Naval Research Laboratory
Washington, DC 20375-5320

NRL/7350/MR—2022/1

Automated Method to Extract Oceanographic and Atmospheric Data from Online Sources

ALLISON M. PENKO

SUNNI SCHOENAUER

Seafloor Sciences Branch
Ocean Sciences Division

KENDAL HALL

Howard University
Washington, DC

October 20, 2022

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From - To)*
20-10-2022	NRL Memorandum Report	31 May 2022 – 05 Aug 2022

4. TITLE AND SUBTITLE

Automated Method to Extract Oceanographic and Atmospheric Data from Online Sources

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Allison M. Penko, Sunni S. Schoenauer, and Kendal Hall*

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--2022/1

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

*Howard University, 2400 6th St NW, Washington, DC 20059

14. ABSTRACT

The US Navy relies on accurate forecasts of the battlespace environment including ocean waves, currents, and water levels near the coast. The forecast models are dependent on the fidelity of their forcing, boundary, and initial conditions. Model developers need automated and efficient methods to obtain data to drive and validate coastal models. This Memorandum Report describes three publicly available online web servers to obtain modeled and observed meteorologic and oceanographic data. A suite of algorithms was developed to extract, reformat, and visualize the data. There is a significant capability gap in the useful dissemination of big data from operational model output. Access to the most recent and reliable data, as well as efficient and automated algorithms, improves the accuracy and fidelity of the Navy's environmental forecasts to the fleet, and advance visualization of the forecasts would provide enhanced information for mission planning.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Allison M. Penko
U	U	U	U	30	**19b. TELEPHONE NUMBER** *(include area code)* (228) 688-4823

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39.18

This page intentionally left blank.

CONTENTS

FIGURES

TABLES

This page intentionally left blank

EXECUTIVE SUMMARY

The US Navy relies on accurate forecasts of the battlespace environment including ocean waves, currents, and water levels near the coast. The forecast models are dependent on the fidelity of their forcing, boundary, and initial conditions. Model developers need automated and efficient methods to obtain data to drive and validate coastal models. This Memorandum Report describes three publicly available online web servers to obtain modeled and observed meteorologic and oceanographic data. A suite of algorithms was developed to extract, reformat, and visualize the data. There is a significant capability gap in the useful dissemination of big data from operational model output. Access to the most recent and reliable data, as well as efficient and automated algorithms, improves the accuracy and fidelity of the Navy's environmental forecasts to the fleet, and advance visualization of the forecasts would provide enhanced information for mission planning.

This page intentionally left blank

AUTOMATED METHOD TO EXTRACT OCEANOGRAPHIC AND ATMOSPHERIC DATA FROM ONLINE SOURCES

1. INTRODUCTION

1.1 Problem and Objective

The US Navy relies on accurate forecasts of the battlespace environment including ocean waves, currents, and water levels near the coast. Model developers at the US Naval Research Laboratory (NRL) need an efficient and automated way to acquire and visualize publicly available global and regional meteorologic and oceanographic (METOC) data to drive and validate models in development. Forecast models are dependent on the fidelity of their forcing, boundary, and initial conditions. The goal of this project was to develop a centralized and efficient approach to extract publicly available METOC data from online sources and reformat data for nearshore high-resolution model forcing and validation.

1.2 Background

NRL researchers develop ocean wave and current models to provide the Navy with information about the operational environment. Global models can simulate the world's ocean dynamics, but at a low resolution not feasible for examining small-scale processes occurring in shallow water near the coast. Often, the Navy needs higher resolution information for a specific operational region. Global model output does not resolve ocean processes at high enough resolution to provide useful information at a tactical scale for mission planning. Therefore, higher resolution models at smaller scales are required for simulating the coastal environment. To address this problem, information from global models (low-resolution) is used to drive regional models with increasing resolution at their boundaries in order to resolve the physical ocean processes at a tactical scale (Figure 1).

A practical example of ocean model nesting is at Duck, NC is shown in Figure 2. In this case, output from the mesoscale basin model (the Navy Coastal Ocean Model - NCOM - US East domain) is needed to drive the mesoscale regional model. Many years of hindcast and nowcast model output of NCOM is publicly available through online web servers. NRL model developers were lacking automated and efficient algorithms to obtain this data in a usable format to generate boundary conditions for high-resolution coastal model forcing. Additionally, these high-resolution models need ground truth data to validate their output. Developers also needed robust algorithms to extract buoy and station observations to compare with and test the model output. This data is also available via publicly available web servers; however, they require an efficient and consistent way to extract specific stations and time periods.

1.3 Naval Relevance

NRL developers need oceanographic data from lower-resolution ocean models to drive higher-resolution coastal models. Accessing this data with efficient and automated algorithms allows for faster capability

Manuscript approved October 17, 2022.

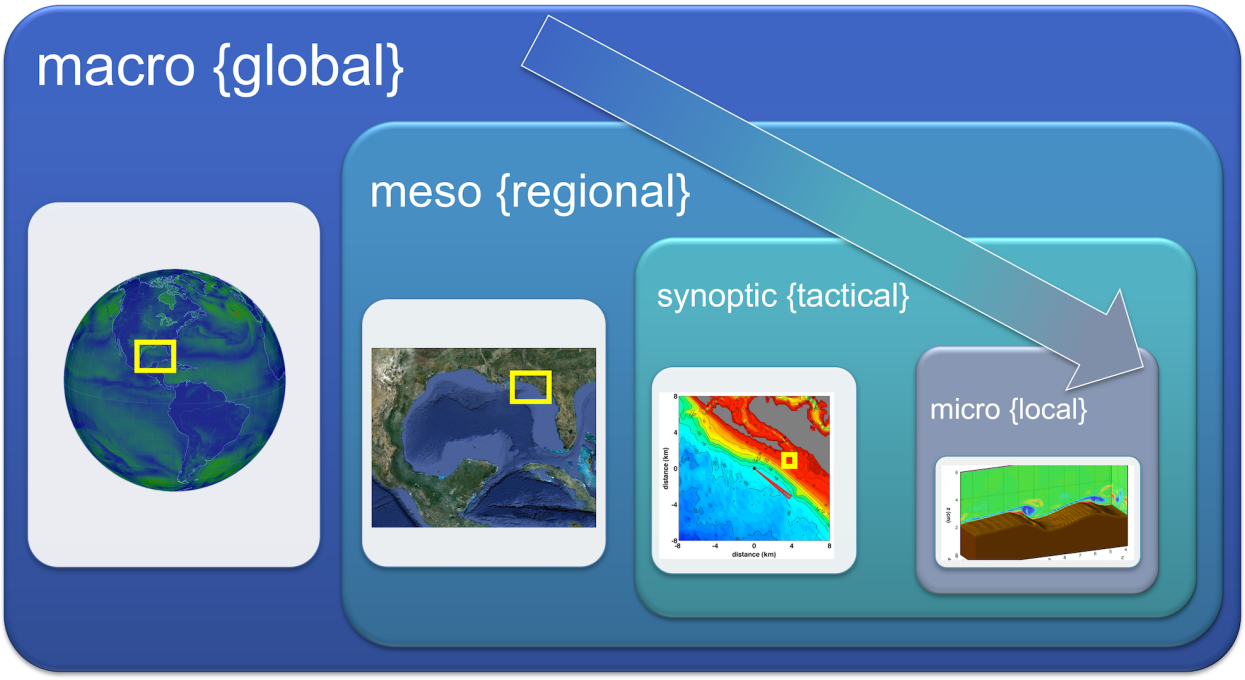

Fig. 1—A graphic depicting the cascade of information from low-resolution models at the macroscale to very high-resolution models at the microscale. Ocean models are typically run by nesting multiple scales where boundary conditions for the higher resolution models are generated from the lower resolution model output.

transition timelines and better visualizations to understand the battlespace environment for mission planning. Additionally, NRL developers need oceanographic observations to evaluate model predictions. Automated, useful, and robust algorithms can decrease the amount of time needed for model development, validation, and testing. These algorithms provide accessible observations that allow for sensitivity testing and optimization of model parameters.

2. METHODOLOGY

A summary of the technical approach taken to accomplish the objective is as follows:

- Create a document detailing available data on three NOAA web servers
- Write a suite of algorithms to download oceanographic data from web servers from simple user input
- Reformat data and save in a consistent and usable format
- Create plots to visualize the extracted data

A flowchart illustrating the approach is shown in Figure 3. First, information about each of the web servers was collected. Next, Python algorithms were written to easily extract data via the command line and a simple user input file and save it into a Network Common Data Format (netCDF). Lastly, plots were generated to visualize the extracted data. The following sections describe in detail the web servers and the code generated to extract the data.

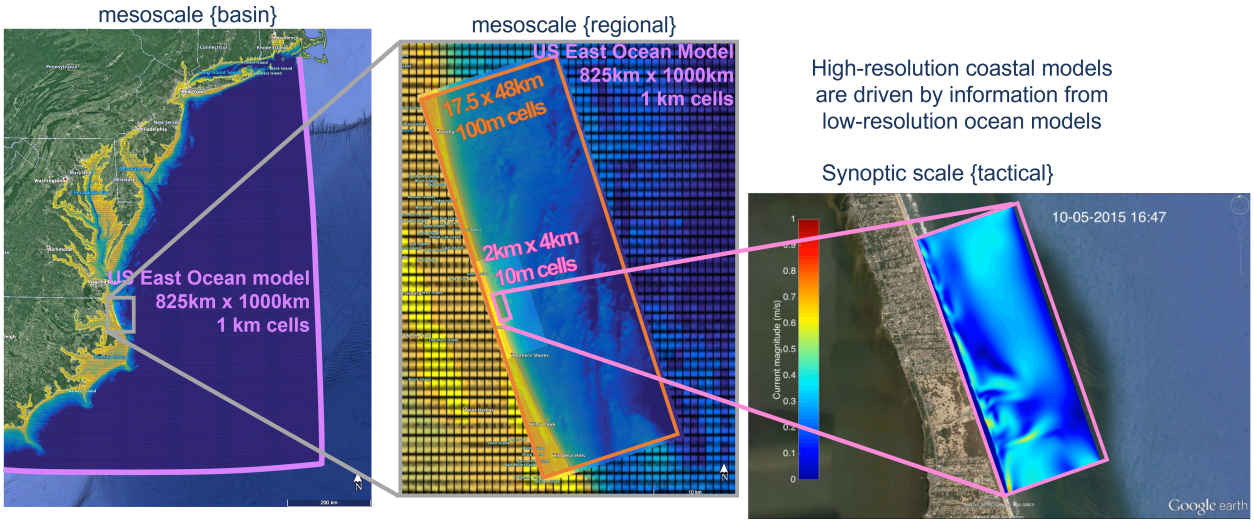

Fig. 2— A mesoscale ocean basin model of the Western Atlantic ocean is driven by waves and currents extracted from a global model of the entire Atlantic ocean at the purple boundaries and outputs waves and currents at 1 km spacing. A mesoscale ocean regional model is driven by waves and currents extracted from the mesoscale US East ocean model at the orange boundaries. A synoptic scale coastal model is driven by waves and currents extracted from the mesoscale regional model at the pink boundaries and can then be used in coastal mission planning. The highest resolution output is a plot of the current magnitude as a large Nor'Easter storm passes by.

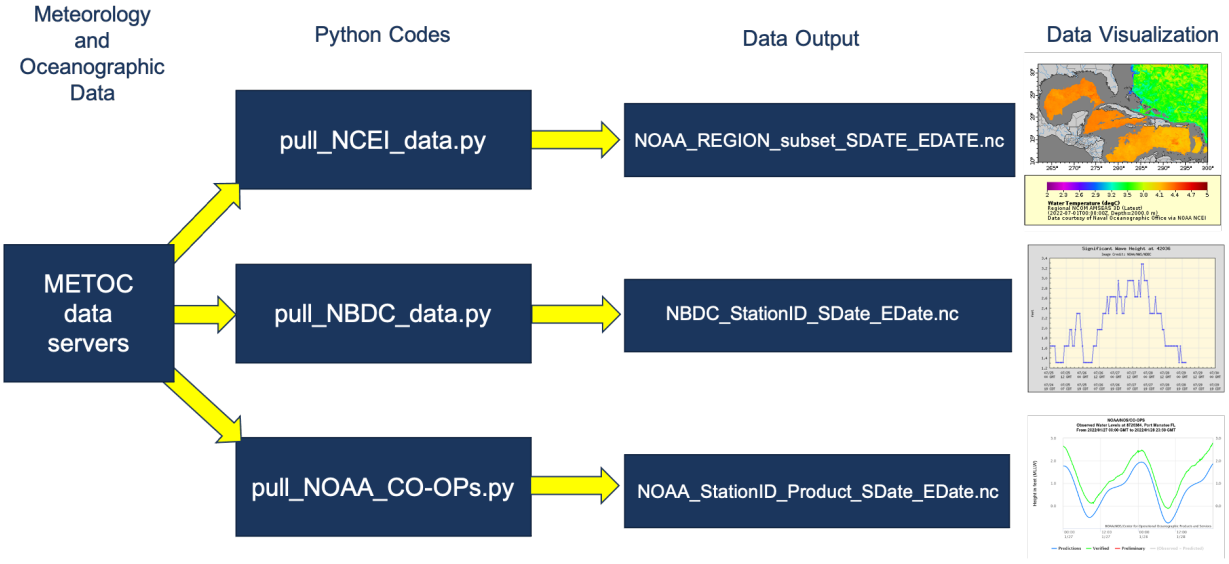

Fig. 3—A flowchart of the technical approach.

2.1 Web Servers

The first step to the technical approach was creating a repository comprised of all the data available on the web servers incorporated in this project. There are three different web servers utilized that contain publicly available data under the National Oceanographic and Atmospheric Administration (NOAA). These are The National Centers for Environmental Information (NCEI), The National Data Buoy Center (NDBC), and The Center for Operational Oceanographic Products and Services (CO-OPS).

2.1.1 National Centers for Environmental Information (NCEI)

The National Centers for Environmental Information (NCEI - `https://www.ncei.noaa.gov/`) provide both historical and near-real-time weather and climate forecast model data. The available models include Navy models that are run operationally all over the globe. The two models used in this work are the Navy Global Hybrid Coordinate Ocean Model (HYCOM) and the Navy Coastal Ocean Model (NCOM). The data is accessible through an Environmental Research Division's Data Access Program (ERRDAP) server.

Available products can be found at:

`https://www.ncei.noaa.gov/products`.

The two Navy models HYCOM and NCOM are available at:

`https://www.ncei.noaa.gov/products/weather-climate-models/ocean`.

Navy Global Hybrid Coordinate Ocean Model (HYCOM)

The Fleet Numerical Meteorology and Oceanography Center (FNMOC) runs a global-scale operational ocean prediction system that produces daily ocean forecasts based on the Navy Global Hybrid Coordinate Ocean Model (HYCOM). This data spans several regions and outputs multiple ocean variables such as surface elevation and water temperature. There are five regions publicly available on NCEI that run at a resolution of $1/12^\circ$ and output predictions every 3 hours. Figure 4 shows the publicly available regions. Tables 1 and 2 give the specific model dataset names, times available, and model outputs.

Spatial resolution: $1/12^\circ$

Temporal output resolution: 3 hours

Regions:
- Western Atlantic - Region 1
- Mid Pacific - Region 6
- Eastern Pacific - Region 7
- Arctic Ocean - Region 17
- Global

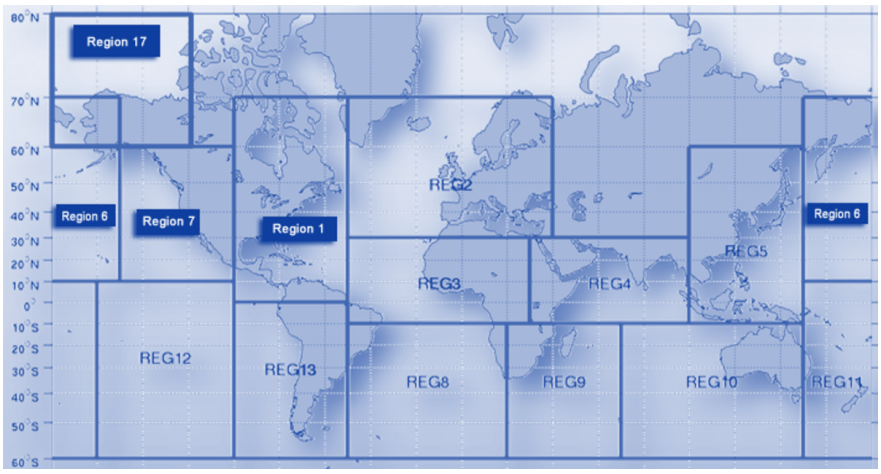

Fig. 4—HYCOM Regions available on the NCEI web server (in highlighted boxes).

Table 1—HYCOM region data descriptions

Region Name	Region number	Resolution	Time period available
Western Atlantic	1	1/12°	2013-02-27 to Present
Mid Pacific	6	1/12°	2013-03-05 to Present
Eastern Pacific	7	1/12°	2013-03-05 to Present
Arctic Ocean	17	1/12°	2013-03-05 to Present
Global	sfc	1/12°	2013-03-05 to Present

Variables:

- Time (time) [seconds since 1970-01-01T00:00:00Z]
- Latitude (latitude) [degrees_north]
- Longitude (longitude) [degrees_east]
- Depth (depth) [m]
- Water Surface Elevation (surf_el) [m]
- Water Temperature (water_temp) [degC]
- Eastward Currents (water_u) [m/s]
- Northward Currents (water_v) [m/s]
- Salinity (salinity) [psu]

Table 2—HYCOM datasets and variables available. In the Dataset name, *N* is the region number from Table 1.

Dataset name	variable name	2D/3D	output time step
HYCOM_reg*N*_latest2d	surf_el	2D	3 hrs
HYCOM_reg*N*_latest3d	water_temp water_u water_v salinity	3D	3 hrs

Regional Navy Coastal Ocean Model (NCOM)

The Fleet Numerical Meteorology and Oceanography Center (FNMOC) operates regional versions of the Navy Coastal Ocean Model (NCOM) for the Gulf of Mexico and Caribbean Sea, the U.S East Coast, and the northeast Pacific (including the Gulf of Alaska).This model outputs multiple ocean variables such as water surface elevation, surface atmospheric pressure, surface roughness, and others. The three regions publicly available on NCEI run at a resolution between $1/36°$ and $1/30°$ and output predictions every 3 hours. Tables 3 and 4 give the specific model dataset names, times available, and model outputs.

Spatial resolution: $1/30°$ and $1/36°$

Temporal output resolution: 3 hours

Regions:
- American Seas
- US East
- Alaska

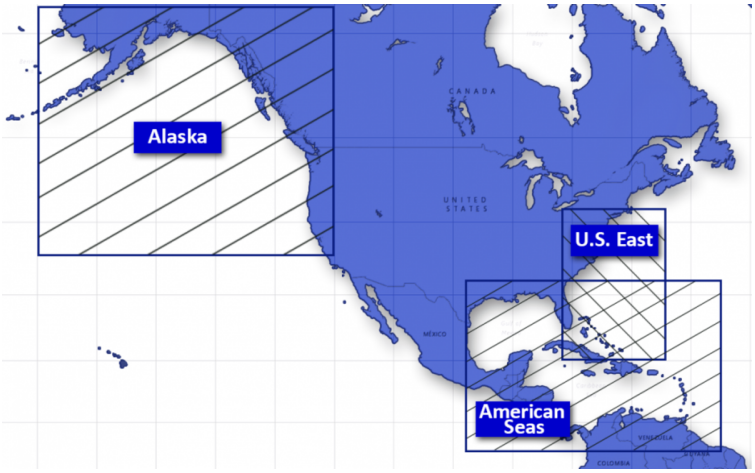

Fig. 5—NCOM Regions available on NCEI.

Variables:
- Time (time) [seconds since 1970-01-01T00:00:00Z]
- Latitude (latitude) [degrees_north]
- Longitude (longitude) [degrees_east]
- Depth (depth) [m]
- Water Surface Elevation (surf_el) [m]
- Surface Atmospheric Pressure (surf_atm_press) [millibar]
- Surface Roughness/Waves (surf_roughness) [m]
- Surface Temperature Flux (surf_temp_flux) [degC meter/second]
- Surface Wind Stress (surf_wnd_stress_gridx, surf_wnd_stress_gridy) [pascal]
- Water Temperature (water_temp) [degC]
- Eastward Currents (water_u) [m/s]
- Northward Currents (water_v) [m/s]
- Salinity (salinity) [psu]
- and others

Table 3—NCOM region data descriptions.

Region	Keyword (*N*)	Resolution	Periods Available	Period name (*P*)
American Seas	amseas	1/36°	04/05/2013 to 12/16/2020	20130405_to_20201216
		1/30°	12/17/2020 to Present	latest
US East	us_east	1/36°	02/08/2009 to 11/18/2009	before_change
		1/36°	04/05/2013 to 12/16/2020	20130405_to_20201216
		1/30°	12/17/2020 to Present	latest
Alaska	alaska	1/30°	03/05/2013 to Present	latest

Table 4—NCOM datasets and variables available on NCEI. In the Dataset name, *N* is the region keyword from Table 3 and *P* is the Dataset period name from Table 3. All periods share the same variables.

Dataset name	variables	2D/3D	Output time
NCOM_*N*_*P*2d	surf_el surf_atm_press surf_temp_flux surf_solar_flux surf_roughness surf_wnd_stress_gridx surf_wind_stress_gridy	2D	3 hrs
NCOM_*N*_*P*3d	water_temp water_u water_v salinity	3D	3 hrs

2.1.2 National Data Buoy Center (NDBC)

The National Data Buoy Center (NDBC - `https://www.ndbc.noaa.gov/`) is a sustainable and resilient marine observation and monitoring infrastructure which enhances healthy ecosystems, communities, and economies. The server is comprised of buoys located around the globe that make observations and store them for scientific access. The data is accessible via a Thematic Real-Time Environmental Distributed Data Services (THREDDS) server. Data is accessible through the NDBC application programming interface (API).

Available data is located at:

`https://www.ndbc.noaa.gov`.

The API information is located at:

`https://pypi.org/project/NDBC/`.

There are several different datasets available on the site (see list below); however, the availability of meteorological and oceanographic variables are specific to each individual station. Currently, the python algorithm only pulls standard meteorological data (stdmet), which includes significant wave height.

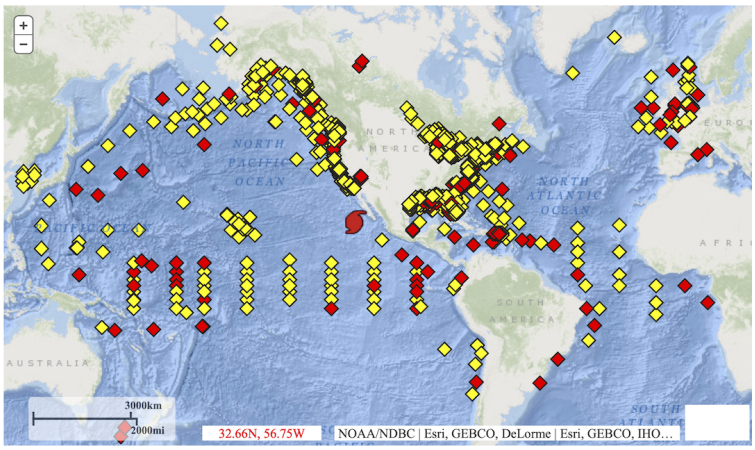

Fig. 6—Buoy Stations available from NDBC.

Datasets Available:

- Acoustic Doppler Current Profiler data (adcp)
- ADCP data with additional QC information (adcp2)
- Continuous Winds data (cwind)
- Deep-ocean Assessment and Reporting of Tsunamis data (dart)
- Marsh-McBirney Current Measurements data (mmbcur)
- Oceanographic data (ocean)
- Peak Winds data (pwind)
- Standard Meteorological data (stdmet)
- Spectral Wave Density data with Spectral Wave Direction data (swden)
- Water Level data (wlevel)

2.1.3 NOAA CO-OPS

The Center for Operational Oceanographic Products and Services (CO-OPS) is the authoritative source for accurate, reliable, and timely tides, water levels, currents, and other coastal oceanographic and meteorological information. CO-OPS maintains ocean observing infrastructure, including more than 200 permanent water level stations on the U.S. coasts and Great Lakes, an integrated system of real-time sensors concentrated in busy seaports, and temporary meters that collect observations for tidal current predictions. The data is accessible through through the CO-OPS API.

Available data is located at:

`https://tidesandcurrents.noaa.gov/`.

The API information is located at:

`https://api.tidesandcurrents.noaa.gov/api/prod/`

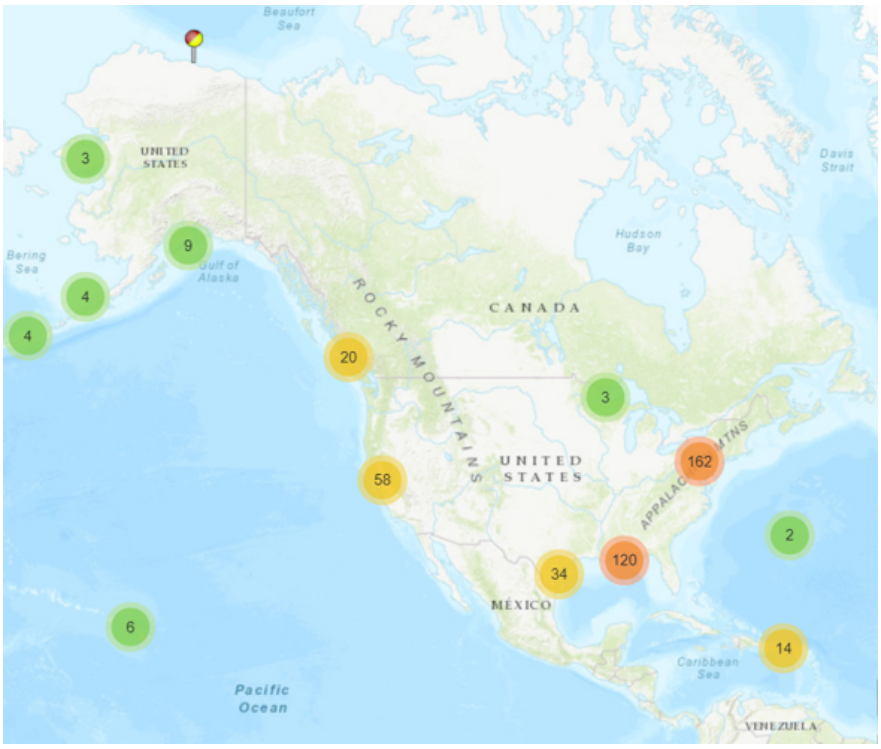

Fig. 7—Stations available through NOAA CO-OPS

Variables:

- Water Level (water_level)
- Air Temperature (air_temperature)
- Water Temperature (water_temperature)
- Wind (wind)
- Air Pressure (air_pressure)
- Air Gap (air_gap)
- Conductivity (conductivity)
- Visibility (visibility)
- Humidity (humidity)
- Salinity (salinity)
- Hourly Height (hourly_height)
- High Low (high_low)
- Daily Mean (daily_mean)
- Monthly Mean (monthly_mean)
- Minute Water Level (one_minute_water_level)
- Predictions (predictions)
- Datums (datums)
- Currents (currents)
- Current Predictions (current_predictions)

2.2 Extraction Codes

The next part of the technical approach was to create a centralized and efficient approach to extract the publicly available METOC data from online sources described in Section 2.1. The code takes user input on the requested location, times, and types of data from a YAML file and extracts the requested data. YAML is a data serialization language that is often used for writing configuration files. The algorithm then saves the data as a netCDF file and outputs a plot of the data. The netCDF can then be used for nearshore high-resolution model forcing and/or validation.

2.2.1 Extracting NCEI Data

The `pull_NCEI_data.py` program extracts model data from the NCEI web server database and saves it in a netCDF file. The model data pulled is determined by variables read in through the `NCEI_input.yml` YAML file that is named on the command line. The netCDF file will be saved in the directory the user has specified on the command line. The following command will run the program:

```
>> python pull_NCEI_data.py -y INPUT_FILE -i INPUT_DIR -o OUTPUT_DIR
```

where

```
INPUT_FILE : YAML input file name with requested parameters (DEFAULT = NCEI_input.yml)
INPUT_DIR : directory location of the input file (DEFAULT = ./)
OUTPUT_DIR : directory location for output to be saved (DEFAULT = ./)
```

A default demo program can be executed with the following command:

```
>> python pull_NCEI_data.py -y NCEI_input_demo.yml
```

The user must input the following information into the `NCEI_input.yml` file:

- dataset: The dataset (model and 2D or 3D variables) requested from the NCEI database
- start_time: The beginning of the time range in the format of yyyy-mm-dd HH:MM:SS
- end_time: The end of the time range in the format of 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
- depth_min: The minimum depth (ONLY IF 3D)
- depth_max: The maximum depth (ONLY IF 3D)
- variable1: METOC variable requested
- variable2: METOC variable requested (ONLY IF 3D)

The `pull_NCEI_data.py` program 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 netCDF for input to the Python routine used for writing lateral boundary conditions for Delft3D-Flexible Mesh (DFM), `generate_BCs_NCOM.py` [1].

An example of an input file for extracting surface elevation from the 2D NCOM American Seas model is shown below.

```

  dataset: 'NCOM_amseas_latest2d'
  two_dim_or_three_dim: '2D'
  lat_min: 15
  lat_max: 32
  lon_min: 262
  lon_max: 280
  start_time: 2021-06-08 00:00:00
  end_time: 2021-06-09 00:00:00
  depth_min: ' '     #Leave depth variables empty (as in ' ') if 2D
  depth_max: ' '
  var1: 'surf_el'
  var2: ' '          #Leave var2 empty (as in ' ') if 2D
```

Listing 1—YAML input file (`NCEI_input.yml`) for the `pull_NCEI_data.py` algorithm.

An example of an input file for extracting velocity from the 3D NCOM American Seas model is shown below.

```

#two_dim_or_three_dim: '2D'
two_dim_or_three_dim: '3D'
#dataset: 'NCOM_amseas_20130405_to_20201216_2d'
dataset: 'NCOM_amseas_20130405_to_20201216_3d'
#dataset: 'NCOM_amseas_latest2d'
#dataset: 'NCOM_amseas_latest3d'
lat_min: 15
lat_max: 32
lon_min: 262
lon_max: 280
start_time: 2020-08-01 00:00:00
end_time: 2020-09-30 00:00:00
#Leave depth variables empty (as in ' ') if 2D
#depth_min: ' '
#depth_max: ' '
depth_min: 0
depth_max:  5000
#var1: 'surf_el'
var1: 'water_u'
#Leave 2nd variable empty (as in ' ') if 2D
#var2: ' '
var2: 'water_v'
```

Listing 2—`NCEI_input.yml` input file for the `pull_NCEI_data.py` algorithm.

Values for 2D and 3D data retrieval are contained in the above `NCEI_input.yml` file as well as keywords for different datasets. The unused options can remain commented. This approach allows for greater efficiency in running the code for varying datasets.

The following section includes the `pull_NCEI_data.py` code and comments.

```
'''
Authors: Kendal Hall, Sunni Schoenauer, and Allison Penko
09 sept 22

'''

from erddapClient import ERDDAP_Griddap
from dateutil import rrule
from datetime import datetime as dtt, timedelta
import argparse
import pathlib
import datetime as dt
from datetime import date
import yaml
import xarray
import time
import netCDF4
import matplotlib, pylab # already in pylab
import math, statistics
import matplotlib.pyplot as plt
import warnings

warnings.simplefilter(action='ignore', category=FutureWarning)

parser = argparse.ArgumentParser()


parser.add_argument('-y', '--yaml-file',
                    help = 'YAML file containing input settings',
                    metavar = 'FILE',
                    default = 'NCEI_input.yml',
                    type = pathlib.Path)


parser.add_argument('-i', '--input-dir',
                    help = 'directory of input data',
                    metavar = 'DIR',
                    default = './',
                    type = pathlib.Path)

parser.add_argument('-o', '--output-dir',
                    help = 'output directory',
                    metavar = 'DIR',
                    default = './',
                    type = pathlib.Path)


args = parser.parse_args()


```

```
# parse the input YAML file
with open(args.yaml_file, 'r') as fp:
    settings = yaml.safe_load(fp)

    # Set all the variables from the yaml file
    dataset = settings['dataset']
    lat_min = settings['lat_min']
    lat_max = settings['lat_max']
    lon_min = settings['lon_min']
    lon_max = settings['lon_max']
    start_time = settings['start_time']
    end_time = settings['end_time']
    depth_min = settings['depth_min']
    depth_max = settings['depth_max']
    var1 = settings['var1']
    var2 = settings['var2']
    two_dim_or_three_dim = settings['two_dim_or_three_dim']

##################Code below is only applicable to 3D code####################
delta1 = timedelta(hours=9)
delta = timedelta(hours=12) # variable to hold 12 hour increments
current_time = start_time # set variable that will hold start time that will increment each iteration
current_end_time = current_time + delta1 # set the end time that will increment each iteration
count = 0 #Set the count that will increment each iteration
##################----------------------------------------####################

#Pull the data from the NCEI website
remote = ERDDAP_Griddap('https://www.ncei.noaa.gov/erddap', dataset)

if two_dim_or_three_dim == '2D':
  # Determine what data to pull by these parameters
        subsetncFile = ( remote.setResultVariables([var1])
                                .setSubset(time=slice(start_time,end_time),
                                           latitude=slice(lat_min, lat_max),
                                           longitude=slice(lon_min, lon_max))
                                .getData(filetype='nc'))

        start_time_n = (start_time.strftime('%Y-%m-%d')) #Strip the start time of %H %M %S
        end_time_n = (end_time.strftime('%Y-%m-%d')) #Strip the current end time of %H %M %S

        fl = f"NCEI_{dataset}_{start_time_n}_{end_time_n}.nc"

        fb = open(fl,'wb')
        fb.write(subsetncFile)
        fb.close()
        ncfile = fb

else:
        #while current is less or equal then end time keep looping
        while current_end_time <= end_time:
          # Determine what data to pull by these parameters
                subsetncFile = ( remote.setResultVariables([var1, var2])
```

```
                                    .setSubset(time=slice(current_time,current_end_time)
    ,
                                               depth = slice(depth_min,depth_max),
                                               latitude=slice(lat_min, lat_max),
                                               longitude=slice(lon_min, lon_max))
                                    .getData(filetype='nc'))

                start_time_n = (current_time.strftime('%Y-%m-%d')) #Strip the current
    start time of %H %M %S
                end_time_n = (current_end_time.strftime('%Y-%m-%d')) #Strip the
    current end time of %H %M %S
                if (count % 2) == 0:
                    fn = f"NCEI_{dataset}_{start_time_n}_{end_time_n}_AM.nc" #Set the
    name of the first 12 hours in AM (To prevent overwriting)
                else:
                    fn = f"NCEI_{dataset}_{start_time_n}_{end_time_n}_PM.nc" #Set the
    name of the last 12 hours in PM

                #Open, write contents to file and close it
                fb = open(fn,'wb')
                fb.write(subsetncFile)
                fb.close()

                #Increment the count, current time and end time to iterate through the
     next 12 hours
                count += 1
                current_time += delta
                current_end_time += delta

        start_time_n = (start_time.strftime('%Y-%m-%d')) #Strip the start time of %H %
    M %S

        # Add results from selected netCDF files to xarray dataset, concatenating
    along named dimension.
        ds = xarray.open_mfdataset(f'NCEI_{dataset}_{start_time.year}*.nc', combine =
    'nested', concat_dim="time")

        # Write xarray dataset to netCDF file with given name.
        ds.to_netcdf(f'NCEI_{dataset}_{start_time_n}_{end_time_n}.nc')
        ncfile = ds


###################--PLOT extracted data--

ncfile = fb

# open file
grid = netCDF4.Dataset(ncfile)
# see what variables are in there
grid.variables

# load coordinates
# the [:,:] are required as grid.variables['var'] is only a pointer
lon = grid.variables['longitude'][:]
lat = grid.variables['latitude'][:] # [:,:]

# load the timesteps
```

```
tim = netCDF4.num2date(grid.variables['time'], units=grid.variables['time'].units)
# see what times are in there; tim[1,-1] ???
start_time = tim[0]
end_time = tim[-1]

# see how many times are in there, and select one
sizevar=grid.variables[var1].shape
print ('size of var1 is: ',sizevar)

# pick a time and load that slice
it  = 1
if two_dim_or_three_dim == '2D':
    val = grid.variables[var1][it,:,:]
else:
    val = grid.variables[var1][it,depth,:,:]

sizeval=val.shape
print ('size of val is: ',sizeval)

#val_array = val.data.flatten()
minval = val[:].min()
maxval = val[:].max()
clim=round(max([abs(minval), abs(maxval)]))
#print ('clim is:',clim)
#print ('max is ', maxval)
#print ('min is ', minval)
#print (type(val))

val_name = grid.variables[var1].long_name
val_units = grid.variables[var1].units
val_time = tim[it]

matplotlib.pyplot.pcolormesh(lon,lat,val,vmin = -clim, vmax = clim, cmap = 'seismic',
    shading = 'auto') # pcolormesh won't work here, it interprets data at
    centersmatplotlib.pyplot.colorbar()
matplotlib.pyplot.axis('tight')
matplotlib.pyplot.title(str(tim[it]) + ' @ depth of ' + str(depth) + 'm')
matplotlib.pyplot.gca().set_aspect(1./math.cos(statistics.mean(plt.ylim())/180*math.pi
    ),adjustable='box') # see Matlab axislat()
cbar = plt.colorbar()
labstr=val_name + ' ' + '(' + val_units + ')' #setting the name of the colorbar with
    units
cbar.set_label(labstr, rotation=90)

filstr = var1 + '_' + val_time.strftime("%Y-%m-%d_%H%M%S") +'.png' #setting name of
    the plot with variable and time
print (filstr)

plt.savefig(filstr, dpi=300, facecolor='white', edgecolor='none') #saving plot to
    current directory
```

2.2.2 Extracting NDBC data

The `pull_NDBC_Data.py` program gathers observation data from the NDBC database and saves it in a netCDF file. The standard meteorological data is pulled, which includes significant wave height. The station

and date are determined by variables read in through the `NDBC_input.yml` YAML file that is named on the command line. The netCDF file will be saved in the directory the user has specified on the command line.

```
>> python pull_NDBC_data.py -y NDBC_input.yml -o . -i .
```

The user must input the following information into the `NDBC_input.yml` file:

- 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)
- station_id: Station ID of requested station

An example of the input file is shown below.

```
start_time: 2013-04-25 23:00:00
end_time: 2013-05-11 01:00:00
station_id: '42012'
```

Listing 3—`NDBC_input.yml` input file for the `pull_NDBC_Data.py` algorithm.

After execution, a message with the first time available and the last time available for the year containing your requested date will be returned in the terminal. For example:

```
The first time available is:  2012-12-31 23:50:00
The last time available is:  2013-12-31 22:50:00
```

Observations may not be available during the requested time period for a given station. In this case, the following error will result:

```
Year 2013 not available.
Please review available station data:
https://www.ndbc.noaa.gov/station_history.php?station=42098

Traceback (most recent call last):
File "pull\_NDBC\_data.py", line 90, in <module>
df = DB.data.get('stdmet').get('data')
AttributeError: 'NoneType' object has no attribute 'get'
```

It is also possible for some times to be available during the year containing the requested date range but not during the requested date range itself. If a netCDF file is created, but the model-data comparison routine fails when you try to use the netCDF as input, check the size of the netCDF's time dimension.

The following section includes the `pull_NDBC_Data.py` code and comments.

```
'''
Authors: Kendal Hall, Sunni Schoenauer, and Allison Penko
09 sept 22

'''


from NDBC.NDBC import DataBuoy
import netCDF4 as nc
import numpy as np
import yaml
import argparse
import pathlib

# Improvements suggested for this code:
# - Better file name, created dynamically, to include station ID and start and
#   end dates
# - Input station ID, years range, and start and end dates as argument to
#   function or as a file

# ---------------------------------- INPUTS ----------------------------------


parser = argparse.ArgumentParser()


parser.add_argument('-y', '--yaml-file',
                    help = 'YAML file containing input settings',
                    metavar = 'FILE',
                    required = True,
                    type = pathlib.Path)


parser.add_argument('-i', '--input-dir',
                    help = 'directory of input data',
                    metavar = 'DIR',
                    required = True,
                    type = pathlib.Path)

parser.add_argument('-o', '--output-dir',
                    help = 'output directory',
                    metavar = 'DIR',
                    required = True,
                    type = pathlib.Path)


args = parser.parse_args()



# parse the input YAML file
with open(args.yaml_file, 'r') as fp:
    settings = yaml.safe_load(fp)

    station_id = settings['station_id']  # Set all the variables from the yaml file
    start_date = settings['start_time']
    end_date = settings['end_time']
```

```
'''
# Supply station ID (either strings or numbers are acceptable):
station_id = '42099'

# Date range for which user wants data:
start_date = datetime.datetime(2021, 5, 24, 23, 0)
end_date = datetime.datetime(2021, 6, 1, 1, 0)

#-------------------------------------------------------------------------

'''

start_year = int(start_date.strftime('%Y'))
end_year = int(end_date.strftime('%Y'))+1

year_range = range(start_year, end_year)
start_date_n = (start_date.strftime('%Y%m%d'))
end_date_n = (end_date.strftime('%Y%m%d'))
fn = f"NDBC_{station_id}_{start_date_n}-{end_date_n}.nc"

# Set abbreviation.
DB = DataBuoy()

# Specify station ID.
DB.set_station_id(station_id) # <- Either strings or numbers are acceptable.

# Get station metadata and data.
DB.get_station_metadata()
DB.get_data(years=year_range, datetime_index=True, data_type='stdmet')

# Access dataframe in dict within a dict.
df = DB.data.get('stdmet').get('data')

# Make new dataframe, keeping only rows with WVHT values.
wvht = df[~df['WVHT'].isna()]

# Print the first and last times available in the file.
print("The first time available is: ", wvht.index[0].to_pydatetime())
print("The last time available is: ", wvht.index[-1].to_pydatetime())

# Access dictionary with units.
units_dict = DB.data.get('stdmet').get('meta').get('units')

# NOTE:-----------------------------------------------------------------------
# When I tried using the months argument for DB.get_data, it would do two
# things wrong:
# It would say every month later than May was not available.
# It would start at Jan regardless of what month I requested to start.
# The solution I found was to request the whole year.
# ----------------------------------------------------------------------------

# Put a start date and end date and cull everything not in date range.
wvht.index = wvht.index.to_pydatetime()
mask = (wvht.index > start_date) & (wvht.index <= end_date)
wvht = wvht.loc[mask]

```

```
# Convert the pandas dataframe to an xarray dataset.
xr_ds = wvht.to_xarray()

# Write the xarray dataset to a netCDF file.
xr_ds.to_netcdf(path=fn, mode='w', format=None, group=None,
                engine=None, encoding=None, unlimited_dims=None, compute=True,\
                    invalid_netcdf=False)

# Get keys in the units_dict.
keys_list = list(units_dict.keys())

# Find the vars in the nc file (or dataset) that are in common with the entries
# in keysList.
ds = nc.Dataset(fn,mode='a')

# Get the names of the variables in the netCDF file.
var_names = ds.variables.keys()

# Go through each variable name.
for name in var_names:
    # If the variable name is in keys of the units_dict,
    if name in keys_list:
        # assign its value to the units attribute to the variable in the netCDF.
        ds.variables[name].units = units_dict[name]

# Close the netCDF file.
ds.close()

# NOTE:-----------------------------------------------------------------------
# Time is called index in this netCDF file.
# It is both a dimension and a variable.
# time is a more conventional name so rename them from index to time.
# In addition, the global attributes need to be added to this file.
# ----------------------------------------------------------------------------

# Put station info in a dict to be used for global attributes.
attr_dict = DB.station_info

# Open the netCDF file to make changes.
f = nc.Dataset(fn,'a')

# Each key, processed to get rid of spaces, needs to be supplied,
# along with its value, to create global attributes.
for name, value in attr_dict.items():
    setattr(f, name.replace(" ", "_"), value)

# Rename the dim index to time.
f.renameDimension(u'index',u'time')

# NOTE:-----------------------------------------------------------------------
# Renaming var index to time replaced all the vals with the default fill val!
# Instead, create a new var, time, using all of index's attributes and vals.
# ----------------------------------------------------------------------------

# Create var, time.
time = f.createVariable('time', np.int_, ('time',))

```

```
# Add vatiable attributes to var, time.
time.units = f.variables['index'].units
time.calendar = f.variables['index'].calendar

# Write the vals to var, time.
time[:] = f.variables['index'][:]

# Close the netCDF file.
f.close()
```

2.2.3 Extracting NOAA CO-OPS data

The `pull_NOAA_CO-OPS_data.py` program gathers observed data (e.g., water level or wind) from the NOAA Tides and Currents database and saves it in a netCDF file. The data pulled is determined by variables read in through the `NOAA_CO-OPS.yml` YAML file that is named on the command line.

```
>> python pull_NOAA_CO-OPs_data.py -y NOAA_CO-OPS.yml -o . -i .
```

The user must input the following information into the `NOAA_CO-OPS.yml` file

- station_id: Station ID of requested station
- data_type: Product type, e.g., 'water_level' or 'wind'
- datum: The data reference point (MSL, MLLW, MLW)
- units_convention: Either 'metric' or 'english'
- start_date: The beginning of the time range (yyyymmdd)
- end_date: The end of the time range (yyyymmdd)
- tzone: The time zone of the data ('gmt')

An example of the `NOAA_CO-OPS.yml` input file is shown below with common CO-OPS stations along the Florida panhandle and west coast.

```
station_id: '8736897' # Coast Guard Sector Mobile, AL
#station_id: '8735180' # Dauphin Island, AL
#station_id: '8729840' # Pensacola, FL
#station_id: '8729108' # Panama City, FL
#station_id: '8728690' # Apalachicola, FL
#station_id: '8727520' # Cedar Key, FL
#station_id: '8726724' # Clearwater Beach, FL
#station_id: '8726384' # Port Manatee, FL
data_type: 'water_level'
#data_type: 'wind'
datum: 'MSL'
units_convention: 'metric'
start_date: '20130425'
end_date: '20130511'
tzone: 'gmt'
```

Listing 4—`NOAA_COOPS.yml` input file for the `pull_NOAA_CO-OPS_data.py` algorithm.

If observations are not available during the requested time period for a particular station, the following error message will result:

```
Traceback (most recent call last):
File "pull_NOAA_CO-OPS_data.py", line 90, in <module>
time_zone = tzone)
File "/python3.7/site-packages/noaa_coops/noaa_coops.py", line 492, in get_data
df = self._url2pandas(data_url, product, num_request_blocks=1)
File "/python3.7/site-packages/noaa_coops/noaa_coops.py", line 411, in
_url2pandas json_dict["error"].get("message", "Error retrieving data")
ValueError: No data was found.
This product may not be offered at this station at the requested time.
```

The following section includes the `pull_NOAA_CO-OPS_data.py` code and comments.

```
#!/usr/bin/env python3
'''
Authors: Kendal Hall, Sunni Schoenauer, and Allison Penko
09 sept 22

'''

import yaml
import argparse
import pathlib
import noaa_coops as nc
import numpy as np
from netCDF4 import Dataset
import os

# INPUTS-------------------------------------------------------------------------
# station_id = '8736897' # Coast Guard Sector Mobile, AL
# station_id = '8735180' # Dauphin Island, AL
# station_id = '8729840' # Pensacola, FL
# station_id = '8729108' # Panama City, FL
# station_id = '8728690' # Apalachicola, FL
# station_id = '8727520' # Cedar Key, FL
# station_id = '8726724' # Clearwater Beach, FL



parser = argparse.ArgumentParser()


parser.add_argument('-y', '--yaml-file',
                    help = 'YAML file containing input settings',
                    metavar = 'FILE',
                    required = True,
                    type = pathlib.Path)


parser.add_argument('-i', '--input-dir',
                    help = 'directory of input data',
                    metavar = 'DIR',
                    required = True,
                    type = pathlib.Path)

```

```
parser.add_argument('-o', '--output-dir',
                    help = 'output directory',
                    metavar = 'DIR',
                    required = True,
                    type = pathlib.Path)


args = parser.parse_args()



# parse the input YAML file
with open(args.yaml_file, 'r') as fp:
    settings = yaml.safe_load(fp)

    station_id = settings['station_id']  # Set all the variables from the yaml file
    start_date = settings['start_date']
    end_date = settings['end_date']
    data_type = settings['data_type']
    datum = settings['datum']
    units_convention = settings['units_convention']
    tzone = settings['tzone']
'''
station_id = '8726384' # Port Manatee, FL
data_type = 'water_level'
datum = 'MSL'
units_convention = 'metric'
start_date = '20210524'
end_date = '20210601'
tzone = 'gmt'
'''
fl = f"NOAA{station_id}_{data_type}_{start_date}_{end_date}.nc"
path = pathlib.Path().resolve()
fn = os.path.join(path,fl)
#-------------------------------------------------------------------------------

# Set the NOAA station for data retrieval.
station = nc.Station(station_id)

# Get the requested data from the named station.
df = station.get_data(
    begin_date = start_date,
    end_date = end_date,
    product = data_type,
    datum = datum,
    units = units_convention,
    time_zone = tzone)

# Dict of metadata that can be added as global attributes
metadata = station.metadata

# Convert the pandas dataframe to an xarray dataset.
xr_ds = df.to_xarray()

# Write the xarray dataset to a netCDF file.
xr_ds.to_netcdf(path=fn, mode='w', format=None, group=None,
                engine=None, encoding=None, unlimited_dims=None, compute=True,\
```

```
                    invalid_netcdf=False)

# NOTE:----------------------------------------------------------------------
# This netCDF has date_time instead of time as the dimension and var names.
# I think I am going to want to do the same thing with this netCDF file as I
# did with the one containing the NDBC data - that is rename the dimension and
# create a new variable, time, using date_time's attribute and vals.
#----------------------------------------------------------------------------

# Put station info in a dict to be used for global attributes.
attr_dict = station.metadata

# Open the netCDF file to make changes.
f = Dataset(fn,'a')

# List of quantities from metadata to be added as global attributes:
global_attr_list = ['id','lat','lng','name','timezone','timezonecorr']

# Each key needs to be supplied, along with its value, to create global
# attributes.
for nm, value in attr_dict.items():
    if nm in global_attr_list:
        Dataset.setncattr(f, nm, value)

# Rename the dim index to time.
f.renameDimension(u'date_time',u'time')

# Create var, time.
time = f.createVariable('time', np.int_, ('time',))

# Add variable attributes to var, time.
time.units = f.variables['date_time'].units
time.calendar = f.variables['date_time'].calendar

# Write the vals to var, time.
time[:] = f.variables['date_time'][:]

# Close the netCDF file.
f.close()
```

3. CONCLUSIONS

High-resolution models allow the Navy to have comprehensive forecasts of their operational environment. Information from low-resolution models is used to create boundaries with increasing resolution at smaller scales. Automated and efficient algorithms to obtain and reformat meteorological data are necessary to both drive and assess these models. Three different algorithms were created to pull publicly available data from web servers containing both model and observational data. NCEI, NDBC, and NOAA CO-OPS datasets are selected through a YAML input file, allowing for ease of use for NRL researchers. Swift methods to access METOC data for model forcing and evaluation ultimately reduces the amount of time developers need for model development and validation.

ACKNOWLEDGMENTS

KH was funded by the Department of Navy Historically Black Colleges and Universities and Minority Institutions (HBCU/MI) Internship Program and supported by The Washington Center. AMP and SSS were funded by the U.S. Naval Research Laboratory Base Program.

REFERENCES

1. S. S. Schoenauer, A. M. Penko, K. L. Edwards, and J. Veeramony, "Delft3D-FM simulation configuration and input file generation.," NRL Memorandum Report NRL/MR/7350--2022/2, 5695, US Naval Research Laboratory, Stennis Space Center, MS, 2022.