U. S. Naval Research Laboratory
Washington, DC 20375-5320

NRL/7350/FR--2024/1

getData: A Toolkit for Automated Retrieval and Formatting of Oceanographic and Atmospheric Data From Online Sources

Ryan E. Phillip and Allison M. Penko

Seafloor Sciences Branch
Ocean Sciences Division

March 22, 2024

DISTRIBUTION STATEMENT A. Approved for public release. Distribution is unlimited.

REPORT DOCUMENTATION PAGE

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ORGANIZATION

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED	
		START DATE	END DATE
March 22, 2024	NRL Formal Report	01 OCT 21	30 SEP 24

4. TITLE AND SUBTITLE
getData: A Toolkit for Automated Retrieval and Formatting of Oceanographic and Atmospheric Data From Online Sources

5a. CONTRACT NUMBER	5b. GRANT NUMBER	5c. PROGRAM ELEMENT NUMBER
		62435N
5d. PROJECT NUMBER	**5e. TASK NUMBER**	**5f. WORK UNIT NUMBER**
100001936878.0010		6C91

6. AUTHOR(S)
Ryan E. Phillip and Allison M. Penko

7. PERFORMING ORGANIZATION / AFFILIATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
1005 Balch Blvd., Stennis Space Center, MS 39529	NRL/7350/FR--2024/1

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR / MONITOR'S ACRONYM(S)	11. SPONSOR / MONITOR'S REPORT NUMBER(S)
Office of Naval Research One Liberty Center 875 N. Randolph Street, Suite 1425 Arlington, VA 22203-1995	ONR	

12. DISTRIBUTION / AVAILABILITY STATEMENT
DISTRIBUTION STATEMENT A. Approved for public release. Distribution is unlimited.

13. SUPPLEMENTAL NOTES

14. ABSTRACT
Meteorological and oceanographic data are needed to drive and validate hydrodynamic models. These data can be obtained from publicly available online sources; however, the processes required for users to acquire and format the data vary greatly between sources. Therefore, manually downloading and reformatting data from different sources can be cumbersome and time consuming. A Python toolkit is presented here that allows for the automatic retrieval and reformatting of meteorological and oceanographic data from common online sources. The toolkit provides an efficient and streamlined method for obtaining atmospheric and hydrodynamic data used in model development, validation, and/or testing.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES
a. REPORT	b. ABSTRACT	c. THIS PAGE		
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	NONE	60

19a. NAME OF RESPONSIBLE PERSON	19b. PHONE NUMBER *(Include area code)*
Ryan E. Phillip	(228) 688-5041

PREVOUS EDITION IS OBSOLETE
STANDARD FORM 298 (REV. 5/2020)
Prescribed by ANSI Std. Z39.18

This page intentionally left blank

CONTENTS

EXECUTIVE SUMMARY E-1

1. INTRODUCTION 1
   1.1 Problem and Objective 1
   1.2 Background 1

2. METHODOLOGY 3
   2.1 Web Servers 3
      2.1.1 National Centers for Environmental Information (NCEI) 3
         2.1.1.1 Navy Global Hybrid Coordinate Ocean Model (HYCOM) 4
         2.1.1.2 Regional Navy Coastal Ocean Model (NCOM) 5
      2.1.2 General Bathymetric Chart of the Oceans (GEBCO) 7
      2.1.3 National Centers for Environmental Prediction (NCEP) 7
         2.1.3.1 WaveWatch III (WW3) Production Hindcast 8
      2.1.4 National Data Buoy Center (NDBC) 10
      2.1.5 NOAA CO-OPS 11
   2.2 Data Downoad Codes 13
      2.2.1 Downloading NCEI HYCOM and NCOM Data 13
      2.2.2 Downloading GEBCO Bathymetry Data 27
      2.2.3 Downloading NCEP WaveWatch III Production Hindcast Data 35
      2.2.4 Downloading NDBC data 42
      2.2.5 Downloading NOAA CO-OPS Data 46

3. CONCLUSIONS 50

ACKNOWLEDGMENTS 50

REFERENCES 50

FIGURES

1 A graphic depicting the cascade of information from lower-resolution global models to higher-resolution regional, tactical, and local models. In order to resolve smaller-scale dynamics in shallow water near the coast, hydrodynamic models are typically run by nesting multiple scales where boundary conditions for the higher-resolution models are generated from the lower-resolution model output. 2

2 A mesoscale ocean basin model of the western Atlantic Ocean is driven by waves and currents downloaded from a global model of the entire Atlantic Ocean at the purple boundaries and outputs waves and currents at 1 km spacing. A mesoscale regional ocean model is driven by waves and currents extracted from the mesoscale US East ocean model at the orange boundaries and outputs waves and currents at 100 m spacing. A synoptic scale tactical coastal model is driven by waves and currents extracted from the mesoscale regional model at the pink boundaries and outputs waves and currents at 10 m spacing. The highest-resolution output is a plot of the current magnitude as a large nor'easter storm passes by. 2

3 A flowchart of the technical approach 3

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

5 NCOM Regions available on NCEI 6

6 WaveWatch III Production Hindcast 30-minute regions available on the NCEP web server; a) global 30 min [ID: glo_30m], b) Arctic Ocean 30 min [ID: ao_30m] 9

7 WaveWatch III Production Hindcast 10-minute regions available on the NCEP web server; a) NW Atlantic 10 min [ID: at_10m], b) US West Coast 10 min [ID: wc_30m], c) Alaskan 10 min [ID: ak_10m], d) East Pacific 10 min [ID: ep_10m] 9

8 WaveWatch III Production Hindcast 4-minute regions available on the NCEP web server; a) Gulf of Mexico and NW Atlantic 4 min [ID: at_4m], b) Alaskan 4 min [ID: ak_4m], c) US West Coast 4 min [ID: wc_4m] 10

9 Buoy stations available from NDBC 11

10 Approximate number and locations of stations available through NOAA CO-OPS 12

11 Example output plots from getNCEI_ncom, (a) water depth and (b) depth-averaged current magnitude 13

12 Example output from getGEBCO_bathy in which water depth is shown 28

13 Example output plots from getNCEP_ww3, (a) water depth, (b) significant wave height, and (c) peak wave period 35

TABLES

1 HYCOM Region Data Descriptions..... 5

2 HYCOM Dataset Names and Variables Available..... 5

3 NCOM Region Data Descriptions..... 6

4 NCOM Datasets and Variables Available on NCEI..... 7

5 WaveWatch III Production Hindcast Model Grid Descriptions..... 8

This page intentionally left blank

EXECUTIVE SUMMARY

Meteorological and oceanographic data are needed to drive and validate hydrodynamic models. These data can be obtained from publicly available online sources; however, the processes required for users to acquire and format the data vary greatly between sources. Therefore, manually downloading and reformatting data from different sources can be cumbersome and time consuming. A Python toolkit is presented here that allows for the automatic retrieval and reformatting of meteorological and oceanographic data from common online sources. The toolkit provides an efficient and streamlined method for obtaining atmospheric and hydrodynamic data used in model development, validation, and/or testing.

This page intentionally left blank

GETDATA: A TOOLKIT FOR AUTOMATED RETRIEVAL AND FORMATTING OF OCEANOGRAPHIC AND ATMOSPHERIC DATA FROM ONLINE SOURCES

1. INTRODUCTION

1.1 Problem and Objective

The skill and accuracy of nearshore hydrodynamic models are dependent on the fidelity of their boundary conditions and initial conditions. These conditions are typically provided by regional and global atmospheric and oceanographic models whose output is stored on web servers that are accessible through online GUIs or internet protocols and are often on multiple servers in varying formats. Manually downloading each of these datasets and reformatting to a usable format for an application can be cumbersome and time consuming for numerical modelers. The goal of the getData Toolkit was to develop an efficient, streamlined approach to download and reformat publicly available global and regional meteorologic and oceanographic (METOC) data within a given region of interest and over a specific time period typically needed to drive nearshore numerical models.

1.2 Background

The US Naval Research Laboratory develops atmospheric and ocean wave and circulation models to provide forecasts of the battlespace environment. Global models simulate the world's ocean dynamics, but are run at a low resolution that does not resolve smaller-scale processes occurring in shallow waters near the coast, where the hydrodynamics are affected by the temporally and spatially varying bathymetry. Therefore, higher-resolution models are required for accurately simulating the shallow-water coastal environment. To address this problem, information is cascaded from global models to regional models to tactical-scale models to local models via simulation nesting. The output from a lower-resolution model is used to drive a higher-resolution model at its surface and lateral boundaries (Fig. 1) to resolve the smaller-scale dynamics occurring near the coast.

A practical example of ocean model nesting at Duck, North Carolina, is shown in Fig. 2. Output from the Navy Coastal Ocean Model (NCOM) with a domain size of 825 km by 1,000 km drives a regional model with a domain size of 17.5 km by 48 km. The output from the regional model then drives a tactical model with a domain size of 2 km by 4 km that can resolve scales of ocean dynamics O(20 m). Many years of hindcasts and operational nowcasts of global and regional models are publicly available through online web servers. However, due to the amount of output and the expertise required to set up a complex, high-resolution tactical-scale model, that data is often not available. Model developers needed automated and efficient algorithms to obtain global and regional model output in a usable format to generate boundary conditions for high-resolution coastal model forecasts. In addition, observations for ground truth data are needed to validate model output. Developers also needed robust algorithms to download wave buoy and observation station data to validate model output and to allow for sensitivity testing and optimization of model parameters.

Manuscript approved March 21, 2024.

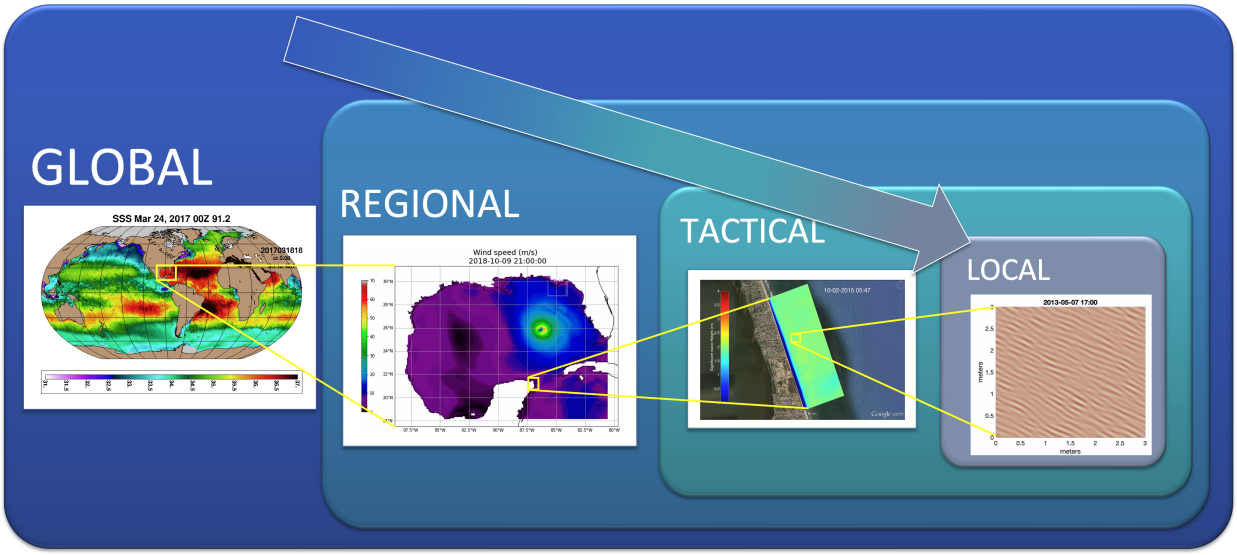


Fig. 1—A graphic depicting the cascade of information from lower-resolution global models to higher-resolution regional, tactical, and local models. In order to resolve smaller-scale dynamics in shallow water near the coast, hydrodynamic models are typically run by nesting multiple scales where boundary conditions for the higher-resolution models are generated from the lower-resolution model output.

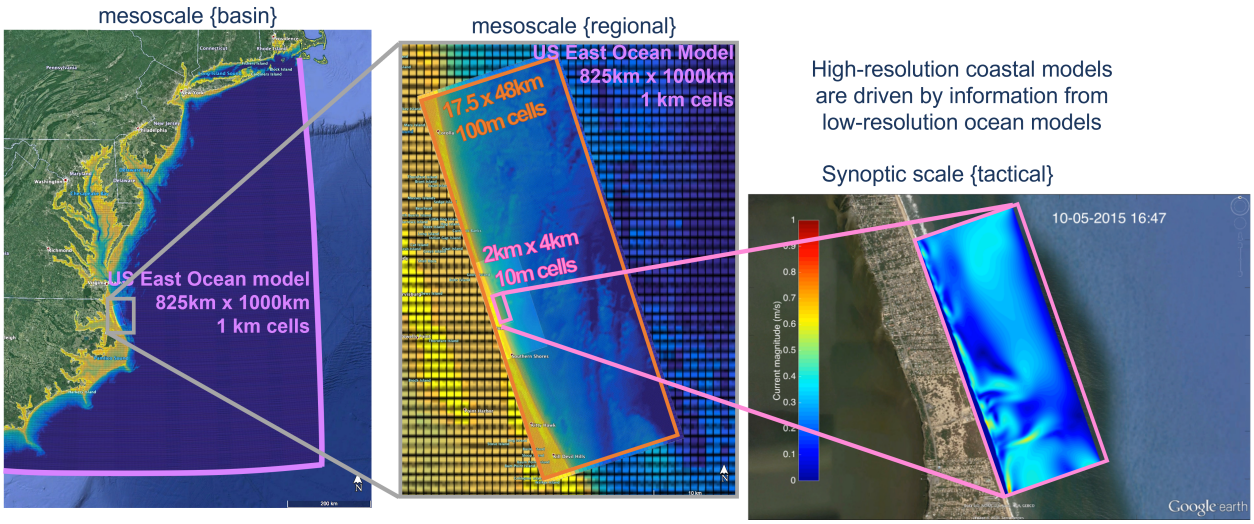


Fig. 2—A mesoscale ocean basin model of the western Atlantic Ocean is driven by waves and currents downloaded from a global model of the entire Atlantic Ocean at the purple boundaries and outputs waves and currents at 1 km spacing. A mesoscale regional ocean model is driven by waves and currents extracted from the mesoscale US East ocean model at the orange boundaries and outputs waves and currents at 100 m spacing. A synoptic scale tactical coastal model is driven by waves and currents extracted from the mesoscale regional model at the pink boundaries and outputs waves and currents at 10 m spacing. The highest-resolution output is a plot of the current magnitude as a large nor'easter storm passes by.

The getData Toolkit provides an automated, robust suite of algorithms that can reduce the time needed to set up complex, high-resolution, nested models by streamlining the acquisition of online operational model output and providing a standardized format for the generation of boundary conditions.

2. METHODOLOGY

The following steps outline the development of the getData Toolkit:

- Determine what oceanographic data is available on publicly accessible web servers.
- Write a suite of algorithms to download oceanographic data from the web server using simple user input files.
- Save the data in a standardized format.

A flowchart illustrating the approach is shown in Fig. 3. First, information about each of the web servers was collected. Next, Python algorithms were written to easily download, process, and reformat data via the command line and a simple user input file and to save it into a network Common Data Form (netCDF) and a comma-separated values (CSV) file. Lastly, plots were generated to visualize the downloaded data. The following sections describe in detail the web servers and the code generated to download the data.

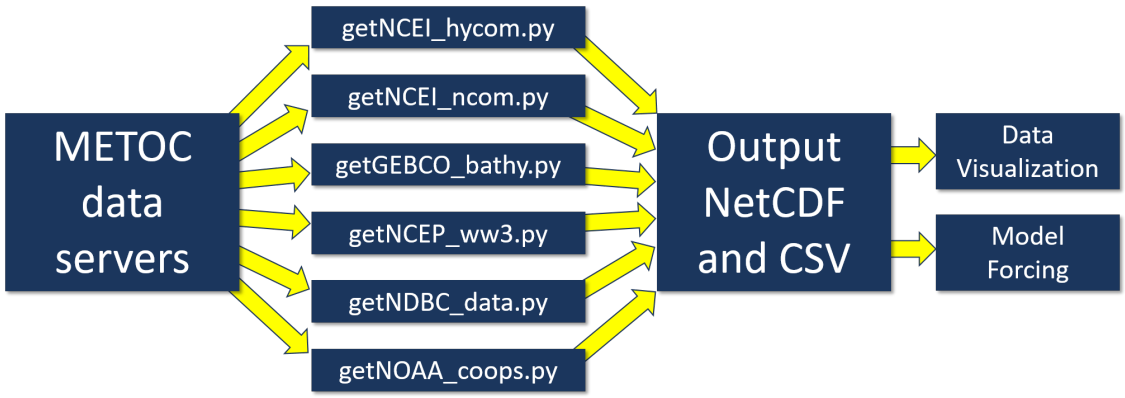


Fig. 3—A flowchart of the technical approach

2.1 Web Servers

The toolkit utilizes several different web servers containing publicly available data hosted by the National Oceanographic and Atmospheric Administration (NOAA) and the General Bathymetric Chart of the Oceans (GEBCO). The web servers include The National Centers for Environmental Information (NCEI), The National Centers for Environmental Prediction (NCEP), The National Data Buoy Center (NDBC), and The Center for Operational Oceanographic Products and Services (CO-OPS), all described in the following sections.

2.1.1 National Centers for Environmental Information (NCEI)

The National Centers for Environmental Information (`https://www.ncei.noaa.gov/`) provide both historical and near-real-time weather and climate forecast model data. The available models include operational

global models developed by NRL. The two models used here are the Navy Global Hybrid Coordinate Ocean Model (HYCOM) [1] and the Navy Coastal Ocean Model (NCOM) [2]. The data is accessible through an Environmental Research Division's Data Access Program (ERRDAP) [3] 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` and are described in the following sections.

2.1.1.1 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). The model outputs several ocean variables, including surface elevation and water temperature. There are five regions publicly available on NCEI (Fig. 4) that output oceanographic predictions every 3 hours at a resolution of $1/12°$. Tables 1 and 2 give the specific model dataset names and times available, and the names and dimensionalities of model outputs, respectively.

Spatial resolution: $1/12°$

Temporal output resolution: 3 hours

Regions:

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

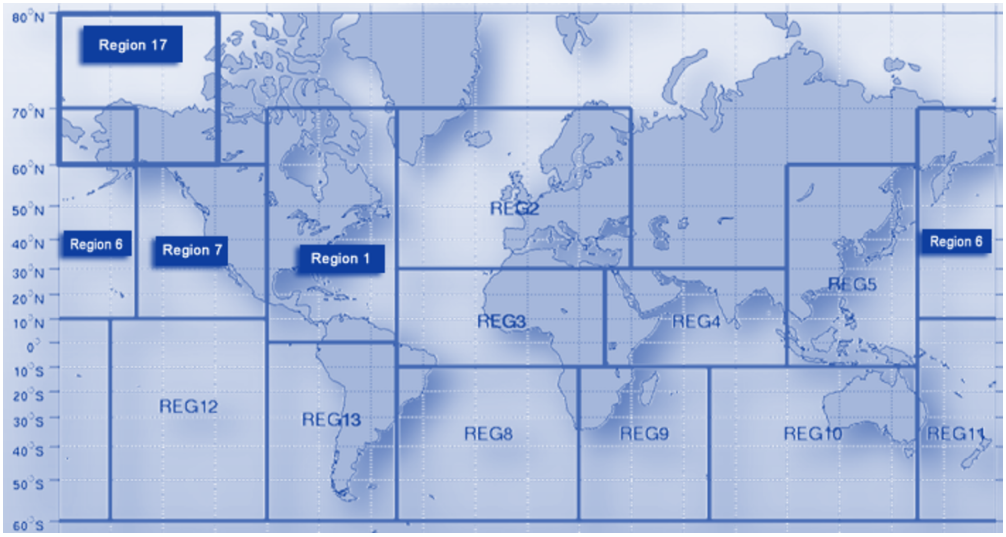


Fig. 4—HYCOM Regions available on the NCEI web server (in highlighted boxes)

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 Dataset Names and Variables Available

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

In the dataset name, *N* is the region number from Table 1.

2.1.1.2 Regional Navy Coastal Ocean Model (NCOM)

The Fleet Numerical Meteorology and Oceanography Center (FNMOC) runs regional simulations of the Navy Coastal Ocean Model (NCOM) in the Gulf of Mexico and the Caribbean Sea, the U.S East Coast, and the northeastern Pacific Ocean (including the Gulf of Alaska); see Fig. 5. The model outputs multiple ocean variables such as water surface elevation, surface atmospheric pressure, surface roughness, and others. The three regions publicly available on NCEI output oceanographic predictions every 3 hours at a resolution between $1/36°$ and $1/30°$. 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

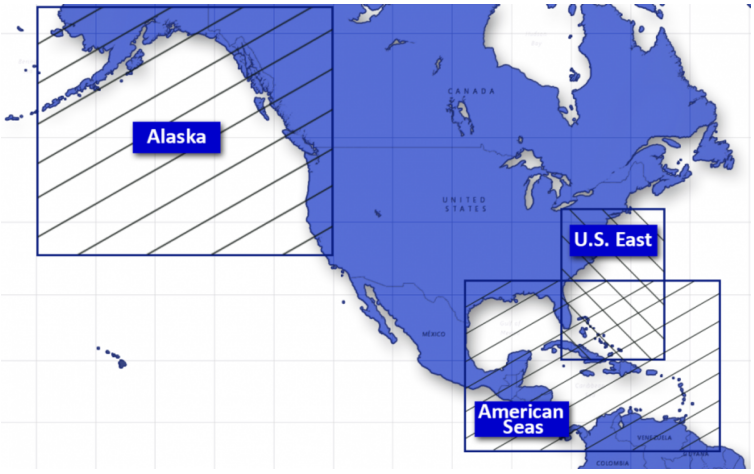


Fig. 5—NCOM Regions available on NCEI

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° 1/30°	04/05/2013 to 12/16/2020 12/17/2020 to present	20130405_to_20201216 latest
US East	us_east	1/36° 1/36° 1/30°	02/08/2009 to 11/18/2009 04/05/2013 to 12/16/2020 12/17/2020 to present	before_change 20130405_to_20201216 latest
Alaska	alaska	1/30°	03/05/2013 to present	latest

Table 4—NCOM Datasets and Variables Available on NCEI

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

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.

2.1.2 General Bathymetric Chart of the Oceans (GEBCO)

The General Bathymetric Chart of the Oceans (GEBCO) is located at `https://www.gebco.net/` [4] and provides publicly available bathymetric data sets. The bathymetry data is available in a variety of formats; however, getData only downloads the NetCDF file format that contains gridded bathymetric data.

Available operational products can be found at `https://www.gebco.net/data_and_products/`.

The GEBCO data download application is available at `https://download.gebco.net/`.

The GEBCO 2023 grid was the most recent grid at the time of this report. It provides global terrain elevation data in meters on a 15-arc-second (~450 m) grid. The global terrain elevation data includes both topographic and bathymetric data. GEBCO data is available for download in the form of a global coverage grid or as user-defined areas via the GEBCO data download application.

Variables:
- Latitude (lat) [degrees north]
- Longitude (lon) [degrees east]
- Elevation relative to sea level (elevation) [m]

2.1.3 National Centers for Environmental Prediction (NCEP)

The Environmental Modeling Center (EMC), located at `https://polar.ncep.noaa.gov/`, is a part of the National Centers for Environmental Prediction (NCEP), located at `https://www.weather.gov/ncep/`, and provides historical and near-real-time marine forecast and reanalysis model data. The EMC has a suite of operational wave models; however, getData only presently retrieves the WaveWatch III (WW3) Production Hindcast [5].

Available operational products can be found at `https://polar.ncep.noaa.gov/`.

The WaveWatch III Hindcast and Reanalysis archives are available at `https://polar.ncep.noaa.gov/waves/hindcasts/`.

2.1.3.1 WaveWatch III (WW3) Production Hindcast

The NCEP EMC runs the operational multigrid spectral wave model WaveWatchIII with operational NCEP winds and ice fields as boundary forcing. The WW3 Production Hindcast archive has output available from February 2005 to May 2019. The data is provided in nine rectilinear global and regional nested grids. WW3 outputs wind and wave predictions every 3 hours at grid resolutions from $1/15°$ to $1/2°$. The higher-resolution grids are only available along the coastal continental US. Table 5 shows the region names and their associated IDs, resolutions, and available time periods. Figures 6–8 depict the 30-minute, 10-minute, and 4-minute regions, respectively, that are available on NCEP.

Spatial resolution: $1/2°, 1/6°, 1/15°$

Temporal output resolution: 3 hours

Regions:
- Global 30 minute
- Arctic Ocean 30 minute
- NW Atlantic 10 minute
- US West Coast 10 minute
- East Pacific 10 minute
- Alaskan 10 minute
- Gulf of Mexico and NW Atlantic 4 minute
- US West Coast 4 minute
- Alaskan 4 minute

Variables:
- Time (valid_time) [datetime64]
- Latitude (latitude) [degrees north]
- Longitude (longitude) [degrees east]
- Significant wave height (swh) [m]
- Peak wave period (perpw) [s]
- Eastward wind velocity (u) [m/s]
- Northward wind velocity (v) [m/s]
- Primary wave direction (dirpw) [degrees true]

Table 5—WaveWatch III Production Hindcast Model Grid Descriptions

Region Name	ID	Resolution (x)	Resolution (y)	Time period available
Global 30 min	glo_30m	1/2°	1/2°	Feb 2005 - May 2019
Arctic Ocean 30 min	ao_30m	1/2°	1/2°	Feb 2005 - May 2019
NW Atlantic 10 min	at_10m	1/6°	1/6°	Feb 2005 - May 2019
US West Coast 10 min	wc_10m	1/6°	1/6°	Feb 2005 - May 2019
East Pacific 10 min	ep_10m	1/6°	1/6°	Feb 2005 - May 2019
Alaskan 10 min	ak_10m	1.5/6°	1/6°	Feb 2005 - May 2019
Gulf of Mexico and NW Atlantic 4 min	at_4m	1/15°	1/15°	Feb 2005 - May 2019
US West Coast 4 min	wc_4m	1/15°	1/15°	Feb 2005 - May 2019
Alaskan 4 min	ak_4m	2/15°	1/15°	Feb 2005 - May 2019

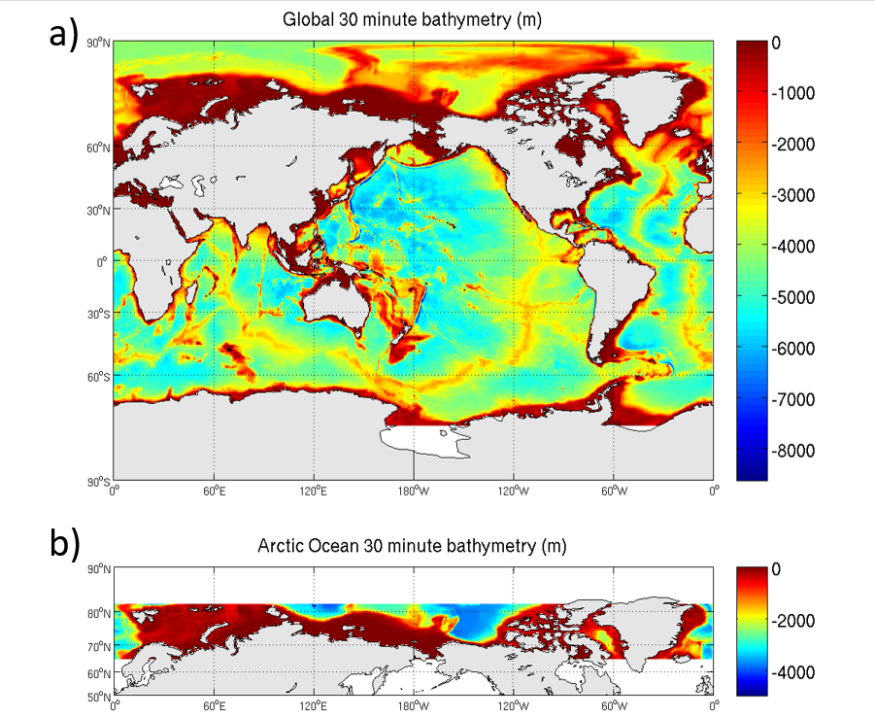


Fig. 6—WaveWatch III Production Hindcast 30-minute regions available on the NCEP web server; a) global 30 min [ID: glo_30m], b) Arctic Ocean 30 min [ID: ao_30m]

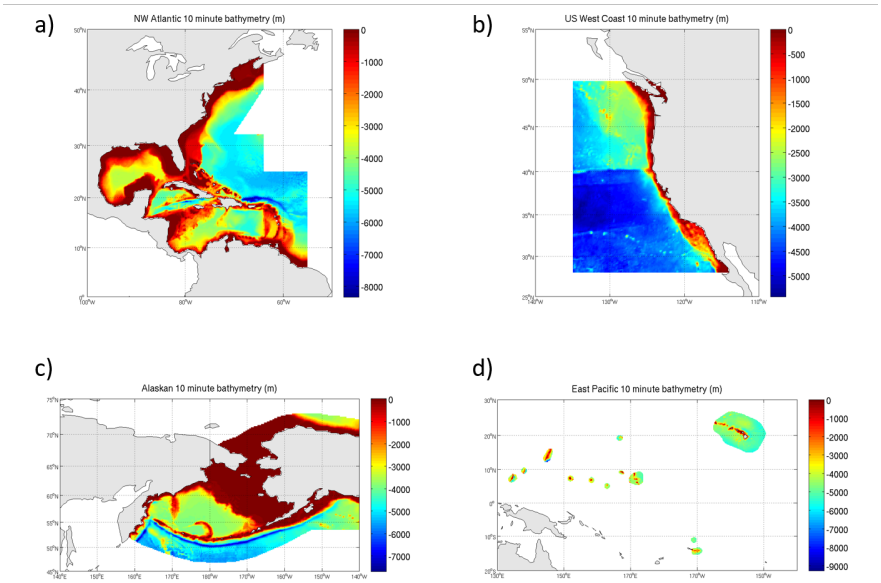


Fig. 7—WaveWatch III Production Hindcast 10-minute regions available on the NCEP web server; a) NW Atlantic 10 min [ID: at_10m], b) US West Coast 10 min [ID: wc_30m], c) Alaskan 10 min [ID: ak_10m], d) East Pacific 10 min [ID: ep_10m]

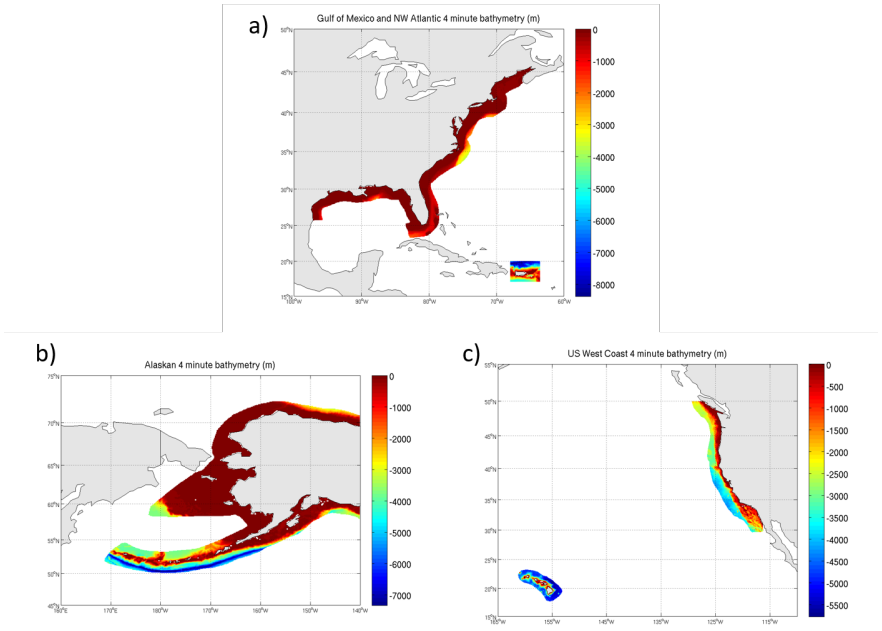


Fig. 8—WaveWatch III Production Hindcast 4-minute regions available on the NCEP web server; a) Gulf of Mexico and NW Atlantic 4 min [ID: at_4m], b) Alaskan 4 min [ID: ak_4m], c) US West Coast 4 min [ID: wc_4m]

2.1.4 National Data Buoy Center (NDBC)

The National Data Buoy Center (NDBC) located at `https://www.ndbc.noaa.gov/` [6] is a sustainable and resilient marine observation and monitoring infrastructure that enhances healthy ecosystems, communities, and economies. The server includes data collected from buoys around the globe (see Fig. 9) 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 is specific to each individual station. Presently, the getData toolkit only retrieves significant wave height from the standard meteorological data (stdmet); however, there are many datasets available globally.

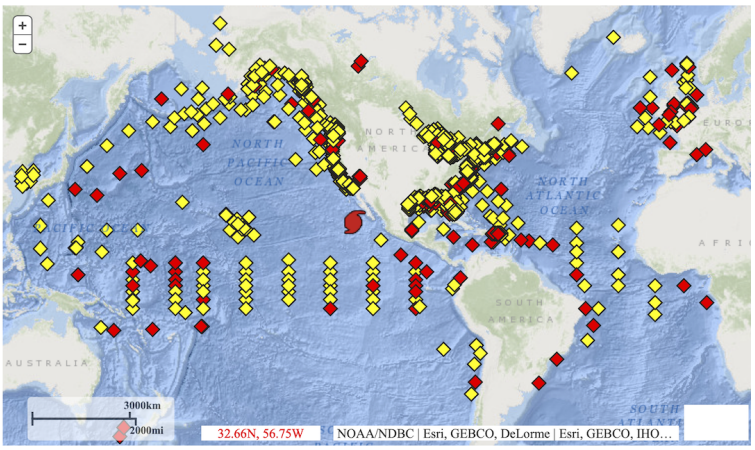


Fig. 9—Buoy stations available from NDBC

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)

2.1.5 NOAA CO-OPS

The Center for Operational Oceanographic Products and Services (CO-OPS) [7] 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 the Great Lakes, an integrated system of real-time sensors concentrated in busy seaports, and temporary meters that collect observations for tidal current predictions (see Fig. 10). 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/`

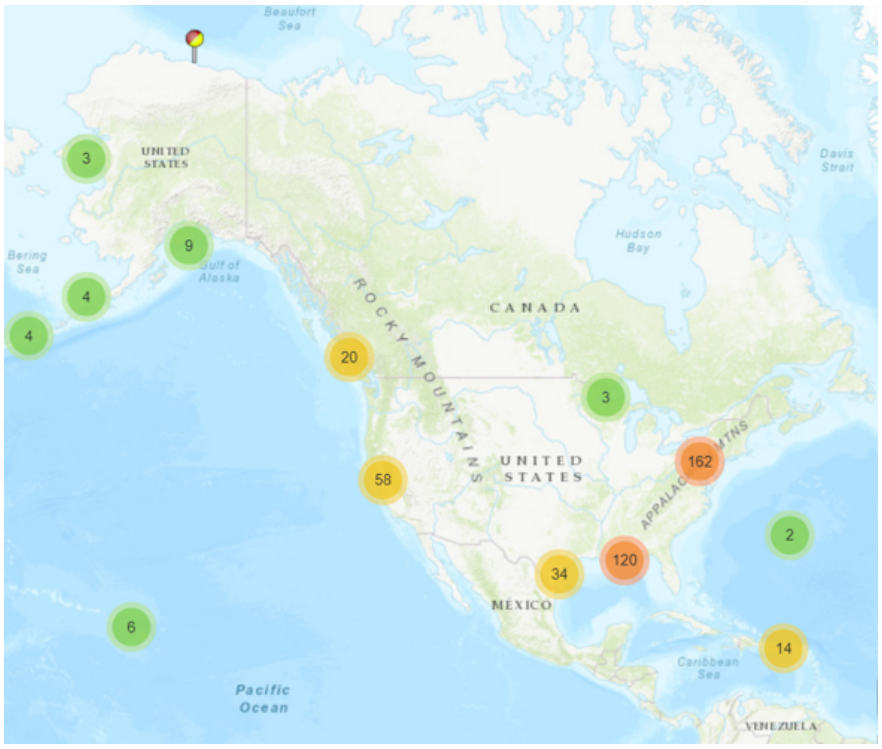


Fig. 10—Approximate number and locations of stations available through NOAA CO-OPS

Variables:

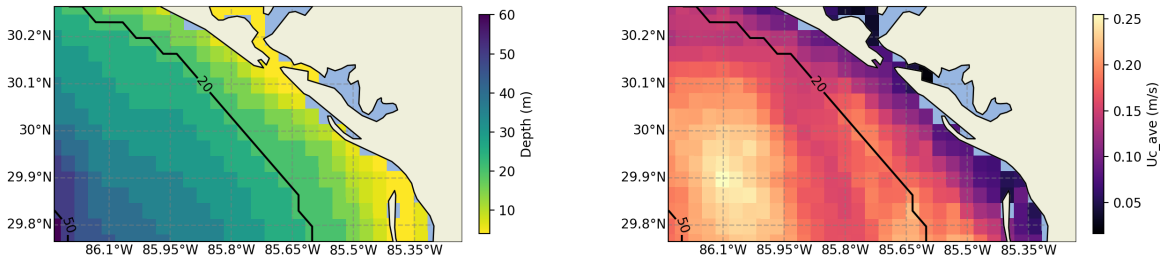
- Water level (water_level)
- Air temperature (air_temperature)
- Water temperature (water_temperature)
- Wind (wind)
- Air pressure (air_pressure)
- Air gap (air_gap)
- Conductivity (conductivity)
- Visibility (visibility)
- Humidity (humidity)
- Salinity (salinity)
- Hourly height (hourly_height)
- 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 Data Downoad Codes

After the online sources and data were identified, an efficient and streamlined framework for retrieving the data was developed from a suite of algorithms. The algorithms read user input of the requested location, times, and types of data from a YAML file and retrieve the requested data. YAML is a data serialization language that is often used for writing configuration files. The algorithm then saves the data as both a netCDF file and a CSV file. The netCDF and/or the CSV can then be used to generate boundary conditions for nested high-resolution models.

2.2.1 Downloading NCEI HYCOM and NCOM Data

The `getNCEI_hycom.py` and `getNCEI_ncom.py` programs download HYCOM and NCOM model data, respectively, from the NCEI web server database and save them in a netCDF and CSV file. The location and time of the downloaded model data pulled is determined by user-defined settings read in through the YAML settings file (`getNCEI_hycom_settings.yml` or `getNCEI_ncom_settings.yml`). The output netCDF and CSV files are saved in the user-defined output directory in the YAML settings file. The `getNCEI_hycom.py` and `getNCEI_ncom.py` programs also output diagnostic plots of the water depth and depth-averaged current magnitude values (Fig. 11).



(a) (b)

Fig. 11—Example output plots from getNCEI_ncom, (a) water depth and (b) depth-averaged current magnitude

The following command will run the program:

```
>> python getNCEI_hycom.py -y INPUT_FILE
```

or

```
>> python getNCEI_ncom.py -y INPUT_FILE
```

where

```
INPUT_FILE : YAML settings file name with requested parameters
(DEFAULT = getNCEI_hycom_settings.yml or getNCEI_ncom_settings.yml)
```

The user must input the following information into the `getNCEI_hycom_settings.yml` or `getNCEI_ncom_settings.yml` file:

- lat_min: the minimum of the latitude range
- lat_max: the maximum of the latitude range
- lon_min: the minimum of the longitude range
- lon_max: the maximum of the longitude range
- start_time: the beginning of the time range in the format YYYY-MM-DD HH:MM:SS
- end_time: the end of the time range in the format YYYY-MM-DD HH:MM:SS
- depth_min: the minimum water depth (depth ranges from 0 to 5000 m)
- depth_max: the maximum water depth (depth ranges from 0 to 5000 m)
- output_dir: path to the output directory relative to the getNCEI_hycom.py or getNCEI_ncom.py directory
- output_file_name: name of the output file excluding the file extension

The `getNCEI_hycom.py` and `getNCEI_ncom.py` programs are capable of obtaining both 2D and 3D variables. Surface elevation (`surf_el`) is a 2D variable and is needed to generate water-level boundary conditions. Eastward (`water_u`) and northward (`water_v`)) current velocities are 3D and are needed to generate velocity boundary conditions. Since users commonly need both 2D and 3D data, the `getNCEI_hycom.py` and `getNCEI_ncom.py` scripts automatically retrieve both. It first downloads the 2D surface elevation data, then downloads the 3D `water_u` and `water_v` data. The algorithm then merges them together into a single dataset and performs some initial processing. Current magnitude (U_c) is a commonly needed parameter for calculating bed shear stress. The equation for current magnitude Eq. (1) uses eastward and northward current velocities; however, because the NCEI HYCOM and NCOM datasets contain eastward and northward current velocity values throughout the water column, it is necessary either to use the bottom velocities or to take the depth average of the water velocities at each point in time. Here, both eastward and northward bottom velocities are downloaded as well as the depth-averaged values so that either could be used in the calculation of U_c, where u_w is either the bottom or depth-averaged eastward current velocity and v_w is either the bottom or depth-averaged northward current velocity. Also, the original time metadata (datetime64 format) is converted to UNIX timestamps. These additional calculations are included in the output netCDF and CSV files.

$$U_c = \sqrt{u_w^2 + v_w^2} \tag{1}$$

An example of a YAML settings input file for downloading HYCOM model data for a defined area and time range is shown below.

Listing 2.1: `getNCEI_hycom_settings.yml` Input File for the `getNCEI_hycom.py` Algorithm

```
lat_min: 29.75               # minimum latitude
lat_max: 30.25               # maximum latitude
lon_min: 273.75               # minimum longitude
lon_max: 274.75               # maximum longitude
start_time: 2015-05-04 00:00:00     # YYYY-MM-DD HH:MM:SS
end_time: 2015-05-07 00:00:00       # YYYY-MM-DD HH:MM:SS
depth_min: 0            # minimum water depth (depth ranges from 0-5000 m)
depth_max: 5000         # maximum water depth (depth ranges from 0-5000 m)
output_dir: "./output"      # output directory path relative to the getNCEI_hycom.py
    directory
output_file_name: "TREX_04may2015-07may2015_NCEI_hycom"   # output file name; please
    do not include a file extension
```

The code automatically calculates which HYCOM or NCOM region, respectively, to pull data from based on the user-defined latitude and longitude bounds in the settings file. A warning is displayed if the user's defined area and/or time range are beyond the limits of the available HYCOM or NCOM data.

The output netCDF and CSV files contain the following variables:

- Time (time) [POSIX timestamp (Unix)]
- Latitude (lat) [degrees north]
- Longitude (lon) [degrees east]
- Depth (depth) [m]
- Water surface elevation (surf_el) [m]
- East velocity component of depth-averaged current (uc_x_ave) [m/s]
- East velocity component of current at model max depth (uc_x_bot) [m/s]
- North velocity component of depth-averaged current (uc_y_ave) [m/s]
- North velocity component of current at model max depth (uc_y_bot) [m/s]
- Depth averaged current magnitude (Uc_ave) [m/s]
- Current magnitude at model max depth (Uc_bot) [m/s]

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

```
'''
Authors: Ryan Phillip
'''

import yaml
import pathlib
import argparse
import datetime
import warnings
from xarray import open_dataset
import os
import numpy as np
import pandas as pd
from erddapClient import ERDDAP_Griddap

from getData_plotting_tools import plot_Uc_ave, plot_depth



def getNCEI_hycom(start_time, end_time, lat_min, lat_max, lon_min, lon_max,
                  depth_min, depth_max, output_dir, output_file_name):
    """
    Parameters
    ----------
    start_time : datetime
        data start date and time.
    end_time : datetime
        data end date and time.
    lat_min : float
        minimum latitude.
    lat_max : float
        maximum latitude.
    lon_min : float
        minimum longitude.
    lon_max : float
```

```
        maximum longitude.
    depth_min : float
        minimum water depth (m).
    depth_max : float
        maximum water depth (m).
    output_dir : string
        filepath to the preprocessing output directory.
    output_file_name : string
        filename of the output NetCDF file.

    Raises
    ------
    ValueError
        User-defined parameter(s) not in range of NCEI data.

    Returns
    -------
    a generated YAML settings file for downloading NCEI data.
    """
    warnings.simplefilter(action='ignore', category=FutureWarning)

    # record starting working directory
    og_dir = os.getcwd()

    # convert negative coordinates to positive
    if lon_min < 0:
        lon_min = 360 - abs(lon_min)
    if lon_max < 0:
        lon_max = 360 - abs(lon_max)

    # 3D HYCOM coordinate boundaries
    # latitudes are Degrees North and longitudes are Degrees East
    reg1_minlat, reg1_maxlat = 0.0, 70.0
    reg1_minlon, reg1_maxlon = 360-99.99996948242188, 360-50.0

    reg6_minlat, reg6_maxlat = 10.0, 70.0
    reg6_minlon, reg6_maxlon = 150.00001525878906, 210.0

    reg7_minlat, reg7_maxlat = 10.0, 60.0
    reg7_minlon, reg7_maxlon = 360-149.99996948242188, 360-100.0

    reg17_minlat, reg17_maxlat = 60.0, 80.0
    reg17_minlon, reg17_maxlon = 360-179.99996948242188, 360-120.0


    # determine region of user-defined coordinates
    if lat_min >= reg1_minlat and lat_max <= reg1_maxlat and \
        lon_min >= reg1_minlon and lon_max <= reg1_maxlon:
            region = 'reg1'
    elif lat_min >= reg6_minlat and lat_max <= reg6_maxlat and \
        lon_min >= reg6_minlon and lon_max <= reg6_maxlon:
            region = 'reg6'
    elif lat_min >= reg7_minlat and lat_max <= reg7_maxlat and \
        lon_min >= reg7_minlon and lon_max <= reg7_maxlon:
            region = 'reg7'
    elif lat_min >= reg17_minlat and lat_max <= reg17_maxlat and \
        lon_min >= reg17_minlon and lon_max <= reg17_maxlon:
```

```
            region = 'reg17'
    else:
        raise ValueError('User-defined area not in range of HYCOM regions')



    # instance of datetime.datetime class
    dt = datetime.datetime

    # 3D HYCOM time coverages
    reg1_start = dt.strptime('2015-03-28 03:00:00', '%Y-%m-%d %H:%M:%S')
    reg1_end = dt.now()

    reg6_start = dt.strptime('2015-03-28 00:00:00', '%Y-%m-%d %H:%M:%S')
    reg6_end = dt.now()

    reg7_start = dt.strptime('2013-03-05 00:00:00', '%Y-%m-%d %H:%M:%S')
    reg7_end = dt.now()

    reg17_start = dt.strptime('2015-03-31 00:00:00', '%Y-%m-%d %H:%M:%S')
    reg17_end = dt.now()



    # determine time period for HYCOM dataset
    if region == 'reg1':
        if start_time >= reg1_start and end_time <= reg1_end:
            period = 'latest'
        else:
            raise ValueError('User-defined time not in range of HYCOM Region 1 time')
    elif region == 'reg6':
        if start_time >= reg6_start and end_time <= reg6_end:
            period = 'latest'
        else:
            raise ValueError('User-defined time not in range of HYCOM Region 6 time')
    elif region == 'reg7':
        if start_time >= reg7_start and end_time <= reg7_end:
            period = 'latest'
        else:
            raise ValueError('User-defined time not in range of HYCOM Region 7 time')
    elif region == 'reg17':
        if start_time >= reg17_start and end_time <= reg17_end:
            period = 'latest'
        else:
            raise ValueError('User-defined time not in range of HYCOM Region 17 time')
    else:
        raise ValueError('HYCOM time period could not be determined')


    # formulate HYCOM dataset name
    dataset_2d = f"HYCOM_{region}_{period}2d"
    dataset_3d = f"HYCOM_{region}_{period}3d"

    # change to the user-defined output directory
    os.chdir(output_dir)

    print('\n')
```

```
    print('Starting NCEI HYCOM data download')

    #Pull the data from the NCEI website
    remote_2d = ERDDAP_Griddap('https://www.ncei.noaa.gov/erddap', dataset_2d)
    remote_3d = ERDDAP_Griddap('https://www.ncei.noaa.gov/erddap', dataset_3d)

    # get subset of 2D dataset for surface elevation
    subsetncFile = ( remote_2d.setResultVariables(['surf_el'])
                    .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_2d}_{start_time_n}_{end_time_n}.nc"

    print(f'Downloaded: {fl}')
    fb = open(fl,'wb')
    fb.write(subsetncFile)
    fb.close()
    elev_ds = open_dataset(fl)

    print('\n')
    print('Finished downloading 2D NCEI files')
    print('Beginning download of 3D NCEI files')

    # get subset of 3D dataset for current velocities
    # Determine what data to pull by these parameters
    subsetncFile = ( remote_3d.setResultVariables(['water_u', 'water_v'])
                    .setSubset(time=slice(start_time,end_time),
                               depth = slice(depth_min,depth_max),
                               latitude=slice(lat_min, lat_max),
                               longitude=slice(lon_min, lon_max))
                    .getData(filetype='nc'))

    fn = f"NCEI_{dataset_3d}_{start_time_n}_{end_time_n}.nc"

    print(f'Downloaded: {fn}')
    #Open, write contents to file and close it
    fb = open(fn,'wb')
    fb.write(subsetncFile)
    fb.close()
    ds = open_dataset(fn)

    print('\n')
    print('Finished downloading 3D NCEI files')

    # merge the elevation data from the 2D dataset into the 3D dataset
    ds = elev_ds.merge(ds)

    ### pre-processing steps to match SeaMoRE convention
    print('\n')
    print('Beginning preliminary calculations and reformatting')

```

```
    # get dimension information (and convert datetime64 to timestamp)
    ctv = ds.time.copy(deep=False).to_numpy()
    ct = np.array([datetime.datetime.timestamp(pd.to_datetime(x)) for x in ctv])
    cd = ds.depth.copy(deep=False).to_numpy()
    clat = ds.latitude.copy(deep=False).to_numpy()
    clon = ds.longitude.copy(deep=False).to_numpy()

    # make meshgrids for indexing and calculations
    cdi, cti, clati, cloni = np.meshgrid(cd, ct, clat, clon)

    # get current velocities
    uc_x = ds.water_u.to_numpy()
    uc_y = ds.water_v.to_numpy()

    # take the means of the current velocities with depth
    with warnings.catch_warnings():
        warnings.filterwarnings(action = 'ignore',
                                message = 'Mean of empty slice')
        uc_x_ave = np.nanmean(uc_x, 1)
        uc_y_ave = np.nanmean(uc_y, 1)

    # select the bottom-most depth for each location
    with warnings.catch_warnings():
        warnings.filterwarnings(action = 'ignore',
                                message = 'All-NaN slice encountered')
        nan_ind = np.argwhere(np.isnan(uc_x))
        cdi_nan = cdi.copy()
        cdi_nan[nan_ind[:, 0], nan_ind[:, 1], nan_ind[:, 2], nan_ind[:, 3]] = np.nan
        cdm = np.nanmax(cdi_nan, 1)

    # determine the bottom current velocities
    uc_x_bot = np.empty([len(ct), len(clat), len(clon)])
    uc_y_bot = np.empty([len(ct), len(clat), len(clon)])
    for time_ind in range(0, len(ct)):
        for lat_ind in range(0, len(clat)):
            for lon_ind in range(0, len(clon)):
                for depth_ind in range(0, len(cd)):
                    if cdi[time_ind, depth_ind, lat_ind, lon_ind] == cdm[time_ind,
lat_ind, lon_ind]:
                        uc_x_bot[time_ind, lat_ind, lon_ind] = uc_x[time_ind,
depth_ind, lat_ind, lon_ind]
                        uc_y_bot[time_ind, lat_ind, lon_ind] = uc_y[time_ind,
depth_ind, lat_ind, lon_ind]
                    elif np.isnan(cdm[time_ind, lat_ind, lon_ind]):
                        uc_x_bot[time_ind, lat_ind, lon_ind] = np.nan
                        uc_y_bot[time_ind, lat_ind, lon_ind] = np.nan

    # calculate current magnitude from the depth-averaged current velocities
    Uc_ave = np.sqrt(uc_x_ave**2 + uc_y_ave**2)
    # calculate current magnitude from the bottom current velocities
    Uc_bot = np.sqrt(uc_x_bot**2 + uc_y_bot**2)
    # make a dictionary of data for conversion and export
    c_dict = {'time': cti[:,0,:,:].flatten(), 'lat': clati[:,0,:,:].flatten(),
              'lon': cloni[:,0,:,:].flatten(), 'depth': cdm.flatten(),
              'surf_el': ds.surf_el.values.flatten(),
              'uc_x_ave': uc_x_ave.flatten(), 'uc_x_bot': uc_x_bot.flatten(),
              'uc_y_ave': uc_y_ave.flatten(), 'uc_y_bot': uc_y_bot.flatten(),
```

```
260                 'Uc_ave': Uc_ave.flatten(), 'Uc_bot': Uc_bot.flatten()}
261
262     # make the dictionary into a dataframe and set the indices
263     c_df = pd.DataFrame(c_dict).set_index(['time', 'lat', 'lon'])
264     # convert the dataframe to an xarray dataset
265     c_ds = c_df.to_xarray()
266
267     # Write xarray dataset to netCDF file
268     c_ds.to_netcdf(f"{output_file_name}.nc")
269     c_df.to_csv(f"{output_file_name}.csv")
270
271     print('\n')
272     print('Calculations and reformatting successful')
273     print('NCEI pre-processing NetCDF file written successfully')
274
275     # close the datasets
276     ds.close()
277     elev_ds.close()
278     c_ds.close()
279
280     print('\n')
281     print('Datasets closed successfully')
282
283     # remove the original .nc files
284     # for file in os.listdir():
285     #     if fnmatch.fnmatch(file, f'NCEI_{dataset_2d}_{start_time.year}*') and file.
    endswith(".nc"):
286     #         os.remove(file)
287     # for file in os.listdir():
288     #     if fnmatch.fnmatch(file, f'NCEI_{dataset_3d}_{start_time.year}*') and file.
    endswith(".nc"):
289     #         os.remove(file)
290
291     # change back to the original working directory
292     os.chdir(og_dir)
293
294     print('\n')
295     print('Task Complete')
296
297
298
299
300 if __name__ == "__main__":
301
302     parser = argparse.ArgumentParser()
303
304     parser.add_argument('-y', '--yaml-file',
305                         help = 'YAML file containing input settings',
306                         metavar = 'FILE',
307                         default = 'getNCEI_hycom_settings.yml',
308                         type = pathlib.Path)
309
310     args = parser.parse_args()
311
312
313     # parse the input YAML file
314     with open(args.yaml_file, 'r') as fp:
```

```
    # with open("getNCEI_hycom_settings_test.yml", 'r') as fp:
        settings = yaml.safe_load(fp)

        # Set all the variables from the yaml file
        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']
        output_dir = settings['output_dir']
        output_file_name = settings['output_file_name']


    # run the getNCEI_hycom function from getData_tools
    getNCEI_hycom(start_time, end_time, lat_min, lat_max, lon_min, lon_max,
                  depth_min, depth_max, output_dir, output_file_name)


    # read the formatted NCEI HYCOM dataset in for plotting
    ds = open_dataset(f"{output_dir}/{output_file_name}.nc")
    df = ds.to_dataframe().reset_index()

    # plot depth-averaged current magnitude (Uc_ave)
    plot_Uc_ave(df, output_dir, output_file_name, plot_time_info=False)

    # plot water depth
    plot_depth(df, output_dir, output_file_name, plot_time_info=False)
```

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

```
'''
Authors: Ryan Phillip
'''

import yaml
import pathlib
import argparse
import datetime
import warnings
from xarray import open_dataset
import os
import numpy as np
import pandas as pd
from erddapClient import ERDDAP_Griddap

from getData_plotting_tools import plot_Uc_ave, plot_depth



def getNCEI_ncom(start_time, end_time, lat_min, lat_max, lon_min, lon_max,
                 depth_min, depth_max, output_dir, output_file_name):
    """
    Parameters
    ----------
```

```
    start_time : datetime
        data start date and time.
    end_time : datetime
        data end date and time.
    lat_min : float
        minimum latitude.
    lat_max : float
        maximum latitude.
    lon_min : float
        minimum longitude.
    lon_max : float
        maximum longitude.
    depth_min : float
        minimum water depth (m).
    depth_max : float
        maximum water depth (m).
    output_dir : string
        filepath to the preprocessing output directory.
    output_file_name : string
        filename of the output NetCDF file.

    Raises
    ------
    ValueError
        User-defined parameter(s) not in range of NCEI data.

    Returns
    -------
    a generated YAML settings file for downloading NCEI data.
    """
    warnings.simplefilter(action='ignore', category=FutureWarning)

    # record starting working directory
    og_dir = os.getcwd()

    # convert negative coordinates to positive
    if lon_min < 0:
        lon_min = 360 - abs(lon_min)
    if lon_max < 0:
        lon_max = 360 - abs(lon_max)

    # 3D NCOM coordinate boundaries
    # latitudes are Degrees North and longitudes are Degrees East
    alaska_minlat, alaska_maxlat = 36.45000076293945, 62.04743957519531
    alaska_minlon, alaska_maxlon = 360-170.0, 360-119.90501403808594

    amseas_minlat, amseas_maxlat = 5.0, 32.0972900390625
    amseas_minlon, amseas_maxlon = 360-98.0, 360-54.904296875

    useast_minlat, useast_maxlat = 20.0, 42.0977897644043
    useast_minlon, useast_maxlon = 360-82.0, 360-63.90179443359375

    # determine region of user-defined coordinates
    if lat_min >= alaska_minlat and lat_max <= alaska_maxlat and \
        lon_min >= alaska_minlon and lon_max <= alaska_maxlon:
            region = 'alaska'
    elif lat_min >= amseas_minlat and lat_max <= amseas_maxlat and \
```

```
        lon_min >= amseas_minlon and lon_max <= amseas_maxlon:
            region = 'amseas'
    elif lat_min >= useast_minlat and lat_max <= useast_maxlat and \
        lon_min >= useast_minlon and lon_max <= useast_maxlon:
            region = 'us_east'
    else:
        raise ValueError('User-defined area not in range of NCOM regions')

    # instance of datetime.datetime class
    dt = datetime.datetime

    # 3D NCOM time coverages
    alaska1_start = dt.strptime('2015-04-12 00:00:00', '%Y-%m-%d %H:%M:%S')
    alaska1_end = dt.strptime('2020-12-20 00:00:00', '%Y-%m-%d %H:%M:%S')
    alaska2_start = dt.strptime('2020-12-14 00:00:00', '%Y-%m-%d %H:%M:%S')
    alaska2_end = dt.now()

    amseas1_start = dt.strptime('2010-05-08 00:00:00', '%Y-%m-%d %H:%M:%S')
    amseas1_end = dt.strptime('2013-04-04 21:00:00', '%Y-%m-%d %H:%M:%S')
    amseas2_start = dt.strptime('2013-04-05 00:00:00', '%Y-%m-%d %H:%M:%S')
    amseas2_end = dt.strptime('2020-12-20 00:00:00', '%Y-%m-%d %H:%M:%S')
    amseas3_start = dt.strptime('2020-12-15 00:00:00', '%Y-%m-%d %H:%M:%S')
    amseas3_end = dt.now()

    useast1_start = dt.strptime('2009-02-08 00:00:00', '%Y-%m-%d %H:%M:%S')
    useast1_end = dt.strptime('2009-11-18 21:00:00', '%Y-%m-%d %H:%M:%S')
    useast2_start = dt.strptime('2009-11-19 00:00:00', '%Y-%m-%d %H:%M:%S')
    useast2_end = dt.strptime('2013-04-04 21:00:00', '%Y-%m-%d %H:%M:%S')
    useast3_start = dt.strptime('2013-04-05 00:00:00','%Y-%m-%d %H:%M:%S')
    useast3_end = dt.strptime('2020-12-20 00:00:00', '%Y-%m-%d %H:%M:%S')
    useast4_start = dt.strptime('2020-12-15 00:00:00', '%Y-%m-%d %H:%M:%S')
    useast4_end = dt.now()

    # determine time period for NCOM dataset
    if region == 'alaska':
        if start_time >= alaska1_start and end_time <= alaska1_end:
            period = '20150412_to_20201216_'
        elif start_time >= alaska2_start and end_time <= alaska2_end:
            period = 'latest'
        else:
            raise ValueError('User-defined time not in range of NCOM ALASKA time')
    elif region == 'amseas':
        if start_time >= amseas1_start and end_time <= amseas1_end:
            period = '20100509_to_20130404_'
        elif start_time >= amseas2_start and end_time <= amseas2_end:
            period = '20130405_to_20201216_'
        elif start_time >= amseas3_start and end_time <= amseas3_end:
            period = 'latest'
        else:
            raise ValueError('User-defined time not in range of NCOM AMSEAS time')
    elif region == 'useast':
        if start_time >= useast1_start and end_time <= useast1_end:
            period = 'before_change_'
        elif start_time >= useast2_start and end_time <= useast2_end:
            period = '20091119_to_20130404_'
        elif start_time >= useast3_start and end_time <= useast3_end:
            period = '20130405_to_20201216_'
```

```
        elif start_time >= useast4_start and end_time <= useast4_end:
            period = 'latest'
        else:
            raise ValueError('User-defined time not in range of NCOM AMSEAS time')
    else:
        raise ValueError('NCOM time period could not be determined')

    # formulate NCOM dataset name
    dataset_2d = f"NCOM_{region}_{period}2d"
    dataset_3d = f"NCOM_{region}_{period}3d"

    # change to the user-defined output directory
    os.chdir(output_dir)

    print('\n')
    print('Starting NCEI NCOM data download')

    #Pull the data from the NCEI website
    remote_2d = ERDDAP_Griddap('https://www.ncei.noaa.gov/erddap', dataset_2d)
    remote_3d = ERDDAP_Griddap('https://www.ncei.noaa.gov/erddap', dataset_3d)

    # get subset of 2D dataset for surface elevation
    subsetncFile = ( remote_2d.setResultVariables(['surf_el'])
                    .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_2d}_{start_time_n}_{end_time_n}.nc"

    print(f'Downloaded: {fl}')
    fb = open(fl,'wb')
    fb.write(subsetncFile)
    fb.close()
    elev_ds = open_dataset(fl)

    print('\n')
    print('Finished downloading 2D NCEI files')
    print('Beginning download of 3D NCEI files')

    # get subset of 3D dataset for current velocities
    # Determine what data to pull by these parameters
    subsetncFile = ( remote_3d.setResultVariables(['water_u', 'water_v'])
                    .setSubset(time=slice(start_time,end_time),
                               depth = slice(depth_min,depth_max),
                               latitude=slice(lat_min, lat_max),
                               longitude=slice(lon_min, lon_max))
                    .getData(filetype='nc'))

    fn = f"NCEI_{dataset_3d}_{start_time_n}_{end_time_n}.nc"

    print(f'Downloaded: {fn}')
    #Open, write contents to file and close it
```

```
    fb = open(fn,'wb')
    fb.write(subsetncFile)
    fb.close()
    ds = open_dataset(fn)

    print('\n')
    print('Finished downloading 3D NCEI files')

    # merge the elevation data from the 2D dataset into the 3D dataset
    ds = elev_ds.merge(ds)

    ### pre-processing steps to match SeaMoRE convention
    print('\n')
    print('Beginning preliminary calculations and reformatting')

    # get dimension information (and convert datetime64 to timestamp)
    ctv = ds.time.copy(deep=False).to_numpy()
    ct = np.array([datetime.datetime.timestamp(pd.to_datetime(x)) for x in ctv])
    cd = ds.depth.copy(deep=False).to_numpy()
    clat = ds.latitude.copy(deep=False).to_numpy()
    clon = ds.longitude.copy(deep=False).to_numpy()

    # make meshgrids for indexing and calculations
    cdi, cti, clati, cloni = np.meshgrid(cd, ct, clat, clon)

    # get current velocities
    uc_x = ds.water_u.to_numpy()
    uc_y = ds.water_v.to_numpy()

    # take the means of the current velocities with depth
    with warnings.catch_warnings():
        warnings.filterwarnings(action = 'ignore',
                                message = 'Mean of empty slice')
        uc_x_ave = np.nanmean(uc_x, 1)
        uc_y_ave = np.nanmean(uc_y, 1)

    # select the bottom-most depth for each location
    with warnings.catch_warnings():
        warnings.filterwarnings(action = 'ignore',
                                message = 'All-NaN slice encountered')
        nan_ind = np.argwhere(np.isnan(uc_x))
        cdi_nan = cdi.copy()
        cdi_nan[nan_ind[:, 0], nan_ind[:, 1], nan_ind[:, 2], nan_ind[:, 3]] = np.nan
        cdm = np.nanmax(cdi_nan, 1)

    # determine the bottom current velocities
    uc_x_bot = np.empty([len(ct), len(clat), len(clon)])
    uc_y_bot = np.empty([len(ct), len(clat), len(clon)])
    for time_ind in range(0, len(ct)):
        for lat_ind in range(0, len(clat)):
            for lon_ind in range(0, len(clon)):
                for depth_ind in range(0, len(cd)):
                    if cdi[time_ind, depth_ind, lat_ind, lon_ind] == cdm[time_ind,
lat_ind, lon_ind]:
                        uc_x_bot[time_ind, lat_ind, lon_ind] = uc_x[time_ind,
depth_ind, lat_ind, lon_ind]
```

```
                        uc_y_bot[time_ind, lat_ind, lon_ind] = uc_y[time_ind,
depth_ind, lat_ind, lon_ind]
                elif np.isnan(cdm[time_ind, lat_ind, lon_ind]):
                    uc_x_bot[time_ind, lat_ind, lon_ind] = np.nan
                    uc_y_bot[time_ind, lat_ind, lon_ind] = np.nan

    # calculate current magnitude from the depth-averaged current velocities
    Uc_ave = np.sqrt(uc_x_ave**2 + uc_y_ave**2)
    # calculate current magnitude from the bottom current velocities
    Uc_bot = np.sqrt(uc_x_bot**2 + uc_y_bot**2)
    # make a dictionary of data for conversion and export
    c_dict = {'time': cti[:,0,:,:].flatten(), 'lat': clati[:,0,:,:].flatten(),
              'lon': cloni[:,0,:,:].flatten(), 'depth': cdm.flatten(),
              'surf_el': ds.surf_el.values.flatten(),
              'uc_x_ave': uc_x_ave.flatten(), 'uc_x_bot': uc_x_bot.flatten(),
              'uc_y_ave': uc_y_ave.flatten(), 'uc_y_bot': uc_y_bot.flatten(),
              'Uc_ave': Uc_ave.flatten(), 'Uc_bot': Uc_bot.flatten()}

    # make the dictionary into a dataframe and set the indices
    c_df = pd.DataFrame(c_dict).set_index(['time', 'lat', 'lon'])
    # convert the dataframe to an xarray dataset
    c_ds = c_df.to_xarray()

    # Write xarray dataset to netCDF file
    c_ds.to_netcdf(f"{output_file_name}.nc")
    c_df.to_csv(f"{output_file_name}.csv")

    print('\n')
    print('Calculations and reformatting successful')
    print('NCEI pre-processing NetCDF file written successfully')

    # close the datasets
    ds.close()
    elev_ds.close()
    c_ds.close()

    print('\n')
    print('Datasets closed successfully')

    # remove the original .nc files
    # for file in os.listdir():
    #     if fnmatch.fnmatch(file, f'NCEI_{dataset_2d}_{start_time.year}*') and file.
endswith(".nc"):
    #         os.remove(file)
    # for file in os.listdir():
    #     if fnmatch.fnmatch(file, f'NCEI_{dataset_3d}_{start_time.year}*') and file.
endswith(".nc"):
    #         os.remove(file)

    # change back to the original working directory
    os.chdir(og_dir)

    print('\n')
    print('Task Complete')
```

```
if __name__ == "__main__":

    parser = argparse.ArgumentParser()

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

    args = parser.parse_args()


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

        # Set all the variables from the yaml file
        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']
        output_dir = settings['output_dir']
        output_file_name = settings['output_file_name']


    # run the getNCEI_ncom function from getData_tools
    getNCEI_ncom(start_time, end_time, lat_min, lat_max, lon_min, lon_max,
                 depth_min, depth_max, output_dir, output_file_name)


    # read the formatted NCEI NCOM dataset in for plotting
    ds = open_dataset(f"{output_dir}/{output_file_name}.nc")
    df = ds.to_dataframe().reset_index()

    # plot depth-averaged current magnitude (Uc_ave)
    plot_Uc_ave(df, output_dir, output_file_name, plot_time_info=False)

    # plot water depth
    plot_depth(df, output_dir, output_file_name, plot_time_info=False)
```

2.2.2 Downloading GEBCO Bathymetry Data

The `getGEBCO_bathy.py` program downloads GEBCO bathymetry data from the GEBCO Gridded Bathymetry Data Download web application and saves them in netCDF and CSV files. When called upon in another program, the getGEBCO_bathy function returns an Xarray dataset of the downloaded bathymetry data. The GEBCO bathymetry data region is determined by user-defined values read in from the YAML settings file (`getGEBCO_bathy_settings.yml`). The output netCDF and CSV files are saved in the user-defined output directory in the YAML settings file. The `getGEBCO_bathy.py` program also outputs a diagnostic plot of water depth values (Fig. 12).

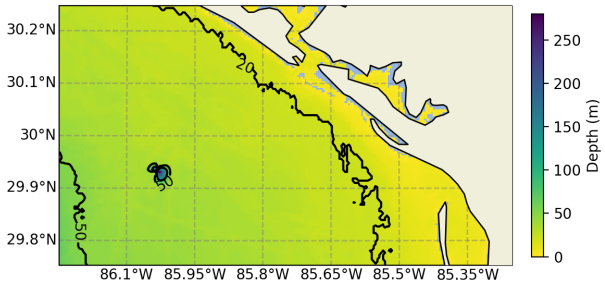


Fig. 12—Example output from getGEBCO_bathy in which water depth is shown

The algorithm runs with the following command:

```
>> python getGEBCO_bathy.py -y INPUT_FILE
```

where

```
INPUT_FILE : YAML settings file name with requested parameters
(DEFAULT = getGEBCO\_bathy\_settings.yml)
```

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

- lat_min: the minimum of the latitude range
- lat_max: the maximum of the latitude range
- lon_min: the minimum of the longitude range
- lon_max: the maximum of the longitude range
- gecko_driver: path to your Firefox geckodriver.exe file
- firefox_binary: path to your Firefox binary (firefox.exe file)
- output_dir: path to the output directory RELATIVE to the `getGEBCO_bathy.py` directory
- output_file_name: name of the output file excluding the file extension

The `getGEBCO_bathy.py` program automatically accesses and fills out the GEBCO Data Download web application form to download bathymetry data based on the user-defined latitude and longitude bounds in the YAML settings file. The Selenium Python package is used to access and manipulate the GEBCO web application. For the Selenium package to operate properly, the appropriate GeckoDriver must be installed on the user's system. Gecko is a web browser engine built into the Mozilla Firefox browser and acts as a link between Selenium and the Mozilla Firefox browser, both required for accessing the GEBCO data download web application through Python.

Obtain the GeckoDriver using the following steps:

1. Select the version of GeckoDriver for the user's operating system located at `https://github.com/mozilla/geckodriver/releases`.
2. Extract the contents from the downloaded GeckoDriver ZIP file.

3. Note the location of the file `geckodriver.exe` (or move it to a desired location) and define the path in the `getGEBCO_bathy_settings.yml` file.

The user must also note the location of their Mozilla Firefox binary (`firefox.exe`) file and define the path in the `getGEBCO_bathy_settings.yml` settings file. If the user does not have Mozilla Firefox installed on their system, it can be downloaded from `https://www.mozilla.org/` and installed by following the on-screen instructions. Common locations of the Mozilla Firefox executable include:

- Windows(64-bit): C:\Program Files\Mozilla Firefox\
- Windows(32-bit): C:\Program Files (x86)\Mozilla Firefox\
- MacOS: /Applications/Firefox or $HOME/Applications/Firefox
- Linux: /usr/bin/firefox

An example of a YAML settings input file for downloading GEBCO bathymetry for a defined area is shown below.

Listing 2.2: `getGEBCO_bathy_settings.yml` Input file for the `getGEBCO_bathy.py` Algorithm

```
lat_min: 29.75              # minimum latitude
lat_max: 30.25              # maximum latitude
lon_min: 273.75              # minimum longitude
lon_max: 274.75              # maximum longitude
gecko_driver: "C:/Program Files/Mozilla Firefox/geckodriver.exe"    # path to your
    Firefox geckodriver.exe file
firefox_binary: "C:/Program Files/Mozilla Firefox/firefox.exe"      # path to your
    Firefox binary (firefox.exe file)
output_dir: "./output"      # output directory filepath relative to the \texttt{
    getGEBCO\_bathy.py} directory
output_file_name: "TREX_04may2015-07may2015_NCEI_hycom"   # output file name; please
    do not include a file extension
```

The `getGEBCO_bathy.py` program will automatically open a Mozilla Firefox browser window. The code will autopopulate the data download form with user-defined values from the `getGEBCO_bathy_settings.yml` settings file. Please note that it is important not to manually manipulate the browser window while the code is running, as interfering with the browser window may cause the code to malfunction. The browser window will automatically close when the GEBCO data is finished downloading.

Note that the GEBCO Data Download web application has a data download limitation. If the size of the requested area is too large and/or the web application's processing queue is overloaded, the web application request will be queued to process at a later time instead of immediately. In this case, the web application will prompt the user for an email for notification of data-processing completion and the `getGEBCO_bathy.py` program will result in an error.

```
FileNotFoundError: [Errno 2] No such file or directory: '<GEBCO file name>.zip'
```

If the program returns the error, the user must decrease the size of the requested area and/or schedule the program to run during non-peak times (e.g., overnight).

The output netCDF and CSV files contain the following variables:

- Latitude (lat) [degrees north]
- Longitude (lon) [degrees east]
- Water depth (depth) [m]

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

```
'''
Authors: Ryan Phillip
'''

import yaml
import pathlib
import argparse
from xarray import open_dataset
import math
import numpy as np
import os
import shutil
import time
import re
from zipfile import ZipFile
import fnmatch
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.firefox.options import Options
from selenium.webdriver.support.wait import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

from getData_plotting_tools import plot_GEBCO_depth



def getGEBCO_bathy(gecko_driver, firefox_binary, lat_min, lat_max, lon_min,
                   lon_max, output_dir):
    """
    Parameters
    ----------
    gecko_driver : string
        path to Firefox geckodriver.exe file.
    firefox_binary : string
        path to Firefox binary (firefox.exe) file.
    lat_min : float
        minimum latitude.
    lat_max : float
        maximum latitude.
    lon_min : float
        minimum longitude.
    lon_max : float
        maximum longitude.
    output_dir : string
        filepath to preprocessing output directory.

    Returns
    -------
    bathy_ds : dataset
```

```
        xarray dataset of extracted GEBCO bathymetry data.
    """
    # record starting working directory
    og_dir = os.path.abspath(os.getcwd())

    print('\n')
    print('Setting up webdriver')
    # set Firefox as the webdriver and prepare to set custom options
    ser = Service(gecko_driver)
    op = Options()
    # save file to path defined for recent download with value 2
    op.set_preference("browser.download.folderList",2)
    # disable display Download Manager window with false value
    op.set_preference("browser.download.manager.showWhenStarting", False)
    # download location
    op.set_preference("browser.download.dir", og_dir)
    # specify Firefox binary location (firefox.exe)
    op.binary_location = firefox_binary
    # MIME set to save file to disk without asking file type to used to open file
    op.set_preference("browser.helperApps.neverAsk.saveToDisk",
                      "application/octet-stream,application/zip")
    # set geckodriver.exe path
    driver = webdriver.Firefox(service=ser, options=op)

    print('\n')
    print('Navigating to GEBCO url')
    # Navigate to GEBCO url
    driver.get("https://download.gebco.net/")

    # relative path to the "Enter Boundaries" max latitude box
    max_lat_xpath = '//*[@id="coordinates-card"]/div[2]/div[1]/div[1]/div/input'

    # relative path to the "Enter Boundaries" min latitude box
    min_lat_xpath = '//*[@id="coordinates-card"]/div[2]/div[3]/div[2]/div/input'

    # relative path to the "Enter Boundaries" max longitude box
    max_lon_xpath = '//*[@id="coordinates-card"]/div[2]/div[2]/div[3]/div/input'

    # relative path to the "Enter Boundaries" min longitude box
    min_lon_xpath = '//*[@id="coordinates-card"]/div[2]/div[2]/div[1]/div/input'

    # relative xpath to the 2D netCDF "Grid" checkbox
    grid_check_xpath = '//*[@id="gridCheck"]'

    # relative xpath to the "Add to basket" button
    add_bask_xpath = '//*[@id="sidebar-add-to-basket"]'

    # relative xpath to the "View basket" button
    view_bask_xpath = '//*[@id="data-selection-card"]/div[3]/button[2]'

    # relative xpath to the "Download your data" button
    dl_data_xpath = '//*[@id="basket-download-button"]'

    # convert lon_max to decimal degrees if necessary
    if lon_max > 180:
        lon_maxd = lon_max-360
    else:
```

```
        lon_maxd = lon_max

    # convert lon_min to decimal degrees if necessary
    if lon_min > 180:
        lon_mind = lon_min-360
    else:
        lon_mind = lon_min

    print('\n')
    print("Inputting values from user's YAML settings file")
    # wait for the web page to load before inputting values
    WebDriverWait(driver, 30).until(EC.presence_of_element_located((By.XPATH,
    max_lat_xpath)))

    # input lat_max value into bounds-north box
    time.sleep(1)
    driver.find_element(By.XPATH, max_lat_xpath).send_keys(Keys.BACKSPACE + str(
    lat_max))

    # input lat_min value into bounds-south box
    time.sleep(1)
    driver.find_element(By.XPATH, min_lat_xpath).send_keys(Keys.BACKSPACE + str(
    lat_min))

    # input lon_max value into bounds-east box
    time.sleep(1)
    driver.find_element(By.XPATH, max_lon_xpath).send_keys(Keys.BACKSPACE + str(
    lon_maxd))

    # input lon_min value into bounds-east box
    time.sleep(1)
    driver.find_element(By.XPATH, min_lon_xpath).send_keys(Keys.BACKSPACE + str(
    lon_mind))

    # check the 2D netCDF "Grid" checkbox
    time.sleep(1)
    driver.find_element(By.XPATH, grid_check_xpath).click()

    # implicitly wait before adding to basket
    time.sleep(1) # seconds

    # click the "Add to basket" button
    WebDriverWait(driver, 30).until(EC.element_to_be_clickable((By.XPATH,
    add_bask_xpath)))
    driver.find_element(By.XPATH, add_bask_xpath).click()

    # implicitly wait for the basket to be populated
    time.sleep(1) # seconds

    # click the "View basket" button
    WebDriverWait(driver, 30).until(EC.element_to_be_clickable((By.XPATH,
    view_bask_xpath)))
    driver.find_element(By.XPATH, view_bask_xpath).click()

    # click the "Download your data" button
    WebDriverWait(driver, 30).until(EC.element_to_be_clickable((By.XPATH,
    dl_data_xpath)))
```

```
    driver.find_element(By.XPATH, dl_data_xpath).click()

    print('\n')
    print('Downloading GEBCO data .zip file')
    print('Please wait. Browser will automatically close after download')
    # wait x seconds to download the zip file
    time.sleep(20) # seconds

    # quit the driver
    driver.quit()

    print('\n')
    print('Unzipping and extracting GEBCO data NetCDF file')
    # find the GEBCO zip file name
    for file in os.listdir():
        if fnmatch.fnmatch(file, 'GEBCO_*') and file.endswith(".zip"):
            zipfile_name = file.split('.')[0]

    # loading the temp.zip and creating a zip object
    with ZipFile(f"{zipfile_name}.zip", 'r') as zObject:
        # extract bathymetry NetCDF file from within the zip folder
        for info in zObject.infolist():
            if re.match(r'.*GEBCO.*\.nc$', info.filename):
                bathy_nc_path = info.filename
                zObject.extract(info.filename, og_dir)
    zObject.close()

    print('\n')
    print('Removing .zip folder')
    # remove the zip file after unzipping
    os.remove(f"{zipfile_name}.zip")

    print('\n')
    print('Moving bathymetry data NetCDF file to user-defined output directory')
    # move the bathy NetCDF from the unzipped folder to the user-defined output dir
    shutil.move(f"{bathy_nc_path}", f"{output_dir}/{bathy_nc_path.split('/')[1]}")

    print('\n')
    print('Removing empty unzipped folder')
    # remove the unzipped folder after extracting the bathymetry data
    shutil.rmtree(f"{bathy_nc_path.split('/')[0]}")

    # read in the bathy NetCDF
    bathy_ds = open_dataset(f"{output_dir}/{bathy_nc_path.split('/')[1]}")

    print('\n')
    print('Reformatting GEBCO bathymetry data')

    # convert any negative longitudes to positive (0-360 degrees)
    b_lon = []
    for x in bathy_ds['lon'].values:
        if x < 0:
            b_lon.append(360 + x)
        else:
            b_lon.append(x)
    bathy_ds['lon'] = b_lon

```

```
    # convert elevation (negatives) to depth (positives)
    b_dep = []
    for x in bathy_ds['elevation'].values.flatten():
        if x <= 0:
            b_dep.append(abs(x))
        else:
            b_dep.append(math.nan)
    bathy_ds['elevation'] = (['lat', 'lon'], np.array(b_dep).reshape(len(bathy_ds.lat), len(bathy_ds.lon)))

    # rename depth to match convention for SeaMoRE
    bathy_ds = bathy_ds.rename({'elevation': 'depth'})

    print('\n')
    print('Process Complete')

    return bathy_ds




if __name__ == "__main__":

    parser = argparse.ArgumentParser()


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


    args = parser.parse_args()


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

        # Set all the variables from the yaml file
        lat_min = settings['lat_min']
        lat_max = settings['lat_max']
        lon_min = settings['lon_min']
        lon_max = settings['lon_max']
        gecko_driver = settings['gecko_driver']
        firefox_binary = settings['firefox_binary']
        output_dir = settings['output_dir']
        output_file_name = settings['output_file_name']


    # run the getGEBCO_bathy function from getData_tools
    GEBCO_bathy_ds = getGEBCO_bathy(gecko_driver, firefox_binary, lat_min, lat_max, lon_min,
                                    lon_max, output_dir)

```

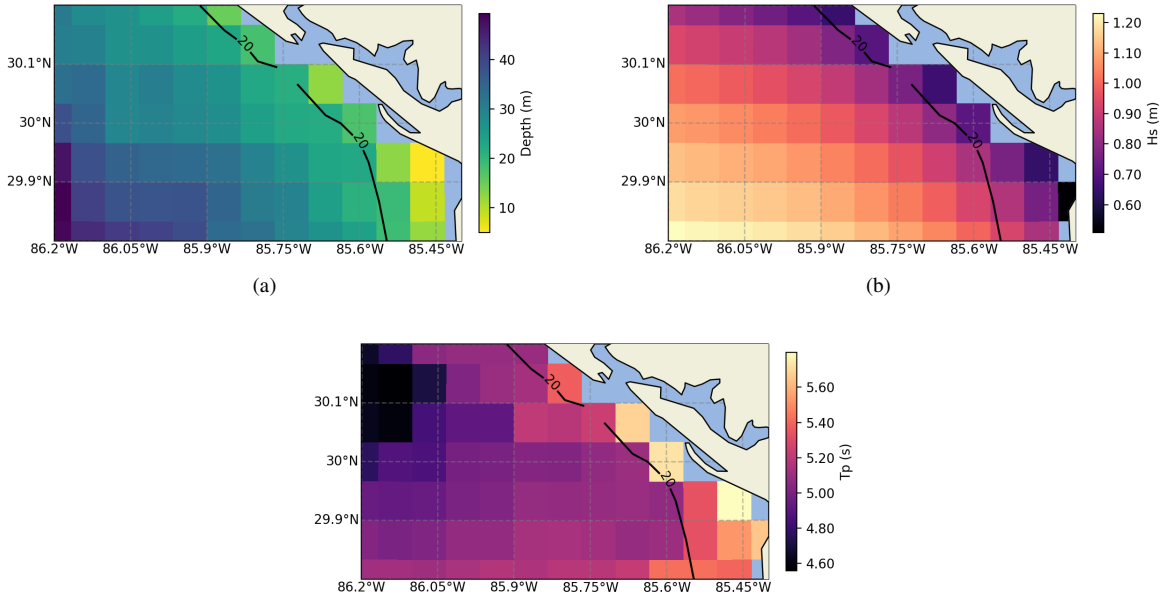
```
    print('\n')
    print('Exporting bathy data')
    # write and output the dataset to a NetCDF and a CSV
    GEBCO_bathy_ds.to_netcdf(f"{output_dir}/{output_file_name}.nc")
    print('Successfully output netCDF')
    GEBCO_bathy_df = GEBCO_bathy_ds.to_dataframe()
    GEBCO_bathy_df.to_csv(f"{output_dir}/{output_file_name}.csv")
    print('Successfully wrote CSV')
    GEBCO_bathy_ds.close()

    # plot water depth
    plot_GEBCO_depth(GEBCO_bathy_df.reset_index(), output_dir, output_file_name,
    plot_time_info=False)

    print('\n')
    print('Task Complete')
```

2.2.3 Downloading NCEP WaveWatch III Production Hindcast Data

The `getNCEP_ww3.py` program downloads WaveWatch III Production Hindcast data from the NCEP database and saves them in netCDF and CSV files. The model data to download is determined by user-defined settings read in from the YAML settings input file (`getNCEP_ww3_settings.yml`). The output netCDF and CSV files are saved in the user-specified output directory defined in the YAML settings file. The `getNCEP_ww3.py` program also outputs diagnostic plots of the water depth, significant wave height, and peak wave period values (Fig. 13).



(a) (b) (c)

Fig. 13—Example output plots from getNCEP_ww3, (a) water depth, (b) significant wave height, and (c) peak wave period

The algorithm is run with the following command:

```
>> python getNCEP_ww3.py -y INPUT_FILE
```

where

```
INPUT_FILE : YAML settings file name with requested parameters
(DEFAULT = getNCEP\_ww3\_settings.yml)
```

The user must input the following information into the `getNCEP_ww3_settings.yml` file:

- lat_min: the minimum of the latitude range
- lat_max: the maximum of the latitude range
- lon_min: the minimum of the longitude range
- lon_max: the maximum of the longitude range
- start_time: the beginning of the time range in the format YYYY-MM-DD HH:MM:SS
- end_time: the end of the time range in the format YYYY-MM-DD HH:MM:SS
- grid_id: WaveWatch III Production Hindcast Grid ID
- gecko_driver: path to your Firefox geckodriver.exe file
- firefox_binary: path to your Firefox binary (firefox.exe file)
- output_dir: path to the output directory relative to the `getNCEP_ww3.py` directory
- output_file_name: name of the output file excluding the file extension

The `getNCEP_ww3.py` program can obtain the following variables from the NCEP WW3 Production Hindcast models: significant wave height (`H_s`), peak wave period (`T_p`), primary wave direction (`D_p`), eastward wind velocity (`wind_u`), and northward wind velocity (`wind_v`). Original datetimes (datetime64 format) are converted to UNIX timestamps. Water depth is not included in any of the NCEP WW3 Production Hindcast datasets, which is problematic because water depth is necessary for later hydrodynamic calculations. As a solution, the `getGEBCO_bathy` function (see Section 2.2.2) is called within the `getNCEP_ww3.py` script to obtain GEBCO bathymetry data. Please refer to Section 2.2.2 to ensure the `getGEBCO_bathy.py` program is set up and installed on the user's system before running the `getNCEP_ww3.py` program. The program downloads the GEBCO bathymetry data, interpolates the bathymetry data to the NCEP WW3 grid, merges the depths with the NCEP WW3 data, and includes the variables together in the output netCDF and CSV files.

An example YAML settings input file for downloading NCEP WaveWatch III Production Hindcast data for a defined area is shown below.

Listing 2.3: `getGEBCO_bathy_settings.yml` Input File for the `getGEBCO_bathy.py` Algorithm

```
lat_min: 29.75              # minimum latitude
lat_max: 30.25              # maximum latitude
lon_min: 273.75              # minimum longitude
lon_max: 274.75              # maximum longitude
start_time: 2013-05-04 00:00:00     # YYYY-MM-DD HH:MM:SS
end_time: 2013-05-07 00:00:00       # YYYY-MM-DD HH:MM:SS
grid_id: "at_4m"           # WaveWatch III Production Hindcast Grid ID
gecko_driver: "C:/Program Files/Mozilla Firefox/geckodriver.exe"    # path to your
    Firefox geckodriver.exe file
firefox_binary: "C:/Program Files/Mozilla Firefox/firefox.exe"      # path to your
    Firefox binary (firefox.exe file)
```

```
output_dir: "./output"      # output directory; please write the filepath relative to
    the getNCEI_hycom.py directory
output_file_name: "TREX_04may2015-07may2015_NCEP_ww3"   # output file name; please do
    not include a file extension
```

A warning is displayed if the user-defined time range is outside the available NCEP WW3 data. Unlike the NCOM and HYCOM model data download programs, the NCEP WW3 program is unable to automatically calculate a grid ID due to the irregularity of the Production Hindcast grids. The user must manually determine which grid is necessary for the area of interest. See Section 2.1.3.1 or the "Model Grid Descriptions" at `https://polar.ncep.noaa.gov/waves/hindcasts/prod-multi_1.php`.

The output netCDF and CSV files contain the following variables:

- Time (time) [POSIX timestamp (Unix)]
- Latitude (lat) [degrees north]
- Longitude (lon) [degrees east]
- Significant wave height (Hs) [m]
- Peak wave period (Tp) [s]
- Eastward wind velocity (wind_u) [m/s]
- Northward wind velocity (wind_v) [m/s]
- Primary wave direction (Dp) [degrees]

The following section includes the `getNCEP_ww3.py` code and comments.

```
'''
Authors: Ryan Phillip
'''

import yaml
import pathlib
import argparse
import datetime
import urllib
from xarray import open_dataset, concat, DataArray
import os
import fnmatch
import numpy as np
import pandas as pd

from getGEBCO_bathy import getGEBCO_bathy
from getData_plotting_tools import plot_Hs, plot_Tp, plot_depth



def getNCEP_ww3(start_time, end_time, lat_min, lat_max, lon_min, lon_max,
                output_dir, grid_id, output_file_name, gecko_driver, firefox_binary):
    """
    Parameters
    ----------
    start_time : datetime
        data start date and time.
    end_time : datetime
        data end date and time.
    lat_min : float
```

```
        minimum latitude.
    lat_max : float
        maximum latitude.
    lon_min : float
        minimum longitude.
    lon_max : float
        maximum longitude.
    output_dir : string
        filepath to preprocessing output directory.
    grid_id : string
        ID of the desired WW3 Production Hindcast Multigrid dataset.
    output_file_name : string
        output file name; please do not include a file extension.
    gecko_driver : string
        path to Firefox geckodriver.exe file.
    firefox_binary : string
        path to Firefox binary (firefox.exe) file.

    Raises
    ------
    ValueError
        User-defined parameter(s) not in range of NCEP data.

    Returns
    -------
    downloads original grib2 WW3 Hindcast Multigrid file(s);
    downloads GEBCO bathymetry data;
    exports original GEBCO bathy NetCDF and a filtered and reformatted NetCDF file.
    """

    # record starting working directory
    og_dir = os.path.abspath(os.getcwd())

    # create a prefix for the URL to the NOAA WW3 Hindcast data repository
    url_prefix = 'https://polar.ncep.noaa.gov/waves/hindcasts/multi_1/'

    # record timestamps of user-defined start/end times
    start_ts = datetime.datetime.timestamp(start_time)
    end_ts = datetime.datetime.timestamp(end_time)

    ## see if user-defined times are in range
    ww3_range_start = 200502
    ww3_range_end = 201905

    start_time = int(f"{str(start_time).split('-')[0] + str(start_time).split('-')[1]}
    ")
    end_time = int(f"{str(end_time).split('-')[0] + str(end_time).split('-')[1]}")

    dates = []
    if start_time < ww3_range_start:
        raise ValueError('Start time out of range of NOAA WW3 Hindcast data; must be
    Feb 2005 or later')
    if end_time > ww3_range_end:
        raise ValueError('End time out of range of NOAA WW3 Hindcast data; must be May
     2019 or earlier')
    if start_time == end_time:
        dates.append(start_time)
```

```
    else:
        cur_time = start_time
        while cur_time != end_time:
            dates.append(cur_time)
            month = int(str(cur_time)[4:6])
            year = int(str(cur_time)[0:4])
            if month%12 == 0:
                cur_time += 89
            else:
                cur_time += 1

    # change to the user-defined output directory
    os.chdir(output_dir)

    print('\n')
    print('Downloading NCEP WW3 Data')
    # make a list of NCEP WW3 output variables
    data_kinds = ['wind', 'hs', 'tp', 'dp']
    ## retrieve WW3 Hindcast data from NCEP database
    filenames = []
    for date in dates:
        for data_kind in data_kinds:

            file_path = f'{date}/gribs/multi_1.{grid_id}.{data_kind}.{date}.grb2'
            url = url_prefix + file_path
            file = f"multi_1.{grid_id}.{data_kind}.{date}.grb2"
            filenames.append(file)
            print(f'Downloading "{file_path}"')

            urllib.request.urlretrieve(url, file)

    print('\n')
    print('Converting, merging, and concatenating NCEP WW3 grib2 files')
    ## merge same-month variables into single datasets
    # then append the merged datasets to a list for later concatenation
    subsets = []
    for date in dates:
        to_merge = []
        for file in os.listdir():
            if str(date) in file and file.endswith(".grb2"):
                file_data = open_dataset(file, engine='cfgrib')
                to_merge.append(file_data)
        merged = to_merge[0]
        for i in range(1, len(to_merge)):
            merged = merged.merge(to_merge[i])
        subsets.append(merged)

    print('\n')
    print('Removing unnecessary files')
    # remove the .idx files but keep the grib2 files
    for file in os.listdir():
        if fnmatch.fnmatch(file, 'multi_1*') and file.endswith(".idx"):
            os.remove(file)

    print('\n')
    print('Reformatting, filtering, and exporting NCEP WW3 data')
    # concatenate the merged datasets together
```

```
    ww3 = concat(subsets, dim = 'step')
    # drop unneeded variables
    ww3 = ww3.drop_vars(['time', 'surface'])
    # rename variables to match convention for SeaMoRE
    ww3 = ww3.rename({'latitude': 'lat', 'longitude': 'lon',
                      'swh': 'Hs', 'perpw': 'Tp', 'valid_time': 'time',
                      'dirpw': 'Dp', 'u': 'wind_u', 'v': 'wind_v'})

    # filter the dataset based on the user-defined area
    mask_lon = (ww3.lon >= lon_min) & (ww3.lon <= lon_max)
    mask_lat = (ww3.lat >= lat_min) & (ww3.lat <= lat_max)
    ww3 = ww3.where(mask_lon & mask_lat, drop = True)

    # convert the dataset to a dataframe to set 'time' as an index
    df = ww3.to_dataframe().drop_duplicates()
    df = df.set_index(['time'], append=True)
    df = df.reset_index().set_index(['time', 'lat', 'lon']).drop('step', axis = 1)
    df = df[~df.index.duplicated()]

    # convert the dataframe with time/lat/lon as indices to a dataset
    # so that time/lat/lon are the dataset dimensions
    ww3_xr = df.to_xarray()
    # convert time from datetime64 to UNIX timestamp
    ww3_xr['time'] = np.array([datetime.datetime.timestamp(pd.to_datetime(x)) for x in
     ww3_xr.time.values])
    # filter the dataset based on the user-defined time range
    # this is necessary because the WW3 Hindcast data is in month-long chunks
    # resulting in some dates/times being out-of-range
    mask_time = (ww3_xr.time >= start_ts) & (ww3_xr.time <= end_ts)
    ww3_xr = ww3_xr.where(mask_time, drop = True)

    # convert Dp from "coming-from" to "going-to"
    tdp = np.array([x+180-360 if x+180>=360 else x+180 for x in ww3_xr.Dp.values.
    flatten()]).reshape(np.shape(ww3_xr.Dp.values))
    tdp_xa = DataArray(tdp, coords={'time':ww3_xr.time.values, 'lat':ww3_xr.lat.values
    ,
                                    'lon':ww3_xr.lon.values},
                       dims=['time', 'lat', 'lon'])
    ww3_xr['Dp'] = tdp_xa

    # change back to the original working directory
    os.chdir(og_dir)

    print('\n')
    print('Beginning download of GEBCO bathymetry data')
    # download GEBCO bathymetry data for the user-defined area
    bathy_ds = getGEBCO_bathy(gecko_driver, firefox_binary, lat_min, lat_max,
                              lon_min, lon_max, output_dir)

    print('\n')
    print('GEBCO bathymetry data successfully downloaded')
    print('Interpolating and merging bathymetry data with WW3 dataset')
    # interpolate the bathy data to the NCEP WW3 grid
    bathy_interp = bathy_ds.interp_like(ww3_xr)

    # expand depth to include a time dimension
    bathy_interp = bathy_interp.expand_dims(time=ww3_xr.time)
```

```
196
197     # merge the interpolated bathy data with the WW3 dataset
198     ww3_xr_merged = bathy_interp.merge(ww3_xr)
199
200     # mask the depths to where WW3 data exists
201     ww3_xr_merged['depth'] = ww3_xr_merged.depth.where(np.isnan(ww3_xr_merged.Hs.
        values) == False)
202
203     # change to the user-defined output directory
204     os.chdir(output_dir)
205
206     # write and output the dataset to a NetCDF and a CSV
207     ww3_xr_merged.to_netcdf(f"{output_file_name}.nc")
208     ww3_df = ww3_xr_merged.to_dataframe()
209     ww3_df.to_csv(f"{output_file_name}.csv")
210
211     print('\n')
212     print('NCEP WW3 pre-processing NetCDF file written and exported successfully')
213
214     # close the datasets
215     ww3_xr.close()
216     ww3_xr_merged.close()
217
218     print('\n')
219     print('Datasets closed successfully')
220
221     # change back to the original working directory
222     os.chdir(og_dir)
223
224     print('\n')
225     print('Task Complete')
226
227
228
229
230 if __name__ == "__main__":
231     parser = argparse.ArgumentParser()
232
233
234     parser.add_argument('-y', '--yaml-file',
235                         help = 'YAML file containing input settings',
236                         metavar = 'FILE',
237                         default = 'getNCEP_ww3_settings.yml',
238                         type = pathlib.Path)
239
240
241     args = parser.parse_args()
242
243
244     # parse the input YAML file
245     with open(args.yaml_file, 'r') as fp:
246     # with open("getNCEP_ww3_settings_test.yml", 'r') as fp:
247         settings = yaml.safe_load(fp)
248
249         # Set all the variables from the yaml file
250         lat_min = settings['lat_min']
251         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']
        grid_id = settings['grid_id']
        gecko_driver = settings['gecko_driver']
        firefox_binary = settings['firefox_binary']
        output_dir = settings['output_dir']
        output_file_name = settings['output_file_name']


    # run the getNCEP_ww3 function from getData_tools
    getNCEP_ww3(start_time, end_time, lat_min-0.25, lat_max+0.25, lon_min-0.25,
    lon_max+0.25,
                output_dir, grid_id, output_file_name, gecko_driver, firefox_binary)


    # read the formatted NCEP WW3 dataset in for plotting
    ds = open_dataset(f"{output_dir}/{output_file_name}.nc")
    df = ds.to_dataframe().reset_index()

    # plot significant wave height (Hs)
    plot_Hs(df, output_dir, output_file_name, plot_time_info=False)

    # plot peak wave period (Tp)
    plot_Tp(df, output_dir, output_file_name, plot_time_info=False)

    # plot water depth
    plot_depth(df, output_dir, output_file_name, plot_time_info=False)

    # close the dataset
    ds.close()
```

2.2.4 Downloading NDBC data

The `getNDBC_data.py` program described here is a slightly modified version of the `pull_NDBC_data.py` program described in the memorandum report Penko et al. (2022) [8]. It is included in this technical report to document the modifications as part of the getData toolkit.

The `getNDBC_data.py` program downloads observation data from the NDBC database and saves it in a netCDF file. The present program downloads significant wave height from the standard meteorological data (stdmet). The station and date are determined by user-defined variables read in from the `getNDBC_settings.yml` YAML file. The netCDF file will be saved in the directory the user specifies in the command line. The program runs using the following command:

```
>> python getNDBC_data.py -y getNDBC_settings.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.

Listing 2.4: `getNDBC_settings.yml` Input File for the `getNDBC_data.py` Algorithm

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

After execution, the program returns the first and last times available for the year containing your requested date. 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 and 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 "getNDBC\_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, check the size of the netCDF's time dimension.

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

```
'''
Authors: Kendal Hall and Allison Penko
'''


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


```

```
# ---------------------------------- 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"
```

```
72
73 # Set abbreviation.
74 DB = DataBuoy()
75
76 # Specify station ID.
77 DB.set_station_id(station_id) # <- Either strings or numbers are acceptable.
78
79 # Get station metadata and data.
80 DB.get_station_metadata()
81 DB.get_data(years=year_range, datetime_index=True, data_type='stdmet')
82
83 # Access dataframe in dict within a dict.
84 df = DB.data.get('stdmet').get('data')
85
86 # Make new dataframe, keeping only rows with WVHT values.
87 wvht = df[~df['WVHT'].isna()]
88
89 # Print the first and last times available in the file.
90 print("The first time available is: ", wvht.index[0].to_pydatetime())
91 print("The last time available is: ", wvht.index[-1].to_pydatetime())
92
93 # Access dictionary with units.
94 units_dict = DB.data.get('stdmet').get('meta').get('units')
95
96 # NOTE:-----------------------------------------------------------------------
97 # When I tried using the months argument for DB.get_data, it would do two
98 # things wrong:
99 # It would say every month later than May was not available.
100 # It would start at Jan regardless of what month I requested to start.
101 # The solution I found was to request the whole year.
102 # ----------------------------------------------------------------------------
103
104 # Put a start date and end date and cull everything not in date range.
105 wvht.index = wvht.index.to_pydatetime()
106 mask = (wvht.index > start_date) & (wvht.index <= end_date)
107 wvht = wvht.loc[mask]
108
109 # Convert the pandas dataframe to an xarray dataset.
110 xr_ds = wvht.to_xarray()
111
112 # Write the xarray dataset to a netCDF file.
113 xr_ds.to_netcdf(path=fn, mode='w', format=None, group=None,
114                 engine=None, encoding=None, unlimited_dims=None, compute=True,\
115                     invalid_netcdf=False)
116
117 # Get keys in the units_dict.
118 keys_list = list(units_dict.keys())
119
120 # Find the vars in the nc file (or dataset) that are in common with the entries
121 # in keysList.
122 ds = nc.Dataset(fn,mode='a')
123
124 # Get the names of the variables in the netCDF file.
125 var_names = ds.variables.keys()
126
127 # Go through each variable name.
128 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()
```

2.2.5 Downloading NOAA CO-OPS Data

The `getNOAA_coops.py` program described here is a slightly modified version of the `pull_NOAA_CO-OPS_Tides_and_Currents.py` program described in the memorandum report Penko et al. (2022) [8]. It is included in this technical report to document the modifications as part of the getData toolkit.

The `getNOAA_coops.py` program downloads observed water level or wind data from the NOAA Tides and Currents database and saves it in a netCDF file. The data downloaded is determined by variables read in from the `getNOAA_coops_settings.yml` file. The program runs using the following command:

```
>> python getNOAA_coops.py -y getNOAA_coops_settings.yml -o . -i .
```

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

- station_id: Station ID of requested station
- data_type: Product type ("water_level' or "wind")
- datum: The data reference point (MSL, MLLW, MLW)
- units_convention ("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 `getNOAA_coops_settings.yml` input file is shown below with common CO-OPS stations along Florida's Panhandle and west coast.

Listing 2.5: `getNOAA_coops_settings.yml` Input File for the `getNOAA_coops.py` Algorithm

```
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'
```

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 "getNOAA_coops.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 `getNOAA_coops.py` code and comments.

```
'''
Authors: Kendal Hall and Allison Penko
'''

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 coastal models allow for skillful forecasts of dynamic shallow-water environments. Information from low-resolution models is typically used to generate boundary conditions for models with increasing resolution at smaller scales. Automated and efficient algorithms to obtain and reformat meteorologic and oceanographic data from operational model output are necessary to both drive and evaluate these models. Six different algorithms were created to download publicly available data from web servers containing both model and observational data. Parameters to retrieve NCEI, NCEP, GEBCO, NDBC, and NOAA CO-OPS datasets are defined through a YAML input file, allowing for ease of use for numerical modelers. Swift methods to access METOC data for model forcing and evaluation ultimately reduce the amount of time developers need for model development and validation.

ACKNOWLEDGMENTS

REP and AMP were funded by the U.S. Naval Research Laboratory Base Program. This technical report is an expansion and update of the memorandum report titled ”Automated Method to Extract Oceanographic and Atmospheric Data from Online Sources” [8]. The work presented in this report makes use of online data sources provided by NOAA’s National Centers for Environmental Information, NOAA’s National Centers for Environmental Prediction, the General Bathymetric Chart of the Oceans, NOAA’s National Data Buoy Center, and NOAA’s Center for Operational Oceanographic Products and Services.

REFERENCES

1. K. R. Wood and J. Rayburn, “FNMOC Navy Global Hybrid Coordinate Ocean Model,” 2023. URL https://www.ncei.noaa.gov/products/weather-climate-models/frnmoc-navy-global-hybrid-ocean.

2. K. R. Wood and J. Rayburn, "FNMOC Regional Navy Coastal Ocean Model," 2023. URL https://www.ncei.noaa.gov/products/weather-climate-models/fnmoc-regional-navy-coastal-ocean.

3. R. A. Simons and J. Chris, "ERDDAP," 2022. URL https://coastwatch.pfeg.noaa.gov/erddap.

4. GEBCO Compilation Group, "GEBCO 2023 Grid," 2023. URL https://www.gebco.net/.

5. National Centers for Environmental Prediction, "WAVEWATCHIII Production Hindcast, Multigrid: Feb 2005 to May 2019," 2019. URL https://polar.ncep.noaa.gov/waves/hindcasts/prod-multi_1.php.

6. National Data Buoy Center, "NDBC," 2023. URL https://www.ndbc.noaa.gov/.

7. Center for Operational Oceanographic Products and Services, "Tides and Currents," 2023. URL https://tidesandcurrents.noaa.gov/.

8. A. M. Penko, S. S. Schoenauer, and K. Hall, "Automated Method to Extract Oceanographic and Atmospheric Data from Online Sources," 2022. URL https://apps.dtic.mil/sti/citations/AD1184125.