then *intermediate* (`i`), *high* (`h`), and *full* (`f`). The first part of the filename can be `gshhs`, or `wdb_borders` of `wdb_rivers`. Only the first refers to shoreline, the others are for rivers and borders.
Before analysing the code, I report an information found in the README that could be useful in the future:

> *The netCDF distribution provides specially processed netCDF representations of GSHHS and WDBII where the polygons and lines have been subdivided and indexed to deliver rapid map-making for GMT. Users who wish to access GSHHG outside of GMT are advised to use the binary and shapefile version of the actual polygons as there is no user documentation for how to access the netCDF files.*

Now, let's go to the code. The following is a little snippet of my tests (to be incrementally enriched):

```
#!/usr/bin/python3

# requirements
import numpy as np

# vars
INPUTFILE="/home/val/work [cmcc]/files/gshhs/gshhg-bin-2.3.7/gshhs_c.b"

# main
if __name__=="__main__":

    # open the input file
    print("Opening input file %s" % INPUTFILE)
    fd = open(INPUTFILE, "r")

    # read the file content as a numpy structure.
    # dtype parameter is needed for binary files
    # since files are big endian integers, we use >i4.
    # we only take the headers, so 8 items
    print("Reading file content")
    fcontent = np.fromfile(fd, dtype=">i4", count=8)

    # extract information about the region
    region = fcontent[0]
    points = fcontent[1]

    # extract flags
    flags = fcontent[2]
    fl_level = flags & 255
    fl_version = (flags >> 8) & 255
    fl_green = (flags >> 16) & 1
    fl_source = (flags >> 24) & 1
    fl_river = (flags >> 25) & 1

    # print some information
    print("Region: %s" % region)
    print("Number of points: %s" % points)
    print("Level: %s" % fl_level)
    print("Version: %s" % fl_version)
    print("Greenwich: %s" % fl_green)
    print("Source: %s" % fl_source)
    print("River: %s" % fl_river)

    # close the input file
    print("Closing input file")
    fd.close()
```

If we want to access the coordinates, before closing the file you can add:

```
    # read coordinates
    print("Reading coordinates")
    coords = np.fromfile(fd, dtype=">i4", count=2*points)
    x=coords[np.arange(0, coords.size ,2)]*1e-6
    y=coords[np.arange(1, coords.size ,2)]*1e-6

    # Each lon, lat pair is stored in micro-degrees in 4-byte signed integer
       format
    print(x[0], y[0])
```

5.4 preproc_winds_mdk2.py

Two functions:

- open_era_subset: it opens a netCDF file (received in input as sFileName) using the class Dataset provided by netCDF4 (so the input is a netCDF file!). It reads the two variables latitude and longitude and performs cropping. Finally it converts time from "hours from 01-01-1900" to the standard pythonic time.
- interp_t: performs time interpolation using numpy functions.

A main part with:

- Hardcoded inputs: an input folder for CMEMS-GLO files (iInputFolder), an output folder for MedSlik-adapted netCDF files (iStorageDirectory), start and end date as well as reference hours (iStartDate, iEndDate, iInputRefHours, and iOutputRefHours). Then, we have the coordinates to trace the grid corners (xE, xW, yN, and yS).
- A loop over the dates. In this loop it generate a NetCDF file for every day. It opens simultaneously a file for day i and for the day after (using open_era_subset), then performs temporal interpolation by calling interp_t (see above) and then creates all the elements that are needed for a netCDF file (attributes, dimensions, variables).

6 Refactoring...

I made a small re-organisation of Svitlana's code, in order to:

- put the software under versioning and be able to keep track of changes and revert if needed;
- simplify the configuration (thanks to a simple configuration file with all the paths);
- include a wrapper that allows having multiple instances of Medslick running simultaneously;
- separate software from data.

The result of this re-organisation is represented by two tarballs: `medslik-src.tar.gz` and `medslik-data.tar.gz`. In each of the files there are simplified installation scripts and readme files. So, in order to install medslik:

```
$ tar xzvf medslik-src.tar.gz
$ cd medslik-src
$ emacs medslik-src/EXE/medslik.conf
$ ./source/compile.sh
```

And then, to add its datasets:

```
$ tar xzvf medslik-data.tar.gz
$ cd medslik-data
$ emacs medslik-data/install.conf
$ ./install.sh
```

These two repositories will be on CMCC's GitHub page soon.

7 Useful links

- Documentation
- Using the GSHHS Shoreline Database
- Download Shoreline/Coastline Resources
- README for NOAA files