ScientiMate

Release 1.0

Nov 03, 2020

Contents

ScientiMate is a library for earth-science data analysis. This library can be used for wide range of data analysis including a time series analysis, signal processing, and geo-data calculation.

Name ScientiMate

Description Earth-Science Data Analysis Library

Version 1.0

Requirements Python (3 or later), NumPy, SciPy, Matplotlib

Developer Arash Karimpour (http://www.arashkarimpour.com)

Documentation https://scientimate.readthedocs.io

Tutorial Video YouTube Playlist

Source Code https://github.com/akarimp/scientimate

Report Issues https://github.com/akarimp/scientimate/issues

CHAPTER 1

User Guide

Setup and Usage

1.1 Introduction

ScientiMate is a library for earth-science data analysis. This library can be used for wide range of data analysis including a time series analysis, signal processing, and geo-data calculation.

1.1.1 ScientiMate Modules

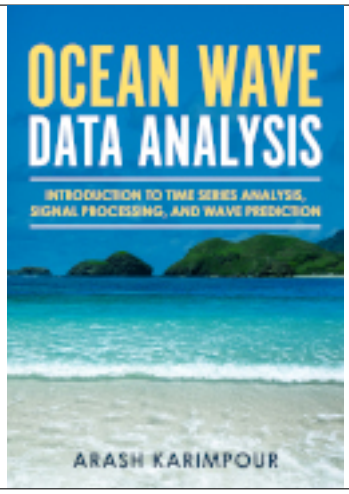
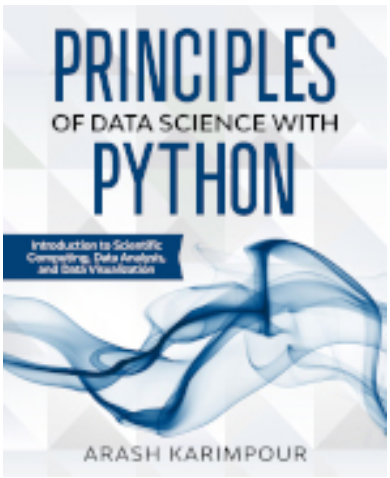
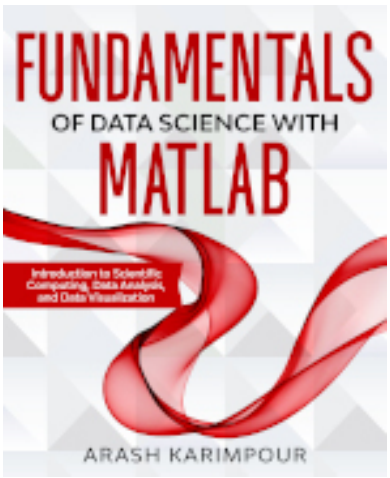
- Data Manipulating
- Hurricane
- Mapping
- OCEANLYZ
- Plotting
- Signal Processing
- Statistics
- Swan
- Water Wave Data Analysis
- Water Wave Directional Analysis
- Water Wave Parametric Model
- Water Wave Properties
- Wind

1.1.2 Citation

Cite this package as:

Karimpour, A. (2020). ScientiMate, Earth-Science Data Analysis Library.

1.1.3 Recommended Books

		
Ocean Wave Data Analysis Introduction to Time Series Analysis, Signal Processing, and Wave Prediction. Order at Amazon: https://www.amazon.com/dp/0692109978	**Principles of Data Science with Python** Introduction to Scientific Computing, Data Analysis, and Data Visualization. Order at Amazon: https://www.amazon.com/dp/1735241008	**Fundamentals of Data Science with MATLAB** Introduction to Scientific Computing, Data Analysis, and Data Visualization. Order at Amazon: https://www.amazon.com/dp/1735241016

1.1.4 License Agreement and Disclaimer

ScientiMate: Earth-Science Data Analysis Library

OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

1.2 Getting Started

In order to use ScientiMate package, first, Python programming language, and then, the ScientiMate package should be installed.

1.2.1 Installation

To use ScientiMate package:

- Install Python
- Install ScientiMate

1) Install Python

First, we need to install Python programming language.

- **Method 1:** Install pure Python from https://www.python.org and then use the **pip** command to install required packages
- **Method 2 (Recommended):** Install Anaconda Python distribution from https://www.anaconda.com and then use the **conda** command to install required packages

2) Install ScientiMate

After Python is installed, we need to install ScientiMate package.

To install ScientiMate via pip (https://pypi.org/project/scientimate):

```
pip install scientimate
```

To install ScientiMate via Anaconda cloud (https://anaconda.org/akarimp/scientimate):

```
conda install -c akarimp scientimate
```

1.2.2 Operating System

This code can be run on Windows, Mac, and Linux.

1.2.3 Required Programing Language

This package can be run by using Python 3 or later (https://www.python.org or https://www.anaconda.com).

1.2.4 Required Package for Python

Following packages are required:

- NumPy (https://numpy.org)
- SciPy (https://www.scipy.org)
- Matplotlib (https://matplotlib.org)

1.2.5 Quick Start

```
import scientimate as sm
import numpy as np

print(sm.__version__)

x=np.linspace(1,10,10)
y=np.zeros((10,2))
y[:,0]=1+np.random.rand(10)
y[:,1]=2+np.random.rand(10)
sm.plot2d(x,y,'line_confid','blue_red','large')
```

Functions Reference

1.3 API

Here is a list of the ScientiMate functions:

1.3.1 Data Manipulating

scientimate.downsamplex

```
x_ds = scientimate.downsamplex(x, RetainRatio)
```

Description

Downsample x data and retain given ratio

Inputs

x x data

RetainRatio=0.5

Define percentage of data to retain, value between 0 and 1
Example: RetainRatio=0.8 means 80% of data are retained

Outputs

x_ds Downsample x data

Examples

```
import scientimate as sm
import numpy as np
from numpy import random

rng = np.random.default_rng()
x=10*rng.random((1000,1))
x_ds=sm.downsamplex(x, 0.7)

rng = np.random.default_rng()
xgrid=(-90-(-91))*rng.random((1000,500))+(-91)
x_ds=sm.downsamplex(xgrid, 0.3)
```

References

scientimate.interpxyz2grid

```
xgrid, ygrid, zgrid = scientimate.interpxyz2grid(x, y, z, gridsize=100, gridsizetype=
↪'points', xmin=None, xmax=None, ymin=None, ymax=None, zmin=None, zmax=None,␣
↪RetainRatio='all', interpMethod='nearest', dispout='no')
```

Description

Interpolate x (longitude), y (latitude) and z (elevation) data into a defined mesh

Inputs

x x (longitude) data extracted from xyz file

y y (latitude) data extracted from xyz file

z z (elevation) data extracted from xyz file

gridsize=100

Grid size in x (longitude) and y (latitude) directions to interpolate elevation data on them
if gridsizetype='length' then gridsize is a distance between grid points
if gridsizetype='points' then gridsize is number of grid points in each direction

gridsizetype='points'

Grid size type
'points': gridsize is considered as number of grid points in each direction
'length': gridsize is considered as length between grid points

xmin=nanmin(x) Minimum x (longitude) of domain to be interpolated

xmax=nanmax(x) Maximum x (longitude) of domain to be interpolated

ymin=nanmin(y) Minimum y (latitude) of domain to be interpolated

ymax=nanmax(y) Maximum y (latitude) of domain to be interpolated

zmin=nanmin(z)

Minimum z (elevation) of domain to be interpolated

All z<zmin would be set to zmin

zmax=nanmax(z)

Maximum z (elevation) of domain to be interpolated
All z>zmax would be set to zmax

RetainRatio='all'

Define to down sample input data or not
'all': data are not down sampled
value between 0 and 1: percentage of retaining data
RetainRatio=0.8 : 80# of data are retained

interpMethod='nearest'

Interpolation method
'linear': Use default or 'linear' method to interpolate
'nearest': Use nearest neighbor method to interpolate

dispout='no' Define to display outputs or not ('yes': display, 'no': not display)

Outputs

xgrid Interpolated x (longitude) data data on defined mesh

ygrid Interpolated y (latitude) data on defined mesh

zgrid Interpolated z (elevation) data on defined mesh

Examples

```
import scientimate as sm
import numpy as np

x=10.*np.random.rand(1000)
y=10.*np.random.rand(1000)
z=x**2+y**2
xgrid,ygrid,zgrid=sm.interpxyz2grid(x,y,z,100,'points',np.nanmin(x),np.nanmax(x),np.
↪nanmin(y),np.nanmax(y),np.nanmin(z),np.nanmax(z),'all','nearest','yes')

x=(-90-(-91))*np.random.rand(1000)+(-91)
y=(31-(30))*np.random.rand(1000)+(30)
z=x**2+y**2
xgrid,ygrid,zgrid=sm.interpxyz2grid(x,y,z,0.005,'length',np.nanmin(x),np.nanmax(x),np.
↪nanmin(y),np.nanmax(y),np.nanmin(z),np.nanmax(z),'all','linear','yes')
```

References

Geospatial data

- https://www.mathworks.com/help/map/finding-geospatial-data.html
- https://maps.ngdc.noaa.gov/viewers/wcs-client/
- https://www.ngdc.noaa.gov/mgg/global/global.html

- https://www.ngdc.noaa.gov/mgg/global/relief/ETOPO1/
- https://www.ngdc.noaa.gov/mgg/image/2minrelief.html
- https://www.ngdc.noaa.gov/mgg/coastal/crm.html
- https://viewer.nationalmap.gov/launch/
- https://earthexplorer.usgs.gov
- http://www.shadedrelief.com/cleantopo2/index.html

scientimate.replacemissing1d

```
xReplaced, NaN_Indx = scientimate.replacemissing1d(x, what2replace='both',␣
↪interpMethod='linear', dispout='no')
```

Description

Replace missing data points in 1d data such as time series

Inputs

x Input data

what2replace='both'

What needs to be replaced
'NaN': replacing NaN data points
'Inf': replacing Inf data points
'both': replacing NaN and Inf data points
Number: replacing data points equal to Number

interpMethod='linear'

Interpolation method for replacing spike points:
Matlab/Octave: 'linear', 'nearest', 'next', 'previous', 'pchip', 'cubic', 'spline'
Python/Scipy : 'linear', 'nearest', 'zero', 'slinear', 'quadratic', 'cubic'

dispout='no' Define to display outputs or not ('yes': display, 'no': not display)

Outputs

xReplaced Replaced data

NaN_Indx Logical index of replaced points

Examples

```
import scientimate as sm
import numpy as np
import scipy as sp
from scipy import signal

fs=128
t=np.linspace(0,9.5,10*fs)
x=np.sin(2*np.pi*0.3*t)+0.1*np.sin(2*np.pi*4*t)
spikeloc=np.arange(10,len(t),100)
x[np.int64(spikeloc+np.round(2*np.random.randn(len(spikeloc))))]=np.nan
x=x+5
x[220:225]=np.nan
xReplaced,NaN_Indx=sm.replacemissing1d(x,'NaN','linear','yes')

fs=2
t=np.linspace(0,1023.5,1024*fs)
x=sp.signal.detrend(0.5*np.cos(2*np.pi*0.2*t)+(-0.1+(0.1-(-0.1)))*np.random.
↪rand(1024*fs))
spikeloc=np.arange(10,len(t),100)
x=x+1
x[np.int64(spikeloc+np.round(2*np.random.randn(len(spikeloc))))]=0
xReplaced,NaN_Indx=sm.replacemissing1d(x,0,'linear','yes')
```

References

scientimate.replaceoutlier

```
xReplaced, outlier_Indx = scientimate.replaceoutlier(x, WindowSize=15, zscore_
↪threshold=2, interpMethod'linear', dispout'no')
```

Description

Remove outliers in the time series using moving z-score window

Inputs

x Input data

WindowSize=15 Window size (number of adjacent elements) that is used for moving window, should be equal or larger than 3

zscore_threshold=2

z-score threshold to define outliers
data in range of x < (xmean-zscore_threshold*std) or x > (xmean+zscore_threshold*std) considered outliers

interpMethod='linear'

Interpolation method for replacing spike points:
Matlab/Octave: 'linear', 'nearest', 'next', 'previous', 'pchip', 'cubic', 'spline'
Python/Scipy : 'linear', 'nearest', 'zero', 'slinear', 'quadratic', 'cubic'

dispout='no' Define to display outputs or not ('yes': display, 'no': not display)

Outputs

xReplaced Replaced data

outlier_Indx Logical index of replaced points

Examples

```
import scientimate as sm
import numpy as np
import scipy as sp
from scipy import signal

fs=128
t=np.linspace(0,9.5,10*fs)
x=np.sin(2*np.pi*0.3*t)+0.1*np.sin(2*np.pi*4*t)
spikeloc=np.arange(10,len(t),100)
x[np.int64(spikeloc+np.round(2*np.random.randn(len(spikeloc))))]=np.sign(np.random.
↪randn(len(spikeloc)))
x[220:225]=1.5
x=x+5
xReplaced,outlier_Indx=sm.replaceoutlier(x,37,2,'linear','yes')

fs=2
t=np.linspace(0,1023.5,1024*fs)
x=sp.signal.detrend(0.5*np.cos(2*np.pi*0.2*t)+(-0.1+(0.1-(-0.1)))*np.random.
↪rand(1024*fs))
spikeloc=np.arange(10,len(t),100)
x[np.int64(spikeloc+np.round(2*np.random.randn(len(spikeloc))))]=np.sign(np.random.
↪randn(len(spikeloc)))
xReplaced,outlier_Indx=sm.replaceoutlier(x,21,2,'linear','yes')
```

References

1.3.2 Hurricane

scientimate.stormsurge1d

```
Eta, x, maxsurgeheight, L = scientimate.stormsurge1d(h0, U10, m=0, dispout='no')
```

Description

Calculate one dimensional storm surge using Dean Dalrymple (1991) method

Inputs

h0 Deep-water depth in (m),

U10 10-min averaged wind velocity at 10 meter above a surface in (m/s)

m=0

Bed slope
Note: m=h0/L, where L is a length of the continental shelf in (m)

dispout='no' Define to display outputs or not ('yes': display, 'no': not display)

Outputs

Eta Surge height along a x axis in (m)

x

Points on x axis in (m)
x=0 located at a far-end of the water boundary
x=L located at a coastline

maxsurgeheight Maximum surge height at a coastline (x=L) in (m)

L

Length of the continental shelf in (m)
L=h0/m if m is not zero
L=50000 if m is zero (L=50000 meter for the hurricane Katrina)

Examples

```
import scientimate as sm

h0=42 #Example: h0=42 m for the hurricane Katrina
U10=40 #Example: U10=40 m/s for the hurricane Katrina
m=0.00084
Eta,x,maxsurgeheight,L=sm.stormsurge1d(h0,U10,m,'yes')
```

References

Dean, R. G., & Dalrymple, R. A. (1991). Water wave mechanics for engineers and scientists (Vol. 2). World Scientific Publishing Company.

Wu, J. (1982). Wind-stress coefficients over sea surface from breeze to hurricane. Journal of Geophysical Research: Oceans, 87(C12), 9704-9706.

scientimate.hurricanewindh80

```
Vxgrid, Vygrid, Vgrid, Vgxgrid, Vgygrid, Vggrid, Pgrid, Rgrid, RVmax, thetagrid,␣
↪thetaVgrid, thetaVgridtan = scientimate.hurricanewindh80(xgrid, ygrid, xCenter,␣
↪yCenter, Pc, Vgmax, Rknown, VgRknown, Rmax=423e3, Pn=101325, Rhoa=1.204, \
    VgToVCoeff=0.8, inflowdirCalcMethod='bretschneider', Vt=0, VtAzmdir=0,␣
↪backwindCalcMethod='no', distCalcMethod='gc', flattendata='no', savedata='no',␣
↪dispout='no')
```

Description

Generate hurricane wind and pressure data on given (x,y) points using method from Holland (1980)

Inputs

xgrid

x (longitude) of points which outputs are calculated at
xgrid can be a single point or 1d or 2d array

ygrid

y (latitude) of points which outputs are calculated at
ygrid can be a single point or 1d or 2d array

xCenter x (longitude) of hurricane center (track)

yCenter y (latitude) of hurricane center (track)

Pc Hurricane central surface pressure in (Pa)

Vgmax

Maximum hurricane 1-min averaged wind velocity at the gradient level in (m/s)
It can be estimated from Vmax=3.44*(1010-Pc*1e-2)^0.644; where Pc is in (Pa)
and, Vmax is 1-min averaged wind at a 10-m elevation in (m/s) (Atkinson & Holliday, 1977)
Hurricane translation velocity (forward velocity) needs to be removed from Vgmax before applying (e.g. Hu et al., 2012)
Hurricane translation velocity (forward velocity) needs to be added after rotational velocity is calculated

Rknown

Radius that hurricane wind velocity is known at that radius in (m)
Rknown should be larger than radius associated to Vgmax

VgRknown Hurricane wind velocity at the gradient level which is known at radius Rknown in (m/s)

Rmax=423e3

Maximum radius of hurricane from hurricane center in (m)
Velocity outputs for points with R>Rmax is set to zero
Median values of Rmax is 423 km (e.g. Chavas & Emanuel 2010; Lin & Chavas, 2012)

Pn=101325

Ambient surface pressure (external pressure) in (Pa)
Standard atmosphere pressure is 101325 (Pa)
Typical value: Pn=101500 (Pa) for the western North Pacific, Pn= 101000 (Pa) for the North Atlantic (Batke et al., 2014)

Rhoa=1.204 Air density at the gradient level in (kg/m3)

VgToVCoeff=0.8

Coefficient to convert gradient wind velocity to wind velocity at 10 m above surface as:
V=VgToVCoeff*Vg, if VgToVCoeff=1, then V=Vg

inflowdirCalcMethod='bretschneider'

Inflow angle calculation method
'no': Inflow angle are not calculated, thetaVgrid=thetaVgridtan
'bretschneider': Inflow angle are calculated based on Bretschneider (1972)
'sobey': Inflow angle are calculated based on Sobey et al. (1977)

Vt=0 Hurricane central translational velocity in (m/s)

VtAzmdir=0

Hurricane center velocity azimuth (bearing) direction in (Degree)
azimuth (bearing) direction which is measured clockwise from the north:
0 (degree): toward North, 90 (degree): toward East, 180 (degree): toward South, 270 (degree): toward West

backwindCalcMethod='no'

Calculation method for adding background wind velocity due to hurricane motion
background wind velocity is added to points with R<=Rmax, e.g. Chavas & Emanuel (2010), Lin & Chavas (2012)
'no': background wind velocity is not added to hurricane velocities
'constant': background wind velocity is added as constant value to hurricane velocities
'slosh': Background wind velocity is calculated and added using SLOSH model by Jelesnianski et al. (1992)
'lin': Background wind velocity is calculated and added based on Lin & Chavas (2012)

distCalcMethod='gc'

Distance calculation method
'cart': Distances are calculated on cartesian coordinate
'gc': Distances are calculated on Great Circle based on Vincenty formula, Vincenty (1975)
Earth radius coonsidered as mean earth radius=6371000 m

flattendata='no'

Define if flat data or not
'no': does not flat the results, outputs are in 3d array
'yes': flat the results, outputs are in 2d array

savedata='no'

Define if save data in a file or not in working folder
'no': does not save,
'yes': save data as ascii 'dat' file, data are flatten regrdless of flattendata value

dispout='no'

Define to display outputs or not
'imagesc': 2 dimensional plot using imagesc or imshow
'pcolor': 2 dimensional plot using pcolor
'contour': 2 dimensional contour plot, number of contour=ncolor
'quiver': 2 dimensional vector plot
'no': not display
Use dispout='no' if calculation mesh is not 2d array
if there is more than one time step, only the last one is plotted
if flattendata='yes'; then dispout is set as dispout='no';

Outputs

Vxgrid

Hurricane 1-min averaged wind velocity at 10 m above surface in x (East) direction on defined mesh in (m/s)
Gradient wind velocity converted to wind velocity at 10 m above surface by V=VgToVCoeff*Vg

Vygrid

Hurricane 1-min averaged wind velocity at 10 m above surface in y (North) direction on defined mesh in (m/s)
Gradient wind velocity converted to wind velocity at 10 m above surface by V=VgToVCoeff*Vg

Vgrid

Resultant hurricane 1-min averaged wind velocity at 10 m above surface (Vx^2+Vy^2)^0.5 on defined mesh in (m/s)
Gradient wind velocity converted to wind velocity at 10 m above surface by V=VgToVCoeff*Vg

Vgxgrid Hurricane 1-min averaged gradient wind velocity at the gradient level in x (East) direction on defined mesh in (m/s)

Vgygrid Hurricane 1-min averaged gradient wind velocity at the gradient level in y (North) direction on defined mesh in (m/s)

Vggrid Resultant hurricane 1-min averaged gradient wind velocity at the gradient level on defined mesh in (m/s)

Pgrid Hurricane surface pressure on defined mesh in (Pa)

Rgrid Distance (radius) from hurricane center to each point on the grid

RVmax Distance (radius) from hurricane center to a location of maximum hurricane wind velocity (m)

thetagrid Angle from hurricane center to each point on the grid in (Degree)

thetaVgrid

Inflow angle (trigonometric direction) of hurricane velocity at each grid point in (Degree)
Inflow angle: angle between the inwardly spiraling surface wind
and the circular isobars around the hurricane center (Boose et al., 2004)

thetaVgridtan

Angle (trigonometric direction) of hurricane velocity at each grid point in (Degree)
thetaVgridtan is tangential angle respect to radius.
Note: Outputs has dimension of [M,N,L] where [M,N] is size of the x-y grid and [L] is number of time steps
If flattendata='yes'; then Outputs has dimension of [M*L,N]
Hurricane translation velocity needs to be added after rotational velocity is calculated
(e.g. Hu et al., 2012; Lin & Chavas, 2012)
Gradient wind velocity is converted to standard wind height as
wind velocity at 10 m above surface by V=VgToVCoeff*Vg
1-min averaged wind velocity needs to be converted to standard duration such as
10-min averaged wind by using a gust factor

Examples

```
import scientimate as sm
import numpy as np
#import matplotlib.pyplot as plt


#EXAMPLE 1

#Creating calculation mesh
xgrid,ygrid=np.meshgrid(np.linspace(-98,-68,100),np.linspace(16,44,100))

#Longitude of Hurricane Katrine center at max velocity
```

(continues on next page)

(continued from previous page)

```
longCenter=-88.6

#Latitude of Hurricane Katrine center at max velocity
latCenter=26.3

#Hurricane Katrina centeral pressure (Pa) at max velocity
Pc=90200

#Hurricane Katrina translational velocity (m/s) at max velocity
Vt=5.18467

#Hurricane Katrina velocity azimuth (bearing) in (Degree) at max velocity
VtAzmdir=306.76219

#Hurricane Katrina 1-min sustained maximum velocity (m/s) at max velocity
Vmax=76.5
Vmax=Vmax-Vt #Removing hurricane translation velocity from Vmax
Vgmax=Vmax/0.8 #Converting surface velocity to gradient velocity

#34 kt (17.49 m/s) wind radii maximum extent in northeastern quadrant in (m) for␣
↪Hurricane Katrina at max velocity
Rknown=370400
VRknown=17.49
VRknown=VRknown-Vt #Removing hurricane translation velocity from VRknown
VgRknown=VRknown/0.8 #Converting surface velocity to gradient velocity

Pn=101325 #Ambient surface pressure (external pressure) in (Pa)
Rhoa=1.204 #Air density in (kg/m3)

Vxgrid,Vygrid,Vgrid,Vgxgrid,Vgygrid,Vggrid,Pgrid,Rgrid,RVmax,thetagrid,thetaVgrid,
↪thetaVgridtan=sm.hurricanewindh80(xgrid,ygrid,longCenter,latCenter,Pc,Vgmax,Rknown,
↪VgRknown,423e3,Pn,Rhoa,\
    0.8,'bretschneider',Vt,VtAzmdir,'slosh','gc','no','no','quiver')

#Converting 1-min sustained wind to 10-min averaged wind using gust factor
#e.g. World Meteorological Organization (2015)
Vxgrid=Vxgrid*0.88
Vygrid=Vygrid*0.88
Vgrid=Vgrid*0.88


#EXAMPLE 2

#Creating calculation mesh
xgrid,ygrid=np.meshgrid(np.linspace(-98,-68,100),np.linspace(16,44,100))

#Longitude of Hurricane Katrine best track
longtrack=[-75.1,-75.7,-76.2,-76.5,-76.9,-77.7,-78.4,-79.0,-79.6,-80.1,-80.3,-81.3,\
    -82.0,-82.6,-83.3,-84.0,-84.7,-85.3,-85.9,-86.7,-87.7,-88.6,-89.2,-89.6,\
    -89.6,-89.6,-89.6,-89.6,-89.1,-88.6,-88.0,-87.0,-85.3,-82.9]

#Latitude of Hurricane Katrine best track
lattrack=[23.1,23.4,23.8,24.5,25.4,26.0,26.1,26.2,26.2,26.0,25.9,25.4,\
    25.1,24.9,24.6,24.4,24.4,24.5,24.8,25.2,25.7,26.3,27.2,28.2,\
    29.3,29.5,30.2,31.1,32.6,34.1,35.6,37.0,38.6,40.1]

#Hurricane Katrina centeral pressure (Pa)
```

(continues on next page)

(continued from previous page)

```
Pc=[100800,100700,100700,100600,100300,100000,99700,99400,98800,98400,98300,98700,\
    97900,96800,95900,95000,94200,94800,94100,93000,90900,90200,90500,91300,\
    92000,92300,92800,94800,96100,97800,98500,99000,99400,99600]

#Hurricane Katrina translational velocity (m/s)
Vt=np.array([0.00000,3.23091,3.13105,3.86928,4.99513,4.82816,3.27813,2.81998,2.77140,
↪2.53041,\
    1.05928,5.30662,3.60661,2.98269,3.61863,3.43691,3.28168,2.85849,3.20404,4.26279,\
    5.31340,5.18467,5.39195,5.46121,5.66270,1.02958,3.60354,4.63312,8.02540,8.01558,\
    8.12721,8.31580,10.75406,12.28350])

#Hurricane Katrina velocity azimuth (bearing) in (Degree)
VtAzmdir=[0.00000,298.67291,311.22135,338.70264,338.13626,309.94476,279.18860,280.
↪65053,270.13245,\
    246.10095,240.96690,241.20181,244.79591,249.93382,244.88325,252.71384,270.14459,
↪280.49918,\
    298.94148,299.05364,299.18896,306.76219,329.36839,340.59069,0.00000,0.00000,0.
↪00000,\
    0.00000,15.67775,15.42254,18.00215,29.63266,39.49673,50.29744]

#Hurricane Katrina 1-min sustained maximum velocity (m/s)
Vmax=np.array([15.3,15.3,15.3,17.850,20.4,22.950,25.5,28.050,30.6,35.7,35.7,33.150,\
    38.250,43.350,45.9,48.450,51.0,51.0,51.0,63.750,73.950,76.5,71.4,63.750,\
    56.1,56.1,53.550,40.8,25.5,20.4,15.3,15.3,15.3,12.750])

Vmax=Vmax-Vt #Removing hurricane translation velocity from Vmax
Vgmax=Vmax/0.8 #Converting surface velocity to gradient velocity

#34 kt (17.49 m/s) wind radii maximum extent in northeastern quadrant in (m) for␣
↪Hurricane Katrina
RknownRaw=[0,0,0,111120,111120,111120,111120,111120,129640,np.nan,129640,138900,\
    138900,138900,166680,240760,240760,259280,259280,296320,333360,370400,370400,
↪370400,\
    np.nan,370400,np.nan,185200,138900,138900,0,0,0,0]

#34 kt (17.49 m/s) wind radii maximum extent in northeastern quadrant in (m) for␣
↪Hurricane Katrina
Rknown=[0,0,0,111120,111120,111120,111120,111120,129640,129640,129640,138900,\
    138900,138900,166680,240760,240760,259280,259280,296320,333360,370400,370400,
↪370400,\
    370400,370400,277800,185200,138900,138900,0,0,0,0]
VRknown=np.ones(34)*17.49
VRknown=VRknown-Vt #Removing hurricane translation velocity from VRknown
VgRknown=VRknown/0.8 #Converting surface velocity to gradient velocity

Pn=101325 #Ambient surface pressure (external pressure) in (Pa)
Rhoa=1.204 #Air density in (kg/m3)

Vxgrid,Vygrid,Vgrid,Vgxgrid,Vgygrid,Vggrid,Pgrid,Rgrid,RVmax,thetagrid,thetaVgrid,
↪thetaVgridtan=smhurricanewindh80(xgrid,ygrid,longtrack[3:27],lattrack[3:27],
↪Pc[3:27],Vgmax[3:27],Rknown[3:27],VgRknown[3:27],423e3,Pn,Rhoa,\
    0.8,'bretschneider',Vt[3:27],VtAzmdir[3:27],'slosh','gc','no','no','quiver')

#Converting 1-min sustained wind to 10-min averaged wind using gust factor
#e.g. World Meteorological Organization (2015)
Vxgrid=Vxgrid*0.88
Vygrid=Vygrid*0.88
```

(continues on next page)

(continued from previous page)

```
Vgrid=Vgrid*0.88


#EXAMPLE 3

xgrid=np.linspace(0,10,100) #(Degree)
ygrid=np.ones(100)*20 #(Degree)
longCenter=0 #(Degree)
latCenter=20 #(Degree)
Pc=90200 #(Pa)
Vt=5.18467 #(m/s)
VtAzmdir=306.76219 #(Degree)
Vmax=76.5 #(m/s)
Vmax=Vmax-Vt
Vgmax=Vmax/0.8 #(m/s)
Rknown=370400 #(m)
VRknown=17.49 #(m/s)
VRknown=VRknown-Vt
VgRknown=VRknown/0.8 #(m/s)
Pn=101325 #Ambient surface pressure (external pressure) in (Pa)
Rhoa=1.204 #Air density in (kg/m3)

Vxgrid,Vygrid,Vgrid,Vgxgrid,Vgygrid,Vggrid,Pgrid,Rgrid,RVmax,thetagrid,thetaVgrid,
↪thetaVgridtan=sm.hurricanewindh80(xgrid,ygrid,longCenter,latCenter,Pc,Vgmax,Rknown,
↪VgRknown,423e3,Pn,Rhoa,\
0.8,'bretschneider',Vt,VtAzmdir,'slosh','gc','no','no','no')
plt.plot(Rgrid,Vgrid)
```

References

Data

- www.nhc.noaa.gov/data/
- www.nhc.noaa.gov/data/hurdat/hurdat2-format-nencpac.pdf
- coast.noaa.gov/hurricanes
- www.aoml.noaa.gov/hrd/data_sub/re_anal.html

Atkinson, G. D., & Holliday, C. R. (1977). Tropical cyclone minimum sea level pressure/maximum sustained wind relationship for the western north Pacific. Monthly Weather Review, 105(4), 421-427.

Batke, S. P., Jocque, M., & Kelly, D. L. (2014). Modelling hurricane exposure and wind speed on a mesoclimate scale: a case study from Cusuco NP, Honduras. PloS one, 9(3), e91306.

Boose, E. R., Serrano, M. I., & Foster, D. R. (2004). Landscape and regional impacts of hurricanes in Puerto Rico. Ecological Monographs, 74(2), 335-352.

Bretschneider, C. L. (1972, January). A non-dimensional stationary hurricane wave model. In Offshore Technology Conference. Offshore Technology Conference.

Chavas, D. R., & Emanuel, K. A. (2010). A QuikSCAT climatology of tropical cyclone size. Geophysical Research Letters, 37(18).

Department of the Army, Waterways Experiment Station, Corps of Engineers, and Coastal Engineering Research Center (1984), Shore Protection Manual, Washington, D.C., vol. 1, 4th ed., 532 pp.

Graham and Numm (1959) Meteorological Conditions Pertinent to Standard Project Hurricane, Atlantic and Gulf Coasts of United States. National Hurricane Research Project. U.S. Weather Service, Report no. 33.

Holland, G. J. (1980). An analytic model of the wind and pressure profiles in hurricanes. Monthly weather review, 108(8), 1212-1218.

Holland, G. (2008). A revised hurricane pressure–wind model. Monthly Weather Review, 136(9), 3432-3445.

Holland, G. J., Belanger, J. I., & Fritz, A. (2010). A revised model for radial profiles of hurricane winds. Monthly Weather Review, 138(12), 4393-4401.

Hu, K., Chen, Q., & Kimball, S. K. (2012). Consistency in hurricane surface wind forecasting: an improved parametric model. Natural hazards, 61(3), 1029-1050.

Jelesnianski, C. P., Chen, J., & Shaffer, W. A. (1992). SLOSH: Sea, lake, and overland surges from hurricanes (Vol. 48). US Department of Commerce, National Oceanic and Atmospheric Administration, National Weather Service.

Lin, N., & Chavas, D. (2012). On hurricane parametric wind and applications in storm surge modeling. Journal of Geophysical Research: Atmospheres, 117(D9).

Phadke, A. C., Martino, C. D., Cheung, K. F., & Houston, S. H. (2003). Modeling of tropical cyclone winds and waves for emergency management. Ocean Engineering, 30(4), 553-578.

Powell, M. D., Vickery, P. J., & Reinhold, T. A. (2003). Reduced drag coefficient for high wind speeds in tropical cyclones. Nature, 422(6929), 279.

Sobey, R. J., Harper, B. A., & Stark, K. P. (1977). Numerical simulation of tropical cyclone storm surge. James Cook University of North Queensland, Department of Civil & Systems Engineering.

U.S. Army Corps of Engineers (2015). Coastal Engineering Manual. Engineer Manual 1110-2-1100, Washington, D.C.: U.S. Army Corps of Engineers.

Valamanesh, V., Myers, A. T., Arwade, S. R., Hajjar, J. F., Hines, E., & Pang, W. (2016). Wind-wave prediction equations for probabilistic offshore hurricane hazard analysis. Natural Hazards, 83(1), 541-562.

Wei, K., Arwade, S. R., Myers, A. T., Valamanesh, V., & Pang, W. (2017). Effect of wind and wave directionality on the structural performance of non-operational offshore wind turbines supported by jackets during hurricanes. Wind Energy, 20(2), 289-303.

World Meteorological Organization. Tropical Cyclone Programme, & Holland, G. J. (2015). Global guide to tropical cyclone forecasting. Secretariat of the World Meteorological Organization.

Young, I. R., & Vinoth, J. (2013). An 'extended fetch' model for the spatial distribution of tropical cyclone wind–waves as observed by altimeter. Ocean Engineering, 70, 14-24.

1.3.3 Mapping

scientimate.convertdir

```
dirout = scientimate.convertdir(dirin, CalcMethod='metetotrig')
```

Description

Convert direction from one system to another one

Inputs

dirin Direction to be converted in (Degree)

CalcMethod='metetotrig'

Direction conversion method
'metetotrig': Converting meteorological direction to trigonometric direction, trig=-mete+270;
'trigtomete': Converting trigonometric direction to meteorological direction, mete=270-trig;
'aztomete': Converting azimuth (bearing) to meteorological direction, mete=az+180;
'metetoaz': Converting meteorological direction to azimuth (bearing), az=mete-180;
'aztotrig': Converting azimuth (bearing) to trigonometric direction, trig=-az+90;
'trigtoaz': Converting trigonometric direction to azimuth (bearing), az=90-trig;
meteorological direction:

0 (degree): from North, 90 (degree): from East, 180 (degree): from South, 270 (degree): from West

azimuth (bearing) direction which is measured clockwise from the north:

0 (degree): toward North, 90 (degree): toward East, 180 (degree): toward South, 270 (degree): toward West

Outputs

dirout Converted direction in (Degree)

Examples

```
import scientimate as sm

metedir=[0,90,180,270,360]
dirout=sm.convertdir(metedir,'metetotrig')

azimuth=[0,90,180,270,360]
dirout=sm.convertdir(azimuth,'aztomete')

azimuth=[0,90,180,270,360]
dirout=sm.convertdir(azimuth,'aztotrig')
```

References

scientimate.distancecart

```
distxy, theta = scientimate.distancecart(x1, y1, x2, y2, CalcMethod='1d', dispout='no
↪')
```

Description

Calculate distance from (x1,y1) to (x2,y2) on cartesian coordinate

Inputs

x1 x of start point (first point)

y1 y of start point (first point)

x2 x of end point (last point)

y2 y of end point (last point)

CalcMethod='1d'

Distance calculation method
'1d': use 1d array
'pdist2': Use 2d distance function
'vector': Use vectorized distance

dispout='no' Define to display outputs or not ('yes': display, 'no': not display)

Outputs

distxy

Distance from (x1,y1) to (x2,y2)
returns M*N array where M=length(x1) and N=length(x2)
mth row associated with mth point in (x,y)
nth column is associated with nth point in (x2,y2)

theta

Angle from start point to end point in (Degree)
returns M*N array where M=length(x1) and N=length(x2)
mth row associated with mth point in (x,y)
nth column is associated with nth point in (x2,y2)

Examples

```
import scientimate as sm
import numpy as np

x1=10*np.random.rand(100)
y1=10*np.random.rand(100)
x2=[2.5,5,7.5]
y2=[3,6,9]
distxy,theta=sm.distancecart(x1,y1,x2,y2,'1d','yes')

x1=10*np.random.rand(100)
y1=10*np.random.rand(100)
x2=100*np.random.rand(10)
y2=100*np.random.rand(10)
distxy,theta=sm.distancecart(x1,y1,x2,y2,'pdist2','yes')
```

References

scientimate.distancegc

```
arclen, azimuthdir, metedir = scientimate.distancegc(lat1, lon1, lat2, lon2,␣
↪CalcMethod='haversine', R=6371000, dispout='no')
```

Description

Calculate distance and azimuth (bearing) between (Latitude,Longitude) points using Great Circle

Inputs

lat1 Latitude (y) of start point (first point) in (Degree)

lon1 Longitude (x) of start point (first point) in (Degree)

lat2 Latitude (y) of end point (last point) in (Degree)

lon2 Longitude (x) of end point (last point) in (Degree)

CalcMethod='haversine'

Distance calculation method
'cos': Spherical law of cosines
'haversine': Haversine formula
'vincenty': Vincenty formula, Vincenty (1975)

R=6371000 Earth radius in (m), mean earth radius=6371000 m

dispout='no' Define to display outputs or not ('yes': display, 'no': not display)

Outputs

arclen Total distance from start point to end point in (m)

azimuthdir

Azimuth (bearing or compass direction) from start point to end point in (Degree)
0 (degree): toward North, 90 (degree): toward East, 180 (degree): toward South, 270 (degree): toward West

metedir

Meteorological direction from start point to end point in (Degree)
0 (degree): from North, 90 (degree): from East, 180 (degree): from South, 270 (degree): from West

Examples

```
import scientimate as sm

lat1=29.5 #First point
lon1=-89.4 #First point
lat2=29.7 #last point
lon2=-89.4 #last point
arclen,azimuthdir,metedir=sm.distancegc(lat1,lon1,lat2,lon2)
```

(continues on next page)

(continued from previous page)

```
lat1=[29.5,29] #First point
lon1=[-89.4,-89] #First point
lat2=[29.7,30] #Last point
lon2=[-89.4,-90] #Last point
arclen,azimuthdir,metedir=sm.distancegc(lat1,lon1,lat2,lon2,'haversine',6371000,'yes')
```

References

Vincenty, T. (1975). Direct and inverse solutions of geodesics on the ellipsoid with application of nested equations. Survey review, 23(176), 88-93.

http://www.movable-type.co.uk/scripts/latlong.html
https://en.wikipedia.org/wiki/Great-circle_distance
https://en.wikipedia.org/wiki/Great-circle_navigation
http:
//www.codeguru.com/cpp/cpp/algorithms/article.php/c5115/Geographic-Distance-and-Azimuth-Calculations.htm

scientimate.endpointcart

```
x2, y2, lineslope = scientimate.endpointcart(x1, y1, linelength=1, lineangle=0,␣
↪dispout='no')
```

Description

Find an end point of the straight line segment from its starting point (x1,y1) and its angle on cartesian coordinate

Inputs

x1 x of start point (first point)

y1 y of start point (first point)

linelength=1 Length of a line segment

lineangle=0 Angle of a line segment from start point (first point) toward end point (last point) in (Degree)

dispout='no' Define to display outputs or not ('yes': display, 'no': not display)

Outputs

x2 x of end point (last point)

y2 y of end point (last point)

lineslope

Slope of a line segment from start point (first point) toward end point (last point) in (Degree)
lineslope=tan(deg2rad(lineangle))

Examples

```
import scientimate as sm

x1=3
y1=3
linelength=5
lineangle=45
x2,y2,lineslope=sm.endpointcart(x1,y1,linelength,lineangle,'yes')

x1=[0,0,0,0,0,0,0,0]
y1=[0,0,0,0,0,0,0,0]
linelength=5
lineangle=[0,45,90,135,180,225,270,315]
x2,y2,lineslope=sm.endpointcart(x1,y1,linelength,lineangle,'no')
```

References

scientimate.globalrelief

```
x, y, z, xgrid, ygrid, zgrid = scientimate.globalrelief(xmin=-180, xmax=180, ymin=-90,
↪ ymax=90, dispout='no')
```

Description

Return x (longitude), y (latitude) and z (elevation) data from ETOPO1 Global Relief Model (Amante & Eakins, 2009) interpolated on 0.125 degree grid

ETOPO1 is 1 arc-minute global relief, however, this data are interpolated on 7.5 arc-minute (0.125 degree)

This data are obtained from ETOPO1 Global Relief Bedrock (grid-registered)

ETOPO1 horizontal datum: WGS 84 geographic

ETOPO1 vertical datum: sea level

https://www.ngdc.noaa.gov/mgg/global/global.html

Inputs

xmin=-180

Minimum x (longitude) of domain to be returned in degree
It should be between -180 and 180 degree

xmax=180

Maximum x (longitude) of domain to be returned in degree
It should be between -180 and 180 degree

ymin=-90

Minimum y (latitude) of domain to be returned in degree
It should be between -90 and 90 degree

ymax=90

Maximum y (latitude) of domain to be returned in degree
It should be between -90 and 90 degree

dispout='no' Define to display outputs or not ('yes': display, 'no': not display)

Outputs

x Interpolated x (longitude) data in degree

y Interpolated y (latitude) data in degree

z Interpolated z (elevation) data in degree

xgrid Interpolated x (longitude) data on 2d mesh in degree

ygrid Interpolated y (latitude) data on 2d mesh in degree

zgrid Interpolated z (elevation) data on 2d mesh in degree

Examples

```
import scientimate as sm

#Globe
x,y,z,xgrid,ygrid,zgrid=sm.globalrelief(-180,-180,-90,90,'yes')

#Middle East
x,y,z,xgrid,ygrid,zgrid=sm.globalrelief(24,64,9,43,'yes')

#North America
x,y,z,xgrid,ygrid,zgrid=sm.globalrelief(-169,-8,5,90,'yes')
```

References

ETOPO1 Global Relief Model

Amante, C. and B.W. Eakins, 2009. ETOPO1 1 Arc-Minute Global Relief Model: Procedures, Data Sources and Analysis. NOAA Technical Memorandum NESDIS NGDC-24. National Geophysical Data Center, NOAA. doi:10.7289/V5C8276M

- https://www.ngdc.noaa.gov/mgg/global/global.html
- https://maps.ngdc.noaa.gov/viewers/grid-extract/index.html
- https://www.ngdc.noaa.gov/mgg/global/relief/ETOPO1/
- https://data.nodc.noaa.gov/cgi-bin/iso?id=gov.noaa.ngdc.mgg.dem:316

GEBCO Global ocean & land terrain models

- https://www.gebco.net/data_and_products/gridded_bathymetry_data/

Natural Earth 1:10m Raster Data

- https://www.naturalearthdata.com/downloads/10m-raster-data/

Geospatial data

- https://www.mathworks.com/help/map/finding-geospatial-data.html

- https://www.ngdc.noaa.gov/mgg/global/etopo2.html
- https://www.ngdc.noaa.gov/mgg/global/etopo5.HTML
- https://www.ngdc.noaa.gov/mgg/image/2minrelief.html
- https://www.ngdc.noaa.gov/mgg/coastal/crm.html
- https://viewer.nationalmap.gov/launch/
- https://earthexplorer.usgs.gov
- http://www.shadedrelief.com/cleantopo2/index.html
- https://www.usna.edu/Users/oceano/pguth/md_help/html/bathymetry.htm
- https://en.wikipedia.org/wiki/Global_relief_model

scientimate.gridgenerator

```
xgrid, ygrid = scientimate.gridgenerator(xmin, xmax, ymin, ymax, gridsize=100,␣
↪gridsizetype='points', dispout='no')
```

Description

Generate 2d x-y grid

Inputs

xmin Minimum x of the domain to be generated

xmax Maximum x of the domain to be generated

ymin Minimum y of the domain to be generated

ymax Maximum y of the domain to be generated

gridsize=100

Grid size in x (longitude) and y (latitude) directions to interpolate elevation data on them
if gridsizetype='length' then gridsize is a distance between grid points
if gridsizetype='points' then gridsize is number of grid points in each direction

gridsizetype='points'

Grid size type
'number': gridsize is considered as number of grid points in each direction
'length': gridsize is considered as length between grid points

dispout='no' Define to display outputs or not ('yes': display, 'no': not display)

Outputs

xgrid x of the defined mesh

ygrid y of the defined mesh

Examples

```
import scientimate as sm
xgrid,ygrid=sm.gridgenerator(0,100,0,100,100,'points','yes')
xgrid,ygrid=sm.gridgenerator(0,100,0,100,20,'length','yes')
```

References

scientimate.readxyz

```
x, y, z = scientimate.readxyz(xyzfilename, xyzfilelocation=None, zscale=1, domain='all
↪', xmin=-180, xmax=180, ymin=-90, ymax=90, savedata='no', outfilename='xyzdata.csv',
↪ outfilelocation=None)
```

Description

Read and extract x (longitude), y (latitude) and z (elevation) data from ASCII gridded (tabular) xyz file
Use readdatafile function for more options

Inputs

xyzfilename

Name of xyz file between ‘ ‘ mark, example: ‘xyzfile.xyz’
xyz file should be in form of 3 coloumn format

xyzfilelocation=pwd Location of xyz file between ‘ ‘ mark, example: ‘C:’

zscale=1 Scale z (elevation) data by factor of zscale

domain=’all’

Define a domain to be extracted from data
‘all’: all xyz data in input file are extracted
‘domain’: only data within a defined domain are extracted

xmin=-180 Minimum x (longitude) of domain to be extracted

xmax=180 Maximum x (longitude) of domain to be extracted

ymin=-90 Minimum y (latitude) of domain to be extracted

ymax=90 Maximum y (latitude) of domain to be extracted

savedata=’no’

Define if save xyz data in a file or not in outfilelocation folder
‘no’: does not save
‘yes’: save xyz data as csv file

outfilename=’xyzdata.csv’

Name of output file between ‘ ‘ mark, example: ‘xyzdata.csv’
outfilename should have ‘.csv’ extension

outfilelocation=pwd Location of output file between ‘ ‘ mark, example: ‘C:’

Outputs

x x (longitude) data extracted from xyz file

y y (latitude) data extracted from xyz file

z z (elevation) data extracted from xyz file

Examples

```
import scientimate as sm

xyzfilename='xyzfile.xyz' #e.g. xyzfilename='PersianGulf_ETOPO1.xyz'
xyzfilelocation='C:/' #e.g. xyzfilelocation='C:/datafolder'
x,y,z=sm.readxyz(xyzfilename,xyzfilelocation)

xyzfilename='xyzfile.xyz' #e.g. xyzfilename='PersianGulf_ETOPO1.xyz'
xyzfilelocation='C:/' #e.g. xyzfilelocation='C:/datafolder'
x,y,z=sm.readxyz(xyzfilename,xyzfilelocation,1,'all',-180,180,-90,90,'no')
```

References

Geospatial data

- https://www.mathworks.com/help/map/finding-geospatial-data.html
- https://maps.ngdc.noaa.gov/viewers/wcs-client/
- https://www.ngdc.noaa.gov/mgg/global/global.html
- https://www.ngdc.noaa.gov/mgg/global/relief/ETOPO1/
- https://www.ngdc.noaa.gov/mgg/image/2minrelief.html
- https://www.ngdc.noaa.gov/mgg/coastal/crm.html
- https://viewer.nationalmap.gov/launch/
- https://earthexplorer.usgs.gov
- http://www.shadedrelief.com/cleantopo2/index.html

scientimate.reckongc

```
lat2, lon2 = scientimate.reckongc(lat1, lon1, arclen, azimuthdir, R=6371000, dispout=
↪'no')
```

Description

Calculate end point (Latitude,Longitude) from start point (Latitude,Longitude) and distance and azimuth (bearing) using Great Circle

Inputs

lat1 Latitude (y) of start point (first point) in (Degree)

lon1 Longitude (x) of start point (first point) in (Degree)

arclen Total distance from start point to end point in (m)

azimuthdir

Azimuth (bearing or compass direction) from start point to end point in (Degree)
0 (degree): toward North, 90 (degree): toward East, 180 (degree): toward South, 270 (degree): toward West

R=6371000 Earth radius in (m), mean earth radius=6371000 m

dispout='no' Define to display outputs or not ('yes': display, 'no': not display)

Outputs

lat2 Latitude (y) of end point (last point) in (Degree)

lon2 Longitude (x) of end point (last point) in (Degree)

Examples

```
import scientimate as sm

lat1=29.5 #First point
lon1=-89.4 #First point
arclen=22239 #Arc length
azimuthdir=0 #Azimuth
lat2,lon2=sm.reckongc(lat1,lon1,arclen,azimuthdir)

lat1=[29.5,29] #First point
lon1=[-89.4,-89] #First point
arclen=[22239,147410] #Arc length
azimuthdir=[0,319.21] #Azimuth
lat2,lon2=sm.reckongc(lat1,lon1,arclen,azimuthdir,6371000,'yes')
```

References

http://www.movable-type.co.uk/scripts/latlong.html
https://en.wikipedia.org/wiki/Great-circle_distance
https://en.wikipedia.org/wiki/Great-circle_navigation
http:
//www.codeguru.com/cpp/cpp/algorithms/article.php/c5115/Geographic-Distance-and-Azimuth-Calculations.htm

1.3.4 OCEANLYZ

OCEANLYZ, Ocean Wave Analyzing Toolbox, is a toolbox for analyzing the wave time series data collected by sensors in open body of water such as ocean, sea, and lake or in a laboratory.
The Python version of OCEANLYZ toolbox is a part of the ScintiMate package.

For more information, visit https://oceanlyz.readthedocs.io

scientimate.oceanlyz (Python Version)

```
oceanlyz_object = scientimate.oceanlyz()
```

DESCRIPTION

Calculate wave properties from water level or water pressure data
For OCEANLYZ Document visit https://oceanlyz.readthedocs.io
For ScientiMate Document visit https://scientimate.readthedocs.io

Required Properties

Following properties are required for all abalysis

data=[]

Water level (water surface elevation, Eta), water depth, or water pressure time series

Data should be a single column array (column vector) without any text
Each burst of data should follow the previous burst without any void

InputType='waterlevel'

Define input data type

InputType='waterlevel': Input data is water level or water depth in (m) If InputType='waterlevel' then OutputType='wave'

InputType='pressure': Input data are water pressure measured by a pressure sensor in (N/m^2) If InputType='pressure' then OutputType='waterlevel' or OutputType='wave+waterlevel'

OutputType='wave'

Define output data type

OutputType='wave': Calculate wave properties from water level or water depth data
OutputType='waterlevel': Calculate water level data from water pressure data measured by a pressure sensor
OutputType='wave+waterlevel': Calculate waves properties and water level data from water pressure data measured by a pressure sensor

AnalysisMethod='spectral'

Analysis method

AnalysisMethod='spectral': Use spectral analysis method / Fast Fourier Transform
AnalysisMethod='zerocross': Use zero-crossing method

n_burst=1

Number of burst(s) in the input file

n_burst = (total number of data points)/(burst_duration*fs)
Example:

Assume data are collected for 6 hours at a sampling frequency of fs=10 Hz
If data are analyzed at intervals of 30 minutes then there are 12 bursts (6 hours/30 minutes=12 bursts)
For 12 bursts of data, which each burst has a duration of 30 minutes, and collected at sampling frequency of fs=10 Hz
burst_duration=(30 min * 60) = 1800 seconds
total number of data points=(number of burst)*(duration of each burst)*(sampling frequency)
total number of data points=(n_burst)*(burst_duration)*(fs)
total number of data points=12 * 1800 * 10

burst_duration=1024 Duration time that data collected in each burst in (second)

fs=2 Sampling frequency that data are collected at in (Hz)

Required Properties for Spectral Analysis

Following properties are needed only if AnalysisMethod='spectral'

fmin=0.05

Minimum frequency to cut off the spectrum below that, i.e. where f<fmin, in (Hz)

Results with frequency f<fmin will be removed from analysis
It should be 0 <= fmin <= (fs/2)
It is a simple high pass filter
Only required if AnalysisMethod='spectral'

fmax=1e6

Maximum frequency to cut off the spectrum beyond that, i.e. where f>fmax, in (Hz)

Results with frequency f>fmax will be removed from analysis
It should be 0 <= fmin <= (fs/2)
It is a simple low pass filter
Only required if AnalysisMethod='spectral'

Required Properties for Pressure Data Analysis

Following properties are needed only if InputType='pressure'

fmaxpcorrCalcMethod='auto'

Define if to calculate fmaxpcorr and ftail or to use user defined

fmaxpcorrCalcMethod='user': use user defined value for fmaxpcorr
fmaxpcorrCalcMethod='auto': automatically define value for fmaxpcorr
Only required if InputType='pressure' and AnalysisMethod='spectral'

Kpafterfmaxpcorr='constant'

Define a pressure response factor, Kp, value for frequency larger than fmaxpcorr

Kpafterfmaxpcorr='nochange': Kp is not changed for frequency larger than fmaxpcorr
Kpafterfmaxpcorr='one': Kp=1 for frequency larger than fmaxpcorr
Kpafterfmaxpcorr='constant': Kp for f larger than fmaxpcorr stays equal to Kp at fmaxpcorr (constant)
Only required if InputType='pressure' and AnalysisMethod='spectral'

fminpcorr=0.15

Minimum frequency that automated calculated fmaxpcorr can have if fmaxpcorrCalcMethod='auto' in (Hz)

If fmaxpcorrCalcMethod='auto', then fmaxpcorr will be checked to be larger or equal to fminpcorr
It should be 0 <= fmin <= (fs/2)
Only required if InputType='pressure' and AnalysisMethod='spectral'

fmaxpcorr=0.55

Maximum frequency for applying pressure attenuation factor in (Hz)

Pressure attenuation factor is not applied on frequency larger than fmaxpcorr
It should be 0 <= fmin <= (fs/2)
Only required if InputType='pressure' and AnalysisMethod='spectral'

heightfrombed=0.0

Pressure sensor height from a bed in (m) Leave heightfrombed=0.0 if data are not measured by a pressure sensor or if a sensor sits on the seabed | Only required if InputType='pressure'

Optional Properties

Following properties are optional

dispout='no'

Define if to plot spectrum or not

dispout='no': Does not plot
dispout='yes': Plot

Rho=1000

Water density (kg/m^3) Only required if InputType='pressure'

nfft=512

Define number of data points in discrete Fourier transform

Should be 2^n
Results will be reported for frequency range of 0 <= f <= (fs/2) with (nfft/2+1) data points
Example: If fs=4 Hz and nfft=512, then output frequency has a range of 0 <= f <= 2 with 257 data points
Only required if AnalysisMethod='spectral'

SeparateSeaSwell='no'

Define if to separate wind sea and swell waves or not

SeparateSeaSwell='yes': Does not separate wind sea and swell waves
SeparateSeaSwell='no': Separates wind sea and swell waves

fmaxswell=0.25

Maximum frequency that swell can have (It is about 0.2 in Gulf of Mexico) in (Hz)

It should be 0 <= fmin <= (fs/2)
Only required if SeparateSeaSwell='yes' and AnalysisMethod='spectral'

fpminswell=0.1

Minimum frequency that swell can have (it is used for Tpswell calculation) in (Hz)

It should be 0 <= fmin <= (fs/2)
Only required if SeparateSeaSwell='yes' and AnalysisMethod='spectral'

tailcorrection='off'

Define if to replace spectrum tail with tail of empirical spectrum (diagnostic tail) or not

tailcorrection='off': Does replace spectrum tail
tailcorrection='jonswap': Replace spectrum tail with JONSWAP Spectrum tail
tailcorrection='tma': Replace spectrum tail with TMA Spectrum tail

For tailcorrection='tma', input data should be water depth

ftailcorrection=0.9

Frequency that spectrum tail replaced after that in (Hz)

ftailcorrection is typically set at ftailcorrection=(2.5*fm) where (fm=1/Tm01)
It should be 0 <= fmin <= (fs/2)
Only required if SeparateSeaSwell='yes' and tailcorrection='jonswap' or tailcorrection='tma'

tailpower=-5

Power that a replaced tail (diagnostic tail)

Replaced tail (diagnostic tail) will be proportional to (f^tailpower)
Recommendation: use tailpower=-3 for shallow water and tailpower=-5 for deep water
Only required if SeparateSeaSwell='yes' and tailcorrection='jonswap' or tailcorrection='tma'

Methods

oceanlyz_object.runoceanlyz() Run oceanlyz and calculate wave properties

Outputs

oceanlyz_object.wave

Calculated wave properties as a Python dictionary

Output is a Python dictionary
Name of output is 'oceanlyz_object.wave'
Values(s) in this dictionary can be called by using 'key'
Example:

oceanlyz_object.wave['Hm0'] : Contain zero-moment wave height
oceanlyz_object.wave['Tp'] : Contain peak wave period
oceanlyz_object.wave['Field_Names'] : Contain key (variable) names in the wave dictionary
oceanlyz_object.wave['Burst_Data'] : Contain data for each burst

Examples

```
#Import libraries
import scientimate as sm
import numpy as np
import matplotlib.pyplot as plt
```

(continues on next page)

(continued from previous page)

```
import os

#Create OCEANLYZ object
#del ocn #Optional
ocn=sm.oceanlyz()

#Read data
#Assume data file is named 'waterpressure_5burst.csv' and is stored in 'C:\oceanlyz_
↪python\Sample_Data'
os.chdir('C:\\oceanlyz_python\\Sample_Data') #Change current path to Sample_Data␣
↪folder
water_pressure=np.genfromtxt('waterpressure_5burst.csv') #Load data

#Input parameters
ocn.data=water_pressure.copy()
ocn.InputType='pressure'
ocn.OutputType='wave+waterlevel'
ocn.AnalysisMethod='spectral'
ocn.n_burst=5
ocn.burst_duration=1024
ocn.fs=10
ocn.fmin=0.05                    #Only required if ocn.AnalysisMethod='spectral'
ocn.fmax=ocn.fs/2                #Only required if ocn.AnalysisMethod='spectral'
ocn.fmaxpcorrCalcMethod='auto'   #Only required if ocn.InputType='pressure' and ocn.
↪AnalysisMethod='spectral'
ocn.Kpafterfmaxpcorr='constant'  #Only required if ocn.InputType='pressure' and ocn.
↪AnalysisMethod='spectral'
ocn.fminpcorr=0.15               #Only required if ocn.InputType='pressure' and ocn.
↪AnalysisMethod='spectral'
ocn.fmaxpcorr=0.55               #Only required if ocn.InputType='pressure' and ocn.
↪AnalysisMethod='spectral'
ocn.heightfrombed=0.05           #Only required if ocn.InputType='pressure' and ocn.
↪AnalysisMethod='spectral'
ocn.dispout='yes'
ocn.Rho=1024                     #Seawater density (Varies)

#Run OCEANLYZ
ocn.runoceanlyz()

#Plot peak wave period (Tp)
plt.plot(ocn.wave['Tp'])
```

References

Karimpour, A., & Chen, Q. (2017). Wind wave analysis in depth limited water using OCEANLYZ, A MATLAB toolbox. Computers & Geosciences, 106, 181-189.

scientimate.PcorFFTFun

```
Eta,ftailcorrection=scientimate.PcorFFTFun(input,fs,duration,nfft,h,heightfrombed,
↪fminpcorr,fmaxpcorr,ftailcorrection,pressureattenuation,autofmaxpcorr,dispout)
```

DESCRIPTION

Apply pressure correction factor to water depth data from pressure gauge reading using FFT

INPUT

input=importdata('h.mat') Load water depth (h)/surface elevation (Eta) data and rename it "input" in (m)

fs=10 Sampling frequency that data collected at in (Hz)

duration=1024 Duration time that data collected in input in each burst in second

nfft=2^10 NFFT for Fast Fourier Transform

h=1 Mean water depth in (m)

heightfrombed=0.0 Sensor height from bed

fminpcorr=0.15 Minimum frequency that automated calculated fmaxpcorr can have if autofmaxpcorr='on' in (Hz)

fmaxpcorr=0.8 Maximum frequency for applying pressure attenuation factor

ftailcorrection=1 Frequency that diagnostic tail apply after that (typically set at 2.5fm, fm=1/Tm01)

pressureattenuation='all'

Define if to apply pressure attenuation factor or not pressureattenuation='off': No pressure attenuation applied

pressureattenuation='on': Pressure attenuation applied without correction after fmaxpcorr

pressureattenuation='all': Pressure attenuation applied with constant correction after fmaxpcorr

autofmaxpcorr='on'

Define if to calculate fmaxpcorr and ftailcorrection based on water depth or not autofmaxpcorr='off': Off

autofmaxpcorr='on': On

dispout='on' Define to display outputs or not ('off': not display, 'on': display)

OUTPUT

Eta Corrected Water Surface Level Time Series (m)

EXAMPLE

```
Eta,ftailcorrection=PcorFFTFun(water_pressure/(1000*9.81),10,1024,256,1.07,0.05,0.15,
↪0.8,1,'all','on','on')
```

scientimate.PcorZerocrossingFun

```
Eta=scientimate.PcorZerocrossingFun(input,fs,duration,h,heightfrombed,dispout)
```

DESCRIPTION

Apply pressure correction factor to water depth data from pressure gauge reading using up-going zerocrossing method

INPUT

input=importdata('h.mat') Load water depth (h)/surface elevation (Eta) data and rename it "input" in (m)

fs=10 Sampling frequency that data collected at in (Hz)

duration=1024 Duration time that data collected in input in each burst in second

h=1 Mean water level (m)

heightfrombed=0.0 Sensor height from bed

dispout='on' Define to display outputs or not ('off': not display, 'on': display)

OUTPUT

Eta Corrected Water Surface Level Time Series in (m)

EXAMPLE

```
Eta=PcorZerocrossingFun(input,10,1024,1.07,0.05,'on')
```

scientimate.SeaSwellFun

```
Hm0,Hm0sea,Hm0swell,Tp,Tpsea,Tpswell,fp,fseparation,f,Syy=scientimate.
↪SeaSwellFun(input,fs,duration,nfft,h,fmin,fmax,ftailcorrection,tailpower,fminswell,
↪fmaxswell,mincutoff,maxcutoff,tailcorrection,dispout)
```

DESCRIPTION

Separate sea wave from swell wave

INPUT

input=importdata('h.mat') Load water depth (h)/surface elevation (Eta) data and rename it "input" in (m)

fs=10 Sampling frequency that data collected at in (Hz)

duration=1024 Duration time that data collected in input in each burst in second

nfft=2^10 NFFT for Fast Fourier Transform

h=1 Mean water depth in (m)

fmin=0.04 Minimum frequency for cut off the lower part of spectra

fmax=1 Maximum frequency for cut off the upper part of spectra

ftailcorrection=1 Frequency that diagnostic tail apply after that (typically set at 2.5fm, fm=1/Tm01)

tailpower=-5 Power that diagnostic tail apply based on that (-3 for shallow water to -5 for deep water)

fminswell=0.1 Minimum frequency that is used for Tpswell calculation

fmaxswell=0.25 Maximum frequency that swell can have, It is about 0.2 in Gulf of Mexico

mincutoff='off'

Define if to cut off the spectra below fmin mincutoff='off': Cutoff off

mincutoff='on': Cutoff on

maxcutoff='off'

Define if to cut off the spectra beyond fmax maxcutoff='off': Cutoff off

maxcutoff='on': Cutoff on

tailcorrection='off'

Define if to apply diagnostic tail correction or not tailcorrection='off': Not apply

tailcorrection='jonswap': JONSWAP Spectrum tail

tailcorrection='tma': TMA Spectrum tail

dispout='on' Define to display outputs or not ('off': not display, 'on': display)

OUTPUT

Hm0 Zero-Moment Wave Height (m)

Hm0sea Sea Zero-Moment Wave Height (m)

Hm0swell Swell Zero-Moment Wave Height (m)

Tp Peak wave period (second)

Tpsea Peak Sea period (second)

Tpswell Peak Swell Period (second)

fp Peak Wave Frequency (Hz)

f Frequency (Hz)

fseparation Sea and Swell Separation Frequency (Hz)

Syy Wave Surface Elevation Power Spectrum (m^2s)

EXAMPLE

```
Hm0,Hm0sea,Hm0swell,Tp,Tpsea,Tpswell,fp,fseparation,f,Syy=SeaSwellFun(water_pressure/
↪(1000*9.81),10,1024,256,1.07,0.05,5,1,-5,0.1,0.25,'on','on','off','on')
```

scientimate.WaveSpectraFun

```
Hm0,Tm01,Tm02,Tp,fp,f,Syy=scientimate.WaveSpectraFun(input,fs,duration,nfft,h,
↪heightfrombed,fmin,fmax,ftailcorrection,tailpower,mincutoff,maxcutoff,
↪tailcorrection,dispout)
```

DESCRIPTION

Calculate wave properties from power spectral density

INPUT

input=importdata('h.mat') Load water depth (h)/surface elevation (Eta) data and rename it "input" in (m)

fs=10 Sampling frequency that data collected at in (Hz)

duration=1024 Duration time that data collected in input in each burst in second

nfft=2^10 NFFT for Fast Fourier Transform

h=1 Mean water depth in (m)

heightfrombed=0.0 Sensor height from bed

fmin=0.04 Minimum frequency for cut off the lower part of spectra

fmax=1 Maximum frequency for cut off the upper part of spectra

ftailcorrection=1 Frequency that diagnostic tail apply after that (typically set at 2.5fm, fm=1/Tm01)

tailpower=-4 Power that diagnostic tail apply based on that (-3 for shallow water to -5 for deep water)

mincutoff='off'

Define if to cut off the spectra below fmin mincutoff='off': Cutoff off

mincutoff='on': Cutoff on

maxcutoff='off'

Define if to cut off the spectra beyond fmax maxcutoff='off': Cutoff off

maxcutoff='on': Cutoff on

tailcorrection='off'

Define if to apply diagnostic tail correction or not tailcorrection='off': Not apply

tailcorrection='jonswap': JONSWAP Spectrum tail

tailcorrection='tma': TMA Spectrum tail

dispout='on' Define to display outputs or not ('off': not display, 'on': display)

OUTPUT

Hm0 Zero-Moment Wave Height (m)

Tm01 Wave Period from m01 (second), Mean Wave Period

Tm02 Wave Period from m02 (second), Mean Zero Crossing Period

Tp Peak Wave Period (second)

fp Peak Wave Frequency (Hz)

f Frequency (Hz)

Syy Wave Surface Elevation Power Spectrum (m^2s)

EXAMPLE

```
Hm0,Tm01,Tm02,Tp,fp,f,Syy=WaveSpectraFun(water_pressure/(1000*9.81),10,1024,256,1.07,
↪0.05,0.05,5,1,-5,'on','on','off','on')
```

scientimate.WaveZerocrossingFun

```
Hs,Hz,Tz,Ts,H,T=scientimate.WaveZerocrossingFun(input,fs,duration,dispout)
```

DESCRIPTION

Calculate wave properties using Up-going zero crossing method

INPUT

input=importdata('h.mat') Load water depth (h)/surface elevation (Eta) data and rename it "input" in (m)

fs=10 Sampling frequency that data collected at in (Hz)

duration=1024 Duration time that data collected in input in each burst in second

dispout='on' Define to display outputs or not ('off': not display, 'on': display)

OUTPUT

Hs Significant Wave Height (m)

Hz Zero Crossing Mean Wave Height (m)

Tz Zero Crossing Mean Wave Period (second)

Ts Significant Wave Period (second)

H Wave Height Data Series (m)

T Wave Period Data Series (second)

EXAMPLE

```
Hs,Hz,Tz,Ts,H,T=WaveZerocrossingFun(water_pressure/(1000*9.81),10,1024,'on')
```

OCEANLYZ toolbox consists of 1 class and 5 functions. The main class in OCEANLYZ toolbox is the oceanlyz(). To run OCEANLYZ toolbox, only the oceanlyz() class is required to be run. Based on parameters set by a user, oceanlyz() calls appropriate function(s) to analyze data. Note that, any of the OCEANLYZ functions might be used separately as well.

For more general purpose functions for wave analysis, see *Water Wave Data Analysis* section below.

1.3.5 Plotting

scientimate.plot2d

```
scientimate.plot2d(x, y=None, plottype='line', cmapcolors='blue', sizestyle='medium')
```

Description

Plot x and y data in 2-d plot

Inputs

x

x data as a 2-d array of (M,N)

y=[]

y data as a 2-d array of (M,N)

plottype='line'

Plot type
'line': line plot
'line_grid': line plot with grid lines
'line_ascend': line plot with ascending line width
'line_ascend_grid': line plot with ascending line width and grid lines
'line_confid': line plot with 95% confidence intervals band (approximated)
'line_confid_grid': line plot with 95% confidence intervals band (approximated) and grid lines
'scatter': scatter plot
'scatter_grid': scatter plot with grid lines
'scatter_ascend': scatter plot with ascending point size
'scatter_ascend_grid': scatter plot with grid lines and ascending point size
'bar': bar plot
'bar_grid': bar plot with grid lines
'bar_stacked': stacked bar plot
'barh': horizontal bar plot
'barh_grid': horizontal bar plot with grid lines
'barh_stacked': horizontal stacked bar plot
'histogram': histogram plot
'histogram_grid': histogram plot with grid lines

cmapcolors='blue'

Colormap style
'blue': blue colormap
'red': red colormap
'green': green colormap
'yellow': yellow colormap
'purple': purple colormap
'brown': brown colormap

'gray': gray colormap
'blue_red': blue-red colormap
'red_blue': red-blue colormap
'blue_green': blue-green colormap
'green_blue': green-blue colormap
'green_yellow': green-yellow colormap
'yellow_green': yellow-green colormap
'red_yellow': red-yellow colormap
'yellow_red': yellow-red colormap
'cyclic': cyclic/oscillation colormap
'seq': sequential colormap
User-defined colors may be used to generate colormap
User-defined colors should be defined as (M,3) array in RGB color format
At least two colors should be defined, i.e. M should be equal or larger than 2
User-defined colors values should be between 0 and 255
Any available colormap name such as 'cool', 'winter', etc also can be used

sizestyle='medium'

Plot drawing size style
'small': small plot size
'medium': medium plot size
'large': large plot size

Outputs

Examples

```
import scientimate as sm
import numpy as np

x=[0,1]
y=np.zeros((2,50))
y[0,:]=np.linspace(2,51,50)
y[1,:]=np.linspace(2,51,50)
sm.plot2d(x,y,'line','blue_red','large')

x=np.linspace(1,10,10)
y=np.zeros((10,2))
y[:,0]=1+np.random.rand(10)
y[:,1]=2+np.random.rand(10)
sm.plot2d(x,y,'line_confid','blue_red','large')

x=np.linspace(0,2*np.pi,1000)
x=np.tile(x[:,np.newaxis],(1,10))
s=np.arange(1,11)
s=np.tile(s[np.newaxis,:],(1000,1))
y=s+np.sin(1.0*np.pi*x)
sm.plot2d(x,y,'line','cool','large')

x=np.random.rand(100,3)
y=np.random.rand(100,3)
```

(continues on next page)

(continued from previous page)

```
y=np.zeros((100,3))
y[:,0]=1+2.0*x[:,0]+np.random.rand(100)
y[:,1]=3+2.0*x[:,1]+np.random.rand(100)
y[:,2]=5+2.0*x[:,2]+np.random.rand(100)
sm.plot2d(x,y,'scatter','seq','large')

x=np.random.rand(100)
y=np.random.rand(100)
sm.plot2d(x,y,'scatter_ascend','purple','large')

x=[[1,1,1],[2,2,2],[3,3,3],[4,4,4]]
y=[[2,3,8],[2,5,6],[5,7,9],[1,2,3]]
sm.plot2d(x,y,'bar','purple','medium')

x=[1,3,5,7,9,11,13,15]
y=[2,3,9,8,2,5,6,9]
sm.plot2d(x,y,'bar','purple','medium')

x=np.random.randn(1000)
sm.plot2d(x,[],'histogram','purple','medium')
```

References

Colormap

- https://matplotlib.org/tutorials/colors/colormaps.html
- https://www.mathworks.com/help/PYTHON/ref/colormap.html
- http://colorbrewer2.org
- http://matplotlib.org/cmocean/
- http://jdherman.github.io/colormap/

Color

- http://htmlcolorcodes.com

scientimate.plot2dsubplot

```
scientimate.plot2dsubplot(x, y=None, plottype='line', cmapcolors='seq', sizestyle=
↪'medium')
```

Description

Plot x and y data in 2-d subplots

Inputs

x x data as a 2-d array of (M,N)

y=[] y data as a 2-d array of (M,N)

plottype='line'

Plot type
'line': line plot
'line_grid': line plot with grid lines
'scatter': scatter plot
'scatter_grid': scatter plot with grid lines
'bar': bar plot
'bar_grid': bar plot with grid lines
'barh': horizontal bar plot
'barh_grid': horizontal bar plot with grid lines
'histogram': histogram plot
'histogram_grid': histogram plot with grid lines

cmapcolors='seq'

Colormap style
'blue': blue colormap
'red': red colormap
'green': green colormap
'yellow': yellow colormap
'purple': purple colormap
'brown': brown colormap
'gray': gray colormap
'blue_red': blue-red colormap
'red_blue': red-blue colormap
'blue_green': blue-green colormap
'green_blue': green-blue colormap
'green_yellow': green-yellow colormap
'yellow_green': yellow-green colormap
'red_yellow': red-yellow colormap
'yellow_red': yellow-red colormap
'cyclic': cyclic/oscillation colormap
'seq': sequential colormap
User-defined colors may be used to generate colormap
User-defined colors should be defined as (M,3) array in RGB color format
At least two colors should be defined, i.e. M should be equal or larger than 2
User-defined colors values should be between 0 and 255
Any available colormap name such as 'cool', 'winter', etc also can be used

sizestyle='medium'

Plot drawing size style
'small': small plot size
'medium': medium plot size
'large': large plot size

Outputs

Examples

```
import scientimate as sm
import numpy as np

x=np.random.rand(31,4)
sm.plot2dsubplot(x,[],'line','blue','medium')

x=np.random.rand(100,9)
sm.plot2dsubplot(x,[],'histogram','purple','medium')
```

References

Colormap

- https://matplotlib.org/tutorials/colors/colormaps.html
- https://www.mathworks.com/help/matlab/ref/colormap.html
- http://colorbrewer2.org
- http://matplotlib.org/cmocean/
- http://jdherman.github.io/colormap/

Color

- http://htmlcolorcodes.com

scientimate.plot2dtimeseries

```
scientimate.plot2dtimeseries(x, starttime='2000-10-20 00:00:00', endtime='2100-10-20␣
↪23:00:00', plottype='line', cmapcolors='blue', sizestyle='medium')
```

Description

Plot x data in 2-d timeseries

Inputs

x x data as a 2-d array of (M,N)

starttime='2000-10-20 00:00:00'

Start date and time for time series generation
Format: 'yyyy-mm-dd HH:MM:SS'

endtime='2100-10-20 23:00:00'

End date and time for time series generation
Format: 'yyyy-mm-dd HH:MM:SS'

plottype='bar'

Plot type

'line': line plot
'line_grid': line plot with grid lines
'line_subplot': line plot with subplots
'line_subplot_grid': line plot with subplots and grid lines
'bar': bar plot
'bar_grid': bar plot with grid lines
'barh': horizontal bar plot
'barh_grid': horizontal bar plot with grid lines

cmapcolors='seq'

Colormap style
'blue': blue colormap
'red': red colormap
'green': green colormap
'yellow': yellow colormap
'purple': purple colormap
'brown': brown colormap
'gray': gray colormap
'blue_red': blue-red colormap
'red_blue': red-blue colormap
'blue_green': blue-green colormap
'green_blue': green-blue colormap
'green_yellow': green-yellow colormap
'yellow_green': yellow-green colormap
'red_yellow': red-yellow colormap
'yellow_red': yellow-red colormap
'cyclic': cyclic/oscillation colormap
'seq': sequential colormap
User-defined colors may be used to generate colormap
User-defined colors should be defined as (M,3) array in RGB color format
At least two colors should be defined, i.e. M should be equal or larger than 2
User-defined colors values should be between 0 and 255
Any available colormap name such as 'cool', 'winter', etc also can be used

sizestyle='medium'

Plot drawing size style
'small': small plot size
'medium': medium plot size
'large': large plot size

Outputs

Examples

```
import scientimate as sm
import numpy as np
```

(continues on next page)

(continued from previous page)

```
x=np.random.rand(366)
starttime='2020-01-01 00:00:00'
endtime='2020-12-31 00:00:00'
sm.plot2dtimeseries(x,starttime,endtime,'line','purple','medium')

x=np.random.rand(31,3)
starttime='2020-01-01 00:00:00'
endtime='2020-01-31 00:00:00'
sm.plot2dtimeseries(x,starttime,endtime,'line_subplot','seq','medium')
```

References

Colormap

- https://matplotlib.org/tutorials/colors/colormaps.html
- https://www.mathworks.com/help/PYTHON/ref/colormap.html
- http://colorbrewer2.org
- http://matplotlib.org/cmocean/
- http://jdherman.github.io/colormap/

Color

- http://htmlcolorcodes.com

scientimate.plot3d

```
scientimate.plot3d(x, y, z, plottype='imagesc', cmapcolors='blue')
```

Description

Plot x , y, and z data in 2-d/3-d contour/surface plot

Inputs

x

x data
Set x=[] if it is not available
It may be 1d or 2d array

y

y data
Set y=[] if it is not available
It may be 1d or 2d array

z

z data
It may be 1d or 2d array

plottype='imagesc'

Plot type
'imagesc': 2 dimensional plot using imagesc or imshow
'pcolor': 2 dimensional plot using pcolor
'contour': 2 dimensional contour plot, number of contour=32
'contourf': 2 dimensional filled contour plot, number of contour=32
'surface': 3 dimensional surface plot

cmapcolors='blue'

Colormap style
'blue': blue colormap
'red': red colormap
'green': green colormap
'yellow': yellow colormap
'purple': purple colormap
'brown': brown colormap
'gray': gray colormap
'blue_red': blue-red colormap
'red_blue': red-blue colormap
'blue_green': blue-green colormap
'green_blue': green-blue colormap
'green_yellow': green-yellow colormap
'yellow_green': yellow-green colormap
'red_yellow': red-yellow colormap
'yellow_red': yellow-red colormap
'cyclic': cyclic/oscillation colormap
'seq': sequential colormap
User-defined colors may be used to generate colormap
User-defined colors should be defined as (M,3) array in RGB color format
At least two colors should be defined, i.e. M should be equal or larger than 2
User-defined colors values should be between 0 and 255
Any available colormap name such as 'cool', 'winter', etc also can be used

Outputs

Examples

```
import scientimate as sm
import numpy as np

x,y=np.meshgrid(np.linspace(-10,10,50),np.linspace(-10,10,50))
r=np.sqrt(x**2+y**2)+1e-10 #Add 1e-10 to prevent divide by 0
z=np.sin(r)/r
sm.plot3d(x,y,z,'surface','purple')

x,y=np.meshgrid(np.linspace(-10,10,21),np.linspace(-10,10,21))
z=(np.sin(x)+np.sin(y))/(x+y+1e-10) #Add 1e-10 to prevent divide by 0
sm.plot3d(x,y,z,'pcolor','purple')
```

(continues on next page)

(continued from previous page)

```
x=10*np.random.rand(1000)
y=10*np.random.rand(1000)
z=x**2+y**2
sm.plot3d(x,y,z,'pcolor','blue')
```

References

Colormap

- https://matplotlib.org/tutorials/colors/colormaps.html
- https://www.mathworks.com/help/PYTHON/ref/colormap.html
- http://colorbrewer2.org
- http://matplotlib.org/cmocean/
- http://jdherman.github.io/colormap/

Color

- http://htmlcolorcodes.com

1.3.6 Signal Processing

scientimate.filtertimeseries

```
xFiltered, t = scientimate.filtertimeseries(x, fs, fcL, fcH, dispout)
```

Description

Filter time-series to retain signals with frequencies of fcL <= f <= fcH
It assumes time-series is stationary

Inputs

x

Time-series

fs

Sampling frequency that data collected at in (Hz)
fs=1/dt where dt is time interval between two successive points

fcL=0

Low cut-off frequency, between 0*fs to 0.5*fs (Hz)
Signals with frequencies f<fcL are removed
Must be between 0 and fs/2

fcH=fs/2

High cut-off frequency, between 0*fs to 0.5*fs (Hz)
Signals with frequencies f>fcH are removed
Must be between 0 and fs/2

dispout='no' Define to display outputs or not ('yes': display, 'no': not display)

Outputs

xFiltered Filtered time-series

t Time

Examples

```
import scientimate as sm
import numpy as np

#Generate time-series from 3 waves with frequencies of 0.5, 2, 4 Hz
fs=32 #Sampling frequency
d=20 #Duration
f1=0.5 #1st wave frequency
f2=2 #2nd wave frequency
f3=4 #3rd wave frequency
a1=1 #1st wave amplitude
a2=0.2 #2nd wave amplitude
a3=0.1 #3rd wave amplitude
dt=1/fs #Time interval
t=np.linspace(0,d-dt,fs*d) #Time data points
x=a1*np.sin(2*np.pi*f1*t)+a2*np.sin(2*np.pi*f2*t)+a3*np.sin(2*np.pi*f3*t) #Time-series

#Keep only wave with f1 frequency and remove waves with f2 and f3 frequencies
fcL=f1-0.2 #Lower cut-off frequency
fcH=f1+0.2 #Higher cut-off frequency
xFiltered, time = sm.filtertimeseries(x, fs, fcL, fcH, 'yes')

#Keep waves with f2 and f3 frequencies and remove wave with f1 frequency
fcL=f2-0.2 #Lower cut-off frequency
fcH=f3+0.2 #Higher cut-off frequency
xFiltered, time = sm.filtertimeseries(x, fs, fcL, fcH, 'yes')
```

References

scientimate.smoothsignal

```
xSmoothed = scientimate.smoothsignal(x, WindowSize=33, method='moveavg', dispout='no')
```

Description

Smooth input data using a window function

Inputs

x Input data

WindowSize=33 Window size (number of adjacent elements)that is used for smoothing, should be equal or larger than 3

method='moveavg'

Smoothing method
'moveavg': moving average
'lowpass': Low-Pass filter
'savgol': Savitzky-Golay filter
'butter': Butterworth filter

dispout='no' Define to display outputs or not ('yes': display, 'no': not display)

Outputs

xSmoothed Smoothed data

Examples

```
import scientimate as sm
import numpy as np
from numpy import random

rng = np.random.default_rng()
x=np.sin(np.linspace(0,6*np.pi,200))+rng.random(200)
xSmoothed=sm.smoothsignal(x,33,'moveavg','yes')
```

References

1.3.7 Statistics

scientimate.curvefit2d

```
c, yPredicted = scientimate.curvefit2d(x, y, MathExpression, coefIniGuess, fitmethod=
↪'fmin', dispout='no')
```

Description

Fit curve to 2 dimensinal input dataset

Inputs

x x data

y y data

MathExpression

Right hand side of the relation y=f(x) as 'f(x)'
Example: y=c[0]*x**2+c[1]*x+c[2] then MathExpression='c[0]*x**2+c[1]*x+c[2]'
Example: y=c[0]*exp(x)+c[1] then MathExpression='c[0]*np.exp(x)+c[1]'
Example: y=c[0]*sin(x)+c[1] then MathExpression='c[0]*np.sin(x)+c[1]'
Desired coefficients to be found should be as : c[0], c[1],...,c[n]

coefIniGuess

Initial guess for desired coefficients to be found
coefIniGuess=[guess0,guess1,...,guessn]
guess0 is initial guess for c[0],...,guessn is initial guess for c[n]

fitmethod

Fitting method:
'lsq': curve-fitting using nonlinear least-squares
'fmin': curve-fitting by minimizing a sum of squared errors

dispout='no' Define to display outputs or not ('yes': display, 'no': not display)

Outputs

c Desired coefficients to be found

yPredicted Predicted value from fitted curve

Examples

```
import scientimate as sm
import numpy as np

x=np.linspace(0,10,100)
y=0.5*x**2+2*x+10+(-10+(10-(-10)))*np.random.rand(100)
c,yPredicted=sm.curvefit2d(x,y,'c[0]*x**2+c[1]*x+c[2]',[1,1,1],'fmin','yes')

x=np.linspace(0,10,100)
y=5*x**2+7+(-200+(200-(-200)))*np.random.rand(100)
c,yPredicted=sm.curvefit2d(x,y,'c[0]*x**c[1]+c[2]',[1,2,10],'lsq','yes')

x=np.linspace(0,10,100)
y=0.5*np.exp(x)+100+(-200+(200-(-200)))*np.random.rand(100)
c,yPredicted=sm.curvefit2d(x,y,'c[0]*np.exp(x)+c[1]',[1,1],'fmin','yes')
```

References

scientimate.dataoverview

```
scientimate.dataoverview(x, y=[], z=[])
```

Description

Display an overview of the input data

Inputs

x x data

y=[] y data (Optional)

z=[] z data (Optional)

Outputs

Examples

```
import scientimate as sm
import numpy as np

x=(-0.1+(0.1-(-0.1)))*np.random.randn(1024*2)
sm.dataoverview(x)

x=(-0.1+(0.1-(-0.1)))*np.random.randn(1024*2)
y=x+(-0.01+(0.01-(-0.01)))*np.random.randn(1024*2)
sm.dataoverview(x,y)

x=(-0.1+(0.1-(-0.1)))*np.random.randn(1024*2)
y=(-0.2+(0.2-(-0.2)))*np.random.randn(1024*2)
z=x**2+y**2+0.1+(-0.1+(0.1-(-0.1)))*np.random.rand(1024*2)
sm.dataoverview(x,y,z)
```

References

- https://docs.python.org/3/library/string.html

scientimate.findextremum

```
xmin, ymin, xmax, ymax = scientimate.findextremum(x, y, winlen=3, dispout='no')
```

Description

Find local extremum (minimum and maximum) in data

Inputs

x x data

y y data

winlen=3

Window length, defines a number of points in sliding window used for defining maximum and minimum
Example: winlen=5 means two points on each side of each data point is used in calculation
Using a larger value for winlen makes it less sensitive
winlen should be an odd number equal or larger than 3

dispout='no' Define to display outputs or not ('yes': display, 'no': not display)

Outputs

xmin x of minmum points

ymin y of minmum points

xmax x of maximum points

ymax y of maximum points

Examples

```
import scientimate as sm
import numpy as np

x=np.linspace(0,30,1000)
y=2*np.exp(-0.1*2*np.pi/5*x)*np.sin(np.sqrt(1-0.1**2)*2*np.pi/5*x)
xmin,ymin,xmax,ymax=sm.findextremum(x,y,3,'yes')

x=np.linspace(0,50,1000)
y=np.sin(x)+0.1*np.random.rand(1000)
xmin,ymin,xmax,ymax=sm.findextremum(x,y,15,'yes')
```

References

scientimate.findknn

```
indxknn, distknn = scientimate.findknn(x, y, xpoint, ypoint, numofneighbors=1,␣
↪distCalcMethod='1d', dispout='no')
```

Description

Find k-nearest neighbors using Euclidean distance

Inputs

x Coordinate of data in x direction

y Coordinate of data in y direction

xpoint Coordinate (in x direction) of point that nearest point to that is disired to be found

ypoint Coordinate (in y direction) of point that nearest point to that is disired to be found

numofneighbors=1 Number of nearest neighbors to (xpoint,ypoint) that are desired to be found

distCalcMethod='1d'

Distance calculation method
'1d': use 1d array
'pdist2': Use 2d distance function
'vector': Use vectorized distance

dispout='no' Define to display outputs or not ('yes': display, 'no': not display)

Outputs

indxknn

Index of nearest neighbors points
returns M*N array where M=length(xpoint) and N=numofneighbors
nth row associated with nth point in (xpoint,ypoint)

distknn

Distance of nearest neighbors points
returns M*N array where M=length(xpoint) and N=numofneighbors
mth row associated with mth point in (xpoint,ypoint)
nth column associated with nth nearest neighbors

Examples

```
import scientimate as sm
import numpy as np

x=10*np.random.rand(100)
y=10*np.random.rand(100)
xpoint=np.mean(x)
ypoint=np.mean(y)
indxknn,distknn=sm.findknn(x,y,xpoint,ypoint,1,'1d','yes')

x=10*np.random.rand(100)
y=10*np.random.rand(100)
xpoint=[2.5,5,7.5]
ypoint=[3,6,9]
indxknn,distknn=sm.findknn(x,y,xpoint,ypoint,10,'1d','yes')
```

References

scientimate.fitgoodness

```
r, R2, RMSE, MAE, SI, NSE, d, Bias, NMBias, RE = scientimate.fitgoodness(x, y,␣
↪dispout='no')
```

Description

Calculate goodness of fit parameters

Inputs

x Dataset with true (exact or expected) values, such as theoretical values

y

Dataset that needs to be evaluated, such as model results or estimated values
Accuracy of y dataset is evaluated against x dataset

dispout='no' Define to display outputs or not ('yes': display, 'no': not display)

Outputs

r Pearson correlation coefficient

R2 Coefficient of determination

RMSE Root mean square error

MAE Mean absolute error

SI Scatter index

NSE Nash Sutcliffe efficiency coefficient

d Index of agreement

Bias Bias

NMBias Normalized mean bias

RE Relative error

Examples

```
import scientimate as sm
import numpy as np

x=(-0.1+(0.1-(-0.1)))*np.random.randn(1024*2)
y=x+(-0.01+(0.01-(-0.01)))*np.random.randn(1024*2)
r,R2,RMSE,MAE,SI,NSE,d,Bias,NMBias,RE=sm.fitgoodness(x,y,'yes')

x=[1,2,3,4,5,6,7,8,9,10]
y=[1.1,1.98,3.3,4.2,4.8,5.95,7.5,7.7,8.99,10.5]
r,R2,RMSE,MAE,SI,NSE,d,Bias,NMBias,RE=sm.fitgoodness(x,y,'yes')
```

References

scientimate.movingwindow

```
moving_statistics = scientimate.movingwindow(x, WindowSize=3, StatisticalMethod='mean
↪', dispout='no')
```

Description

Calculate statistics of moving window through 1-d x data

Inputs

x Input data

WindowSize=3

Window size (number of adjacent elements) that is used for moving window, should be equal or larger than 3
Window size should be an odd integer

StatisticalMethod='mean'

Statistical value of moving window to be reported:
'mean': Moving mean
'std': Moving standard deviation
'min': Moving minimum
'max': Moving maximum
'sum': Moving sum

dispout='no' Define to display outputs or not ('yes': display, 'no': not display)

Outputs

moving_statistics Statistical value of moving window

Examples

```
import scientimate as sm
import numpy as np

fs=128
t=np.linspace(0,9.5,10*fs)
x=np.sin(2*np.pi*0.3*t)+0.1*np.sin(2*np.pi*4*t)
moving_statistics=sm.movingwindow(x,37,'mean','yes')
```

References

scientimate.probability1d

```
fdensity, fdensitycumulative, bincenter, xmean, xstd = scientimate.probability1d(x,␣
↪binedge=None, dispout='no')
```

Description

Calculate 1D probability density distribution for a given dataset

Inputs

x Input data

binedge

Bin edges
length(binedge)=number of bin +1
If there are N bins in histogram/distribution, then values in binedge are as:
1st value: left edge of 1st bin, 2nd value: left edge of 2nd bin, . . .
(N)th value: left edge of last bin, (N+1)th value: right edge of last bin

dispout='no' Define to display outputs or not ('yes': display, 'no': not display)

Outputs

fdensity Probability density distribution

fdensitycumulative Cumulative probability density distribution

bincenter Bin center

xmean Mean value of input data

xstd Standard deviation of input data

Examples

```
import scientimate as sm
import numpy as np

x=(-0.1+(0.1-(-0.1)))*np.random.randn(1024*2)
binedge=np.linspace(min(x),max(x),11)
fdensity,fdensitycumulative,bincenter,xmean,xstd=sm.probability1d(x,binedge,'yes')
```

References

scientimate.similaritymeasure

```
d = scientimate.similaritymeasure(x, y, CalcMethod='euclidean', dispout='no')
```

Description

Measure similarity between two arrays

Inputs

x First array, its similarity is measured against y array

y Second array, its similarity is measured against x array

CalcMethod='euclidean'

Similarity calculation method
'euclidean': Euclidean distance
'manhattan': Manhattan distance
'minkowski': Minkowski distance (power=3)
'cosine': Cosine distance
'pearson': Pearson's correlation coefficient
'spearman': spearman's correlation coefficient
'norm': Absolute difference of norm
'covariance': Covariance
'inv_covariance': Euclidean distance of inverse covariance
'histogram': Mean of absolute difference of histogram
t-test': Two-sample t-test statistic

dispout='no' Define to display outputs or not ('yes': display, 'no': not display)

Outputs

d Arrays similarity measure

Examples

```
import scientimate as sm
import numpy as np

x=[0,2,4,6]
y=[2,3,5,7]
d=sm.similaritymeasure(x,y,'euclidean','yes')

x=[1,2,3,4,5,6,7,8,9,10]
y=[1.1,1.98,3.3,4.2,4.8,5.95,7.5,7.7,8.99,10.5]
d=sm.similaritymeasure(x,y,'pearson','yes')
```

References

Kianimajd, A., Ruano, M. G., Carvalho, P., Henriques, J., Rocha, T., Paredes, S., & Ruano, A. E. (2017). Comparison of different methods of measuring similarity in physiologic time series. IFAC-PapersOnLine, 50(1), 11005-11010.

- https://en.wikipedia.org/wiki/Similarity_measure
- https://en.wikipedia.org/wiki/Goodness_of_fit
- https://dataaspirant.com/2015/04/11/five-most-popular-similarity-measures-implementation-in-python/
- https://towardsdatascience.com/similarity-measures-e3dbd4e58660
- https://www.mathworks.com/matlabcentral/answers/377944-how-to-calculate-a-percentage-of-similarity-between-two-arrays

- https://en.wikipedia.org/wiki/Template_matching
- https://www.mathworks.com/help/images/ref/normxcorr2.html

1.3.8 Swan

scientimate.swandepthgrid

```
swandepth, swangrid, ncellx, ncelly, ngridx, ngridy = scientimate.swandepthgrid(xgrid,
↪ ygrid, zgrid, zmin=None, zmax=None, nanreplacement='no', savedata='no',␣
↪xyoutfilename='swangrid.xy', zoutfilename='swandepth.dep', outfilelocation=None,␣
↪dispout='no')
```

Description

Generate SWAN depth file and its associated x-y grid file

Inputs

xgrid x (longitude) of grid points as a [M*N] array

ygrid y (latitude) of grid points as a [M*N] array

zgrid

z (elevation) of grid points as a [M*N] array
z>0 is land, z<0 is water, z=0 is water surface

zmin=nanmin(zgrid)

Minimum z (elevation) to be considered
All z<zmin would be set to NaN

zmax=nanmax(zgrid)

Maximum z (elevation) to be considered
All z>zmax would be set to NaN
Example: to remove all land values from z data, set zmax=0

nanreplacement='no'

Replace NaN values with nanreplacement
By seting up an exception value equal to nanreplacement in SWAN input file, SWAN disregards these data points
If nanreplacement='no', then it does not replace NaN values
Example, nanreplacement=-999 will replace all NaN values with -999
Then if -999 is set as an exception in SWAN input file, SWAN disregards all -999 values

savedata='no'

Define if save data in a file or not
'no': does not save
'yes': save data as ascii file

xyoutfilename='swangrid.xy'

Name of output grid file between ‘ ‘ mark, example: ‘swangrid.xy’
xyoutfilename should have ‘.xy’ extension

zoutfilename=’swandepth.dep’

Name of output depth file between ‘ ‘ mark, example: ‘swandepth.dep’
zoutfilename should have ‘.dep’ extension

outfilelocation=pwd Location of output file between ‘ ‘ mark, example: ‘C:’ in MATLAB, or ‘C:/’ in Python

dispout=’no’ Define to display outputs or not (‘yes’: display, ‘no’: not display)

Outputs

swandepth

Depth data formated for SWAN
Note: SWAN (Delft3D) needs depth data not bathymetry data.
It means value below water surface should be positive and above surface negative.
Note: All NaN values are replaced with nanreplacement
Set up an exception value equal to nanreplacement in SWAN to disregard these data points

swangrid Grid data formated for SWAN

ncellx

Number of cells in x direction (ncellx=ngridx-1)
In SWAN, ncellx is equal to a number of meshes in computational grid in x direction

ncelly

Number of cells in y direction (ncelly=ngridy-1)
In SWAN, ncelly is equal to a number of meshes in computational grid in y direction

ngridx Number of grid points in x direction

ngridy Number of grid points in y direction

Examples

```
import scientimate as sm
import numpy as np
import scipy as sp
from scipy import interpolate

x=10.*np.random.rand(1000)
y=10*np.random.rand(1000)
z=x**2+y**2-np.mean(x**2+y**2)
xgrid,ygrid=np.meshgrid(np.linspace(np.min(x),np.max(x),100),np.linspace(np.min(y),np.
↪max(y),100))
zgrid=sp.interpolate.griddata((x,y),z,(xgrid,ygrid))
zmin=np.nanmin(zgrid)
zmax=np.nanmax(zgrid)
nanreplacement=-999
savedata='no'
xyoutfilename='swangrid.xy'
zoutfilename='swandepth.dep'
```

(continues on next page)

(continued from previous page)

```
outfilelocation=None
swandepth,swangrid,ncellx,ncelly,ngridx,ngridy=sm.swandepthgrid(xgrid,ygrid,zgrid,
↪zmin,zmax,nanreplacement,savedata,xyoutfilename,zoutfilename,outfilelocation,'yes')
```

References

Booij, N. R. R. C., Ris, R. C., & Holthuijsen, L. H. (1999). A third-generation wave model for coastal regions: 1. Model description and validation. Journal of geophysical research: Oceans, 104(C4), 7649-7666.

SWAN Team. (2007). S WAN user manual. Delft University of Technology. The Netherlands.

scientimate.swanvectorvarspconst

```
swanvectorvariable = scientimate.swanvectorvarspconst(Vx, Vy, savedata='no',␣
↪outfilename='swanwind.wnd', outfilelocation=None)
```

Description

Generate SWAN file for spatially constant vector variable

Inputs

Vx

Variable in x direction (x component of input variable)
If size of Vx>1, then it is considered as a time series
1st element is 1st time step, 2nd element is 2nd time step, . . .

Vy

Variable in y direction (y component of input variable)
Should have a same size as Vx

savedata='no'

Define if save data in a file or not
'no': does not save
'yes': save data as ascii file

outfilename='swanwind.wnd'

Name of output file between ' ' mark, example: 'swanwind.wnd'
outfilename should have proper name and extension

outfilelocation=pwd Location of output file between ' ' mark, example: 'C:' in MATLAB, or 'C:/' in Python

Outputs

swanvectorvariable

Spatially constant vector variable formated for SWAN

Note: Vector variable at each time step is assigned into 4 points,
assuming the vector variable domain is defined by 4 points, one at each corner

Examples

```
import scientimate as sm

windvelx=[10.5,10.6,10.55] #Data for 3 time steps
windvely=[2.5,2.6,2.55] #Data for 3 time steps
savedata='no'
outfilename='swanwind.wnd'
outfilelocation=None
swanvectorvariable=sm.swanvectorvarspconst(windvelx,windvely,savedata,outfilename,
↪outfilelocation)
```

References

Booij, N. R. R. C., Ris, R. C., & Holthuijsen, L. H. (1999). A third-generation wave model for coastal regions: 1. Model description and validation. Journal of geophysical research: Oceans, 104(C4), 7649-7666.

SWAN Team. (2007). S WAN user manual. Delft University of Technology. The Netherlands.

scientimate.swanwaterlevelspconst

```
swanwaterlevel = scientimate.swanwaterlevelspconst(waterlevel, savedata='no',␣
↪outfilename='swanwaterlevel.wl', outfilelocation=None)
```

Description

Generate SWAN water level file for spatially constant water level
This function can be used for any other scalar variable as well

Inputs

waterlevel

Water level (or any scalar variable)
If size of waterlevel>1, then it is considered as a time series
1st element is 1st time step, 2nd element is 2nd time step, . . .

savedata='no'

Define if save data in a file or not
'no': does not save
'yes': save data as ascii file

outfilename='swanwaterlevel.wl'

Name of output file between ' ' mark, example: 'swanwaterlevel.wl'
outfilename should have '.wl' extension

In case of using other scalar varable than water level, use proper name and extension

outfilelocation=pwd Location of output file between ‘ ‘ mark, example: ‘C:’ in MATLAB, or ‘C:/’ in Python

Outputs

swanwaterlevel

Spatially constant water level data (or any scalar variable) formated for SWAN
Note: Water level at each time step is assigned into 4 points,
assuming the water level domain is defined by 4 points, one at each corner

Examples

```
import scientimate as sm

waterlevel=[0.5,0.6,0.55] #Data for 3 time steps
savedata='no'
outfilename='swanwaterlevel.wl'
outfilelocation=None
swanwaterlevel=sm.swanwaterlevelspconst(waterlevel,savedata,outfilename,
↪outfilelocation)
```

References

Booij, N. R. R. C., Ris, R. C., & Holthuijsen, L. H. (1999). A third-generation wave model for coastal regions: 1. Model description and validation. Journal of geophysical research: Oceans, 104(C4), 7649-7666.

SWAN Team. (2007). S WAN user manual. Delft University of Technology. The Netherlands.

scientimate.swanwindspconst

```
swanwind = scientimate.swanwindspconst(windvel, winddir, winddirtype='mete',␣
↪windvelmin=0, savedata='no', outfilename='swanwind.wnd', outfilelocation=None)
```

Description

Generate SWAN wind file for spatially constant wind

Inputs

windvel

Wind velocity
If size of windvel>1, then it is considered as a time series
1st element is 1st time step, 2nd element is 2nd time step, . . .

winddir Wind direction in (Degree)

winddirtype=’mete’

Define wind direction type
'mete': meteorological wind direction
Meteorological direction represents a direction wind comes from and is measured counter-clockwise from the North
0 (degree): from North, 90 (degree): from East, 180 (degree): from South, 270 (degree): from West
'trig': trigonometric wind direction

windvelmin=0 Minimum allowed wind velocity

savedata='no'

Define if save data in a file or not
'no': does not save
'yes': save data as ascii file

outfilename='swanwind.wnd'

Name of output file between ' ' mark, example: 'swanwind.wnd'
outfilename should have '.wnd' extension

outfilelocation=pwd Location of output file between ' ' mark, example: 'C:' in MATLAB, or 'C:/' in Python

Outputs

swanwind

Spatially constant wind data formated for SWAN
Note: Wind at each time step is assigned into 4 points,
assuming the wind domain is defined by 4 points, one at each corner

Examples

```
import scientimate as sm

windvel=[10.5,10.6,10.55] #Data for 3 time steps
winddir=[30,32,28] #Data for 3 time steps
winddirtype='mete'
windvelmin=2.5
savedata='no'
outfilename='swanwind.wnd'
outfilelocation=None
swanwind=sm.swanwindspconst(windvel,winddir,winddirtype,windvelmin,savedata,
↪outfilename,outfilelocation)
```

References

Booij, N. R. R. C., Ris, R. C., & Holthuijsen, L. H. (1999). A third-generation wave model for coastal regions: 1. Model description and validation. Journal of geophysical research: Oceans, 104(C4), 7649-7666.

SWAN Team. (2007). S WAN user manual. Delft University of Technology. The Netherlands.

1.3.9 Water Wave Data Analysis

scientimate.diagnostictail

```
fOut, SyyOut, Hm0, fp, Tp, Tm01, Tm02 = scientimate.diagnostictail(fIn, SyyIn, ftail,␣
↪tailtype='jonswap', tailpower=-5, h=0, transfCalcMethod='approx', kCalcMethod='beji
↪', dispout='no')
```

Description

Replace a spectrum tail with JONSWAP (Hasselmann et al., 1973) or TMA Spectrum (Bouws et al., 1985)

Inputs

fIn Frequency (Hz), Input

SyyIn Power spectral density (m^2/Hz), Input

ftail Frequency that diagnostic tail apply after that (typically: ftail=2.5fm where fm=1/Tm01)

tailtype='jonswap'

Define type of the diagnostic tail to be applied
'jonswap': JONSWAP Spectrum tail, 'tma': TMA Spectrum tail

tailpower=-5

Tail power that diagnostic tail apply based on that
tailpower=-3 for shallow water, tailpower=-5 for deep water

h=0 Mean water depth in (m)

transfCalcMethod='approx'

Transformation function from JONSWAP into TMA calculation method
'approx': approximated method, 'tucker': Tucker (1994), 'kitaigordskii': Kitaigordskii et al. (1975)

kCalcMethod='beji'

Wave number calculation method
'hunt': Hunt (1979), 'beji': Beji (2013), 'vatankhah': Vatankhah and Aghashariatmadari (2013)
'goda': Goda (2010), 'exact': calculate exact value

dispout='no' Define to display outputs or not ('yes': display, 'no': not display)

Outputs

fOut Frequency (Hz), Output

SyyOut Power spectral density (m^2/Hz) with diagnostic tail, Output

Hm0 Zero-Moment Wave Height (m)

fp Peak wave frequency (Hz)

Tp Peak wave period (second)

Tm01 Wave Period from m01 (second), Mean Wave Period

Tm02 Wave Period from m02 (second), Mean Zero Crossing Period

Examples

```
import scientimate as sm
import numpy as np

N=2**11 #Total number of points
fs=8 #Sampling frequency
df=fs/N #Frequency difference
f=np.arange(0,fs/2+df,df) #Frequency vector
f[0]=f[1]/2 #Assign temporarily non-zero value to fisrt element of f to prevent␣
↪division by zero
Syy=0.016*9.81**2/((2*np.pi)**4*(f**5))*np.exp(-1.25*(0.33/f)**4) #Calculating␣
↪Spectrum
f[0]=0
Syy[0]=0

fOut,SyyOut,Hm0,fp,Tp,Tm01,Tm02=sm.diagnostictail(f,Syy,0.5,'jonswap',-5,5,'approx',
↪'beji','yes')

fOut,SyyOut,Hm0,fp,Tp,Tm01,Tm02=sm.diagnostictail(f,Syy,0.5,'tma',-3,5,'approx','beji
↪','yes')
```

References

Beji, S. (2013). Improved explicit approximation of linear dispersion relationship for gravity waves. Coastal Engineering, 73, 11-12.

Bouws, E.; Günther, H.; Rosenthal, W., and Vincent, C.L., (1985). Similarity of the wind wave spectrum in finite depth water: 1. Spectral form. Journal of Geophysical Research: Oceans, 90(C1), 975-986.

Goda, Y. (2010). Random seas and design of maritime structures. World scientific.

Hasselmann, K.; Barnett, T. P.; Bouws, E.; Carlson, H.; Cartwright, D. E.; Enke, K.; Ewing, J. A.; Gienapp, H.; Hasselmann, D. E.; Kruseman, P.; Meerbrug, A.; Muller, P.; Olbers, D. J.; Richter, K.; Sell, W., and Walden, H., (1973). Measurements of wind-wave growth and swell decay during the Joint North Sea Wave Project (JONSWAP). Deutsche Hydrographische Zeitschrift A80(12), 95p.

Hunt, J. N. (1979). Direct solution of wave dispersion equation. Journal of the Waterway Port Coastal and Ocean Division, 105(4), 457-459.

Vatankhah, A. R., & Aghashariatmadari, Z. (2013). Improved explicit approximation of linear dispersion relationship for gravity waves: A discussion. Coastal engineering, 78, 21-22.

scientimate.seaswell1d

```
fseparation, Hm0, Hm0sea, Hm0swell, Tp, Tpsea, Tpswell, m0, m0sea, m0swell, fp, Tm01,␣
↪Tm02 = scientimate.seaswell1d(f, Syy, fsepCalcMethod='hwang', fu=0.5, fmaxswell=0.2,
↪ fpminswell=0, Windvel=10, dispout='no')
```

Description

Partition (separate) wind sea from swell in a power spectral density using an one dimensional method

Inputs

f Frequency (Hz)

Syy Power spectral density (m^2/Hz)

fsepCalcMethod='hwang'

Wind sea swell separating calculation method
'celerity': using deep water wave celerity, 'gilhousen': Gilhousen and Hervey (2001),
'hwang': Hwang et al. (2012), 'exact': calculate exact value

fu=0.5 An upper bound of a spectrum integration frequency (Hz)

fmaxswell=0.25 Maximum frequency that swell can have, It is about 0.2 in Gulf of Mexico

fpminswell=0.1 A lower bound of a spectrum (minimum frequency) that is used for Tpswell calculation

Windvel=10 Wind velocity (m/s), only required for Gilhousen and Hervey (2001) method

dispout='no' Define to display outputs or not ('yes': display, 'no': not display)

Outputs

fseparation Wind sea and Swell Separation Frequency (Hz)

Hm0 Zero-Moment Wave Height (m)

Hm0sea Sea Zero-Moment Wave Height (m)

Hm0swell Swell Zero-Moment Wave Height (m)

Tp Peak wave period (second)

Tpsea Peak Sea period (second)

Tpswell Peak Swell Period (second)

m0 Zero-Moment of the power spectral density (m^2)

m0sea Zero-Moment of the wind sea spectrum (m^2)

m0swell Zero-Moment of the swell spectrum (m^2)

fp Peak wave frequency (Hz)

Tm01 Wave Period from m01 (second), Mean Wave Period

Tm02 Wave Period from m02 (second), Mean Zero Crossing Period

Examples

```
import scientimate as sm
import numpy as np

N=2**11 #Total number of points
fs=2 #Sampling frequency
df=fs/N #Frequency difference
f=np.arange(0,fs/2+df,df) #Frequency vector
f[0]=f[1]/2 #Assign temporarily non-zero value to fisrt element of f to prevent␣
↪division by zero
SyySea=0.016*9.81**2/((2*np.pi)**4*(f**5))*np.exp(-1.25*(0.33/f)**4) #Calculating␣
↪Spectrum
SyySwell=0.016*9.81**2/((2*np.pi)**4*(f**5))*np.exp(-1.25*(0.15/f)**4)*0.005
↪#Calculating Spectrum
Syy=SyySea+SyySwell
f[0]=0
Syy[0]=0
fseparation,Hm0,Hm0sea,Hm0swell,Tp,Tpsea,Tpswell,m0,m0sea,m0swell,fp,Tm01,Tm02=sm.
↪seaswell1d(f,Syy,'exact',0.5,0.3,0,10,'yes')
```

References

Gilhousen, D. B., & Hervey, R. (2002). Improved estimates of swell from moored buoys. In Ocean Wave Measurement and Analysis (2001) (pp. 387-393).

Hwang, P. A., Ocampo-Torres, F. J., & García-Nava, H. (2012). Wind sea and swell separation of 1D wave spectrum by a spectrum integration method. Journal of Atmospheric and Oceanic Technology, 29(1), 116-128.

scientimate.wavefrompressurepsd

```
Hm0, fp, Tp, Tm01, Tm02, f, Syy, m0, ftail = scientimate.wavefrompressurepsd(P, fs, h,
↪ heightfrombed=0, fmaxpcorr=None, fminpcorr=0, fcL=0, fcH=None, fmaxpcorrCalcMethod=
↪'auto', Kpafterfmaxpcorr='constant', kCalcMethod='beji', Rho=1000, nfft=None,␣
↪SegmentSize=256, OverlapSize=128, dispout='no')
```

Description

Calculate wave properties from water pressure by converting it to water surface elevation power spectral density

Inputs

P Water pressure time series data in (N/m^2)

fs Sampling frequency that data collected at in (Hz)

h Water depth in (m)

heightfrombed=0 Height from bed that data collected at in (m)

fmaxpcorr=fs/2

Maximum frequency that a pressure attenuation factor applies up on that (Hz)
If fmaxpcorrCalcMethod='user', then the smaller of calculated and user defined fmaxpcorr will be chosen

fminpcorr=0

Minimum frequency that is used for defining fmaxpcorr if fmaxpcorrCalcMethod='auto' (Hz)
fminpcorr should be smaller than fp
If swell energy exists, fminpcorr should be smaller than fp of wind sea (fpsea) and larger than fp of swell (fpswell) if there swell

fcL=0 Low cut-off frequency, between 0*fs to 0.5*fs (Hz)

fcH=fs/2 High cut-off frequency, between 0*fs to 0.5*fs (Hz)

fmaxpcorrCalcMethod='auto'

Define if to calculate fmaxpcorr and ftail or to use user defined
'user': use user defined value for fmaxpcorr
'auto': automatically define value for fmaxpcorr

Kpafterfmaxpcorr='constant'

Define a apressure response factor, Kp, value for frequency larger than fmaxpcorr
'nochange': Kp is not changed for frequency larger than fKuvmin
'one': Kp=1 for frequency larger than fmaxpcorr
'constant': Kp for f larger than fmaxpcorr stays equal to Kp at fmaxpcorr (constant)

kCalcMethod='beji'

Wave number calculation method
'hunt': Hunt (1979), 'beji': Beji (2013), 'vatankhah': Vatankhah and Aghashariatmadari (2013)
'goda': Goda (2010), 'exact': calculate exact value

Rho=1000 Water density (kg/m^3)

nfft=length(Eta) Total number of points between 0 and fs that spectrum reports at is (nfft+1)

SegmentSize=256 Segment size, data are divided into the segments each has a total element equal to SegmentSize

OverlapSize=128 Number of data points that are overlapped with data in previous segments

dispout='no' Define to display outputs or not ('yes': display, 'no': not display)

Outputs

Hm0 Zero-Moment Wave Height (m)

fp Peak wave frequency (Hz)

Tp Peak wave period (second)

Tm01 Wave Period from m01 (second), Mean Wave Period

Tm02 Wave Period from m02 (second), Mean Zero Crossing Period

f Frequency (Hz)

Syy Power spectral density (m^2/Hz)

m0 Zero-Moment of the power spectral density (m^2)

ftail Frequency that diagnostic tail apply after that (typically: ftail=2.5fm where fm=1/Tm01)

Examples

```
import scientimate as sm
import numpy as np
import scipy as sp
from scipy import signal

fs=2 #Sampling frequency
duration=1024 #Duration of the data
N=fs*duration #Total number of points
df=fs/N #Frequency difference
dt=1/fs #Time difference, dt=1/fs
t=np.linspace(0,duration-dt,N) #Time
Eta=sp.signal.detrend(0.5*np.cos(2*np.pi*0.2*t)+(-0.1+(0.1-(-0.1)))*np.random.rand(N))
hfrombed=4
h=5
k=0.2
P=Eta*9.81*1000*(np.cosh(k*hfrombed)/np.cosh(k*h))
Hm0,fp,Tp,Tm01,Tm02,f,Syy,m0,ftail=sm.wavefrompressurepsd(P,fs,5,4,0.7,0.15,0,fs/2,
↪'auto','constant','beji',1025,N,256,128,'yes')
```

References

Beji, S. (2013). Improved explicit approximation of linear dispersion relationship for gravity waves. Coastal Engineering, 73, 11-12.

Goda, Y. (2010). Random seas and design of maritime structures. World scientific.

Hunt, J. N. (1979). Direct solution of wave dispersion equation. Journal of the Waterway Port Coastal and Ocean Division, 105(4), 457-459.

Vatankhah, A. R., & Aghashariatmadari, Z. (2013). Improved explicit approximation of linear dispersion relationship for gravity waves: A discussion. Coastal engineering, 78, 21-22.

Welch, P. (1967). The use of fast Fourier transform for the estimation of power spectra: a method based on time averaging over short, modified periodograms. IEEE Transactions on audio and electroacoustics, 15(2), 70-73.

scientimate.wavefrompressurezcross

```
Hs, Ts, Hz, Tz, Hrms, H, T, Eta, t = scientimate.wavefrompressurezcross(P, fs, h,␣
↪heightfrombed=0, Kpmin=0.15, kCalcMethod='beji', Rho=1000, dispout='no')
```

Description

Calculate wave properties from water pressure by using an upward zero crossing method

Inputs

P Water pressure time series data in (N/m^2)

fs Sampling frequency that data collected at in (Hz)

h Water depth in (m)

heightfrombed=0 Height from bed that data collected at in (m)

Kpmin=0.15

Minimum acceptable value for a pressure response factor
If Kpmin=0.15, it avoid wave amplification larger than 6 times (1/0.15)

kCalcMethod='beji'

Wave number calculation method
'hunt': Hunt (1979), 'beji': Beji (2013), 'vatankhah': Vatankhah and Aghashariatmadari (2013)
'goda': Goda (2010), 'exact': calculate exact value

Rho=1000 Water density (kg/m^3)

dispout='no' Define to display outputs or not ('yes': display, 'no': not display)

Outputs

Hs Significant Wave Height (m)

Ts Significant Wave Period (second)

Hz Zero Crossing Mean Wave Height (m)

Tz Zero Crossing Mean Wave Period (second)

Hrms Root Mean Square Wave Height (m)

H Wave Height Data Series array (m)

T Wave Period Data Series array (second)

Eta Water surface elevation time series in (m)

t Time (s)

Examples

```
import scientimate as sm
import numpy as np
import scipy as sp
from scipy import signal

fs=2 #Sampling frequency
duration=1024 #Duration of the data
N=fs*duration #Total number of points
df=fs/N #Frequency difference
dt=1/fs #Time difference, dt=1/fs
t=np.linspace(0,duration-dt,N) #Time
Eta=sp.signal.detrend(0.5*np.cos(2*np.pi*0.2*t)+(-0.1+(0.1-(-0.1)))*np.random.rand(N))
hfrombed=4
h=5
k=0.2
P=Eta*9.81*1000*(np.cosh(k*hfrombed)/np.cosh(k*h))
Hs,Ts,Hz,Tz,Hrms,H,T,Eta,t=sm.wavefrompressurezcross(P,fs,5,4,0.15,'beji',1025,'yes')
```

References

Beji, S. (2013). Improved explicit approximation of linear dispersion relationship for gravity waves. Coastal Engineering, 73, 11-12.

Goda, Y. (2010). Random seas and design of maritime structures. World scientific.

Hunt, J. N. (1979). Direct solution of wave dispersion equation. Journal of the Waterway Port Coastal and Ocean Division, 105(4), 457-459.

Vatankhah, A. R., & Aghashariatmadari, Z. (2013). Improved explicit approximation of linear dispersion relationship for gravity waves: A discussion. Coastal engineering, 78, 21-22.

scientimate.wavefromsurfaceelevpsd

```
Hm0, fp, Tp, Tm01, Tm02, f, Syy, m0 = scientimate.wavefromsurfaceelevpsd(Eta, fs,␣
↪fcL=0, fcH=None, nfft=None, SegmentSize=256, OverlapSize=128, dispout='no')
```

Description

Calculate wave properties from water surface elevation power spectral density

Inputs

Eta Water surface elevation time series data in (m)

fs Sampling frequency that data collected at in (Hz)

fcL=0 Low cut-off frequency, between 0*fs to 0.5*fs (Hz)

fcH=fs/2 High cut-off frequency, between 0*fs to 0.5*fs (Hz)

nfft=length(Eta) Total number of points between 0 and fs that spectrum reports at is (nfft+1)

SegmentSize=256 Segment size, data are divided into the segments each has a total element equal to SegmentSize

OverlapSize=128 Number of data points that are overlaped with data in previous segments

dispout='no' Define to display outputs or not ('yes': display, 'no': not display)

Outputs

Hm0 Zero-Moment Wave Height (m)

fp Peak wave frequency (Hz)

Tp Peak wave period (second)

Tm01 Wave Period from m01 (second), Mean Wave Period

Tm02 Wave Period from m02 (second), Mean Zero Crossing Period

f Frequency (Hz)

Syy Power spectral density (m^2/Hz)

m0 Zero-Moment of the power spectral density (m^2)

Examples

```
import scientimate as sm
import numpy as np
import scipy as sp
from scipy import signal

fs=2 #Sampling frequency
duration=1024 #Duration of the data
N=fs*duration #Total number of points
df=fs/N #Frequency difference
dt=1/fs #Time difference, dt=1/fs
t=np.linspace(0,duration-dt,N) #Time
Eta=sp.signal.detrend(0.5*np.cos(2*np.pi*0.2*t)+(-0.1+(0.1-(-0.1)))*np.random.rand(N))
Hm0,fp,Tp,Tm01,Tm02,f,Syy,m0=sm.wavefromsurfaceelevpsd(Eta,fs,0,fs/2,N,256,128,'yes')
```

References

Welch, P. (1967). The use of fast Fourier transform for the estimation of power spectra: a method based on time averaging over short, modified periodograms. IEEE Transactions on audio and electroacoustics, 15(2), 70-73.

scientimate.wavefromsurfaceelevzcross

```
Hs, Ts, Hz, Tz, Hrms, H, T = scientimate.wavefromsurfaceelevzcross(Eta, fs, dispout=
↪'no')
```

Description

Calculate wave properties from water surface elevation by using an upward zero crossing method

Inputs

Eta Water surface elevation time series data in (m)

fs Sampling frequency that data collected at in (Hz)

dispout='no' Define to display outputs or not ('yes': display, 'no': not display)

Outputs

Hs Significant Wave Height (m)

Ts Significant Wave Period (second)

Hz Zero Crossing Mean Wave Height (m)

Tz Zero Crossing Mean Wave Period (second)

Hrms Root Mean Square Wave Height (m)

H Wave Height Data Series array (m)

T Wave Period Data Series array (second)

Examples

```
import scientimate as sm
import numpy as np
import scipy as sp
from scipy import signal

fs=2 #Sampling frequency
duration=1024 #Duration of the data
N=fs*duration #Total number of points
df=fs/N #Frequency difference
dt=1/fs #Time difference, dt=1/fs
t=np.linspace(0,duration-dt,N) #Time
Eta=sp.signal.detrend(0.5*np.cos(2*np.pi*0.2*t)+(-0.1+(0.1-(-0.1)))*np.random.rand(N))
Hs,Ts,Hz,Tz,Hrms,H,T=sm.wavefromsurfaceelevzcross(Eta,fs,'yes')
```

References

scientimate.wavepropfrompsd

```
Hm0, fp, Tp, Tm01, Tm02, m0, m1, m2 = scientimate.wavepropfrompsd(Syy, f, fcL=0,␣
↪fcH=None, dispout='no')
```

Description

Calculate wave properties from a power spectral density

Inputs

Syy Power spectral density (m^2/Hz)

f Frequency (Hz)

fcL=0 Low cut-off frequency, between 0*fs to 0.5*fs (Hz)

fcH=max(f) High cut-off frequency, between 0*fs to 0.5*fs (Hz)

dispout='no' Define to display outputs or not ('yes': display, 'no': not display)

Outputs

Hm0 Zero-Moment Wave Height (m)

fp Peak wave frequency (Hz)

Tp Peak wave period (second)

Tm01 Wave Period from m01 (second), Mean Wave Period

Tm02 Wave Period from m02 (second), Mean Zero Crossing Period

m0 Zero-Moment of the power spectral density (m^2)

m1 First-Moment of the power spectral density (m^2)

m2 Second-Moment of the power spectral density (m^2)

Examples

```
import scientimate as sm
import numpy as np

N=2**11
fs=8
df=fs/N #Frequency difference
f=np.arange(0,fs/2+df,df) #Frequency vector
f[0]=f[1]/2 #Assign temporarily non-zero value to fisrt element of f to prevent␣
↪division by zero
Syy=0.016*9.81**2/((2*np.pi)**4*(f**5))*np.exp(-1.25*(0.33/f)**4) #Calculating␣
↪Spectrum
f[0]=0
Syy[0]=0
Hm0,fp,Tp,Tm01,Tm02,m0,m1,m2=sm.wavepropfrompsd(Syy,f,0,np.max(f),'yes')
```

References

For Ocean Wave Analyzing Toolbox, see *OCEANLYZ* section above.

1.3.10 Water Wave Directional Analysis

scientimate.directionalpsd

```
Syy2d, f2d, theta = scientimate.directionalpsd(Syy, f, Wavedir=0, calcMethod=
↪'mitsuyasu', Windvel=0, dtheta=15, dispout='no')
```

Description

Calculate wave directional spectrum using parametric directional spreading function such as:
Mitsuyasu et al. (1975), Hasselmann et al. (1980) and Donelan et al. (1985)

Inputs

Syy One dimensional power spectral density (m^2/Hz)

f Frequency (Hz)

Wavedir=0 Mean wave direction between 0 and 360 (Degree)

CalcMethod='mitsuyasu'

Directional wave spectrum calculation method
'pierson': Pierson et al. (1955), 'cos2': D=1/pi*(cos((theta-theta_mean)/2))^2
'mitsuyasu': Mitsuyasu (1975), 'hasselmann': Hasselmann (1980), 'donelan': Donelan et al. (1985)
'flat': uniform distribution in all directions

Windvel=0

Wind velocity at 10 meter above surface level in (m/s)
Wind velocity is required for Mitsuyasu (1975) and Hasselmann (1980) methods
For other methods use Windvel=0

dtheta=15 Direction interval at which directional spectrum calculated between 0 and 360 (Degree)

dispout='no'

Define to display outputs or not
'2d': 2 dimensional plot, 'surface': Surface plot, 'polar': Polar plot, 'no': not display

Outputs

Syy2d Directional wave power spectral density (m^2/Hz/Degree)

f2d Directional frequency (Hz)

theta Direction (Degree)

Examples

```
import scientimate as sm
import numpy as np

N=2**11 #Total number of points
fs=8 #Sampling frequency
df=fs/N #Frequency difference
f=np.arange(0,fs/2+df,df) #Frequency vector
f[0]=f[1]/2 #Assign temporarily non-zero value to fisrt element of f to prevent␣
↪division by zero
Syy=0.016*9.81**2/((2*np.pi)**4*(f**5))*np.exp(-1.25*(0.33/f)**4) #Calculating␣
↪Spectrum
f[0]=0
Syy[0]=0
Syy2d,f2d,theta=sm.directionalpsd(Syy,f,45,'mitsuyasu',10,15,'polar')
```

References

Banner, M. L. (1990). Equilibrium spectra of wind waves. Journal of Physical Oceanography, 20(7), 966-984.

Donelan, M. A., Hamilton, J., & Hui, W. (1985). Directional spectra of wind-generated waves. Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences, 315(1534), 509-562.

Ewans, K. C. (1998). Observations of the directional spectrum of fetch-limited waves. Journal of Physical Oceanography, 28(3), 495-512.

Goda, Y. (1999). A comparative review on the functional forms of directional wave spectrum. Coastal Engineering Journal, 41(01), 1-20.

Hasselmann, D. E., Dunckel, M., & Ewing, J. A. (1980). Directional wave spectra observed during JONSWAP 1973. Journal of physical oceanography, 10(8), 1264-1280.

Hwang, P. A., & Wang, D. W. (2001). Directional distributions and mean square slopes in the equilibrium and saturation ranges of the wave spectrum. Journal of physical oceanography, 31(5), 1346-1360.

Mitsuyasu, H., Tasai, F., Suhara, T., Mizuno, S., Ohkusu, M., Honda, T., & Rikiishi, K. (1975). Observations of the directional spectrum of ocean WavesUsing a cloverleaf buoy. Journal of Physical Oceanography, 5(4), 750-760.

Pierson, W. J., Neumann, G., & James, R. W. (1955). Practical methods for observing and forecasting ocean waves by means of wave spectra and statistics. Publication 603, U.S. Navy Hydrographic Office, 284 pp.

Sorensen, R. M. (2005). Basic coastal engineering (Vol. 10). Springer Science & Business Media.

scientimate.enu2truenorth

```
directionTN = scientimate.enu2truenorth(directionENU, dispout='no')
```

Description

Convert mathematical direction (angle) from ENU (East North Up) coordinate system to compass direction with respect to true north

Inputs

directionENU

Direction (angle) in ENU (East North Up) coordinate system between 0 and 360 (Degree)
If coordinate system is ENU, then x is East and y is North

dispout='no'

Define to display outputs or not ('yes': display, 'no': not display)
Note: inputs can be as a single value or a 1-D vertical array

Outputs

directionTN

Direction (angle) in compass direction with respect to true north (Degree)
In true north coordinate system, wave comes from as:
0 degree: from north, 90 degree: from east, 180 degree: from south, 270 degree: from west

Examples

```
import scientimate as sm
import numpy as np

directionTN=sm.enu2truenorth(90,'yes')

directionTN=sm.enu2truenorth([15,30,45,60,90],'no')

directionTN=sm.enu2truenorth(np.array([15,30,45,60,90]),'no')
```

References

1.3.11 Water Wave Parametric Model

scientimate.parametricwavedeep

```
H, T, Ehat, fphat, m0, Fetchhat = parametricwavedeep(windvel, Fetch, CalcMethod=
↪'jonswap', dispout='no')
```

Description

Calculate wave properties using parametric wave models in deep water

Inputs

windvel

Wind velocity in (m/s)
Wind velocity should be measured (or represents velocity) at 10 m above surface
Wind velocity should be a 10 minutes averaged wind for all methods, except for for 'cem' and 'spmdeep' methods
For 'cem' and 'spmdeep' methods, wind velocity should be converted to duration of sustained wind by using gust factor

Fetch Wind fetch in (m)

CalcMethod='jonswap'

Parametric wave model to be used for wave calculation
'wilson': Use method by Wislon (1965)
'jonswap': Use method by Hasselmann et al. (1973) known as JONSWAP
'spmdeep': Use method by Shore Protection Manual (SPM),
U.S. Army Corps of Engineers (1984) in deep water
'kahma': Use method by Kahma and Calkoen (1992)
'hwang': Use method by Hwang and Wang (2004)
'cem': Use method by Coastal Engineering Manual (CEM),
U.S. Army Corps of Engineers (2015)

dispout='no' Define to display outputs or not ('yes': display, 'no': not display)

Outputs

H

Predicted wave height in (m)
For all methods excepth for 'wilson': H=Hm0 where, Hm0 is a zero-moment wave height
For 'wilson' method: H=Hs, where Hs is a significant wave height

T

Predicted wave period in (s)
For all methods excepth for 'wilson': T=Tp where, Tp is a peak wave period

For 'wilson' method: T=Ts, where Ts is a significant wave period (Ts=0.95Tp)

Ehat Predicted dimensionless wave energy, Ehat=g^2*m0/U10^4

fphat Predicted dimensionless peak wave frequency, fphat=fp*U10/g

m0 Zero-moment of water surface elevation power spectral density in (m^2)

Fetchhat

Dimensionless Fetch: Fetchhat=g*Fetch/U10^2
Note, g=9.81: gravitational acceleration
U10: wind velocity
fp: peak wave frequency

Examples

```
import scientimate as sm
import numpy as np

windvel=10*np.random.rand(100)
Fetch=10000*np.random.rand(100)
H,T,Ehat,fphat,m0,Fetchhat=sm.parametricwavedeep(windvel,Fetch,'jonswap','no')

windvel=10
Fetch=np.arange(1e3,1e6,1000)
H,T,Ehat,fphat,m0,Fetchhat=sm.parametricwavedeep(windvel,Fetch,'jonswap','yes')
```

References

Department of the Army, Waterways Experiment Station, Corps of Engineers, and Coastal Engineering Research Center (1984), Shore Protection Manual, Washington, D.C., vol. 1, 4th ed., 532 pp.

Hasselmann, K.; Barnett, T. P.; Bouws, E.; Carlson, H.; Cartwright, D. E.; Enke, K.; Ewing, J. A.; Gienapp, H.; Hasselmann, D. E.; Kruseman, P.; Meerbrug, A.; Muller, P.; Olbers, D. J.; Richter, K.; Sell, W., and Walden, H., (1973). Measurements of wind-wave growth and swell decay during the Joint North Sea Wave Project (JONSWAP). Deutsche Hydrographische Zeitschrift A80(12), 95p.

Holthuijsen, L. H. (2007). Waves in oceanic and coastal waters. Cambridge university press.

Hwang, P. A., & Wang, D. W. (2004). Field measurements of duration-limited growth of wind-generated ocean surface waves at young stage of development. Journal of Physical Oceanography, 34(10), 2316-2326.

Kahma, K. K., & Calkoen, C. J. (1992). Reconciling discrepancies in the observed growth of wind-generated waves. Journal of Physical Oceanography, 22(12), 1389-1405.

Pierson, W. J., & Moskowitz, L. (1964). A proposed spectral form for fully developed wind seas based on the similarity theory of SA Kitaigorodskii. Journal of geophysical research, 69(24), 5181-5190.

U.S. Army Corps of Engineers (2015). Coastal Engineering Manual. Engineer Manual 1110-2-1100, Washington, D.C.: U.S. Army Corps of Engineers.

Wilson, B. W. (1965). Numerical prediction of ocean waves in the North Atlantic for December, 1959. Ocean Dynamics, 18(3), 114-130.

scientimate.parametricwaveshallow

```
H, T, Ehat, fphat, m0, Fetchhat, hhat = scientimate.parametricwaveshallow(windvel,␣
↪Fetch, hmean, CalcMethod='karimpour', dispout='no')
```

Description

Calculate wave properties using parametric wave models in shallow and intermediate water

Inputs

windvel

Wind velocity in (m/s)
Wind velocity should be measured (or represents velocity) at 10 m above surface
Wind velocity should be a 10 minutes averaged wind for all methods, except for for 'spmshallow' methods
For 'spmshallow' methods, wind velocity should be converted to duration of sustained wind by using gust factor

Fetch Wind fetch in (m)

hmean Mean water depth along a wind fetch in (m)

CalcMethod='karimpour'

Parametric wave model to be used for wave calculation
'spmshallow': Use method by Shore Protection Manual (SPM),
U.S. Army Corps of Engineers (1984) in shallow and intermediate water water
'young': Use method by Young and Verhagen (1996) and Young and Babanin (2006)
'karimpour': Use method by Karimpour et al. (2017)

dispout='no' Define to display outputs or not ('yes': display, 'no': not display)

Outputs

H

Predicted wave height in (m)
For all methods: H=Hm0 where, Hm0 is a zero-moment wave height

T

Predicted wave period in (s)
For all methods excepth for 'spmshallow': T=Tp where, Tp is a peak wave period
For 'spmshallow' method: T=Ts, where Ts is a significant wave period (Ts=0.95Tp)

Ehat Predicted dimensionless wave energy, Ehat=g^2*m0/U10^4

fphat Predicted dimensionless peak wave frequency, fphat=fp*U10/g

m0 Zero-moment of water surface elevation power spectral density in (m^2)

Fetchhat Dimensionless wind fetch: Fetchhat=g*Fetch/U10^2

hhat

Dimensionless mean water depth along a wind fetch: hhat=g*hmean/U10^2
Note, g=9.81: gravitational acceleration
U10: wind velocity
fp: peak wave frequency
hmean: mean depth along a wind fetch

Examples

```
import scientimate as sm
import numpy as np

windvel=10*np.random.rand(100)
Fetch=10000*np.random.rand(100)
hmean=3*np.random.rand(100)
H,T,Ehat,fphat,m0,Fetchhat,hhat=sm.parametricwaveshallow(windvel,Fetch,hmean,
↪'karimpour','no')

windvel=10
Fetch=np.arange(1e3,1e6,1000)
hmean=3
H,T,Ehat,fphat,m0,Fetchhat,hhat=sm.parametricwaveshallow(windvel,Fetch,hmean,
↪'karimpour','yes')
```

References

Bretschneider, C. L. (1952). Revised wave forecasting relationships. Coastal Engineering Proceedings, 1(2), 1.

Bretschneider, C. L. (1958). Revisions in wave forecasting: deep and shallow water. Coastal Engineering Proceedings, 1(6), 3.

Department of the Army, Waterways Experiment Station, Corps of Engineers, and Coastal Engineering Research Center (1984), Shore Protection Manual, Washington, D.C., vol. 1, 4th ed., 532 pp.

Karimpour, A., Chen, Q., & Twilley, R. R. (2017). Wind Wave Behavior in Fetch and Depth Limited Estuaries. Scientific reports, 7, 40654.

Pierson, W. J., & Moskowitz, L. (1964). A proposed spectral form for fully developed wind seas based on the similarity theory of SA Kitaigorodskii. Journal of geophysical research, 69(24), 5181-5190.

Sverdrup, H. U., & Munk, W. H. (1947). Wind, sea, and swell: theory of relations for forecasting. U.S. Navy Department, Hydrographic Office, Publication No. 601, 44 pp.

Young, I. R., & Verhagen, L. A. (1996). The growth of fetch limited waves in water of finite depth. Part 1. Total energy and peak frequency. Coastal Engineering, 29(1-2), 47-78.

Young, I. R., & Babanin, A. V. (2006). The form of the asymptotic depth-limited wind wave frequency spectrum. Journal of Geophysical Research: Oceans, 111(C6).

1.3.12 Water Wave Properties

scientimate.bretpsd

```
f, Syy, Hm0, fp, Tp, Tm01, Tm02 = scientimate.bretpsd(Hs=1, fp=0.33, fs=2, N=256,␣
↪dispout='no')
```

Description

Calculate Bretschneider spectrum (power spectral density), (Bretschneider, 1959), ITTC spectrum

Inputs

Hs=1 Significant wave height (m)

fp=0.33 Peak wave frequency in (Hz)

fs=2 Sampling frequency that data collected at in (Hz)

N=256 Total number of points between 0 and fs that spectrum reports at is (N+1)

dispout='no' Define to display outputs or not ('yes': display, 'no': not display)

Outputs

f Frequency (Hz)

Syy Wave Energy Power Spectrum (m^2/Hz)

Hm0 Zero-Moment Wave Height (m)

fp Peak wave frequency (Hz)

Tp Peak wave period (second)

Tm01 Wave Period from m01 (second), Mean Wave Period

Tm02 Wave Period from m02 (second), Mean Zero Crossing Period

Examples

```
import scientimate as sm
f,Syy,Hm0,fp,Tp,Tm01,Tm02=sm.bretpsd(1,0.33,2,256,'yes')
```

References

Bretschneider, C. L. (1959). Wave variability and wave spectra for wind-generated gravity waves (No. TM-118). CORPS OF ENGINEERS WASHINGTON DC BEACH EROSION BOARD.

Stansberg, C. T., Contento, G., Hong, S. W., Irani, M., Ishida, S., & Mercier, R. (2002). The specialist committee on waves final report and recommendations to the 23rd ITTC. In Proceedings of the 23rd ITTC (Vol. 2, pp. 505-551).

Zwolan, P., & Czaplewski, K. (2012). Sea waves models used in maritime simulators. Zeszyty Naukowe/Akademia Morska w Szczecinie, (32 (104) z. 2), 186-190.

scientimate.donelanpsd

```
f, Syy, Hm0, fp, Tp, Tm01, Tm02 = scientimate.donelanpsd(U10=10, F=10000, fp=0.33,␣
↪fs=2, N=256, CalSpectralSP='yes', dispout='no')
```

Description

Calculate Donelan spectrum (power spectral density), (Donelan et al. 1985)

Inputs

U10=10 Wind velocity at 10 meter above surface level in (m/s)

F=10000 Wind fetch length in (m)

fp=0.33

Peak wave frequency (fp=1/Tp) in (Hz)
If CalSpectralSP='yes'; then fp is calculated from U10 and F

fs=2 Sampling frequency that data collected at in (Hz)

N=256

Total number of points between 0 and fs-df, where df=fs/N
Spectrum is reported between 0 and fs/2 with total number of points equal to (N/2+1)
Total number of points between 0 and fs is (N+1)

CalSpectralSP='yes' Define to calculate spectral shape parameters or not ('yes': calculate, 'no': use given parameters by user)

dispout='no' Define to display outputs or not ('yes': display, 'no': not display)

Outputs

f Frequency (Hz)

Syy Wave Energy Power Spectrum (m^2/Hz)

Hm0 Zero-Moment Wave Height (m)

fp Peak wave frequency (Hz)

Tp Peak wave period (second)

Tm01 Wave Period from m01 (second), Mean Wave Period

Tm02 Wave Period from m02 (second), Mean Zero Crossing Period

Examples

```
import scientimate as sm
f,Syy,Hm0,fp,Tp,Tm01,Tm02=sm.donelanpsd(10,10000,0.33,2,256,'yes','yes')
```

References

Donelan, M.A.; Hamilton, J., and Hui, W., 1985. Directional spectra of wind-generated waves. Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences, 315(1534), 509â€“562.

scientimate.jonswappsd

```
f, Syy, Hm0, fp, Tp, Tm01, Tm02 = scientimate.jonswappsd(U10=10, F=10000, fp=0.33,␣
↪fs=2, N=256, gama=3.3, CalSpectralSP='yes', dispout='no')
```

Description

Calculate JONSWAP spectrum (power spectral density), (Hasselmann et al. 1973)

Inputs

U10=10 Wind velocity at 10 meter above surface level in (m/s)

F=10000 Wind fetch length in (m)

fp=0.33

Peak wave frequency (fp=1/Tp) in (Hz)
If CalSpectralSP='yes'; then fp is calculated from U10 and F

gama=3.3 Peak enhancement parameter (between 1 and 7)

fs=2 Sampling frequency that data collected at in (Hz)

N=256 Total number of points between 0 and fs that spectrum reports at is (N+1)

CalSpectralSP='yes' Define to calculate spectral shape parameters or not ('yes': calculate, 'no': use given parameters by user)

dispout='no' Define to display outputs or not ('yes': display, 'no': not display)

Outputs

f Frequency (Hz)

Syy Wave Energy Power Spectrum (m^2/Hz)

Hm0 Zero-Moment Wave Height (m)

fp Peak wave frequency (Hz)

Tp Peak wave period (second)

Tm01 Wave Period from m01 (second), Mean Wave Period

Tm02 Wave Period from m02 (second), Mean Zero Crossing Period

Examples

```
import scientimate as sm
f,Syy,Hm0,fp,Tp,Tm01,Tm02=sm.jonswappsd(10,10000,0.33,2,256,3.3,'yes','yes')
```

References

Hasselmann, K.; Barnett, T. P.; Bouws, E.; Carlson, H.; Cartwright, D. E.; Enke, K.; Ewing, J. A.; Gienapp, H.; Hasselmann, D. E.; Kruseman, P.; Meerbrug, A.; Muller, P.; Olbers, D. J.; Richter, K.; Sell, W., and Walden, H., (1973). Measurements of wind-wave growth and swell decay during the Joint North Sea Wave Project (JONSWAP). Deutsche Hydrographische Zeitschrift A80(12), 95p.

scientimate.pmpsd

```
f, Syy, Hm0, fp, Tp, Tm01, Tm02 = scientimate.pmpsd(U195=10, fp=0.33, fs=2, N=256,␣
↪CalSpectralSP='yes', dispout='no')
```

Description

Calculate Pierson-Moskowitz spectrum (power spectral density), (Pierson and Moskowitz 1964)

Inputs

U195=10 Wind velocity at 19.5 meter above surface level in (m/s)

fp=0.33

Peak wave frequency (fp=1/Tp) in (Hz)
If CalSpectralSP='yes'; then fp is calculated from U195

fs=8 Sampling frequency that data collected at in (Hz)

N=256 Total number of points between 0 and fs that spectrum reports at is (N+1)

CalSpectralSP='yes' Define to calculate spectral shape parameters or not ('yes': calculate, 'no': use given parameters by user)

dispout='no' Define to display outputs or not ('yes': display, 'no': not display)

Outputs

f Frequency (Hz)

Syy Wave Energy Power Spectrum (m^2/Hz)

Hm0 Zero-Moment Wave Height (m)

fp Peak wave frequency (Hz)

Tp Peak wave period (second)

Tm01 Wave Period from m01 (second), Mean Wave Period

Tm02 Wave Period from m02 (second), Mean Zero Crossing Period

Examples

```
import scientimate as sm
f,Syy,Hm0,fp,Tp,Tm01,Tm02=sm.pmpsd(10,0.33,2,256,'yes','yes')
```

References

Pierson, W. J., & Moskowitz, L. (1964). A proposed spectral form for fully developed wind seas based on the similarity theory of SA Kitaigorodskii. Journal of geophysical research, 69(24), 5181-5190.

scientimate.pressureresponse

```
Kp, f = scientimate.pressureresponse(f, h, heightfrombed, kCalcMethod='beji', dispout=
↪'no')
```

Description

Calculate a pressure response factor

Inputs

f Frequency in (Hz)

h Water depth in (m)

heightfrombed Height from bed that Kp calculated at in (m)

kCalcMethod='beji'

Wave number calculation method
'hunt': Hunt (1979), 'beji': Beji (2013), 'vatankhah': Vatankhah and Aghashariatmadari (2013)
'goda': Goda (2010), 'exact': calculate exact value

dispout='no'

Define to display outputs or not ('yes': display, 'no': not display)
Note: inputs can be as a single value or a 1-D vertical array

Outputs

Kp Pressure response factor

f Frequency (Hz)

Examples

```
import scientimate as sm
import numpy as np

Kp,f=sm.pressureresponse(0.2,1,0.2,'beji','no')

Kp,f=sm.pressureresponse([0.2,0.25],[0.5,0.6],0.2,'exact','yes')

Kp,f=sm.pressureresponse(np.array([0.2,0.25]),np.array([0.5,0.6]),0.2,'exact','yes')
```

References

Beji, S. (2013). Improved explicit approximation of linear dispersion relationship for gravity waves. Coastal Engineering, 73, 11-12.

Goda, Y. (2010). Random seas and design of maritime structures. World scientific.

Hunt, J. N. (1979). Direct solution of wave dispersion equation. Journal of the Waterway Port Coastal and Ocean Division, 105(4), 457-459.

Vatankhah, A. R., & Aghashariatmadari, Z. (2013). Improved explicit approximation of linear dispersion relationship for gravity waves: A discussion. Coastal engineering, 78, 21-22.

scientimate.tmapsd

```
f, Syy, Hm0, fp, Tp, Tm01, Tm02, PHI = scientimate.tmapsd(U10=10, F=10000, h=5, fp=0.
↪33, fs=2, N=256, CalSpectralSP='yes', transfCalcMethod='approx', kCalcMethod='beji',
↪ dispout='no')
```

Description

Calculate TMA spectrum (power spectral density), (Bouws et al. 1985)

Inputs

U10=10 Wind velocity at 10 meter above surface level in (m/s)

F=10000 Wind fetch length in (m)

h=5 Mean water depth in (m)

fp=0.33

Peak wave frequency (fp=1/Tp) in (Hz)
If CalSpectralSP='yes'; then fp is calculated from U10 and F

fs=2 Sampling frequency that data collected at in (Hz)

N=256 Total number of points between 0 and fs that spectrum reports at is (N+1)

CalSpectralSP='yes' Define to calculate spectral shape parameters or not ('yes': calculate, 'no': use given parameters by user)

transfCalcMethod='approx'

Transformation function from JONSWAP into TMA calculation method
'approx': approximated method, 'tucker': Tucker (1994), 'kitaigordskii': Kitaigordskii et al. (1975)

kCalcMethod='beji'

Wave number calculation method
'hunt': Hunt (1979), 'beji': Beji (2013), 'vatankhah': Vatankhah and Aghashariatmadari (2013)
'goda': Goda (2010), 'exact': calculate exact value

dispout='no' Define to display outputs or not ('yes': display, 'no': not display)

Outputs

f Frequency (Hz)

Syy Wave Energy Power Spectrum (m^2/Hz)

Hm0 Zero-Moment Wave Height (m)

fp Peak wave frequency (Hz)

Tp Peak wave period (second)

Tm01 Wave Period from m01 (second), Mean Wave Period

Tm02 Wave Period from m02 (second), Mean Zero Crossing Period

PHI Transformation function from JONSWAP into TMA

Examples

```
import scientimate as sm
f,Syy,Hm0,fp,Tp,Tm01,Tm02,PHI=sm.tmapsd(10,10000,5,0.33,2,256,'yes','approx','beji',
↪'yes')
```

References

Beji, S. (2013). Improved explicit approximation of linear dispersion relationship for gravity waves. Coastal Engineering, 73, 11-12.

Bouws, E.; GÃ¼nther, H.; Rosenthal, W., and Vincent, C.L., (1985). Similarity of the wind wave spectrum in finite depth water: 1. Spectral form. Journal of Geophysical Research: Oceans, 90(C1), 975-986.

Goda, Y. (2010). Random seas and design of maritime structures. World scientific.

Hunt, J. N. (1979). Direct solution of wave dispersion equation. Journal of the Waterway Port Coastal and Ocean Division, 105(4), 457-459.

Kitaigordskii, S. A., Krasitskii, V. P., & Zaslavskii, M. M. (1975). On Phillips' theory of equilibrium range in the spectra of wind-generated gravity waves. Journal of Physical Oceanography, 5(3), 410-420.

Vatankhah, A. R., & Aghashariatmadari, Z. (2013). Improved explicit approximation of linear dispersion relationship for gravity waves: A discussion. Coastal engineering, 78, 21-22.

Tucker, M. J. (1994). Nearshore waveheight during storms. Coastal Engineering, 24(1-2), 111-136.

scientimate.wavedispersion

```
k, L, C, Cg = scientimate.wavedispersion(h, T, kCalcMethod='beji')
```

Description

Solve water wave dispersion relation
Calculate wave number (k), wave length (L), wave celereity (C), and wave group velocity (Cg) using linear wave theory

Inputs

h Water depth in (m)

T

Wave period in (s)
If peak wave frequency (Tp) is used, calculated values represent peak wave

kCalcMethod='beji'

Wave number calculation method
'hunt': Hunt (1979), 'beji': Beji (2013), 'vatankhah': Vatankhah and Aghashariatmadari (2013)
'goda': Goda (2010), 'exact': calculate exact value
Note: inputs can be as a single value or a 1-D vertical array

Outputs

k Wave number in (radian/m)

L Wave length in (m)

C Wave celerity in (m/s)

Cg Wave group celerity in (m/s)

Examples

```
import scientimate as sm
import numpy as np

k,L,C,Cg=sm.wavedispersion(1,3,'beji')

k,L,C,Cg=sm.wavedispersion([1,1.1],[3,3.1],'exact')

k,L,C,Cg=sm.wavedispersion(np.array([1,1.1]),np.array([3,3.1]),'exact')
```

References

Beji, S. (2013). Improved explicit approximation of linear dispersion relationship for gravity waves. Coastal Engineering, 73, 11-12.

Goda, Y. (2010). Random seas and design of maritime structures. World scientific.

Hunt, J. N. (1979). Direct solution of wave dispersion equation. Journal of the Waterway Port Coastal and Ocean Division, 105(4), 457-459.

Vatankhah, A. R., & Aghashariatmadari, Z. (2013). Improved explicit approximation of linear dispersion relationship for gravity waves: A discussion. Coastal engineering, 78, 21-22.

1.3.13 Wind

scientimate.directionavg

```
diravg = scientimate.directionavg(direction, NPointsAvg=None, NPointsInterval=None,␣
↪dispout='no')
```

Description

Average direction

Inputs

direction Direction time series data in (degree)

NPointsAvg=length(direction(:,1)) Number of data points from start of each section (interval) to be averaged

NPointsInterval=length(direction(:,1)) Number of points that each section (interval) has

dispout='no' Define to display outputs or not ('yes': display, 'no': not display)

Outputs

diravg Averaged direction data in (degree)

Examples

```
import scientimate as sm
import numpy as np

direction=45*np.random.rand(10)
diravg=sm.directionavg(direction)

direction=225*np.random.rand(5*60) #One data point every minute for 5 hours
diravg=sm.directionavg(direction,10,60,'yes')
```

References

Yamartino, R. J. (1984). A comparison of several "single-pass" estimators of the standard deviation of wind direction. Journal of Climate and Applied Meteorology, 23(9), 1362-1366.

scientimate.surfaceroughness

```
ustar, z0, d = scientimate.surfaceroughness(z, u, delta=None, dispout='no')
```

Description

Calculate shear velocity and surface roughness from a given velocity profile using Karimpour et al. (2012) method

Inputs

z Distance from a surface (elevation, height) in (m)

u Velocity at z in (m/s)

delta=max(z) Boundary layer height in (m)

dispout='no' Define to display outputs or not ('yes': display, 'no': not display)

Outputs

z0 Surface roughness in (m)

ustar Shear velocity (u*) in (m/s)

d

Zero plane displacement distance in (m)
Note: Above values are for a logarithmic velocity profile as:
u=(u*/K)*ln((z-d)/z0)

Examples

```
import scientimate as sm
import numpy as np

z=np.arange(0.1,1,0.05)
u=2/0.4*np.log((z-0.003)/0.002)
ustar,z0,d=sm.surfaceroughness(z,u,np.max(z),'yes')
```

References

Karimpour, A., Kaye, N. B., & Baratian-Ghorghi, Z. (2012). Modeling the neutrally stable atmospheric boundary layer for laboratory scale studies of the built environment. Building and Environment, 49, 203-211.

scientimate.sustainedwindduration

```
SustWindDur = scientimate.sustainedwindduration(windvel, winddir, MaxwindvelVar=2.5,␣
↪MaxwinddirVar=15, WindInterval=3600, dispout='no')
```

Description

Calculate the sustained wind duration

Inputs

windvel Wind velocity time series data in (m/s)

winddir Wind direction time series data in (Degree)

MaxwindvelVar=2.5

Maximum allowed wind velocity variation around a mean to allow wind to be considered sustained in (m/s)
Coastal Engineering Manual (2015) suggests 2.5 m/s

MaxwinddirVar=15

Maximum allowed wind direction variation around a mean to allow wind to be considered sustained in (Degree)
Coastal Engineering Manual (2015) suggests 15 degree

WindInterval=3600 Time interval between two consecutive wind measurements (data points) in (second)

dispout='no' Define to display outputs or not ('yes': display, 'no': not display)

Outputs

SustWindDur Sustained Wind Duration in (second)

Examples

```
import scientimate as sm

#Data from https://tidesandcurrents.noaa.gov for Grand Isle, LA, USA (8761724), for␣
↪June 1st, 2017, reported hourly
windvel=[3,4.7,4.9,5.3,3.3,3.4,3.3,3.8,3.1,2,1.3,1.2,1.5,3.2,2.9,3,2.9,3.7,3.7,3.1,3.
↪4,2.6,2.5,2.5] #24 Hour wind velocity
winddir=[78,86,88,107,131,151,163,163,153,150,148,105,105,75,95,103,97,103,108,111,
↪124,183,171,113] #24 Hour wind direction
SustWindDur=sm.sustainedwindduration(windvel,winddir,2.5,15,3600,'yes')
```

References

U.S. Army Corps of Engineers (2015). Coastal Engineering Manual. Engineer Manual 1110-2-1100, Washington, D.C.: U.S. Army Corps of Engineers.

Yamartino, R. J. (1984). A comparison of several "single-pass" estimators of the standard deviation of wind direction. Journal of Climate and Applied Meteorology, 23(9), 1362-1366.

scientimate.windavg

```
windvelavg, winddiravg = scientimate.windavg(windvel, winddir, NPointsAvg=None,␣
↪NPointsInterval=None, dispout='no')
```

Description

Average wind velocity and wind direction

Inputs

windvel Wind velocity time series data

winddir Wind direction time series data in (degree)

NPointsAvg=length(windvel(:,1)) Number of data points from start of each section (interval) to be averaged

NPointsInterval=length(windvel(:,1)) Number of points that each section (interval) has

dispout='no' Define to display outputs or not ('yes': display, 'no': not display)

Outputs

windvelavg Averaged wind velocity data

winddiravg Averaged wind direction data in (degree)

Examples

```
import scientimate as sm
import numpy as np

windvel=10*np.random.rand(10)
winddir=45*np.random.rand(10)
windvelavg,winddiravg=sm.windavg(windvel,winddir)

windvel=10*np.random.rand(5*60) #One data point every minute for 5 hours
winddir=225*np.random.rand(5*60) #One data point every minute for 5 hours
windvelavg,winddiravg=sm.windavg(windvel,winddir,10,60,'yes')
```

References

Yamartino, R. J. (1984). A comparison of several "single-pass" estimators of the standard deviation of wind direction. Journal of Climate and Applied Meteorology, 23(9), 1362-1366.

scientimate.winddrag

```
CD, Tauwind, Ustar = scientimate.winddrag(U10, CalcMethod='wu', Rhoa=1.204, dispout=
↪'no')
```

Description

Calculate wind drag coefficient, wind shear stress, and wind shear velocity

Inputs

U10

Wind velocity in (m/s)

Wind velocity should be measured (or represents velocity) at 10 m above surface

kCalcMethod='beji'

Drag coefficient calculation method
'vandoren': Van Doren (1953)
'garratt': Garratt (1977)
'smith': Smith (1977)
'large': Large & Pond (1981)
'wu': Wu (1982)
'hwang': Hwang (2011)
'zijlema': Zijlema et al. (2012)
'cem': Use method by Coastal Engineering Manual (CEM),
U.S. Army Corps of Engineers (2015)

Rhoa=1.204 Air density in (kg/m3)

dispout='no' Define to display outputs or not ('yes': display, 'no': not display)

Outputs

CD Wind drag coefficient

Tauwind Wind shear stress in (N/m^2), Tauwind=Rhoa*CD*U10^2

Ustar Wind shear velocity in (m/s), Ustar=(CD*(U10^2))^0.5

Examples

```
import scientimate as sm
import numpy as np

U10=np.arange(4,40,0.1)
CD,Tauwind,Ustar=sm.winddrag(U10,'wu',1.204,'yes')
```

References

Bryant, K. M., & Akbar, M. (2016). An Exploration of Wind Stress Calculation Techniques in Hurricane Storm Surge Modeling. Journal of Marine Science and Engineering, 4(3), 58.

Chen, Y., & Yu, X. (2017). Sensitivity of storm wave modeling to wind stress evaluation methods. Journal of Advances in Modeling Earth Systems.

Dean, R. G., & Dalrymple, R. A. (1991). Water wave mechanics for engineers and scientists (Vol. 2). world scientific publishing Co Inc.

Garratt, J. R. (1977). Review of drag coefficients over oceans and continents. Monthly weather review, 105(7), 915-929.

Hwang, P. A. (2011). A note on the ocean surface roughness spectrum. Journal of Atmospheric and Oceanic Technology, 28(3), 436-443.

Large, W. G., & Pond, S. (1981). Open ocean momentum flux measurements in moderate to strong winds. Journal of physical oceanography, 11(3), 324-336.

Rogers, W. E., Babanin, A. V., & Wang, D. W. (2012). Observation-consistent input and whitecapping dissipation in a model for wind-generated surface waves: Description and simple calculations. Journal of Atmospheric and Oceanic Technology, 29(9), 1329-1346.

Smith, S. D. (1980). Wind stress and heat flux over the ocean in gale force winds. Journal of Physical Oceanography, 10(5), 709-726.

U.S. Army Corps of Engineers (2015). Coastal Engineering Manual. Engineer Manual 1110-2-1100, Washington, D.C.: U.S. Army Corps of Engineers.

VanDorn, W. G. (1953). Wind stress on an artificial pond. Journal of Marine Research, 12(3), 249-276.

Wu, J. (1982). Wind-stress coefficients over sea surface from breeze to hurricane. Journal of Geophysical Research: Oceans, 87(C12), 9704-9706.

Zijlema, M., Van Vledder, G. P., & Holthuijsen, L. H. (2012). Bottom friction and wind drag for wave models. Coastal Engineering, 65, 19-26.

scientimate.windgustfactor

```
G = scientimate.windgustfactor(t, t0=3600, CalcMethod='cem', Iu=0, dispout='no')
```

Description

Convert wind velocity of duration t0 to t

Inputs

t

Wind averaging duration to convert to (e.g. Sustained Wind Duration) in (second)
Example: for 10-min wind averaging duration: t=600 (s)

t0=3600

Wind averaging duration that data originally averaged at in (second)
Example: for 60-min wind averaging duration: t0=3600 (s)
If t0 is not defined, then t0 is considered as t0=3600 s
t0 can only have one element

CalcMethod='cem'

Wind gust factor calculation method
'durst': Durst (1960)
'cem': Coastal Engineering Manual, U.S. Army Corps of Engineers (2015)
'cook': Cook (1985) which is 3600 s gust factor based on Wieringa (1973) 600 s gust factor
'krayer': Krayer & Marshall (1992)
'cem' for t<=3600 s is similar to Dusrt (1960) relationship
'cem' for t>3600 s is Cook (1985) relationship with Iu=0.155
'cook' for Iu=0.175 is the commonly plotted curve

Iu=0 Wind longitudinal turbulence intensity

dispout='no' Define to display outputs or not ('yes': display, 'no': not display)

Outputs

G

Wind gust factor
G=U(t)/U(t0)
G=(wind velocity avereged over t seconds)/(wind velocity avereged over t0 seconds)
If t0=3600 s then G is calculated respect with 1-hour avereged wind velocity
If t0=3600 s then G=(wind velocity avereged over t seconds)/(wind velocity avereged over 3600 seconds)
If t0=3600 s then G=U(t)/U(3600)

Examples

```
import scientimate as sm
import numpy as np

t=600
G=sm.windgustfactor(t)

t=range(1,3601,1)
t0=3600
G=sm.windgustfactor(t,t0,'durst',0,'yes')

t=range(1,36001,1)
t0=3600
G=sm.windgustfactor(t,t0,'cem',0,'yes')

t=np.arange(1,3601,1)
t0=3600
G=sm.windgustfactor(t,t0,'cook',0.175,'yes')

t=range(1,3601,1)
t0=3600
G=sm.windgustfactor(t,t0,'krayer',0,'yes')
```

References

American Society of Civil Engineers, ASCE-7. (2005). Minimum design loads for buildings and other structures (Vol. 7). Amer Society of Civil Engineers.

Cook, N. J. (1985). The designer's guide to wind loading on building structures. Part I: Background, damage survey, wind data, and structural classification. Building Research Establishment, Watford. G_t_3600=1+0.42*Iu*log(3600/t)

Durst, C. S. (1960). Wind speeds over short periods of time. Meteor. Mag, 89(1056), 181-187. G_t_3600=0.977+(0.64/(1+(t/38.14)**0.685)) #from ASCE-7 (2005) G_t_3600=0.96+(0.648/(1+(t/38)**0.638)) #from Krayer and Marshall (1992)

Krayer, W. R., & Marshall, R. D. (1992). Gust factors applied to hurricane winds. Bulletin of the American Meteorological Society, 73(5), 613-618. G_t_3600=0.96+(0.839/(1+(t/36.27)**0.655))

U.S. Army Corps of Engineers (2015). Coastal Engineering Manual. Engineer Manual 1110-2-1100, Washington, D.C.: U.S. Army Corps of Engineers.

Wieringa, J. (1973). Gust factors over open water and built-up country. Boundary-Layer Meteorology, 3(4), 424-441. G_t_600=1+(1.42+0.3013*log((600/t)-4))*Iu

windspectrum

```
S, f = windspectrum(U10, Sigma, L, CalcMethod, fmin, fmax, dispout)
```

Description

Calculate wind spectrum with 513 frequencies

Inputs

U10 Wind velocity at 10 m above surface in (m/s)

Sigma

Wind velocity standard deviation

IEC 61400-1:

Sigma_u=Sigma
Sigma_v=0.8*Sigma
Sigma_w=0.5*Sigma

L

Wind velocity integral length scale

Suggestion to calculate L (wind velocity integral length scale):

For Kaimal (1972) suggested by DNV-RP-C205 (2010): L=300*(z/300)^(0.46+0.074*log(z0))

For Kaimal (1972) suggested by DNV-RP-C205 (2010) based on IEC 61400-1:

L=3.33*z for 0<z<=60
L=200 for z>=60

For Kaimal (1972) suggested by IEC 61400-1:

Lambda=0.7*z for for z<=60
Lambda=42 m for for z>60
Lu=8.1*Lambda
Lv=2.7*Lambda
Lw=0.66*Lambda

For Davenport (1961): Lu=1200 m

For Harris (1971): The same as the one for Kaimal (1972) suggested by DNV-RP-C205 (2010)

z Height from surface in (m)

z0 Surface roughness

CalcMethod='kaimal_dnv'

Wind spectrum calculation method
'dnv': DNV-RP-C205 (2010)
'kaimal_dnv': Kaimal (1972) suggested by DNV-RP-C205 (2010)
'kaimal_iec': Kaimal (1972) suggested by IEC 61400-1
'davenport': Davenport (1961)

'harris': Harris (1971)

fmin=0.002 Lower boundary of spectrum in (Hz)

fmax=0.3

Upper boundary of spectrum in (Hz)
Kaimal is valid in range from 0.002 Hz to 0.3 Hz (from 7.5 to 1000 cycles/hour)

dispout='no' Define to display outputs or not ('yes': display, 'no': not display)

Outputs

S Wind spectrum in ((m/s)^2/Hz)

f Frequency in (Hz)

Examples

```
import scientimate as sm

U10=15
Sigma=1
z=30
L=3.33*z
[S, f] = sm.windspectrum(U10, Sigma, L, 'kaimal_dnv', 0, 1, 'yes')

U10=15
Sigma=1
z=30
Lambda=0.7*z
L==8.1*Lambda
[S, f] = sm.windspectrum(U10, Sigma, L, 'kaimal_iec', 0, 1, 'yes')
```

References

Bhattacharya, S. (2019). Design of foundations for offshore wind turbines. Wiley.

Bec, J. (2010). Influence of wind spectrum formula choice on footbridge response. In 5th international symposium on computational wind engineering (pp. 23-27).

Branlard, E. (2010). Generation of time series from a spectrum. Technical University Denmark. National Laboratory for Sustainable Energy.

Davenport, A. G. (1961). The spectrum of horizontal gustiness near the ground in high winds. Quarterly Journal of the Royal Meteorological Society, 87(372), 194-211.

Harris, R. I. The Nature of Wind, Proc. of the ModernDesign of Wind Sensitive Structures, Construction, Industry Research and Information Association, 1971, London, U. K

Kaimal, J. C., Wyngaard, J. C. J., Izumi, Y., & Coté, O. R. (1972). Spectral characteristics of surface-layer turbulence. Quarterly Journal of the Royal Meteorological Society, 98(417), 563-589.

Rose, S., & Apt, J. (2012). Generating wind time series as a hybrid of measured and simulated data. Wind Energy, 15(5), 699-715.

Udoh, I. E., & Zou, J. (2018). Wind spectral characteristics on strength design of floating offshore wind turbines. Ocean Systems Engineering, 8(3), 281-312.

VERITAS, D. N. (2010). ENVIRONMENTAL CONDITIONS AND ENVIRONMENTAL LOADS.

https://www.mathworks.com/help/aeroblks/wind.html

https://www.mathworks.com/help/aeroblks/vonkarmanwindturbulencemodelcontinuous.html

scientimate.windvelz1toz2

```
Uz2PawLaw, Uz2LogLaw, Uz2LogLawCh = scientimate.windvelz1toz2(Uz1, z1, z2=10, alpha=1/
↪7, z0=0.005, alphaCh=0.0145)
```

Description

Convert wind velocity from first height, z1 (m), to second height, z2 (m), (e.g. 10 (m)) above surface

Inputs

Uz1 Wind velocity at z1 Height in (m/s)

z1 First Height (e.g. sensor height) in (m)

z2=10.0 Second Height that wind velocity is converted to in (m)

alpha=1/7

Exponent of power law velocity profile, Uz1/Uz2=(z1/z2)**(alpha)
water: alpha=1/11.5, open area: alpha=1/9.5, suburban area: alpha=1/7, urban area: alpha=1/5

z0=0.005

Surface roughness in logarithmic velocity profile in (m), U/Ustar=(1/0.41)*ln(z/z0)
water: z0=0.005 (m), open area: z0=0.02 (m), suburban area: z0=0.3 (m), urban area: z0=2 (m)

alphaCh=0.0145

Charnock (1955) parameters to calculate ocean surface roughness, z0=alphaCh*(Ustar^2/g)
Charnock (1955): alphaCh=0.012, Garratt (1977): alphaCh=0.0145, Wu (1980): alphaCh=0.0185

Outputs

Uz2PawLaw

Wind velocity at height of z2 above surface in (m/s)
calculated from power law velocity profile

Uz2LogLaw

Wind velocity at height of z2 above surface in (m/s)
calculated from logarithmic velocity profile

Uz2LogLawCh

Wind velocity at height of z2 above surface in (m/s)
calculated from logarithmic velocity profile

using Charnock (1955) relationship for wind over ocean

Examples

```
import scientimate as sm
import numpy as np

Uz2PawLaw,Uz2LogLaw,Uz2LogLawCh=sm.windvelz1toz2(8,15,10)

Uz1=10.*np.random.rand(100)
Uz2PawLaw,Uz2LogLaw,Uz2LogLawCh=sm.windvelz1toz2(Uz1,15,10,1/7,0.005,0.0145)
```

References

Charnock, H. (1955). Wind stress on a water surface. Quarterly Journal of the Royal Meteorological Society, 81(350), 639-640.

Garratt, J. R. (1977). Review of drag coefficients over oceans and continents. Monthly weather review, 105(7), 915-929.

Wu, J. (1980). Wind-stress coefficients over sea surface near neutral conditionsâ€”A revisit. Journal of Physical Oceanography, 10(5), 727-740.

1.4 Changelog

1.4.1 Version 1.0

What is new in ver 1.0:

- The first version of ScientiMate is released
- Release date: 2020-11-03

1.5 Contribute to ScientiMate

If you have Python (or MATLAB) code(s) that could be beneficial to research community, you can consider to share it with community through ScientiMate.

If you would like your code(s) to be included in future releases of ScientiMate, please email me your code. My contact information could be found at http://www.arashkarimpour.com

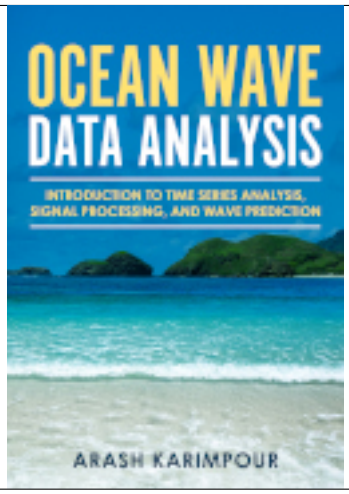
By sending me your code(s) you agree to following terms:

- You developed the entire code(s)
- You are the only owner of the code(s) and you have the full ownership of the code(s)
- There is no restriction in any form for the code(s) to be distributed as an open source code(s)
- You give up all your rights on the code(s)
- ScientiMate developer will decide if the code(s) fits the scope of ScientimMate and if to be included in ScientiMate

- ScientiMate developer will decide if to use entire, part of, or none of the code(s)
- ScientiMate developer can make any change in the code(s) or may use any part of the code(s) in other code(s)
- ScientiMate developer will decide if the code(s) to be included in or be removed from any version ScientiMate

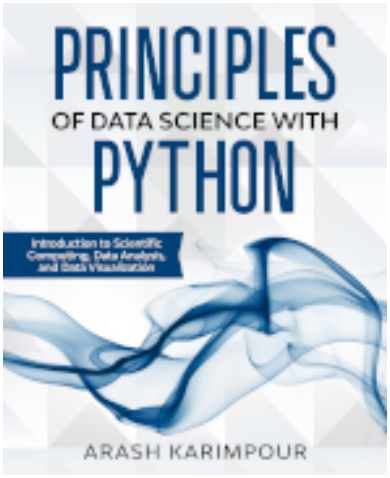
CHAPTER 2

Recommended Books



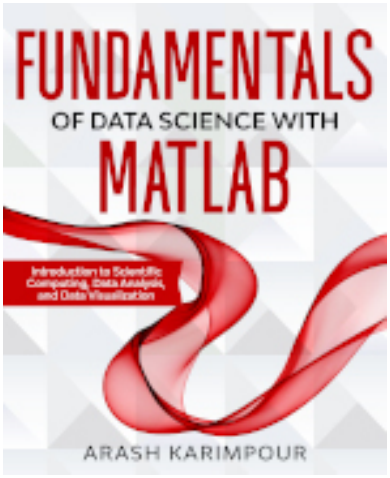
Ocean Wave Data Analysis
Introduction to Time Series Analysis, Signal Processing, and Wave Prediction.

Order at Amazon: https://www.amazon.com/dp/0692109978



Principles of Data Science with Python
Introduction to Scientific Computing, Data Analysis, and Data Visualization.

Order at Amazon: https://www.amazon.com/dp/1735241008



Fundamentals of Data Science with MATLAB
Introduction to Scientific Computing, Data Analysis, and Data Visualization.

Order at Amazon: https://www.amazon.com/dp/1735241016