XBdiagnostic Documentation

Release 1.0

Tim Leijnse

Jun 29, 2017

Contents:

'XBdiagnostic' is the project within the 'openearth' environment where diagnostic tests are created for the process-based model of XBeach. The tests run beside of the current skillbed and are more focussed on testing specific modules of the code rather then the performance of the entire model. Herefore the setup of the models is simple and only the relevant processes to test the module are turned on. The idea is that by testing specific processes and functionalities of the code, more insight is created whether they still perform as intended. Also bugs should be easier to find because of a structured development of the tests.

The idea of this online manual is to outline the setup of the framework for the diagnostic tests, the approaches for the different diagnostic tests and to make the scripts more understandable. The source code of the scripts can be found at https://github.com/openearth/xbeach-test-python.

CHAPTER 1

Project

Introduction to XBeach

XBeach is a state-of-the-art open-source process-based morphological model which was originally developed to simulate hydrodynamic and morphodynamic processes and impacts on sandy coasts with a domain size of kilometers and on the time scale of storms. Since then, the model has been applied to other types of coasts and and is continuously being developed ever since. At Deltares, the XBeach development team regularly implements new functionalities and processes according to the latest research. Continuous development of a complex model like XBeach requires continuous testing.

Testing XBeach

Currently the latest version of XBeach is tested in the Skillbed, see http://oss.deltares.nl/web/xbeach/skillbed. Here many large models are run and their results are compared with (field)data. For the diagnostic tests the focuss is more at testing specific modules, rather than the whole model. The tests will be similar to integration tests, as often used in software testing, but then in a more elaborate way. Therefore the name of diagnostic tests is chosen, with the goal of testing specific processes and functionalities of XBeach.

Structure of the diagnostic tests

The structure of different diagnostic tests is initially based on vital processes of XBeach as wave, flow, sediment transport and avalanching which are each tested in a seperate diagnostic test. Within this diagnostic tests you can have a subdivision of differenct tests/cases/runs/checks etc. Usually you want to differ certain parameters/options within a diagnostic test one at a time. This can be done by first specifying certain tests that should be varied, within these you can specify certain cases that should be varied. Within these you can have different runs and so on, see the avalanching diagnostic tests for an example. The variety of tests will later on be extended to other functionalities like the use of groundwater flow and vegetation.

Structure of the scripts

To realise the different diagnostic tests a structure of Python scripts is made (see Fig. 1.1). At first there is a user input file where all desired parameters, setting and options are specified. From here these are used in the setup and analyze_this scripts. In the setup file the desired folder structures and XBeach input files are created for the different models within one diagnostic test. To create the desired bathymetries a seperate file called bathy.py is used, which is shared between all diagnostic tests. To create the XBeach user input files (params.txt, jonswap.txt and grid files etc) are made using xbeach.py from https://github.com/openearth/xbeach-tools-python. Hereafter all these different models are run on the on the H6 cluster via the TeamCity build server using generic files check_runs.py, clean_cluster.py and runall.py (see source code).

When everything is run in Xbeach the following step is to analyse the results, using the analyze_this.py. Here every run is subjected to a series of checks to see if the results are satisfactory, performed by checks.py. Every check gets a code 0 when the result is satisfactory, 1 when the check is run but the result is not satisfacty and 2 when the check was not performed correctly. These codes are stored in a SQL type database using the script database.py. The checks and database files are also shared amongst the different diagnostic tests. At last the existence of basic logfiles should be mentioned.

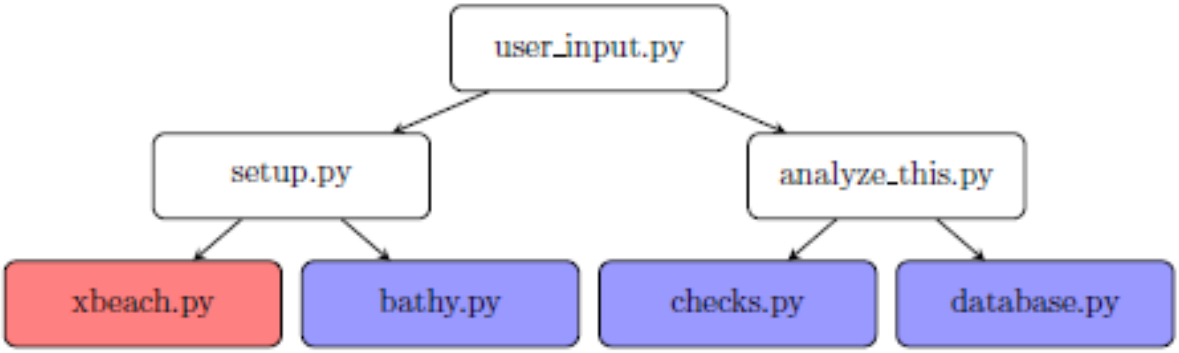


Fig. 1.1: Overview of the scripts with the white boxes representing scripts that are modified per diagnostic test. The blue boxes represent scripts that are shared between all diagnostic tests and that can be extended if a new functionality is needed. And the red box represents the shared script of openearth/xbeach-tools-python/xbeachtools.

For a description of the specific scripts see section ‘Scripts’.

Development of diagnostic tests

During the development of the framework for diagnostic tests at first a test for the XBeach module of Avalanching was created. The tests were aloborately tested and showed the potential of diagnostic tests, since multiple bugs were bound in the XBeach code. Thereafter the first steps for a diagnostic test for waves was made, however this is not completely finished and tested yet.

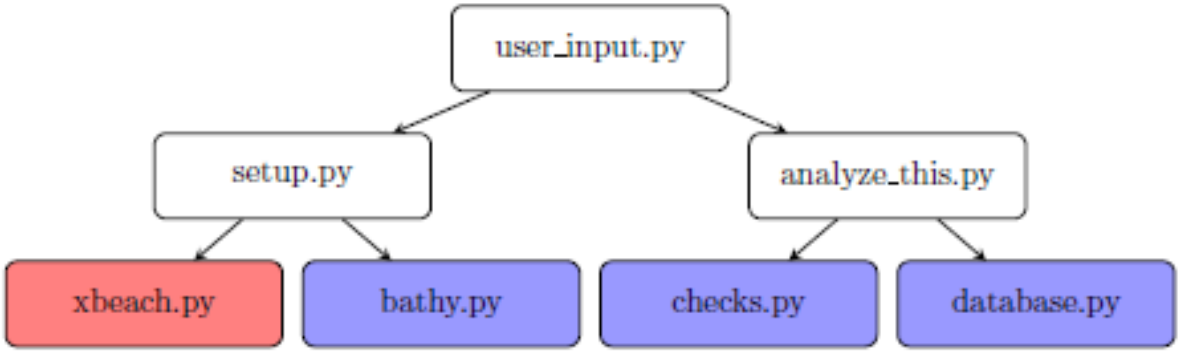
For the future the vision is to first develop diagnostic tests for all vital XBeach processes of waves, flow, sediment transport, morphology and avalanching. With this structured approach it gets more easy to pinpoint where in the code something goes as the result of a change in the source code . Also the code of XBeach gets improved as a beneficial side effect when bugs are found.

Thereafter other functionalities like groundwaterflow and vegetation can also get dedicated diagnostic tests to also ensure their performance in time.

CHAPTER 2

Scripts

In this section the general structure of all the scripts will be explained, the specific implementation for the diagnostic tests is explained per diagnostic test.



user_input

The applied structure for the user input file is the use of multiple dictionaries containing all desired parameters and options, and is altered for every different diagnostic test. These different dictionaries are also exported as txt-files so the specific input is still known afterwards. The txt-files are also used by analyze_this.py so that the analysis scripts run independent of the setup scripts.

At first there is the dictionary for parameter settings that should end up in XBeach' params.txt file (dictP). Here parameters about which processes should be turned on or off, the grid, boundaries, output and others are specified. The values specified here can be seen as the default settings for the designed diagnostic test. Later some of these parameters can be varied during different tests/cases/runs.

Thereafter comes the dictionary for bathymetry settings (dictB), which are used to construct the necessary (different) bathymetries and corresponding XBeach grid files. In here desired sizes and types of bathymetry can be specified, as well as options as for instance the ability to extend the last grid cell for a certain amount of cells.

Then comes the dictionary for other user input (dictU) regarding the types of module/tests/cases/runs and what parameters should be altered between them.

The final dictionary is regarding the applied checks in analyze_this (dictC). There can be specified what kind of individual and comparison checks are required and what the desired constraints are.

Hereafther these dictionaries are written into txt-files.

setup

The applied structure for the setup file is the use of multiple for-loops, and is altered for every different diagnostic test. There are loops for tests/cases/runs etc which are used to make this into folder structures in which the XBeach input files can be written. Within these loops specific parameters can be changed, according to user_input, so different XBeach input files can be created. The desired bathymetries are made using the generic script bathy.py. This together with the specified parameters is turned into XBeach' params.txt, x.txt, (y.txt) and z.txt files using the generic script xbeach.py. These are stored in a specific folder together with a shell script that ensures that it can be run on the cluster.

xbeach

xbeach.py is part of xbeach-tools-python (https://github.com/openearth/xbeach-tools-python) which is a toolbox constructing, running and analysing XBeach models. The script from the Python 3 branch is used, because all the other scripts are made to be compatible with Python 3 (and if possible Python 2 as well) From this script the class XBeachModel is used to construct the XBeach input files. For the class XBeachBathymetry the functionalities 'mirror', 'turn' and 'gridextend' are created, which are used to first extend the grid by copying the last grid cells for a specified number of times. Thereafter the bathymetry is mirrored when specified, and the grid and bathymetry can be turned 90 degrees as well.

bathy

bathy.py is a script to create different bathymetry profiles, and is shared between the diagnostic tests. The class Bathymetry consists of multiple default values which can be overruled to the desired parameters. The structure of the script is that you have certain 'general building blocks' and 'specific bathymetry profiles'. The first contains general shapes as a dean profile, horizontal bottom and a linear slope as well as a function to make a grid longshore uniform. The second contains different profiles which are combinations of the general building blocks, here you can think of a 2D dean profile or a partly horizontal and partly sloping profile. Both parts of the script can be extended with any profile desired.

Current building blocks consist of 'dean1' (based on OET: dean_beach_profile.m, more general than dean2) and 'dean2' (more specific use of Dean profile based on Coastal Dynamics 1 lecture notes). As well as 'hor' for a horizontal bed, 'sloping' for a sloping bed and the option 'yuniform' to convert a 1D profile into a longshore uniform 2D profile. Current profiles consist of 'flat_1d' and 'flat_2d' which are flat profiles as the name suggests. Furthermore there is 'dune_1d' and 'dune_2d' which are a combination of a Dean profile (dean2) for the offshore part and a sloping profile for the onshore part. And at last there is 'dean1_2d' which is added later for the diagnostic test for waves, which consists of an full 2D Dean profile based on dean1

Running the scripts

Now all XBeach input files are created they can be run on the cluster using the files runall.py, check_runs.py and clean_cluster.py. These are based on generic scripts and do not need explanation regarding the diagnostic tests.

analyze_this

After all XBeach models have run the results are stored in the specified folder structure. Then these results are analysed using the scripts analyze_this.py which is diagnostic test specific. The structure of the script is the same as setup.py in the way that it uses the same for-loop structure. But in this script it reads this input structure from the dictionary txt-files created by user_input.py. Then using these loops all the different netCDF XBeach output files are read, to get the desired parameters.

Thereafter starts a new for-loop to perform multiple checks on each netCDF file, as specified in dictionary C. If a certain check like 'massbalance' is specified, this will be performed using checks.py. In this generic file different kinds of checks are specified with all a specific needed input, but all with a single value as result. If a check has a satisfactory result it gets check = 0, if the test has run but the result is unsatisfactory it gets check = 1. When a check is called for but something went wrong it gets check = 2.

The results of all performed checks are stored in a database using database.py. From here all checks with a value > 0 can be used to send an error message to the responsible person.

There is also the possibility to for instance make plots here and include them in the folder structure.

checks

checks.py is a script containing different functions to test the XBeach results, and is shared between the diagnostic tests. It contains checks that are performed over the whole grid (e.g. a mass balance check) and checks that are performed over a certain transect (usually the middle one). Herefore there are also some additional functions to filter out the middle transect or calculating the slope per grid cell. The output of all checks is a single value.

Current checks can be devided into checks on the whole grid and checks on a middle transect. For the first there is 'bedlevelchange', looking if there is bed level change at all, 'massbalance', looking at the mass balance (can be used for 'zb' as well as 'zs') and 'massbalance_intime' checks the mass balance in time. For waves there is 'wave_generation' which looks on a specific location (e.g. at the offshore boundary or close to the coast) whether waves are created, and there is 'n_Hrms' which looks along the grid cells of the n-direction whether there is a large deviation in mean Hrms along the m-transect compared to the mean Hrms of the whole grid. For checks using a middle transect there are checks for checking slope in m- and n-direction ('m_slope' and 'n_slope'), bed levels across mpi boundaries in both directions ('m_mpi' and 'n_mpi') and 'rmse_comp' which calculated the Root Mean Squared Error between the final bed level of a middle transect of a run and the corresponding benchmark run.

database

database.py is a script to make and use a database to store the results of the checks per module/test/case/run/check, and is shared between the diagnostic tests. It uses a SLQ type database, for now sqlite3 for Python as a local database. The database is human readable with a program like 'DB browser for sqlite'.

read_from_database

read_from_database.py is a script to read the one and two codes from the database for a specific revision of the trunk version of XBeach. Both are captured in a dictionary and written away to a text file. When error codes have occurred these are send to the necessary recipients.

CHAPTER 3

Diagnostic test for Avalanching

In this section the implementation of the general setup of scripts as explained in 'Scripts' for the testing of the module Avalanching will be explained.

Model setup

The goal is to setup a model to check whether avalanching occurs correctly, herefore it is the idea to make a 2D bathymetry with a dune that initially is too steep. This should make sure that the dune slides away and should initiate avalanching immediately. Also the avalanching mechanism is then not reliant on other parts of the code like (long) waves and the wetcell mechanism, which makes sure that you only test the working of avalanching. This is necessary if you want to come as close to a unit test as possible with a diagnostic test.

To make sure that the dune should slide away immediately the dune slope should be larger than the critical avalanching slope under water ('wetslp') and the critical avalanching slope above water ('dryslp'), which are 0.3 and 1.0 (-) respectively by default. A 30m wide dune is chosen with a slope of 1.5, giving a height of 45m. The rest of the profile consists of a Dean profile, with a depth of about -1.15m at the offshore boundary and 0m at the dune foot.

Because the profile initially is too steep it should slide away regardless whether there are (long) waves, the modules 'swave' and 'lwave' are turned off. Because the wave modules are turned off, parameters/files like 'instat', jonswap.txt, 'bcfile' are not specified. The module 'flow' is turned off as well since it is not necessary for avalanching. On the other hand the module 'avalanching' is of course turned on. Since this module uses 'morphology', which on its part uses 'sedtrans', both these modules are turned on as well. The rest of the parameters and options are not specified and are thereby the default values are used. This holds for instance for options like 'nonh' and 'gwflow' which are turned off as default. For parameters like 'D50', 'dryslp' and 'wedslp' their default values are used.

Another important option is the choice of boundaries. To have as little influence of the boundaries as possible it is chosen to have wall boundaries everywhere, this type is a simple no flux boundary condition. This choice was later also usefull when calculating the mass balance since no mass can be transported out of the model. Other chosen input is a constant water level of 0m (zs0=0), a maximum bed level change due to avalanching of 1 m/s/m (dzmax=1) instead of the default 0.05 to speed up the process, a NetCDF output format (outputputformat=netcdf) and a default model runtime of 600s (tstop=600), which is sufficient to contain the whole avalanching phase of the dune.

Hereafter the used set of tests/cases/runs becomes important. During testing of the results of a preliminary version of the diagnostic test it became clear that it was necessary to distinct the process of avalanching in multiple directions,

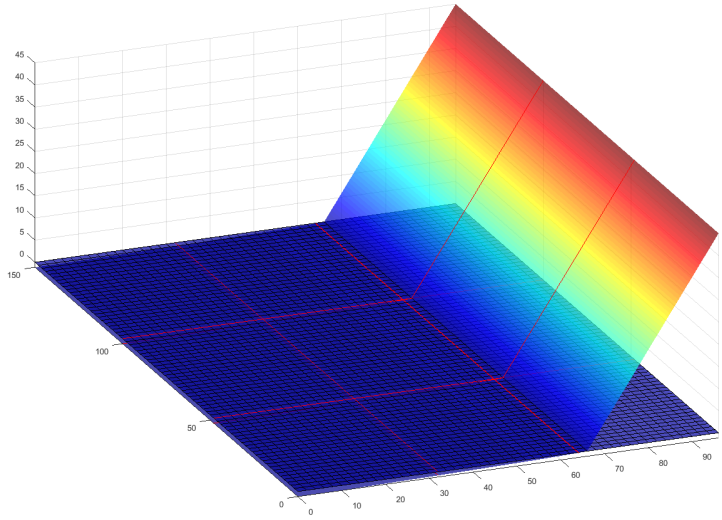


Fig. 3.1: Grid and bathymetry for the inital setup

also because the process of avalanching in x- and y-direction are coded seperately. Herefore you have tests for the positive/negative x-direction and the positive/negative y-direction. As a control there is a fifth test with a completely horizontal bathymetry for which no avalanching should occur. Thereafter you have multiple cases, it turned out that some parameters needed to be tested with other values than their default as put in the P dictionary. So apart from the standard case with those default values you have the cases: 'dzmax_0.05', 'morfac_10', '**zs0_**-1', 'zs0_45'. Per case only one parameter is changed at a time. Therafter you have multiple runs, some problems have occured with the combination of avalanching and running with MPI so this is tested. The runs 'benchmark' (=m1n1, 2D), 'm3n1'(2D), 'm1n3'(2D), 'm3n3'(2D), 'm1' (1D), 'm3' (1D) can be distinguished. Hereby the two 1D runs are not for the tests with avalanching in the y-direction, XBeach 1D is automatically run in the x-direction. The 'm' and 'n' correspond to the directions 'x' and 'y' and the numbers in how many submodels the model is devided in that direction, see the figure below:

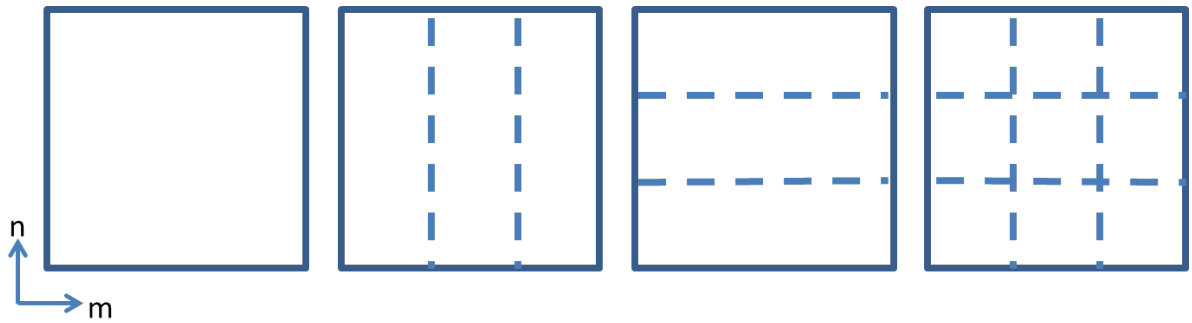


Fig. 3.2: MPI configurations m1n1 (first), m3n1 (second), m1n3 (third) and m3n3 (fourth)

In the paragraphs below notable input of the specific scripts is mentioned, if it is not yet adressed above. All the scripts and their input can be found at https://github.com/openearth/xbeach-test-python/xbeachtest.

user_input_avalanching

What can be noticed here are the different dictionaries based on the parameters/options/setup explained before. To this can be added that an extra section in the script is made to make the different cases in such a way that an extra parameter value can easily be added. Under 'CASES USER INPUT' you find the parameters that should be changed compared to dictionary P, in current setup the morfac, dzmax and zs0. This is then made into parameter lists, so the parameters can automatically be altered in setup_avalanching.py. Here also the different case names are created and all this info is then added to dictionary U. The last thing is that for a case with dzmax=0.05 you need a longer runtime 'tstop', which is therefore set to 3000s for this case.

setup_avalanching

The first part of the script makes the folder structure and varies the specific parameters as needed. Thereafter XBeach-Model from xbeach.py and Bathymetry of bathy.py are the classes used to make the bathymetry and XBeach input files. For the bathymetry there is varied between the shapes 'flat_1d', 'flat_2d', 'dune_1d' and 'dune_2d'. These are used to make the input for XBeachBathymetry from xbeach.py. In here the options of 'gridextend', 'mirror' and 'turn' were created which are called for when needed. As is common in modelling practice, there are three grid cells with equal depth at the offshore boundary. But to avoid boundary problems at dune side the grid is extended here as well. Thereafter XBeachModel is used to write all files into the needed folder.

analyze_this_avalanching

After all the different XBeach models have run the results can be analysed. First the dictionaries are loaded from the text files, after which the loops of the folder structure are used. Then for every run the needed variables zs and zb are read from the netCDF files, just as some other parameters. Thereafter a new for-loop is started for the checks to be performed, these are added to 'checklist'. Within this loop all the checks asked for in dictionary C will be performed, all with a value for check as outcome (or also a massbalance for that check). It can be seen that there are quite a number of if/elif/else statements because different tests/cases/runs require different input for the checks.

For the avalanching test there at first has been made use of the check 'bedlevelchange', to look whether there is bed level change at all. Thereafter an important check is 'massbalance' to look if the amount of mass within the model does not change. Then the slopes in m- and n-direction are checked, based on the appropriate locations and slopes specified in dictionary C. Thereafter comes the mpi check in m- and/or n-direction when appropriate for the run. At last the benchmarkcomparison in m- and/or n-direction is performed, which looks at the Root Mean Squared Error between the final bed level of the benchmark and the other corresponding runs.

(Possibly temporary:) At the end of the script a plot is made for every run to visualise the initial and final bed levels along the middle transect. With this plot many off the occurring error codes 1 can be explained.

checks

For the avalanching diagnostic test the checks 'bedlevelchange', 'massbalance', 'm_slope', 'n_slope', 'm_mpi', 'n_mpi' and 'rmse_comp' were created.

database

The only added feature here is that the massbalance result of that check can be stored in the database. This can be used to adress how serious the massbalance deviations are.

CHAPTER 4

Diagnostic test for Waves

In this section the implementation of the general setup of scripts as explained in 'Scripts' for the testing of the module Waves will be explained.

Model setup

The goal is to setup a model to check whether waves are created and propagated correctly. The test is not fully developped yet but the first setup is implemented. After collaboration the first setup focusses on the Surfbeat mode of XBeach in combination with different lateral boundary conditions. Herefore the hydrodynamic processes 'swave' and 'lwave' are turned on (just as flow for now) and morphodynamic processes as 'sedtrans', 'morphology' and 'avalanching' are turned off.

The bathymetry consists of a Dean profile from -15m to +2m, after which there is a steeper linear dune slope of 0.1. For now the grid is equidistant with dx=10m and dy=20m, but it is advised to make 'dx' depth dependent to optimise the internal timestep of XBeach because of the CFL conditions (not yet implemented, can be done as in OET: xb_grid_xgrid.m).

For the boundaries 'front' is set to 'abs_2d', 'back' is set to 'wall', 'left' and 'right' are the default of 'neumann' and 'lateralwave' is altered between 'cyclic', 'neumann' and 'wavecrest'. Furthermore as in the avalanching test also the combination with MPI is tested, but now only the 2D options 'm1n1' (the benchmark), 'm3n1', 'm1n3' and 'm3n3' are used because the addition of 1D runs is no added value.

Because wave are now added to the XBeach model, a wave spectrum should be specified. For this the commonly used Jonswap spectrum is added, with a seperate input file called jonswap.txt. In here a wave height Hm0 of 3m is specified, together with Tp=8s, an altering wave angle 'mainang', an altering value for 's' and the XBeach default values of gammajsp= 3.3 and fnyq= 0.3.

The things that are tested for now is the performance of XBeach Surfbeat under different wave angles, Jonswap spectra, lateral boundaries and MPI configurations. This leads to the following folder structure: tests ('neumann', 'cyclic', 'wavecrest'), cases ('mainang_240', 'mainang_270', 'mainang_300'), the new layer subcases('s_10', 's_100000') and runs ('benchmark', 'm3n1', 'm1n3', 'm3n3'). So an extra layer called subcases is added to change the parameter 's' of the Jonswap spectrum, for the large value of 100000 you get a spectrum of only one frequency and thus monochromatic waves and the value of 10 is the default value in XBeach.

So compared to the avalanching test there are less parameter changes, different cases and exceptions, which makes the setup and analysis a lot cleaner. In the paragraphs below notable input of the specific scripts is mentioned, if it is not yet adressed above. All the scripts and their input can be found at https://github.com/openearth/xbeach-test-python/xbeachtest.

user_input_waves

The biggest difference here is the addition of a new dictionary W, used for the wave input to create the Jonswap spectrum in setup_waves.py. Another is a smaller value for 'tintg' because a higher resolution in time is needed and the addition of more variables to 'nglobalvar'. Another thing to notice is that the parameter 'cyclic' is varied along the tests. For the test 'cyclic', cyclic= 0 but for the other tests cyclic= 0.

setup_waves

Here the extra loop for subcases is added, a different bathymetry profile is called for (dean1_2d) and there is now a section which writes the jonswap.txt files to the specific folder.

analyze_this_waves

Here also the extra loop for subcases is added, as well as reading out of more variables from the netCDF files. Hereafter there are a number of new or differenly applied tests. The check 'bedlevelchange' is used twice, one to look if 'zb' does not change during running as you would expect and one to look if 'zs' does change during running because of created waves. The check 'massbalance' is also used twice, one used on 'zb' to look if no sediment is somehow dissapearing (the boundaries are now open) and one to look at 'zs' to look at the water mass. For this last purpose also a new check is created 'massbalance_intime', which looks at the output steps if the mass balance is not too much up. This can also be used to plot this behaviour in time. Thereafter you get the two checks of 'wave_generation', where there is at first looked at the offshore boundary whether means of variables H, ue, ve, ui and vi are not zero (indicating that no waves are created). Then this also done for a location close to the coast to see if waves arrive over there. Note that ui and vi are only created at the offshore boundary so these are not checked. At last there is now the check 'n_Hrms' which looks along the grid cells of the n-direction whether there is a large deviation in mean Hrms along the m-transect compared to the mean Hrms of the whole grid. Besides the output 'check' also 'Hmean_ratio' is an output which can be and is plotted.

At least one more test should be added to in some way compare the results of the benchmark with that of the other runs. But their can probabaly be also thought of other individual checks that could be usefull.

(Possibly temporary:) After all these checks there are now 3 figures that are created per run. In the first the result of 'Hmean_ratio' is plotted along the y-axis to give more insight in the results. The second figure containts a snap shot of the Hrms on the last timestep, which gives insight in the wave direction, wave groupiness and shadow zone. The last figure plot the massbalance of 'zs' in time, which gives insight in the amount of water entering and leaving the model.

checks

For the waves diagnostic test the checks 'massbalance_intime', 'wave_generation and 'n_Hrms' were created.

database

The database is slightly altered to facility the addition of subcases.

CHAPTER 5

Acknowledgements

The project is initially developed by Tim Leijnse at Deltares as part of an internship regarding ‘Developing diagnostic tests for XBeach’ (insert link to TUD repository).

The project is currently maintained by Bas Hoonhout at Deltares.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search