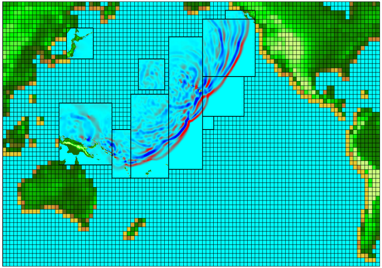
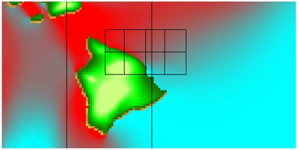
Modeling the 2011 Tohoko Tsunami with GeoClaw

David L. George[1]

[1]Cascades Volcano Observatory, U.S. Geological Survey

PASI, Valparaiso, Chile, Jan. 2013





2004 Indian Ocean Tsunami

Fault motion ≈ 600s.

Propagation ≈ 24 hrs.

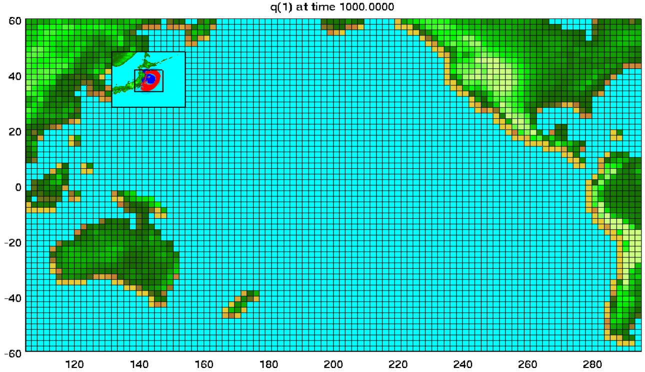
2004 Indian Ocean Tsunami

Propagation ≈ 24 hrs.

Overview

1. Example GeoClaw simulation of the Tohoku Tsunami of 2011
2. Setting-up a GeoClaw simulation
    - parameters set in file `setrun.py`
    - providing topography/bathymetry
    - providing source models
    - types of output available
3. Interpreting results of numerical simulations

March 2011 Tohoku Tsunami

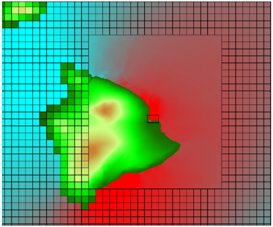


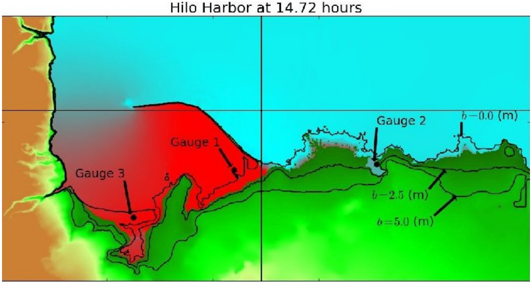
Tohoku Tsunami: Inundation modeling for Hilo

Resolving inundation and global scale propagation with 5 levels of refinement, with ratios: 8, 4, 16, 32.

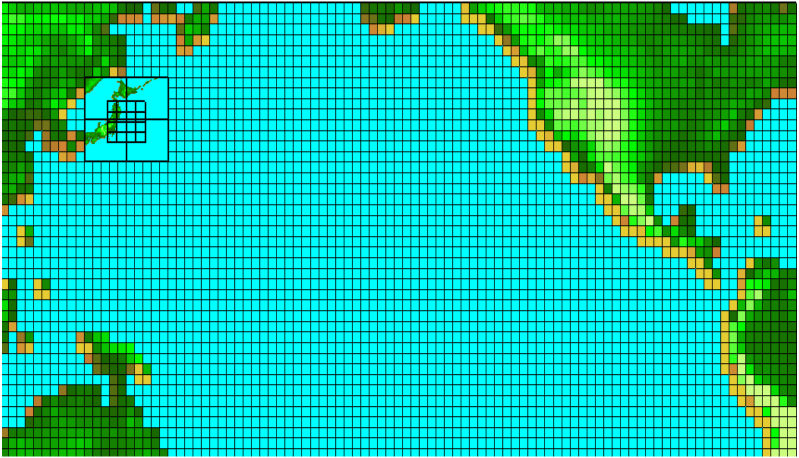
Resolution ≈ 160 km on Level 1 and ≈ 10m on Level 5.

Total refinement factor: 2^{14} in each direction!





Tohoku Tsunami: Inundation modeling for Hilo



Simulating the Tohoku tsunami with Geoclaw

What is needed by GeoClaw to simulate tsunami initiation, propagation and inundation?

1. simulation parameters in `setrun.py`
2. topography/bathymetry ("topo" files comprising the domain)
   - several standard DEM formats accepted
   - multiple files can be used
   - any resolution can be used
3. a source model (a "dtopo" file)
   - "dtopo" file describes the seafloor displacement
   - displacement can be static or dynamic
   - GeoClaw provides tools to convert standard fault parameters into dtopo

setrun.py

- A GeoClaw simulation runs in an application directory.
- All specific runtime parameters are set in `setrun.py`:

```
#------------------------------
def setrun(claw_pkg='geoclaw'):
#------------------------------

    """
    Define the parameters used for running Clawpack.

    INPUT:
        claw_pkg expected to be "geoclaw" for this setrun.

    OUTPUT:
        rundata - object of class ClawRunData

    """
```

setrun.py: *e.g.*, computational domain

```
# Number of space dimensions:
clawdata.ndim = ndim

# Lower and upper edge of computational domain:
clawdata.xlower = 105.0
clawdata.xupper = 295.0

clawdata.ylower = -60.0
clawdata.yupper =  60.0

# Number of grid cells:
clawdata.mx = 100
clawdata.my = 60
```

setrun.py: *e.g.*, output times

```
# Initial time:
clawdata.t0 = 0.0

# -------------
# Output times:
#--------------

clawdata.outstyle = 1

if clawdata.outstyle==1:
    # Output nout frames at equally spaced times:
    clawdata.nout = 80
    clawdata.tfinal = 80.0e3
```

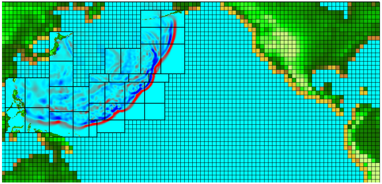
setrun.py: controlling AMR

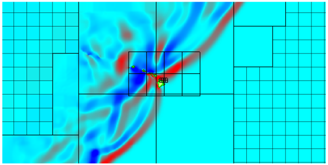
- we have multiple grid levels: level-1,. . . ,level-5.
- the level-1 grid (coarsest grid) comprises the domain.
- a level-(l+1) grid is refined from a level-1 grid by specified integer ratios: r_x^l, r_y^l

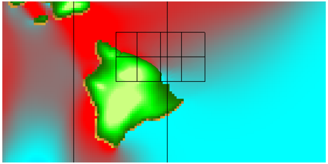
  in this example:

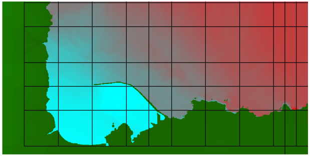
$$[r_x^{1,2,3,4}] = [r_y^{1,2,3,4}] = [16, 4, 16, 32]$$

setrun.py: controlling AMR









setrun.py: controlling AMR

```
# AMR parameters:

# max number of refinement levels:
mxnest = 5

clawdata.mxnest = -mxnest
# negative ==> anisotropic refinement in x,y,t

# List of refinement ratios at each level
#(length at least mxnest-1)

clawdata.inratx = [16,4,16,32]
clawdata.inraty = [16,4,16,32]
clawdata.inratt = [16,4,16,4]
```

`setrun.py`: controlling AMR

- Note: ordinarily a level-(l+1) grid must take $\max(r_x^l, r_y^l)$ timesteps per level-l timestep.
- tsunamis are unique in that wave-speeds are often slower on fine grids ($u \pm \sqrt{gh}$).
- $\rightarrow$ sometimes a fine grid can get away with fewer timesteps.
- GeoClaw allows temporal refinement to be taken care of automatically:

  setrun.py:

  ```
  geodata.variable_dt_refinement_ratios = True
  ```

`setrun.py`: controlling refinement

GeoClaw refinement metric: $|\eta|$

- $|\eta| \neq 0$ simply the presence of "waves"
- if $|\eta^l_{ij}| >$ `geodata.wavetolerance` $\rightarrow$ cell $\mathcal{C}^l_{ij}$ is flagged

Regions

- refinement will occur to the highest level allowed at $s = (x, y)$
- refinement can also be enforced to a minimum level at $s = (x, y)$

Summary:

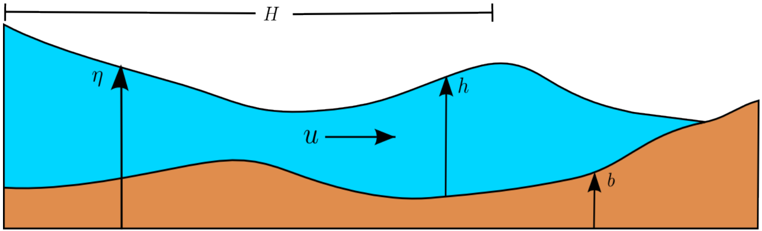
- every point s in the domain has a minimum and maximum level
- refinement will occur to the minimum level at s always
- refinement will occur to the maximum level at s if cell is flagged

due to clustering flagged cells into patches some points will be refined higher

Topography

shallow water water equations need topography value:

$$\frac{\partial h}{\partial t} + \frac{\partial (hu)}{\partial x} + \frac{\partial (hv)}{\partial y} = 0,$$
$$\frac{\partial (hu)}{\partial t} + \frac{\partial}{\partial x}(hu^2 + \frac{1}{2}gh^2) + \frac{\partial (huv)}{\partial y} = -gh\frac{\partial b}{\partial x},$$
$$\frac{\partial (hv)}{\partial t} + \frac{\partial (huv)}{\partial x} + \frac{\partial}{\partial y}(hv^2 + \frac{1}{2}gh^2) = -gh\frac{\partial b}{\partial y}.$$



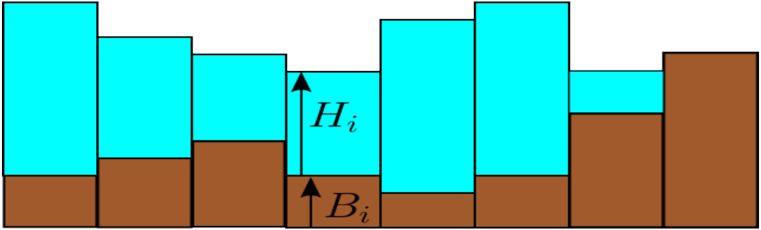
Topography

shallow water water equations need topography value:

$$\frac{\partial h}{\partial t} + \frac{\partial (hu)}{\partial x} + \frac{\partial (hv)}{\partial y} = 0,$$

$$\frac{\partial (hu)}{\partial t} + \frac{\partial}{\partial x}(hu^2 + \frac{1}{2}gh^2) + \frac{\partial (huv)}{\partial y} = -gh\frac{\partial b}{\partial x},$$

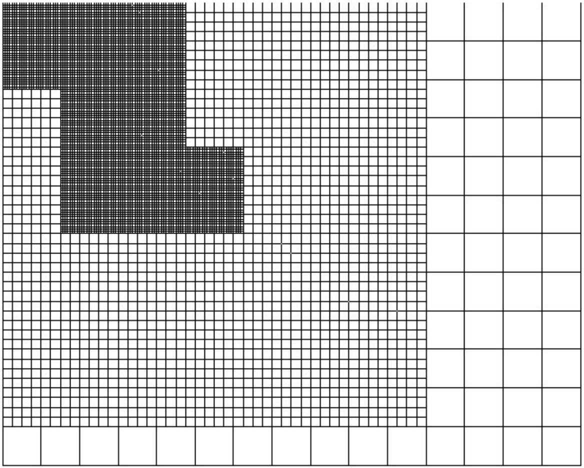
$$\frac{\partial (hv)}{\partial t} + \frac{\partial (huv)}{\partial x} + \frac{\partial}{\partial y}(hv^2 + \frac{1}{2}gh^2) = -gh\frac{\partial b}{\partial y}.$$



Topography

- topography files are DEMs: ASCII text files
- topography may be in latitude-longitude or Cartesian coordinates
- DEM grid spacing must be equal in x (longitude) and y (latitude)
- multiple DEM files (of same coordinate system) allowed
- DEM grids may overlap
- union of DEM grids must comprise computational domain
- note: $b(x, y) < 0 \rightarrow$ below sealevel

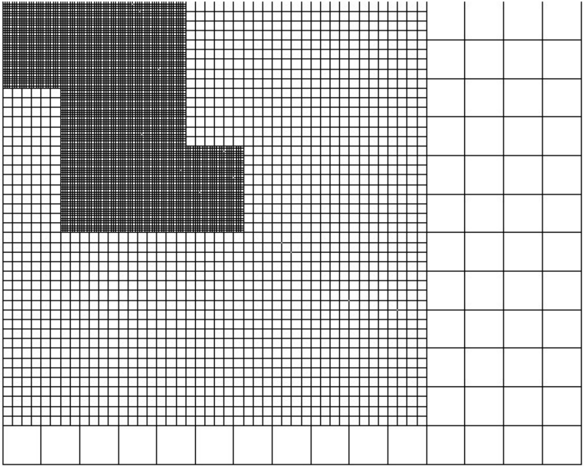
Topography



Topography

- Do not confuse topography grids with computational grids!
- DEMs are discrete uniform gridded data
- However: from DEMs GeoClaw builds a unique piecewise continuous surface: DEMs $\to B(x, y, t)$
- Some properties of $B(x, y, t)$:
    - $B(x, y, t)$ is a piecewise bilinear function
    - $B(x, y, t)$ 4 DEM nodes define each bilinear
    - $B(x, y, t)$ is continuous except at DEM boundaries
    - where DEMs overlap, finest grid is used

Topography



Topography

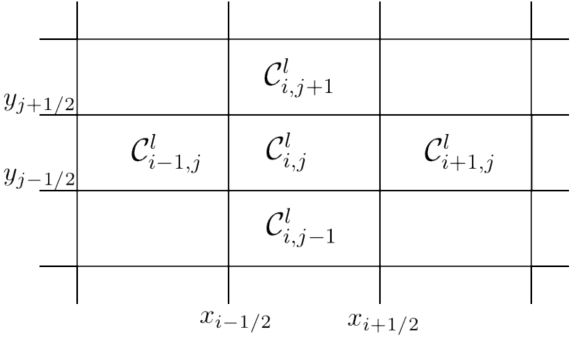
Summary:

- true topography: $b(x, y, t)$
- DEMs: $b^1_{x0,y0}, \ldots, b^1_{xN1,yM1}$; $b^2_{x0,y0}, \ldots, b^2_{xN2,yM2}$ etc.
- GeoClaw: $B(x, y, t)$
- computational grid cell $\mathcal{C}^l_{i,j}$: $B^l_{i,j}$

Topography

Determining B^l_{ij}:
$B(x,y,t)$ is exactly integrated over $\mathcal{C}^l_{ij}$.

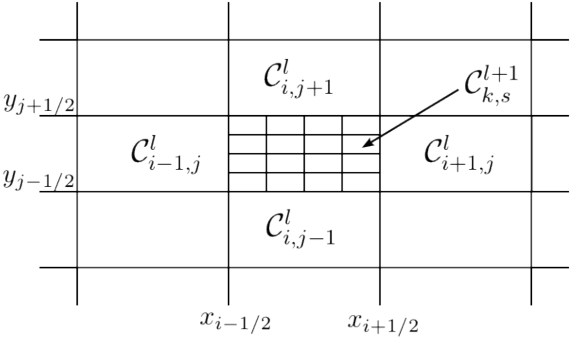
$$B^l_{ij} = \frac{1}{|\mathcal{C}^l_{ij}|}\int_{\mathcal{C}^l_{ij}} B(x,y)\,dx\,dy$$



Topography

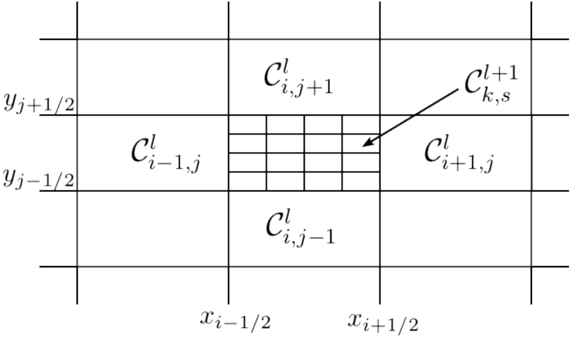
Determining B_{ks}^{l+1}:
$B(x,y,t)$ is exactly integrated over $\mathcal{C}_{ks}^{l+1}$.

$$B_{ks}^{l+1} = \frac{1}{|\mathcal{C}_{ks}^{l+1}|} \int_{\mathcal{C}_{ks}^{l+1}} B(x,y)\, dx\, dy$$



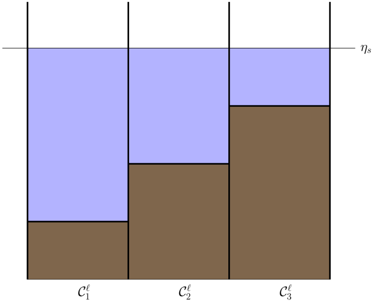
Topography

Need $|\mathcal{C}_{ij}^l|B_{ij}^l = \sum_{ks\in ij} |\mathcal{C}_{ks}^{l+1}|B_{ij}^{l+1}$



Topography

Need $|\mathcal{C}_{ij}^{l}|B_{ij}^{l} = \sum_{ks \in ij} |\mathcal{C}_{ks}^{l+1}|B_{ij}^{l+1}$



setrun.py: loading topography

```
geodata.topofiles = []
# for topography, append lines of the form
#[topotype, minlevel, maxlevel, t1, t2, fname]

geodata.topofiles.append([3, 1, 3, 0.0, 5.e3,"some/path/name"])
```

setrun.py: loading topography

```
topo = os.environ['TOPO']
topopath0 = os.path.join('etopo1min139E147E34N41N.tt3')
topopath1 = os.path.join('etopo4min100E300E65S65N.tt3')
topopath2 = os.path.join('etopo1min200E210E18N22N.tt3')
topopath3 = os.path.join('hilo_3s_E.tt3')
topopath4 = os.path.join('hilo_city_1_3s_E.tt3')


geodata.topofiles = []
# for topography, append lines of the form
#[topotype, minlevel, maxlevel, t1, t2, fname]

geodata.topofiles.append([3, 3, 3, 0.0, 5.e3, topopath0])
geodata.topofiles.append([3, 1, 2, 0.0, 1e10, topopath1])
geodata.topofiles.append([3, 1, 3, 0.0, 1e10, topopath2])
geodata.topofiles.append([3, 1, 4, 0.0, 1e10, topopath3])
geodata.topofiles.append([3, 1, 5, 0.0, 1e10, topopath4])
```

topo file formats

topotype = 1: columns of data: x, y, z

sometopofile.tt1:

```
0.0 1.0 2.56
0.1 1.0 2.44
...
...
0.0 0.9 -4.65
0.1 0.9 -4.54
...
...
1.0 0.0 3.2
```

topo file formats

topotype = 2: header describes grid format:

somefile.tt3:

```
972 ncols
547 nrows
204.9 xll
19.68 yll
9.25925925926e-05 cellsize
-9999 nodata_value
-45.63
-44.44
-39.84
...
...
12.08
```

topo file formats

topotype = 3: header describes grid format:

somefile.tt2:

```
972 ncols
547 nrows
204.9 xll
19.68 yll
9.25925925926e-05 cellsize
-9999 nodata_value
98.00827   97.43748   96.72208   95.9384    95.29736 ...
78.07087   79.46182   77.61262   76.70646   76.69659 ...
...
32.91422   30.52451   29.49668   28.92588   29.49668
```

setrun.py: controlling AMR

setting regions manually:

```
# == setregions.data values ==
geodata.regions = []
# to specify regions of refinement append lines of the form
#  [minlevel,maxlevel,t1,t2,x1,x2,y1,y2]
# geodata.regions.append([4,5,0.0,80.e3,110.,120.,0.,10.])
```

setrun.py: source model

tsunami source is a “dtopo” file
file format is similar to topo file

```
t0 x0 yN dz0
t0 x1 yN dz0
...
t0 x0 yN-1 dz0
t0 x1 yN-1 dz0
...
...
t1 x0 yN dz1
t1 x1 yN dz1
...
t1 x0 yN-1 dz1
t1 x1 yN-1 dz1
...
```

setrun.py: source model

```
geodata.dtopofiles = []
# for moving topography, append lines of the form:
#   [topotype, minlevel,maxlevel,fname] (topotype =1 only)

geodata.dtopofiles.append([1,3,3,'../topo/subfault.tt1'])
```

getting "dtopo" files

configuration file: `usgs100227.cfg`

```
Fault_Width 100.e3
Fault_Length 450.e3
Slip_Angle 104.0
Dip_Angle 14.0
Strike_Direction 16.
Dislocation 15.0
Epicenter_Latitude -35.826
Epicenter_Longitude -72.668
Focal_Depth 35.e3
mx 100
my 100
ylower -40.0
yupper -30.0
xlower -77.0
xupper -67.0
```

source models

multiple subfaults:

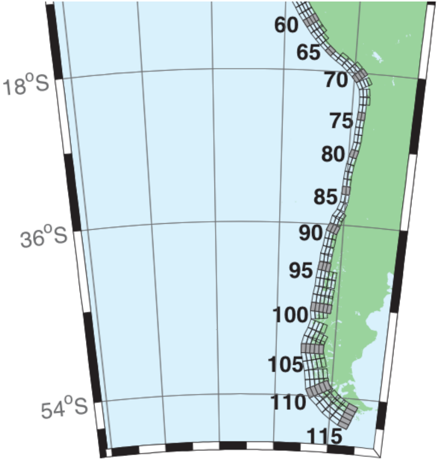


Figure B2: Central and South America Subduction Zone unit sources.

setrun.py: gauges

time-series output
"gauges:"

```
# == setgauges.data values ==
geodata.gauges = []
# append lines of the form  [gaugeno, x, y, t0, tf]
geodata.gauges.append([1, -155.056+360, 19.731, 25.e3, 40e3
geodata.gauges.append([2, -155.029+360, 19.732, 25.e3, 40e3
geodata.gauges.append([3, -155.075+360, 19.722, 25.e3, 40e3
geodata.gauges.append([4, 156.516, 19.642, 25.e3, 40e3])
```

setrun.py: gauges

National Oceanic and Atmospheric Administration — Pacific Marine Environmental Laboratory

NOAA Center for Tsunami Research

Developing methods and tools to reduce tsunami hazard and protect life

Home | Tsunami Forecasting | Hazard Assessment | Research | DART | Events | Info

DART® (Deep-ocean Assessment and Reporting of Tsunamis)

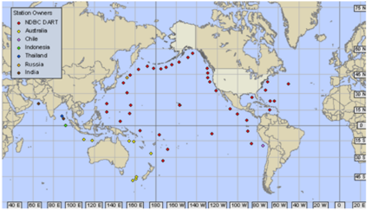
DART® tsunami technology

DART® real-time tsunami monitoring systems, developed by PMEL, are positioned at strategic locations throughout the ocean and play a critical role in tsunami forecasting.

DART® Patent Licensing Application form (PDF format) and Instructions (Word format)

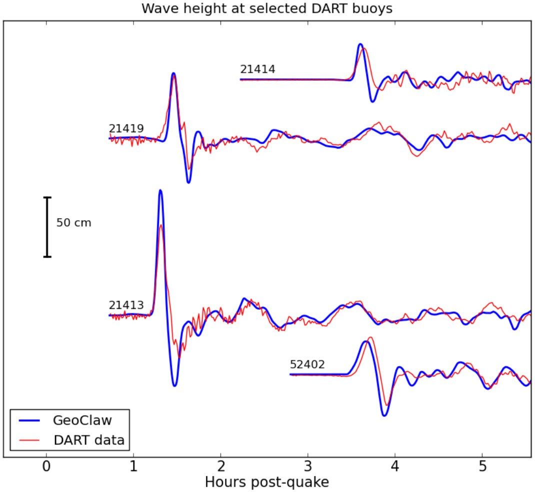
DART® - ETD buoy system
Deep Ocean Assessment of Tsunami (DART)
Easy to Deploy (ETD)

YouTube See YouTube video about the DART-ETD



The current deployed DART location (from NDBC) are shown on a map.
These can also be viewed with an interactive map (from NGDC).

setrun.py: gauges



setrun.py: misc.

```
# == setgeo.data values ==
geodata.igravity = 1
geodata.gravity = 9.81
geodata.icoordsys = 2
geodata.icoriolis = 1
geodata.Rearth = Rearth

# == settsunami.data values ==
geodata.sealevel = 0.
geodata.drytolerance = 1.e-3
geodata.wavetolerance = 5.e-2
geodata.depthdeep = 1.e2
geodata.maxleveldeep = 5
geodata.ifriction = 1
geodata.coeffmanning = 0.025
geodata.frictiondepth = 100.0
```