geoclaw_driver/run_CC_CSZ_South.py

```
"""
================================================

N O R T H   S U L A W E S I   T S U N A M I
configuration : PTHA scenario for north sulawesi

================================================

run_PTHA for Sulawesi scenario:

"""

from clawpack.geoclaw import dtopotools
import os
import numpy as np
import matplotlib.pyplot as plt

try:
    CLAW = os.environ['CLAW']
except:
    raise Exception("*** Must first set CLAW enviornment variable")

try:
    import rcrom
except:
    raise Exception("*** rcrom.py not in path: set PYTHONPATH")

# Scratch directory for storing topo and dtopo files:
scratch_dir = os.path.join(CLAW, 'geoclaw', 'scratch')
driver_home = os.getcwd()      # directory where all runs will be done

# ============================================================================
# setrun, setgeo for the coarse grid runs are defined in setrun.py
#
#    setrun_coarse
#    setgeo_coarse
#
# these set as the default template, then the iteration function for the
# GeoClawInput class is used to appropriately change the settings,
# e.g., fine grid runs, earthquake magnitudes, run to final time, etc.
# ============================================================================

from setrun import setrun_coarse, setgeo_coarse

# ============================================================================
# Iterator for the runs
#
#       iter_fun() is the iterator function for the GeoClawInput class
```

```
#      run parameters (grid-size, earthquake parameters) as well as
#      run_ids, rundirs, etc. are changed here
#
# ============================================================================

def iter_fun(self):
    r"""
    This function will be used to iterate GeoClawInput
    Total 802 runs:

         run_id + 1
         ------------------------------------------------------------
         1    100    200    300    400    500    600    700    800 801 802
  grid-size  |          coarse          |          fine           | c | f |
     Mw      |  8.6 |  8.8 |  9.0 |  9.2 |  8.6 |  8.8 |  9.0 |  9.2 | 0.0 0.0

    """

    run_id = self._run_id
    etopo_dir = driver_home
    topodir = driver_home

    # load input info
    if self._input_info == None:
        scn_fname = os.path.join(self._run_home,'scenario_pts.txt')
        scn = np.loadtxt(scn_fname)
        scn_list = scn.tolist()
    else:
        scn_list = self._input_info

    # total number of runs
    # M = len(scn_list)

    M = 1 # <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< test

    N = 8*M + 2     # 8*M runs plus two empty bathymetry runs

    if run_id == N:
        raise StopIteration()

    else:

        #=========================
        # set coarse and fine grids
        #
        t_shelf = 0.   # time approaching continental slope
        t_harbor = 0.  # time approaching harbor

        if ((run_id >= 0) and (run_id < 4*M)) or (run_id == 8*M):
            #------------------
```

```
        # setrun for coarse
        #
        grid = 'coarse'

        self._rundata.amrdata.amr_levels_max = 4
        # coarse grid run = 10sec
        # dx = 30min, 5min, 1min, 10sec
        self._rundata.amrdata.refinement_ratios_x = [6, 5, 6]
        self._rundata.amrdata.refinement_ratios_y = [6, 5, 6]
        self._rundata.amrdata.refinement_ratios_t = [6, 5, 6]


        # add topography (coarse)
        topofiles = self._rundata.topo_data.topofiles
        # for topography, append lines of the form
        #    [topotype, minlevel, maxlevel, t1, t2, fname]
        topofiles = []

        topofiles.append([3, 1, 4, 0., 1.e10, \
              os.path.join(etopo_dir, 'etopo1_-130_-124_38_45_1min.asc')])
        topofiles.append([-3, 3, 4, 0., 1.e10, \
              os.path.join(topodir, 'cc-1sec.asc')])

        # add regions
        regions = self._rundata.regiondata.regions
        # between shelf and CC
        regions = []
        regions.append(\
              [2, 3, t_shelf, 1e9, -125, -124.05, 40.5, 43])
        regions.append(\
              [3, 4, t_harbor, 1e9, -124.26, -124.14, 41.67,   41.79])
        regions.append(\
              [4, 4, t_harbor, 1e9, -124.218,-124.17, 41.7345, 41.77])

        # == fgmax.data values ==
        fgmax_files = self._rundata.fgmax_data.fgmax_files
        fgmax_files = []

        # for fixed grids append to this list names of any fgmax input files
        fgmax1_fname = os.path.join(driver_home,'fgmax1_coarse.txt')
        fgmax2_fname = os.path.join(driver_home,'fgmax2_coarse.txt')
        fgmax3_fname = os.path.join(driver_home,'fgmax3_coarse.txt')

        fgmax_files.append(fgmax1_fname)
        fgmax_files.append(fgmax2_fname)
        fgmax_files.append(fgmax3_fname)

        self._rundata.fgmax_data.num_fgmax_val = 2


```

```
    elif ((run_id >= 4*M) and (run_id < 8*M)) or (run_id == 8*M+1):
        #----------------
        # setrun for fine
        #
        grid = 'fine'

        self._rundata.amrdata.amr_levels_max = 6

        # fine grid run = 2/3 seconds
        # dx = 30 minutes, 5 minutes, 1 minute, 10 seconds, 2 seconds, 2/3 seconds
        self._rundata.amrdata.refinement_ratios_x = [6, 5, 6, 5, 3]
        self._rundata.amrdata.refinement_ratios_y = [6, 5, 6, 5, 3]
        self._rundata.amrdata.refinement_ratios_t = [6, 5, 6, 5, 3]

        regions = self._rundata.regiondata.regions
        regions = []
        # between shelf and CC
        regions.append(\
              [2, 4, t_shelf, 1e9, -125, -124.05, 40.5, 43])
        regions.append(\
              [4, 5, t_harbor, 1e9, -124.26, -124.14, 41.67,   41.79])
        regions.append(\
              [6, 6, t_harbor, 1e9, -124.218,-124.17, 41.7345, 41.77])

        # add topography (fine)
        topofiles = self._rundata.topo_data.topofiles
        # for topography, append lines of the form
        #    [topotype, minlevel, maxlevel, t1, t2, fname]
        topofiles = []

        topofiles.append([3, 1, 6, 0., 1.e10, \
              os.path.join(etopo_dir, 'etopo1_-130_-124_38_45_1min.asc')])
        topofiles.append([-3, 4, 6, 0., 1.e10, \
              os.path.join(topodir, 'cc-1sec.asc')])
        topofiles.append([3, 6, 6, 0., 1.e10, \
              os.path.join(topodir,'cc-1_3sec-c_pierless.asc')])

         # == fgmax.data values ==
        fgmax_files = self._rundata.fgmax_data.fgmax_files
        fgmax_files = []

        # for fixed grids append to this list names of any fgmax input files
        fgmax1_fname = os.path.join(driver_home,'fgmax1_fine.txt')
        fgmax2_fname = os.path.join(driver_home,'fgmax2_fine.txt')
        fgmax3_fname = os.path.join(driver_home,'fgmax3_fine.txt')

        fgmax_files.append(fgmax1_fname)
        fgmax_files.append(fgmax2_fname)
        fgmax_files.append(fgmax3_fname)

```

```
199             self._rundata.fgmax_data.num_fgmax_val = 2
200
201
202
203         #
204         # set desired magnitude
205         #
206         if ((run_id >= 0) and (run_id < M)) \
207                         or ((run_id >= 4*M) and (run_id < 5*M)):
208             self.KL_Mw_desired = 8.6
209         elif ((run_id >= M) and (run_id < 2*M)) \
210                         or ((run_id >= 5*M) and (run_id < 6*M)):
211             self.KL_Mw_desired = 8.8
212         elif ((run_id >= 2*M) and (run_id < 3*M)) \
213                         or ((run_id >= 6*M) and (run_id < 7*M)):
214             self.KL_Mw_desired = 9.0
215         elif ((run_id >= 3*M) and (run_id < 4*M)) \
216                         or ((run_id >= 7*M) and (run_id < 8*M)):
217             self.KL_Mw_desired = 9.2
218
219         #
220         # set slip distribution
221         #
222         # run_id_mod = run_id - 100*(run_id/100)
223         run_id_mod = run_id % 100  # ensures integer index
224         m = scn_list[run_id_mod]
225         self.set_KL_slip(m)
226
227         if run_id < 8*M:
228             dir_grid_Mw = '../geoclaw_output/' + str(grid) + '_' + str(self.KL_Mw_desired)
229             self._rundir = os.path.join(dir_grid_Mw, 'run_' + str(run_id_mod))
230         else:
231             # empty runs to obtain bathymetry
232
233             dir_grid_Mw = '../geoclaw_output/' + str(grid) + '_B0'
234             self._rundir = dir_grid_Mw
235             self.KL_Mw_desired = 0.0
236             self.set_KL_slip([0.]*len(m))   # set output
237             self._rundata.clawdata.output_times = [1.0, 3.0]
238
239         self._run_id += 1
240
241         return self
242
243
244
245 if __name__=='__main__':
246
247
248     # ============================================================================
```

```
    # Set CSZ fault geometry / parameters
    #
    # Restrict to southern portion of CSZ:
    # The experiments peformed in the paper use only the southern portion of CSZ
    # the first 8 subfaults from those above.
    #
    # ==============================================================================



    column_map = {"longitude":1, "latitude":2, "depth":3, "strike":4,
                  "length":5, "width":6, "dip":7}
    defaults = {'rake': 90, 'slip':1.0}
    coordinate_specification = 'top center'
    input_units = {'slip': 'm', 'depth': 'km', 'length': 'km', 'width': 'km'}
    rupture_type = 'static'
    skiprows = 1
    delimiter = ','

    fault = dtopotools.Fault()
    fault.read('CSZe01.csv', column_map, coordinate_specification,
               rupture_type,skiprows, delimiter, input_units, defaults)
    print ("There are %s subfaults" % len(fault.subfaults))

    for s in fault.subfaults:
        s.longitude = s.longitude - 360.  # adjust to W coordinates

    # Select only southern subfaults
    fault.subfaults = fault.subfaults[:8]

    # Read topography for contour lines:
    from clawpack.geoclaw.topotools import Topography
    topo = Topography()
    topo.read('../DataFiles/etopo1_-130_-124_38_45_1min.asc',3)


    if 0:
        plt.figure(figsize=(10,4))
        ax = plt.subplot(121);
        fault.plot_subfaults(ax)
        plt.xticks(range(-126,-123));
        plt.contourf(topo.X,topo.Y,topo.Z,[0,20000],colors=[[.3,1,.3]])
        plt.savefig('fault.png', dpi=200)

    # Subdivide each subfault further
    new_subfaults = []  # to accumulate all new subfaults
    phi_plate = 60.  # angle oceanic plate moves clockwise from north,
                     # to set rake

    for subfault in fault.subfaults:
```

```
299         subfault.rake = subfault.strike - phi_plate - 180.
300         # subdivide into nstrike x ndip subfaults,
301         # based on the dimensions of the fault:
302         nstrike = int(subfault.length/8000)
303         ndip = int(subfault.width/8000)
304         f = dtopotools.SubdividedPlaneFault(subfault, nstrike, ndip)
305         new_subfaults = new_subfaults + f.subfaults
306 
307     # reset fault.subfaults to the new list of all subfaults after subdividing:
308     new_fault = dtopotools.Fault(subfaults = new_subfaults)
309     n = len(new_fault.subfaults)
310     print ("Subdivided fault has %s subfaults" % n)
311 
312     # set up taper function w.r.t depth
313     def tau(d):
314         return  1. - np.exp((d - max_depth)*5./max_depth)
315 
316 
317     # Correlation lengths Lstrike and Ldip:
318     Lstrike = 130e3
319     Ldip = 40e3
320     max_depth = 20000.
321 
322 
323     # ============================================================================
324     # Execute runs
325     # ============================================================================
326 
327 
328     drom0 = rcrom.Drom()    # initialize Drom object
329 
330     drom0.GeoClawInput.fault = new_fault    # set fault
331     drom0.GeoClawInput.set_iter(iter_fun)   # set iterator
332     drom0.GeoClawInput.set_rundata(setrun=setrun_coarse, setgeo=setgeo_coarse)
333     drom0.GeoClawInput.KL_expand(Lstrike=Lstrike,Ldip=Ldip,\
334                 distribution='Lognormal', tau=tau, nterms=20, KL_Mw_desired=9.0)
335 
336 
337 
338     for geoclawinput0 in drom0.GeoClawInput:
339 
340         print(geoclawinput0._rundir + ': ' + str(geoclawinput0.KL_Mw_desired))
341         drom0.evaluate_hdm()    # run geoclaw
342 # do ivar = 1, nvar
343 #         qr(ivar,lind) = qc1d(ivar,index)
344 #       end do
345 
346 
```