Breakwater

Release 1.0

Aug 13, 2020

CONTENTS

Date July 2020

Author Sander Winkel

Master thesis Developing a design automation tool for the conceptual design of breakwaters

Version 1.0

CHAPTER

ONE

TABLE OF CONTENTS

1.1 Introduction

`breakwater` was developed during my master thesis, see the report here. The objective of my thesis was to develop a design automation tool that is able to quickly design different types of breakwaters during the design process. In this introductory chapter the purpose, background and features of thee tool are discussed.

1.1.1 Purpose

The main goal of the tool is to support the designer in exploring different types of breakwaters, see Figure 1.1 for a cross section of the breakwater types defined by CIRIA, CUR, CETMEF (2007). The main use case of the tool is thus to make a conceptual design of a breakwater using one of the design classes, see Chapter 7. However, because all functions and classes are also available from `breakwater.core` it is also possible to develop your own design automation script.

Note: Due to the limited time not all breakwater types defined by CIRIA, CUR, CETMEF (2007) could be implemented. Currently only the following structures have been implemented: conventional rubble mound breakwaters with rock and armour units as armour layer, caisson breakwaters and vertically composite breakwaters.

1.1.2 Background Information

Over the past years several tools have been developed to support the designer during the breakwater design process, see Figure 1.2. For example the tools developed by Sijbesma (2019) and Laenen (2000), who both used a probabilistic design approach to design breakwaters. The tool of Sijbesma (2019) only includes the conventional rubble mound breakwater with rock as armour layer, where the tool of Laenen (2000) also includes a caisson breakwater. However, the freedom of the designer is limited in the tool of Laenen (2000) as, for instance, the crest width is fixed to 8.6 m.

Furthermore, most tools are not open source available, and can thus not be used by other designers, or for educational purposes. Moreover, during interviews it appeared that a probabilistic design approach is perceived as a disadvantage by designers as such a design approach often results in a too conservative, and thus too expansive design. The main argument against using probabilistic design approaches is that there is often a lack of data to effectively use a probabilistic design approach (Winkel, 2020).

However, the use of generating, and exploring more concepts is beneficial for the design process, as Davila Delgado and Hofmeyer (2013) showed in their experimental study. Furthermore, the reports of McKinsey&Company (2017) and Deloitte (2019) state that the construction industry needs to adopt new digital technologies to increase the efficiency and productivity, which has been lagging compared to other industries over the past decades.

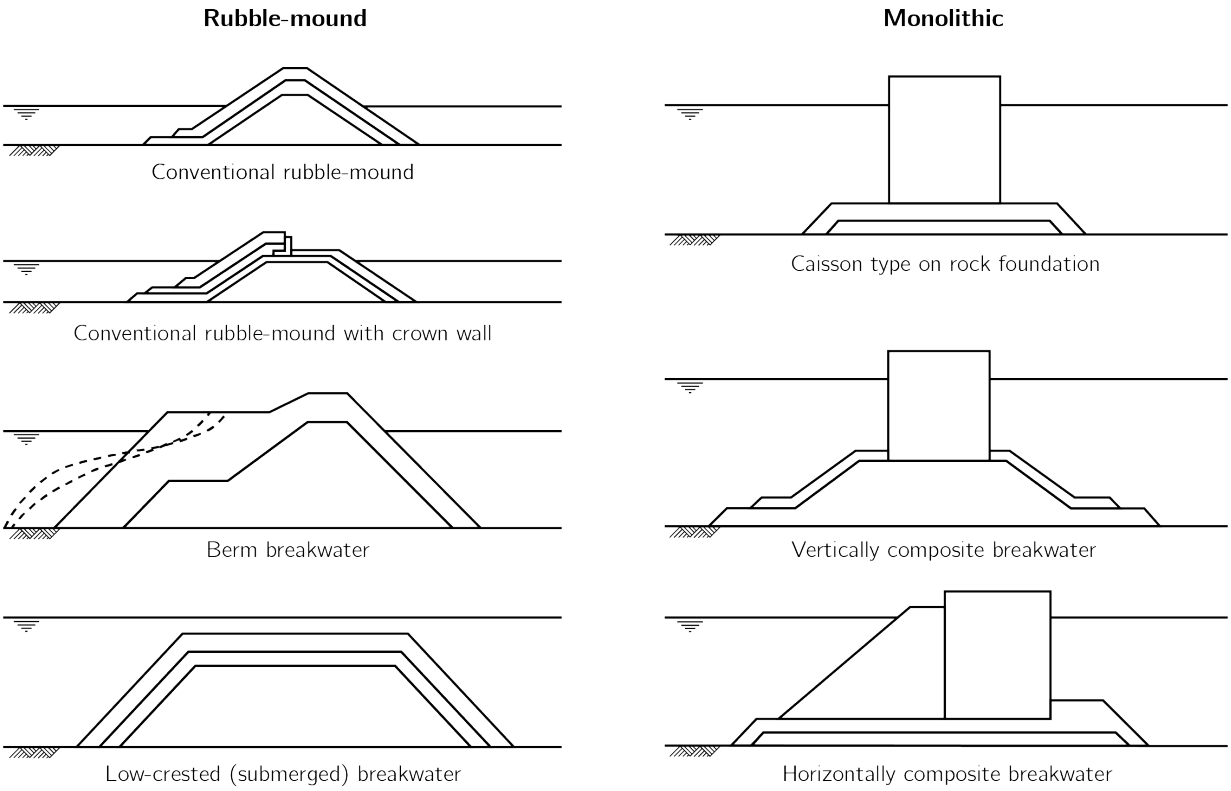


Fig. 1: Figure 1.1: Typical cross sections of various types of breakwaters, with the rubble mound types on the left and the monolithic types on the right. Redrawn from CIRIA, CUR, CETMEF (2007, p.781)

Therefore, `breakwater` aims to automate the synthesis, simulation and evaluation steps of the conceptual design phase, see Figure 1.2. This is done to support the designer in exploring concepts, so that the designer can investigate the influence of several parameters on the design and cost.

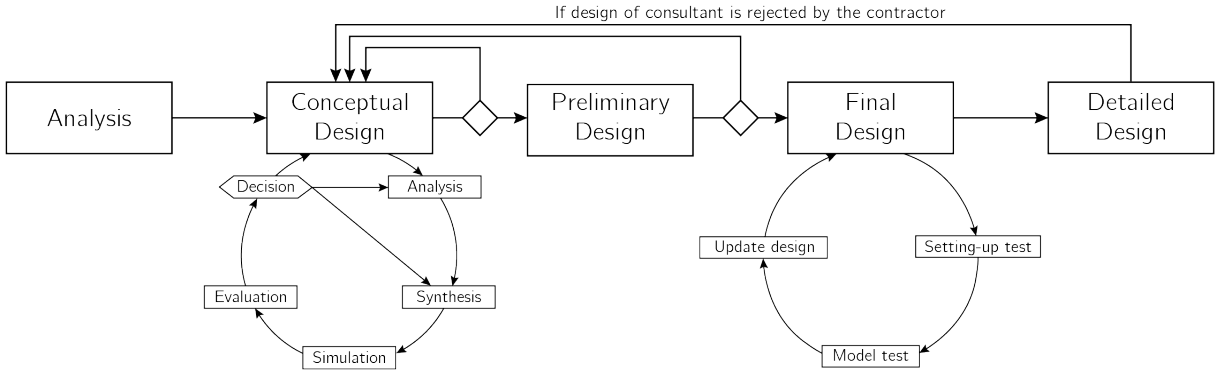


Fig. 2: Figure 1.2: General design approach as described in Winkel (2020)

1.1.3 Features

`breakwater` offers the following features to support the designer:

- Design a rubble mound breakwater with rock or concrete armour units as armour layer, with *bw.RockRubbleMound* or *bw.ConcreteRubbleMound*
- Design a vertical or vertically composite breakwater with *bw.Caisson*
- Design with an interactive design application by using *bw.interactive_design*. With this application several parameters can be changed with sliders, to assess the influence of certain parameters on the design and cost.
- Design multiple breakwaters at ones by using *bw.Configurations*. With this class multiple breakwater types can be designed at ones, these concepts can than be assess by using a multi-criteria analysis or with the DesignExplorer.
- Use the functions and classes from `breakwater.core` to create your own design automation approach. The `breakwater.core` consist of all functions and classes defined in Chapters 8 to 11

1.1.4 Getting Started

This documentation provides all information required to install the package, see Chapter 2, and start designing breakwaters with Python, all subsequent chapters.

1.2 Installation

This chapter explains the dependencies and the different ways to install the package.

1.2.1 Dependencies

For the package to work some prerequisite packages must be installed on you system. It might be that you have already installed these packages, since it are quite common packages.

- Python : Version 3.6 or higher
- NumPy : The fundamental scientific programming package, it provides a multidimensional array type and many useful functions for numerical analysis.
- SciPy : Library of algorithms for mathematics, science and engineering.
- Matplotlib : Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.
- Pandas : A fast, powerful, flexible and easy to use open source data analysis and manipulation tool
- tabulate : Pretty-print tabular data

The following are optional dependencies and are only required for some features:

- XlsxWriter : a module that can be used to write text, numbers, formulas and hyperlinks to multiple worksheets in an Excel 2007+ XLSX file. Required dependency for *bw.generate_excel*
- Basemap : The matplotlib basemap toolkit is a library for plotting 2D data on maps in Python. Required dependency for *breakwater.database.BreakwaterDatabase*

1.2.2 Installation

`breakwater` can be installed from PyPI or with the source code from the GitHub repository.

From PyPI

The latest release is available at the Python package index

```
pip install breakwater
```

From GitHub

Download or clone the source code from the GitHub repository. Then move into the directory with the source code and run:

```
python setup.py install
```

1.2.3 Bugs and Feature Requests

Problems with the installation, bugs in the code or feature request can be reported at the issue tracker of the GitHub repository. Comments and questions are always welcome and can be send to pybreakwater@gmail.com.

1.3 Tutorial

This tutorial guides you through some typical use cases. See table 1 for the discussed tutorials.

Table 1: List of tutorials and description

Tutorial	File	Description
Rubble Mound with Rock	`example1.py`	Using BattjesGroenendijk to compute the wave heights and design with single `LimitState`
Rubble Mound with Xbloc	`example1.py`	Using BattjesGroenendijk to compute the wave heights and design with single `LimitState`
Vertical with Rock	`example2.py`	Using goda_wave_heights to compute wave heights and design with two `LimitState` objects
Parametric design with Python	`example3.py`	Parametric design for Rubble Mound breakwater with Rock (RRM) and Xbloc (CRM)
Parametric design from Excel	`example4.py`	Generate Excel input file and make a parametric design from Excel an input file

1.3.1 Rubble Mound Design

Two designs are elaborated on. First a conventional rubble mound breakwater with rock as armour layer and secondly, one with armour units as armour layer. This example can also be found in `example1.py`

Rock

First the package must be imported.

```
import breakwater as bw
```

Importing `breakwater` like this will automatically import the most commonly used classes and functions. These are, all design classes, all materials, the limit state and the functions to compute wave heights. Which is also the next step. Compute a missing wave height, H2% in this case, and define a limit state.

```
battjes = bw.BattjesGroenendijk(Hm0=4.4, h=15, slope_foreshore=(1,100))
H2_per = battjes.get_Hp(0.02)

ULS = bw.LimitState(
    h=15, Hs=4.5, Hm0=4.4, H2_per=H2_per, Tp=9.4, Tm=8.8, T_m_min_1=9.7,
    Sd=5, Nod=2, q=20, label='ULS')
```

The next step is to define a material for the armour layer, in this case we will first design a breakwater with rock as armour layer. We use the standard rock grading defined in the standard NEN-EN 13383 (2002) with a density of 2650 kg/m^3, which is the default value.

```
NEN = bw.RockGrading(rho=2650)
```

We are now ready to design a breakwater.

```
RRM = bw.RockRubbleMound(
    slope=(2,3), slope_foreshore=(1,100), rho_w=1025, B=5.5, N=2100,
    LimitState=ULS, Grading=NEN, Dn50_core=0.4)
```

The breakwater is designed, let's first inspect if any warnings were encountered during the design process.

```
RRM.print_logger(level='warnings')
```

No warnings were encountered, so we are ready to explore the design. Let's first see at how the breakwater looks, and the details of the generated variants.

```
RRM.plot('all')
RRM.print_variant('all')
```

We can see that two variants have been designed, with a different filter layer but the same underlayer. Finally, let's inspect the validity ranges as the equations of van der Meer for the armour layer and toe are experimental formulae

```
RRM.check_validity()
```

From the table the validity ranges, used value and if the parameter is in range can be read. It is up to the user to assess if the used values are reasonable.

Armour Units

The first steps are the same as when rock was the armour layer, as can be seen from the code.

```
import breakwater as bw

battjes = bw.BattjesGroenendijk(Hm0=4.4, h=15, slope_foreshore=(1,100))
H2_per = battjes.get_Hp(0.02)

ULS = bw.LimitState(
    h=15, Hs=4.5, Hm0=4.4, H2_per=H2_per, Tp=9.4, Tm=8.8, T_m_min_1=9.7,
    Sd=5, Nod=2, q=20, label='ULS')

NEN = bw.RockGrading(rho=2650)
```

It is still required to define a rock grading as the underlayer of the breakwater is made out of rock. We will now define the armour units. It is possible to define a custom armour layer with `bw.ConcreteArmour`, but in the example we will use a predefined one.

```
xbloc = bw.Xbloc()
```

We are now ready to design a breakwater with Xbloc as armour layer.

```
CRM = bw.ConcreteRubbleMound(
    slope=(2,3), slope_foreshore=(1,100), B=5.5, rho_w=1025, LimitState=ULS,
    ArmourUnit=xbloc, Grading=NEN, Dn50_core=0.4)
```

The breakwater is designed, again let's first inspect if any warnings were encountered during the design process.

```
CRM.print_logger(level='warnings')
```

Again, no warnings were encountered during the design, so we are ready to explore the generated design. So let's again plot and print all variants.

```
CRM.plot('all')
CRM.print_variant('all')
```

These values can now be compared to the concept with rock as armour layer, or further designed as the geotechnical stability is not designed by the tool.

1.3.2 Monolithic Breakwater Design

In this section it is explained how a vertical breakwater with rock as armour layer of the foundation can be designed. This example can also be found in `example2.py`

First the package must be imported.

```
import breakwater as bw
```

Importing `breakwater` like this will automatically import the most commonly used classes and functions. These are, all design classes, all materials, the limit state and the functions to compute wave heights. Which is also the next step. We will transform the deep water wave height to a design wave height, Hmax, with the empirical formulae derived by Goda (2000).

```
H13_ULS, Hmax_ULS = bw.goda_wave_heights(
    h=15.1, d=12, Ho=5.3, T=9.4, slope_foreshore=(1,100))

H13_SLS, Hmax_SLS = bw.goda_wave_heights(
    h=12.1, d=9, Ho=3.3, T=7.9, slope_foreshore=(1,100))
```

The next step is to define the limit state functions. In this example we will define two limit states, the ultimate limit state (ULS) and a serviceability limit state (SLS).

```
ULS = bw.LimitState(
    h=15.1, H13=H13_ULS, Hmax=Hmax_ULS, T13=9.4, q=30, label='ULS')
SLS = bw.LimitState(
    h=12.1, H13=H13_SLS, Hmax=Hmax_SLS, T13=7.9, q=15, label='SLS')

ULS.transform_periods(0.5)
SLS.transform_periods(0.5)
```

As you can see we also used the method `transform_periods()` to transform the missing wave periods, especially $T_{m-1.0}$ as it is used to compute the required crest freeboard. Since the assumption that we are in deep water is not valid the method will display a LimitStateWarning, the better practice is to derive the wave periods from a model like SWAN.

The next step is to define a material for the armour layer off the foundation, in this example we will use the default rock grading of the NEN-EN 13383 (2002).

```
NEN = bw.RockGrading()
```

We are now ready to design a vertical breakwater!

```
RC = bw.Caisson(
    Pc=0.2, rho_c=2400, rho_fill=1600, rho_w=1000, Bm=8, hb=2, layers=2,
    BermMaterial=NEN, LimitState=[ULS, SLS], slope_foreshore=(1,100), mu=0.5,
    beta=15)
```

Let us first inspect the logger, we will print the info level as well in this example because this allows us to see which overtopping formula is used. Furthermore, we can also see if overturning or sliding was normative for the computation of the required width.

```
RC.print_logger(level='info')
```

We now know some general info, so let's explore the design. Let's see at how the breakwater looks, and the details of the generated variants.

```
RC.plot('all')
RC.print_variant('all')
```

1.3.3 Design Automation

In this section a design automation tool is used. First the design is made fully in Python and secondly the design is made from and Excel input file. The values used in this section are based on the ones used in *Rubble Mound Design*.

With Python

This example can also be found in `example3.py`

Similar to the example for a rubble mound breakwater we again begin with by defining the wave heights and materials to used.

```
battjes = bw.BattjesGroenendijk(Hm0=4.4, h=15, slope_foreshore=(1,100))
H2_per = battjes.get_Hp(0.02)

ULS = bw.LimitState(
    h=15, Hs=4.5, Hm0=4.4, H2_per=H2_per, Tp=9.4, Tm=8.8, T_m_min_1=9.7,
    Sd=5, Nod=2, q=20, label='ULS')

NEN = bw.RockGrading(rho=2650)
xbloc = bw.Xbloc()
```

Now that the hydraulic conditions and materials are defined, we can use the `Configurations` class. This means that multiple configurations of the specified breakwater type(s) are designed, in the example the parameters allowed to vary are: the slope, the width of the breakwater (B) and den nominal diameter of the core (Dn50_core).

```
configs = bw.Configurations(
    structure=['RRM', 'CRM'], LimitState=ULS, rho_w=1025,
    slope_foreshore=(1,100), Grading=NEN, slope=((1,3), (3,4), 4), B=(5, 8, 4),
    Dn50_core=(0.2, 0.4, 3), N=2100, ArmourUnit=xbloc)
```

The first step is to see if, and which, warnings have been encountered during the design process.

```
configs.show_warnings()
```

In the table we can see that a lot of warnings have been encountered, the most are related to the hydraulic input. Instead of H13 we specified Hs in the LimitState, but the design looks for H13. This value is not found and thus Hs is used instead, which is fine as H13 is a definition of Hs. However, the tool does inform the user about this behaviour.

As 96 concepts are a lot of concepts to filter by yourself, the concepts can be exported to Design Explorer 2. This online tool allows the user to visually filter the concepts, see *to_design_explorer*. We export the generated concepts with the following parameters: the slope, the class of the armour layer, the width of the breakwater and the crest freeboard.

```
configs.to_design_explorer(params=['slope', 'class armour', 'B', 'Rc'])
```

The last step is to save the generated concepts to a .breakwaters file. This allows the user to access all the concepts at a latter moment without designing them again.

```
configs.to_breakwaters('example3')
```

The concepts can be reloaded with *bw.read_breakwaters*.

With Excel input file

As mentioned in the introduction of this section a design can also be made from an Excel input file. Note that this script is available in `example4.py`.

> **Warning:** While it is possible to make your own Excel input file, you are advised to use *bw.generate_excel* to generate the required excel file.

The first step is to create the required Excel input file.

```
import breakwater as bw

bw.generate_excel('config input.xlsx')
```

Now you can give the required input in an Excel file, so give your input in the Excel file. We can now import the Excel file and make a design for multiple configurations.

```
configs = bw.read_configurations(
    'config input.xlsx', structure=['RRM', 'CRM'], kd=16, name='Xbloc')
```

This function returns a *bw.Configurations* object. Therefore, it is now possible to show the warnings, export the concepts to the design explorer, just as we have done in the previous example.

Note: In this example we specified all classes of Xbloc in the Excel file. However, as these armour units are already defined in `breakwater` the better practice is to specify them in *bw.read_configurations* with the keyword argument ArmourUnits. This allows `breakwater` to compute the correction factor, which results in a better design.

> **Warning:** It is currently not possible to use the Excel input file with multiple limit states. Want to design with multiple limit states? See the previous subsection, *With Python*

1.4 Limit State

The design conditions, such as the design wave height and the damage number, are described in a limit state. A limit state is defined by standard NEN-EN 1990:2002 (2002), as the state beyond which the structure no longer fulfils the relevant design criteria. Two of the most common limit states are:

- The Ultimate Limit State (ULS), which is the state associated with collapse of the breakwater or with other similar forms of failure
- The Serviceability Limit State (SLS), which is the state that corresponds to the conditions beyond which specified service requirements for the breakwater or part of the breakwater are no longer met.

1.4.1 Define Limit State

class breakwater.conditions.**LimitState**(*h*, *label*, ***kwargs*)

Generate LimitState for design

Define a design limit state, for instance the Ultimate Limit State (ULS) or Serviceability Limit State (SLS). The defined limit state[s] can then be given as arguments to design one of the supported breakwater types, see Table 1 for the required parameters for these design classes. A full list of possible arguments can be seen under Keyword Arguments.

Table 1: Required parameters for each design class

Parameter	RRM	CRM	RC	CC
h	x	x	x	x
q	x	x	x	x
H13	o [1]	o [1]	o [1]	o [1]
Hm0	o [1]	o [1]	o [1]	o [1]
H2_per	o [2]			
Hmax			x	x
Tm	o [3]			
T13			o [3]	o [3]
T_m_min_1	o [3]	o [3]	o [3]	o [3]
Nod	o [4]	x		
Sd	o [4]			

[1] uses *get_Hs()* to get Hs, so at least one of the wave heights must be given

[2] if H2_per not specified in the limit state it is computed with *get_H2()*

[3] wave periods can be transformed in deep water with *transform_periods()*

[4] either Nod or Sd is required, the other can be computed with *Nod()* or *Sd()*

Note: The design classes *bw.RockRubbleMound*, *bw.ConcreteRubbleMound* and *bw.Caisson* perform [1] and [2] automatically, but not [3] and [4]

Parameters

- **h** (*float*) – the water depth [m]
- **label** (*str*) – label to identify the LimitState, for instance ULS or SLS

Keyword Arguments

- **q** (*float*) – mean overtopping discharge per meter structure width [l/s per m]
- **Hs** (*float*) – the significant wave height [m]
- **H13** (*float*) – mean of the highest 1/3 of the wave heights [m]
- **Hm0** (*float*) – the spectral wave height [m]
- **Ho** (*float*) – deep water wave height [m]
- **H2_per** (*float*) – the wave height exceeded by 2% of the waves [m]
- **Hmax** (*float*) – design wave height for the Goda formula, equal to the mean of the highest 1/250 of the wave heights [m].
- **H1250** (*float*) – mean of the highest 1/250 of the wave heights [m], automatically interpreted as Hmax.
- **Tm** (*float*) – the mean wave period [s]
- **T13** (*float*) – the significant wave period [s]
- **Ts** (*float*) – the significant wave period [s]
- **T_m_min_1** (*float*) – $T_{m-1.0}$, the energy wave period [s]
- **Tp** (*float*) – the peak period [s]
- **Nod** (*float*) – Damage number, used in the formula for the toe stability. Can also be computed with *Nod()* [-]
- **Sd** (*float*) – Damage level parameter, used in Van der Meer formula. Can also be computed with *Sd()* [-]

conditions
Dictionary with all parameters

Type dict

deep_water
True if deep water assumption is valid, False if not valid. The bool is computed with the Tm, Tp, T_m_min_1 or T13 (in this order)

Type bool

h
the water depth [m]

Type float

label
identifier of the LimitState

Type str

L(*period*, *deep_water=False*)
Compute the wave length with the dispersion relation

$$L = L_o \tanh\left(\frac{2\pi h}{L}\right)$$

in which the deep water wave length is:

$$L_o = \frac{gT^2}{2\pi}$$

Parameters

- **period** (*str*) – keyword argument of a wave period, must be in LimitState
- **deep_water** (*bool, optional, default: False*) – if False the dispersion relation will be used, if True the deep water wave length will be used. Note that this parameter is not equal to the attribute deep_water

Raises **KeyError** – If the specified wave period is not in the LimitState

Nod(*G*, *nv*)

Compute the damage number Nod

This method approximates Nod by using equation 5.150 from the Rock Manual (CIRIA, CUR, CETMEF, 2007), and adds Nod to the LimitState. The equation to compute Nod is given as:

$$N_{od} = G(1 - n_v)S_d$$

Parameters

- **G** (*float*) – gradation factor [-]
- **nv** (*float*) – porosity of the armour layer [-]

Returns **Nod** (*float*) – damage number [-]

Raises **KeyError** – If Sd is not in the LimitState, use *Sd()* instead

Sd(*G*, *nv*)

Compute the damage level parameter Sd

This method approximates Sd by using equation 5.150 from the Rock Manual (CIRIA, CUR, CETMEF, 2007), and adds Sd to the LimitState. The equation to compute Sd is given as:

$$S_d = \frac{N_{od}}{G(1 - n_v)}$$

Parameters

- **G** (*float*) – gradation factor [-]
- **nv** (*float*) – porosity of the armour layer [-]

Returns **Sd** (*float*) – damage level parameter [-]

Raises **KeyError** – If Nod is not in the LimitState, use *Nod()* instead

check_deep_water()

Check if the deep water assumption is valid

Deep water assumption is valid if h/L > 0.5, where L is computed with the dispersion relation using the mean period. Method updates the attribute *deep_water* based on the check.

Raises **KeyError** – If Tm is not in the LimitState

get_H2(*slope_foreshore*)

Get the wave height exceeded by 2% of the waves, H2%

Attempts to return H2% from the defined limit state, if not defined in the limit state it is computed with `BattjesGroenendijk`.

Parameters **slope_foreshore** (*float*) – the slope of the foreshore [rad]

Returns **H2_per** (*float*) – the wave height exceeded by 2% of the waves [m]

Raises **KeyError** – If Hm0 is not in the LimitState

get_Hs(*definition*)
Get the significant wave height

Multiple definitions are used for the significant wave height, and formulas use different definitions. This method returns the desired definition of the significant wave height. For overtopping this is Hm0 and H1/3 is used in the Van der Meer and Hudson formula.

Note: If Hm0, Hs or H13 are in the LimitState this method will **always** return a significant wave height. The table depicts the order in which Hs is returned if a value is missing. For example: if the chosen definition is Hm0, then first Hm0 is returned, if Hm0 is not included in the Limitstate Hs will be retuned, if Hs is also missing H13 will be returned.

definition	Hm0	Hs	H13
1	Hm0	Hs	H13
2	Hs	H13	Hs
3	H13	Hm0	Hm0

Parameters **definition** (*{'Hm0', 'Hs', 'H13'}*) – definition of the significant wave height to use

Returns **Hs** (*float*) – significant wave height based on the definition

Raises

- **InputError** – If there is no significant wave height (Hm0, Hs, H1/3) in the LimitState
- **KeyError** – If the definition is not 'Hm0', 'H13' or 'Hs'

s(*number=None*, *H=None*, *T=None*)
Compute the fictitious wave steepness

$$s_0 = \frac{H}{L_o}$$

the fictitious wave steepness combines the value of the wave height at the location of the breakwater with the deep water wave length. It is possible to select a number or specify which wave height and period must be used for the computation.

Note: When computing the fictitious wave steepness the significant wave height is needed, to get the significant wave height form the LimitState the method *get_Hs()* is used.

Parameters

- **number** (*{'mean', 'spectral'}, optional, default: None*) – definition to use, will automatically select the correct definitions for H and T
- **H** (*str, optional, default: None*) – wave height in the LimitState
- **T** (*str, optional, default: None*) – wave period in the LimitState
- **returns** –
- **s_0** (*float*) – the fictitious wave steepness

Raises

- **InputError** – If there is no significant wave height (Hm0, Hs, H1/3) in the LimitState.

- **KeyError** – if the specified wave period is not in the LimitState

surf_similarity(*alpha*, *number=None*, *H=None*, *T=None*)
Compute the surf similarity parameter

$$\xi = \frac{tan\alpha}{\sqrt{s_o}}$$

Computes the surf similarity parameter, also known as the Iribarren number. It is possible to select a number or specify which wave height and period must be used for the computation.

Parameters

- **alpha** (*float*) – slope of the structure [rad]
- **number** (*{'mean', 'spectral'}, optional, default: None*) – definition to use, will automatically select the correct definitions for H and T
- **H** (*str, optional, default: None*) – wave height in the LimitState
- **T** (*str, optional, default: None*) – wave period in the LimitState

Returns **xi** (*float*) – the surf similarity parameter [-]

Raises

- **InputError** – If there is no significant wave height (Hm0, Hs, H1/3) in the LimitState
- **KeyError** – If the specified wave period is not in the LimitState

transform_periods(*scalar*)
Transform the wave periods with the Rayleigh characteristics

Transforms the missing wave periods in the LimitState with the characteristics for Rayleigh distributed waves in deep water, by using the wave periods which are in the LimitState.

Table 2: characteristic wave periods (Van den Bos and Verhagen, 2018)

Name	T/Tp
T_p	1
T_m	0.75 to 0.85
T_s	0.90 to 0.95
$T_{m-1.0}$	1.1

Parameters **scalar** (*float*) – Should be between 0 and 1, 0 being the lower bound, 1 the upper bound of the characteristics. For instance, scalar of 1 results in Tm/Tp = 0.85.

Raises **InputError** – If the scalar is not between 0 and 1

1.4.2 Compute wave heights

Design criteria for coastal structures require, among other parameters, a certain type of characteristic wave height, for instance the significant wave height. In deep water the behaviour of the waves can be determined with the characteristic of the Rayleigh distribution. However, in shallow water the waves are no longer Rayleigh distributed, but a designer still wants to determine the significant wave height.

Two methods have been implemented to determine the nearshore wave characteristics. The first is Battjes and Groenendijk, which determines the wave characteristics from the output of a wave energy model (Battjes and Groenendijk, 2000). Secondly, the formulation by Goda (2000) which is based on experimental data.

Battjes and Groenendijk

class breakwater.core.battjes.**BattjesGroenendijk**(*Hm0*, *h*, *slope_foreshore*)

Formulas of Battjes and Groenendijk (2000)

The wave height distributions on shallow foreshores deviates from those in deep water due to the effects of the restricted depth-to-height ratio and of wave breaking. Battjes and Groenendijk (2000), therefore derived a generalised empirical parameterisations based on laboratory data to determine these effects. This allows for the computation of all statistical wave heights based on the spectral wave height, water depth and slope of the foreshore.

Parameters

- **Hm0** (*float*) – the spectral wave height [m]
- **h** (*float*) – the water depth [m]
- **slope_foreshore** (*float or tuple*) – the slope of the foreshore [rad], or as tuple formatted as (V,H)

Hrms

the root-mean-square wave height [m]

Type float

Htr_tilde

the transitional wave height, made dimensionless with Hrms [m]

Type float

k1, k2

shape parameters of the Composite Weibull Distribution (CWD) [-]

Type floats

static gammainc_lower(*a*, *x*)

Lower incomplete gamma function

Defined as

$$\Gamma(a,x) = \int_0^x t^{a-1}e^{-t}dt$$

Implemented using scipy.special as gamma(a)*gammainc(a,x)

Parameters

- **a** (*float*) – positive parameter
- **x** (*float*) – nonnegative argument

Returns *float* – value of the lower incomplete gamma function

static gammainc_upper(*a*, *x*)

Upper incomplete gamma function

Defined as

$$\Gamma(a,x) = \int_x^\infty t^{a-1}e^{-t}dt$$

Implemented using scipy.special as gamma(a)*gammaincc(a,x)

Parameters

- **a** (*float*) – positive parameter

- **x** (*float*) – nonnegative argument

Returns *float* – value of the upper incomplete gamma function

get_Hn(*N*)
Computes $H_{1/N}$

This method computes the mean of the highest 1/N-part of the wave heights. It uses scipy.optimize.fsolve to determine H1~ and H2~, which are then used to compute $H_{1/N}$.

Parameters **N** (*float*) – highest 1/N-part of the wave heights [-]

Returns **H_1/N** (*float*) – mean of the highest 1/N part of the waves

Raises **ValueError** – if N is smaller than 1

get_Hp(*P*)
Computes H_P

This method computes the wave height exceeded by P% of the waves. It uses scipy.optimize.fsolve to determine H1~ and H2~, which are then used to compute H_P

Parameters **P** (*float*) – exceedance probability as a fractal [-]

Returns **H_P** (*float*) – wave height exceeded by P% of the waves

Raises **ValueError** – If P is entered as a percentage, must be entered as P%/100

Goda wave heights

breakwater.core.goda.**goda_wave_heights**(*h*, *d*, *Ho*, *T*, *slope_foreshore*, *Ks=None*, *factor=1.8*)
Compute the design wave height

Computes the design wave height $H_{1/3}$ and H_{max} with the empirical formulas of Goda (2000)

Parameters

- **h** (*float*) – water depth [m]
- **d** (*float*) – water depth in front of the caisson, on top of the foundation [m]
- **Ho** (*float*) – deep water wave height [m]
- **T** (*float*) – wave period, Goda (2000) advises to use $T_{1/3}$ [s]
- **slope_foreshore** (*tuple*) – slope of the foreshore (V, H). For example a slope of 1:100 is defined as (1,100)
- **Ks** (*float, optional, default: None*) – non-linear shoaling coefficient. If the shoaling coefficient is not specified it is automatically computed with `shoaling_coefficient()` [-]
- **factor** (*float, optional, default: 1.8*) – Hmax is a probabilistic quantity. But to avoid possible confusion in design, a definite value of Hmax = 1.8 * Ks * H0 is recommended. The value of 1.8 is, however, a recommendation and the user is free to choose another value, such as 1.6 or 2.0 (Goda, 2000)

breakwater.utils.wave.**shoaling_coefficient**(*h*, *T*, *H0*, *linear=False*)
Closed form equation for the non-linear shoaling coefficient

Computes the non-linear shoaling coefficient using a closed form equation. The equation uses the coefficient derived in Kweon and Goda (1996).

Parameters

- **h** (*float*) – water depth [m]
- **T** (*float*) – wave period, Goda (2000) advises to use $T_{1/3}$ [s]
- **H0** (*float*) – equivalent deep water significant wave height [m]
- **linear** (*bool, optional, default: False*) – if true the linear shoaling coefficient is returned, if false the non-linear shoaling coefficient is returned

Returns **Ks** (*float*) – the non-linear shoaling coefficient

1.5 Material

The most important construction materials for breakwaters are quarry stone and concrete, as this is the material used in the armour layer. Both types can be used for designing breakwaters, the rock grading is defined in the first section, whereafter the concrete armour units are defined.

1.5.1 Rock

class breakwater.material.**RockGrading**(*grading=None, rho=2650, y_NLL=0.06, y_NUL=0.9*)

Standard rock grading of the standard NEN-EN 13383-1 (2002)

Create a Rock Grading to determine the rock class in `RockRubbleMound`, `ConcreteRubbleMound`, `RubbleMound` or `Caisson`. By default the standard rock grading of the NEN-EN 13383-1 (2002) is used. However, it is possible to change the grading to a user defined custom grading by giving one in the arguments.

Parameters

- **grading** (*dict, optional, default: None*) – If specified the custom grading will replace the default rock grading of the NEN-EN 13383-1. The custom grading must be formatted as {str: {'M50': [float, float], 'NLL': float, 'NUL': float}}, with str the rock class and a lower and upper bound for M50. Furthermore, the NLL and NUL mass must be specified.
- **rho** (*float, optional, default: 2650*) – density of the armourstone [kg/m^3]
- **y_NLL** (*float, optional, default: 0.06*) – the fraction passing at the nominal lower limit mass (NLL) of a grading
- **y_NUL** (*float, optional, default: 0.9*) – the fraction passing at the nominal upper limit mass (NUL) of a grading

grading
dictionary of the chosen rock grading

Type dict

rho
density of the armourstone

Type float

y_NLL
the fraction passing at the nominal lower limit mass (NLL)

Type float

y_NUL
the fraction passing at the nominal upper limit mass (NUL)

Type float

add_cost(*cost*)
Add cost per m^3 of each Rock Class

Add the cost per m^3 of each Rock Class to the *grading*

Parameters **cost** (*dict*) – cost of each Rock Class, keys must be identical to the name of the rock class

Raises

- **KeyError** – if a rock class of the given cost is not in the grading
- **InputError** – if a rock class of the *grading* has no price

get_class(*Dn50*)
Get the rock class for a given Dn50

Parameters **Dn50** (*float*) – nominal diameter of the armourstone [m]

Returns *str* – Rock class

Raises **RockGradingError** – If the computed Dn50 of the armour layer is out of range for the specified rock grading

get_class_dn50(*class_*)
Get the average Dn50 of a rock class

Parameters **class** (*str*) – Rock class

Returns **Dn50** (*float*) – nominal diameter of the armourstone [m]

Raises **KeyError** – If the specified rock class is not in the rock grading

plot_rosin_rammler(*class_*)
plot the Rosin-Rammler curve for an idealised gradings

Parameters **class** (*str*) – Rock class

Raises **KeyError** – If the specified rock class is not in the rock grading

rosin_rammler(*class_*, *y*)
Compute the Rosin-Rammler curve for an idealised gradings

The Rosin-Rammler curve can be used to interpolate between the limits of standard gradings (CIRIA, CUR, CETMEF, 2007). Given that two fixed point on the curve are known, for instance the NLL and NUL limits, M50 and the uniformity coefficient, n_RRM, can be computed. These values are subsequently used to compute My.

$$M_y = M_{50} \left(\frac{\ln(1-y)}{\ln(0.5)} \right)^{1/n_{RRM}}$$

Parameters

- **class** (*str*) – Rock class
- **y** (*float*) – the fraction passing value

Returns **My** (*float*) – the mass corresponding to that value using a percentage subscript to express that fraction [kg]

Raises **KeyError** – If the specified rock class is not in the rock grading

1.5.2 Concrete Armour Units

The availability of rock is limited. Furthermore, natural blocks weighing more than 10 tons are very rare. However, sometimes a block larger than 10 tons is needed or a breakwater is constructed in an area with limited availability of rock. In these cases artificial blocks, made of concrete, are used.

Several types of concrete armour units have been developed over the years. Currently only Xbloc and XblocPlus from Delta Marine Consultants have been incorporated in the package. However, it is possible to define another type of concrete armour unit by using `ConcreteArmour`.

General

class breakwater.material.**ConcreteArmour**(*kd*, *units*, *name=None*, *rho=2400*)

Define concrete armour units

Define a type of concrete armour unit, *Xbloc* and *XblocPlus* have been predefined.

Parameters

- **kd** (*int*) – Stability coefficient [-]
- **units** (*dict*) – Dictionary of the defined concrete armour units. Format must be {V: dict}, where V is the volume of the armour unit (float) and dict = {'D': float, 'h': float, 'Vc': float} in which D is the diameter, h is the thickness of the armour layers and Vc is the volume of concrete.
- **name** (*str, optional, default: None*) – name of the ArmourUnit, by default the name of the armour unit is derived from the name of the class.
- **rho** (*float, optional, default: 2400*) – density of the concrete used to make the armour units [kg/m^3]

kd

Stability coefficient [-]

Type int

units

Dictionary of the defined concrete armour units. Format must be {V: dict}, where V is the volume of the armour unit (float) and dict = {'D': float, 'h': float, 'Vc': float} in which D is the diameter, h is the thickness of the armour layers and Vc is the volume of concrete.

Type dict

rho

density of the concrete used to make the armour units [kg/m^3]

Type float, optional, default: 2400

get_class(*d*)

Get the volume of a standard concrete armour unit

Parameters **d** (*float*) – minimal required diameter [m]

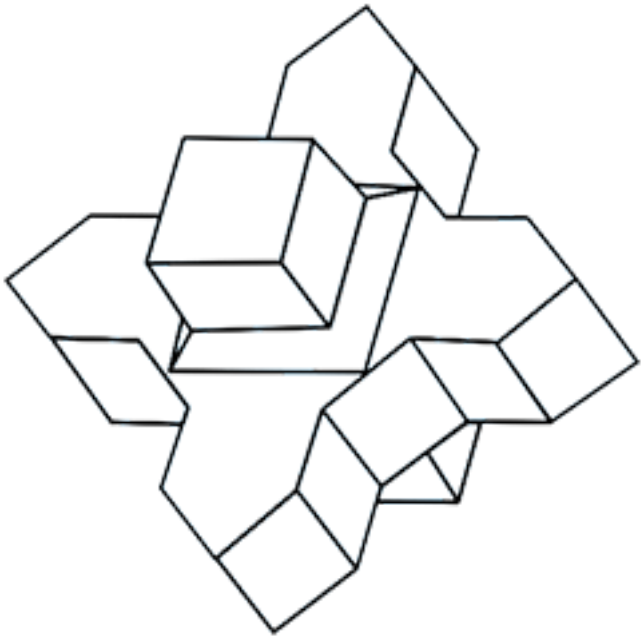
Returns *float* – Volume of a standard armour unit

Raises **ArmourUnitsError** – If the computed Dn of the armour layer is out of range for the specified armour units

Predefined types

class breakwater.material.**Xbloc**(*rho=2400*)

Xbloc concrete armour units



Xbloc is an armour unit developed by Delta Marine Consultants (DMC). All units have been predefined in *units*, with the values from table 1 from Delta Marine Consultants (2018).

Parameters **rho** (*float, optional, default: 2400*) – density of the concrete used to make the armour units

kd

Stability coefficient [-]

Type int

units

Dictionary of the defined concrete armour units. Format is {V: dict}, where V is the volume of the Xbloc (float) and dict = {'D': float, 'h': float, 'Vc': float} in which D is the diameter, h is the thickness of the armour layers and Vc is the volume of concrete.

Type dict

rho

density of the concrete used to make the armour units [kg/m^3]

Type float, optional, default: 2400

correction_factor(*Hs*, *h*, *Rc*, *occurrence_hs*, *slope*, *slope_foreshore*, *permeability*, *logger=None*, ***kwargs*)

Determine correction factor for Xbloc

Correction factors for phenomena which require an increase in the Xbloc unit sizes. For the conceptual design of structures the correction factor must be multiplied with the volume (Delta Marine Consultants, 2018).

Parameters

- **Hs** (*float*) – significant wave height [m]
- **h** (*float*) – water depth [m]
- **Rc** (*float*) – Crest freeboard [m]
- **occurrence_hs** (*bool*) – True if frequent occurrence of the near-design wave height during the lifetime of the structure. False if no frequent occurrence
- **slope** (*float*) – slope of the armour layer [rad]
- **slope_foreshore** (*float*) – slope of the foreshore [rad]
- **permeability** (*{'permeable', 'low', 'impermeable'}*) – Permeability of the core
- **logger** (*dict, optional, default: None*) – dict to log messages, must have keys 'INFO' and 'WARNINGS'

Returns *float* – The correction factor to be applied on the volume. Returns 1 if no correction factor is needed.

get_class(*d*)

Get the volume of a standard concrete armour unit

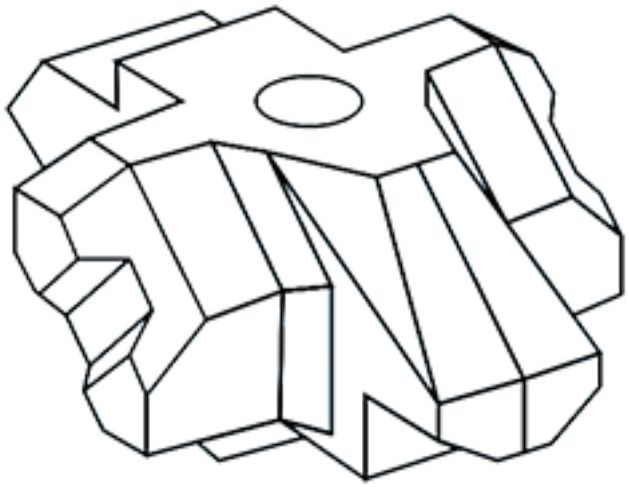
Parameters **d** (*float*) – minimal required diameter [m]

Returns *float* – Volume of a standard armour unit

Raises **ArmourUnitsError** – If the computed Dn of the armour layer is out of range for the specified armour units

class breakwater.material.**XblocPlus**(*rho=2400*)

XblocPlus concrete armour units



XblocPlus is an armour unit developed by Delta Marine Consultants (DMC). All units have been predefined in *units*, with the values from table 2 from Delta Marine Consultants (2018).

Parameters **rho** (*float, optional, default: 2400*) – density of the concrete used to make the armour units

kd

Stability coefficient [-]

Type int

units
Dictionary of the defined concrete armour units. Format is {V: dict}, where V is the volume of the Xbloc-Plus unit (float) and dict = {'L1': float, 'L2': float, 'L3': float, 'h': float 'Vc': float} in which L1, L2 and L3 are the dimensions, h is the thickness of the armour layers and Vc is the volume of concrete.

Type dict

rho
density of the concrete used to make the armour units [kg/m^3]

Type float, optional, default: 2400

correction_factor(*Hs*, *Rc*, *slope_foreshore*, *permeability*, *logger=None*, ***kwargs*)
Determine correction factor for XblocPlus

Correction factors for phenomena which require an increase in the XblocPlus unit sizes. For the conceptual design of structures the correction factor must be multiplied with the volume (Delta Marine Consultants, 2018).

Parameters

- **Hs** (*float*) – significant wave height [m]
- **Rc** (*float*) – Crest freeboard [m]
- **slope_foreshore** (*float*) – slope of the foreshore [rad]
- **permeability** (*str*) – Permeability of the core {'permeable', 'low', 'impermeable'}
- **logger** (*dict, optional, default: None*) – dict to log messages, must have keys 'INFO' and 'WARNINGS'

Returns *float* – The correction factor to be applied on the volume. Returns 1 if no correction factor is needed.

get_class(*d*)
Get the volume of a standard concrete armour unit

Parameters **d** (*float*) – minimal required diameter [m]

Returns *float* – Volume of a standard armour unit

Raises **ArmourUnitsError** – If the computed Dn of the armour layer is out of range for the specified armour units

User defined Armour Unit

As mentioned at the start of this section it is possible to define another type of Armour Unit by using the general class *bw.ConcreteArmour*. In this subsection an example is given how to define a custom armour unit, which can be used in *bw.ConcreteRubbleMound* to design a breakwater. The following code must be used to define the custom armour unit:

```
# define class of the armour unit which inherits from bw.ConcreteArmour
class CustomArmourUnit(bw.ConcreteArmour):
    """ Custom Armour Unit class """

    # define the init method, note that it is not required to give a
    # keyword argument.
    def __init__(self, rho=2400):
        """ See help(CustomArmourUnit) for more info """
        # define the armour units in the init method
```

(continues on next page)

(continued from previous page)

```
        units = {
            1: {'D': 1, 'h': 1, 'Vc': 1},
            2: {'D': 2, 'h': 2, 'Vc': 2},
            3: {'D': 3, 'h': 3, 'Vc': 3},
            4: {'D': 4, 'h': 4, 'Vc': 4},
        }

        # call the init method of bw.ConcreteArmour with super and pass
        # the following arguments: kd factor for Hudson formula, name of
        # the armour unit for overtopping and rho for the density
        super().__init__(kd=10, units=units, name='CustomArmourUnit', rho=rho)

    # this is an optional step. It is possible to define a method
    # to compute a correction factor to be applied on the computed
    # required nominal diameter. Method must allow for kwargs input
    # as several parameters are passed to the method, it is not
    # necessary to use all of the passed parameters
    def correction_factor(self, h, Hs, Rc, **kwargs):
        # it is then possible to build logic in order to determine the
        # correction factor, for example:
        # define list to store more correction factors
        correction = []

        # check for water depth wave height relation
        if h > 2.5*Hs:
            correction.append(1.3)
        else:
            pass

        # check for low crested structure
        if Rc/Hs < 1:
            correction.append(1.5)
        else:
            pass

        # check if any correction factors have been added
        if any(correction):
            # return maximum value
            return max(correction)
        else:
            # return 1, in other words, no correction factor
            # make sure a correction_factor is always returned
            return 1
```

Warning: When using one of the design classes, the supported armour units are limited to the armour units for which an influence factor for the permeability and roughness of the slope has been determined.

Note: It is not required to add a method to compute a correction factor, the design classes do raise an exception if there is no method to compute a correction factor. However, in case you want to define a method to compute a correction factor the following kwargs can be used to determine the correction factor:

Argument	Type	Description
h	float	water depth
Hs	float	significant wave height
Rc	float	crest freeboard
occurrence_hs	bool	frequent occurrence of the near-design wave height during the during the lifetime of the structure
slope	float	slope of the armour layer in radians
slope_foreshore	float	slope of the foreshore in radians
permeability	str	permeability of the core {permeable, low, impermeable}
Dn	float	nominal diameter of the armour units in the armour layer
layers	int	number of layers in the armour layer
B	float	crest width
beta	float	angle between direction of wave approach and a line normal to the breakwater in degrees

1.6 Design Automation

1.6.1 File Reader

breakwater.design.**read_breakwaters**(*filepath*)

Read a breakwaters file into Configurations

Parameters **filepath** (*path*) – any valid string path is acceptable, filepath must end with .breakwaters

Returns *bw.Configurations* – a Configurations object with breakwater concepts

1.6.2 Interactive Design Application

breakwater.interactive.**interactive_design**(*structure*, *LimitState*, *Grading*, *ArmourUnit=None*, *BermMaterial=None*, *cost=None*, *Soil=None*, *display_warnings=True*)

Interactive Design Application

Application to interactively design breakwaters. All parameters can be specified in the application, together with varying parameters. On the Interactive Design Page it is then possible to change the value of a varying parameter with a slider. The design can then be updated and a cross-section of the new design is shown.

Parameters

- **structure** (*{'RRM', 'CRM', 'RC', 'CC'}*) – structure for which conceptual designs must be generated. RRM for a rubble mound with rock as armour layer, CRM for

a rubble mound with concrete armour units as armour layer, RC for a vertical (composite) breakwater with rock as armour layer for the foundation and CC for a vertical (composite) breakwater with concrete armour units as armour layer for the foundation.

- **LimitState** (`LimitState` or list of `LimitState`) – ULS, SLS or another limit state defined with `LimitState`
- **Grading** (`RockGrading`) – standard rock grading defined in the NEN-EN 13383-1 or a user defined rock grading. Required for all parameters
- **ArmourUnit** (*obj, optional, default: None*) – armour unit class which inherits from `ConcreteArmour`, for instance `Xbloc` or `XblocPlus`. This argument is used for RRM.
- **BermMaterial** (*obj, optional, default: None*) – armour unit class which inherits from `ConcreteArmour`, for instance `Xbloc` or `XblocPlus`. This argument is used for CC.
- **Soil** (`Soil`, optional, default: None) – by default Soil is None, which means that the geotechnical checks are not performed. By specifying a Soil object, the geotechnical checks are automatically performed. Optional argument for RC and CC.
- **cost** (*dict, optional, default: None*) – by default no cost are computed, when the cost must be computed all relevant cost must be included. Optional keys of the dict are: core_price for the price of the core material, unit_price for the price of armour units, concrete_price for the price of concrete, fill_price for the price of the fill material of the caisson. If not included in the prices it is also possible to specify transport_price for the price of transporting rocks, dry_dock for the rent of a dry dock, which is divided through the length of the breakwater.
- **display_warnings** (*bool, optional, default: True*) – if warnings must be displayed when designing

1.6.3 Configurations Design

class breakwater.design.**Configurations**(*structure*, *LimitState*, *rho_w*, *slope_foreshore*, *Grading*, *Soil=None*, *safety=1*, ***kwargs*)

Make a conceptual design for multiple breakwaters

With this class a parametric design of one or more types of breakwaters. The types of structures included in the tool are: a rubble mound breakwater with rock (RRM) or concrete armour units (CRM) as armour layer and a vertical (composite) breakwater with rock (RC) or concrete armour units (CC) as armour layer of the foundation layer. Each structure has its own set of required (table 1) and optional (table 2) parameters.

For the parametric design some parameters are allowed to vary, these parameters have a superscript in table 1 and 2. The input of a varying parameter must be a tuple with a length of 3, i.e. (min, max, num), where min and max are the minimum and maximum value and num is the number of samples to generate.

Note: Alternatively, a design can also be made from an Excel input file by using *read_configurations()*. The required Excel input file can be generated by using *bw.generate_excel*

Table 1: required parameters

Parameter	RRM	CRM	RC	CC
LimitState	x	x	x	x
rho_w	x	x	x	x
slope_foreshore	x	x	x	x
Grading	x	x	x	x
slope [1]	x	x		
B [1]	x	x		
Dn50_core [1]	x	x		
N	x			
ArmourUnit		x		
Pc [1]			x	x
rho_c			x	x
rho_fill			x	x
Bm [1]			x	x
hb [1]			x	x
mu			x	x
BermMaterial				x

[1] Parameter is allowed to vary, enter as a tuple with (min, max, num), where min and max are the minimum and maximum value and num is the number of samples to generate.

Table 2: Optional parameters with default values

Parameter	Default	RRM	CRM	RC	CC
safety	1	o	o	o	o
beta	0	o	o	o	o
Soil	None	o	o	o	o
slope_toe [1]	(2,3)	o	o		
phi	40	o	o		
B_toe [1]	None	o	o		
vdm	max	o			
layers_rock	2	o		o	
layers_units	1		o		o
layers_underlayer	2	o	o		
filter_rule	None		o		o
pe_max	500			o	o
SF_sliding	1.2			o	o
SF_turning	1.2			o	o
slope_foundation [1]	(2,3)			o	o
lambda_	[1,1,1]			o	o

[1] Parameter is allowed to vary, enter as a tuple with (min, max, num), where min and max are the minimum and maximum value and num is the number of samples to generate.

Parameters

- **structure** (*{'RRM', 'CRM', 'RC', 'CC'}*) – structure for which conceptual designs must be generated. RRM for a rubble mound with rock as armour layer, CRM for a rubble mound with concrete armour units as armour layer, RC for a vertical (composite) breakwater with rock as armour layer for the foundation and CC for a vertical (composite) breakwater with concrete armour units as armour layer for the foundation.
- **LimitState** (`LimitState` or list of `LimitState`) – ULS, SLS or another limit state defined with `LimitState`

- **rho_w** (*float*) – density of water [kg/m^3]
- **slope_foreshore** (*tuple*) – slope of the foreshore (V, H). For example a slope of 1:100 is defined as (1, 100)
- **Grading** (RockGrading) – standard rock grading defined in the NEN-EN 13383-1 or a user defined rock grading.
- **safety** (*float, optional, default: 1*) – safety factor of design (number of standard deviations from the mean)

Keyword Arguments

- **slope** (*tuple*) – Slope of the armour layer (V,H). For example a slope of 3V:4H is defined as (3,4). Required for RRM and CRM.
- **B** (*float*) – Crest width [m], required for RRM and CRM
- **Dn50_core** (*float*) – nominal diameter for the stones in the core of the breakwater [m], required for RRM and CRM
- **N** (*int*) – Number of incident waves at the toe of the structure [-], required for RRM
- **ArmourUnit** (*obj*) – armour unit class which inherits from ConcreteArmour, for instance Xbloc or XblocPlus. Required argument for CRM.
- **Pc** (*float*) – contribution of concrete to the total mass of the caisson. value between 0 and 1. Required argument for RC and CC.
- **rho_c** (*float*) – density of concrete [kg/m^3], required argument for RC and CC.
- **rho_fill** (*float*) – density of the fill material [kg/m^3], required argument for RC and CC.
- **Bm** (*float*) – width of the berm [m], required argument for RC and CC.
- **hb** (*float*) – height of the foundation layer [m], required argument for RC and CC.
- **mu** (*float*) – friction factor between the caisson and the foundation [-], required argument for RC and CC.
- **BermMaterial** (*obj*) – should be a RockGrading or armour unit class which inherits from ConcreteArmour, for instance Xbloc or XblocPlus. Required argument for CC
- **beta** (*float, optional, default: 0*) – angle between direction of wave approach and a line normal to the breakwater (degrees). Optional argument for RC and CC, default value is 0.
- **slope_toe** (*tuple, optional, default: (2,3)*) – Slope of the armour layer (V,H). For example a slope of 2V:3H is defined as (2,3). Optional argument for RRM and CRM, default value is (2,3)
- **B_toe** (*float, optional, default: None*) – width of the top of the toe in meters. By default the width of toe is taken as 3 * Dn50_toe. Optional argument for RRM and CRM.
- **vdm** (*{min, max, avg}, optional, default: max*) – value to return in case both the deep and shallow water formula are valid. min for the lowest value, max for the highest value and avg for the average value, default is max. Optional argument for RRM.
- **layers_rock** (*int, optional, default: 2*) – number of layer in the armour layer for a rubble mound breakwater with rock as armour layer. Optional argument for RRM and RC, default value is 2

- **layers_units** (*int, optional, default: 1*) – number of layer in the armour layer for a rubble mound breakwater with concrete armour units as armour layer. Optional argument for RRM and CC, default value is 1
- **layers_underlayer** (*int, optional, default: 2*) – number of layers in the underlayer. Optional argument for RRM and CRM
- **filter_rule** (*{'Rock', 'Xbloc', 'XblocPlus'}, optional, default: None*) – filter rule to use for the substructure of the breakwater, for Rock, Xbloc and XblocPlus the correct filter rule is automatically selected. In case another type of armour layer is used one of these filter rules must be chosen. Optional argument for CRM, required if armour unit is not Xbloc or XblocPlus, default value is None
- **pe_max** (*float, optional, default: 500*) – maximum value of the bearing pressure at the heel of the caisson. Default value is set to 500 kPa, Goda (2000) advises a value between 400 and 500 kPa. Optional parameter for RC and CC
- **SF_sliding** (*float, optional, default: 1.2*) – safety factor against sliding. Default value according to Goda (2000). Optional argument for RC and CC, default value is 1.2
- **SF_turning** (*float, optional, default: 1.2*) – safety factor against sliding. Default value according to Goda (2000). Optional argument for RC and CC, default value is 1.2
- **slope_foundation** (*tuple, optional, default: (2,3)*) – Slope of the armour layer (V,H). For example a slope of 2V:3H is defined as (2,3). Optional argument for RC and CC, default is (2,3)
- **lambda** (*list, optional, default: [1,1,1]*) – modification factors of Takahasi (2002) for alternative monolithic breakwater. Input must be lambda_= [$\lambda_1, \lambda_2, \lambda_3$]. Optional argument for RC and CC, default value is [1,1,1]
- **Soil** (Soil, optional, default: None) – by default Soil is None, which means that the geotechnical checks are not performed. By specifying a Soil object, the geotechnical checks are automatically performed. Optional argument for all structures
- **phi** (*float, optional, default: 40*) – internal friction angle of rock [degrees]. Optional argument for RRM and CRM.

df

DataFrame with all generated concepts

Type pd.DataFrame

add_cost(*core_price=None*, *unit_price=None*, *concrete_price=None*, *fill_price=None*, *transport_cost=None*, *investment=None*, *length=None*)

Compute the cost of each concept

Compute the cost of each concept and add the cost to the *df*. The cost of the rocks must be specified in the RockGrading. If transport cost are not included in the price of rocks or core_price it can be given with the argument transport_cost. For a Caisson breakwater it is possible to specify the investment for renting a dry dock, the investment is divided through the length of the breakwater to get the investment cost per meter.

Note: The transport_cost are not added to the price of the armour layer. The assumption has been made that the cost of producing and transporting the armour units is included in the unit_price.

Parameters

- **core_price** (*float, optional, default: None*) – cost of the core material per m^3, required for RRM and CRM
- **unit_price** (*float, optional, default: None*) – the cost of an armour unit per m^3, required for CRM and CC
- **concrete_price** (*float, optional, default: None*) – price of concrete per m^3, required for RC and CC
- **fill_price** (*float, optional, default: None*) – price of the fill material per m^3, required for RC and CC
- **transport_cost** (*float, optional, default: None*) – the cost to transport a m^3 of rock from the quarry to the project location
- **investment** (*float*) – the investment required to rent a dry dock
- **length** (*float*) – length of the breakwater [m]

cost_influence()
Plot the influence of the varying parameters on the cost

Method to see the influence of the varying parameters on the cost of a concept. If more than 1 parameter has been specified as a varying parameter the x-axis is normalised (from 0 to 1) so that all parameters can be plotted on the same axis.

Note: When more than one varying parameter is specified several lines must be plotted. To show the influence of one varying parameter on the cost the other varying parameters are set to their lower bound value.

---

get_concept(*id=None*, *CaseNo=None*)
Get the specified concept

Parameters

- **id** (*str, optional, default: None*) – id of the concept, the id of a concept for a rubble mound breakwater out of rock is for instance: RRM.1
- **CaseNo** (*int, optional, default: None*) – CaseNo if the concept, only available if an export for the Design Explorer has been made with *to_design_explorer()*

Returns **concept** (*obj*) – a breakwater concept, for instance `RockRubbleMound`

Raises

- **InputError** – if a concept is not selected with CaseNo or id, or if there is no concept with the specified CaseNo or id
- **KeyError** – if a concept can't be selected by CaseNo because the method *to_design_explorer()* has not yet been used, and the CaseNo have therefore not yet been added to *df*

show_warnings()
Print all warnings encountered during the design

Method prints a table containing the unique warning messages, with the time the warning was encountered during the design.

to_breakwaters(*save_name*)
Save the df with designs in a pickle object

Method will save the df with all breakwater concepts in a pickle object with extension .breakwaters.

Parameters **save_name** (*str*) – name of the file

to_design_explorer(*params*, *mkdir='DesignExplorer'*, *slopes='angles'*, *merge_Bm=True*, *merge_slope_toe=True*)

Export concepts to Design Explorer 2

Creates a folder that can be used in Design Explorer 2, the folder consists of the cross sections of all concepts and a csv file with the data of the concepts. Parameters supported for export can be seen in table 3. To use the folder in Design Explorer 2, follow these steps:

- Upload the folder to your Google Drive
- Share the folder and get the shareable link
- Go to http://tt-acm.github.io/DesignExplorer/
- Click on *Get Data* and paste the shareable link below: *From the cloud*
- Click on *Load Data*
- Enjoy exploring!

Table 3: possible parameter to export

Parameter	RRM + CRM	RC + CC
cost	o	o
B	o	o
Rc	o	o
computed Dn50 armour	o	o
class armour	o	o
class Dn50 armour	o	o
computed Dn50 underlayer	o	o
class underlayer	o	o
class Dn50 underlayer	o	o
slope	o	
slope_toe	o	o [1]
Dn50_core	o	
B_toe	o	
computed Dn50 filter	o	
class filter	o	
class Dn50 filter	o	
Pc		o
hb		o
h_acc		o
Bm	o [2]	o
slope_foundation		o
UC [3]		o

[1] slope_foundation is interpreted as slope_toe if merge_slope_toe is set to True

[2] B_toe is interpreted as Bm if merge_Bm is set to True

[3] UC is the unity check for the bearing capacity of the subsoil

Parameters

- **params** (*list*) – list of parameters for the design explorer, parameter must be str. See table 3 for the parameters that can be exported/
- **mkdir** (*str, optional, default: concepts*) – creates a folder in which all cross sections are saved, upload the created folder to your Google Drive to use Design Explorer 2
- **slopes** (*{angles, tuples}, optional, default: angles*) – how the slopes must be exported. tuples will export the slope as (V,H) and angles as an angle in degrees
- **merge_Bm** (*bool, optional, default: True*) – True if Bm must be merged with B_toe when a Rubble Mound and Vertical breakwater have been designed, False if you do not want to merge the columns.
- **merge_slope_toe** (*bool, optional, default: True*) – True if slope_foundation must be merged with slope_toe when a Rubble Mound and Vertical breakwater have been designed, False if you do not want to merge the columns.

Raises **KeyError** – if the cost are asked to export but not yet specified

1.7 Breakwater types

There are several types of breakwaters, the different types can be divided into two categories. Rubble mound breakwaters which are made out of large heaps of loose elements, and monolithic breakwaters which have a cross section acting as one block, for instance a caisson. Figure 7.1 depicts the representative cross sections for all breakwater types defined by CIRIA, CUR, CETMEF (2007, p. 781).

The following breakwater types from Figure 7.1 have been implemented: conventional rubble mound breakwater, caisson breakwater and the vertically composite breakwater. For each of these structures a class is defined with which a conceptual design can be made. These classes automatically use the functions and classes from the `breakwater.core`, see Figure 7.2.

1.7.1 Conventional Rubble Mound

As mentioned in the introduction a rubble mound breakwater is made out of large heaps of loose elements, the armour layer of these types are made with either rock or concrete armour units such as Xbloc or XblocPlus. Both types of armour layer can be used to design a breakwater. Figure 7.3 depicts the used definition of a conventional rubble mound breakwater, for both types of armour layer.

Rock

class breakwater.rubble.**RockRubbleMound**(*slope*, *slope_foreshore*, *rho_w*, *B*, *N*, *LimitState*, *Grading*, *Dn50_core*, *safety=1*, *slope_toe=(2, 3)*, *B_toe=None*, *beta=0*, *layers=2*, *layers_underlayer=2*, *vdm='max'*, *Soil=None*, *phi=40*, *id=None*, ***kwargs*)

Design a breakwater with Rock as armour layer

Makes a conceptual design for a conventional rubble mound breakwater with rock as the armour layer, for one or several limit states. The following computations are performed:

- The armour layer is designed with the Van der Meer formulas for deep and shallow water (van der Meer, 1988; van Gent et al., 2003).

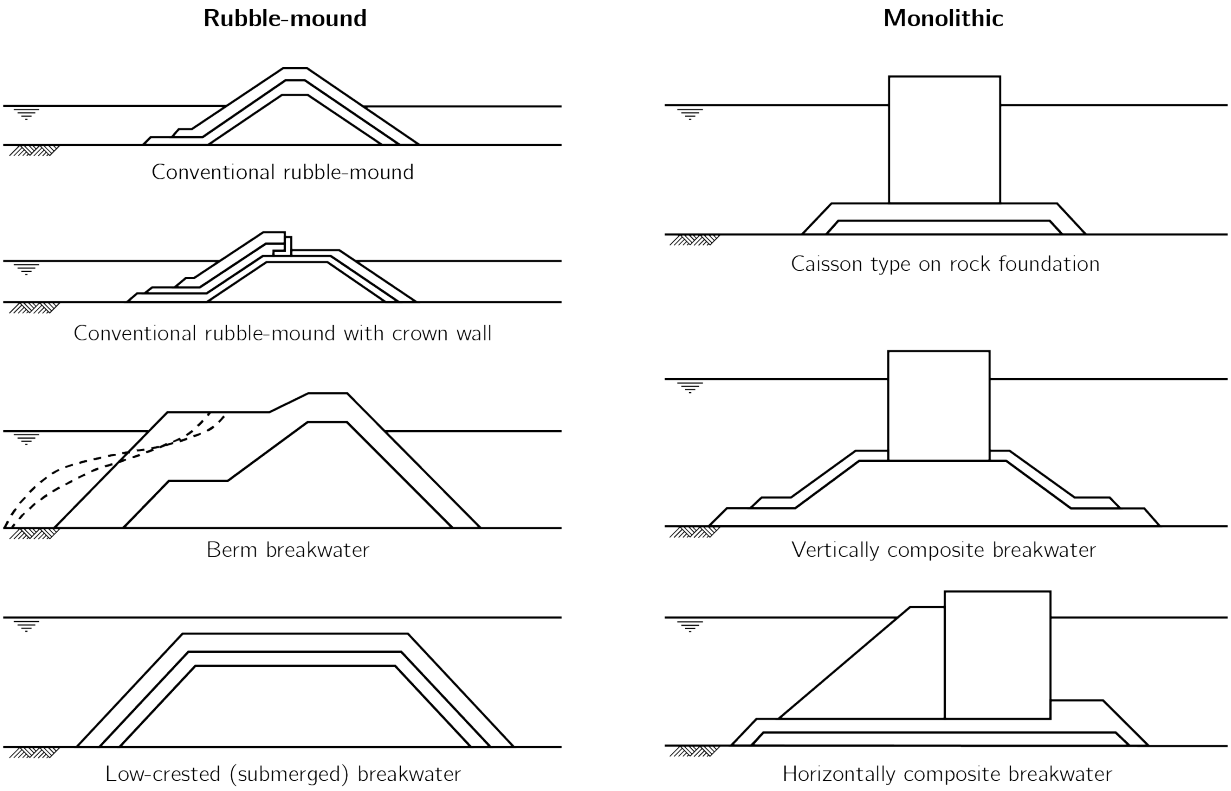


Fig. 3: Figure 7.1: Representative cross section of various types of breakwaters, Rubble mound types on the left and the monolithic types on the right. Redrawn from CIRIA, CUR, CETMEF (2007, p. 781).

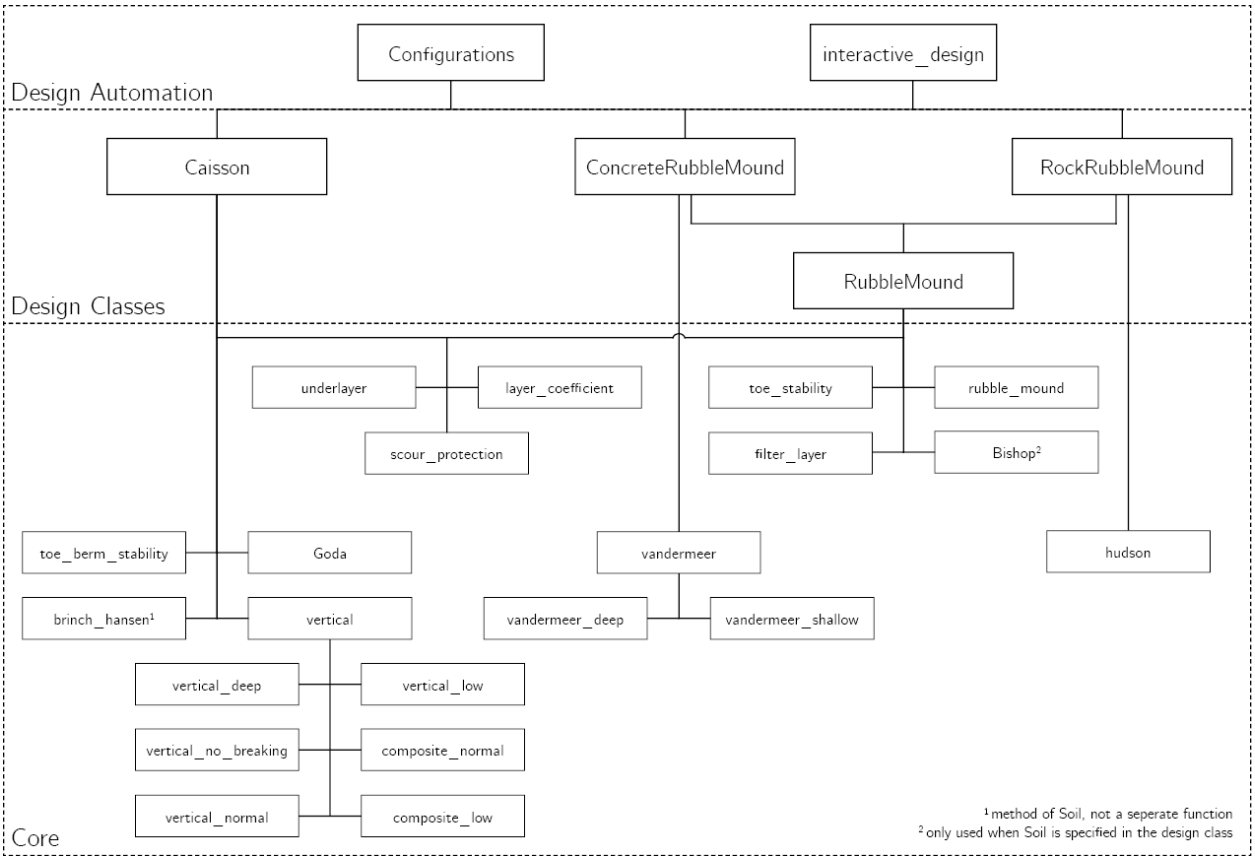


Fig. 4: Figure 7.2: Implementation of the `breakwater.core`, all functions and classes from Chapters 8 to 11, in the design classes.

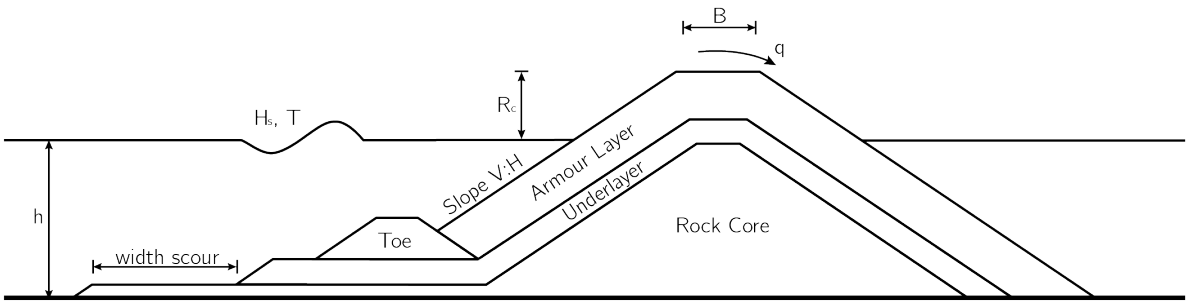


Fig. 5: Figure 7.3: Definitions for a rubble mound breakwater

- The underlayer is designed by using the rules for the underlayer
- A filter layer is designed if one is needed, depends on `Dn50_core`
- The toe is designed with the toe stability formula of Van der Meer (1998).
- The crest freeboard is computed with the formula from EurOtop (2018)
- The required width of the scour protection with Sumer and Fredsoe (2000)
- If a `Soil` is specified, a slip circle analysis is performed with `Bishop`

Note: Depending on the input it might be that more rock classes are possible for the underlayer (and filter layer). In case the upper bound of the underlayer rule results in a different rock class as the lower bound, a new variant is generated. See *variantIDs* for a list of generated variants.

Note: The notional permeability, P, in the van der Meer formula is set to a constant value of 0.4. This is due to the fact that the substructure of the breakwater will always have an underlayer.

Parameters

- **slope** (*tuple*) – Slope of the armour layer (V, H). For example a slope of 3V:4H is defined as (3, 4)
- **slope_foreshore** (*tuple*) – slope of the foreshore (V, H). For example a slope of 1:100 is defined as (1, 100)
- **rho_w** (*float*) – density of water [kg/m^3]
- **B** (*float*) – Crest width [m]
- **N** (*int*) – Number of incident waves at the toe of the structure [-]
- **LimitState** (`LimitState` or list of `LimitState`) – ULS, SLS or another limit state defined with `LimitState`
- **Grading** (`RockGrading`) – standard rock grading defined in the NEN-EN 13383-1 or a user defined rock grading
- **Dn50_core** (*float*) – nominal diameter for the stones in the core of the breakwater [m]
- **safety** (*float, optional, default: 1*) – safety factor of design (number of standard deviations from the mean)
- **slope_toe** (*tuple, optional, default: (2,3)*) – slope of the toe
- **B_toe** (*float, optional, default: None*) – width of the top of the toe in meters. By default the width of toe is taken as 3 * Dn50_toe.
- **beta** (*float, optional, default: 0*) – angle between direction of wave approach and a line normal to the breakwater (degrees).
- **layers** (*int, optional, default: 2*) – number of layers in the armour layer
- **layers_underlayer** (*int, optional, default: 2*) – number of layers in the underlayer
- **vdm** (*{min, max, avg}, optional, default: max*) – value to return in case both the deep and shallow water formula are valid. min for the lowest value, max for the highest value and avg for the average value, default is max.

- **Soil** (Soil, optional, default: None) – by default Soil is None, which means that the geotechnical checks are not performed. By specifying a Soil object, the geotechnical checks are automatically performed.
- **phi** (*float, optional, default: 40*) – internal friction angle of rock [degrees]
- **id** (*int, optional, default: None*) – add a unique id to the breakwater

logger
dict of warnings and messages

Type dict

structure
dictionary with the computed Dn50, rock class, and average Dn50 of the rock class for each layer and the toe. This dictionary includes all variants, use *get_variant()* to get the parameters of one specific variant. Alternatively, *print_variant()* can be used to print the details of one, multiple or all variants.

Type dict

alpha
slope of the structure in radians

Type float

id
unique id of the breakwater

Type int

variantIDs
list with the IDs of the variants generated for this rubble mound breakwater.

Type list

Rc
the crest freeboard of the structure [m]

Type float

width_scour
the required length of the scour protection [m]

Type float

area(*variantID*)
Compute the area of all layers

Method computes the area of each layer using Gauss's area formula. Which is given by the following formula:

$$\mathbf{A} = \frac{1}{2}|\sum_{i=1}^{n-1} x_i y_{i+1} + x_n y_1 - \sum_{i=1}^{n-1} x_{i+1} y_i - x_1 y_n|$$

Parameters **variantID** (*str*) – identifier of the variant, see *variantIDs* for a list of all generated variants.

Returns *dict* – dict with the area of each layer

check_validity(*decimals=3*)
Check if the used parameters are within the validity range

The Van der Meer equations for deep and shallow water are empirical equations, meaning that they are based on experiments. Therefore, the Van der Meer equations are, strictly speaking, only valid if the

parameters are within the range of the parameters from the experiments. This method prints a table from which the validity range, used value and if the parameter is within validity range can be read.

Parameters **decimals** (*int, optional, default: 3*) – number of decimals

cost (**variants*, *core_price*, *transport_cost=None*, *output='variant'*)
Compute the cost per meter for each variant

Method to compute the cost of each generated variant, the cost is computed per meter. The cost of the rocks must be specified in the RockGrading. If transport cost are not included in the price of rocks or core_price it can be given with the argument transport_cost.

Parameters

- ***variants** (*str*) – IDs of the variants to plot, see *variantIDs* for a list of all generated variants. If 'all' is in the arguments, all variants will be plotted.
- **core_price** (*float*) – cost of the core material per m^3
- **transport_cost** (*float, optional, default: None*) – the cost to transport a m^3 of rock from the quarry to the project location
- **output** (*{variant, layer, average}*) – format of the output dict, variant returns the total cost of each variant, layer the cost of each layer for each variant and average returns the average cost.

Returns *dict* – the cost

Raises **RockGradingError** – if no pricing is included in the given RockGrading

get_variant (*variantID*)
Get the dimensions for the specified variant

Parameters **variantID** (*str*) – identifier of the variant, see *variantIDs* for a list of all generated variants.

Returns *dict* – Parameters and values (Dn50, Rock class) for one variant

Raises **KeyError** – If there is no variant with the given identifier

plot (**variants*, *wlev=None*, *save_name=None*, *show_wave=True*)
Plot the cross section of the specified breakwater(s)

Parameters

- ***variants** (*str*) – IDs of the variants to plot, see *variantIDs* for a list of all generated variants. If 'all' is in the arguments, all variants will be plotted.
- **wlev** (*str, optional, default: None*) – label of the `LimitState` from which the water level will be plotted. If no value is specified the water level from the normative limit state is used, which is the normative LimitState from the crest freeboard computation.
- **save_name** (*str, optional, default: None*) – if given the cross section is not shown but saved with the given name
- **show_wave** (*bool, optional, default: True*) – True if the a wave with wave height Hs must be plotted, False if no wave height must be plotted. Note that if a wave height is plotted the wave length is scaled.

Raises

- **InputError** – If no variants are specified or if the label of wlev is not a valid label of a `LimitState`

- **KeyError** – If there is no variant with the given identifier

print_logger(*level='warnings'*)
Print messages and warnings from the logger

Parameters **msg_level** (*{'info', 'warnings'}, optional, default: 'warnings'*) – specify print level, highest level is warnings and lowest level is info. Note that the info level will also print all warnings

print_variant(**variants*, *decimals=3*)
Print the details for the specified variant(s)

This method will print the computed Dn50, rock class, average Dn50 of the class, normative LimitState for all layers and specified variants.

Parameters

- ***variants** (*str*) – IDs of the variants to plot, see *variantIDs* for a list of all generated variants. If 'all' is in the arguments, all variants will be plotted.
- **decimals** (*int, optional, default: 3*) – number of decimal places to round to

Raises

- **InputError** – If no arguments are specified
- **KeyError** – If there is no variant with the given identifier

Armour Units

class breakwater.rubble.**ConcreteRubbleMound**(*slope*, *slope_foreshore*, *B*, *rho_w*, *LimitState*, *ArmourUnit*, *Grading*, *Dn50_core*, *safety=1*, *slope_toe=(2, 3)*, *B_toe=None*, *beta=0*, *layers=1*, *layers_underlayer=2*, *filter_rule=None*, *Soil=None*, *phi=40*, *id=None*, ***kwargs*)
Design a breakwater with concrete armour units as armour layer

Makes a conceptual design for a conventional rubble mound breakwater with armour units as the armour layer, for one or several limit states. The following computations are performed:

- The armour layer is designed with the Hudson formula (Hudson, 1959), with $H_{1/3}$ as wave height, in accordance with the design guidelines for Xbloc and XblocPlus (Delta Marine Consultants, 2018).
- The underlayer is designed by using the rules for the underlayer
- A filter layer is designed if one is needed, depends on `Dn50_core`
- The toe is designed with the toe stability formula of Van der Meer (1998).
- The crest freeboard is computed with the formula from EurOtop (2018)
- The required width of the scour protection with Sumer and Fredsoe (2000)
- If a `Soil` is specified, a slip circle analysis is performed with `Bishop`

Note: Depending on the input it might be that more rock classes are possible for the underlayer (and filter layer). In case the upper bound of the underlayer rule results in a different rock class as the lower bound, a new variant is generated. See *variantIDs* for a list of generated variants.

Warning: Currently only the rules for the underlayer of Rock, Xbloc and XblocPlus have been implemented. However, it is possible to design with another type of armour units. In this case the filter rule must manually be set to Rock, Xbloc or XblocPlus.

Parameters

- **slope** (*tuple*) – Slope of the armour layer (V, H). For example a slope of 3V:4H is defined as (3, 4)
- **slope_foreshore** (*tuple*) – slope of the foreshore (V, H). For example a slope of 1:100 is defined as (1, 100)
- **B** (*float*) – Crest width [m]
- **rho_w** (*float*) – density of water [kg/m^3]
- **LimitState** (`LimitState` or list of `LimitState`) – ULS, SLS or another limit state defined with `LimitState`
- **ArmourUnit** (*obj*) – armour unit class which inherits from `ConcreteArmour`, for instance `Xbloc` or `XblocPlus`
- **Grading** (`RockGrading`) – standard rock grading defined in the NEN-EN 13383-1 or a user defined rock grading
- **Dn50_core** (*float*) – nominal diameter for the stones in the core of the breakwater [m]
- **safety** (*float, optional, default: 1*) – safety factor of design (number of standard deviations from the mean)
- **slope_toe** (*tuple, optional, default: (2,3)*) – slope of the toe
- **B_toe** (*float, optional, default: None*) – width of the top of the toe in meters. By default the width of toe is taken as 3 * Dn50_toe.
- **beta** (*float, optional, default: 0*) – angle between direction of wave approach and a line normal to the breakwater (degrees).
- **layers** (*int, optional, default: 1*) – number of layers in the armour layer
- **layers_underlayer** (*int, optional, default: 2*) – number of layers in the underlayer
- **filter_rule** (*{'Rock', 'Xbloc', 'XblocPlus'}, optional, default: None*) – filter rule to use for the substructure of the breakwater, for Rock, Xbloc and XblocPlus the correct filter rule is automatically selected. In case another type of armour layer is used one of these filter rules must be chosen.
- **Soil** (`Soil`, optional, default: None) – by default Soil is None, which means that the geotechnical checks are not performed. By specifying a Soil object, the geotechnical checks are automatically performed.
- **phi** (*float, optional, default: 40*) – internal friction angle of rock [degrees]
- **id** (*int, optional, default: None*) – add a unique id to the breakwater

logger
dict of warnings and messages

Type dict

structure
dictionary with the computed Dn50, rock class, and average Dn50 of the rock class for each layer and the toe. This dictionary includes all variants, use *get_variant()* to get the parameters of one specific variant. Alternatively, *print_variant()* can be used to print the details of one, multiple or all variants.

Type dict

alpha
slope of the structure in radians

Type float

id
unique id of the breakwater

Type int

variantIDs
list with the IDs of the variants generated for this rubble mound breakwater.

Type list

Rc
the crest freeboard of the structure [m]

Type float

width_scour
the required length of the scour protection [m]

Type float

area(*variantID*)
Compute the area of all layers

Method computes the area of each layer using Gauss's area formula. Which is given by the following formula:

$$\mathbf{A} = \frac{1}{2}|\sum_{i=1}^{n-1} x_i y_{i+1} + x_n y_1 - \sum_{i=1}^{n-1} x_{i+1} y_i - x_1 y_n|$$

Parameters **variantID** (*str*) – identifier of the variant, see *variantIDs* for a list of all generated variants.

Returns *dict* – dict with the area of each layer

cost(**variants*, *core_price*, *unit_price*, *transport_cost=None*, *output='variant'*)
Compute the cost per meter for each variant

Method to compute the cost of each generated variant, the cost is computed per meter. The cost of the rocks in the substructure must be specified in the RockGrading. If transport cost are not included in the price of rocks or core_price it can be given with the argument transport_cost.

Note: The transport_cost are not added to the price of the armour layer. The assumption has been made that the cost of producing and transporting the armour units is included in the unit_price.

Parameters

- ***variants** (*str*) – IDs of the variants to plot, see *variantIDs* for a list of all generated variants. If 'all' is in the arguments, all variants will be plotted.
- **core_price** (*float*) – cost of the core material per m^3

- **unit_price** (*float*) – the cost of an armour unit per m^3
- **transport_cost** (*float, optional, default: None*) – the cost to transport a m^3 of rock from the quarry to the project location
- **output** (*{variant, layer, average}*) – format of the output dict, variant returns the total cost of each variant, layer the cost of each layer for each variant and average returns the average cost.

Returns *dict* – the cost

Raises **RockGradingError** – if no pricing is included in the given RockGrading

get_variant(*variantID*)

Get the dimensions for the specified variant

Parameters **variantID** (*str*) – identifier of the variant, see *variantIDs* for a list of all generated variants.

Returns *dict* – Parameters and values (Dn50, Rock class) for one variant

Raises **KeyError** – If there is no variant with the given identifier

plot(**variants*, *wlev=None*, *save_name=None*, *show_wave=True*)

Plot the cross section of the specified breakwater(s)

Parameters

- ***variants** (*str*) – IDs of the variants to plot, see *variantIDs* for a list of all generated variants. If 'all' is in the arguments, all variants will be plotted.
- **wlev** (*str, optional, default: None*) – label of the `LimitState` from which the water level will be plotted. If no value is specified the water level from the normative limit state is used, which is the normative LimitState from the crest freeboard computation.
- **save_name** (*str, optional, default: None*) – if given the cross section is not shown but saved with the given name
- **show_wave** (*bool, optional, default: True*) – True if the a wave with wave height Hs must be plotted, False if no wave height must be plotted. Note that if a wave height is plotted the wave length is scaled.

Raises

- **InputError** – If no variants are specified or if the label of wlev is not a valid label of a `LimitState`
- **KeyError** – If there is no variant with the given identifier

print_logger(*level='warnings'*)

Print messages and warnings from the logger

Parameters **msg_level** (*{'info', 'warnings'}, optional, default: 'warnings'*) – specify print level, highest level is warnings and lowest level is info. Note that the info level will also print all warnings

print_variant(**variants*, *decimals=3*)

Print the details for the specified variant(s)

This method will print the computed Dn50, rock class, average Dn50 of the class, normative LimitState for all layers and specified variants.

Parameters

- ***variants** (*str*) – IDs of the variants to plot, see *variantIDs* for a list of all generated variants. If 'all' is in the arguments, all variants will be plotted.
- **decimals** (*int, optional, default: 3*) – number of decimal places to round to

Raises

- **InputError** – If no arguments are specified
- **KeyError** – If there is no variant with the given identifier

1.7.2 (Composite) vertical

The caisson type and vertical composite breakwater are included in one design class as they are basically the same structures. The main difference is the water depth immediately in front of the caisson. In this package the classification criteria from EurOtop (2018) is used, which classifies a vertical breakwater as vertical if $\frac{d}{h} > 0.6$, else the breakwater is classified as a vertically composite breakwater. In Figure 7.4 the definition of all parameters is depicted.

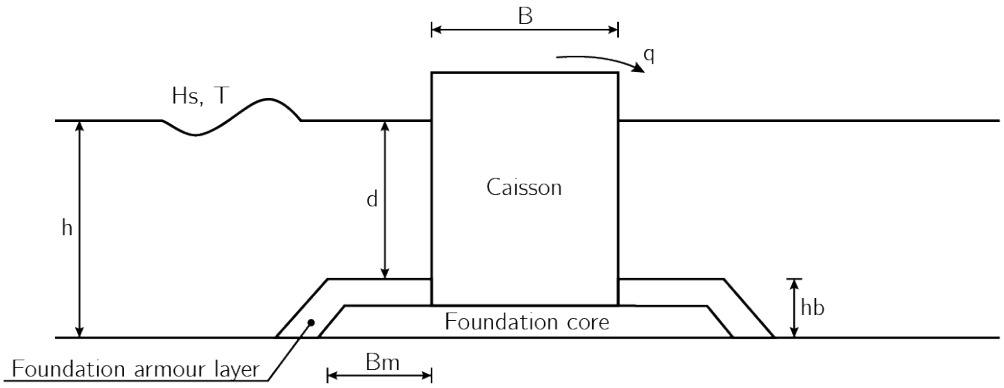


Fig. 6: Figure 7.4: Definitions for a (composite) vertical breakwater

class breakwater.caisson.**Caisson**(*Pc, rho_c, rho_fill, rho_w, Bm, hb, layers, BermMaterial, LimitState, slope_foreshore, mu, safety=1, SF_sliding=1.2, SF_turning=1.2, beta=0, slope_foundation=(2, 3), lambda_=[1, 1, 1], pe_max=500, filter_rule=None, Grading=None, Soil=None, id=None, **kwargs*)

Design a (composite) vertical breakwater

Makes a conceptual design of a vertical or composite vertical breakwater, with a caisson on a rubble mound foundation. The following computations are performed:

- The necessary size of the armour layer of the foundation is designed with the modified Tanimoto formula (Takahashi, 2002).
- The required stone size for the core of the foundation
- The water depth in front of the caisson is computed based on the dimensions of the foundation and water depth
- The crest freeboard is computed with the formulae from EurOtop (2018), `vertical()` is used, which automatically classifies the breakwater so that the correct formula is used.
- The required width of the caisson is computed with the extended Goda formula (Takahasi, 2002).

- The required width of the scour protection with Sumer and Fredsoe (2000). Note that a scour protection is only added if the width of the foundation is not sufficient.
- If a Soil is specified the bearing capacity of the soil will also be checked with Brinch Hansen (1970).

Parameters

- **Pc** (*float*) – contribution of concrete to the total mass of the caisson. value between 0 and 1
- **rho_c** (*float*) – density of concrete [kg/m^3]
- **rho_fill** (*float*) – density of the fill material [kg/m^3]
- **rho_w** (*float*) – density of water [kg/m^3]
- **Bm** (*float*) – width of the berm [m]
- **hb** (*float*) – height of the foundation layer [m]
- **layers** (*int*) – number of layers in the armour layer of the toe berm
- **BermMaterial** (*obj*) – should be a `RockGrading` or armour unit class which inherits from `ConcreteArmour`, for instance `Xbloc` or `XblocPlus`
- **LimitState** (`LimitState` or list of `LimitState`) – ULS, SLS or another limit state defined with `LimitState`
- **slope_foreshore** (*tuple*) – slope of the foreshore (V, H). For example a slope of 1:100 is defined as (1,100)
- **mu** (*float*) – friction factor between the caisson and the foundation [-]
- **safety** (*float, optional, default: 1*) – safety factor of design (number of standard deviations from the mean)
- **SF_sliding** (*float, optional, default: 1.2*) – safety factor against sliding. Default value according to Goda (2000)
- **SF_turning** (*float, optional, default: 1.2*) – safety factor against sliding. Default value according to Goda (2000)
- **beta** (*float, optional, default: 0*) – angle between direction of wave approach and a line normal to the breakwater [degrees]
- **slope_foundation** (*tuple, optional, default: (2,3)*) – Slope of the armour layer (V, H). For example a slope of 2V:3H is defined as (2,3)
- **lambda** (*list, optional, default: [1, 1, 1]*) – modification factors of Takahasi (2002) for alternative monolithic breakwater. Input must be lambda_= [$\lambda_1, \lambda_2, \lambda_3$].
- **pe_max** (*float, optional, default: 500*) – maximum value of the bearing pressure at the heel of the caisson. Default value is set to 500 kPa, Goda (2000) advises a value between 400 and 500 kPa. [kPa]
- **filter_rule** (*{'Rock', 'Xbloc', 'XblocPlus'}, optional, default: None*) – filter rule to use for the substructure of the breakwater, for rock, Xbloc and XblocPlus the correct filter rule is automatically selected. In case another type of armour layer is used one of these filter rules must be chosen.
- **Grading** (`RockGrading`, optional, default: None) – standard rock grading defined in the NEN-EN 13383-1 or a user defined rock grading. Required if the BermMaterial is not a `RockGrading`.

- **Soil** (Soil, optional, default: None) – by default Soil is None, which means that the geotechnical checks are not performed. By specifying a Soil object, the geotechnical checks are automatically performed.
- **id** (*int, optional, default: None*) – unique id of the breakwater

logger
dict of warnings and messages

Type dict

structure
dictionary with the computed Dn50, rock class, and average Dn50 of the rock class for each layer and the toe. This dictionary includes all variants, use *get_variant()* to get the parameters of one specific variant. Alternatively, *print_variant()* can be used to print the details of one, multiple or all variants.

Type dict

id
unique id of the breakwater

Type int

variantIDs
list with the IDs of the variants generated for this rubble mound breakwater.

Type list

price
cost of each variant, generated when *cost()* or *dry_dock()* is used.

Type dict

width_scour
the required length of the scour protection [m]. The distance is measured from the front of the caisson

Type float

geotechnical
dictionary with the computed values from the Brinch Hansen equation, has keys: Fr with the resistance, N with the net downward force and UC, which is the unity check (N/Fr). This dictionary is generated when the Soil argument is specified.

Type dict

area(*variantID*)
Compute the area of all layers

Method computes the area of each layer using Gauss's area formula. Which is given by the following formula:

$$\mathbf{A} = \frac{1}{2}|\sum_{i=1}^{n-1} x_i y_{i+1} + x_n y_1 - \sum_{i=1}^{n-1} x_{i+1} y_i - x_1 y_n|$$

Parameters **variantID** (*str*) – identifier of the variant, see *variantIDs* for a list of all generated variants.

Returns *dict* – dict with the area of each layer

cost(**variants*, *concrete_price*, *fill_price*, *unit_price=None*, *output='variant'*)
Compute the cost per meter for each variant

Method to compute the cost of each generated variant, the cost is computed per meter

Parameters

- ***variants** (*str*) – IDs of the variants to plot, see *variantIDs* for a list of all generated variants. If 'all' is in the arguments, all variants will be plotted.
- **concrete_price** (*float*) – price of concrete per m^3
- **fill_price** (*float*) – price of the fill material per m^3
- **unit_price** (*float, optional, default: None*) – the cost of an armour unit per m^3, required if the armour layer of the foundation is made out of armour units
- **output** (*{variant, layer, average}*) – format of the output dict, variant returns the total cost of each variant, layer the cost of each layer for each variant and average returns the average cost.

Returns *dict* – the cost

Raises **RockGradingError** – if no pricing is included in the given RockGrading

dry_dock(*investment*, *length*)

Add the investment cost of a dry dock to the concept

This method adds the investment required to rent a dry dock to the concept. The investment cost is added to the concept by dividing the investment trough the length of the breakwater to get the required investment per running meter.

Parameters

- **investment** (*float*) – the investment required to rent a dry dock
- **length** (*float*) – length of the breakwater [m]

get_variant(*variantID*)

Get the dimensions for the specified variant

Parameters **variantID** (*str*) – identifier of the variant, see *variantIDs* for a list of all generated variants.

Returns *dict* – Parameters and values of the caisson (B, Rc) and the foundation layer (Dn50, Rock class) for one variant.

Raises **KeyError** – If there is no variant with the given identifier

plot(**variants*, *wlev=None*, *save_name=None*, *show_wave=True*)

Plot the cross section of the specified breakwater(s)

Parameters

- ***variants** (*str*) – IDs of the variants to plot, see *variantIDs* for a list of all generated variants. If 'all' is in the arguments, all variants will be plotted.
- **wlev** (*str, optional, default: None*) – label of the `LimitState` from which the water level will be plotted. If no value is specified the water level from the normative limit state is used, which is the normative LimitState from the crest freeboard computation.
- **save_name** (*str, optional, default: None*) – if given the cross section is not shown but saved with the given name
- **show_wave** (*bool, optional, default: True*) – True if the a wave with wave height Hs must be plotted, False if no wave height must be plotted. Note that if a wave height is plotted the wave length is scaled.

Raises

- **InputError** – If no variants are specified or if the label of wlev is not a valid label of a LimitState
- **KeyError** – If there is no variant with the given identifier

plot_pressure()
Plot pressure distribution computed extended Goda formula

Plots the pressure distribution computed with the extended Goda formula (Takahasi, 2002) together with the dimensions of the monolithic breakwater.

> **Warning:** Do not read the dimensions of the monolithic breakwater from the axes of the figure. The correct dimensions of the monolithic breakwater are depicted in the figure, or use *get_variant()* or *print_variant()*.

print_logger(*level='warnings'*)
Print messages and warnings in the logger

Parameters **msg_level** (*{'info', 'warnings'}, optional, default: 'warnings'*) – specify print level, highest level is warnings and lowest level is info. Note that the info level will also print all warnings

print_variant(**variants*, *decimals=3*)
Print the details for the specified variant(s)

This method will print the details of the structure for the specified variant(s). It prints the dimensions of the caisson (B, Rc) and the dimensions of the foundation layer (Dn50, rock class). Furthermore, from the table the normative LimitState for the design can be read.

Parameters

- ***variants** (*str*) – IDs of the variants to plot, see *variantIDs* for a list of all generated variants. If 'all' is in the arguments, all variants will be plotted.
- **decimals** (*int, optional, default: 3*) – number of decimal places to round to

Raises

- **InputError** – If no arguments are specified
- **KeyError** – If there is no variant with the given identifier

1.8 Stability of Rock and Armour Units

The stability of the breakwater is divided into three sections, the stability of the armour layer, the toe and the substructure of the breakwater. With the functions in substructure the required nominal diameter of the first underlayer and filter layer can be computed.

1.8.1 Armour Layer

The armour layer is the outermost layer of the breakwater, the stability of this layer is thus important for the overall stability of the breakwater. The stability of this layer can be computed with either the Hudson formula or the Van der Meer formula

Hudson

breakwater.core.stability.**hudson**(*H*, *Kd*, *Delta*, *alpha*)

Hudson formula for armour layer stability (Hudson, 1959)

The Hudson formula is based on experimental research on the stability of rock and armour units (Tetrapods and Dolose) on a slope. The formula is derived by fitting a line through the data points. All unknown factors are included in the 'dustbin' factor k_D (Van den Bos and Verhagen, 2018). The Hudson formula expressed in the form with the stability number is:

$$\frac{H}{\Delta D_{n50}} = \sqrt[3]{k_D \cot \alpha}$$

Warning: The formula is based on regular waves (Hudson, 1959), therefore the wave height H to use in case of irregular waves is not defined. CERC (1984, p. 7-203) advised in the Shore Protection Manual to use $H_{1/10}$, the average of the highest 10% of the waves.

Note: Although the Hudson formula is derived for the stability of rock and armour units, Van den Bos and Verhagen (2018) recommend to use the Van der Meer formula for the stability of rock. As the Van der Meer formula includes the effects of storm duration, wave period, the structures permeability and a clearly defined damage level (CIRIA, CUR, CETMEF, 2007, p. 567).

Parameters

- **H** (*float*) – The design wave height [m]
- **Kd** (*int*) – Stability coefficient [-]
- **Delta** (*float*) – Relative buoyant density [-]
- **alpha** (*float*) – Angle of the front slope [rad]

Returns **Dn50** (*float*) – the nominal diameter [m]

Van der Meer

Three functions are implemented for the Van der Meer equations, one for deep water conditions, one for shallow water conditions and a full functions. The latter function takes into account the range in which the functions are applicable. The applicability range is determined by the ratio of the water depth over the significant wave height, see table 8.1 for these ranges.

breakwater.core.stability.**vandermeer**(*LimitState*, *Delta*, *P*, *N*, *alpha*, *slope_foreshore*, *val='max'*, *safety=1*, *logger=None*)

Van der Meer formulae for deep and shallow water conditions

Implementation of Van der Meer formulae for deep and shallow water conditions. The function first determines if the water depth and wave height are in range of the shallow or deep water formulae, and then uses the correct formulae to compute the nominal diameter of the armour layer. In case the input is in the range of the deep and shallow water formulae both formulae are used to compute the diameter, the largest diameter of the two is then returned.

Parameters

- **LimitState** (LimitState) – ULS, SLS or another limit state defined with LimitState

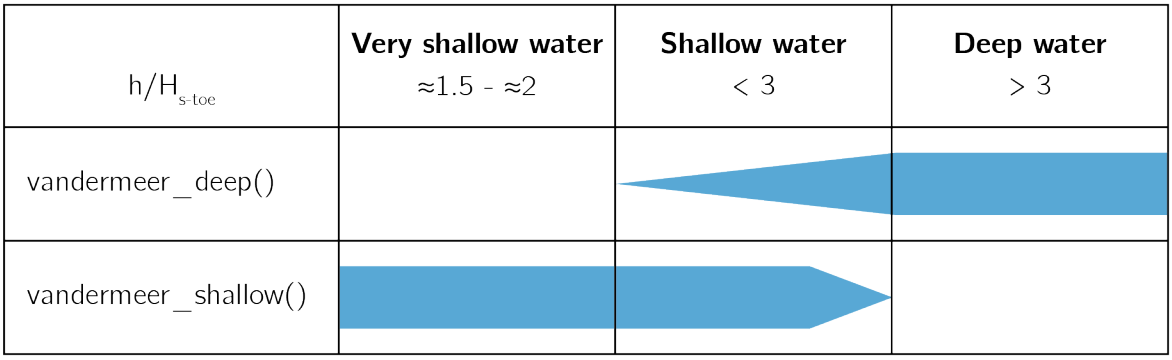


Fig. 7: Table 8.1: Range of applicability of the Van der Meer equations for deep and shallow water conditions (CIRIA, CUR, CETMEF, 2007, p. 579)

- **Delta** (*float*) – Relative buoyant density [-]
- **P** (*float*) – Notional permeability of the structure [-]
- **N** (*int*) – Number of incident waves at the toe of the structure [-]
- **alpha** (*float*) – Angle of the front slope [rad]
- **slope_foreshore** (*float*) – slope of the foreshore [rad]
- **val** (*{min, max, avg}, optional, default: max*) – value to return in case both the deep and shallow water formula are valid. min for the lowest value, max for the highest value and avg for the average value, default is max.
- **safety** (*float, optional, default: 1*) – With this parameter the model constants, Cpl and Cs, can be decreased to increase the safety. This is implemented as $C_{pl} = \mu - safety \cdot \sigma$.
- **logger** (*dict, optional, default: None*) – dict to log messages, must have keys 'INFO' and 'WARNINGS'

Returns **Dn50** (*float*) – the nominal diameter of the armourstone [m]

breakwater.core.stability.**vandermeer_deep**(*Hs*, *Delta*, *P*, *Sd*, *N*, *xi_m*, *alpha*, *safety=1*)

Van der Meer formulae for deep water conditions

For deep water conditions Van der Meer (1988) derived formulae to predict the stability of armourstone on uniform armourstone slopes with crests above the maximum run-up level. These formulae are only valid in deep water conditions, which is defined as $h > 3 \cdot H_{s-toe}$, where h is the water depth on the toe of the structure and H_{s-toe} the significant wave height at the toe. The Van der Meer formulae are given by:

$$\frac{H_s}{\Delta D_{n50}} = c_{pl} P^{0.18} \left(\frac{S}{\sqrt{N}}\right)^{0.2} \xi_m^{-0.5} \quad \text{for plunging waves}$$

$$\frac{H_s}{\Delta D_{n50}} = c_s P^{-0.13} \left(\frac{S}{\sqrt{N}}\right)^{0.2} \sqrt{\cot \alpha}\, \xi_m^{P} \quad \text{for surging waves}$$

As the formulae are based on a large amount of experiments, there is a certain reliability for the model constants. For C_{pl} this is given by 6.2 ± 0.4 and 1 ± 0.08 for C_s. Furthermore, each parameter has its own range of validity. The range of validity of parameters is presented in the table below.

Parameter	Symbol	Range
Slope angle	tan(alpha)	1:6 - 1:1.5
Number of waves	N	< 7500
Fictitious wave steepness	s_om	0.01 - 0.06
Surf similarity parameter	xi_m	0.7 - 7
Relative buoyant density of armour	Delta	1 - 2.1
Relative water depth at toe	h/H_s-toe	> 3
Notional permeability parameter	P	0.1 - 0.6
Armour gradation	Dn85/Dn15	< 2.5
Damage-storm duration ratio	Sd/sqrt(N)	< 0.9
Stability number	Hs/(Delta Dn50)	1 - 4
Damage level parameter	Sd	1 - 20

Parameters

- **Hs** (*float*) – The significant wave height, $H_{1/3}$ of the incident waves at the toe [m]
- **Delta** (*float*) – Relative buoyant density [-]
- **P** (*float*) – Notional permeability of the structure [-]
- **Sd** (*float*) – Damage level parameter [-]
- **N** (*int*) – Number of incident waves at the toe of the structure [-]
- **xi_m** (*float*) – ξ_m, the surf-similarity parameter computed with the mean wave period T_m [-]
- **alpha** (*float*) – Angle of the front slope [rad]
- **safety** (*float, optional, default: 1*) – With this parameter the model constants, Cpl and Cs, can be decreased to increase the safety. This is implemented as $C_{pl} = \mu - safety \cdot \sigma$.

Returns **Dn50** (*float*) – the nominal diameter of the armourstone [m]

breakwater.core.stability.**vandermeer_shallow**(*Hs*, *H2*, *Delta*, *P*, *Sd*, *N*, *xi_s_min_1*, *alpha*, *safety=1*)

Van der Meer formulae for shallow water conditions

Based on the analysis of the stability of rock-armoured slopes in many conditions, mainly focussed on conditions with shallow foreshores, it was proposed in Van Gent et al (2003) to modify the formulae of Van der Meer (1988) to extend its field of application. The Van der Meer formulae for shallow water are:

$$\frac{H_s}{\Delta D_{n50}} = c_{pl} P^{0.18} \left(\frac{S}{\sqrt{N}}\right)^{0.2} \left(\frac{H_s}{H_{2\%}}\right) \xi_{m-1,0}^{-0.5} \quad \text{for plunging waves}$$

$$\frac{H_s}{\Delta D_{n50}} = c_s P^{-0.13} \left(\frac{S}{\sqrt{N}}\right)^{0.2} \left(\frac{H_s}{H_{2\%}}\right) \sqrt{\cot\alpha}\, \xi_{m-1,0}^{P} \quad \text{for surging waves}$$

As the formulae are based on a large amount of experiments, there is a certain reliability for the model constants. For C_{pl} this is given by 8.4 ± 0.7 and 1.3 ± 0.15 for C_s. Furthermore, each parameter has its own range of validity. The range of validity of parameters is presented in the table below.

Parameter	Symbol	Range
Slope angle	tan(alpha)	1:4 - 1:2
Number of waves	N	< 3000
Fictitious wave steepness	s_om	0.01 - 0.06
Surf similarity parameter	xi_m	1 - 5
Surf similarity parameter	xi_m_min_1	1.3 - 6.5
Wave height ratio	H2%/Hs	1.2 - 1.4
Deep-water H over h at the toe	Hso/h	0.25 - 1.5
Armour gradation	Dn85/Dn15	1.4 - 2.0
Core material - armour ratio	Dn50-core/Dn50	0 - 0.3
Stability number	Hs/(Delta Dn50)	0.5 - 4.5
Damage level parameter	Sd	< 30

Parameters

- **Hs** (*float*) – The significant wave height, $H_{1/3}$ of the incident waves at the toe [m]
- **H2** (*float*) – $H_{2\%}$, wave height exceeded by 2% of the incident waves at the toe [m]
- **Delta** (*float*) – Relative buoyant density [-]
- **P** (*float*) – Notional permeability of the structure [-]
- **Sd** (*float*) – Damage level parameter [-]
- **N** (*int*) – Number of incident waves at the toe of the structure [-]
- **xi_m_min_1** (*float*) – $\xi_{s-1.0}$, the surf-similarity parameter computed with the energy wave period $T_{m-1.0}$ [-]
- **alpha** (*float*) – Angle of the front slope [rad]
- **safety** (*float, optional, default: 1*) – With this parameter the model constants, Cpl and Cs, can be decreased to increase the safety. This is implemented as $C_{pl} = \mu - safety \cdot \sigma$.

Returns **Dn50** (*float*) – the nominal diameter of the armourstone [m]

1.8.2 Toe

breakwater.core.toe.**toe_stability**(*Hs*, *h*, *ht*, *Delta*, *Nod*)

Toe stability formula of Van der Meer (1998)

The armour layer should is supported by a toe, the formula of Van der Meer (1998) computes the minimal required nominal diameter of the stones necessary. The formula is given as:

$$\frac{H_s}{\Delta D_{n50}} = \left(6.2\frac{h_t}{h} + 2\right) N_{od}^{0.15}$$

The formula is based on experiments and the range of validity of the parameters can be seen in the table below.

Parameter	Symbol	Range
toe depth over water depth	ht/h	0.4 - 0.9
toe depth over nominal diameter	ht/Dn50	3 - 25

Parameters

- **Hs** (*float*) – The significant wave height of the incident waves [m]

- **h** (*float*) – The water depth in front of the toe [m]
- **ht** (*float*) – The water depth on top of the toe [m]
- **Delta** (*float*) – Relative buoyant density [-]
- **Nod** (*float*) – Damage number [-]

Returns **Dn50** (*float*) – Nominal diameter of the armourstones in the toe [m]

breakwater.core.toe.**toe_berm_stability**(*Hs*, *T*, *d*, *Bm*, *Delta*, *beta=0*, *alpha_s=0.45*)

Berm protection to caisson or vertical wall breakwaters

Compute the nominal diameter of the armourstone on the berm of a caisson or vertical wall breakwaters with the modified Tanimoto formula from Takahasi (2002).

$$D_{n50} = \frac{H_s}{\Delta N_s}$$

in which:

$$N_s = \max\left\{1.8, \left(1.3\frac{1-\kappa}{\kappa^{1/3}}\frac{h'}{H_s} + 1.8\exp\left[-1.5\frac{(1-\kappa)^2}{\kappa^{1/3}}\frac{h'}{H_s}\right]\right)\right\}$$

where:

$$\kappa = \frac{4\pi h'/L'}{\sinh\left(4\pi h'/L'\right)}\kappa_2$$

and:

$$\kappa_2 = \max\left\{\alpha_S \sin^2\beta\cos^2\left(\frac{2\pi x}{L'}\cos\beta\right),\ \cos^2\beta\sin^2\left(\frac{2\pi x}{L'}\cos\beta\right)\right\}$$

Parameters

- **Hs** (*float*) – mean of the highest 1/3 of the wave heights [m]
- **T** (*float*) – wave period (s)
- **d** (*float*) – water depth at which the armour is placed [m]
- **Bm** (*float*) – width of the berm [m]
- **Delta** (*float*) – Relative buoyant density [-]
- **beta** (*float, optional, default: 0*) – angle between direction of wave approach and a line normal to the breakwater [rad]
- **alpha_s** (*float, optional, default: 0.45*) – factor the include the effect of the slope, the value of 0.45 is given by measures data (Goda, 2000)

Returns **Dn50** (*float*) – Nominal diameter of the armourstones on the berm [m]

1.8.3 Substructure

breakwater.core.substructure.**underlayer**(*Dn_armour*, *armour_layer*, *rho*, *rho_rock=2650*)

Design first underlayer of a rubble mound breakwater

Design the layer of rock immediately below the armour layer, the nominal diameter of this layer (the underlayer) is determined using one of the following rules:

- Rock (CIRIA, CUR, CETMEF, 2007, p. 630)

$M_{50u} = \frac{M_{50a}}{15}$ to $\frac{M_{50a}}{10}$

- Xbloc (Delta Marine Consultants, 2018)
  $M_{50u} = \frac{M_{50a}}{15}$ to $\frac{M_{50a}}{6}$
- XblocPlus (Delta Marine Consultants, 2018)
  $M_{50u} = \frac{M_{50a}}{20}$ to $\frac{M_{50a}}{8}$

Parameters

- **Dn_armour** (*float*) – nominal diameter of the armour layer [m]
- **armour_layer** (*{'Rock', 'Xbloc', 'XblocPlus'}*) – type of the armour layer
- **rho** (*float, optional, default: 2650*) – density of the armourstone [kg/m^3]
- **rho_rock** (*float, optional, default: 2650*) – density of the rock used in the underlayer [kg/m^3]

Returns *[float, float]* – list with the upper bound and lower bound limit of the Dn50 of the underlayer

breakwater.core.substructure.**filter_layers**(*Dn*, *rho=2650*)

Design filter layer of a rubble mound breakwater

Design a layer of rock below the first underlayer, or subsequent layers of rock. The nominal diameter of this layer (the filter layer) is given by the following rule:

- Rock (Van den Bos and Verhagen, 2018, p. 142)
  $M_{50l} = \frac{M_{50u}}{25}$ to $\frac{M_{50u}}{10}$

Parameters

- **Dn** (*float*) – nominal diameter of the layer above the filter layer [m]
- **rho** (*float, optional, default: 2650*) – density of the armourstone [kg/m^3]

Returns *[float, float]* – list with the upper bound and lower bound limit of the Dn50 of the filter layer

breakwater.core.substructure.**layer_coefficient**(*material*, *layers=None*, *placement=None*)

Get the layer thickness coefficient

Determine the layer thickness coefficient, k_t. In Table 1 and 2 the included layer thickness coefficients can be found.

Table 1: k_t values for Rock (CIRIA, CUR, CETMEF, 2007, p. 126)

Layers	Placement	Value
1	Dense	0.84
2	Standard	0.91
2	Dense	0.91

Table 2: k_t values for Armour Units (CIRIA, CUR, CETMEF, 2007, p. 260)

Armour Unit	Layers	Value
Cubes	2	1.1
Tetrapods	2	1.02
Dolos	2	0.94
Accropode	1	1.29
CoreLoc	1	1.52
Xbloc	1	1.4
XblocPlus	1	1.33

Parameters

- **material** (*str*) – material of the layer
- **layers** (*int, optional, default: None*) – number of layers, required if the material is rock. For armour units the number of layers is not required, but a warning will be shown if the specified layer is different from the table
- **placement** (*{Standard, Dense}, optional, default: None*) – placement of the material, required if the material is rock

Returns **kt** (*float*) – the layer thickness coefficient

1.9 Stability of Monolithic breakwaters

The implemented formula to determine the stability of a monolithic breakwater is the extended Goda formula. In figure 9.1 the definition of the used parameters can be seen.

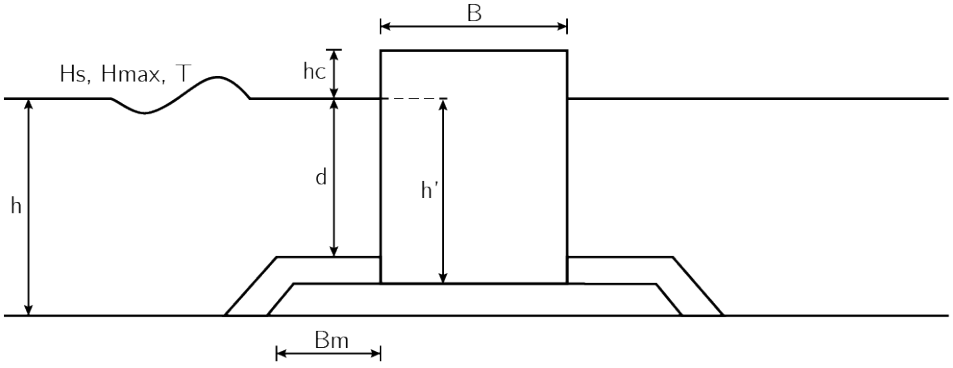


Fig. 8: Figure 9.1: Definition of the used parameters in the extended Goda formula

class breakwater.core.goda.**Goda**(*Hs, Hmax, h, d, h_acc, hc, Bm, T, beta, rho, slope_foreshore, B=None, lambda_=[1, 1, 1], logger=None*)

Compute wave pressure with the extended Goda formula (Takahasi, 2002)

Goda (1992) analysed many of the successful and unsuccessful monolithic breakwaters and developed a practical formula that can be used to analyse the stability of monolithic breakwaters. The formula developed by Goda was not meant to compute the pressures for breaking waves (impulsive conditions), therefore Takahasi (2002) included an impulsive pressure coefficient in the formula.

Warning: Goda (1992) advices to avoid impulsive pressures when designing monolithic breakwaters.

Parameters

- **Hs** (*float*) – mean of the highest 1/3 of the wave heights [m].
- **Hmax** (*float*) – design wave height, equal to the mean of the highest 1/250 of the wave heights [m].
- **h** (*float*) – water depth [m]
- **d** (*float*) – water depth in front of the caisson, on top of the foundation [m]
- **h_acc** (*float*) – submerged depth of the caisson [m]
- **hc** (*float*) – height of the caisson above the water line [m]
- **Bm** (*float*) – width of the berm [m]
- **T** (*float*) – wave period, Goda (2000) advises to use $T_{1/3}$ [s]
- **beta** (*float*) – angle between direction of wave approach and a line normal to the breakwater [rad]
- **rho** (*float*) – density of water [kg/m^3]
- **slope_foreshore** (*float*) – slope of the foreshore [rad]
- **B** (*float, optional, default: None*) – width of the monolithic breakwater [m]
- **lambda** (*list, optional, default: [1, 1, 1]*) – modification factors of Takahasi (2002) for alternative monolithic breakwater. Input must be lambda_= [$\lambda_1, \lambda_2, \lambda_3$].
- **logger** (*dict, optional, default: None*) – dict to log messages, must have keys 'INFO' and 'WARNINGS'

hb
offshore water depth at a distance of five times Hs (=H13) [m]

Type float

L
wave length computed with the dispersion relation [m]

Type float

eta_star
the elevation to which the wave pressure is exerted [m]

Type float

p1, p3, p4
representative wave pressure intensities [Pa]

Type floats

pu
uplift pressure [Pa]

Type float

h_c_star
elevation to which the wave pressure is exerted on the caisson, minimum value of hc and eta_star [m]

Type float

B
width of the monolithic breakwater [m]. None by default so it can be computed with *required_width()*

Type float

Ma()
Compute the moment around the center of the caisson

> **Warning:** This method assumes a symmetric caisson

Returns *float* – moment around the center of the caisson [Nm]

Mp()
Compute moment at the heel due to the pressure

Returns *float* – moment around the heel due to the horizontal pressures [Nm]

Mu()
Compute moment at the heel due to the uplift

Returns *float* – moment around the heel due to the uplift pressure [Nm]

P()
Compute horizontal force due to the pressure

Returns *float* – horizontal force due to the pressures [Pa]

U()
Compute force due to the uplift pressure

Returns *float* – vertical uplift pressure [Pa]

bearing_pressure(*Pc*, *rho_c*, *rho_fill*, *t=0.5*, *B=None*)
compute the bearing pressure at the heel

Method to compute the bearing pressure at the heel of the caisson.

$$p_e = \begin{cases} \frac{2W_e}{3t_e} & : t_e \leq \frac{1}{3}B \\ \frac{2W_e}{B}\left(2 - 3\frac{t_e}{B}\right) & : t_e > \frac{1}{3}B \end{cases}$$

in which:

$$t_e = \frac{M_e}{W_e}, \quad M_e = Mgt - M_U - M_p, \quad W_e = Mg - U$$

Parameters

- **Pc** (*float*) – contribution of concrete to the total mass of the caisson. value between 0 and 1
- **rho_c** (*float*) – density of concrete [kg/m^3]
- **rho_f** (*float*) – density of the fill material, for instance sand [kg/m^3]
- **t** (*scalar, optional, default: 0.5*) – horizontal distance to the centre of gravity [m]
- **B** (*float, optional, default: None*) – width of the monolithic breakwater [m], used with *bearing_pressure_width()* to compute the required width to satisfy the maximum bearing pressure.

Returns **pe** (*float*) – the bearing pressure at the heel of the caisson [Pa]

bearing_pressure_width(*B1*, *Pc*, *rho_c*, *rho_fill*, *pe_max*, *t=0.5*)
Compute the required width for the bearing pressure

Method uses fsolve from scipy to compute the width that satisfy the maximum bearing pressure of the foundation.

Parameters

- **B1** (*float*) – first estimate for the width [m]
- **Pc** (*float*) – contribution of concrete to the total mass of the caisson. value between 0 and 1
- **rho_c** (*float*) – density of concrete [kg/m^3]
- **rho_f** (*float*) – density of the fill material, for instance sand [kg/m^3]
- **pe_max** (*float*) – maximum value of the bearing pressure at the heel of the caisson. Goda (2000) advises a value between 400 and 500 kPa.
- **t** (*scalar, optional, default: 0.5*) – horizontal distance to the centre of gravity [m]

Returns *float* – required width for the maximum bearing pressure [m]

eccentricity(*M*)
Compute the eccentricity of the net vertical force

Parameters **M** (*float*) – mass of the caisson [kg]

Returns *float* – eccentricity of the net vertical force [m]

effective_width(*M*)
Compute the effective width

The effective width must be used for geotechnical computations, due to the fact that the net vertical force of the caisson is eccentric.

Parameters **M** (*float*) – mass of the caisson [kg]

Returns *float* – the effective width [m]

mass(*Pc*, *rho_c*, *rho_fill*)
Compute the mass of the caisson

Parameters

- **Pc** (*float*) – contribution of concrete to the total mass of the caisson. value between 0 and 1
- **rho_c** (*float*) – density of concrete [kg/m^3]
- **rho_f** (*float*) – density of the fill material, for instance sand [kg/m^3]

Returns **m** (*float*) – minimal required mass per meter length to satisfy the safety factors [kg/m]

plot()
Plot pressure distribution

Plots the pressure distribution together with the dimensions of the monolithic breakwater.

> **Warning:** Do not read the dimensions of the monolithic breakwater from the axes of the figure. The correct dimensions of the monolithic breakwater can be read from the figure.

required_mass(*mu*, *t=0.5*, *SF_sliding=1.2*, *SF_turning=1.2*, *logger=None*)
Compute required mass of the monolithic breakwater

Compute the minimal required mass of the monolithic breakwater based on the failure mechanisms sliding and overturning.

$$M_{sliding} = \frac{PSF_{sliding}}{g\mu} + \frac{U}{g} + \rho B h'$$

$$M_{turning} = \frac{M_p SF_{turning}}{gt} + \frac{M_u}{gt} + \rho B h'$$

Parameters

- **mu** (*float*) – friction factor between the caisson and the foundation [-]
- **t** (*scalar, optional, default: 0.5*) – horizontal distance to the centre of gravity [m]
- **SF_sliding** (*float, optional, default: 1.2*) – safety factor against sliding. Default value according to Goda (2000)
- **SF_turning** (*float, optional, default: 1.2*) – safety factor against sliding. Default value according to Goda (2000)
- **logger** (*dict, optional, default: None*) – dict to log messages, must have keys 'INFO' and 'WARNINGS'

Returns **mass** (*float*) – minimal required mass per meter length to satisfy the safety factors [kg/m]

required_width(*Pc*, *rho_c*, *rho_f*, *rho_w*, *mu*, *t=0.5*, *SF_sliding=1.2*, *SF_turning=1.2*, *logger=None*)
Compute the required width of the monolithic breakwater

Compute the minimal required width of the monolithic breakwater based on the failure mechanisms sliding and overturning.

Parameters

- **Pc** (*float*) – contribution of concrete to the total mass of the caisson. value between 0 and 1
- **rho_c** (*float*) – density of concrete [kg/m^3]
- **rho_f** (*float*) – density of the fill material, for instance sand [kg/m^3]
- **rho_w** (*float*) – density of water [kg/m^3]
- **mu** (*float*) – friction factor between the caisson and the foundation [-]
- **t** (*scalar, optional, default: 0.5*) – horizontal distance to the centre of gravity [m]
- **SF_sliding** (*float, optional, default: 1.2*) – safety factor against sliding. Default value according to Goda (2000)
- **SF_turning** (*float, optional, default: 1.2*) – safety factor against sliding. Default value according to Goda (2000)
- **logger** (*dict, optional, default: None*) – dict to log messages, must have keys 'INFO' and 'WARNINGS'

Returns **B** (*float*) – minimal required width to satisfy the safety factors [m]

1.10 Overtopping

One of the failure mechanisms of a breakwater is overtopping. Overtopping is defined as the amount of water passing over the crest of the structure per unit of time, in practice the discharge is often expressed per meter width of the breakwater as q [$l/s/m$].

Warning: The specific discharge per meter width, q, must be given in $l/s/m$, this is automatically converted to $m^3/s/m$

The EurOtop manual is the result of extensive research and experimental studies on overtopping. All formulas in this module can be found in the second edition of this manual. The following structures are defined: coastal dikes, rubble mound breakwaters, vertical walls and vertical composite walls

1.10.1 Rubble mound

Rubble mound breakwaters are characterized by a core with some porosity or permeability, covered by a sloping porous armour layer consisting of large rock or concrete armour units (for example Xbloc). However, the formula for coastal and river dikes can be used for a wider range of slopes, and therefore allows for more flexible input parameters. Since the formula for a rubble mound breakwater is a simplified case of the coastal or river dike, the formula for the dike is implemented. The definitions of the variables are presented in Figure 10.1.

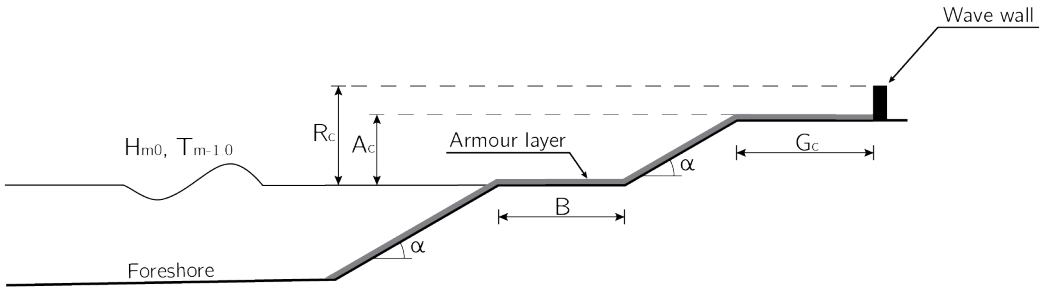


Fig. 9: Figure 10.1: schematisation of a rubble mound breakwater with definitions of variables

Formula

The general formula for the average overtopping discharge on a slope (dike, levee, embankment) implemented is the mean value approach (EurOtop, 2018). EurOtop (2018) advises against the use of a mean value approach for design or assessment purposes. Instead, EurOtop (2018) advises to increase the average discharge by one standard deviation for a design or assessment. Therefore, 1 is the default setting for the `safety` parameter.

`breakwater.core.overtopping.`**`rubble_mound`**(*Hm0*, *q*, *xi_m_min_1*, *alpha*, *beta*, *gamma_b*, *gamma_v*, *gam_star*, *armour_layer*, *layers=1*, *permeability='permeable'*, *safety=1*, *Gc=None*, *Dn50=None*, *limit=True*)

Compute the crest freeboard of a rubble mound breakwater

Computes the crest freeboard of a rubble mound breakwater using equation 5.10 and 5.12 from EurOtop (2018).

$$\frac{q}{\sqrt{g \cdot H_{m0}^3}} = \frac{0.023}{\sqrt{\tan\alpha}} \gamma_b \cdot \xi_{m-1,0} \cdot \exp\left[-\left(2.7 \frac{R_c}{\xi_{m-1,0} \cdot H_{m0} \cdot \gamma_b \cdot \gamma_f \cdot \gamma_\beta \cdot \gamma_v}\right)^{1.3}\right]$$

with a maximum of

$$\frac{q}{\sqrt{g \cdot H_{m0}^3}} = 0.09 \cdot \exp\left[-\left(1.5 \frac{R_c}{H_{m0} \cdot \gamma_f \cdot \gamma_\beta \cdot \gamma^*}\right)^{1.3}\right]$$

The reliability of the first equation is given by σ (0.023) = 0.003 and σ (2.7) = 0.20, and for the second equation by σ (0.09) = 0.0135 and σ (1.5) = 0.15.

Parameters

- **Hm0** (*float*) – the spectral wave height [m]
- **q** (*float*) – mean overtopping discharge per meter structure width [l/s per m]
- **xi_m_min_1** (*float*) – $\xi_{m-1.0}$, the surf-similarity parameter computed with the energy wave period $T_{m-1.0}$ [-]
- **alpha** (*float*) – Angle of the front slope [rad]
- **beta** (*float*) – the angle of wave attack [rad]
- **gamma_b** (*float*) – influence factor for a berm [-]
- **gamma_v** (*float*) – influence factor for a vertical wall [-]
- **gamma_star** (*float*) – overall influence factor for a storm wall on slope or promenade [-]
- **armour_layer** (*str*) – name of the material of the armour layer, supported materials: Smooth, Rock, Cubes, Antifers, HARO, Tetrapods, Dolose, Accropode I, Xbloc, CoreLoc, Accropode II, Cubipods
- **layers** (*{1, 2}, default: 1*) – number of layers in the armour layer, required if the armour layer is made out of: Rock, Cubes or Cubipods
- **permeability** (*{'permeable', 'impermeable'}, default: 'permeable'*) – permeability of the armour layer, required if the armour layer is made out of Rock
- **safety** (*float, optional, default: 1*) – safety factor of the design, positive values increase the safety of the design by increasing the mean value of the model constants with the number of standard deviations specified. In accordance with the recommendation from EurOtop (2018), the default value is set to 1 standard deviation.
- **Gc** (*float, optional, default: None*) – Effect of armoured crest width [m]
- **Dn50** (*float, optional, default: None*) – nominal diameter of the armour units on the crest [m]
- **limit** (*bool, optional, default: True*) – If True, the discharge will be set to the limit for zero discharge in case the given discharge is below this limit. If False, the discharge will not be changed.

Returns **Rc** (*float*) – the crest freeboard of the structure [m]

Influence factors

The influence of roughness elements, oblique wave attack, berms, etc. are taken into account by introducing influence factors

breakwater.core.overtopping.**gamma_f**(*armour_layer*, *xi_m_min_1*, *layers=None*, *permeability=None*, *placement=None*)

Influence factor for the permeability and roughness of the slope

Computes the influence factor on roughness with table 6.2 from EurOtop (2018). These values are derived for $2.8 \leq \xi_{m-1.0} \leq 4.5$, in case of larger surf-similarity parameters the influence factor for roughness is increased using equation 6.7 from EurOtop (2018)

Type of armour layer	γ_f
Smooth impermeable surface	1.00
Rock (1 layer, impermeable core)	0.60
Rock (1 layer, permeable core)	0.45
Rock (2 layers, impermeable core)	0.55
Rock (2 layers, permeable core)	0.40
Cubes (1 layer, flat positioning)	0.49
Cubes (2 layers, random positioning)	0.47
Antifers	0.50
HARO	0.47
Tetrapods	0.38
Dolos	0.43
Accropode I	0.46
Xbloc, CoreLoc, Accropode II	0.44
XblocPlus	0.45
Cubipods (1 layer)	0.49
Cubipods (2 layers)	0.47

Parameters

- **armour_layer** (*str*) – name of the material of the armour layer, supported materials: Smooth, Rock, Cubes, Antifers, HARO, Tetrapods, Dolos, Accropode I, Xbloc, XblocPlus, CoreLoc, Accropode II, Cubipods
- **xi_m_min_1** (*float*) – the surf-similarity parameter [-]
- **layers** (*{1, 2}, optional, default: None*) – number of layers in the armour layer, required if the armour layer is made out of: Rock, Cubes or Cubipods
- **permeability** (*{'permeable', 'impermeable'}, optional, default: None*) – permeability of the armour layer, required if the armour layer is made out of Rock
- **placement** (*{'flat', 'random'}, optional, default: None*) – placement of the armour layer, required if the armour layer is made out of Cubes

Returns **gamma_f** (*float*) – the influence factor for roughness

Raises **Keyerror** – If the armour layer is not in table 6.2 from EurOtop (2018)

breakwater.core.overtopping.**gamma_beta**(*beta*)

Influence factor for oblique wave attack

Computes the influence factor for oblique wave attack with equation 5.29 from EurOtop (2018).

$$\gamma_\beta = 1 - 0.0033 \mid \beta \mid$$

with a maximum of $\gamma_\beta = 0.736$ for $\mid \beta \mid > 80$

Parameters **beta** (*float*) – the angle of wave attack [rad]

Returns **gamma_beta** (*float*) – the influence factor for oblique wave attack

Note: The formulas for $\gamma_b, \gamma_v, \gamma_*$ have not yet been implemented. These must thus be computed by hand.

1.10.2 Vertical and Composite Vertical

The vertical and composite vertical walls are comparable structure, the difference between is the depth in front of the vertical wall. A composite vertical wall is fronted by a berm or toe mound below the water level, whereas a vertical wall is not fronted by such a berm, in that case d = h. In figure 10.2 a definition sketch of both structures is presented.

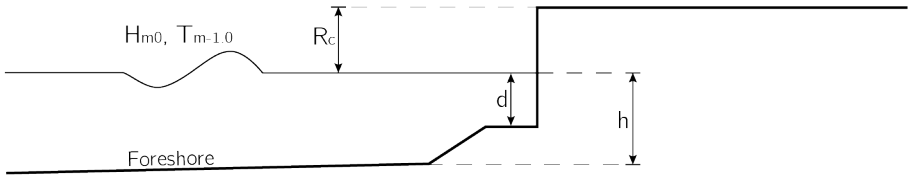


Fig. 10: Figure 10.2: schematisation of a vertical or composite vertical wall with definitions of variables

More than one equation have been derived to compute the overtopping discharge, or crest freeboard. Therefore, a general formula is implemented which automatically classifies the structure in order so that the correct formula is used.

General Formula

The general formula is an implementation of the decision chart from EurOtop (2018). In figure 10.3 the implemented decision chart is depicted with the references to the individual formulas.

`breakwater.core.overtopping.`**`vertical`**(*Hm0*, *q*, *h*, *d*, *L_m_min_1*, *s_m_min_1*, *safety=1*, *logger=None*, *limit=True*)

Compute crest freeboard for vertical and composite vertical walls

Compute the crest freeboard, Rc, of a vertical or composite vertical wall for a given mean overtop discharge. The function is an implementation of the decision chart, figure 7.2, from EurOtop (2018). The function determines if the input classifies as a vertical or composite vertical wall, if breaking is possible and if the structure has a low freeboard. Based on the classification the function calls the corresponding function and computes the freeboard.

Parameters

- **`Hm0`** (*float*) – the spectral wave height [m]
- **`q`** (*float*) – mean overtopping discharge per meter structure width [l/s per m]
- **`h`** (*float*) – water depth in front of the toe of the structure [m]
- **`d`** (*float*) – water depth above the toe of the structure [m]
- **`L_m_min_1`** (*float*) – $L_{m-1.0}$, spectral wave length in deep water [m]
- **`s_m_min_1`** (*float*) – $s_{m-1.0}$, wave steepness with the spectral wave length ($L_{m-1.0}$) [-]

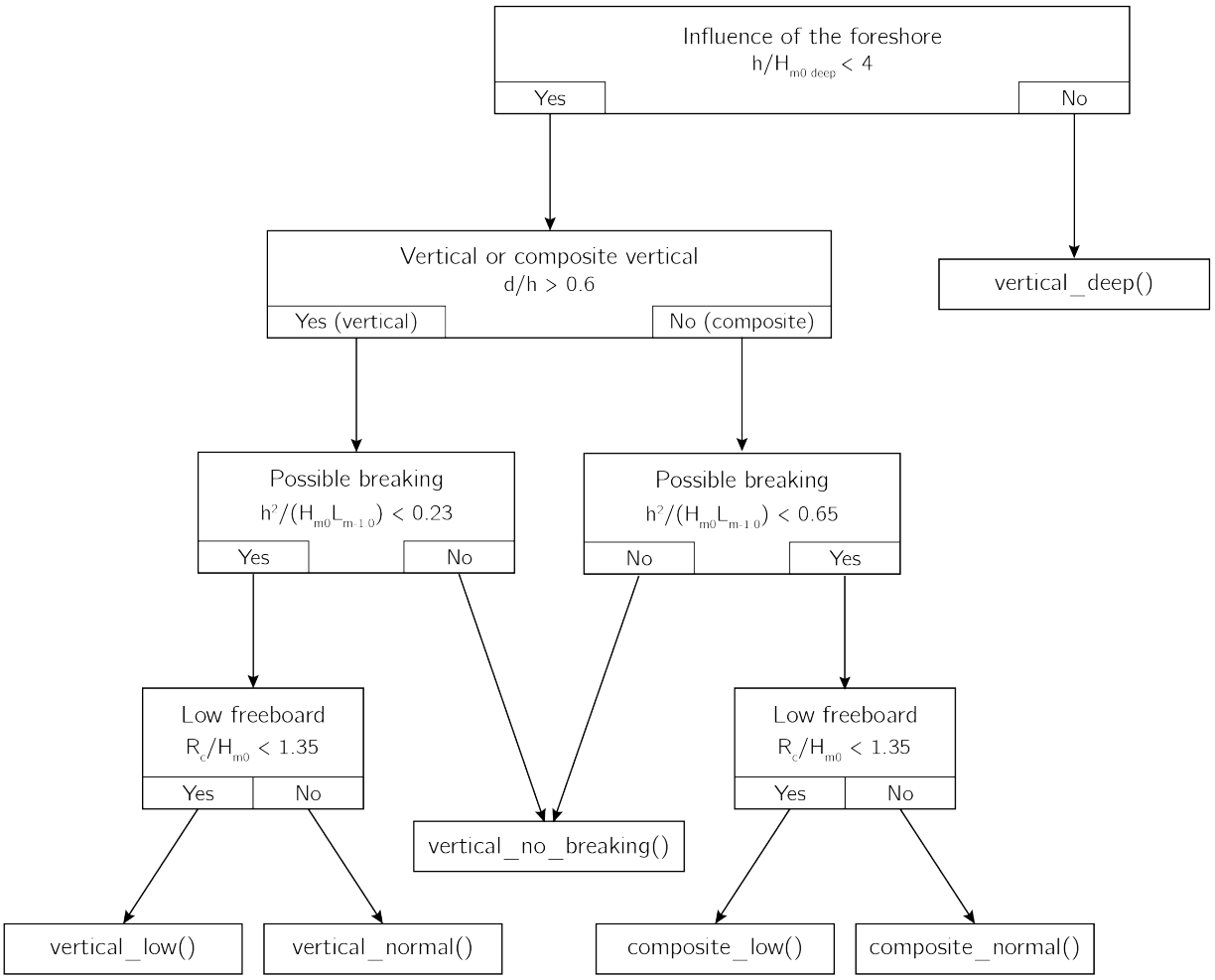


Fig. 11: Figure 10.3: decision chart for prediction of overtopping discharge for a vertical or composite vertical wall

- **safety** (*float, optional, default: 1*) – safety factor of the design, positive values increase the safety of the design by increasing the mean value of the model constants with the number of standard deviations specified. In accordance with the recommendation from EurOtop (2018), the default value is set to 1 standard deviation.
- **logger** (*dict, optional, default: None*) – dict to log messages, must have keys 'INFO' and 'WARNINGS'
- **limit** (*bool, optional, default: True*) – If True, the discharge will be set to the limit for zero discharge in case the given discharge is below this limit. If False, the discharge will not be changed.

Returns **Rc** (*float*) – the crest freeboard of the structure [m]

Formulas

breakwater.core.overtopping.**vertical_deep**(*Hm0*, *q*, *safety=1*, *limit=True*)

Rc if the foreshore does not have an influence

Compute the crest freeboard for a vertical or composite vertical wall if the foreshore does not have an influence. Implementation of equation 7.1 from EurOtop (2018).

$$\frac{q}{\sqrt{g \cdot H_{m0}^3}} = 0.047 \cdot \exp\left[-\left(2.35\frac{R_c}{H_{m0}}\right)^{1.3}\right]$$

The reliability of the equation is given by σ (0.047) = 0.007 and σ (2.35) = 0.23.

Parameters

- **Hm0** (*float*) – the spectral wave height [m]
- **q** (*float*) – mean overtopping discharge per meter structure width [l/s per m]
- **safety** (*float, optional, default: 1*) – safety factor of the design, positive values increase the safety of the design by increasing the mean value of the model constants with the number of standard deviations specified. In accordance with the recommendation from EurOtop (2018), the default value is set to 1 standard deviation.
- **limit** (*bool, optional, default: True*) – If True, the discharge will be set to the limit for zero discharge in case the given discharge is below this limit. If False, the discharge will not be changed.

Returns **Rc** (*float*) – the crest freeboard of the structure [m]

breakwater.core.overtopping.**vertical_no_breaking**(*Hm0*, *q*, *safety=1*, *limit=True*)

Rc if no possibility for breaking waves

Compute the crest freeboard for a vertical or composite vertical wall if there are no breaking waves. Implementation of equation 7.5 from EurOtop (2018)

$$\frac{q}{\sqrt{gH_{m0}^3}} = 0.05\exp\left(-2.78\frac{R_c}{H_{m0}}\right)$$

The reliability of the equation is given by σ (0.05) = 0.012 and σ (2.78) = 0.17.

Parameters

- **Hm0** (*float*) – the spectral wave height [m]
- **q** (*float*) – mean overtopping discharge per meter structure width [l/s per m]

- **safety** (*float, optional, default: 1*) – safety factor of the design, positive values increase the safety of the design by increasing the mean value of the model constants with the number of standard deviations specified. In accordance with the recommendation from EurOtop (2018), the default value is set to 1 standard deviation.
- **limit** (*bool, optional, default: True*) – If True, the discharge will be set to the limit for zero discharge in case the given discharge is below this limit. If False, the discharge will not be changed.

Returns **Rc** (*float*) – the crest freeboard of the structure [m]

breakwater.core.overtopping.**vertical_normal**(*Hm0*, *q*, *h*, *s_m_min_1*, *safety=1*, *limit=True*)

Rc for a vertical wall if normal freeboard and breaking waves

Compute the crest freeboard for a vertical wall if there is a possibility for breaking waves, but the freeboard is not low. Implementation of equation 7.8 from EurOtop (2018)

$$\frac{q}{\sqrt{gH_{m0}^3}} = 0.0014 \left(\frac{H_{m0}}{hs_{m-1,0}}\right)^{0.5} \left(\frac{R_c}{H_{m0}}\right)^{-3}$$

The reliability of the equation is given by σ (0.0014) = 0.0006

Parameters

- **Hm0** (*float*) – the spectral wave height [m]
- **q** (*float*) – mean overtopping discharge per meter structure width [l/s per m]
- **h** (*float*) – water depth in front of the toe of the structure [m]
- **s_m_min_1** (*float*) – $s_{m-1.0}$, wave steepness with the spectral wave length ($L_{m-1.0}$) [-]
- **safety** (*float, optional, default: 1*) – safety factor of the design, positive values increase the safety of the design by increasing the mean value of the model constants with the number of standard deviations specified. In accordance with the recommendation from EurOtop (2018), the default value is set to 1 standard deviation.
- **limit** (*bool, optional, default: True*) – If True, the discharge will be set to the limit for zero discharge in case the given discharge is below this limit. If False, the discharge will not be changed.

Returns **Rc** (*float*) – the crest freeboard of the structure [m]

breakwater.core.overtopping.**vertical_low**(*Hm0*, *q*, *h*, *s_m_min_1*, *safety=1*, *limit=True*)

Rc for a vertical wall if low freeboard and breaking waves

Compute the crest freeboard for a vertical wall if there is a possibility for breaking waves, and the freeboard is low. Implementation of equation 7.7 from EurOtop (2018)

$$\frac{q}{\sqrt{gH_{m0}^3}} = 0.011 \left(\frac{H_{m0}}{hs_{m-1,0}}\right)^{0.5} \exp\left(-2.2\frac{R_c}{H_{m0}}\right)$$

The reliability of the equation is given by σ (0.011) = 0.0045

Parameters

- **Hm0** (*float*) – the spectral wave height [m]
- **q** (*float*) – mean overtopping discharge per meter structure width [l/s per m]
- **h** (*float*) – water depth in front of the toe of the structure [m]

- **s_m_min_1** (*float*) – $s_{m-1.0}$, wave steepness with the spectral wave length ($L_{m-1.0}$) [-]
- **safety** (*float, optional, default: 1*) – safety factor of the design, positive values increase the safety of the design by increasing the mean value of the model constants with the number of standard deviations specified. In accordance with the recommendation from EurOtop (2018), the default value is set to 1 standard deviation.
- **limit** (*bool, optional, default: True*) – If True, the discharge will be set to the limit for zero discharge in case the given discharge is below this limit. If False, the discharge will not be changed.

Returns **Rc** (*float*) – the crest freeboard of the structure [m]

breakwater.core.overtopping.**composite_normal**(*Hm0*, *q*, *h*, *d*, *s_m_min_1*, *safety=1*, *limit=True*)

Rc for a composite wall if normal freeboard and breaking waves

Compute the crest freeboard for a composite vertical wall if there is a possibility for breaking waves, but the freeboard is not low. Implementation of equation 7.14 from EurOtop (2018)

$$\frac{q}{\sqrt{gH_{m0}^3}} = 1.3\left(\frac{d}{h}\right)^{0.5} 0.0014\left(\frac{H_{m0}}{hs_{m-1,0}}\right)^{0.5}\left(\frac{R_c}{H_{m0}}\right)^{-3}$$

The reliability of the equation is given by σ (0.0014) = 0.0006.

Parameters

- **Hm0** (*float*) – the spectral wave height [m]
- **q** (*float*) – mean overtopping discharge per meter structure width [l/s per m]
- **h** (*float*) – water depth in front of the toe of the structure [m]
- **d** (*float*) – water depth above the toe of the structure [m]
- **s_m_min_1** (*float*) – $s_{m-1.0}$, wave steepness with the spectral wave length ($L_{m-1.0}$) [-]
- **safety** (*float, optional, default: 1*) – safety factor of the design, positive values increase the safety of the design by increasing the mean value of the model constants with the number of standard deviations specified. In accordance with the recommendation from EurOtop (2018), the default value is set to 1 standard deviation.
- **limit** (*bool, optional, default: True*) – If True, the discharge will be set to the limit for zero discharge in case the given discharge is below this limit. If False, the discharge will not be changed.

Returns **Rc** (*float*) – the crest freeboard of the structure [m]

breakwater.core.overtopping.**composite_low**(*Hm0*, *q*, *h*, *d*, *s_m_min_1*, *safety=1*, *limit=True*)

Rc for a composite wall if low freeboard and breaking waves

Compute the crest freeboard for a composite vertical wall if there is a possibility for breaking waves, and the freeboard is low. Implementation of equation 7.15 from EurOtop (2018)

$$\frac{q}{\sqrt{gH_{m0}^3}} = 1.3\left(\frac{d}{h}\right)^{0.5} 0.011\left(\frac{H_{m0}}{hs_{m-1,0}}\right)^{0.5}\exp\left(-2.2\frac{R_c}{H_{m0}}\right)$$

The reliability of the equation is given by σ (0.011) = 0.0045.

Parameters

- **Hm0** (*float*) – the spectral wave height [m]

- **q** (*float*) – mean overtopping discharge per meter structure width [l/s per m]
- **h** (*float*) – water depth in front of the toe of the structure [m]
- **d** (*float*) – water depth above the toe of the structure [m]
- **s_m_min_1** (*float*) – $s_{m-1.0}$, wave steepness with the spectral wave length ($L_{m-1.0}$) [-]
- **safety** (*float, optional, default: 1*) – safety factor of the design, positive values increase the safety of the design by increasing the mean value of the model constants with the number of standard deviations specified. In accordance with the recommendation from EurOtop (2018), the default value is set to 1 standard deviation.
- **limit** (*bool, optional, default: True*) – If True, the discharge will be set to the limit for zero discharge in case the given discharge is below this limit. If False, the discharge will not be changed.

Returns **Rc** (*float*) – the crest freeboard of the structure [m]

1.11 Geotechnical stability

This chapter includes the implemented geotechnical equations. These computations are automatically executed by the design classes when a `Soil` is specified. For the `Caisson` class the `brinch_hansen()` method is used to compute the bearing capacity of the soil. In `RockRubbleMound` and `ConcreteRubbleMound` a slip circle analysis is performed with `Bishop`. The required length of the scour protection is computed for all design classes.

1.11.1 Define Soil

class breakwater.core.soil.**Soil**(*c*, *phi*, *gamma=None*, *rho=None*, *n=None*)

Define soil

Define the subsoil on which the breakwater is constructed. This class is used in the design classes for making the appropriate geotechnical computations.

Note: It is currently not possible to use several soil layers in the design classes. The soil must therefore be a homogeneous soil.

Parameters

- **c** (*float*) – cohesion of the soil [kPa]
- **phi** (*float*) – internal friction angle [degrees]
- **gamma** (*float, optional, default: None*) – volumetric weight of the soil [kN/m^3]
- **rho** (*float, optional, default: None*) – the density of the soil [kg/m^3]
- **n** (*float, optional, default: None*) – porosity of the soil

c

cohesion of the soil [kPa]

Type float

phi
internal friction angle [rad]

Type float

gamma
volumetric weight of the soil [kN/m^3]

Type float

gamma_sat
saturated volumetric weight of the soil [kN/m^3]

Type float

n
porosity of the soil

Type float, optional, default: None

brinch_hansen(*p*, *t*, *B*, *L*, *q*, *rho_w=None*, *sat=True*)
Brinch-Hansen

implementation of the Brinch Hansen equation to determine the bearing capacity of the soil per unit length. The equation is given by (Brinch Hansen, 1970):

$$p = i_c s_c c N_c + i_q s_q q N_q + i_\gamma s_\gamma \frac{1}{2} \gamma B N_\gamma$$

In which i and s are the inclination and shape factor, and N are dimensionless constants. Note that compared to the original equation the depth, base and ground inclination factors have been omitted. The latter two because the assumption is made that the foundation is never constructed at an angle, and the depth factor because the foundation is not placed in the soil but on top of the bottom.

Parameters

- **t** (*float*) – horizontal stress [kPa]
- **p** (*float*) – vertical stress [kPa]
- **q** (*float*) – overburden [kPa]
- **B** (*float*) – width of the structure [m]
- **L** (*float*) – length of the structure [m], set to None if the structure is a long structure and the shape factors can be neglected.
- **rho_w** (*float, optional, default: None*) – density of water, by default the soil is not submerged [kg/m^3]
- **sat** (*bool, optional, default: True*) – True if the saturated volumetric weight of the soil must be used, False if the dry volumetric weight of the soil must be used.

Returns **p** (*float*) – bearing capacity of the foundation [kPa]

saturated_weight(*gamma_sat=None*, *rho_w=None*)
Method to add the saturated weight

Method to add the properties of the soil if it is saturated. The saturated soil properties can be added by setting gamma_sat or by specifing rho_w. In case of the latter the saturated weight is computed with the porosity.

Parameters

- **gamma_sat** (*float, optional, default: None*) – saturated volumetric weight of the soil [kN/m^3]
- **rho_w** (*float, optional, default: None*) – density of water [kg/m^3]

Raises **InputError** – if the porosity is not specified and the saturated volumetric weight is computed with rho_w

1.11.2 Slip Circle

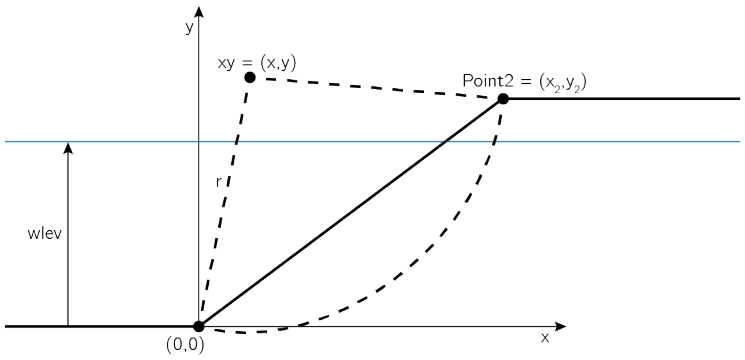
class breakwater.core.bishop.**Bishop**(*point2*, *y_step=1*, *wlev=None*, *SlipCircle=None*)

Bishop slip circles

Class for the computation of the factor of safety against slip failure. This factor of safety is computed with the following equation (Verruijt, 2012):

$$F = \frac{\sum \frac{c+(\gamma h - p)\tan\phi}{\cos\alpha(1+\tan\alpha\tan\phi/F)}}{\sum \gamma h \sin\alpha}$$

The top of the soil is defined by one point, point2, see the Figure. All slip circles must go through (0,0) and point2, this means that if a custom slip circle is defined this circle must go through both points. If a custom slip circle is not given, slip circles are automatically generated. These circles are generated with an interval of `y_step`.



Parameters

- **point2** (*tuple*) – x and y coordinate of point2
- **y_step** (*float, optional, default: 1*) – step size of the y coordinate for generating circles, used if a custom circle is not defined
- **wlev** (*float, optional, defautl: None*) – y coordinate of the water level, by default the water level is None, which means that there is no water
- **SlipCircle** (*SlipCircle*) – user defined custom slip circle, circle must be defined with *SlipCircle*

x2, y2
x and y coordinate of point2

Type float

wlev
y coordinate of the water level

Type float

circles
dictionary with the centre and radius of the circle. Slices are added to the dict when generated during the computation.

Type dict

layers
dictionary with the defined layers

Type dict

normative
id of the normative *SlipCircle*

Type int

add_layer(*gamma*, *c*, *phi*, *name*, *ymin=None*, *ymax=None*, *gamma_sat=None*)
Add layer to the soil

Add a soil layer for the computation. In case of a homogeneous soil the arguments `ymin` and `ymax` do not have to be specified. However, if the soil consists of several layers these parameters must be given for each layer.

Warning: soil layers must be added sequentially from the lowest layer to the highest layer.

Parameters

- **gamma** (*float*) – volumetric weight of the material [kN/m^3]
- **c** (*float*) – cohesion of the material [kPa]
- **phi** (*float*) – internal friction angle [deg]
- **name** (*str*) – name of the layer
- **ymin** (*float, optional, default: None*) – y coordinate of the start of the layer, required if more than 1 layer is added
- **ymax** (*float, optional, default: None*) – y coordinate of the end of the layer, required if more than 1 layer is added
- **gamma_sat** (*float, optional, default: None*) – saturated volumetric weight of the material [kN/m^3]. Only required if a *wlev* is specified.

Raises **InputError** – if more than 1 layer is specified and ymin and/or ymax has not been specified or gamma_sat is not given when a wlev has been specified

compute(*num_slices*, *max_iter=50*, *ftol=0.05*, *gamma_w=None*)
Compute the factor of safety

Method to compute the factor of safety for all generated circles, or the specified circle. Updates *normative* with the id of the normative slip circle, i.e. the slip circle with the lowest factor of safety.

Parameters

- **num_slices** (*int*) – number of slices
- **max_iter** (*int, optional, default: 50*) – maximum number of iterations
- **ftol** (*float, optional, default: 0.05*) – break criterium, when the change between the previous and the current factor of safety is below the change the iteration is ended

- **gamma_w** (*float, optional, default: None*) – volumetric weight of water [kN/m^3]

plot(*id=None*, *show_slices=False*)
Plot slip circle(s)

Method to plot the slip circle(s)

Parameters

- **id** (*int, optional, default: None*) – id of the slip circle to plot, by default all slip circles are plotted
- **show_slices** (*bool, optional, default: False*) – if the slices must be shown, only available after the computation as the slices are not generated before the computation. The default value is False, meaning that the slices will not be plotted.

class breakwater.core.bishop.**SlipCircle**(*centre*, *r*)
Define a slip circle

Define a custom slip circle to use in *Bishop*

Parameters

- **centre** (*tuple*) – x and y coordinate of the centre of the circle
- **r** (*float*) – radius of the circle

xy
x and y coordinate of the centre of the circle

Type tuple

r
radius of the circle

Type float

F
computed factor of safety

Type float

slices
dictionary with the coordinates, height and slip angle of the slices

Type dict

make_slices(*num_slices*, *point2*, *layers*)
Make slices

Method to divide the given circle into slices

Parameters

- **num_slices** (*int*) – number of slices
- **circle** (*dict*) – dictionary with the centre and radius of the circle
- **layers** (*dict*) – dictionary with the layers

plot(*show_slices=False*)
Plot the circle

Parameters **show_slices** (*bool, optional, default: False*) – if the slices must be shown, only available after the computation as the slices are not generated before the computation. The default value is False, meaning that the slices will not be plotted.

1.11.3 Scour

breakwater.core.scour.**scour_protection**(*L*, *slope=None*)

Compute the length of the scour protection

Compute the estimated length of the scour hole with Sumer and Fredsoe (2000). In the table the estimated length of the scour hole from the experimental study is presented.

slope of the breakwater	estimated length
Vertical breakwater	1.0(L/4)
1:1.2	0.6(L/4)
1:1.75	0.4(L/4)

Parameters

- **L** (*float*) – wave length at the toe of the structure, computed with the mean wave period [m]
- **slope** (*float, optional, default: None*) – Slope of the armour layer (V, H), for example a slope of 3V:4H is defined as (3, 4). By default the length of the scour protection for a vertical breakwater is computed.

Returns *float* – the required length of the scour protection

1.12 Reading Excel files

Besides specifying the input in Python it is also possible to load a parametric design or material from an Excel file. The Excel input file can be made by the user in Excel, but it advised to use *bw.generate_excel* to generate the required Excel input file.

1.12.1 Creating Excel input file

breakwater.utils.input_generator.**generate_excel**(*filepath*, *input='configurations'*, *structure=None*)

Generate excel file for design input

Parameters

- **filepath** (*str*) – location to save the excel input file
- **input** (*str, optional, default: configurations*) – specify which type of input excel must be generated, possible arguments: configurations (default), parameters, LimitState, Grading, ArmourUnits
- **structure** (*{RRM, CRM, RC, CC}, optional, default: None*) – structure for which the input sheet must be generated. RRM for a rubble mound with rock as armour layer, CRM for a rubble mound with concrete armour units as armour layer, RC for a vertical (composite) breakwater with rock as armour layer for the foundation and CC for a vertical (composite) breakwater with concrete armour units as armour layer for the foundation.

Raises **TypeError** – if the extension of the filepath is not .xlsx

1.12.2 Parametric Design

breakwater.design.**read_configurations**(*filepath*, *structure*, *kd=None*, *name=None*, *LS=None*, *Grading=None*, *ArmourUnits=None*, *BermMaterial=None*)

Conceptual design for multiple breakwaters from an Excel file

Make a conceptual design for multiple (types) of breakwaters from an Excel input file. The Excel input file can be generated with *bw.generate_excel*

Parameters

- **structure** (*{'RRM', 'CRM', 'RC', 'CC'}*) – structure for which conceptual designs must be generated. RRM for a rubble mound with rock as armour layer, CRM for a rubble mound with concrete armour units as armour layer, RC for a vertical (composite) breakwater with rock as armour layer for the foundation and CC for a vertical (composite) breakwater with concrete armour units as armour layer for the foundation.
- **kd** (*int, optional, default: None*) – stability coefficient [-]
- **name** (*str, optional, default: None*) – name of the ArmourUnit
- **LS** (py:class:*LimitState*, optional, default: None) – ULS, SLS or another limit state defined with `LimitState`, by default the LimitState is read from the Excel input file
- **Grading** (`RockGrading`, optional, default: None) – standard rock grading defined in the NEN-EN 13383-1 or a user defined rock grading. By default the Grading is read from the Excel input file
- **ArmourUnit** (*obj, optional, default: None*) – armour unit class which inherits from `ConcreteArmour`, for instance `Xbloc` or `XblocPlus`. By default the ArmourUnit is read from the Excel input file. This argument is used for RRM.
- **BermMaterial** (*obj, optional, default: None*) – armour unit class which inherits from `ConcreteArmour`, for instance `Xbloc` or `XblocPlus`. By default the BermMaterial is read from the Excel input file. This argument is used for CC.

Returns *bw.Configurations* – a Configurations object with breakwater concepts

1.12.3 Material

breakwater.material.**read_grading**(*filepath*, *rho=2650*, *sheet_name=0*)

Load a rock grading from an excel or csv file

Excel input file can be generated by using *bw.generate_excel* with input argument 'Grading'. Alternatively, when using your own Excel file it must, at least, have the following headers.

Rock Class	M50 Lower	M50 Upper	NLL	NUL

Parameters

- **filepath** (*str*) – a valid filepath
- **rho** (*float, optional, default: 2650*) – density of the armourstone [kg/m^3]
- **sheet_name** (*str, optional, default: 0*) – name of the sheet with the RockGrading to read, default value of 0 will result in reading the first sheet

Returns *bw.RockGrading* – a rock grading object

breakwater.material.**read_units**(*filepath*, *kd*, *name*, *rho=2400*, *sheet_name=0*)
Load ArmourUnits from an excel or csv file

Excel input file can be generated by using *bw.generate_excel* with input argument 'ArmourUnits'. Alternatively, when using your own Excel file it must, at least, have the following headers.

Volume	D	h	Vc

Note: Do not use this function to read Xbloc or XblocPlus armour units from an Excel file. Use the predefined, *bw.Xbloc* or *bw.XblocPlus* instead.

Parameters

- **filepath** (*str*) – a valid filepath
- **kd** (*int*) – Stability coefficient [-]
- **name** (*str*) – name of the ArmourUnit
- **rho** (*float, optional, default: 2400*) – density of the concrete [kg/m^3]
- **sheet_name** (*str*) – name of the sheet with the armour units to read, default value of 0 will result in reading the first sheet

Returns *bw.ConcreteArmour* – an armour units object

1.13 Breakwater Database

class breakwater.database.database.**BreakwaterDatabase**(*update=False*)
Breakwater Database

Import the breakwater database developed by (Allsop et al., 2009) consist of completed breakwater projects with data ranging from design wave height to contractor. The constructed breakwaters are classified by breakwater type, the following types are currently included: Rubble Mound, Composite, Berm, Caisson and Revetments.

The breakwater database is a separate module of `breakwater` and can be imported with the following command:

```
from breakwater.database import BreakwaterDatabase
```

Note: To use this module the `Basemap` package is required, this dependency is additional to the dependencies mentioned in Section 2.1. See the following link for an installation of `Basemap`

Parameters **update** (*bool, optional, default: False*) – if False the data is not updated and the included data is loaded, if True the database is loaded from the *source*

df
DataFrame of the breakwater database

Type pd.DataFrame

source
url of the source

Type str

correlation(*param1*, *param2*, *bw_type=None*, *method='pearson'*)
Compute the correlation between two parameters

Parameters

- **param1** (*str*) – name of parameter 1
- **param2** (*str*) – name of parameter 2
- **bw_type** (*str, optional, default: None*) – if specified only the values of the given bw_type are considered
- **method** (*{pearson, spearman}, optional, default: pearson*) – method of correlation

Returns *tuple* – correlation between the two parameters and the p-value

cross_section(*id*, *B=None*, *Rc=None*, *h=None*, *slope=None*)
Plot a cross section of the breakwater

Method to plot a cross section of a breakwater in the database. The breakwater is selected by the id of the breakwater. In case data is missing to plot the breakwater it is possible to specify these as arguments.

Warning: plot function currently only supports Rubble Mound and caisson breakwaters

Parameters

- **id** (*int*) – id of the breakwater to plot
- **B** (*float, optional, default: None*) – specify custom crest width
- **Rc** (*float, optional, default: None*) – specify custom crest height
- **h** (*float, optional, default: None*) – specify a custom water level
- **slope** (*tuple, optional, default: None*) – specify a custom slope, must be specified as a tuple (V, H)

Raises **ValueError** – If data required to plot a cross section is missing

hist(*param*, *show_unclassified=False*, *exclude=None*, *min_data=5*, *xmax=None*, *bins=10*)
Plot a histogram of a parameter

Parameters

- **param** (*str*) – name of the parameter
- **show_unclassified** (*bool, optional, default: False*) – True is unclassified breakwaters with a coordinate must be plotted, False is unclassified breakwaters must not be plotted.
- **exclude** (*list, optional, default: None*) – list of breakwater types to exclude from the plot
- **min_data** (*int, optional, default: 5*) – minimum number of datapoints required for plotting, if the data for a bw type is less than this limit it will be skipped
- **xmax** (*float, optional, default: None*) – maximum x coordinate of the histogram, by default this limit is automatically determined
- **bins** (*int, optional, default: 10*) – number of bins

map(*area=[]*, *resolution='c'*, *show_unclassified=False*, *exclude=None*)
Plot the breakwaters on a world map

Method to plot all breakwaters with coordinates on a map of the world, or part of the world if an area is specified. Method uses `Basemap` to generate the map.

Parameters

- **area** (*list, optional, default: []*) – specify the coordinates of the area to plot. Use following format [llcrnrlon, llcrnrlat, urcrnrlon, urcrnrlat]
- **resolution** (*str, optional, default: c*) – resolution of the map to use. Can be c (crude), l (low), i (intermediate), h (high), f (full).
- **show_unclassified** (*bool, optional, default: False*) – True is unclassified breakwaters with a coordinate must be plotted, False is unclassified breakwaters must not be plotted.
- **exclude** (*list, optional, default: None*) – list of breakwater types to exclude from the plot

report(*save=True*, *save_path='data_report.xlsx'*, *decimals=3*)
Make a report of the data in the database

Method to generate a data report of the database. For each breakwater type the total and missing number of datapoints is determined. Furthermore, for numerical values the mean and standard deviation is also computed.

Parameters

- **save** (*bool, optional, default: True*) – If True an Excel version of the data report is generated, use `save_path` to specify a save path
- **save_path** (*str*) – File path to save the Excel file to
- **decimals** (*int, optional, default: 3*) – Number of decimal places to round to (default: 0). If decimals is negative, it specifies the number of positions to the left of the decimal point.

Returns *pd.DataFrame* – if `save` is False a DataFrame of the report is returned

scatter(*param1*, *param2*, *show_unclassified=False*, *exclude=None*, *min_data=5*, *xmax=None*, *ymax=None*, *bins_param1=10*, *bins_param2=10*)
Make a scatter plot of two parameters

Method to generate a scatter plot with histograms for two parameters in the database.

Parameters

- **param1** (*str*) – name of parameter 1
- **param2** (*str*) – name of parameter 2
- **show_unclassified** (*bool, optional, default: False*) – True is unclassified breakwaters with a coordinate must be plotted, False is unclassified breakwaters must not be plotted.
- **exclude** (*list, optional, default: None*) – list of breakwater types to exclude from the plot
- **min_data** (*int, optional, default: 5*) – minimum number of datapoints required for plotting, if the data for a bw type is less than this limit it will be skipped
- **xmax** (*float, optional, default: None*) – maximum x coordinate of the scatter plot, by default this limit is automatically determined

- **ymax** (*float, optional, default: None*) – maximum y coordinate of the scatter plot, by default this limit is automatically determined
- **bins_param1** (*str*) – number of bins for param2
- **bins_param2** (*str*) – number of bins for param2

unclassified
Get the number of unclassified breakwaters

1.14 References

Allsop, N. W., Cork, R. S., and Verhagen, H. J. (2009). A database of major breakwaters around the world. Coasts, Marine Structures and Breakwaters: Adapting to Change - Proceedings of the 9th International Conference, 2:676–679.

Battjes, J. A., & Groenendijk, H. W. (2000). Wave height distributions on shallow foreshores. Coastal Engineering, 40(3), 161–182. https://doi.org/10.1016/S0378-3839(00)00007-7

Brinch Hansen, J. (1970). A Revised and Extended Formula for Bearing Capacity. Bulletin No. 28 of the Danish Geotechnical Institute, 5–11.

CERC (1984). Shore protection manual. Vicksburg, Miss: Dept. of the Army, Waterways Experiment Station, Corps of Engineers, Coastal Engineering Research Center. https://doi.org/10.5962/bhl.title.47830

CIRIA, CUR, CETMEF (2007). The Rock Manual. The use of rock in hydraulic engineering (2nd edition). C683, CIRA, London, 2nd edition

Davila Delgado, J. M. and Hofmeyer, H. (2013). Automated generation of structural solutions based on spatial designs. Automation in Construction, 35:528–541.

Deloitte (2019). Winning with connected construction Digital opportunities in engineering and construction.

Delta Marine Consultants (2018) Guidelines for Xbloc® Concept Designs. Retrieved from: https://www.xbloc.com/sites/default/files/domain-671/documents/xbloc-guidelines_2018-671-1532949730287638300.pdf

EurOtop, 2018. Manual on wave overtopping of sea defences and related structures. An overtopping manual largely based on European research, but for worldwide application. Van der Meer, J.W., Allsop, N.W.H., Bruce, T., De Rouck, J., Kortenhaus, A., Pullen, T., Schüttrumpf, H., Troch, P. and Zanuttigh, B., http://www.overtopping-manual.com/.

Goda, Y. (1992). the Design of Upright Breakwaters. 547–568.

Goda, Y. (2000). Random Seas and Design of Maritime Structures (2nd ed.). Singapore: World Scientific.

Hudson, R. Y. (1959). Laboratory investigation of rubble-mound breakwaters. Journal of the Waterways and Harbors Division, 85, 610–659. Proc. ASCE 85 WW 3, ASCE, New York, USA

Kweon, H.-M., & Goda, Y. (1996). A parametric model for random wave deformation by breaking on arbitrary beach profiles. Coastal Engineering Proceedings, 1(25). https://doi.org/10.9753/icce.v25.%p

Laenen, K. (2000). Probabilistisch model voor het vergelijken van twee golfbrekertype op basis van economische optimalisatie. Master thesis, Delft University of Technology.

McKinsey&Company (2017). Reinventing Construction: A Route To Higher Productivity.

Nederlands Normalisatie Instituut (2002). Armourstone - Part 1: Specification NEN-EN 13383-1:2015. Delft, The Netherlands: Euronorm CEN

Nederlands Normalisatie Instituut (2002). Eurocode - Basis of structural design NEN-EN 1990:2002. Delft, The Netherlands: Euronorm CEN

Sijbesma, F. B. (2019). A parametric approach to a probabilistic design of rubble mound slope protection. Master thesis, Delft University of Technology.

Sumer, B. M., & Fredsøe, J. (2000). Experimental study of 2D scour and its protection at a rubble-mound breakwater. Coastal Engineering, 40(1), 59–87. https://doi.org/10.1016/S0378-3839(00)00006-5

Takahashi, S. (2002) Design of Vertical Breakwaters, Short course presented at the 28th ICCE, Cardiff. Updated version of Japanese Reference Document no.34, Port andHarbour Research Institute, Ministry of Transport, Yokosuka, Japan

Van den Bos, J. and Verhagen, H. (2018). Breakwater Design. Lecture notes, Delft University of Technology.

Van der Meer, J. W. (1988). "Rock slopes and gravel beaches under wave attack". Delft University of Technology. Delft, The Netherlands.

van der Meer, J. W. (1998). Geometrical design of coastal structures. In Geometrical design of coastal structures (pp. 161–175). https://doi.org/10.1201/9781315141329

Van Gent, M. R. A., Smale, A. J., & Kuiper, C. (2003). Stability of rock slopes with shallow foreshores. Coastal Structures 2003 - Proceedings of the Conference, 40733(January), 100–112. https://doi.org/10.1061/40733(147)9

Verruijt, A. (2012). Soill Mechanics. Delft, The Netherlands: TU Delft. https://geo.verruijt.net/

Winkel, S. (2020). Developing a design automation tool to improve the breakwater design process. Master thesis, Delft University of Technology.

INDEX

F

G

H

I

K

L

M

N

V

W

X

Y