Read the Docs Template Documentation

Release 0.1

Read the Docs

Nov 18, 2021

GETTING STARTED

1 Installation **3**
- 1.1 Conda . . . 3
- 1.2 PyPI . . . 3
- 1.3 GitHub . . . 3

2 Dependencies **5**

3 Examples **7**
- 3.1 CMIP6 . . . 7

4 API Reference **13**
- 4.1 Tracking . . . 13

5 Citing Ocetrac **15**
- 5.1 Project Contributors . . . 15

6 Contribution Guide **17**
- 6.1 Feature requests and feedback . . . 17
- 6.2 Report bugs . . . 17
- 6.3 Fix bugs . . . 18
- 6.4 Preparing Pull Requests . . . 18

7 Wishlist **21**

Index **23**

Ocetrac is a Python 3.6+ package designed to label and track the evolution of unique geospatial features in gridded datasets. The package is designed to accept data that have already been preprocessed, meaning that the data only contain values the user is interested in tracking. Ocetrac operates lazily with Dask so that it is memory uninhibited and fast through parallelized execution. We provide examples and demonstrate best practices as developed by the Climate Data Science Lab at Columbia University. Here you will find instructions on how to install ocetrac, use it's API, and contribute to future releases.

For recommendations or bug reports, please visit: https://github.com/ocetrac/ocetrac/issues/new

CHAPTER

ONE

INSTALLATION

Notes on how to install ocetrac.

1.1 Conda

The easiest way to install the packge is with conda: `conda install -c conda-forge ocetrac`.

1.2 PyPI

You can also install with pip: `pip install ocetrac`.

1.3 GitHub

For the most up to date version of ocetrac, you can install directly from the online repository hosted on GitLab.

1. Clone ocetrac to your local machine: `git clone https://github.com/ocetrac/ocetrac`
2. Change to the parent directory of ocetrac
3. Install ocetrac with `pip install -e ./ocetrac`. This will allow changes you make locally, to be reflected when you import the package in Python

CHAPTER

TWO

DEPENDENCIES

The only requirement is Python >=3.6. The following dependencies will also be installed:

- xarray
- dask
- scipy
- scikit-image

CHAPTER

THREE

EXAMPLES

3.1 CMIP6

Here is a quick example of how to use `Tracker.track` to detect and track **marine heatwaves** from ocean temperature data available from CMIP6.

First import numpy, xarray, matplotlib, intake, cmip6_preprocessing, ocetrac, and dask:

[1]:
```
import numpy as np
import xarray as xr
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
import intake
from cmip6_preprocessing.preprocessing import combined_preprocessing
import ocetrac
from dask_gateway import Gateway
from dask.distributed import Client
from dask import delayed
import dask
dask.config.set({"array.slicing.split_large_chunks": False});
```

Start a dask kubernetes cluster:

[2]:
```
gateway = Gateway()
cluster = gateway.new_cluster()
cluster.adapt(minimum=1, maximum=20)
client = Client(cluster)
cluster
```

```
VBox(children=(HTML(value='<h2>GatewayCluster</h2>'), HBox(children=(HTML(value='\n<div>\
↪n<style scoped>\n    ...
```

3.1.1 NOAA-GFDL CM4

An Earth System Model collection for CMIP6 Zarr data resides on Pangeo's Google Storage. We will create a query to import data from the historical simulation of monthly ocean surface temperatures provided by the NOAA-Geophysical Fluid Dynamics Global Climate Model.

[3]:
```
col = intake.open_esm_datastore("https://storage.googleapis.com/cmip6/pangeo-cmip6.json")

# create a query dictionary
query = col.search(experiment_id=['historical'], # all forcing of the recent past
                   table_id='Omon', # ocean monthly data
                   source_id='GFDL-CM4', # GFDL Climate Model 4
                   variable_id=['tos'], # temperature ocean surface
                   grid_label='gr', # regridded data on target grid
                  )
query
```

```
<IPython.core.display.HTML object>
```

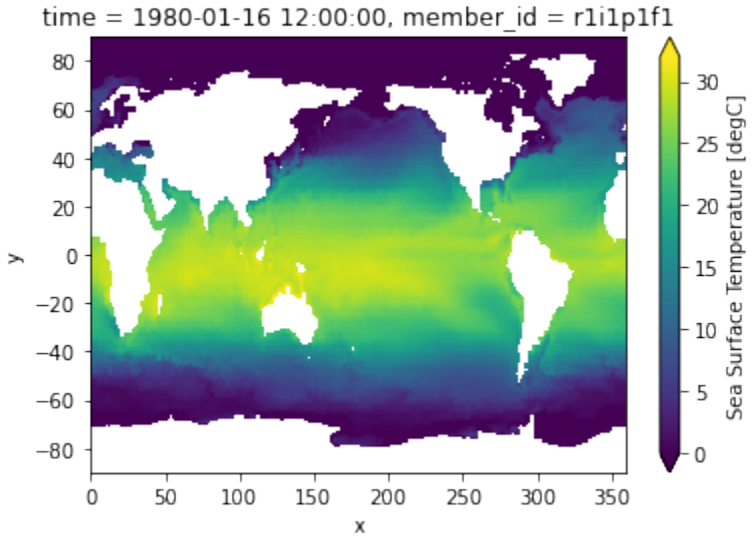
[4]:
```
ds = query.to_dataset_dict(zarr_kwargs={'consolidated': True},
                           storage_options={'token': 'anon'},
                           preprocess=combined_preprocessing,
                          )

tos = ds['CMIP.NOAA-GFDL.GFDL-CM4.historical.Omon.gr'].tos.isel(member_id=0).
↪sel(time=slice('1980', '2014'))
tos
```

```
--> The keys in the returned dictionary of datasets are constructed as follows:
        'activity_id.institution_id.source_id.experiment_id.table_id.grid_label'
```

```
<IPython.core.display.HTML object>
```

[4]:
```
<xarray.DataArray 'tos' (time: 420, y: 180, x: 360)>
dask.array<getitem, shape=(420, 180, 360), dtype=float32, chunksize=(120, 180, 360),␣
↪chunktype=numpy.ndarray>
Coordinates:
  * y          (y) float64 -89.5 -88.5 -87.5 -86.5 -85.5 ... 86.5 87.5 88.5 89.5
  * x          (x) float64 0.5 1.5 2.5 3.5 4.5 ... 355.5 356.5 357.5 358.5 359.5
  * time       (time) object 1980-01-16 12:00:00 ... 2014-12-16 12:00:00
    lon        (x, y) float64 0.5 0.5 0.5 0.5 0.5 ... 359.5 359.5 359.5 359.5
    lat        (x, y) float64 -89.5 -88.5 -87.5 -86.5 ... 86.5 87.5 88.5 89.5
    member_id  <U8 'r1i1p1f1'
Attributes:
    cell_measures:  area: areacello
    cell_methods:   area: mean where sea time: mean
    comment:        Model data on the 1x1 grid includes values in all cells f...
    interp_method:  conserve_order1
    long_name:      Sea Surface Temperature
    original_name:  tos
    standard_name:  sea_surface_temperature
    units:          degC
```

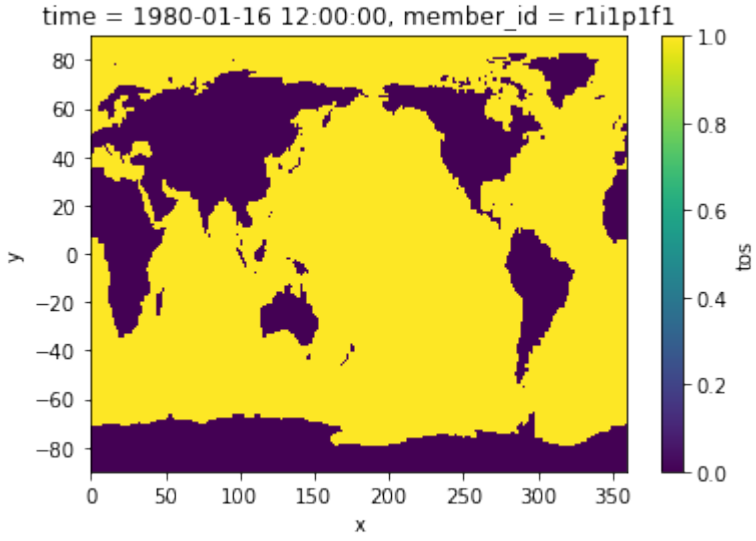
[5]:
```
tos.isel(time=0).plot(vmin=0, vmax=32);
```



Create a binary mask for ocean points:

[6]:
```
mask_ocean = 1 * np.ones(tos.shape[1:]) * np.isfinite(tos.isel(time=0))
mask_land = 0 * np.ones(tos.shape[1:]) * np.isnan(tos.isel(time=0))
mask = mask_ocean + mask_land
mask.plot()
```

[6]:
```
<matplotlib.collections.QuadMesh at 0x7f7cd06f0e80>
```



3.1.2 Preprocess the Data

We will simply define monthly anomalies by subtracting the monthly climatology averaged across 1980–2014. Anomalies that exceed the monthly 90th percentile will be considered here as extreme.

[7]:
```
climatology = tos.groupby(tos.time.dt.month).mean()
anomaly = tos.groupby(tos.time.dt.month) - climatology

# Rechunk time dim
if tos.chunks:
    tos = tos.chunk({'time': -1})

percentile = .9
threshold = tos.groupby(tos.time.dt.month).quantile(percentile, dim='time', keep_
↪attrs=True, skipna=True)
hot_water = anomaly.groupby(tos.time.dt.month).where(tos.groupby(tos.time.dt.month)>
↪threshold)
```

```
/srv/conda/envs/notebook/lib/python3.8/site-packages/xarray/core/indexing.py:1369:␣
↪PerformanceWarning: Slicing with an out-of-order index is generating 35 times more␣
↪chunks
  return self.array[key]
```

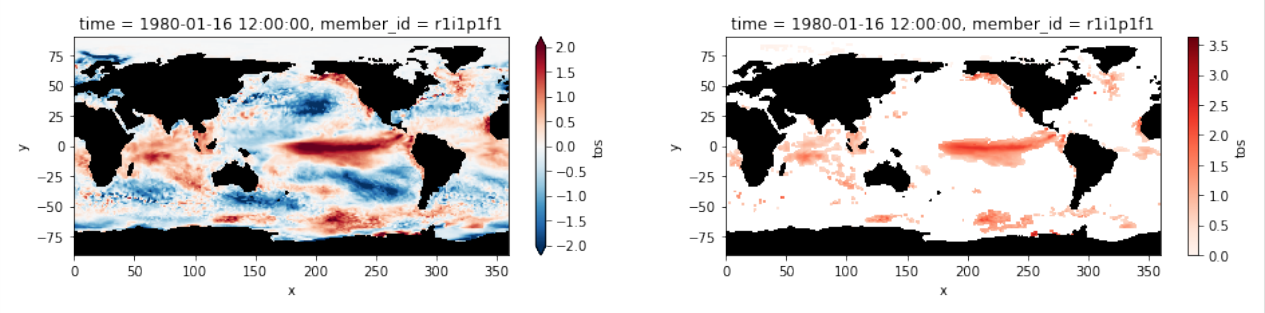
[8]:
```
%%time
hot_water.load()
client.close()
```

```
CPU times: user 661 ms, sys: 136 ms, total: 797 ms
Wall time: 1min 46s
```

The plots below shows sea surface temperature anomalies averaged for the month of January 1980. The right panel shows only the **extreme** anomalies exceeding the 90th percentile.

[9]:
```
plt.figure(figsize=(16,3))

ax1 = plt.subplot(121);anomaly.isel(time=0).plot(cmap='RdBu_r', vmin=-2, vmax=2, extend=
↪'both')
mask.where(mask==0).plot.contourf(colors='k', add_colorbar=False); ax1.set_aspect('equal
↪');

ax2 = plt.subplot(122); hot_water.isel(time=0).plot(cmap='Reds', vmin=0);
mask.where(mask==0).plot.contourf(colors='k', add_colorbar=False); ax2.set_aspect('equal
↪');
```

```
/srv/conda/envs/notebook/lib/python3.8/site-packages/dask/array/numpy_compat.py:40:␣
↪RuntimeWarning: invalid value encountered in true_divide
  x = np.divide(x1, x2, out)
```



3.1.3 Run Ocetrac

Using the extreme SST anomalies only, we use Ocetrac to label and track marine heatwave events.

We need to define two key parameters that can be tuned bases on the resolution of the dataset and distribution of data. The first is `radius` which represents the number of grid cells defining the width of the structuring element used in the morphological operations (i.e., opening and closing). The second is `min_size_quartile` that is used as a threshold to subsample the objects accroding the the distribution of object area.

[10]:
```
%%time
Tracker = ocetrac.Tracker(hot_water, mask, radius=2, min_size_quartile=0.75, timedim =
↪'time', xdim = 'x', ydim='y', positive=True)
blobs = Tracker.track()
```

```
minimum area: 107.0
inital objects identified       15682
final objects tracked    903
CPU times: user 37.1 s, sys: 1.44 s, total: 38.5 s
Wall time: 38.6 s
```

Let's take a look at some of the attributes.

There were over 15,500 MHW object detected. After connecting these events in time and eliminating objects smaller than the 75th percentile (equivalent to the area of 107 grid cells), 903 total MHWs are identified between 1980–2014.

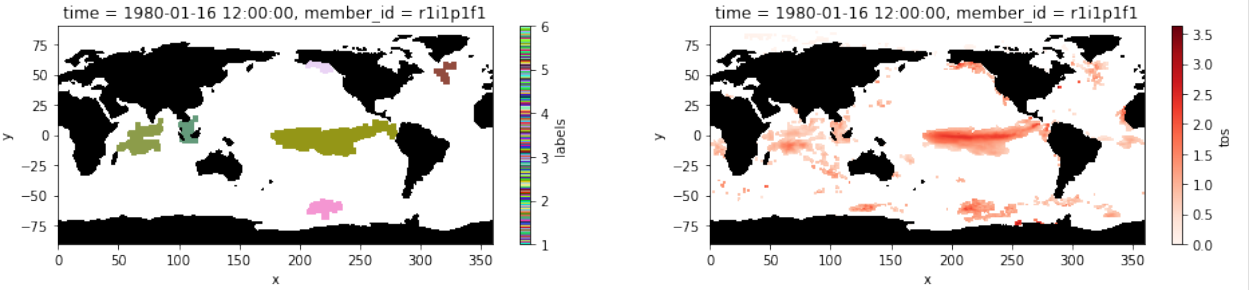
[11]:
```
blobs.attrs
```

[11]:
```
{'inital objects identified': 15682,
 'final objects tracked': 903,
 'radius': 2,
 'size quantile threshold': 0.75,
 'min area': 107.0,
 'percent area reject': 0.17453639541742397,
 'percent area accept': 0.8254636045825761}
```

3.1.4 Visualize Output

Plot the labeled marine heatwaves on January 1980 and compare it to the input image of the preprocessed extreme sea surface temperature anomalies.

[12]:
```
maxl = int(np.nanmax(blobs.values))
cm = ListedColormap(np.random.random(size=(maxl, 3)).tolist())

plt.figure(figsize=(16,3))
ax1 = plt.subplot(121);blobs.isel(time=0).plot(cmap= cm)
mask.where(mask==0).plot.contourf(colors='k', add_colorbar=False); ax1.set_aspect('equal
↪')
ax2 = plt.subplot(122); hot_water.isel(time=0).plot(cmap='Reds', vmin=0);
mask.where(mask==0).plot.contourf(colors='k', add_colorbar=False); ax2.set_aspect('equal
↪');
```



CHAPTER

FOUR

API REFERENCE

The API reference is automatically generated from the function docstrings in the octrac package. Refer to the examples in the sidebar for reference on how to use the functions.

4.1 Tracking

Tracker.track()	Label and track image features.

4.1.1 ocetrac.Tracker.track

Tracker.**track**()

Label and track image features.

Parameters

- **da** (*xarray.DataArray*) – The data to label.
- **mask** (*xarray.DataArray*) – The mask of ponts to ignore. Must be binary where 1 = true point and 0 = background to be ignored.
- **radius** (*int*) – The size of the structuring element used in morphological opening and closing. Radius specified by the number of grid units.
- **min_size_quartile** (*float*) – The quantile used to define the threshold of the smallest area object retained in tracking. Value should be between 0 and 1.
- **timedim** (*str*) – The name of the time dimension
- **xdim** (*str*) – The name of the x dimension
- **ydim** (*str*) – The namne of the y dimension
- **positive** (*bool*) – True if da values are expected to be positive, false if they are negative. Default argument is True

Returns **labels** – Integer labels of the connected regions.

Return type xarray.DataArray

CHAPTER

FIVE

CITING OCETRAC

5.1 Project Contributors

The following people have made contributions to the project (in alphabetical order by last name) and are considered "Ocetrac Developers". These contributors will be added as authors upon the next major release of ocetrac (i.e. new DOI release).

- Ryan Abernathey - Columbia University, USA. (ORCID: 0000-0001-5999-4917)
- Julius Busecke - Columbia University, USA. (ORCID: 0000-0001-8571-865X)
- David John Gagne - National Center for Atmospheric Research, USA. (ORCID: 0000-0002-0469-2740)
- Hillary Scannell - Columbia University, USA. (ORCID: 0000-0002-6604-1695)
- LuAnne Thompson - University of Washington, USA. (ORCID: 0000-0001-8295-0533)
- Daniel Whitt - National Aeronautics and Space Administration, Ames Research Center, USA.

CHAPTER

SIX

CONTRIBUTION GUIDE

Contributions are highly welcomed and appreciated. Every little help counts, so do not hesitate! You can make a high impact on `ocetrac` just by using it, being involved in discussions and reporting issues.

The following sections cover some general guidelines regarding development in `ocetrac` for maintainers and contributors.

Nothing here is set in stone and can't be changed. Feel free to suggest improvements or changes in the workflow.

Contribution links

- *Contribution Guide*
  - *Feature requests and feedback*
  - *Report bugs*
  - *Fix bugs*
  - *Preparing Pull Requests*

6.1 Feature requests and feedback

We are eager to hear about your requests for new features and any suggestions about the API, infrastructure, and so on. Feel free to start a discussion about these on the discussions tab on github under the "ideas" section.

After discussion with a few community members, and agreement that the feature should be added and who will work on it, a new issue should be opened. In the issue, please make sure to explain in detail how the feature should work and keep the scope as narrow as possible. This will make it easier to implement in small PRs.

6.2 Report bugs

Report bugs for `ocetrac` in the issue tracker with the label "bug".

If you can write a demonstration test that currently fails but should pass that is a very useful commit to make as well, even if you cannot fix the bug itself.

6.3 Fix bugs

Look through the GitHub issues for bugs.

Talk to developers to find out how you can fix specific bugs.

6.4 Preparing Pull Requests

1. Fork the ocetrac GitHub repository. It's fine to use `ocetrac` as your fork repository name because it will live under your username.
2. Clone your fork locally using git, connect your repository to the upstream (main project), and create a branch:

```
$ git clone git@github.com:YOUR_GITHUB_USERNAME/ocetrac.git # clone to local machine
$ cd ocetrac
$ git remote add upstream git@github.com:ocetrac/ocetrac.git # connect to upstream␣
↪remote

# now, to fix a bug or add feature create your own branch off "main":

$ git checkout -b your-bugfix-feature-branch-name main # Create a new branch where␣
↪you will make changes
```

   If you need some help with Git, follow this quick start guide: https://git.wiki.kernel.org/index.php/QuickStart

3. Set up a [conda](environment) with all necessary dependencies:

```
$ conda env create -f ci/environment-py3.8.yml
```

4. Activate your environment:

```
$ conda activate test_env_ocetrac
```

   Make sure you are in this environment when working on changes in the future too.

5. Install the Ocetrac package:

```
$ pip install -e . --no-deps
```

6. Before you modify anything, ensure that the setup works by executing all tests:

```
$ pytest
```

   You want to see an output indicating no failures, like this:

```
$ ========================== n passed, j warnings in 17.07s␣
↪============================
```

7. Install pre-commit and its hook on the `ocetrac` repo:

```
$ pip install --user pre-commit
$ pre-commit install
```

   Afterwards `pre-commit` will run whenever you commit. If some errors are reported by pre-commit you should format the code by running:

```
$ pre-commit run --all-files
```

and then try to commit again.

https://pre-commit.com/ is a framework for managing and maintaining multi-language pre-commit hooks to ensure code-style and code formatting is consistent.

> You can now edit your local working copy and run/add tests as necessary. Please follow PEP-8 for naming. When committing, `pre-commit` will modify the files as needed, or will generally be quite clear about what you need to do to pass the commit test.

8. Break your edits up into reasonably sized commits:

```
$ git commit -a -m "<commit message>"
$ git push -u
```

Committing will run the pre-commit hooks (isort, black and flake8). Pushing will run the pre-push hooks (pytest and coverage)

We highly recommend using test driven development, but our coverage requirement is low at the moment due to lack of tests. If you are able to write tests, please stick to xarray's testing recommendations.

9. Add yourself to the Project Contributors list via `./docs/authors.md`.
10. Finally, submit a pull request (PR) through the GitHub website using this data:

```
head-fork: YOUR_GITHUB_USERNAME/ocetrac
compare: your-branch-name

base-fork: ocetrac/ocetrac
base: main
```

The merged pull request will undergo the same testing that your local branch had to pass when pushing.

11. After your pull request is merged into the *ocetrac/main*, you will need to fetch those changes and rebase your main so that your main reflects the latest version of ocetrac. The changes should be fetched and incorporated (rebase) also right before you are planning to introduce changes.:

```
$ git checkout main # switch back to main branch
$ git fetch upstream  # Download all changes from central upstream repo
$ git rebase upstream/main  # Apply the changes that have been made to central repo,
$ # since your last fetch, onto your main.
$ git branch -d your-bugfix-feature-branch-name  # to delete the branch after PR is␣
↪approved
```

CHAPTER

SEVEN

WISHLIST

By adopting open-source best practices, we hope ocetrac will grow into a widely used, community-based project. We anticipate ocetrac will have broad applications in geoscience and are excited to see it used in other domains besides oceanography.

INDEX

T

track() (*ocetrac.Tracker method*), 13