NERA

Network of European Research Infrastructures for Earthquake Risk Assessment and Mitigation

Report

Fast Finite fault determination

Activity:	*JRA2: Tools for real time seismology, acquisition and mining*
Activity number:	*D12.4*
Deliverable:	*Toolbox 3: fast finite fault determinations*
Deliverable number:	*D12.4*
Responsible activity leader:	*Alberto Michelini*
Responsible participant:	*INGV*
Author:	*Alberto Michelini*

Seventh Framework Programme
EC project number: 262330

Summary

This report addresses the work done within the WP12 for what concerns the development and the enablement of software designed to calculate rapidly and insert into shakemap of the causative, finite fault(s). This is relevant in order to provide more reliable shakemaps of events larger than M5.5, approximately.

The toolbox software includes the two following procedures

1. Fault scaling (INGV)
2. Fault creation (INGV)

The two procedures allow for the rapid calculation of the fault extension from magnitude using the well known empirical relations of Wells and Coppersmith (1994) (New empirical relationships among magnitude, rupture length, rupture width, rupture area, and surface displacement. *Bulletin of the Seismological Society of America*, *84*(4), 974–1002.). The fault creation procedure is necessary in order to insert the fault into the input parameter files of the software shakemap.

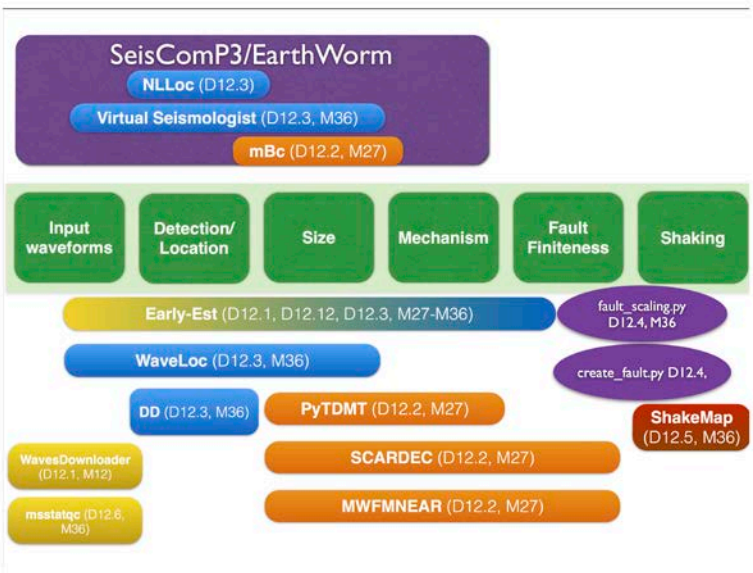

Figure 1. Simplified diagram showing the software implemented in JRA2 and the associated deliverable. The D12.4 deliverable corresponds to utility software shown in the deep purple color to the right.

Preamble

This deliverable is extremely concise since it only refers to the utility software required to interface the results obtained by the earthquake location and source mechanism software of deliverables D12.2 and D12.3 into the shakemap software.

Fault scaling

This software consists of a python script (`MwtoSRL_RLD_RW_RA_X.py; X=N,R,SS`) that determines various quantities of interest (Surface Rupture Length (SRL), sub-surface Rupture L. (RLD), down-dip rupture width (RW) and Rupture Area (RA). The regression coefficients are taken from Wells & Coppersmith (BSSA, 1994, Table 2a). For completeness, the three routines for normal (N), strike slip (SS) and reverse (R) fault faulting are reported below.

```
#
def MwtoSRL_RLD_RW_RA_N(Mw):
    """
    Determines Surface Rupture Length (SRL), sub-surface Rupture L. (RLD)
    down-dip rupture width (RW) and Rupture Area (RA)  from moment magnitude
    for a Normal fault.
    The regression is taken from Wells & Coppersmith (BSSA, 1994, Table 2a).
    """
    SRL=-2.01 + 0.5 * Mw
    RLD=-1.88 + 0.50 * Mw
    RW=-1.14 + 0.35 * Mw
    RA=-2.87 + 0.82 * Mw
    return 10**SRL,10**RLD,10**RW,10**RA
#
def MwtoSRL_RLD_RW_RA_SS(Mw):
    """
    Determines Surface Rupture Length (SRL), sub-surface Rupture L. (RLD)
    down-dip rupture width (RW) and Rupture Area (RA)  from moment magnitude
    for a Strike-Slip fault.
    The regression is taken from Wells & Coppersmith (BSSA, 1994, Table 2a).
    """
    SRL=-3.55 + 0.74 * Mw
    RLD=-2.57 + 0.62 * Mw
    RW=-0.76 + 0.27 * Mw
    RA=-3.42 + 0.90 * Mw
    return 10**SRL,10**RLD,10**RW,10**RA
#
def MwtoSRL_RLD_RW_RA_R(Mw):
    """
    Determines Surface Rupture Length (SRL), sub-surface Rupture L. (RLD)
    down-dip rupture width (RW) and Rupture Area (RA)  from moment magnitude
    for a Reverse fault.
    The regression is taken from Wells & Coppersmith (BSSA, 1994, Table 2a).
    """
    SRL=-2.86 + 0.63 * Mw
    RLD=-2.42 + 0.58 * Mw
    RW=-1.61 + 0.41 * Mw
    RA=-3.99 + 0.98 * Mw
    return 10**SRL,10**RLD,10**RW,10**RA
#
```

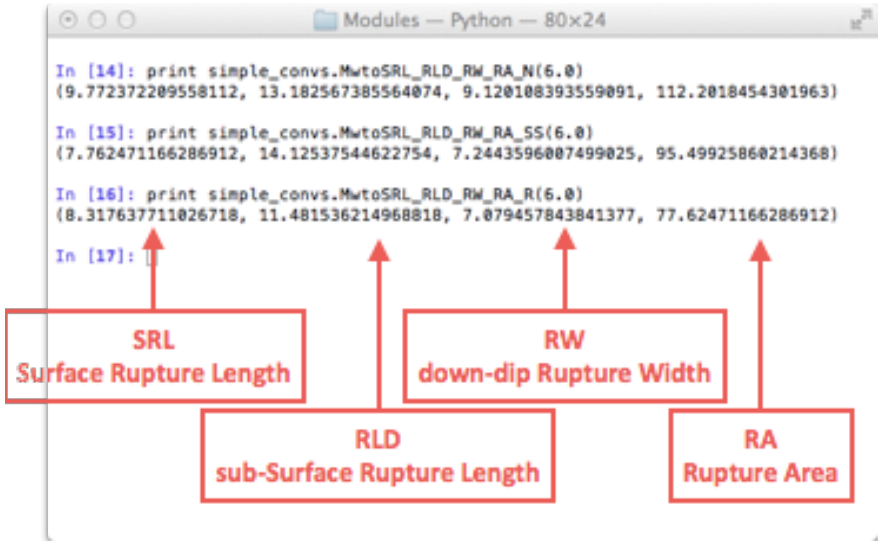

Figure 2. Example of application of the magnitude to fault extension routines for a M6 earthquake.

Fault creation

This software consists also of a python script (`create_fault.py`) that is listed in the box below.

```
#!/usr/bin/env python
# encoding: utf-8

import os, sys, optparse
#
from numpy import *
from pylab import *
#
from mpl_toolkits.basemap.pyproj import Proj
from mpl_toolkits.basemap.pyproj import Geod
from mpl_toolkits.basemap import Basemap as Basemap

q = optparse.OptionParser()
q.add_option('--verbose','-v',action='store_true',help='print debugging
information to stout')
q.add_option('--lat_eq',action='store',help="Earthquake Latitude")
q.add_option('--lon_eq',action='store',help="Earthquake Longitude")
q.add_option('--z_eq',action='store',help="Earthquake Depth")

q.add_option('--strike',action='store',help="Strike")
q.add_option('--dip',action='store',help="Dip")

q.add_option('--length',action='store',help="Fault Length taking EQ hypo as
center")
q.add_option('--width',action='store',help="Fault Width taking EQ hypo as
center")

(options,arguments)=q.parse_args()

if options.lat_eq==None:
   raise UserWarning("No EQ latitude supplied")
   quit()
if options.lon_eq==None:
   raise UserWarning("No EQ longitude supplied")
   quit()
if options.z_eq==None:
   raise UserWarning("No EQ depth supplied")
   quit()
if options.strike==None:
   raise UserWarning("No EQ mechanism strike supplied")
   quit()
if options.dip==None:
   raise UserWarning("No EQ mechanism  supplied")
   quit()
if options.length==None:
   raise UserWarning("No EQ fault length supplied")
   quit()
if options.width==None:
   raise UserWarning("No EQ fault width supplied")
   quit()


if options.verbose:
  print ""
  print "Input parameters:"
  print "-----------------"
  print "Hypocenter        = %s %s
%s"%(options.lat_eq,options.lon_eq,options.z_eq)
  print "Mechanism (strike,dip)    = %s %s"%(options.strike,options.dip)
  print "Fault dimensions (length,width) %s %s"%(options.length,options.width)

#
p = Proj(proj='utm',zone=32,ellps='WGS84')
#
```

```
# epicenter
epicenter=(float(options.lat_eq),float(options.lon_eq))
depth=-float(options.z_eq)*1000.
# define grid origin
(lat0,lon0)=epicenter
#
rotang_deg=float(options.strike)
rotang_rad= rotang_deg * pi / 180.0

dip_ang=float(options.dip)
dip_ang_rad=dip_ang * pi / 180.0

X0,Y0 = p(lon0, lat0)
XYZ0=array([[X0],[Y0],[depth]])
#
fault_dimensions=(float(options.width)*1000.,float(options.length)*1000.)
(Xlength,Ylength)=fault_dimensions
#
# start from midddle of the axis
Xmid=Xlength/2.0
Ymid=Ylength/2.0
#
# create horizontal fault with vertices counterclockwise (x1,y1), (x2,y1),
(x2,y2), (x1,y2), (x1,y1)
Xs=[X0-Xmid,X0+Xmid,X0+Xmid,X0-Xmid,X0-Xmid]
Ys=[Y0-Ymid,Y0-Ymid,Y0+Ymid,Y0+Ymid,Y0-Ymid]
Zs=[depth,depth,depth,depth,depth]
#
# direction cosines of the starting reference frame
X1=array([1,0,0])
X2=array([0,1,0])
X3=array([0,0,1])
#
XX=array([X1,X2,X3])
# Set up dip rotation matrix - rotation about the Y axis pointing N
#
U1=array([cos(dip_ang_rad),0.,sin(dip_ang_rad)])
U2=array([0.,1.,0.])
U3=array([-sin(dip_ang_rad),0.,cos(dip_ang_rad)])
#
U=array([U1,U2,U3])

# set up the rotation matrix
r1=[]
#print XX
for j in range(len(U1)):
    for i in range(len(X1)):
        r1.append(dot(U[j],XX[i]))

#r=array(r1)
R1=reshape(array(r1),(3,3))
#
# set horizontal rotation about the vertical axis
U1=array([cos(rotang_rad),sin(rotang_rad),0.])
U2=array([-sin(rotang_rad),cos(rotang_rad),0.])
U3=array([0.,0.,1.])
#
U=array([U1,U2,U3])
#
```

```
#
r2=[]
for j in range(len(U1)):
    for i in range(len(X1)):
        r2.append(dot(U[j],XX[i]))

R2=reshape(array(r2),(3,3))
#
lons=[]
lats=[]
#
lonrots=[]
latrots=[]
zrots=[]
#
for i in range(len(Xs)):
#   for y in Ys:
          # inverse transform needed later
          lon, lat = p(Xs[i], Ys[i], inverse=True)
          lons.append(lon)
          lats.append(lat)
          #
          # start first rotation for dip
          X=array([[Xs[i]],[Ys[i]],[depth]]) - XYZ0
          #
          Xrot=dot(R1,X) + XYZ0
          #
          xrot=Xrot[0,0]
          yrot=Xrot[1,0]
          zrot=Xrot[2,0]
          #
          # start second rotation for azimuth
          X=array([[xrot],[yrot],[zrot]]) - XYZ0
          Xrot=dot(R2,X) + XYZ0

          xrot=Xrot[0,0]
          yrot=Xrot[1,0]
          zrot=Xrot[2,0]
          #
          lonrot, latrot = p(xrot, yrot, inverse=True) # inverse transform
          lonrots.append(lonrot)
          latrots.append(latrot)
          zrots.append(zrot)

f = open('rotated_fault.txt', 'w')
for i in range(len(latrots)):
    f.write("%10.5f %10.5f %10.3f\n" %(latrots[i],lonrots[i],-zrots[i]/1000.))
f.close()
# begin plotting of the points
# Lambert Conformal Conic map.
m = Basemap(llcrnrlon=5.,llcrnrlat=35.,urcrnrlon=23.,urcrnrlat=48.,
            projection='lcc',lat_1=38.,lat_2=45.,lon_0=10.,
            resolution ='l',area_thresh=1000.)
# create figure.
fig=figure()

m.drawcoastlines()
m.drawcountries()
m.drawmapboundary(fill_color='#99ffff')
m.fillcontinents(color='#cc9966',lake_color='#99ffff')

m.drawparallels(arange(35,50,2),labels=[1,1,0,0])
m.drawmeridians(arange(5,20,2),labels=[0,0,0,1])

#
# compute the native map projection coordinates for cities.
x,y = m(lons,lats)
```

```
#
m.drawparallels(arange(35,50,2),labels=[1,1,0,0])
m.drawmeridians(arange(5,20,2),labels=[0,0,0,1])

#
# compute the native map projection coordinates for cities.
x,y = m(lons,lats)
# plot filled circles at the locations of the cities.
m.plot(x,y,'b-')

x,y = m(lonrots,latrots)
m.plot(x,y,'r-',ms=5.0)

show()
#
```

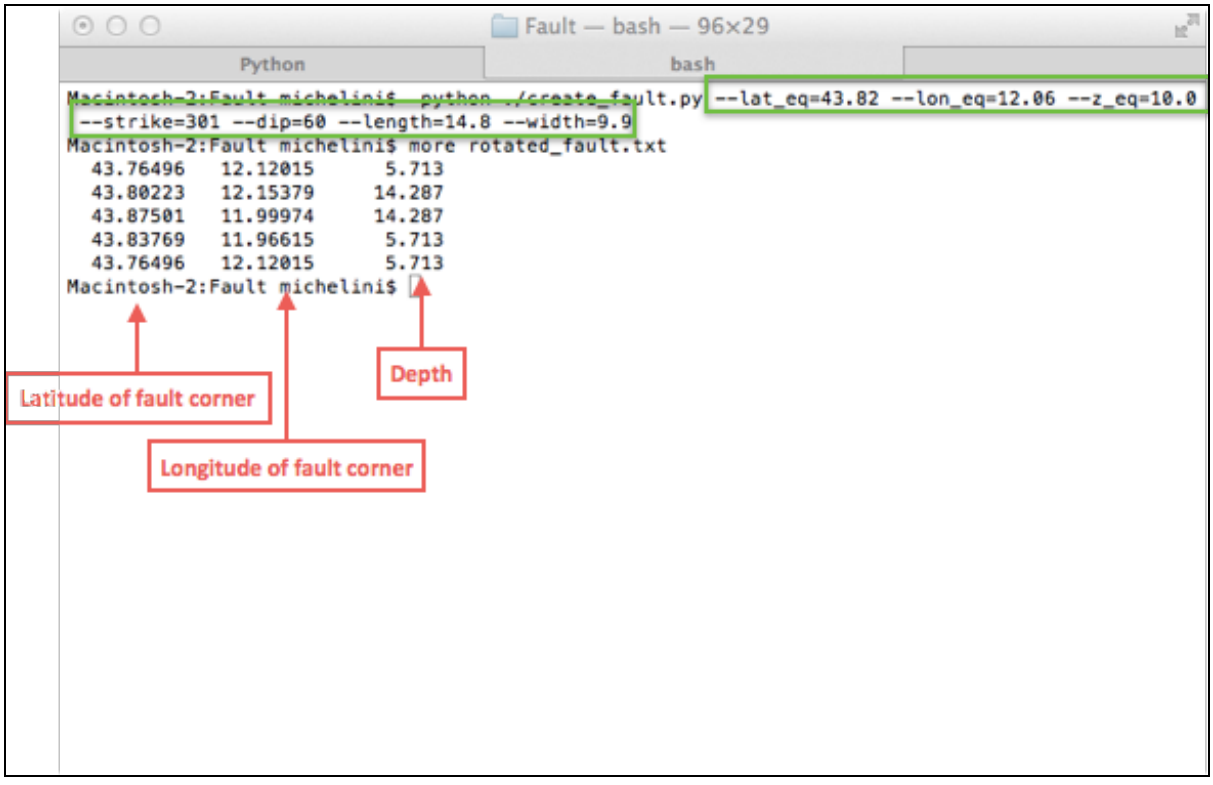

Figure 3. Example application of the create_fault.py procedure.

Conclusions

The two scripts provided above allow for a very fast – though manual since fault insertion must be performed carefully – procedure for inserting the causative fault into the Shakemap procedure (Wald, D. J., Quitoriano, V., Heaton, T. H., Kanamori, H., Scrivner, C. W., & Worden, C. B. (1999). TriNet ``ShakeMaps“”: Rapid Generation of Peak Ground Motion and Intensity Maps for Earthquakes in Southern California. *Earthquake Spectra*, *Earthquake Spectra*, 537).