# OCEANLYZ

*Release 2.0*

Oct 27, 2020

# Contents

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.

This toolbox contains functions that each one is suitable for a particular purpose. Both spectral and zero-crossing methods are offered for wave analysis. This toolbox can calculate wave properties such as zero-moment wave height, significant wave height, mean wave height, peak wave period and mean period. This toolbox can correct and account for the pressure attention (pressure loss) in the water column for data collected by a pressure sensor. This toolbox can separate wind sea and swell energies and reports their properties.

**Name** OCEANLYZ

**Description** Ocean Wave Analyzing Toolbox

**Version** 2.0

**Requirements** MATLAB, or GNU Octave, or Python (3 or later)

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

**Documentation** https://oceanlyz.readthedocs.io

**Tutorial Video** YouTube Playlist

**Source Code** https://github.com/akarimp/oceanlyz

**Report Issues** https://github.com/akarimp/oceanlyz/issues

CHAPTER 1

# User Guide

**Setup and Usage**

## 1.1 Introduction

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.

This toolbox contains functions that each one is suitable for a particular purpose. Both spectral and zero-crossing methods are offered for wave analysis. This toolbox can calculate wave properties such as zero-moment wave height, significant wave height, mean wave height, peak wave period and mean period. This toolbox can correct and account for the pressure attention (pressure loss) in the water column for data collected by a pressure sensor. This toolbox can separate wind sea and swell energies and reports their properties.

### 1.1.1 Selected applications for OCEANLYZ toolbox

- Calculate wave properties from field/lab measured data.
- Use spectral analysis and zero-crossing method for wave analyzing.
- Partition wave spectra and separate wind-sea and swell data.
- Correct pressure data that read by a pressure sensor to account for pressure attenuation in water depth.
- replace spectrum tail with tail of empirical spectrum (diagnostic tail).

### 1.1.2 Selected parameters that can be calculated with OCEANLYZ toolbox

**Wave Height and Spectrum**

- Zero-Moment Wave Height
- Sea/Swell Wave Height

- Significant Wave Height
- Mean Wave Height
- Wave Power Spectral Density

**Wave Period**

- Peak Wave Frequency
- Peak Wave Period
- Peak Sea/Swell Period
- Mean Wave Period
- Significant Wave Period

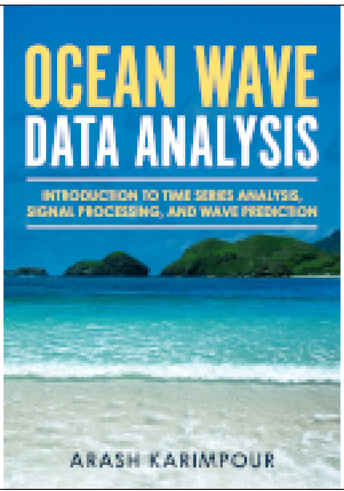
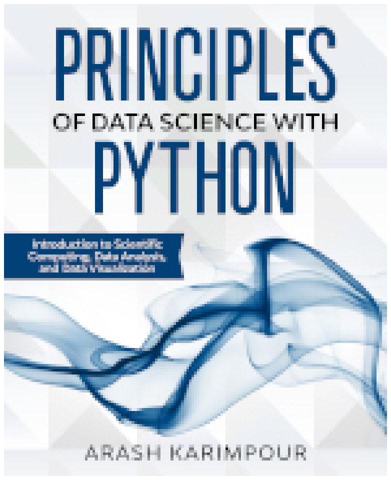
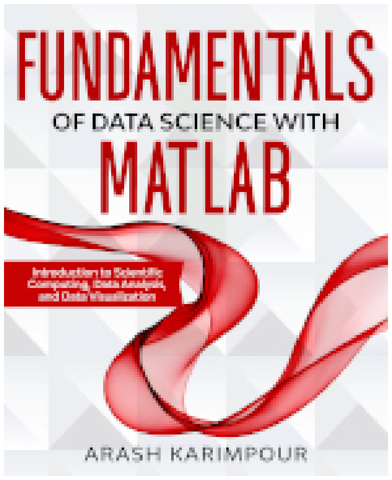
### 1.1.3 Citation

Cite this toolbox as:

Karimpour, A., & Chen, Q. (2017). Wind Wave Analysis in Depth Limited Water Using OCEANLYZ, a MATLAB toolbox. Computers & Geosciences.

Link: https://www.sciencedirect.com/science/article/pii/S0098300417306489

### 1.1.4 Recommended Book

| | | |
|---|---|---|
|   OCEAN WAVE DATA ANALYSIS<br>INTRODUCTION TO TIME SERIES ANALYSIS, SIGNAL PROCESSING, AND WAVE PREDICTION<br>ARASH KARIMPOUR |   PRINCIPLES OF DATA SCIENCE WITH PYTHON<br>Introduction to Scientific Computing, Data Analysis, and Data Visualization<br>ARASH KARIMPOUR |   FUNDAMENTALS OF DATA SCIENCE WITH MATLAB<br>Introduction to Scientific Computing, Data Analysis, and Data Visualization<br>ARASH KARIMPOUR |
| **Ocean Wave Data Analysis** Introduction to Time Series Analysis, Signal Processing, and Wave Prediction.<br><br>Order at Amazon: https://www.amazon.com/dp/0692109978 | **Principles of Data Science with Python** Introduction to Scientific Computing, Data Analysis, and Data Visualization.<br><br>Order at Amazon: https://www.amazon.com/dp/1735241008 | **Fundamentals of Data Science with MATLAB** Introduction to Scientific Computing, Data Analysis, and Data Visualization.<br><br>Order at Amazon: https://www.amazon.com/dp/1735241016 |

### 1.1.5 License Agreement and Disclaimer

## 1.2 Getting Started (MATLAB Version)

In order to use MATLAB version of OCEANLYZ toolbox, first, one of the MATLAB or GNU Octave programming language along with required packages should be installed (Refer to required packages for MATLAB/GNU Octave).

### 1.2.1 Installation

To use MATLAB version of OCEANLYZ toolbox:

- Install MATLAB or GNU Octave
- Download OCEANLYZ:

  GitHub: https://github.com/akarimp/oceanlyz/releases/download/2.0/oceanlyz_2_0.zip
  CNET: https://download.cnet.com/Oceanlyz/3000-2054_4-75833686.html

- Unzip OCEANLYZ in any location you choose such as “C:\”

One way to access OCEANLYZ is to copy OCEANLYZ files and its sub-folders in your desire working directory and then use it there. Another option is to add OCEANLYZ folder to MATLAB path. By doing that, you always have access to OCEANLYZ from any working directory. For example, if OCEANLYZ files are in “C:\oceanlyz” folder then it can be added to path as:

```
OceanlyzPath=genpath('C:\oceanlyz'); %Generating path for OCEANLYZ folder and its sub-
↪folders
addpath(OceanlyzPath);
```

To remove OCEANLYZ from the path use:

```
rmpath(OceanlyzPath);
```

### 1.2.2 Operating System

This code can be run on Windows, Mac and Linux. However, make sure any given path is compatible with a running operating system. In particular, "\" is used in Windows path, while "/" is used in Mac or Linux path. For example, if a path is "C:\" on Windows machine, it would be "C:/" on Mac or Linux.

### 1.2.3 Required Programing Language

The MATLAB version of this toolbox can be run by using MATLAB (https://www.mathworks.com) or GNU Octave (https://www.gnu.org/software/octave).

### 1.2.4 Required Package for MATLAB

MATLAB users need to install MATLAB Signal Processing Toolbox (https://www.mathworks.com/products/signal.html) for running the OCEANLYZ spectral analysis. It gives OCEANLYZ access to MATLAB Welch's power spectral density calculation. However, MATLAB Signal Processing Toolbox it is not required for zero-crossing analysis.

### 1.2.5 Required Package for GNU Octave

GNU Octave users need to install/load GNU Octave Signal package (https://octave.sourceforge.io/signal/index.html) for running the OCEANLYZ spectral analysis. It gives OCEANLYZ access to GNU Octave Welch's power spectral density calculation. However, GNU Octave Signal package it is not required for zero-crossing analysis.

GNU Octave comes with Signal package but it needs to loaded every time GNU Octave starts. The Signal package can be loaded inside GNU Octave by using a following command in a command window (This should be done every time GNU Octave is opened):

```
>> pkg load signal
```

If GNU Octave Signal Package is not already installed, it should be first installed from Octave Forge (https://octave.sourceforge.io), and then get loaded by using the following commands in a command window:

```
>> pkg install -forge signal
>> pkg load signal
```

### 1.2.6 Quick Start

- Open MATLAB or GNU Octave
- Change a current folder (working directory) to a folder that contains OCEANLYZ files, for example "C:\oceanlyz", in MATLAB or GNU Octave.
- Create OCEANLYZ object such as "ocn=oceanlyz" in MATLAB or GNU Octave and set/modify its properties based on the dataset and required analysis.
- Run a method as "ocn.runoceanlyz()" in MATLAB or GNU Octave to start calculations.

### 1.2.7 Download Previous Version

Download previous MATLAB version of OCEANLYZ toolbox

- Ver 1.5: https://github.com/akarimp/oceanlyz/releases/download/1.5/oceanlyz_1_5.zip

- Ver 1.4: https://github.com/akarimp/oceanlyz/releases/download/1.5/oceanlyz_1_4.zip

## 1.3 Getting Started (Python Version)

The Python version of OCEANLYZ toolbox is part of ScintiMate package (https://scientimate.readthedocs.io).

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

### 1.3.1 Installation

To use Python version of OCEANLYZ toolbox:

- Install Python
- Install ScientiMate (Python version of OCEANLYZ toolbox is part of ScintiMate package)

**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
```

### 1.3.2 Operating System

This code can be run on Windows, Mac, and Linux. However, make sure any given path is compatible with a running operating system. In particular, "\" is used in Windows path, while "/" is used in Mac or Linux path. For example, if a path is "C:\" on Windows machine, it would be "C:/" on Mac or Linux.

### 1.3.3 Required Programing Language

The Python version of this toolbox can be run by using Python 3 or later (https://www.python.org or https://www.anaconda.com).

### 1.3.4 Required Package for Python

Following packages are required:

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

### 1.3.5 Quick Start

- Open Python
- Import ScientiMate package by using "import scientimate as sm"
- Create OCEANLYZ object such as "ocn=sm.oceanlyz()" in Python and set/modify its properties based on the dataset and required analysis.
- Run a method as "ocn.runoceanlyz()" in Python to start calculations.

**Functions Reference**

## 1.4 API

Here is a list of the OCEANLYZ functions:

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.

### 1.4.1 Functions Description

#### Functions Description

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.

OCEANLYZ requires time-series data which can be in a form of the water depth, water surface elevation, or water pressure. If a pressure sensor is used for data collection, then data should be corrected for pressure attenuation in the water column. In such cases, only pressure data should be used as an input. In any other cases, either of water depth or water surface elevation time series can be used as an input.

The main class to Run OCEANLYZ toolbox is:

| Class | Description |
|---|---|
| ocean-lyz | Calculate wave properties from water level or water pressure data (This the main class that runs OCEANLYZ) |

The functions that are used in OCEANLYZ toolbox are:

| Function | Description |
|---|---|
| PcorFFTFun | Corrects water depth data for pressure attenuation effect using spectral analysis |
| PcorZerocrossing-Fun | Corrects water depth data for pressure attenuation effect using zero-crossing |
| SeaSwellFun | Partition (separate) wind sea from swell in a power spectral density using an one dimensional method |
| WaveSpectraFun | Calculates wave properties from water surface elevation using spectral analysis |
| WaveZerocrossing-Fun | Calculates wave properties from water surface elevation using zero-crossing |

### 1.4.2 Functions (MATLAB Version)

#### oceanlyz (MATLAB Version)

```
oceanlyz_object = oceanlyz
```

##### DESCRIPTION

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

##### Required Properties

Following properties are required for all analysis

**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 structure array**

Output is a structure array
Name of output is 'oceanlyz_object.wave'
Field(s) in this structure array can be called by using '.'
Example:

oceanlyz_object.wave.Hm0 : Contain zero-moment wave height
oceanlyz_object.wave.Tp : Contain peak wave period

oceanlyz_object.wave.Field_Names : Contain field (variable) names in the wave array
oceanlyz_object.wave.Burst_Data : Contain data for each burst

### Examples

```
%Change current working directory to OCEANLYZ folder
%Assume OCEANLYZ files are in 'C:\oceanlyz_matlab' folder
cd('C:\oceanlyz_matlab')

%Create OCEANLYZ object
%clear ocn %Optional
ocn=oceanlyz;

%Read data
%Assume data file is named 'waterpressure_5burst.csv' and is stored in 'C:\oceanlyz_
↪matlab\Sample_Data'
current_folder=pwd;                     %Current (OCEANLYZ) path
cd('C:\oceanlyz_matlab\Sample_Data') %Change current path to Sample_Data folder
water_pressure=importdata('waterpressure_5burst.csv'); %Load data
cd(current_folder)                      %Change current path to OCEANLYZ folder

%Input parameters
ocn.data=water_pressure;
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)
plot(ocn.wave.Tp(1,:))
```

### References

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

### PcorFFTFun

```
[Eta,ftailcorrection]=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')
```

### PcorZerocrossingFun

```
[Eta]=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');
```

### SeaSwellFun

```
[Hm0,Hm0sea,Hm0swell,Tp,Tpsea,Tpswell,fp,fseparation,f,Syy]=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')
```

### WaveSpectraFun

```
[Hm0,Tm01,Tm02,Tp,fp,f,Syy]=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')
```

### WaveZerocrossingFun

```
[Hs,Hz,Tz,Ts,H,T]=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')
```

## 1.4.3 Functions (Python Version)

The Python version of OCEANLYZ toolbox is a part of the ScintiMate package.
For more information, visit https://scientimate.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
import os

#Create OCEANLYZ object
#del ocn #Optional
```

(continues on next page)

(continued from previous page)

```
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')
```

## 1.5 Changelog

### 1.5.1 Version 2.0

What is new in ver 2.0:

- From version 2.0, in addition to MATLAB and GNU Octave, OCEANLYZ is available for Python language through ScientiMate package
- From version 2.0, OCEANLYZ is a class (instead of function). It allows OCEANLYZ to be called inside user code.
- Release date: 2020-10-27

### 1.5.2 Version 1.5

What is new in ver 1.5:

- Now, a wave spectrum in PcorFFTFun function is obtained from pwelch function instead of fft function
- Parameter 'burst' changed to 'n_burst'
- Parameter 'duration' changed to 'burst_duration'
- Release date: 2020-8-5

### 1.5.3 Version 1.4

What is new in ver 1.4:

- Now, OCEANLYZ can be run on both Matlab and GNU Octave
- Now, a separate input file is used to define calculation parameters
- Wavenumber calculation performance is improved
- TMA diagnostic tail is improved
- User manual is re-written
- Release date: 2018-6-22

### 1.5.4 Version 1.3

What is new in ver 1.3:

- Tail correction for spectral analysis is improved
- Correcting pressure data for pressure attenuation is improved
- Sea/Swell partitioning is improved
- Release date: 2017-6-27

### 1.5.5 Version 1.2

What is new in ver 1.2:

- The NFFT now can be assigned as an input.
- Automatic calculation of the upper limit frequency for the dynamic pressure corrections added.
- Zero-Crossing method modified.
- Mean water level calculation modified.
- Significant wave period added.
- Calculation of peak wave frequency based on the weighted integral added.
- Parameter notations are updated.
- Release date: 2014-12

### 1.5.6 Version 1.1

What is new in ver 1.1:

- Initial version of OCEANLYZ is released
- Release date: 2013-12

**Example**

## 1.6 Example (MATLAB Version)

Here is an example shows how to use OCEANLYZ. In this example, we use a provided sample file "waterpressure_5burst.csv" as input data. This sample file contains five bursts of water pressure data recorded with a pressure sensor. Sample file may be downloaded at https://github.com/akarimp/oceanlyz/releases/tag/2.0 .

Measurement properties for "waterpressure_5burst.csv" are:

| Properties | Value | OCEANLYZ Properties |
|---|---|---|
| File name | waterpressure_5burst.csv | |
| Data type | Water pressure | obj.InputType='pressure' |
| Number of recorded burst (n_burst) | 5 | obj.n_burst=5 |
| Sampling frequency (fs) | 10 (Hz) | obj.fs=10 |
| Recording duration (burst_duration) | 1024 (second) | obj.burst_duration=1024 |
| Pressure sensor height from bed (heightfrombed) | 0.05 (m) | obj.heightfrombed=0.05 |
| Mean water depth (h) | Varies in each burst | |

To start using OCEANLYZ, first, we need to be in a folder that contains OCEANLYZ files. Assume OCEANLYZ files are in 'C:\oceanlyz_matlab'. First, we change current working directory to OCEANLYZ folder as:

```
cd('C:\oceanlyz_matlab') %Change current working directory to OCEANLYZ folder
```

Next, we download water pressure dataset ("waterpressure_5burst.csv"), we unzip it and copy sample files in a desired folder.

Assume we are currently in OCEANLYZ folder 'C:\oceanlyz_matlab' and downloaded sample data file is stored in 'C:\oceanlyz_matlab\Sample_Data'. Then, we load data as:

```
current_folder = pwd;                    %Current (OCEANLYZ) path
cd('C:\oceanlyz_matlab\Sample_Data') %Change current folder to a folder that contains␣
↪data file
water_pressure = importdata('waterpressure_5burst.csv'); %Load data
cd(current_folder)                       %Change current folder back to initial␣
↪(OCEANLYZ) folder
```

We can plot data if we need to as:

```
plot(water_pressure)
xlabel('Sample points')
ylabel('Water Pressure (N/m^2)')
```

Then, we need to create an OCEANLYZ object as:

```
%Create OCEANLYZ object
ocn = oceanlyz;
```

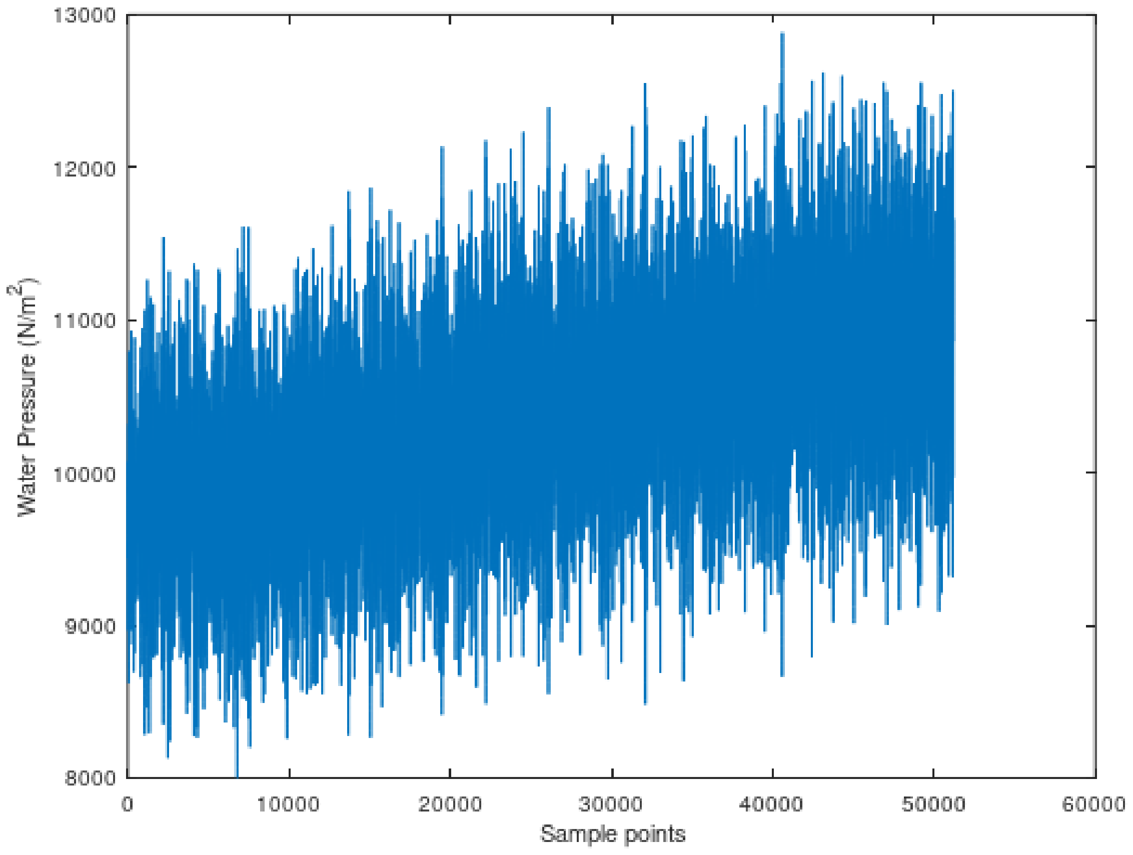


Fig. 1: Figure 1: Plot of input water pressure data

Next, we assign wave data to OCEANLYZ object as:

```
%Input data
ocn.data = water_pressure;
```

Now, we set up OCEANLYZ properties as:

```
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;
ocn.fmax=ocn.fs/2;
ocn.fmaxpcorrCalcMethod='auto';    %Only required if ocn.InputType='pressure'
ocn.Kpafterfmaxpcorr='constant';   %Only required if ocn.InputType='pressure'
ocn.fminpcorr=0.15;                %Only required if ocn.InputType='pressure'
ocn.fmaxpcorr=0.55;                %Only required if ocn.InputType='pressure'
ocn.heightfrombed=0.05;            %Only required if ocn.InputType='pressure'
ocn.dispout='yes';
ocn.Rho=1024;                      %Seawater density (Varies)
```

After all required properties are set, we can run OCEANLYZ as:

```
ocn.runoceanlyz()
```

Output is stored as a structure array. Name of output is 'oceanlyz_object.wave'. Field(s) in this structure array can be called by using '.' For example oceanlyz_object.wave.Hm0 contains zero-moment wave height and oceanlyz_object.wave.Tp contains peak wave period.

Here we show how to plot zero-moment wave height:

```
Hm0 = ocn.wave.Hm0; %zero-moment wave height
plot(Hm0)
xlabel('Burst Number')
ylabel('Hm0 (m)')
```

Similarly, we can plot wave spectrum for the first burst:

```
f = ocn.wave.f; %frequency of the first burst
Syy = ocn.wave.Syy; %spectrum of the first burst
plot(f(1,:),Syy(1,:))
xlabel('f (Hz)')
ylabel('Syy (m^2/Hz)')
```

### 1.6.1 Notes

**Note1:** If data are collected in continuous mode and you need to analyze them in smaller blocks, you can analyze it in a burst mode. For that, you choose n_burst and burst_duration as follow:

The burst_duration is equal to a period of time that you want data analyzed over that. For example, if you need wave properties reported every 15 min, then the burst_duration would be 15*60 second.

the n_burst is equal to the total length of the time series divided by the burst_duration. The n_burst should be an integer. So, if the total length of the time series divided by the burst_duration leads to a decimal number, then data should be shortened to avoid that.

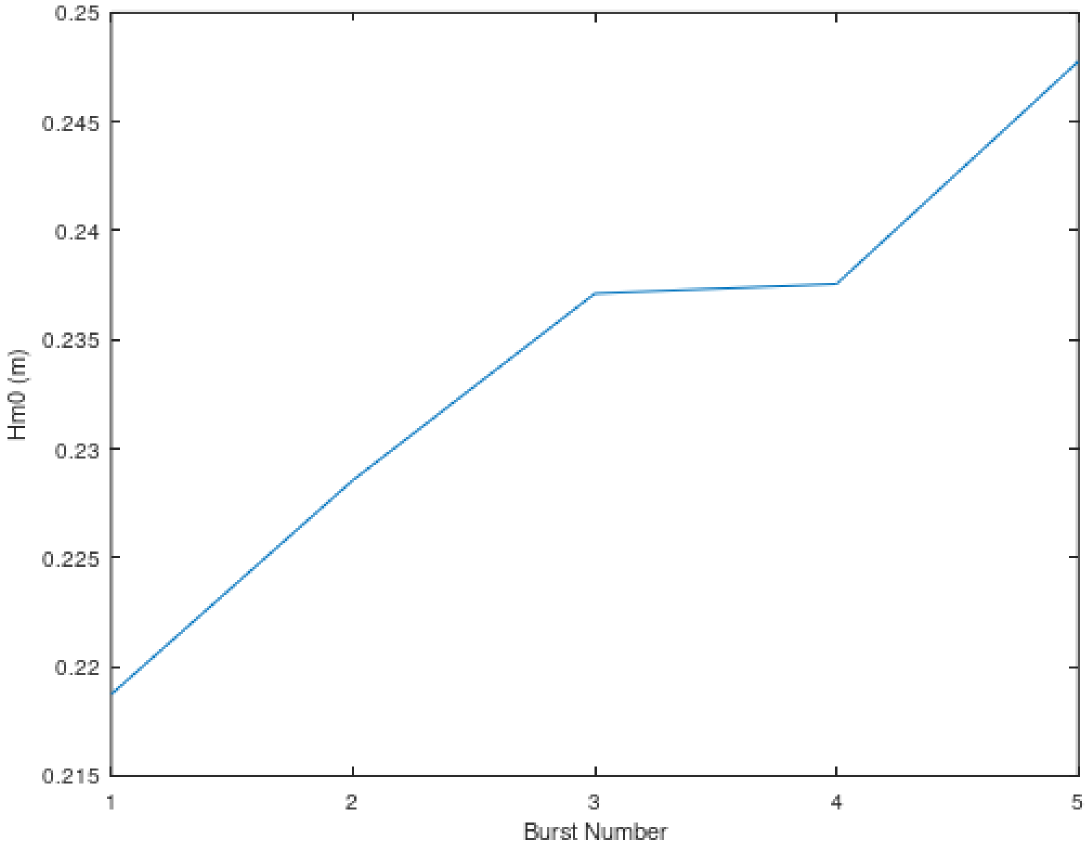


Fig. 2: Figure 2: Plot of $H_{m0}$ versus burst number

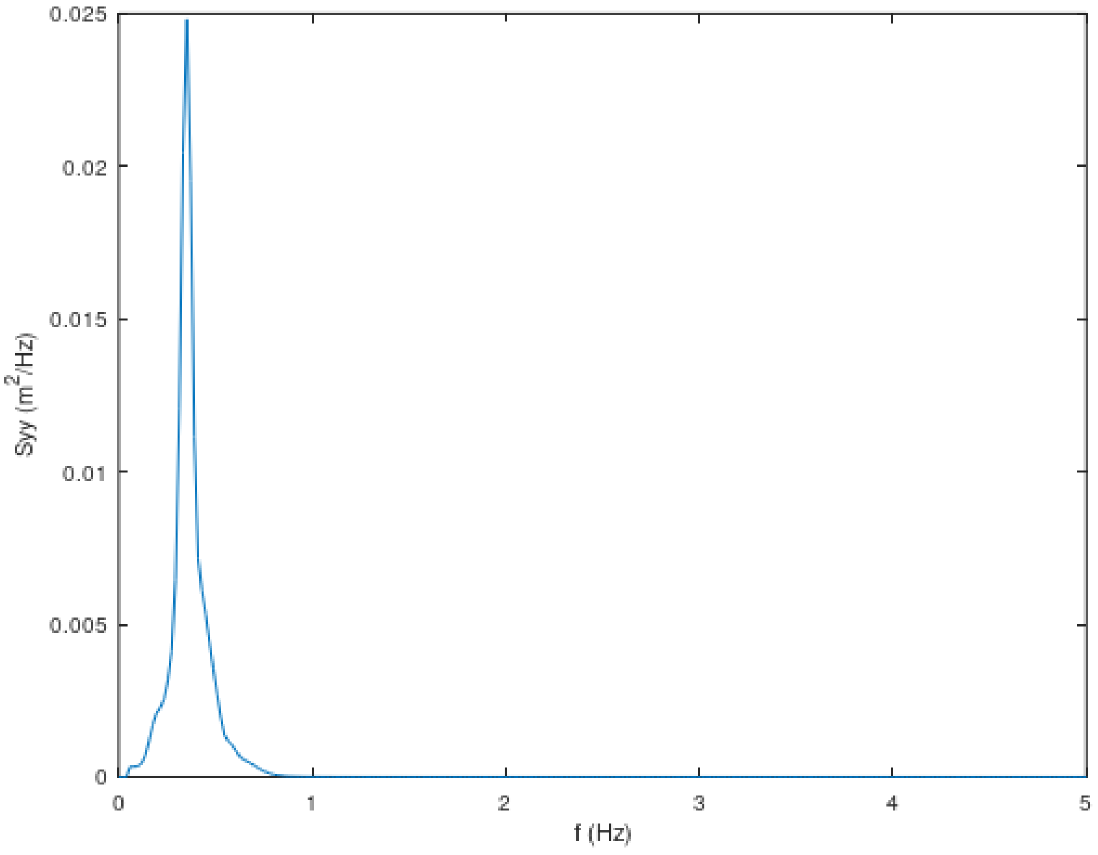


Fig. 3: Figure 3: Plot of $S_{yy}$ versus f

**Note2:** Welch spectrum is used to calculate a power spectral density. In all spectral calculation, a default window function with a default overlap window between segments are used.

**Note3:** If fmaxpcorrCalcMethod='auto', then OCEANLYZ calculates fmaxpcorr based on water depth and a sensor height from a seabed (refer to Applying Pressure Response Factor section). A maximum value for calculated fmaxpcorr will be limited to the value user set for fmaxpcorr.

## 1.7 Example (Python Version)

Here is an example shows how to use OCEANLYZ. In this example, we use a provided sample file "waterpressure_5burst.csv" as input data. This sample file contains five bursts of water pressure data recorded with a pressure sensor. Sample file may be downloaded at https://github.com/akarimp/oceanlyz/releases/tag/2.0 .

Measurement properties for "waterpressure_5burst.csv" are:

| Properties | Value | OCEANLYZ Properties |
|---|---|---|
| File name | waterpressure_5burst.csv | |
| Data type | Water pressure | obj.InputType='pressure' |
| Number of recorded burst (n_burst) | 5 | obj.n_burst=5 |
| Sampling frequency (fs) | 10 (Hz) | obj.fs=10 |
| Recording duration (burst_duration) | 1024 (second) | obj.burst_duration=1024 |
| Pressure sensor height from bed (heightfrombed) | 0.05 (m) | obj.heightfrombed=0.05 |
| Mean water depth (h) | Varies in each burst | |

To start using OCEANLYZ, first, we need to be import required libraries. Remember, Python version of OCEANLYZ is part of ScientiMate, and therefore, we need to import ScientiMate to use OCEANLYZ:

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

Next, we download water pressure dataset ("waterpressure_5burst.csv"), we unzip it and copy sample files in a desired folder.

Assume downloaded sample data file is stored in 'C:\oceanlyz_python\Sample_Data'. Then, we load data as:

```
os.chdir('C:\\oceanlyz_python\\Sample_Data') #Change current folder to a folder that␣
↪contains data file
water_pressure = np.genfromtxt('waterpressure_5burst.csv') #Load data
```

We can plot data if we need to as:

```
plt.plot(water_pressure)
plt.xlabel('Sample points')
plt.ylabel('Water Pressure (N/m^2)')
```

Then, we need to create an OCEANLYZ object as:

```
#Create OCEANLYZ object
ocn = sm.oceanlyz()
```

Next, we assign wave data to OCEANLYZ object as:

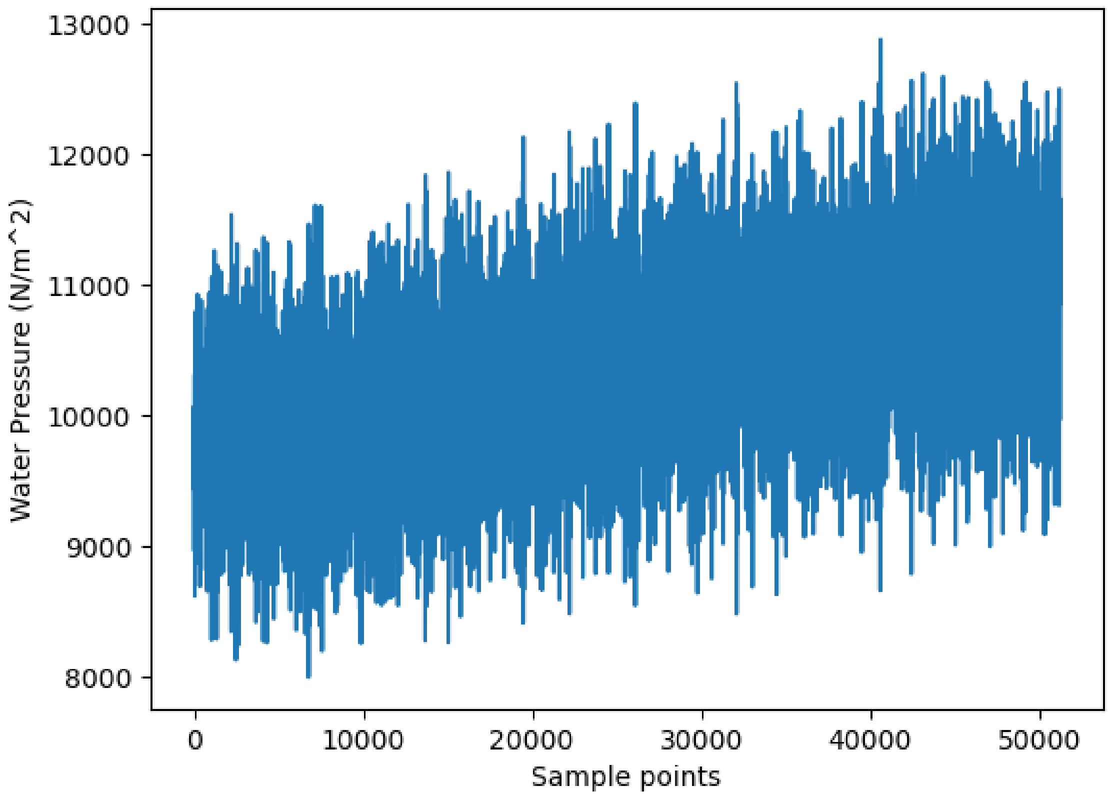


Fig. 4: Figure 1: Plot of input water pressure data

```
#Input data
ocn.data = water_pressure.copy()
```

Now, we set up OCEANLYZ properties as:

```
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
ocn.fmax=ocn.fs/2
ocn.fmaxpcorrCalcMethod='auto'    #Only required if ocn.InputType='pressure'
ocn.Kpafterfmaxpcorr='constant'   #Only required if ocn.InputType='pressure'
ocn.fminpcorr=0.15                #Only required if ocn.InputType='pressure'
ocn.fmaxpcorr=0.55                #Only required if ocn.InputType='pressure'
ocn.heightfrombed=0.05            #Only required if ocn.InputType='pressure'
ocn.dispout='yes'
ocn.Rho=1024                      #Seawater density (Varies)
```

After all required properties are set, we can run OCEANLYZ as:

```
ocn.runoceanlyz()
```

Output is stored as a structure array. Name of output is 'oceanlyz_object.wave'. Field(s) in this structure array can be called by using '.' For example oceanlyz_object.wave.Hm0 contains zero-moment wave height and oceanlyz_object.wave.Tp contains peak wave period.

Here we show how to plot zero-moment wave height:

```
Hm0 = ocn.wave['Hm0'] #zero-moment wave height
plt.plot(Hm0)
plt.xlabel('Burst Number')
plt.ylabel('Hm0 (m)')
```

Similarly, we can plot wave spectrum for the first burst:

```
f = ocn.wave['f'] #frequency of the first burst
Syy = ocn.wave['Syy'] #spectrum of the first burst
plt.plot(f[0,:],Syy[0,:])
plt.xlabel('f (Hz)')
plt.ylabel('Syy (m^2/Hz)')
```

### 1.7.1 Notes

**Note1:** If data are collected in continuous mode and you need to analyze them in smaller blocks, you can analyze it in a burst mode. For that, you choose n_burst and burst_duration as follow:

The burst_duration is equal to a period of time that you want data analyzed over that. For example, if you need wave properties reported every 15 min, then the burst_duration would be 15*60 second.

the n_burst is equal to the total length of the time series divided by the burst_duration. The n_burst should be an integer. So, if the total length of the time series divided by the burst_duration leads to a decimal number, then data should be shortened to avoid that.

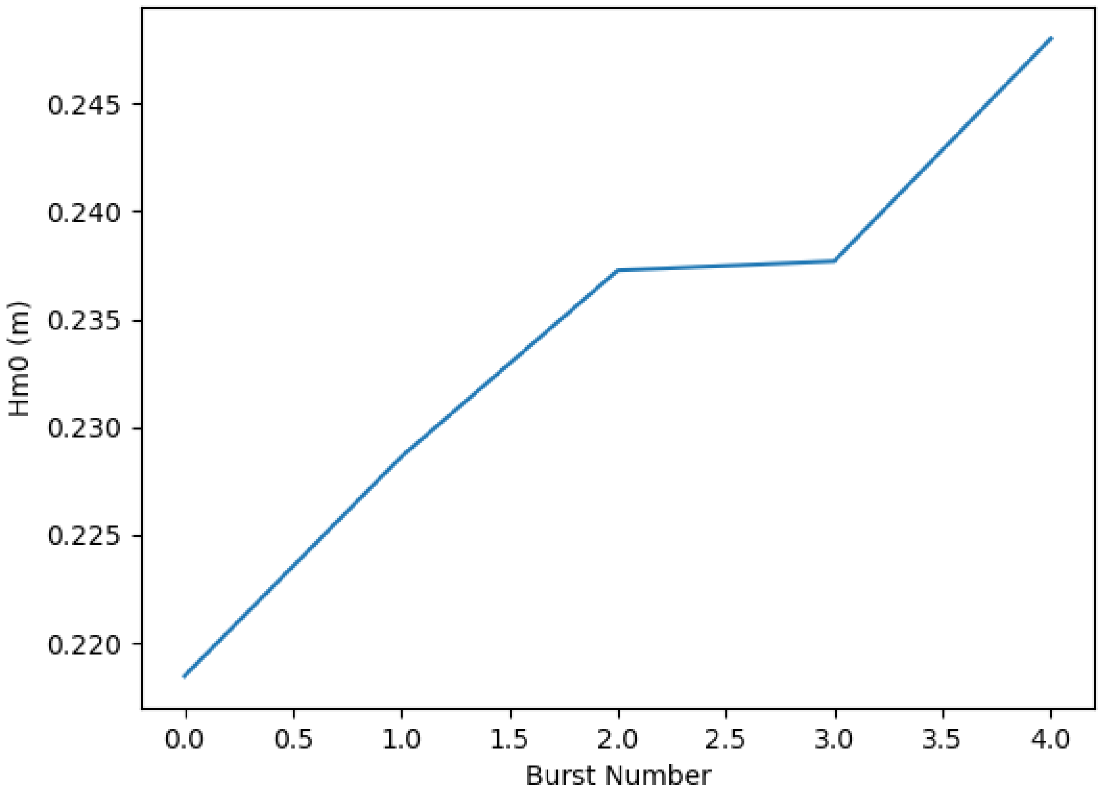


Fig. 5: Figure 2: Plot of $H_{m0}$ versus burst number

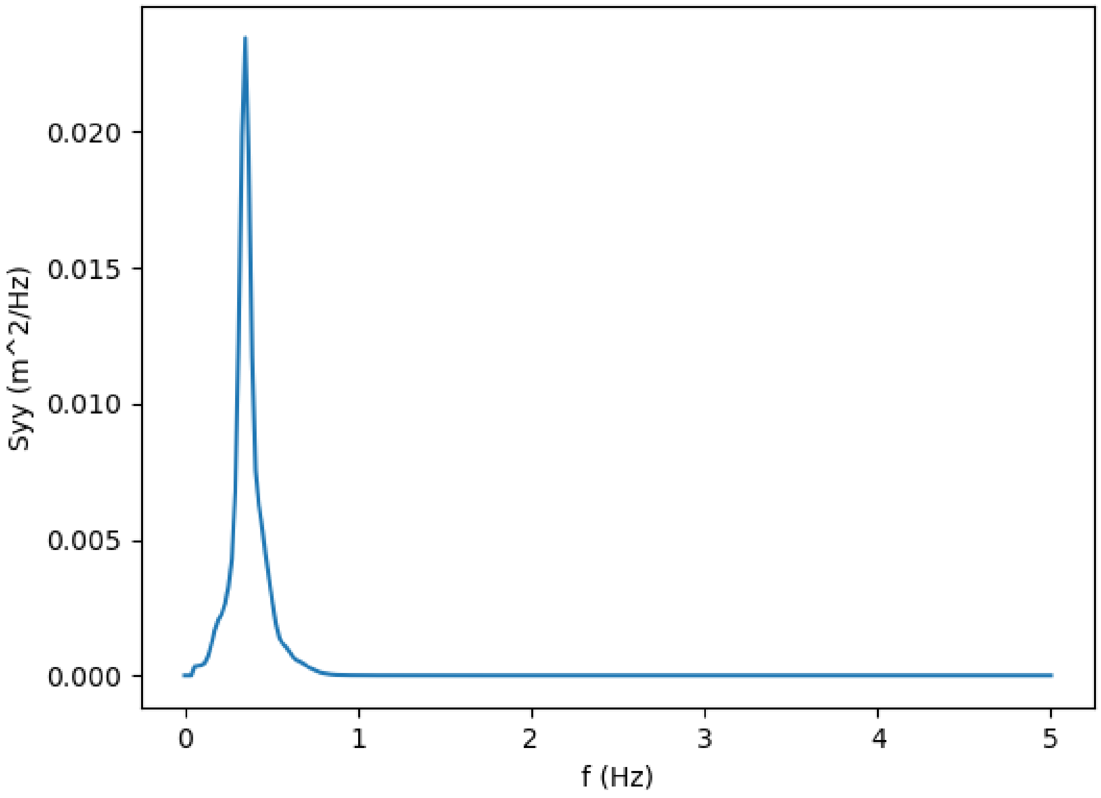


Fig. 6: Figure 3: Plot of $S_{yy}$ versus f

**Note2:** Welch spectrum is used to calculate a power spectral density. In all spectral calculation, a default window function with a default overlap window between segments are used.

**Note3:** If fmaxpcorrCalcMethod='auto', then OCEANLYZ calculates fmaxpcorr based on water depth and a sensor height from a seabed (refer to Applying Pressure Response Factor section). A maximum value for calculated fmaxpcorr will be limited to the value user set for fmaxpcorr.

**Prepare Data File**

## 1.8 Prepare Input Data File

This section explains about a format of input data that is being used in a calculation. Input data may contains a time-series of water surface elevation, water depth, or water pressure.

For MATLAB version of toolbox, input data should be prepared as a 1-d vector (column array with only one column).

For Python version of toolbox, input data should be prepared as a 1-d array.

Input data should not contain any text. The schematics of input data for data recorded in a burst mode and a continuous mode are demonstrated in following sections.

### 1.8.1 Data Recorded in Burst Mode

If data are recorded in a burst mode, it means that data are collected for a specific duration (burst), then followed by idle duration (no data are collected), then followed by another recording period, and so on. In the burst measurement, this pattern of recording/not recording is repeated throughout the measurement. This usually is used in cases that we need to save battery and memory.

For example, an instrument may set to record data for 20 minutes and then goes to sleep to save the battery and memory for 40 minutes, before it wakes up again and repeats the recording/sleeping procedure again and again.

Assuming we have M bursts of data, and each burst contains N data points, then a schematic of input data array is:

| | |
|---|---|
| Burst 1, data point 1 | |
| Burst 1, data point 2 | |
| Burst 1, data point 3 | |
| … | |
| … | |
| … | |
| Burst 1, data point N-2 | |
| Burst 1, data point N-1 | |
| Burst 1, data point N | |
| Burst 2, data point 1 | |
| Burst 2, data point 2 | |
| Burst 2, data point 3 | |
| … | |
| … | |
| … | |
| Burst 2, data point N-2 | |
| Burst 2, data point N-1 | |
| Burst 2, data point N | |
| … | |

Continued on next page

Table 1 – continued from previous page

| | |
|---|---|
| … | |
| … | |
| Burst M, data point 1 | |
| Burst M, data point 2 | |
| Burst M, data point 3 | |
| … | |
| … | |
| … | |
| Burst M, data point N-2 | |
| Burst M, data point N-1 | |
| Burst M, data point N | |

In the burst mode:

- M : The total number of bursts
- N : The total number of samples in each burst
- d : The duration of each burst in seconds
- fs: The sampling frequency that data are collected

The total number of samples in each burst (N) is:

$$N = fs * d$$

The total number of samples in input file is:

$$fs * d * M$$

And an input data array has a size of:

$$(fs * d * M, 1)$$

### 1.8.2 Data Recorded in Continuous Mode:

If data are recorded in a continuous mode, it mean that data are recorded in one run without any gap throughout the measurement.

If the total number of recorded data points is equal to N, then a schematic of input data array is:

| | |
|---|---|
| data point 1 | |
| data point 2 | |
| data point 3 | |
| … | |
| … | |
| … | |
| data point N-2 | |
| data point N-1 | |
| data point N | |

In the burst mode:

- N : The total number of samples in dataset
- d : The duration of measurement in seconds
- fs: The sampling frequency that data are collected

The total number of samples in input file (N) is:

$$N = fs * d$$

And an input data array has a size of:

$$(fs * d, 1)$$

### 1.8.3 Analyze Continuous Data as Burst Data:

If data in a continuous mode are collected for a long period, then we may need to split it into several blocks data and analyze each block separately. In such cases, we may consider each of these data block as one burst.

To define a number of bursts when data are collected in a continuous mode, first, we need to define a length (duration) of each burst (data block). Then, the total number of bursts is obtained by dividing the total duration that data are collected by a duration one burst.

For example, consider data that were recorded continuously for 24 hours. Now, assume we want to split data into blocks of 30 minutes. Here, each 30 minutes of data is one burst. Now, the total number of bursts in this case is equal to the 24 hours divided by 0.5 hour (30 minutes), i.e. (24 / 0.5)=48. In this example, a duration of each burst is (30*60) seconds and therefore the total number of samples in each burst is (fs * 30 * 60).

## 1.9 Sample Data Files

Two sample data files are provided to be used with OCEANLYZ. The first file is named "waterpressure_1burst.csv", and contains one burst of water pressure data recorded with a pressure sensor. The second file is named "waterpressure_5burst.csv", and contains five bursts of water pressure data recorded with a pressure sensor. Data in either of these two files can be used as input data for OCEANLYZ.

### 1.9.1 Download

Download sample data files:

- waterpressure_1burst.csv: https://github.com/akarimp/oceanlyz/releases/download/2.0/waterpressure_1burst.csv
- waterpressure_5burst.csv: https://github.com/akarimp/oceanlyz/releases/download/2.0/waterpressure_5burst.csv

### 1.9.2 Properties

Measurement properties for a file named "waterpressure_1burst.csv" are:

| Properties | Value | OCEANLYZ Properties |
|---|---|---|
| File name | waterpressure_1burst.csv | |
| Data type | Water pressure | obj.InputType='pressure' |
| Number of recorded burst (n_burst) | 1 | obj.n_burst=1 |
| Sampling frequency (fs) | 10 (Hz) | obj.fs=10 |
| Recording duration (burst_duration) | 1024 (second) | obj.burst_duration=1024 |
| Pressure sensor height from bed (heightfrombed) | 0.05 (m) | obj.heightfrombed=0.05 |
| Mean water depth (h) | 1.07 (m) | |

Measurement properties for a file named "waterpressure_5burst.csv" are:

| Properties | Value | OCEANLYZ Properties |
|---|---|---|
| File name | waterpressure_5burst.csv | |
| Data type | Water pressure | obj.InputType='pressure' |
| Number of recorded burst (n_burst) | 5 | obj.n_burst=5 |
| Sampling frequency (fs) | 10 (Hz) | obj.fs=10 |
| Recording duration (burst_duration) | 1024 (second) | obj.burst_duration=1024 |
| Pressure sensor height from bed (heightfrombed) | 0.05 (m) | obj.heightfrombed=0.05 |
| Mean water depth (h) | Varies in each burst | |

**Technical Notes**

## 1.10 Correct Data Measured by Pressure Sensor

Dynamic pressure from water wave will attenuate by moving from a water surface toward a seabed. Because of that, data collected by a pressure sensor have lower magnitudes compared to actual pressure values. To fix this issue and account for the pressure loss in the water column, pressure data collected by a pressure sensor should be divided by a pressure response factor. To do that, first, pressure data should be converted to water depth as:

$$h_{sensor} = \frac{P}{\rho g} + d_s$$

Then, data are corrected as:

$$h = \frac{h_sensor}{K_P}$$

$$K_P = \frac{\cosh(kd_s)}{\cosh(kh_m)}$$

where, $h_sensor$ is water depth read by a pressure sensor, P is pressure, $K_P$ is a pressure response factor, k is a wavenumber, $h_m$ is a mean water depth, and d_s is a distance of the sensor location from a seabed.

One method to use these equations to correct the data is to use them on every single wave. To do that, a zero-crossing method is used to calculate $K_P$ for every single wave. Then, $K_P$ can be applied to each wave height or on water level time series corresponded to each wave.

Another method to use these equations to correct the data is to use Fast Fourier transform and convert a time series to the frequency domain, and then applying $K_P$ on each frequency. In that case, calculated $K_P$ in the frequency domain decreases from 1 to 0 as frequency, f, increases (Figure 1). It means that, a value of $1/K_P$ will be 1 for f=0, and increases toward infinity as a frequency increases. Because of that, if $K_P$ applies to an entire frequency domain, it results in will have very large values in the high frequency region. To avoid that, lower and upper limits for frequencies should be considered, where $K_P$ is not applied to raw data outside of that range.

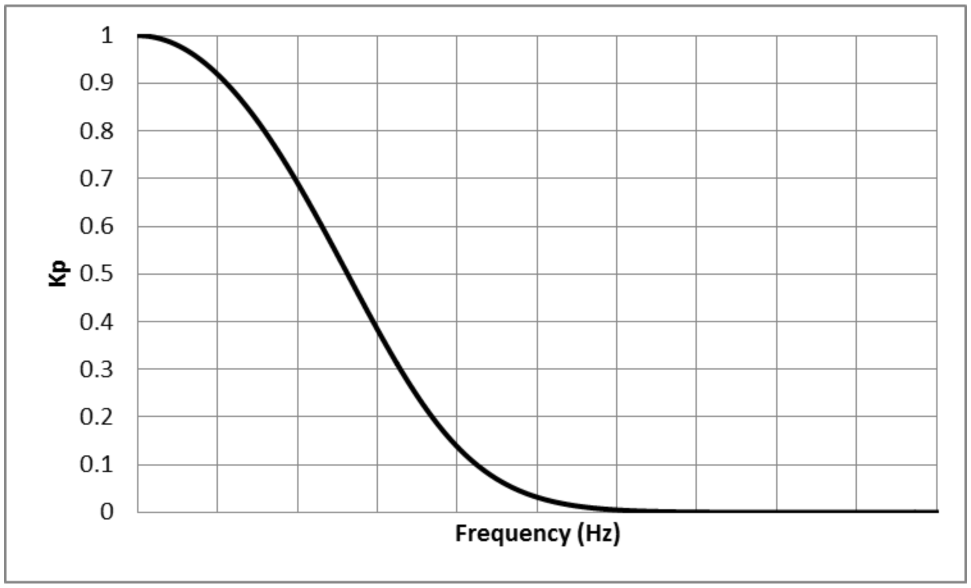


Fig. 7: Figure 1: Schematic trend of $K_P$ versus f

A lower-frequency limit for applying $K_P$ is the smallest frequency that is available in a time series such as f=0.04 to f=0.05 Hz. But an upper-frequency limit is depending on the deployment situations. As it was pointed out, wave pressure attenuates from a water surface toward a seabed (Figure 2). As a wave gets smaller (its frequency increases) it damps in water depth sooner. Because of that, there is a limit that a wave smaller than that (wave with larger frequency than that limit) cannot be sensed by a sensor, as their effect is damped completely before reaching a sensor depth. In that case, $K_P$ should only apply to a range of frequencies that their effects reach a sensor location. So, for any case, depending on deployment situations, a range of frequencies that are large enough to reach a sensor should be calculated, and $K_P$ only applies to that range.

To calculate a maximum frequency that $K_P$ should be applied, deep-water condition can be considered as $h = \frac{L}{2}$ or $kh = \pi$. Considering a sensor setup as described in Figure 2, it can be written:

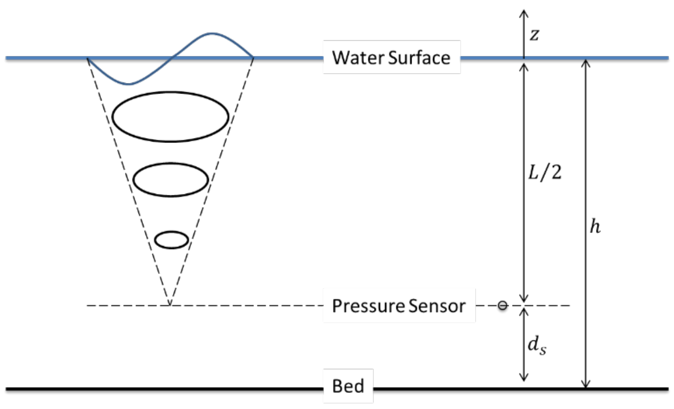


Fig. 8: Figure 2: Schematic of sensor deployment setup

$$h - d_s = \frac{L}{2} \; or \; k(h - d_s) = \pi \; so \; k = \frac{\pi}{(h - d_s)}$$

$$K_{P-min}(h, d_s) = \frac{\cosh(k(h+z))}{\cosh(kh)} = \frac{\cosh(kd_s)}{\cosh(kh)} = \frac{\cosh(\frac{d_s}{(h-d_s)}\pi)}{\cosh(\frac{h}{(h-d_s)}\pi)}$$

Then, a maximum frequency that $K_P$ should apply to data up to that frequency is:

$$\omega^2 = gk \tanh(kh)$$

$$(2\pi f)^2 = gk \tanh(kh)$$

$$f = \frac{\sqrt{gk \tanh(kh)}}{2\pi}$$

$$f_{max(for\ pressure\ correction)} = \frac{\sqrt{g\frac{\pi}{(h-d_s)} \tanh(\frac{h}{(h-d_s)}\pi)}}{2\pi}$$

For the case that a pressure sensor sits on a seabed (i.e. $d_s = 0$ and $kh = \pi$), the $K_{P-min}$ and $f_{max(for\ pressure\ correction)}$ are:

$$K_{P-min}(h,0) = \frac{\cosh(0)}{\cosh(\pi)} = 0.0863$$

$$f_{max(for\ pressure\ correction)} = \frac{\sqrt{g\frac{\pi}{h}\tanh(\pi)}}{2\pi}$$

Figures 3 and 4 show the maximum frequency that $K_P$ should not apply beyond that frequency.

For more details on this topic refer to Karimpour and Chen (2017) and Karimpour (2018).

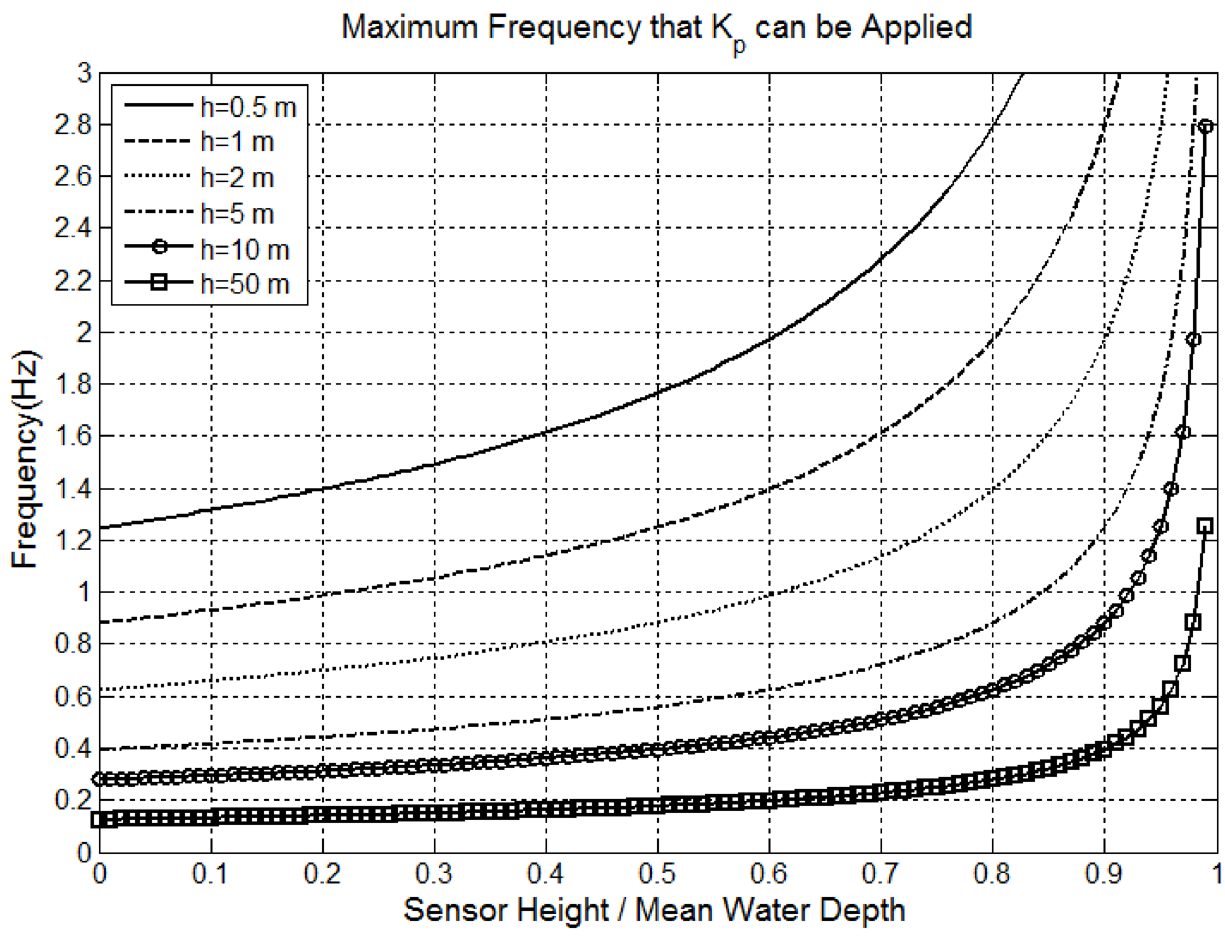


Fig. 9: Figure 3: Maximum frequency that $K_P$ should not applied beyond that frequency.

### 1.10.1 References

- Karimpour, A., & Chen, Q. (2017). Wind Wave Analysis in Depth Limited Water Using OCEANLYZ, a MATLAB toolbox. Computers & Geosciences, 106,181-189.

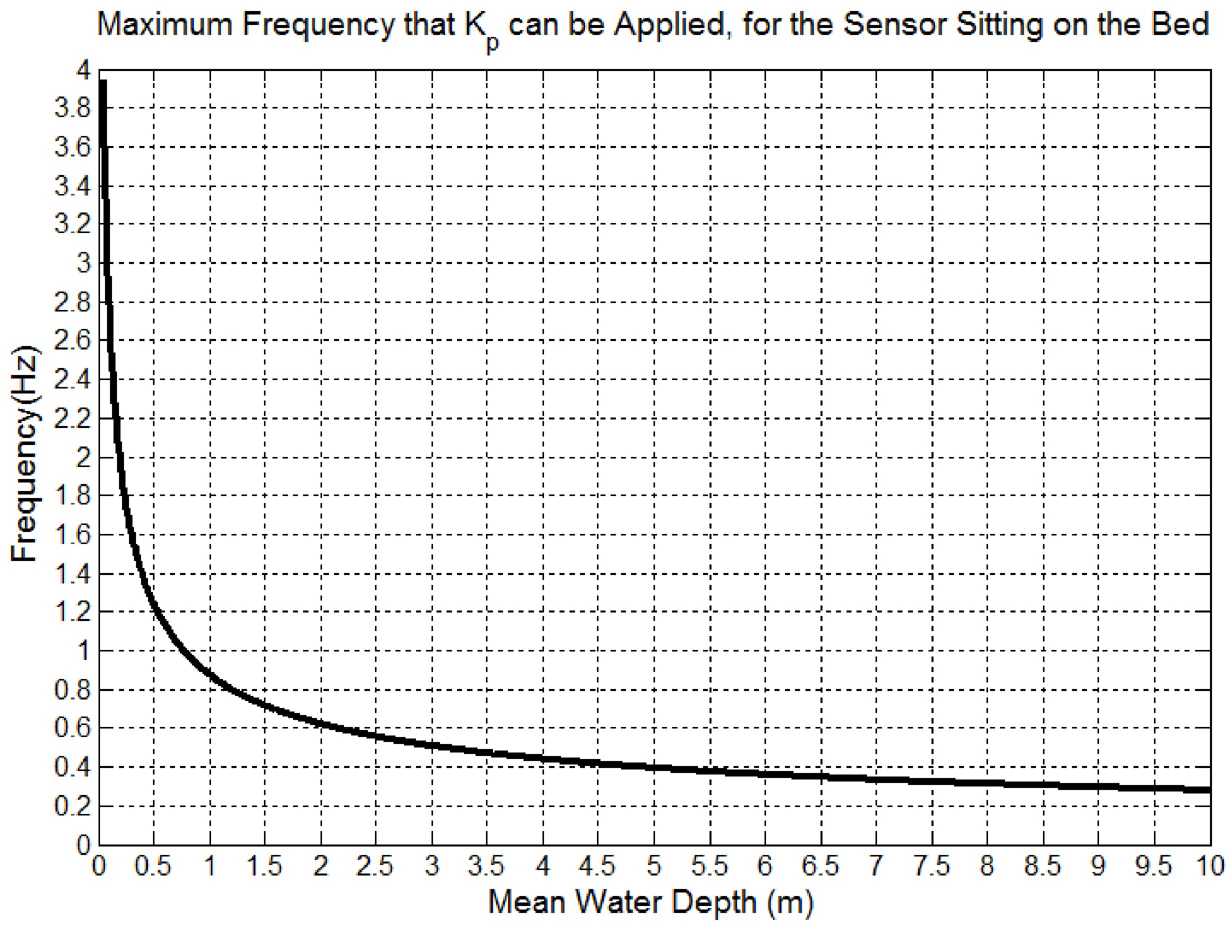


Fig. 10: Figure 4: Maximum frequency that $K_P$ should not apply beyond that frequency, for a sensor sitting on a seabed.

- Karimpour A., (2018), Ocean Wave Data Analysis: Introduction to Time Series Analysis, Signal Processing, and Wave Prediction, KDP.

## 1.11 Replace Spectrum Tail with Diagnostic Tail

If for any reason the values of spectrum tail (a high frequency section of the wave spectrum) are not reliable, then spectrum tail may be replaced with tail of empirical spectrum (diagnostic tail). It is not recommended to use a diagnostic tail for measured data, unless either data are recorded with a low sampling frequency which leads to no data in higher frequency range, or higher frequency data are contaminated with noise. In these cases, a higher section of the spectrum can be replaced with tail of empirical spectrum as (Siadatmousavi et al. 2012):

$$S_{yy}(f) = S_{yy}(f_{tail}) \times (\frac{f}{f_{tail}})^{(-n)} \text{ for } f > f_{tail}$$

where, $S_{yy}(f)$ is a water surface elevation power spectral density, f is a frequency, $f_{tail}$ is a frequency that a tail applied after that, and n is a power coefficient. The f_tail typically set at 2.5f_m, where $f_m = 1/T_{m01}$ is a mean frequency (Ardhuin et al. 2010). A value of n depends on deployment conditions, however, typically it is -5 for deep and -3 for shallow water (e.g. Kaihatu et al. 2007, Siadatmousavi et al. 2012).

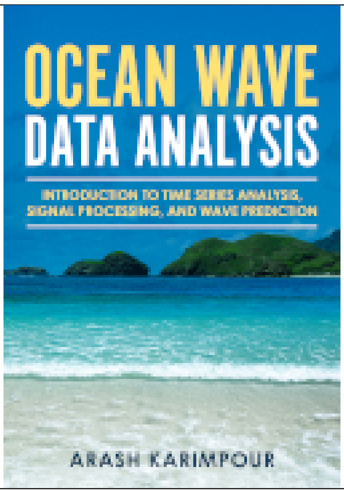
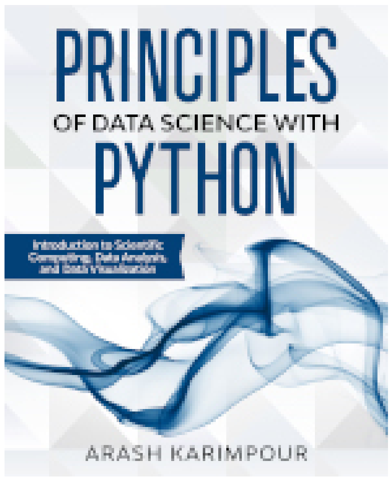
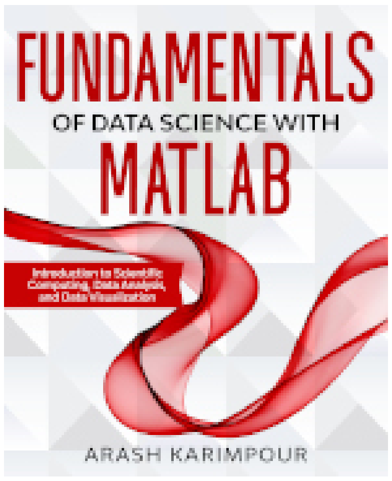
For more details on this topic refer to Karimpour and Chen (2017) and Karimpour (2018).

### 1.11.1 References

- Ardhuin, F., Rogers, E., Babanin, A. V., Filipot, J. F., Magne, R., Roland, A., … & Collard, F. (2010). Semiempirical dissipation source functions for ocean waves. Part I: Definition, calibration, and validation. Journal of Physical Oceanography, 40(9), 1917-1941.
- Kaihatu, J. M., Veeramony, J., Edwards, K. L. & Kirby, J. T. 2007 Asymptotic behaviour of frequency and wave number spectra of nearshore shoaling and breaking waves. J. Geophys. Res. 112, C06016
- Karimpour, A., & Chen, Q. (2017). Wind Wave Analysis in Depth Limited Water Using OCEANLYZ, a MATLAB toolbox. Computers & Geosciences, 106,181-189.
- Karimpour A., (2018), Ocean Wave Data Analysis: Introduction to Time Series Analysis, Signal Processing, and Wave Prediction, KDP.
- Siadatmousavi, S. M., Jose, F., & Stone, G. W. (2011). On the importance of high frequency tail in third generation wave models. Coastal Engineering.

CHAPTER 2

# Recommended Books

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