Naval Research Laboratory
Washington, DC 20375-5320

NRL/7350/MR—2022/2

Delft3D-FM Simulation Configuration and Input File Generation

SUNNI S. SCHOENAUER
ALLISON M. PENKO

Seafloor Sciences Branch
Ocean Sciences Division

KACEY L. EDWARDS
JAY VEERAMONY

Ocean Dynamics and Prediction Branch
Ocean Sciences Division

October 31, 2022

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From - To)*
31-10-2022	NRL Memorandum Report	01 Oct 2021 – 30 Sept 2024

4. TITLE AND SUBTITLE

Delft3D-FM Simulation Configuration and Input File Generation

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER
62435N

6. AUTHOR(S)

Sunni S. Schoenauer, Allison M. Penko, Kacey L. Edwards, and Jay Veeramony

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER
6C91

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Research Laboratory
1005 Balch Blvd
Stennis Space Center, MS 39529

8. PERFORMING ORGANIZATION REPORT NUMBER

NRL/7350/MR--2022/2

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Office of Naval Research
One Liberty Center
875 N. Randolph Street, Suite 1425
Arlington, VA 22203-1995

10. SPONSOR / MONITOR'S ACRONYM(S)

ONR

11. SPONSOR / MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A: Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This document describes the setup of Delft3D-Flexible Mesh (DFM) (coupled FLOW and WAVE) on the Navy DoD Supercomputing Resource Center (Navy DSRC) High Performance Computing (HPC) system. New model creation is described to provide the user with a familiarity of the process and the files. The chief purpose of this document is to serve as a step-by-step guide for model setup when using an existing model of the user's domain as the starting point. The procedure for parameter ensemble creation is also explained.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
					Allison M. Penko
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**	U	46	**19b. TELEPHONE NUMBER** *(include area code)*
U	U	U			(228) 688-4823

This page intentionally left blank.

CONTENTS

EXECUTIVE SUMMARY

The purpose of this document is to give an overview of the setup of Delft3D-Flexible Mesh (DFM) (coupled FLOW and WAVE) on the Navy DoD Supercomputing Resource Center (Navy DSRC) High Performance Computing (HPC) system and to serve as a step-by-step guide for model setup starting from existing model input files of a specific domain.

This page intentionally left blank

DELFT3D-FM SIMULATION CONFIGURATION AND INPUT FILE GENERATION

1. INTRODUCTION

This report outlines the set up of Delft3D-Flexible Mesh (DFM) coupled FLOW and WAVE modeling system on the DoD High Performance Computers (HPC). The primary focus is to serve as a step-by-step guide for model setup from an existing model input files of a given domain. Section 2 provides a description of model input file creation. This section is included to familiarize the user with the set-up process and input files, but is not intended to serve as a step-by-step guide to a new model setup. Section 5 outlines the required steps for parameter ensemble creation.

1.1 Background

The US Navy has employed the Delft3D4 (D3D4) modeling system [1] to provide forecasts of water levels, currents, and surface waves in coastal and nearshore environments for over a decade. Delft3D4 allows for coupled circulation and wave simulations on structured grids with quadrilateral elements.

The capacity for high model resolution is crucial because the quality of simulated water levels, currents, and wave parameters in nearshore and coastal environments depends significantly on bathymetric features with short horizontal length scales (tens of meters or less). Though the D3D4 system is capable of producing dependable forecasts at the desired resolution, it requires a cumbersome, nested approach of grids with increasing resolutions in areas of interest [2].

A flexible mesh capability was recently developed for the D3D4 modeling suite: Delft3D-Flexible Mesh (DFM) [3]. The advantage of DFM is that the computational grid for the circulation model is not restricted to quadrilateral elements. The elements can range from triangles to hexagons. Therefore, a flexible mesh of varying resolutions can be generated by combining multiple structured grids using triangular, pentagonal, and hexagonal elements. This allows for simulation of large areas combining lower resolution grids with higher resolution grids in areas of interest.

Replacing the structured grid with the new flexible mesh is expected to allow operators to highly resolve processes and improve predictions close to the coastline without compromising computational time or requiring a cumbersome nested approach. A current limitation of the DFM system is that it can perform only barotropic simulations. At the time of this report, the baroclinic version of DFM remains in Beta testing [2].

2. MODEL DIRECTORY SET-UP

This section outlines the recommended configurations for a model directory setup on a DoD HPC including the necessary files for running the model and the required directory structure.

Manuscript approved October 27, 2022.

2.1 Directory Configuration and Contents

2.1.1 Top-level Directory

The top-level directory should be named using the model <run-id> [3]. Here we use the example of "regionA" to denote a model domain setup in a given domain.

The required subdirectories in the top-level directory regionA for the example are:

```
regionA/dflowfm
regionA/wave
regionA/winds
```

The dflowfm directory contains all the input files for the FLOW circulation module, including the hydrodynamic lateral boundary conditions forcing the simulation. The wave directory contains all the input files for the WAVE module, including wave lateral boundary conditions forcing the simulation. The winds directory contains the wind forcing files used by both the FLOW and WAVE modules.

Additionally, the top-level directory should contain the Portable Bash Script (PBS) run script that will be submitted to the PBS batch queue on the HPC machine and the model configuration file:

run_dimr.pbs	model simulation PBS batch script
dimr_config.xml	DIMR model configuration file

The location of the compiled DFM model executable is indicated in the model PBS run script, <.pbs>. The executable can exist in the user's home directory, or in the home directory of another user in the group with the correct permissions.

2.1.2 dflowfm Subdirectory

The dflowfm subdirectory contains all the input files required to run the FLOW module of DFM:

<_net.nc>	unstructured grid file
<.pli>	boundary condition location file(s) (polyline file format)
<.bc>	lateral boundary forcing file(s)
<_bnd.ext>	file containing lateral boundary conditions files' specifications
<.ext>	file containing meteorological forcing files' specifications
<.xyn>	list of point locations (lat,lon) of observation stations with columns x, y, station name
<.mdu>	master definition file for parameters in the FLOW model

2.1.3 wave Subdirectory

The wave subdirectory contains all the input files required to run the WAVE module of DFM:

<.dep>	bathymetry or water depth file
<.grd>	curvilinear grid file
<.mdw>	master definition file for parameters in the WAVE model

2.1.4 `winds` Subdirectory

The `winds` subdirectory contains the meteorological files that are forced at the air-sea boundary of FLOW and WAVE:

<`.amp`>	meteorological forcing file (pressure)
<`.amu`>	meteorological forcing file (wind, u-component)
<`.amv`>	meteorological forcing file (wind, v-component)

3. MODEL INPUT FILES

Generating the model input files for DFM is similar to D3D4. Locally accessible tools are first used to configure model settings and inputs. The input files are then transferred to the DoD HPC machines where the simulation is executed. The required inputs include a grid file, bathymetry (depth) file, circulation (FLOW) model boundary conditions, meteorological boundary conditions, and wave (WAVE) model boundary conditions [2].

3.1 Model Run Script and DIMR Configuration File

Two files need to be placed in the top-level directory. They are the model run script, `run_dimr.pbs`, and the DIMR input file, `dimr_config.xml` file.

3.1.1 Model Run Script

The model run script, `run_dimr.pbs`, is used to launch the simulation on HPC. The run script we use was written especially for our HPC configuration and can be copied from the directory of another user in the group.

3.1.2 DIMR Input File

Both FLOW and WAVE are used as dynamic libraries. DIMR is the executable that steers both dynamic libraries. It prescribes when the FLOW computation should be paused to perform a WAVE calculation. The input file is usually named `dimr_config.xml` [3].

If a copy of `dimr_config.xml` is not created during initial model setup, one can be copied from the directory of another user in the group.

3.2 Grid and Bathymetry Generation

The model grid consists of mostly rectangular grid cells. The grid cell size is refined (i.e., reduced) in areas of the highest interest or where higher resolution is necessary. Triangular grid cells are used to transition from areas of low resolution to areas of high resolution. Grid and bathymetry generation utilizes Delft DashBoard (DDB), the RGFGrid Graphical User Interface (GUI), the DeltaShell GUI, and shell scripts (`refine_netfile_gebco.sh` and). Shell scripts are run on the Navy DoD Supercomputing Resource Center (Navy DSRC). DDB is used to generate a beginning (base) grid and to access bathymetry for the base grid (The General Bathymetric Chart of the Oceans (GEBCO 2019)) [4] and for areas of refinement (Coastal Relief Model (CRM)) [5] [6] [7] [8]. The base grid and bathymetry information are inputs to the shell scripts. Given the inputs and several tunable parameters, the shell scripts determine the refined mesh areas. The GUIs are used to save the refined grids in usable formats. Step-by-step instructions for grid generation and refinement can be found in Appendix A of *An Evaluation of Delft3D Flexible Mesh Applied to Southern California and the Gulf of Mexico* [2]. The <`_net.nc`> file results from this procedure.

3.3 Observation Points File

The points file with observation stations, <.xyn>, indicates the location of point output, which will be written to the the <_his.nc> file. The <_his.nc> file contains time series with DFM model results for a number of stations. The <.xyn> file is generated using DDB at the time the base grid is created. If you choose not to create it in this manner, it can be created in a text editor, such as Vim. Each entry belongs on its own line. Its format is longitude (-180 to 180 convention), latitude, and station ID contained in quotes.

Example of <.xyn> file contents:

```
-84.516000  28.501000  "42036"
-86.008000  28.788000  "42039"
-88.226000  29.208000  "42040"
```

3.4 Hydrodynamic Boundary Conditions

3.4.1 Delft DashBoard Method

The DDB Model Maker Toolbox is used to define open boundaries and generate astronomic water level and normal velocity boundary conditions. Polyline files, <.pli>, are made during this process.

3.4.2 Python Method Using Naval Operational Model Sources

An alternative approach to generate boundary conditions is to pull boundary condition information from a regional model, like the Navy Coastal Ocean Model (NCOM) [9]; this approach to generate boundary conditions utilizes external software written in Python (`generate_BCs_NCOM.py`), not DDB [2]. In order to use this alternative method, one needs polyline files made for the domain, which is done using DDB.

The inputs to `generate_BCs_NCOM.py` include <.pli> files(s), a network Common Data Form (NetCDF) file containing the regional model (e.g., NCOM) output (`ncFile`), and the quantity from the model to be used for boundary condition creation (`quantity`). The command syntax is as follows:

```
>> python generate_BCs_NCOM.py [-h] --pli-list [PLI_LIST [PLI_LIST ...]]
[--bc-filename BC_FILENAME] [--nlevels N] [--depth-avg] [--plot] ncFile quantity
```

positional arguments	
`ncFile`	NCOM NetCDF output containing boundary support points and duration of Delft3D simulation
`quantity`	NCOM variable; must be either `salinity`, `water_temp`, `velocity`, or `surf_el`
optional arguments	
`-h, -help`	show this help message and exit
`-pli-list [PLI_LIST [PLI_LIST ...]]`	list of boundary support point polyline filenames
`-bc-filename BC_FILENAME`	optional filename for Delft3D boundary condition filename
`-nlevels N`	optional number of vertical levels for 3D boundary conditions
`-depth-avg`	flag to enable depth averaged output
`-plot`	flag to enable plotting

See Appendix A.1 for the `generate_BCs_NCOM.py` Python code.

Creation of NetCDF Files Used as Input to Python Method

The starting point for boundary condition creation from NCOM is the creation of the `ncFile` used as input to `generate_BCs_NCOM.py`. A Python script, `pull_NCEI_data.py`, along with its accompanying input file, `NCEI_input.yml`, are used to obtain the NetCDF files [10]. The routine retrieves NCOM model output from the National Centers for Environmental Information (NCEI) Environmental Research Division's Data Access Program (ERDDAP) server and saves it in a NetCDF file. The data pulled is determined by variables supplied in the `NCEI_input.yml` yaml file that is named on the command line.

Syntax for running `pull_NCEI_data.py` is as follows:

```
>>python pull_NCEI_data.py -y NCEI_input.yml -o . -i .
```

Values needed for `NCEI_input.yml`:

dataset:	The dataset (model and 2D or 3D variables) requested from the NCEI database
variable:	METOC variable requested
start_time:	The beginning of the time range (yyyy-mm-dd HH:MM:SS)
end_time:	The end of the time range (yyyy-mm-dd HH:MM:SS)
latitude_min:	The minimum of the latitude range
latitude_max:	The maximum of the latitude range
longitude_min:	The minimum of the longitude range
longitude_max:	The maximum of the longitude range

The `pull_NCEI_data.py` code can be used for obtaining results from both 2D and 3D models. A 2D dataset containing surface elevation (`surf_el`) is needed for the creation of water level boundary conditions. When using `pull_NCEI_data.py` to obtain 2D NCOM output, only one netCDF file is made. 3D NCOM output (`water_u` and `water_v`) is needed for creating velocity boundary conditions. For obtaining 3D NCOM, the script automatically pulls two files per day (AM and PM) for all days in the range. Then it concatenates the 3D NCOM netCDF files to arrive at a single `ncFile` for input to `generate_BCs_NCOM.py`.

Date range and spatial coverage for NCOM can be found at:

https://www.ncei.noaa.gov/products/weather-climate-models/fnmoc-regional-navy-coastal-ocean

The `pull_NCEI_data.py` Python code and `NCEI_input.yml`, its accompanying yaml input file, are available in the US Naval Research Laboratory (NRL) Memorandum Report, *Automated Method to Extract Oceanographic and Atmospheric Data from Online Sources* [10].

Once the 2D `ncFile` and the 3D `ncFile` are obtained/constructed, they can be used with `generate_BCs_NCOM.py` to create the needed BCs, in this example, WaterLevel.bc and Velocity.bc.

Execution of generate_BCs_NCOM.py

A Python environment, dfm, must be created before using `generate_BCs_NCOM.py` [11]. After it is created, it must be activated:

```
>> conda activate dfm
```

After activation, execute the routine from the command line.

In the below example, the polyline files and the `ncFiles` are in the directory where the boundary condition creation scripts are executed. The <.bc> file will be written to the directory where `generate_BCs_NCOM.py` is executed.

Example usage for water level boundary condition creation:

```
>> python3 /path/to/generate_BCs_NCOM.py --pli-list bnd001.pli bnd002.pli
--bc-filename WaterLevel.bc amseas2d_subset_20130420_20130511.nc surf_el
```

Example usage for velocity boundary condition creation (depth-averaged):

```
>> python3 /path/to/generate_BCs_NCOM.py --pli-list bnd001.pli bnd002.pli --bc-
filename Velocity.bc --depth-avg amseas3d_subset_20130420_20130511.nc velocity
```

3.5 Meteorological Forcing

External code written in Python is used to generate atmospheric forcing files. Meteorological forcing is obtained from the Coupled Ocean/Atmosphere Mesoscale Prediction System (COAMPS) model using `coamps2meteo.py`. If COAMPS forcing is unavailable or unsuitable, meteorological forcing can be obtained from the ERA5 model using `ERA5_to_meteo.py`. The user must first check to see that coverage is available for the time period and region. For COAMPS, connect to the host server and check the available dates, by year, for the area of interest.

Execution of `coamps2meteo.py`

The `coamps2meteo.py` code requires input of the name of a COAMPS region, a start date, an end date, and the path to the directory containing the COAMPS output. It accesses and reads the COAMPS output and writes the spatially varying meteorological input files <.amu>, <.amv>, and <.amp> for use in DFM [2].

The dfm Python environment, used when creating boundary conditions from NCOM output, must also be used when running `coamps2meteo.py`. It must be activated before using the routines. After activation, execute the routine from the command line.

The command syntax for running `coamps2meteo.py` is as follows:

```
>> python coamps2meteo.py [-h] -r REGION -s STR_DATE -e END_DATE -d DATADIR
```

where

`REGION` is one of:

arctic	cencoos_n3	e_pac_comp	europe2	hawaii_n3	mbay_n4
n_ind2	southwest_asia	useast	cen_amer	centam	eqam
europe3	indo	mideast	nwatl	southwest_asia2	was3
cen_america	centio	eqam_comp	fukushima	mbay_n1	MREA04
nwpac	southwest_asia2_n3	w_atl	cencoos_n1	eafr	euro
hawaii	mbay_n2	nepac	socal	southwest_asia3	w_pac
cencoos_n2	e_pac	europe	hawaii_comp	mbay_n3	n_ind
somalia	test2	w_pac2			

`STR_DATE`	start date in YYYYMMDD format
`END_DATE`	end date in YYYYMMDD format
`DATADIR`	location of the COAMPS output files

`DATADIR` currently requires file tree to be analogous to `NOGAPS/COAMPS` tree (e.g., `DATADIR/region/YYYY`).

The following files are produced:

```
coamps\_start\_date\_end\_date\_region.amp \\
coamps\_start\_date\_end\_date\_region.amu \\
coamps\_start\_date\_end\_date\_region.amv \\
coamps\_region\_start\_date\_end\_date\_today.log \\
```

Example usage for `coamps2meteo.py`:

```
>> python3 /path/to/coamps2meteo.py -r w_atl -s 20130425 -e 20130511 -d /path/to/
NOGAPS/COAMPSg/
```

See Appendix A.2 for the `coamps2meteo.py` Python code.

3.6 File Transfer to HPC

If boundary conditions files and meteorological forcing files were created on your local system, you can scp them to the directories on the HPC. If any changes were made to the polyline files, transfer the updated polyline files that correspond to the boundary condition files as well. To do this, one has to have an active ticket on the HPC. The following command is performed in the local directory containing the files.

The form of the scp command follows:

```
>> scp <file> user@system.hpc_center.hpc.mil:/p/work1/user/regionA/dflowfm
```

An example of the scp command is:

```
>> scp *.bc user@gaffney.navydsrc.hpc.mil:/p/work1/user/regionA/dflowfm
```

The files that need to be copied to HPC are:

<*.bc> to dflowfm
<*.pli> to dflowfm
<*.am*> to winds

Then login to the HPC system to confirm successful transfer of the new files:

```
>> ls /p/work1/user/regionA/dflowfm/
```

```
bnd001.pli
bnd002.pli
Velocity.bc
WaterLevel.bc
```

```
>> ls /p/work1/user/regionA/winds/
```

```
coamps_20130425_20130511_w_atl.amp
coamps_20130425_20130511_w_atl.amu
coamps_20130425_20130511_w_atl.amv
```

3.7 External Forcing Files

The files containing lateral boundary conditions files' specifications, <_bnd.ext>, and the meteorological forcing files' specifications, <.ext>, are known as external forcing files. They are not the boundary condition files or the spatially-varying meteorological forcing files themselves; they contain the meta data for the actual forcing data files. They are made in DeltaShell. The names of the boundary conditions files and meteorological forcing files are needed for making <_bnd.ext> and <.ext>, respectively.

Two definition files for the external forcing are supported, each with their own format. The <.ext > is the old style, and details about meteorological forcing must be contained in this kind of external forcing file. Boundary conditions may use this kind of file [3]. The <_bnd.ext> file is the new style and can be used for boundary conditions but not meteorological forcing files [3]. The final step in specifying boundary conditions is pointing to the <_bnd.ext> and <.ext> files in the <.mdu> file [3]. For further explanation see Section 4.9.

If you do not create these files in DeltaShell, they can be created using a text editor such as Vim.

3.7.1 New Style External Forcing File: <_bnd.ext>

In the <_bnd.ext> file, there is one [boundary] entry per polyline file per boundary condition. The following code taken from a <_bnd.ext> file is for a simulation using two polyline files (bnd001.pli and bnd002.pli) and two boundary conditions (contained in WaterLevel.bc and Velocity.bc).

Example of <_bnd.ext> file contents:

```
[boundary]
quantity=waterlevelbnd
locationFile=bnd001.pli
forcingFile=WaterLevel.bc
returnTime=0.0000000e+000

[boundary]
quantity=waterlevelbnd
locationFile=bnd002.pli
forcingFile=WaterLevel.bc
returnTime=0.0000000e+000

[boundary]
quantity=uxuyadvectionvelocitybnd
locationFile=bnd001.pli
forcingFile=Velocity.bc
returnTime=0.0000000e+000

[boundary]
quantity=uxuyadvectionvelocitybnd
locationFile=bnd002.pli
forcingFile=Velocity.bc
returnTime=0.0000000e+000
```

quantity needs to match the information in the respective <.bc> file. Keywords and information about the file format can be found in DFM manual [3].

3.7.2 Old Style External Forcing File: <.ext>

Example of <.ext> file contents:

```
QUANTITY=windx
FILENAME=../winds/coamps_20130425_20130511_w_atl.amu
FILETYPE=4
METHOD=2
OPERAND=+
```

```
QUANTITY=windy
FILENAME=../winds/coamps_20130425_20130511_w_atl.amv
FILETYPE=4
METHOD=2
OPERAND=+

QUANTITY=atmosphericpressure
FILENAME=../winds/coamps_20130425_20130511_w_atl.amp
FILETYPE=4
METHOD=2
OPERAND=+
```

The location of the meteorological forcing files is given with respect to the location of the <.ext> file.

3.8 Master Definition File for Hydrodynamics

The master definition file is created by the DeltaShell GUI. The bathymetric sample files are imported and interpolated to the grid to form the bathymetric surface. The boundary definitions and conditions are imported; the atmospheric forcing files are imported, and the observation points for point output are imported. The user checks default model parameter settings and makes adjustments, if needed. The model configuration is saved and ported to a HPC machine where it can be run in parallel or serial using PBS scripting [2].

3.9 Coupling with Waves

After successful configuration of a FLOW circulation model, it can be coupled with the WAVE model if desired. Configuring the WAVE model begins with development of the grid, <.grd>, and bathymetry, <.dep>, using the Model Maker Toolbox in DDB. The WAVE model can be forced by only the meteorological boundary conditions or a combination of the meteorological boundary conditions and wave boundary conditions at the grid boundaries. Because the atmospheric forcing is shared with the FLOW model, the user must ensure that the atmospheric input files cover the WAVE grid, which is typically larger than the FLOW grid. If wave boundary conditions are desired at the grid boundaries, they are generated from a regional model like Simulating WAves Nearshore (SWAN) or WaveWatch 3. In this case, the DeltaShell GUI is used to define the WAVE boundaries along the grid boundaries with spectral points evenly distributed on the boundaries. These points are added to the regional wave model as output locations, and the regional model is set to produce 2D spectra (in the SWAN format) at the output point locations for input to the DFM model. The 2D spectra filenames are added to the WAVE master definition file, <.mdw> [2].

The WAVE grid, bathymetry, and lateral boundary conditions (if applicable) are imported to DeltaShell, and switches to couple the FLOW and WAVE models are engaged. The coupled model configuration is ported to an HPC machine, where it can be run in parallel using PBS scripting.

4. STARTING FROM AN EXISTING SIMULATION

It is common to run multiple models using the same domain. The purpose of this section is to detail what information can be duplicated from an existing model of a specific domain and the required information

updates. Some files will need to be created for each unique study period. In those cases, the user will be directed to the descriptions of file creation in Section 3.

4.1 Approach

By starting from a previous model of your domain, many of the required files already exist and are ready for use in a new model. If the study period is changed, new boundary conditions and meteorological forcing will require recreation. The following files can be copied from a previous simulation that uses the same domain. Whether a file can be used without changes or whether it may require updating, is listed below:

model run script <.pbs>	may require update
DIMR input file <.xml>	may require update
unstructured grid file <_net.nc>	no changes
observation points file <.xyn>	may require update
polyline file(s) <.pli>	may require update (usually not)
new style external forcing file <_.bnd.ext>	may require update (usually not)
old style external forcing file <.ext>	may require update
mater definition file for FLOW <.mdu>	may require update
water depth file <.dep>	no changes
curvilinear grid file <.dep>	no changes (unless new bathymetry is acquired)
mater definition file for WAVE <.mdw>	may require update

4.2 HPC Directory Structure Creation

Set up the directory structure on HPC according to guidance in Section 2.1.

4.3 Model Run Script and DIMR Input File

Two files are required in the top-level directory. They are the model run script, `run_dimr.pbs`, and the DIMR input file, `dimr_config.xml`.

The `run_dimr.pbs` and `dimr_config.xml` can be copied from a previous simulation's directory.

Example of the PBS commands in the header of `run_dimr.pbs`:

```
#!/bin/bash
#PBS -A NRLSS06632HCF
#PBS -q high
#PBS -N regionA
#PBS -o log.dimr
#PBS -M sunni.schoenauer@nrlssc.navy.mil
#PBS -m be
#PBS -j oe
#PBS -W umask=0022
#PBS -l walltime=18:00:00
#PBS -l select=1:ncpus=24
#PBS -l place=scatter:excl
```

PBS commands in `run_dimr.pbs` that may require updating:

```
-A  HPC_project
-q  queue_priority
-N  job_name
-M  e-mail_address
-l  wall_time
```

Additionally, any commands pertaining to archiving the completed simulation (at bottom of script) may require updating.

In `dimr_config.xml`, check `workingDir` names and `inputFile` names in both the `Waves` and `FlowFM` sections. What is visible in the below code, taken from `dimr_config.xml`, corresponds to a FLOW subdirectory named `dflowfm` and a WAVE subdirectory named `wave`. The names of the <.mdw> and <.mdu> files should match the names of the files in your `wave` and `dflowfm` sub-directories.

Example of sections of `dimr_config.xml` that may require updating:

```
<component name="Waves">
<workingDir>wave</workingDir>
<inputFile>regionA.mdw</inputFile>

<component name="FlowFM">
<workingDir>dflowfm</workingDir>
<inputFile>regionA.mdu</inputFile>
```

4.4 Unstructured Grid File and Observation Points File

The unstructured grid file, <_net.nc>, and the points file with observation stations, <.xyn>, can be copied from an existing model of your domain to the `dflowfm` directory. The unstructured grid file <_net.nc> will not need to be changed. The points file with observation stations <.xyn> may need to be updated to include observation points corresponding to the instruments deployed during your study period.

The following was added to the <.xyn> file for the `regionA` study period:

```
-85.673510  30.080170  "QuadPod07m"
-85.688940  30.050390  "QuadPod20m"
```

For more information about <.xyn> file contents, see Section 3.3

4.5 Boundary Condition Creation

Boundary conditions need to be created for each new study period. Refer to Section 3.4 for instructions.

4.5.1 `ncFiles` for Input to Boundary Condition Creation Routine

Use `pull_NCEI_data.py` to obtain the needed `ncFiles` for input to `generate_BCs_NCOM.py`. (Refer to instructions in Section 3.4.2.)

4.5.2 Polyline Files

It is important to note that if NCOM is used for creating boundary conditions and the previous model used astronomic tides for boundary forcing, not all support points in the polyline files may have NCOM results available. If this is the case, any points that missing (e.g., on land) in NCOM can be removed, and performance may be increased.

4.5.3 Executing the Boundary Condition Creation Routine

Execute `generate_BCs_NCOM.py` according to the instructions in Section 3.4.

4.6 Meteorological Forcing Creation

Meteorological forcing needs to be created for each new study period. Refer to Section 3.5 for instructions.

4.7 File Transfer to HPC

Transfer any files you have created (or changed) on your local system to your directories on HPC. The boundary conditions file(s), <.bc>, and the polyline file(s), <.pli>, will need to be copied into the `dflowfm` subdirectory. The meteorological forcing files, <.am*>, will need to be copied into the `winds` subdirectory. Refer to the instructions in Section 3.6.

4.8 External Forcing Files

4.8.1 New Style External Forcing File: <_bnd.ext>

The file containing the lateral boundary conditions files' specifications, <_bnd.ext>, can be copied from an existing model of your domain to the `dflowfm` directory. It may need to be updated to match the names of the polyline files and the boundary condition types if any changes have been made since the previous simulation. In the <_bnd.ext> file, there is one [boundary] entry per polyline file per boundary condition.

The contents from a <_bnd.ext> file (shown below) are for a simulation using two polyline files (`bnd001.pli` and `bnd002.pli`), and two boundary conditions (contained in `WaterLevel.bc` and `Velocity.bc`).

```
[boundary]
quantity=waterlevelbnd
locationFile=bnd001.pli
forcingFile=WaterLevel.bc
returnTime=0.0000000e+000

[boundary]
quantity=waterlevelbnd
locationFile=bnd002.pli
forcingFile=WaterLevel.bc
returnTime=0.0000000e+000
```

```
[boundary]
quantity=uxuyadvectionvelocitybnd
locationFile=bnd001.pli
forcingFile=Velocity.bc
returnTime=0.0000000e+000

[boundary]
quantity=uxuyadvectionvelocitybnd
locationFile=bnd002.pli
forcingFile=Velocity.bc
returnTime=0.0000000e+000
```

The `quantity` value needs to match the information in the respective boundary condition file. For example, `WaterLevel.bc`'s `Quantity` line (pictured below) matches the `quantity` listed in the <`_bnd.ext`> file. Similarly, `Velocity.bc` file's `Vector` line (pictured below) matches the `quantity` listed in the <`_bnd.ext`> file.

Header lines of `WaterLevel.bc`:

```
           [forcing]
                Name  = bnd001_0001
            Function  = timeseries
  Time-interpolation  = linear
            Quantity  = time
                Unit  = seconds since 2013-04-25 00:00:00
            Quantity  = waterlevelbnd
                Unit  = m
```

Header lines of `Velocity.bc`:

```
           [forcing]
                Name  = bnd001_0001
            Function  = timeseries
  Time-interpolation  = linear
            Quantity  = time
                Unit  = seconds since 2013-04-25 00:00:00
              Vector  = uxuyadvectionvelocitybnd:ux,uy
            Quantity  = ux
                Unit  = -
            Quantity  = uy
                Unit  = -
```

4.8.2 Old Style External Forcing File: <.ext>

The file containing the meteorological forcing files' specifications, <`.ext`>, can be copied from an existing model of your domain to the `dflowfm` directory. The meteorological forcing file names contained in the

<.ext> file, i.e., FILENAME, need to match the names of files contained in the winds subdirectory. The location must be supplied with respect to the location of the <.ext> file. Specifically, since the <.ext> file is located in the dflowfm directory, the path to the <*.am*> files from dflowfm needs to match the path listed after FILENAME.

```
QUANTITY=windx
FILENAME=../winds/coamps_20130425_20130511_w_atl.amu
FILETYPE=4
METHOD=2
OPERAND=+

QUANTITY=windy
FILENAME=../winds/coamps_20130425_20130511_w_atl.amv
FILETYPE=4
METHOD=2
OPERAND=+

QUANTITY=atmosphericpressure
FILENAME=../winds/coamps_20130425_20130511_w_atl.amp
FILETYPE=4
METHOD=2
OPERAND=+
```

4.9 Master Definition File for FLOW Model

A copy of the master definition file for the FLOW model, <.mdu>, can be copied from an existing model of your domain to the dflowfm directory. The name of the <.mdu> file will need to match the name specified in the dimr_config.xml file (Section 4.3). The recommended naming convention is <run-id>.mdu, where <run-id> consists of only letters and numbers, up to 252 characters [3].

The following <.mdu> file contents may require updating:

```
           [geometry]
              NetFile  must match the name of the <_net.nc> in dflowfm

               [time]
              RefDate  often the time the model starts
               TStart  start time w.r.t.  RefDate (in TUnit)
                TStop  stop time w.r.t.  RefDate (in TUnit)

  [external forcing]
         ExtForceFile  must match name of <.ext> file in dflowfm
      ExtForceFileNew  must match name of <_bnd.ext> file in dflowfm

             [output]
              ObsFile  must match name of <.xyn> file in dflowfm
```

Boundary condition and forcing coverage must be available between (and including) TStart and TStop.

4.10 Water Depth and Curvilinear Grid Files

The <.dep> and <.grd> files can be copied from an existing model of your domain to the wave directory. No changes will need to be made to these files.

4.11 Master Definition File for WAVE Model

A copy of the master definition file for the WAVE model, <.mdw>, can be copied from an existing model of the domain to the wave directory. The name of the file will need to match the name specified in the dimr_config.xml file (Section 4.3). The recommended naming convention is <run-id>.mdw, where <run-id> consists of only letters and numbers, up to 252 characters [3].

The following <.mdw> file contents may require updating:

```
      [General]
  ReferenceDate  must match RefDate from the <.mdu> file

       [Output]
        COMFile  = ../dflowfm/output/<run-id>_com.nc
                 (Note:  <run-id> must match in <.mdu> and <.mdw> files)

       [Domain]
           Grid  must match name of <.grd> file in wave
       BedLevel  must match name of <.dep> file in wave
```

4.12 Model Execution

Now that the directory structure with the model inputs has been set-up, verify that its contents match the requirements described in Section 2.1. Then, submit the run from the top-level directory using the following command:

```
>> qsub run_dimr.pbs
```

The PBS run id will be output to the command line:

```
3675481.pbsserver
```

To check the status of the job, you can use the qstat or qviewcommand. The -u flag followed by your username will allow you to see only your jobs.

For example:

```
>> qstat -u siqueira
```

```
pbsserver:

                                                                  Req'd   Req'd      Elap
  Job ID         Username  Queue  Jobname  SessID  NDS  TSK  Memory  Time   S  Time
  -----          ---       ---    ----     --      -    -    --      --     -  --
  3675481.pbs*   siqueira  high   regionA  22002   1    24   -       18:00  R  01:14
```

Once the simulation is complete, copy the results file, <_his.nc>, to your local directory for the model. To do this, logout of the HPC system, change to the local directory where you want to save the results file, and perform a scp:

```
>> exit

logout
Connection to gaffney.navydsrc.hpc.mil closed.

>> cd Documents/cases/regionA/
>> scp siqueira@gaffney.navydsrc.hpc.mil:/p/work1/siqueira/regionA/
dflowfm/output/regionA_0000_his.nc .
```

5. CREATING A PARAMETER ENSEMBLE

5.1 Introduction

Once all of the inputs for the model have been completed, a parameter ensemble can be created. A parameter ensemble is a group of simulations whose members are identical in all ways except for the model parameter(s) being tested. These ensembles allow for model parameter sensitivity studies (e.g., using the Generalized Likelihood Uncertainty Estimation (GLUE) method [12]). A set of Python and Bash scripts called `instantiate` creates the inputs (e.g., unique <.mdu> or <.mdw> files) for each ensemble member. The code and further information are available on the NRL HPC GitLab repository, `instantiate_dfm` [13].

We recommend a version of Python (e.g., Miniconda [14], or Anaconda) be installed in your local HPC home directory versus using the HPC version of Python to allow for greater control over the libraries and packages. The user must initialize conda (either within the user's `.bashrc` file or manually after login). It is also recommended that the user activate the Python `dfm` environment to ensure the required libraries and packages are installed.

5.2 Ensemble Directory Structure

A directory structure with a `template` directory should be created. The contents of the `template` directory will be used to create each ensemble member's directory, `case_00**`. For example:

```
regionA
regionA/template
regionA/template/dflowfm
regionA/template/wave
regionA/winds
```

To save disk space, the winds directory should only be in the top-level directory and not in each ensemble member's directory. Verify that the meteorological forcing files are in the correct directory, and the <.ext> file includes the correct path location:

```
>> cd case_0000/dflowfm/
>> vim ESTCP2021.ext
```

Copy the name of one of the meteorological forcing files from the <.ext> file, including path.
Use it in the following `ls` command:

```
>> ls ../../winds/coamps_20210402_20210604_nwatl.amp
```

The output below shows it is visible from the location of the <.ext> file.

```
../../winds/coamps_20210402_20210604_nwatl.amp
```

5.3 Required Instantiate Files

The `01_generate_ensembles_and_submission_script.py` Python script uses the template directory and instantiates it for every possible combination of values of variables as described in an input file, `input_params.txt`. Each instance is placed within a numbered `"case_NNNN"` in a user-defined output path [13].

In addition to the files described in Section 2.1, a copy of the `instantiate` script (`01_generate_ensembles_and_submission_script.py`) and the input parameter file (`input_params.txt`) must be in the top-level directory.

Example contents of `input_params.txt`:

```
UnifFrictCoef     0.005  0.035  7
UnifFrictCoef1D   0                 # REQUIRED so UnifFrictCoef val is not replaced
UnifFrictCoefLin  0                 # REQUIRED so UnifFrictCoef val is not replaced
Vicouv            0.0    20.0   9
```

This example file specifies that 7 values of `UnifFrictCoef` should be used, ranging from 0.005 to 0.035 in evenly spaced increments. Note that the next two lines specify that `UnifFrictCoef1D` and `UnifFrictCoefLin` will be written as 0 so the code does not replace their values with the `UnifFrictCoef` values. This legacy code bug should be fixed in the future. The final line specifies that 9 values of `Vicouv` will be used, ranging from 0.0 to 20.0 in evenly spaced increments. In this example, the parameters intended for replacement with the specified values are all in the <.mdu> file.

5.4 Modification to Model Files

No changes need to be made to the model input files in the `template` directory EXCEPT for the `run_id` name. In order for a unique job name to be written to each ensemble member's `run_dimr.pbs` file, the characters `@( f'run_id')@` must be inserted into the `run_dimr.pbs` file contained in the `template` directory:

```
#PBS  -N  regionA_@(f'run_id')@
```

This modification will allow for unique job names in the batch queue of the HPC system, allowing for ease of model simulation tracking.

5.5 Running Instantiate

A typical command on HPC for instantiating an ensemble is:

```
>> python 01_generate_ensembles_and_submission_script.py -i './template/
input_params.txt' -t './template/' -o '.'
```

Command line switches for `01_generate_ensembles_and_submission_script.py` include:

```
'-i'  '-input-filename'  help=input file describing ensembles
                         default='input_params.txt'
'-t'  '-template-dir'    help=input template directory
                         default='../template'
'-o'  '-output-dir'      help=output directory
                         default='../output'
'-q'  '-quiet'           help=do not print anything except error msgs
                         action=store_true
'-y'  '-yes'             help=confirm all
                         action=store_true
```

When executed, the program prints the values that will be written to each ensemble member's files. Make sure to verify they are correct. If so, answer `y` to the prompts:

```
Continue instantiating? (y/n)
Delete yamls when done parsing? (y/n)
```

The `case_00*` directories will be created after giving confirmation.

5.5.1 Confirming Values in Ensemble Members' Files

It is a good practice to use `grep` commands to confirm the values in the files.

Comannd syntax is as follows:

```
>> grep <parameter> case_00*/dflowfm/<run-id>.mdu
>> grep <run-id> case_00*/run_dimr.pbs
```

Example of the grep commands:

```
>> grep Vicouv case_00*/dflowfm/regionA.mdu
>> grep regionA case_00*/run_dimr.pbs
```

5.6 Preparing to Launch the Ensemble Simulation

The `01_generate_ensembles_and_submission_script.py` code generates a script, `02_submit_all_cases.sh`, that is used to submit the batch scripts to all of the ensemble simulation cases to the PBS queue. The file will be created in a write-protected mode.

You then must change the permissions of the `02_submit_all_cases.sh` script:

```
>> chmod 755 02_submit_all_cases.sh
```

Consider making copies of `02_submit_all_cases.sh` with partial lists of cases to submit so under 100 jobs are submitted at once. For example, make three submission scripts with the submission commands for only a portion of the simulations. For an ensemble with 63 members, consider three groups and duplicate the submission script, `02_submit_all_cases.sh`.

```
group1  0-20
group2  21-41
group3  42-62
```

```
>> cp  02_submit_all_cases.sh  02_submit_all_cases_group1.sh
>> cp  02_submit_all_cases.sh  02_submit_all_cases_group2.sh
>> cp  02_submit_all_cases.sh  02_submit_all_cases_group3.sh
```

To quickly eliminate the unwanted lines from each files, use Vim line editor command, `dd N`, to delete `N` lines from the cursor location.

5.7 Launching the Ensemble Simulation

The `02_submit_all_cases.sh` file(s) can be executed from the command line from the top-level directory.

```
>> ./02_submit_all_cases.sh
```

5.7.1 HPC Reservations

If you use a reservation for running the ensemble, remove the `-q` and `-A` lines from the `run_dimr.pbs` in the `template` directory, and include the queue and project info contained in the reservation as part of each line beginning with `qsub` in the `02_submit_all_cases.sh` file.

Example instructions from a HPC reservation e-mail:

Please note that you need to submit a job to your reservation in order to make use of it:

`"qsub -q R3755198 -A NRLSS06632054 jobfile"` will assign the job to your reservation.

Example lines from modified `02_submit_all_cases.sh`:

```
cd ./case_0000
qsub -q R3755198 -A NRLSS06632054 run_dimr.pbs
cd ../
sleep 2
```

REFERENCES

1. G. R. Lesser, J. v. Roelvink, J. T. M. van Kester, and G. Stelling, "Development and validation of a three-dimensional morphological model," *Coastal engineering* **51**(8-9), 883–915 (2004).

2. K. L. Edwards, S. S. Schoenauer, A. M. Penko, J. Veeramony, C. A. Blain, and T. Campbell, "An Evaluation of Delft3D Flexible Mesh Applied to Southern California and the Gulf of Mexico.," NRL Memorandum Report NRL/7320/MR–2022/7, 5682, US Naval Research Laboratory, Stennis Space Center, MS, 2022.

3. Deltares, *D-Flow Flexible Mesh, User Manual* (Deltares Systems, 2022).

4. K. Marks et al., *The IHO-IOC GEBCO Cook Book.* (International Hydrographic Organization, Intergovernmental Oceanographic . . . , 2019).

5. N. N. G. D. Center, "U.S. Coastal Relief Model - Southern California vers. 2.," dataset, National Geophysical Data Center, NOAA, NOAA National Centers for Environmental Information, 2012. doi:10.7289/V5V985ZM.

6. N. N. G. D. Center, "U.S. Coastal Relief Model Vol.3 - Florida and East Gulf of Mexico.," dataset, National Geophysical Data Center, NOAA, NOAA National Centers for Environmental Information, 2001. https://doi.org/10.7289/V5W66HPP.

7. N. N. G. D. Center, "U.S. Coastal Relief Model Vol.4 - Central Gulf of Mexico.," dataset, National Geophysical Data Center, NOAA, NOAA National Centers for Environmental Information, 2001. https://doi.org/10.7289/V54Q7RW0.

8. N. N. G. D. Center, "U.S. Coastal Relief Model Vol.5 - Western Gulf of Mexico.," dataset, National Geophysical Data Center, NOAA, NOAA National Centers for Environmental Information, 2001. https://doi.org/10.7289/V5QJ7F79.

9. P. J. Martin, C. N. Barron, L. F. Smedstad, T. J. Campbell, A. J. Wallcraft, R. C. Rhodes, C. Rowley, T. L. Townsend, and S. N. Carroll, "User's manual for the Navy Coastal Ocean Model (NCOM) version 4.0," NAVAL RESEARCH LAB STENNIS SPACE CENTER MS OCEAN DYNAMICS AND PREDICTION BRANCH, 2009.

10. A. M. Penko, S. S. Schoenauer, and K. Hall, "Automated Method to Extract Oceanographic and Atmospheric Data from Online Sources.," NRL Memorandum Report NRL/7320/MR–2022/1, 5698, US Naval Research Laboratory, Stennis Space Center, MS, 2022.

11. J. Veeramony, S. Schoenauer, A. Penko, and K. Edwards, "Delft3DFM-Tools," 2022. URL `http://vmlud010.nrlssc.navy.mil/veeramon/delft3dfm-tools`.

12. J. A. Simmons, M. D. Harley, L. A. Marshall, I. L. Turner, K. D. Splinter, and R. J. Cox, "Calibrating and assessing uncertainty in coastal numerical models," *Coastal Engineering* **125**, 28–41 (2017).

13. A. Penko, "instantiate_dfm," 2022. URL `https://gitlab.hpc.mil/allison.penko/instantiate_dfm`.

14. "Miniconda - conda documentation," `https://docs.conda.io/en/latest/miniconda.html`. Accessed: 2022-09-25.

Appendix A

PYTHON CODE FOR BOUNDARY CONDITION AND METEOROLOGICAL FORCING CREATION

A.1 generate_BCs_NCOM.py

The following section contains the `generate_BCs_NCOM.py` Python code used to create lateral boundary condition files, <*.bc>.

```
#!/usr/bin/env python
# coding: utf-8

# version 0.1   2020/01/08  -- Cody Johnson

"""
# Updates:
# 2021/09/05 - Jay Veeramony
#              Added ability to output water level boundary conditions

Issues identified:

1. If the boundary point in Delft3D-FM is a land point in the NCOM/HYCOM output
due to resolution issues, this routine throws an error. There needs to be a
check for this
"""

#import sys
from pathlib import Path

import argparse
import cartopy.crs as ccrs
#import cartopy.feature as cfeature
import cmocean.cm as cmo
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import xarray as xr
import sys

### functions ###
def write_wl_for_pli(bc_fn, gcm, pli, quantity):
    """
    append or write 2D boundary conditions for water level

    bc_fn = path to write or append boundary condition data
    gcm = general circulation model output which contains boundary points (xr.
    DataArray)
    pli = table of boundary support points (pd.DataFrame)
    quantity = variable to output to the BC files (salinity or water_temp)
    depth_avg = flag to enable depth averaging
```

```
    """

    with open(bc_fn, "a") as f:

        gcm_refd, ref_date = assign_seconds_from_refdate(gcm)

        for _, (x_pli, y_pli, pli_point_name) in pli.iterrows():

            if x_pli < 0:
                x_pli_east = x_pli + 360
            else:
                x_pli_east = x_pli

            bc_data = waterlevel_to_pli_point(gcm_refd, x_pli_east, y_pli)

            write_wl_record(f, bc_data, pli_point_name, ref_date)

def write_wl_record(f, bc_data, pli_point_name, ref_date):
    """
    append or write time series water level boundary conditions

    f = file descriptor for writing boundary condition output
    bc_data = data at with percent bed coords (xr.DataFrame)
    pli_point_name = name for entry
    ref_date = date used to calculate offset of the record in seconds
    """

    # write a record
    f.write("[forcing]\n")
    f.write(f"Name                            = {pli_point_name}\n")
    f.write("Function                        = timeseries\n")
    f.write("Time-interpolation              = linear\n")
    f.write("Quantity                        = time\n")
    f.write(f"Unit                            = seconds since {ref_date}\n")
    f.write("Quantity                        = waterlevelbnd\n")
    f.write("Unit                            = m\n")

    # write data after converting to dataframe and iterating over the rows
    for td, value in bc_data.to_dataframe()['surf_el'].iteritems():
        value = f"{value:.03f}"
        f.write(f"{td} {value}\n")

    f.write("\n")


def write_t3d_for_pli(bc_fn, gcm, pli, quantity, nlevels, depth_avg):
    """
    append or write 3D boundary conditions for quantities

    bc_fn = path to write or append boundary condition data
    gcm = general circulation model output which contains boundary points (xr.
    DataArray)
    pli = table of boundary support points (pd.DataFrame)
    quantity = variable to output to the BC files (salinity or water_temp)
    depth_avg = flag to enable depth averaging
    """
    with open(bc_fn, "a") as f:
```

```
98
99          gcm_refd, ref_date = assign_seconds_from_refdate(gcm)
100
101         for _, (x_pli, y_pli, pli_point_name) in pli.iterrows():
102
103             if x_pli < 0:
104                 x_pli_east = x_pli + 360
105             else:
106                 x_pli_east = x_pli
107
108             if (quantity == "salinity") or (quantity == "water_temp"):
109                 bc_data = interpolate_to_pli_point(
110                     gcm_refd, quantity, x_pli_east, y_pli, nlevels
111                 )
112
113                 if depth_avg:
114                     write_ts2d_record(f, bc_data, pli_point_name, quantity, ref_date)
115                 else:
116                     write_ts3d_record(f, bc_data, pli_point_name, quantity, ref_date)
117
118             # in the case of velocity both components are interpolated
119             elif quantity == "velocity":
120                 bc_data = interpolate_to_pli_point(
121                     gcm_refd, ["water_u", "water_v"], x_pli_east, y_pli, nlevels
122                 )
123                 if depth_avg:
124                     write_vector_2d_record(f, bc_data, pli_point_name, quantity,
    ref_date)
125                 else:
126                     write_vector_3d_record(f, bc_data, pli_point_name, quantity,
    ref_date)
127
128 def write_vector_2d_record(f, bc_data, pli_point_name, quantity, ref_date):
129     """
130     append or write time series boundary conditions for depth averaged velocity
131
132     f = file descriptor for writing boundary condition output
133     bc_data = data at with percent bed coords (xr.DataFrame)
134     pli_point_name = name for entry
135     quantity = variable to output to the BC files
136     ref_date = date used to calculate offset of the record in seconds
137     """
138
139     # get units for quantity
140     if quantity == "velocity":
141         vector = "uxuyadvectionvelocitybnd:ux,uy"
142         quantbndx = "ux"
143         quantbndy = "uy"
144         x_comp = "water_u"
145         y_comp = "water_v"
146         units = "-"
147     else:
148         print('quantity should be "velocity"\n')
149         raise ValueError
150
151     # write a record
152     f.write("[forcing]\n")
```

```
    f.write(f"Name                            = {pli_point_name}\n")
    f.write("Function                        = timeseries\n")
    f.write("Time-interpolation              = linear\n")
    f.write("Quantity                        = time\n")
    f.write(f"Unit                            = seconds since {ref_date}\n")
    f.write(f"Vector                          = {vector}\n")
    f.write(f"Quantity                        = {quantbndx}\n")
    f.write(f"Unit                            = {units}\n")
    f.write(f"Quantity                        = {quantbndy}\n")
    f.write(f"Unit                            = {units}\n")

    # write data after converting to dataframe and iterating over the rows
    for td, values in (
        bc_data.to_dataframe()[[x_comp, y_comp]].unstack().iterrows()
    ):
        x_comp_val = values.water_u.mean()
        y_comp_val = values.water_v.mean()

        #values_str = [ f'{x_comp_val:0.2f}' ' ' f'{y_comp_val:.02f}']
        f.write(f"{td} {x_comp_val:0.2f} {y_comp_val:.02f}\n")

    f.write("\n")

def write_ts2d_record(f, bc_data, pli_point_name, quantity, ref_date):
    """
    append or write time series boundary conditions for depth averaged quantities

    f = file descriptor for writing boundary condition output
    bc_data = data at with percent bed coords (xr.DataFrame)
    pli_point_name = name for entry
    quantity = variable to output to the BC files
    ref_date = date used to calculate offset of the record in seconds
    """

    # get units for quantity
    if quantity == "salinity":
        quantbnd = "salinitybnd"
        units = "ppt"
    elif quantity == "water_temp":
        quantbnd = "temperaturebnd"
        units = "deg_C"
    else:
        print('quantity needs to be either "salinity" or "water_temp"\n')
        raise ValueError

    # write a record
    f.write("[forcing]\n")
    f.write(f"Name                            = {pli_point_name}\n")
    f.write("Function                        = t3d\n")
    f.write("Quantity                        = time\n")
    f.write(f"Unit                            = seconds since {ref_date}\n")
    f.write(f"Quantity                        = {quantbnd}\n")
    f.write(f"Unit                            = {units}\n")

    # write data after converting to dataframe and iterating over the rows
    for td, values in bc_data.to_dataframe()[quantity].unstack().iterrows():

```

```
        # take mean of values to get depth averaged
        value = values.mean()

        # see results of interpolation
        if value > 100.0:
            print(
                f"Problem with {quantity} exceeding maximum allowed value: {values.max
    ():.03f} ppt."
            )
        elif value < 0.0:
            print(
                f"Problem with {quantity} becoming negative: {values.max():.03f} ppt."
            )
            print(f"Negative value for {quantity} has been set to 0.01 {units}.")
            value = 0.01

        value = f"{value:.02f}"
        f.write(f"{td} {value}\n")

    f.write("\n")


def write_vector_3d_record(f, bc_data, pli_point_name, quantity, ref_date):
    """
    append or write 3D boundary conditions for quantities

    f = file descriptor for writing boundary condition output
    bc_data = data at with percent bed coords (xr.DataFrame)
    pli_point_name = name for entry
    quantity = variable to output to the BC files
    ref_date = date used to calculate offset of the record in seconds
    """

    if quantity == "velocity":
        vector = "uxuyadvectionvelocitybnd:ux,uy"
        quantbndx = "ux"
        quantbndy = "uy"
        x_comp = "water_u"
        y_comp = "water_v"
        units = "-"
    else:
        print('quantity should be "velocity"\n')
        raise ValueError

    # convert percent from bed into formated string
    pos_spec = [f"{perc:.02f}" for perc in bc_data.perc_from_bed.data]
    pos_spec_str = " ".join(pos_spec[::-1])  # reverse order for D3D

    # write a record
    f.write("[forcing]\n")
    f.write(f"Name                            = {pli_point_name}\n")
    f.write("Function                        = t3d\n")
    f.write("Time-interpolation              = linear\n")
    f.write("Vertical position type          = percentage from bed\n")
    f.write(f"Vertical position specification = {pos_spec_str}\n")
    f.write("Vertical interpolation          = linear\n")
    f.write("Quantity                        = time\n")
```

```
    f.write(f"Unit                                = seconds since {ref_date}\n")
    f.write(f"Vector                              = {vector}\n")

    # loop over number of vertical positions
    for vert_pos in range(1, len(pos_spec) + 1):
        f.write(f"Quantity                            = {quantbndx}\n")
        f.write(f"Unit                                = {units}\n")
        f.write(f"Vertical position                   = {vert_pos}\n")

        f.write(f"Quantity                            = {quantbndy}\n")
        f.write(f"Unit                                = {units}\n")
        f.write(f"Vertical position                   = {vert_pos}\n")

    # write data after converting to dataframe and iterating over the rows
    for td, values in (
        bc_data.to_dataframe()[[x_comp, y_comp]].unstack().iterrows()
    ):

        # get componets as array in order to format for d3d input
        x_comp_vals = values[x_comp].values[::-1]  # reverse order for D3D
        y_comp_vals = values[y_comp].values[::-1]  # reverse order for D3D
        values = [
            f"{x_comp_val:.02f} {y_comp_val:.03f}"
            for x_comp_val, y_comp_val in zip(x_comp_vals, y_comp_vals)
        ]
        values_str = " ".join(values)
        f.write(f"{td} {values_str}\n")

    f.write("\n")


def write_ts3d_record(f, bc_data, pli_point_name, quantity, ref_date):
    """
    append or write 3D boundary conditions for quantities

    f = file descriptor for writing boundary condition output
    bc_data = data at with percent bed coords (xr.DataFrame)
    pli_point_name = name for entry
    quantity = variable to output to the BC files
    ref_date = date used to calculate offset of the record in seconds
    """

    # get units for quantity
    if quantity == "salinity":
        quantbnd = "salinitybnd"
        units = "ppt"
    elif quantity == "water_temp":
        quantbnd = "temperaturebnd"
        units = "deg_C"
    else:
        print('quantity needs to be either "salinity" or "water_temp"\n')
        raise ValueError

    # convert percent from bed into formated string
    pos_spec = [f"{perc:.02f}" for perc in bc_data.perc_from_bed.data]
    pos_spec_str = " ".join(pos_spec[::-1])  # reverse order for D3D

```

```
    # write a record
    f.write("[forcing]\n")
    f.write(f"Name                                = {pli_point_name}\n")
    f.write("Function                            = t3d\n")
    f.write("Time-interpolation                  = linear\n")
    f.write("Vertical position type              = percentage from bed\n")
    f.write(f"Vertical position specification = {pos_spec_str}\n")
    f.write("Vertical interpolation              = linear\n")
    f.write("Quantity                            = time\n")
    f.write(f"Unit                                = seconds since {ref_date}\n")

    # loop over number of vertical positions
    for vert_pos in range(1, len(pos_spec) + 1):
        f.write(f"Quantity                                = {quantbnd}\n")
        f.write(f"Unit                                    = {units}\n")
        f.write(f"Vertical position                       = {vert_pos}\n")

    # write data after converting to dataframe and iterating over the rows
    for td, values in bc_data.to_dataframe()[quantity].unstack().iterrows():

        # see results of interpolation
        if values.max() > 100.0:
            print(
                f"problem with {quantity} exceeding maximum allowed value: {values.max
    ():.03f} ppt"
            )
        elif values.min() < 0.0:
            print(f"problem with {quantity} becoming negative: {values.max():.03f} ppt
    ")
            print(f"Negative values for {quantity} has been set to 0.01 {units}.")
            values.where(values > 0.01, 0.01, inplace=True)

        values = [f"{value:.02f}" for value in values]
        values_str = " ".join(values[::-1])  # reverse order for D3D
        f.write(f"{td} {values_str}\n")

    f.write("\n")


def assign_seconds_from_refdate(gcm):
    """
    This func assigns seconds from a user specified ref date as coords.
    This is how D3D interpolates the boundary conditions in time.

    gcm = model output to add coords to
    """
    ref_date = gcm.time.data[0]
    ref_dt = pd.to_datetime(ref_date)
    ref_date_str = ref_dt.strftime("%Y-%m-%d %H:%M:%S")
    timedeltas = pd.to_datetime(gcm.time.data) - ref_dt
    seconds = timedeltas.days * 24 * 60 * 60 + timedeltas.seconds
    gcm = gcm.assign_coords(coords={"seconds_from_ref": ("time", seconds)})
    return gcm.swap_dims({"time": "seconds_from_ref"}), ref_date_str


def interpolate_to_pli_point(gcm_refd, quantity, x_pli_east, y_pli, nlevels):
    """interpolates the quanitites to the sigma depths and pli coords
```

```
    gcm_refd = gcm with new time coordinates
    quantity = variable to output to the BC files (salinity or water_temp)
    x_pli_east = longitude of pli point in degrees east from meridian (NCOM convention)
    y_pli = latitude
    """

    # interpolate to pli point and drop data below bed level at nearest gcm_refd point
    bc_data = (
        gcm_refd[quantity]
        .interp(lon=x_pli_east, lat=y_pli)
        .dropna(dim="depth")
        .squeeze()
    )

    # add coordinate for percent from bed. D3D uses this in its bc file format
    gcm_refd_zb = bc_data.depth[-1]  # get bed level of gcm_refd point
    if nlevels > 1:
        sigma_lvls = np.arange(nlevels)/(nlevels-1)*100
        zlvls = sigma_lvls[:]*gcm_refd_zb.data/100
        bc_data = bc_data.interp(depth=zlvls)

    perc_from_bed = 100 * (-1 * bc_data.depth + gcm_refd_zb) / gcm_refd_zb
    bc_data = bc_data.assign_coords(coords={"perc_from_bed": ("depth", perc_from_bed.data)})

    return bc_data

def waterlevel_to_pli_point(gcm_refd, x_pli_east, y_pli):
    """interpolates the quanitites to the sigma depths and pli coords

    gcm_refd = gcm with new time coordinates
    x_pli_east = longitude of pli point in degrees east from meridian (NCOM convention)
    y_pli = latitude
    """

    # interpolate to pli point and drop data below bed level at nearest gcm_refd point
    bc_data = (
        gcm_refd['surf_el']
        .interp(lon=x_pli_east, lat=y_pli)
        .squeeze()
    )

    return bc_data


### main loop ###
if __name__ == "__main__":

    ### arguments ###
    parser = argparse.ArgumentParser()
    parser.add_argument(
        "nc",
        help="NCOM NetCDF output containing boundary support points and duration of Delft3D simulation",
```

```
    )
    parser.add_argument(
        "quantity",
        help='NCOM variable. Must be either "salintiy", "water_temp", or "velocity"',
    )
    parser.add_argument(
        "--pli-list",
        nargs="*",
        type=str,
        help="list of boundary support point polyline filenames",
        required=True,
        dest="pli_list",
    )
    parser.add_argument(
        "--nlevels",
        default=11,
        type=int,
        help="Number of vertical levels",
        dest="nlevels",
    )
    parser.add_argument(
        "--bc-filename",
        help="Optional filename for Delft3D boundary condition filename",
        type=str,
        dest="bc_filename",
    )
    parser.add_argument(
        "--depth-avg",
        help="flag to enable depth averaged output",
        default=False,
        action="store_true",
        dest="depth_avg",
    )
    parser.add_argument(
        "--plot",
        help="flag to enable plotting",
        default=False,
        action="store_true",
        dest="plot",
    )

    args = parser.parse_args()

    gcm = args.nc
    quantity = args.quantity
    pli_list = args.pli_list
    depth_avg = args.depth_avg
    plot = args.plot
    nlevels = args.nlevels

    # validate arguments
    if (
        (quantity != "salinity")
        and (quantity != "water_temp")
        and (quantity != "velocity")
        and (quantity != "surf_el")
    ):
```

```
        print(
            f'<quantity> was specfied as {quantity}, but should be either "salinity", "water_temp", "velocity" or "surf_el".'
        )
        raise ValueError

    # open gcm NetCDF output as Xarray dataset
    try:
        gcm = xr.open_dataset(Path(gcm), drop_variables="tau")
    except FileNotFoundError as e:
        print("f<NCOM output> should be path to NCOM NetCDF output")
        raise e

    # Some netcdf files have "latitude" and "longitude" instead of "lat" and "lon"
    # In such cases, rename the indices
    try:
        gcm = gcm.rename({'latitude':'lat', 'longitude':'lon'})
    except ValueError:
        pass

    # set default boundary condition filename depending on quantity
    bc_fn = args.bc_filename
    if bc_fn == None:
        if quantity == "salinity":
            bc_fn = Path("Salinity.bc")
        elif quantity == "water_temp":
            bc_fn = Path("Temperature.bc")
        elif quantity == "velocity":
            bc_fn = Path("Velocity.bc")
        elif quantity == "surf_el":
            bc_fn = Path("Waterlevel.bc")

    # Overwrite the file on the first instance (i.e., clean up whatever is in the file)
    # There is probably a more elegant way to do this?
    with open(bc_fn, 'w') as f:
        f.write("")

    # pli files opened as Pandas DataFrames
    pli_points = []
    for pli_fn in pli_list:
        print(f"Reading in file: {pli_fn}")
        pli = pd.read_csv(
            pli_fn, sep="\s+", skiprows=2, header=None, names=["x", "y", "point_id"]
        )

        # Check if there are points in the pli file outside the lat/lon extent in the
        # input file. If the points are outside the domain in the provided netcdf file
        # this will either throw Nan's (for water level BCs) or cause interpolation
        # errors (for the other boundary conditions)
        pli_lat, pli_lon = pli["y"].values, pli["x"].values+360
        latmin, latmax = min(gcm.coords["lat"].values), max(gcm.coords["lat"].values)
        lonmin, lonmax = min(gcm.coords["lon"].values), max(gcm.coords["lon"].values)
        if  (any(pli_lon < lonmin) or any(pli_lon > lonmax) or
             any(pli_lat < latmin) or any(pli_lat > latmax)):
            if (all(pli_lon < lonmin) or all(pli_lon > lonmax) or
                 all(pli_lat < latmin) or all(pli_lat > latmax)):
```

```
                print("All points in the pli file are outside the domain in the netcdf
    file")
                print("Please fix before continuing.")
                print("Exiting routine...")
            else:
                print("Some points in the pli file are outside the domain in the
    netcdf file")
                print("Please remove these from the pli before continuing.")
                print("Exiting routine...")
                for i in range(pli_lon.size):
                    if (pli_lon[i] < lonmin or pli_lon[i] > lonmax or
                        pli_lat[i] < latmin or pli_lat[i] > latmax):
                        print(f"{pli_lon[i]-360:0.05f}, {pli_lat[i]:.05f} ")
            sys.exit()

        # Move forward if all points in the pli file are inside the domain
        if quantity == "surf_el":
            write_wl_for_pli(bc_fn, gcm, pli, quantity)
        else:
            write_t3d_for_pli(bc_fn, gcm, pli, quantity, nlevels, depth_avg=depth_avg)

        # add points to list for visualization
        pli_points.append(pli)

    # concat pli points
    pli_points = pd.concat(pli_points)



    ### visualization ###
    # color map depending on quantity
    if plot:
        matplotlib.use('Agg')
        if quantity == "salinity":
            cmap = cmo.haline
        elif quantity == "water_temp":
            cmap = cmo.thermal
        elif quantity == "velocity":
            cmap = "jet"
        elif quantity == "surf_el":
            cmap = "jet"

        # setup orthographic projection for geographic data
        fig, ax = plt.subplots(
            1, 1, subplot_kw={"projection": ccrs.PlateCarree()}, figsize=(16, 9)
        )
        ax.gridlines(draw_labels=True)
        ax.coastlines(resolution='50m')

        # plot initial quantity at surface
        if (quantity == "salinity") or (quantity == "water_temp"):
            gcm[quantity].isel(time=0, depth=0).plot(
                ax=ax, transform=ccrs.PlateCarree(), cmap=cmap
            )
        elif quantity == "velocity":
            tmp = gcm.isel(time=0, depth=0)
            tmp["magnitude"] = np.sqrt(tmp["water_u"] ** 2 + tmp["water_v"] ** 2)
```

```
            tmp["magnitude"].plot(
                ax=ax,
                transform=ccrs.PlateCarree(),
                cmap=cmap,
                cbar_kwargs={"label": "velocity magnitude [m/s]"},
            )
        elif quantity == "surf_el":
            gcm[quantity].isel(time=0).plot(
                ax=ax, transform=ccrs.PlateCarree(), cmap=cmap
            )


        # add coastline for reference
        # ax.add_feature(cfeature.COASTLINE, edgecolor="0.3")

        # boundary condition support points
        pli_points.plot.scatter(
            "x", "y", marker="x", color="k", ax=ax, transform=ccrs.PlateCarree()
        )

        fig.savefig("point_output_locations.png", bbox_inches="tight")
```

A.2 coamps2meteo.py

The following section contains the `coamps2meteo.py` Python code used to create meteorological forcing files, <.amp>, <.amu>, and <.amv>.

```
# -*- coding: utf-8 -*-
"""
    Given a region, start date and end date creates meteorological forcing
    files from COAMPS

    Usage:
    $ python coamps2meteo.py --region COAMPSREGION --start-date start_date
    --end-date end_date --datadir COAMPS_DATADIR
    date format: YYYYMMDD

    Examples:

        $ python coamps2meteo.py --region eqam --start-date 20190101 --end-date
        20190107 --datadir /u/NOGAPS/COAMPSg

        $ python coamps2meteo.py -r eqam -s 20190101 -e 20190107 -d
        /u/NOGAPS/COAMPSg
"""
# Created By: Jay Veeramony
# Version: 1.1
# Created on: Unknown
# Last modified: 2022-07-22
#     Various modifications to bring code in line with python best practice
#     Combined routines


import argparse
import logging
import os
```

```
import time
import sys
import shutil

from datetime import date, datetime, timedelta
from pathlib import Path
from subprocess import call

import numpy as np
from scipy.io import loadmat


def list_inputfiles(coamps_region, sdtg, edtg, ddir, component):
    """
    Creates list of input files that are read from the COAMPS database

    Args:
        coamps_region (str): Model region from which data is needed
        sdtg (str): Start date for data retrieval (YYYYMMDD format)
        edtg (str): End date for data retrieval (YYYYMMDD format)
        ddir (str): Data directory
        component: Meteorological component to retrieve, valid values are
                    "pres", "wnd_ucmp", "wnd_vcmp"

    Returns:
        List of files that contain the data. These files could be gzipped.

    """
    sdtg = datetime.strptime(sdtg, '%Y%m%d')
    edtg = datetime.strptime(edtg, '%Y%m%d')

    ddir = Path(ddir).resolve()/coamps_region

    dirs = list(ddir.glob('20*'))
    print("Type of dirs:", type(dirs))
    print("Size of dirs:", len(dirs))

    # check if directory is empty (likely a typhoon mounting error)
    if not dirs:
        print('dirs is empty')
        sys.exit('directory empty or nonexistant. Check mount of typhoon')

    # initial file
    fileslist = list(dirs[0].glob(coamps_region+'_'+component+'.*'))

    # rest of files
    for idir in range(1, len(dirs)):
        fileslist.extend(dirs[idir].glob(coamps_region+'_'+component+'.*'))

    file_dates = list()
    for filename in fileslist:
        file_dates.append(str(filename).split('.')[1])

    #
    # sort files and dates, necessary bc zipped files
    #
    file_dates = np.asarray(file_dates, dtype='int')
```

```
    idx = np.argsort(file_dates)
    file_dates = file_dates[idx]

    fileslist = np.array(fileslist)
    fileslist = fileslist[idx]

    tstart = int(sdtg.strftime('%Y%m%d%H'))
    tend = int(edtg.strftime('%Y%m%d%H'))

    idx = (file_dates >= tstart)*(file_dates <= tend)
    return fileslist[idx]


def append_met_data(met_file, ref_time, datafile, nlon, nlat):
    """
    Appends the data to the meteorological file

    Args:
        met_file: file being written
        ref_time: Reference time
        datafile: file from list of files between start/end date
        nlon, nlat: size of data array (obtained from model info)

    """
    print('reading file: ', datafile)
    if str(datafile).endswith('gz'):
        print('zipped file to be copied locally/unzipped')
        zipfile = os.path.basename(files[0])
        shutil.copyfile(files[0], './'+zipfile)
        call(['gunzip', zipfile])

        # reassign datafile to just unzipped file
        cols = zipfile.split('.')
        datafile = './'+cols[0]+'.'+cols[1]

        file_time = cols[1]
        time_array = datetime.strptime(file_time, '%Y%m%d%H') + \
            np.arange(12) * timedelta(hours=1)
        data = read_coamps(datafile, nlon, nlat, 12)

        with open(metfile, 'a') as fid:
            for i in range(len(time_array)):
                tc = time.mktime(time_array[i].timetuple()) - \
                        time.mktime(ref_time.timetuple())
                logging.info('TIME = %.3f hours since %s # %s', tc/3600,
                             ref_time.strftime('%Y%m%d%H'),
                             time_array[i].strftime('%Y%m%d%H'))
                fid.write('TIME = {0} hours since {1} # {2}'.
                          format(tc/3600, ref_time, time_array[i]))
                fid.write('\n')
                for m in reversed(range(data.shape[1])):
                    for j in data[i, m, :]:
                        fid.write('%.2f ' % j)
                    fid.write('\n')

        os.remove(datafile)
    else:
```

```
        file_time = str(datafile).split('.')[-1]
        time_array = datetime.strptime(file_time, '%Y%m%d%H') + \
            np.arange(12) * timedelta(hours=1)
        data = read_coamps(datafile, nlon, nlat, 12)
        with open(met_file, 'a') as fid:
            for i in range(len(time_array)):
                tc = time.mktime(time_array[i].timetuple()) - \
                     time.mktime(ref_time.timetuple())
                logging.info('TIME = %.3f hours since %s # %s',
                             tc/3600, ref_time.strftime('%Y%m%d%H'),
                             time_array[i].strftime('%Y%m%d%H'))
                fid.write('TIME = {0} hours since {1} # {2}'.
                          format(tc/3600, ref_time, time_array[i]))
                fid.write('\n')
                for m in reversed(range(data.shape[1])):
                    for j in data[i, m, :]:
                        fid.write('%.2f ' % j)
                    fid.write('\n')


def get_modelinfo(area_name):
    '''
    Given the name of an area with met data, extract information about the grid
    such as its location, grid size and grid resolution

    Args:
        area_name(str): COAMPS model region

    Returns:
        sdir (str): Sub-directory name to check for data.
        ssdir (str) : Sub-sub directory name
        nx (int) : Number of longitude points on the grid.
        ny (int) : Number of latitude points on the grid.
        minlat (float) : Origin - latitude.
        minlon (float) : Origin - longitude.
        maxlat (float) : max extent of latitude.
        maxlon (float) : max extent of longitude.

    '''
    path = os.path.dirname(os.path.abspath(__file__))
    area = loadmat(path+'/modelinfo.mat')  # load mat-file
    mdata = area['modelinfo']  # variable in mat file
    mdtype = mdata.dtype  # dtypes of structures are "unsized objects"

    # * SciPy reads in structures as structured NumPy arrays of dtype object *
    # The size of the array is the size of the structure array, not the number
    # elements in any particular field. The shape defaults to 2-dimensional.  *
    # For convenience make a dictionary of the data using the names from dtype

    ndata = {n: mdata[0][n] for n in mdtype.names}

    # When reading in the Matlab mat file, the different integers and floats
    # have varying dtypes, presumably because Matlab uses the lowest size
    # needed to store the data to save space. Make them all uniform size here
    # for convenience

    ndata['nx'] = ndata['nx'].astype('int')
```

```
    ndata['ny'] = ndata['ny'].astype('int')
    ndata['minlat'] = ndata['minlat'].astype('float64')
    ndata['minlon'] = ndata['minlon'].astype('float64')
    ndata['maxlat'] = ndata['maxlat'].astype('float64')
    ndata['maxlon'] = ndata['maxlon'].astype('float64')

    # Get line number where data is to be retrieved from
    iloc = np.squeeze(np.where(ndata['model1'] == area_name))

    sdir = str(np.squeeze(ndata['model1'][iloc]))
    ssdir = str(np.squeeze(ndata['model2'][iloc]))
    # nx = ndata['nx'][iloc]
    # ny = ndata['ny'][iloc]
    # minlat = ndata['minlat'][iloc]
    # minlon = ndata['minlon'][iloc]
    # maxlat = ndata['maxlat'][iloc]
    # maxlon = ndata['maxlon'][iloc]
    # return sdir, ssdir, nx, ny, minlat, minlon, maxlat, maxlon
    area_info = {'sdir': sdir,
                 'ssdir': ssdir,
                 'nx': ndata['nx'][iloc],
                 'ny': ndata['ny'][iloc],
                 'minlat': ndata['minlat'][iloc],
                 'minlon': ndata['minlon'][iloc],
                 'maxlat': ndata['maxlat'][iloc],
                 'maxlon': ndata['maxlon'][iloc]}
    return area_info


def read_coamps(data_file, nlon, nlat, ntime):
    """
    Given a binary file with meteorology data in /u/NOGAPS,
    reads it into a data array Parameters

    Args:
        data_file (str): File containing binary data.
        nlon (int) : Number of longitude locations.
        nlat (int) : Number of latitude locations.
        ntime (int) : Number of taus.

    Returns:
        data (float) : Matrix of size (nlon, nlat, ntime) containing the data.
    """
    header = np.zeros(ntime)
    footer = np.zeros(ntime)
    data = np.zeros((nlon, nlat, ntime))

    with open(data_file, 'rb') as fid:
        for j in range(ntime):
            header[j] = np.fromfile(fid, dtype='>u4', count=1)
            local_data = np.fromfile(fid, dtype='>f4', count=nlon*nlat)
            footer[j] = np.fromfile(fid, dtype='>u4', count=1)

            # Reshape the matrix in fortran order
            local_data = np.reshape(local_data, (nlon, nlat), order='F')
            data[:, :, j] = local_data

```

```
        if header[0] != footer[0]:
            print('Header ' + str(header[0]))
            print('Footer ' + str(footer[0]))
            print('File seems to be corrupt or read order is wrong')

        # To keep it same as the openearth tools structure, permute
        # the array to have time loop on the first dimension and
        # longitude on the third dimension
        data = data.transpose(2, 1, 0)

        return data


def write_met_header(met_var, met_file, nlon, nlat, minlon, maxlon,
                     minlat, maxlat):
    """
    Writes header for meteorological file

    """
    dlon = (maxlon - minlon)/(nlon - 1)
    dlat = (maxlat - minlat)/(nlat - 1)

    with open(met_file, 'w') as fid:
        fid.write('### START OF HEADER\n')
        fid.write('FileVersion      = 1.03\n')
        fid.write('filetype         = meteo_on_equidistant_grid\n')
        fid.write('n_cols           = %5d \n' % nlon)
        fid.write('n_rows           = %5d \n' % nlat)
        fid.write('grid_unit        = ' + 'deg' + '\n')
        fid.write('x_llcorner       = %5.2f \n' % minlon)
        fid.write('y_llcorner       = %5.2f \n' % minlat)
        fid.write('dx               = %5.2f \n' % (dlon))
        fid.write('dy               = %5.2f \n' % (dlat))
        fid.write('n_quantity       =    1  \n')

        if met_var == 'pres':
            fid.write('quantity1        = ' + 'air_pressure' + '\n')
            fid.write('unit             = ' + 'Pa' + '\n')
            nodata_value = 101300.0
        elif met_var == 'wnd_ucmp':
            fid.write('quantity1        = ' + 'x_wind' + '\n')
            fid.write('unit             = ' + 'm/s' + '\n')
            nodata_value = 0.0
        elif met_var == 'wnd_vcmp':
            fid.write('quantity1        = ' + 'y_wind' + '\n')
            fid.write('unit             = ' + 'm/s' + '\n')
            nodata_value = 0.0
        else:
            sys.exit('Incorrect variable names')

        fid.write('NODATA_value     = %.2f \n' % (nodata_value))
        fid.write('### END OF HEADER\n')


"""
Start of the main routing
"""
```

```
if __name__ == "__main__":
    valid_choices = ['arctic', 'cencoos_n3', 'e_pac_comp', 'europe2',
                     'hawaii_n3', 'mbay_n4', 'n_ind2', 'southwest_asia',
                     'useast', 'cen_amer', 'centam', 'eqam', 'europe3',
                     'indo', 'mideast', 'nwatl', 'southwest_asia2', 'was3',
                     'cen_america', 'centio', 'eqam_comp', 'fukushima',
                     'mbay_n1', 'MREA04', 'nwpac', 'southwest_asia2_n3',
                     'w_atl', 'cencoos_n1', 'eafr', 'euro', 'hawaii',
                     'mbay_n2', 'nepac', 'socal', 'southwest_asia3', 'w_pac',
                     'cencoos_n2 ', 'e_pac', 'europe', 'hawaii_comp',
                     'mbay_n3', 'n_ind', 'somalia', 'test2', 'w_pac2']

    parser = argparse.ArgumentParser()
    parser.add_argument("-r", "--region",
                        type=str,
                        help="COAMPS region of interest",
                        choices=valid_choices,
                        required=True)
    parser.add_argument("-s", "--start-date",
                        type=str,
                        help="data start date YYYYMMDD",
                        required=True,
                        dest="str_date")
    parser.add_argument("-e", "--end-date",
                        type=str,
                        help="data end date YYYMMDD",
                        required=True,
                        dest="end_date")
    parser.add_argument("-d", "--datadir",
                        type=str,
                        help="DATADIR = data directory",
                        default='/u/NOGAPS/COAMPSg',
                        required=True,
                        dest="datadir")

    args = parser.parse_args()
    region = args.region
    str_date = args.str_date
    end_date = args.end_date
    if str_date > end_date:
        sys.exit('input error: {} occurs before {}'
                 .format(end_date, str_date))

    datadir = args.datadir

    logfilename = ('coamps_'+region+'_'+str_date+'_'+end_date+'_'
                   + date.today().strftime('%Y%h%d')+'.log')

    logging.basicConfig(filename=logfilename, level=logging.INFO,
                        format='%(levelname)s:%(message)s')
    logging.info('Command line arguments: %s %s %s', sys.argv[1],
                 sys.argv[2], sys.argv[3])
    coampsdata = get_modelinfo(region)
    nx = coampsdata['nx']
    ny = coampsdata['ny']
    minlon = coampsdata['minlon']
    maxlon = coampsdata['maxlon']
```

```
    minlat = coampsdata['minlat']
    maxlat = coampsdata['maxlat']

    # note that 'pres' is only used for getting dates
    files = list_inputfiles(region, str_date, end_date, datadir, 'pres')
    if len(files) > 0:
        print('Available range of files:')
        print(str(files[0]).split('.')[1])
        print(str(files[-1]).split('.')[1])
    else:
        sys.exit('No files found, check dates and region')

    pars = ['pres', 'wnd_ucmp', 'wnd_vcmp']

    for var in pars:
        if var == 'pres':
            metfile = Path('./coamps_'+str_date+'_'+end_date+'_'+region+'.amp')
            write_met_header(var, metfile, nx, ny, minlon, maxlon,
                             minlat, maxlat)
            files = list_inputfiles(region, str_date, end_date, datadir, var)
            # fileTime = files[0].split('.')[1]
            fileTime = str(files[0]).split('.')[1]
            refTime = datetime.strptime(fileTime, '%Y%m%d%H')
            for f in files:
                logging.info('****Reading coamps file %s****', f)
                append_met_data(metfile, refTime, f, nx, ny)
        elif var == 'wnd_ucmp':
            metfile = Path('./coamps_'+str_date+'_'+end_date+'_'+region+'.amu')
            write_met_header(var, metfile, nx, ny, minlon, maxlon,
                             minlat, maxlat)
            files = list_inputfiles(region, str_date, end_date, datadir, var)
            fileTime = str(files[0]).split('.')[1]
            refTime = datetime.strptime(fileTime, '%Y%m%d%H')
            for f in files:
                logging.info('****Reading coamps file %s****', f)
                append_met_data(metfile, refTime, f, nx, ny)
        elif var == 'wnd_vcmp':
            metfile = Path('./coamps_'+str_date+'_'+end_date+'_'+region+'.amv')
            write_met_header(var, metfile, nx, ny, minlon, maxlon,
                             minlat, maxlat)
            files = list_inputfiles(region, str_date, end_date, datadir, var)
            fileTime = str(files[0]).split('.')[1]
            refTime = datetime.strptime(fileTime, '%Y%m%d%H')
            for f in files:
                logging.info('****Reading coamps file %s****', f)
                append_met_data(metfile, refTime, f, nx, ny)
        else:
            sys.exit('Incorrect variable names')
```