Table of Contents

Preface

This Python-style Coulomb package is used to compute Coulomb stress changes on fixed receiver faults and varying receiver faults. It is evolved from the original AutoCoulomb program (Wang et al., 2021), which was developed using the MATLAB, Fortran and Shell languages and runs on a Mac or Linux operating system. The prior AutoCoulomb program is suitable for the computation of Coulomb stress changes resulting from rectangular dislocations, while the adapted Python-style program here crunches data starting from the level of stress tensors in addition to maintaining the main functionality of dealing with rectangular

dislocations for the prior AutoCoulomb program. The adaptation is driven by the consideration that some types of stress tensors are associated with volcanism or surface loading rather than earthquake faulting, and are sometimes generated using point sources or triangular dislocation elements via dislocation modeling, surface load modeling or finite element simulation. Plus, wrapping up the core of the Coulomb failure model in the Python language allows the package to be imported into the Python world, extending the application of the Coulomb failure model to various stress modeling.

(on Jan.4,2024 by Jianjun Wang, Wuhan University. jjwang@sgg.whu.edu.cn)

Preparation

Create a virtual environment

- download the coulomb package (coulomb-master)
- create a folder in a terminal: mkdir coulomb_dir
- move 'coulomb-master' into the folder 'coulomb_dir' and unzip it
- change the current directory: cd ./coulomb_dir
- create a virtual environment: python3 -m venv Coulomb_ENV
- activate the virtual environment: source ./Coulomb_ENV/bin/activate
- update the pip package: python -m pip install --upgrade pip

> Once activated, keep the virtual environment.

Install dependent packages

- **used packages:**
  - **numpy**
  - **matplotlib**
  - **scipy**
  - **pathlib**
  - shutil
  - re
  - os
  - **importlib**
  - **f2py**
  - **jupyter**

- **installation command: pip install package_names**

> - It would be much better to install all packages as aformentioned beforehand.
> - If one wouldn't like to work in the Jupyter environment, there is no need to install Jupyter and to create a kernel later. It's just fine to work in the Python environment via Python scripts or command lines in a Python terminal.

Preinstall tools

The tools of **gcc** and **gfortran** should be installed in advance (e.g., brew install gcc gfortran).

Install the coulomb package

- go to the unzipped folder *coulomb-master*: cd ./coulomb-master_
- install the coulomb package: source ./run.sh; cd ..

Build a kernel of the Jupyter

- **Build a kernel:**
  - pip install ipykernel
  - python -m ipykernel install --user --name Coulomb_ENV --display-name Coulomb_ENV

Open the Jupyter

- create a new notebook: File-> New Notebook
- change the current kernel: Kernel->Change kernel->Coulomb_ENV

Data format

Slip model

./coulomb-master/coulomb/data/_slipmodel.txt

> 1

30 102 0 100 20 -50 50 -20 0 90 90 1 0 0

- 1: the number of fault planes or subpatches. It's also the total number of the lines of the slip-model file excluding the first line.
- 30: the latitude of the reference point on the fault plane, in degrees.
- 120: the longitude of the reference point on the fault plane, in degrees.
- 0: the depth of the reference point on the fault plane, in km.
- 100: the length of the fault plane, in km.
- 20: the width of the fault plane, in km.
- -50: AL1. the distance from the reference point to the left edge of the fault plane is |AL1|, in km.
- 50: AL2. the distance from the reference point to the right edge of the fault plane is AL2, in km.
- -20: AW1. the distance from the reference point to the bottom edge of the fault plane is |AW1|, in km.
- 0: AW2. the distance from the reference point to the upper edge of the fault plane is AW2, in km.
- 90: the strike angle of the fault plane, in degrees.
- 90: the dip angle of the fault plane, in degrees.
- 1: the strike-slip dislocation along strike, in meters.
- 0: the dip-slip dislocation along updip, in meters.
- 0: the tensile-slip dislocation along normal to the fault plane, in meters.

./coulomb-master/coulomb/data/_slipmodelxy.txt

> 1

0 0 0 100 20 -50 50 -20 0 90 90 1 0 0

- 1: the number of fault planes or subpatches. It's also the total number of the lines of the slip-model file excluding the first line.
- 0: the x component of the reference point on the fault plane, in degrees. The x axis is due north.
- 0: the y component of the reference point on the fault plane, in degrees. The y axis is due east.
- 0: the depth of the reference point on the fault plane, in km.
- 100: the length of the fault plane, in km.
- 20: the width of the fault plane, in km.
- -50: AL1. the distance from the reference point to the left edge of the fault plane is |AL1|, in km.
- 50: AL2. the distance from the reference point to the right edge of the fault plane is AL2, in km.
- -20: AW1. the distance from the reference point to the bottom edge of the fault plane is |AW1|, in km.
- 0: AW2. the distance from the reference point to the upper edge of the fault plane is AW2, in km.

- 90: the strike angle of the fault plane, in degrees.
- 90: the dip angle of the fault plane, in degrees.
- 1: the strike-slip dislocation along strike, in meters.
- 0: the dip-slip dislocation along updip, in meters.
- 0: the tensile-slip dislocation along normal to the fault plane, in meters.

Sampling points

./coulomb-master/coulomb/data/_sampling.txt

> 1681
>
> 29.000000 101.000000 10.000000

- 1681: the total number of sampling points.
- 29: the latitude of the first grid point, in degrees.
- 101: the longitude of the first grid point, in degrees.
- 10: the depth of the first grid point, in km.

./coulomb-master/coulomb/data/_samplingvaryingpoints.txt

> 1681
>
> 29.000000 101.000000 10.000000 90.000000
> 90.000000 0.000000 0.400000 0.000000

- 1681: the total number of sampling points.
- 29: the latitude of the first grid point, in degrees.
- 101: the longitude of the first grid point, in degrees.
- 10: the depth of the first grid point, in km.
- 90: the strike angle of a receiver fault, in degrees.
- 90: the dip angle of the receiver fault, in degrees.
- 0: the rake angle of the receiver fault, in degrees.
- 0.4: the friction coefficient.
- 0.0: the skempton coefficient.

./coulomb-master/coulomb/data/_samplingxy.txt

> 1681

-100.000000 -100.000000 10.000000

- 1681: the total number of sampling points.
- -100: the x component of the first grid point, in km. The x axis is due north.

- -100: the y component of the first grid point, in km. The y axis is due east.
- 10: the depth of the first grid point, in km.

./coulomb-master/data/_samplingvaryingpointsxy.txt

> 1681

-100.000000 -100.000000 10.000000 90.000000 90.000000 0.000000 0.400000 0.000000

- 1681: the total number of sampling points.
- -100: the x component of the first grid point, in km. The x axis is due north.
- -100: the y component of the first grid point, in km. The y axis is due east.
- 10: the depth of the first grid point, in km.
- 90: the strike angle of a receiver fault, in degrees.
- 90: the dip angle of the receiver fault, in degrees.
- 0: the rake angle of the receiver fault, in degrees.
- 0.4: the friction coefficient.
- 0.0: the skempton coefficient.

Stresses

./coulomb-master/coulomb/data/_stress.txt

> e11 e12 e13 e22 e23 e33

1.515333e-01 1.023907e-01 2.225554e-02 1.013377e-01 1.680584e-02 2.453851e-03

- the first line is a header line. e11, e12, e13, e22, e23 and e33 are the six components of the stress tensors that are located in a local coordinate system whose x is due north, y is due east and z is upward. The six components have units of bars.
- 1.515333e-01: e11 or exx
- 1.023907e-01: e12 or exy
- 2.225554e-02: e13 or exz
- 1.013377e-01: e22 or eyy
- 1.680584e-02: e23 or eyz
- 2.453851e-03: e33 or ezz

Profile

./coulomb-master/coulomb/data/_profile.txt

> 1

30 101 0 200 30 0 200 -30 0 90 90 0 0 0 0 0 0

- 1: the number of profiles. It's also the total number of the lines of the profile file excluding the first line.
- 30: the latitude of the reference point on the profile plane, in degrees.
- 101: the longitude of the reference point on the profile plane, in degrees.
- 0: the depth of the reference point on the profile plane, in km.
- 200: the length of the profile plane, in km.
- 30: the width of the profile plane, in km.
- 0: AL1. the distance from the reference point to the left edge of the profile plane is |AL1|, in km.
- 200: AL2. the distance from the reference point to the right edge of the profile plane is AL2, in km.
- -30: AW1. the distance from the reference point to the bottom edge of the profile plane is |AW1|, in km.
- 0: AW2. the distance from the reference point to the upper edge of the profile plane is AW2, in km.
- 90: the strike angle of the profile plane, in degrees.
- 90: the dip angle of the profile plane, in degrees.
- 0: the strike-slip dislocation along strike, in meters. This is a dummy value.
- 0: the dip-slip dislocation along updip, in meters. This is a dummy value.
- 0: the tensile-slip dislocation along normal to the fault plane, in meters. This is a dummy value.

Aftershocks

./coulomb-master/coulomb/data/_aftershocks.txt

> lon. lat. depth

102 30 10

- 102: the longitude of a aftershock, in degrees.
- 30: the latitude of the aftershock, in degrees.
- 10: the depth of the aftershock, in km.

Coulomb stress changes

The basic format of a file with Coulomb stress changes is as follows: The first line is a headerline with five columns. The first column is the longitude in degrees, the second one is the latitude in degrees, the third one is the shear stress changes in bars, the fourth one is the normal stress changes in bars, and the fifth one is the Coulomb stress

changes in bars. The first line is followed by the data block, each column of which corresponds to the column in the header row.

Generate sampling points

sampling_points_given_depth(hrange,vrange,depth=10,outputfile='samplingpoints.tx

case 1: generate a sampling file associated with a Cartesian coordinate system whose x axis is due north and y axis is due east

> The data format of the following outputfile is referred to _samplingxy.txt.

In [1]:
```
import coulomb as cs
import numpy as np
hrange=np.array([-100,100,5]) #[minx,maxx,dx]
vrange=np.array([-100,100,5]) #[miny,maxy,dy]
cs.sampling_points_given_depth(hrange,vrange,depth=10,
                          outputfile='samplingxy.txt')
```

```
/Users/mac/Downloads/coulomb_dir/samplingxy.txt is created.
```

case 2: generate a sampling file associated with a geodetic coordinate system

> The data format of the following outputfile is referred to _sampling.txt.

In [2]:
```
import coulomb as cs
import numpy as np
hrange=np.array([101,103,0.05])
vrange=np.array([29,31,0.05])
cs.sampling_points_given_depth(hrange,vrange,depth=10,
                               outputfile='sampling.txt')
```

```
/Users/mac/Downloads/coulomb_dir/sampling.txt is created.
```

A quick and easy recipe

- To compute Coulomb stress changes from the level of the stress tensors that are primed, go to section 10 by invoking the 'onestepCFS' function. In this case, the Coulomb stress changes can be resolved on fixed or varying receiver faults.
- To compute Coulomb stress changes from the level of a rectangular source fault, go to section 11 by invoking the 'AutoCoulomb' function. In this case, the Coulomb stress changes can also be resolved on fixed or varying receiver faults.

CFF

The function 'CFF' is used to compute the Coulomb stress change with a single row of stress tensor and a single receiver fault. The single row of stress tensor is in the order of e11, e12, e13, e22, e23 and e33. The subscripts of 1, 2 and 3 correspond to the x, y and z axes, respectively. The x axis is due north, the y axis is due east and the z axis is upward. The single receiver fault is related to the three parameters of strike, dip and rake, three of which are scalar. The shear, normal and Coulomb stresses are in bars hereinafter, which are the same as those of the six components of the stress tensor. The strike, dip and rake angles are in degrees hereinafter.

CFF(stress, strike, dip, rake, friction, skempton)

In [3]:
```
import coulomb as cs
import numpy as np
```

In [4]:
```
x=np.array([2,6,4,1,5,3])
shearstress,normalstress,coulombstress=cs.CFF(x,1,2,3,0.4,0.0)
print('shear stress: ',shearstress, 'bar\n',
      'normal stress: ',normalstress, 'bar\n',
      'Coulomb stress: ',coulombstress, 'bar')
```

```
shear stress:  4.033615118568241 bar
 normal stress:  3.3411689643875904 bar
 Coulomb stress:  5.370082704323277 bar
```

computeCFS

The function 'computeCFS' is used to compute the Coulomb stress changes on fixed receiver faults at many grid points. The six components of the stress tensor at each of these grid points are also in the order of e11, e12, e13, e22, e23 and e33, which have the same meaning as those in the above case of CFF. Regarding the fixed receiver faults, the strike, dip and rake angles are the same at all grid points. By projecting the stress tensors onto the fixed receiver faults, the resulting Coulomb stress changes are saved in the file named coulombfile. The coulombfile includes a header line and a subsequent data block with nrow times 3. The first column is the shear stress changes, the second one the normal stress changes and the third one the Coulomb stress changes.

computeCFS(stressfile, coulombfile, strike, dip, rake, friction, skempton)

In [5]:
```
import coulomb as cs
import numpy as np
import os
#generate a stress file
x=np.array([[2,6,4,1,5,3],[2,6,4,1,5,3]])
stressfile='temp_stress.txt';
cs.writetextfile(stressfile,x,headers=['e11 e12 e13 e22 e23 e33'])
#set input parameters
coulombfile='temp_coulomb.txt'
strike=1
```

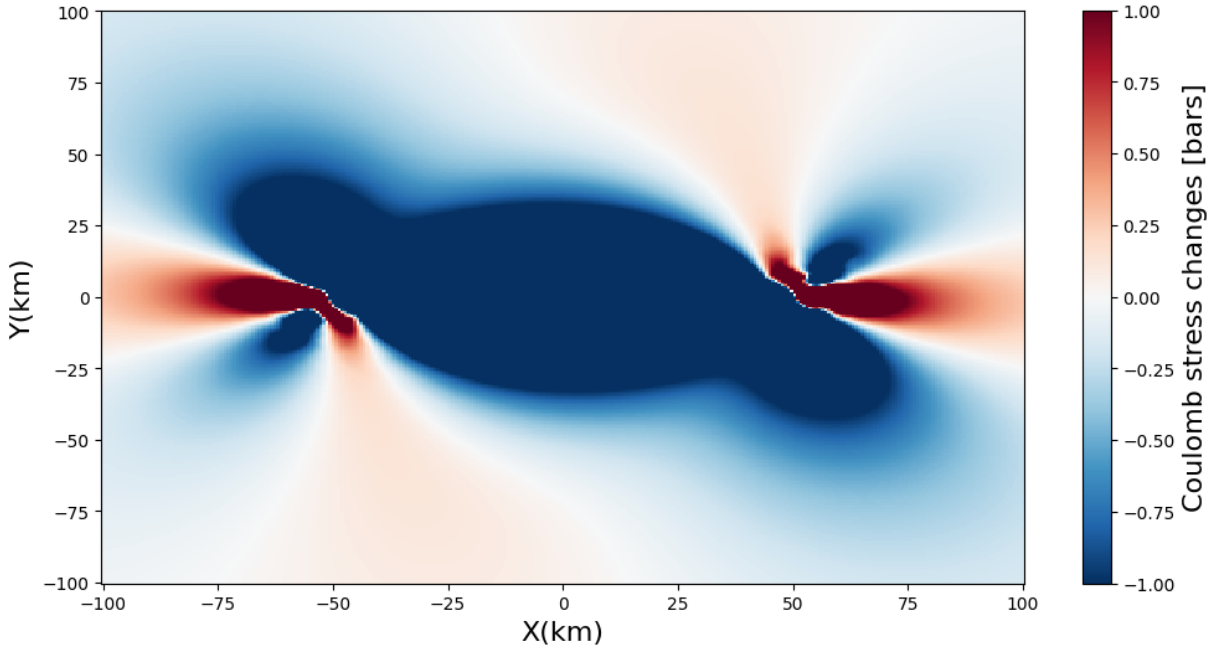
```
dip=2
rake=3
friction=0.4
skempton=0.0
#compute CFS
cs.computeCFS(stressfile,coulombfile,strike,dip,rake,friction,skempton)
data=cs.readtextfile(coulombfile,headerlines=1)
print('shear stress: ',data[:,0], 'bar\n',
      'normal stress: ',data[:,1], 'bar\n',
      'Coulomb stress: ',data[:,2], 'bar')
os.remove(stressfile)
os.remove(coulombfile)
```

```
temp_stress.txt is saved.
 -----the parameters of receiver faults-----
   1.0000000000000000        2.0000000000000000        3.0000000000000000
   0.40000000000000002        0.0000000000000000
 processing...
temp_coulomb.txt is created.
shear stress:  [4.03361512 4.03361512] bar
 normal stress:  [3.34116896 3.34116896] bar
 Coulomb stress:  [5.3700827 5.3700827] bar
```

In the following case, the stress tensors are read from the built-in file (_stress.txt). The resulting Coulomb stress changes are written to the file **'coulomb.txt'**. This file has three columns with a headerline. The first column is related to shear stress changes, the second normal stress changes and the third Coulomb stress changes. In order to display the Coulomb stress map, the first two columns of the sampling points from the built-in file (_samplingxy.txt) and the third column of the Coulomb stress changes from the file **'coulomb.txt'** are concatenated.

In [6]:
```
import coulomb as cs
import numpy as np
stressfile='_stress.txt' #it's a built-in file
stressfile=cs.getabspath(stressfile)
print(stressfile)
coulombfile='coulomb.txt'
strike=90
dip=90
rake=0
friction=0.4
skempton=0.0
cs.computeCFS(stressfile,coulombfile,strike,dip,rake,friction,skempton)
data=cs.readtextfile(coulombfile,headerlines=1)
#read data from the built-in data package.
coord=cs.readtextfile('_samplingxy.txt',headerlines=1,bpackage=1)
coord_data=np.stack((coord.T[1],coord.T[0],data.T[2]),axis=1)
#it's crucial to remove rows with nan before drawing a CFS map.
coord_data=cs.removenan(coord_data)
cs.drawCFS(coord_data.T[0],coord_data.T[1],coord_data.T[2],
           bgeo=False,ndenser=300,cr=[-1,1])
```

```
/Users/mac/Downloads/coulomb_dir/coulomb-master/coulomb/data/_stress.txt
 -----the parameters of receiver faults-----
   90.000000000000000        90.000000000000000        0.0000000000000000
   0.40000000000000002        0.0000000000000000
 processing...
coulomb.txt is created.
*the lines with nan*
[[-50.   0.  nan]
 [ 50.   0.  nan]]
**********
```



arrayCFS

The function 'arrayCFS' is used to compute Coulomb stress changes on fixed receiver faults at multiple grid points or on varying receiver faults at those grid points. The difference between the 'arrayCFS' function and the 'computeCFS' function is that the 'arrayCFS' function can now handle the calculation of Coulomb stress changes on varying receiver faults. In addition, stress tensors are now not read from a stress file but instead inputted from an input argument. Also, the results of shear, normal and Coulomb stress changes are the return values of the function rather than a result that is saved to a file.

arrayCFS(stress, strike, dip, rake, friction, skempton)

If Coulomb stress changes are to be resolved on fixed receiver faults, the arguments of friction and skempton should be scalar. If Coulomb stress changes are to be resolved on varying receiver faults, the lengths of the arguments of strike, dip and rake should be the same, which should be equal to the row of the argument of stress. The arguments of friction and skempton are either scalars or vectors with the same

length as the arguments of strike, dip or rake. In a simple way, make their lengths equal.

In [7]:
```
import coulomb as cs
import numpy as np
x=np.array([[2,6,4,1,5,3],[2,6,4,1,5,3]])
shearstress,normalstress,coulombstress=cs.arrayCFS(x,[1],[2],[3],
                                                   [0.4],[0.0])
print('shear stress: ',shearstress, 'bar\n',
      'normal stress: ',normalstress, 'bar\n',
      'Coulomb stress: ',coulombstress, 'bar')
print('-----------------------------------------')
shearstress,normalstress,coulombstress=cs.arrayCFS(x,[1,1],[2,2],[3,3],
                                                   [0.4],[0.0])
print('shear stress: ',shearstress, 'bar\n',
      'normal stress: ',normalstress, 'bar\n',
      'Coulomb stress: ',coulombstress, 'bar')
print('-----------------------------------------')
shearstress,normalstress,coulombstress=cs.arrayCFS(x,[1,1],[2,2],[3,3],
                                                   [0.4,0.4],[0.0,0.0])
print('shear stress: ',shearstress, 'bar\n',
      'normal stress: ',normalstress, 'bar\n',
      'Coulomb stress: ',coulombstress, 'bar')
```

```
shear stress:  [4.03361512 4.03361512] bar
 normal stress:  [3.34116896 3.34116896] bar
 Coulomb stress:  [5.3700827 5.3700827] bar
-----------------------------------------
shear stress:  [4.03361512 4.03361512] bar
 normal stress:  [3.34116896 3.34116896] bar
 Coulomb stress:  [5.3700827 5.3700827] bar
-----------------------------------------
shear stress:  [4.03361512 4.03361512] bar
 normal stress:  [3.34116896 3.34116896] bar
 Coulomb stress:  [5.3700827 5.3700827] bar
```

onestepCFS

The function 'onestepCFS' is used to compute Coulomb stress changes on fixed receiver faults or varying receiver faults. Stress tensors are read from a stress file. The sampling points are read from a sampling file. Receiver faults come from the sampling file or are asigned with the argument of **kwargs.

If Coulomb stress changes are calculated on fixed receiver faults and their strike, dip and rake angles are given via the keyworded, variable-length argumuent list ***kwargs*, this function is equivalent to the function of 'computeCFS'. If Coulomb stress changes are calculated on different receiver faults and their strike, dip and rake angles are also given via the argument list, this function is equivalent to the function of 'arrayCFS'. In this case, the only difference is that the stress tensors are here read from the stress file rather than given as an input argument as in the 'arrayCFS' function.

onestepCFS(stressfile, samplingfile, coulombfile='coulomb.txt', bdraw=True, bgeo=True, cr=[-0.5, 0.5], ct=False, ndenser=300, cmap='RdBu_r',kwargs)**

- If the value of the argument bdraw is True, a Coulomb stress map is to be drawn. Otherwise, the map will not be displayed. If only to calculate Coulomb stress changes at some scattered points, set bdraw=False.
- If the sampling points from the sampling file named as the value of the argument of samplingfile are related to a geodetic coordinate system, set bgeo=True. If they are related to a Cartesian coordinate system, set bgeo=False.
- All keyworded, variable-length arguments should emerge at the end of the input argument list.

Set fixed receiver faults with kwargs

- **case 1: one row of stress tensors**

In [8]:
```
import coulomb as cs
import numpy as np
import os
#generate a stress file
x=np.array([[2,6,4,1,5,3]])
stressfile='temp_stress.txt';
cs.writetextfile(stressfile,x,headers=['e11 e12 e13 e22 e23 e33'])
#generate a sampling file with fixed receiver faults
samplingfile='temp_sampling.txt'
y=np.array([[1,1,1]]) #[x,y,depth]
cs.writetextfile(samplingfile,y,headers=['1'])
#set values for input arguments
coulombfile='temp_coulomb.txt'
strike=1
dip=2
rake=3
friction=0.4
skempton=0.0
cs.onestepCFS(stressfile,samplingfile,coulombfile,bdraw=False,
              strike=strike,dip=dip,rake=rake,
              friction=friction,skempton=skempton)
data=cs.readtextfile(coulombfile,headerlines=1)
print('shear stress: ',data[:,3], 'bar\n',
      'normal stress: ',data[:,4], 'bar\n',
      'Coulomb stress: ',data[:,5], 'bar')
os.remove(stressfile)
os.remove(samplingfile)
os.remove(coulombfile)
```

```
temp_stress.txt is saved.
temp_sampling.txt is saved.
temp_coulomb.txt is saved.
shear stress:  [4.03361512] bar
 normal stress:  [3.34116896] bar
 Coulomb stress:  [5.3700827] bar
```

- **case 2: two rows of stress tensors**

In [9]:
```
import coulomb as cs
import numpy as np
import os
#generate a stress file
x=np.array([[2,6,4,1,5,3],[2,6,4,1,5,3]])
stressfile='temp_stress.txt';
cs.writetextfile(stressfile,x,headers=['e11 e12 e13 e22 e23 e33'])
#generate a sampling file with fixed(varying) receiver faults
samplingfile='temp_sampling.txt'
y=np.array([[1,1,1],[1,1,1]]) #[x,y,depth]
cs.writetextfile(samplingfile,y,headers=['1'])
#set values for input arguments
coulombfile='temp_coulomb.txt'
strike=np.array([1,1])
dip=np.array([2,2])
rake=np.array([3,3])
friction=np.array([0.4,0.4])
skempton=np.array([0.0,0.0])

strike=[1,1]
dip=[2,2]
rake=[3,3]
friction=[0.4,0.4]
skempton=[0.0,0.0]

cs.onestepCFS(stressfile,samplingfile,coulombfile,bdraw=False,
              strike=strike,dip=dip,rake=rake,
              friction=friction,skempton=skempton)
data=cs.readtextfile(coulombfile,headerlines=1)
print('shear stress: ',data[:,3], 'bar\n',
      'normal stress: ',data[:,4], 'bar\n',
      'Coulomb stress: ',data[:,5], 'bar')
os.remove(stressfile)
os.remove(samplingfile)
os.remove(coulombfile)
```

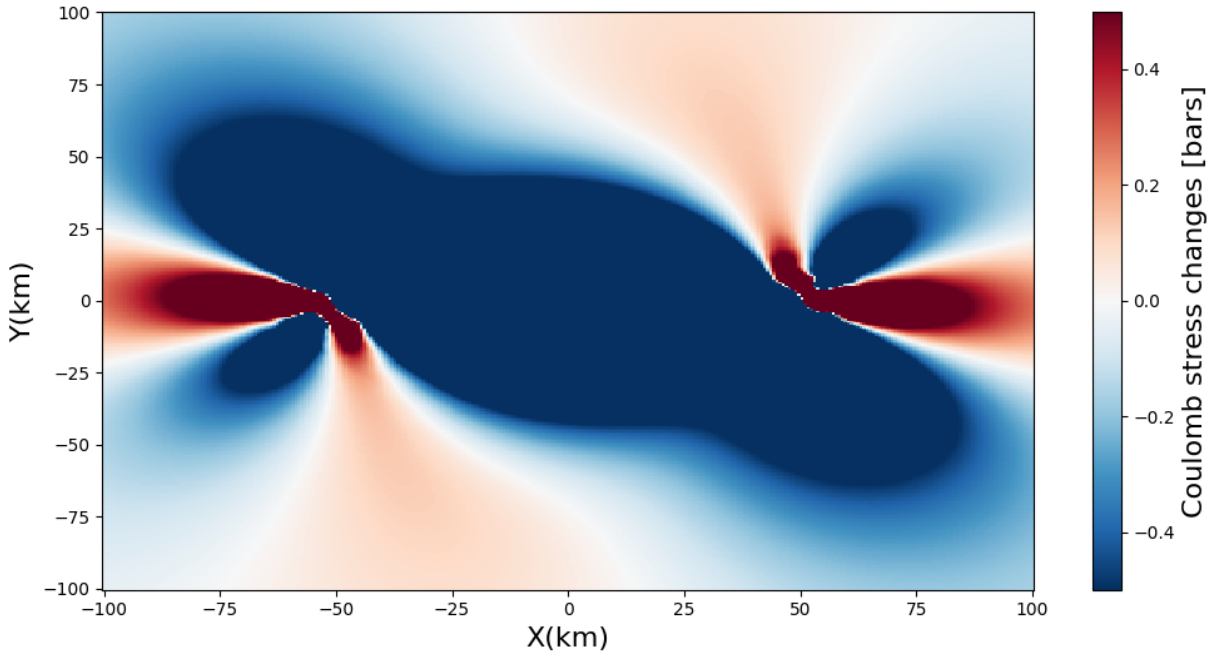
```
temp_stress.txt is saved.
temp_sampling.txt is saved.
temp_coulomb.txt is saved.
shear stress:  [4.03361512 4.03361512] bar
 normal stress:  [3.34116896 3.34116896] bar
 Coulomb stress:  [5.3700827 5.3700827] bar
```

case 3: n rows of stress tensors

In this case, both stress tensors and sampling points are read from the built-in files of _stress.txt and _samplingxy.txt.

In [10]:
```
import coulomb as cs
import numpy as np
stressfile='_stress.txt';
samplingfile='_samplingxy.txt'
coulombfile='temp_coulomb.txt'
stressabspath=cs.getabspath(stressfile)
samplingfileabspath=cs.getabspath(samplingfile)
data=cs.readtextfile(stressabspath,headerlines=1)
nrow=data.shape[0]
E=np.ones(nrow)
strike=90*E
dip=90*E
rake=0*E
friction=0.4*E
skempton=0.0*E
cs.onestepCFS(stressabspath,samplingfileabspath,coulombfile,bgeo=False,
              strike=strike,dip=dip,rake=rake,
              friction=friction,
              skempton=skempton)
```

```
temp_coulomb.txt is saved.
***************the lines with nan***************
[[-50.   0.  10.  nan  nan  nan]
 [ 50.   0.  10.  nan  nan  nan]]
************************************************
```



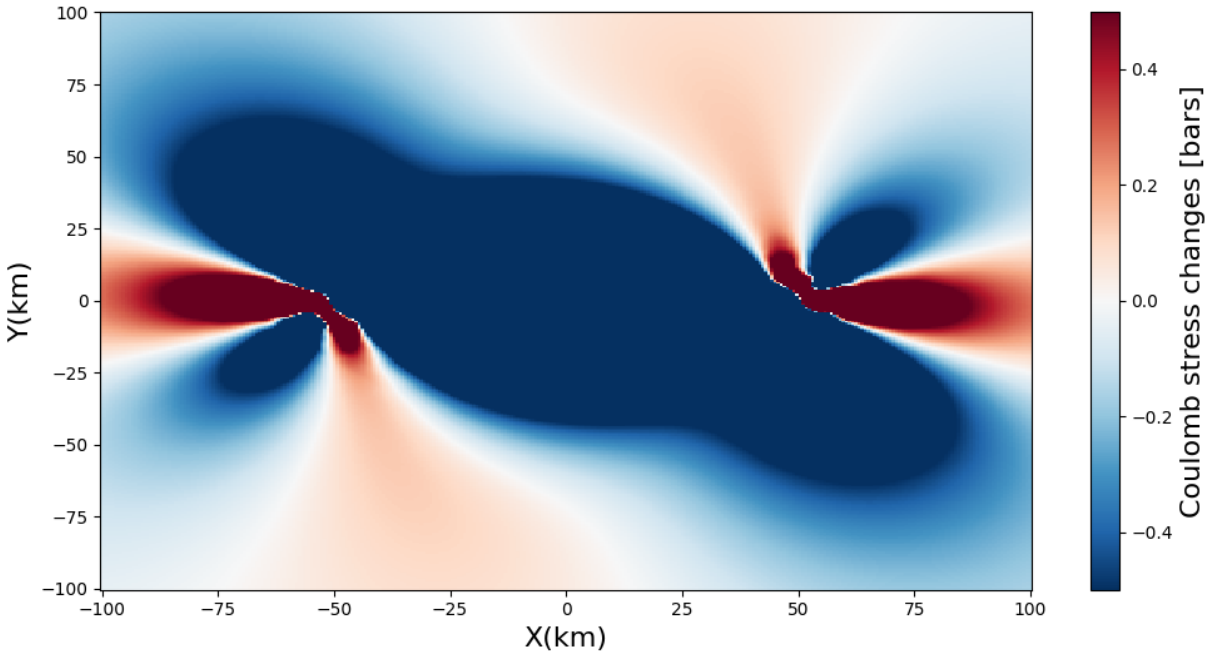
Set different receiver faults with a sampling file

In this case, stress tensors are read from the built-in file (_stress.txt), and the strike, dip and rake angles of a receiver fault, together with the friction and Skempton

coefficients, are also read from the built-in sampling file (_samplingvaryingpointsxy.txt).

In [11]:
```
import coulomb as cs
stressfile='_stress.txt';
samplingfile='_samplingvaryingpointsxy.txt'
coulombfile='temp_coulomb.txt'
stressabspath=cs.getabspath(stressfile)
samplingfileabspath=cs.getabspath(samplingfile)
cs.onestepCFS(stressabspath,samplingfileabspath,coulombfile,bgeo=False)
```

```
temp_coulomb.txt is saved.
***************the lines with nan***************
[[-50.   0.  10.  nan  nan  nan]
 [ 50.   0.  10.  nan  nan  nan]]
**************************************************
```



AutoCoulomb

The function 'AutoCoulomb' is used to compute Coulomb stress changes using rectangular dislocations. Stress tensors resulting from rectangular dislocations are calculated using the Okada's subroutine of DC3D.f. These stress tensors are then projected onto fixed receiver faults with constant strike, dip and rake angles or varying receiver faults with different strike, dip and rake angles. Accordingly, Coulomb stress changes can be resolved on fixed or varying receiver faults.

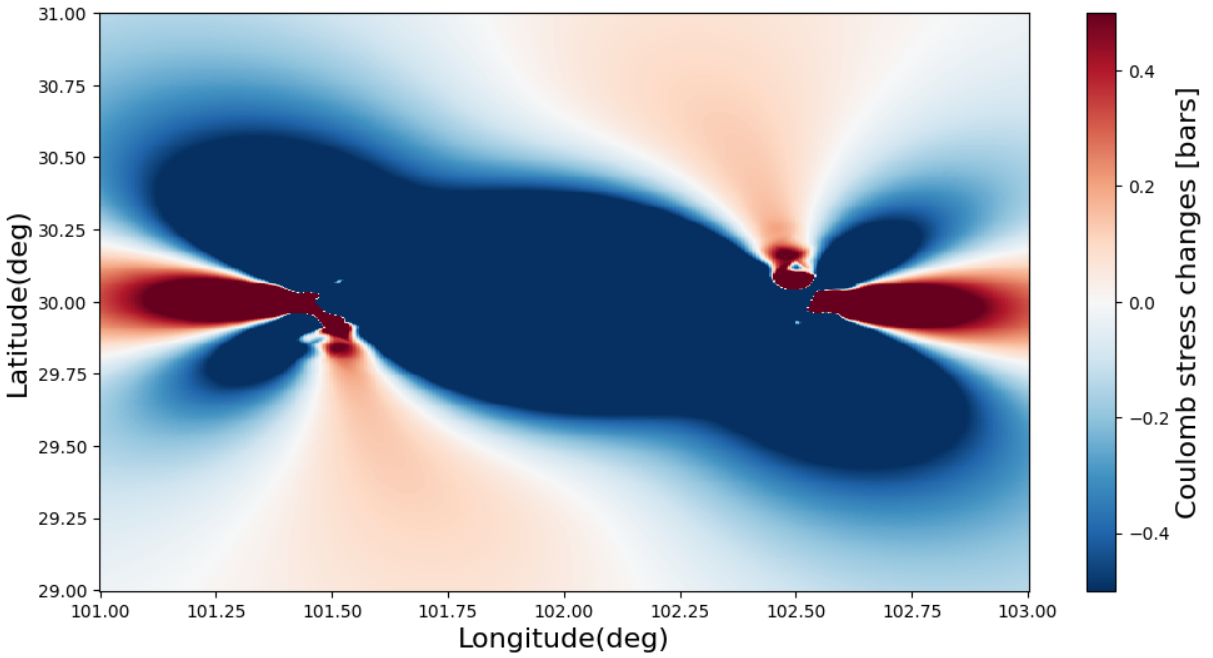
**AutoCoulomb(slipmodel,samplingfile,meridian=999,stressfile='stress.txt',coulombfile
friction=0.4,skempton=0.0,bgeo=True,ndenser=100,bdraw=True,cr=
[-1,1],ct=False,cmap='RdBu_r')**

case 1: Coulomb stress changes on fixed receiver faults in a geodetic coordinate system

- If both the reference points of the subpatches from a slip model and the sampling points from a sampling file are referred to a geodetic coordinate system, the keyword argument 'bgeo' should be True,i.e. bgeo=True (default).
- The source fault and the receiver fault are read from the built-in file (_slipmodel.txt), and the built-in sampling file (_sampling.txt).
- As a geodetic coordinate system is considered and the Gaussian projection is to be performed in this case, the longitude of a central meridian must be specified.

In [12]:
```
import coulomb as cs
slipmodel='_slipmodel.txt'
samplingfile='_sampling.txt'
coulombfile='temp_coulomb.txt'
cs.AutoCoulomb(slipmodel,samplingfile,coulombfile=coulombfile,meridian=102,
               strike=90,dip=90,rake=0,friction=0.4,skempton=0.0,
               ndenser=500,cr=[-0.5,0.5],ct=False)
```

```
processing...
----it's done!----
```



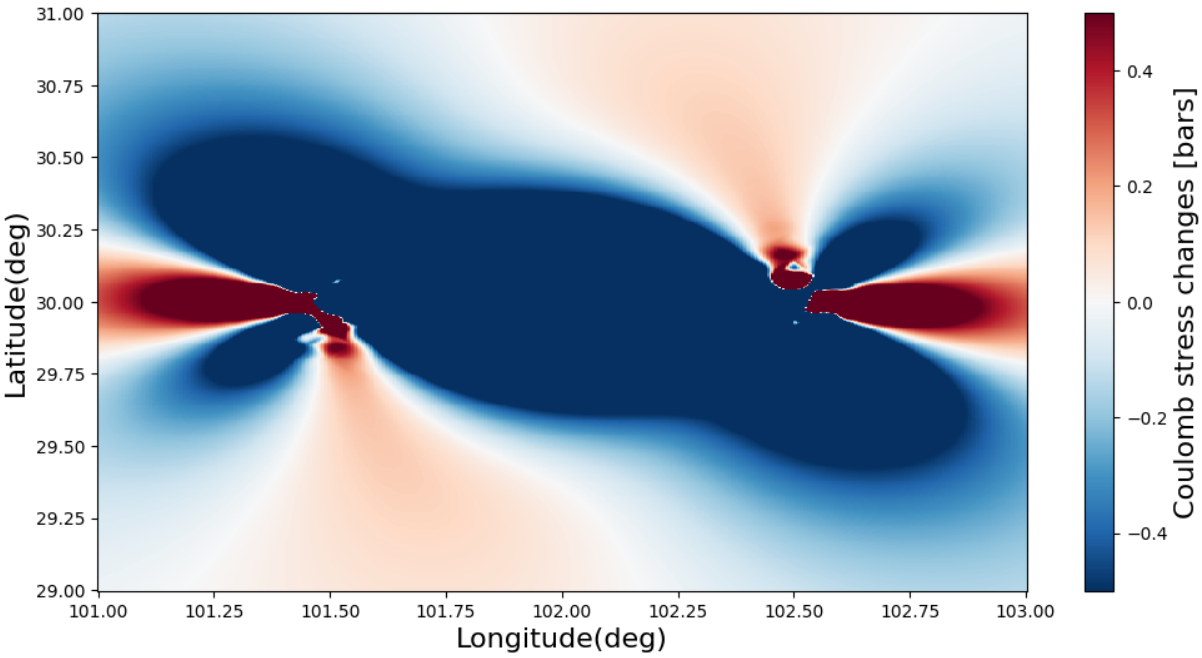
case 2: Coulomb stress changes on varying receiver faults in a geodetic coordinate system

- If both the reference points of the subpatches from a slip model and the sampling points from a sampling file are referred to a geodetic coordinate system, the keyword argument 'bgeo' should be True,i.e. bgeo=True (default).

- The source fault and the receiver fault are read from the built-in file (_slipmodel.txt), and the built-in sampling file (_samplingvaryingpoints.txt).
- As a geodetic coordinate system is considered and the Gaussian projection is to be performed in this case, the longitude of a central meridian must be specified.

In [13]:
```
import coulomb as cs
slipmodel='_slipmodel.txt'
samplingfile='_samplingvaryingpoints.txt'
coulombfile='temp_coulomb.txt'
cs.AutoCoulomb(slipmodel,samplingfile,coulombfile=coulombfile,meridian=102,
               ndenser=500,cr=[-0.5,0.5],ct=False)
```

```
processing...
----it's done!----
```



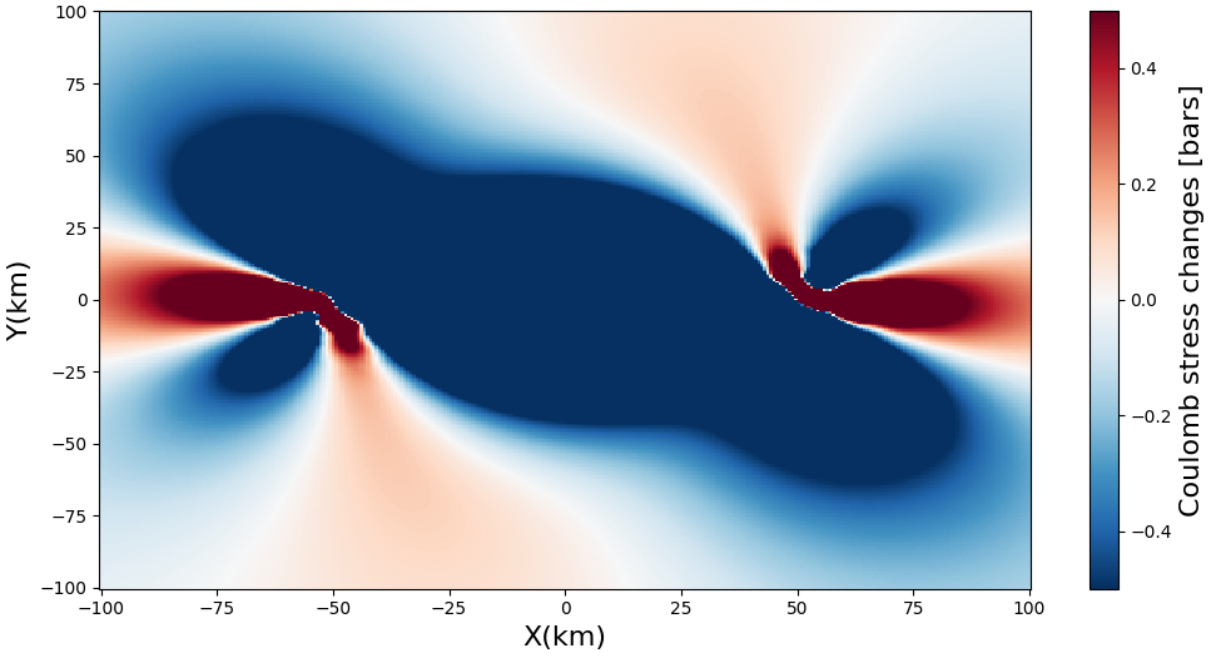
case 3: Coulomb stress changes on fixed receiver faults in a Cartesian coordinate system

- If both the reference points of the subpatches from a slip model and the sampling points from a sampling file are referred to a Cartesian coordinate system (say UTM), the keyword argument 'bgeo' should be False,i.e. bgeo=False.
- The source fault and the receiver fault are read from the built-in file (_slipmodelxy.txt), and the built-in sampling file (_samplingxy.txt).

In [14]:
```
import coulomb as cs
slipmodel='_slipmodelxy.txt'
samplingfile='_samplingxy.txt'
coulombfile='temp_coulomb.txt'
cs.AutoCoulomb(slipmodel,samplingfile,coulombfile=coulombfile,bgeo=False,
```

```
                strike=90,dip=90,rake=0,friction=0.4,skempton=0.0,
                ndenser=300,cr=[-0.5,0.5],ct=False)
```

```
processing...
----it's done!----
```

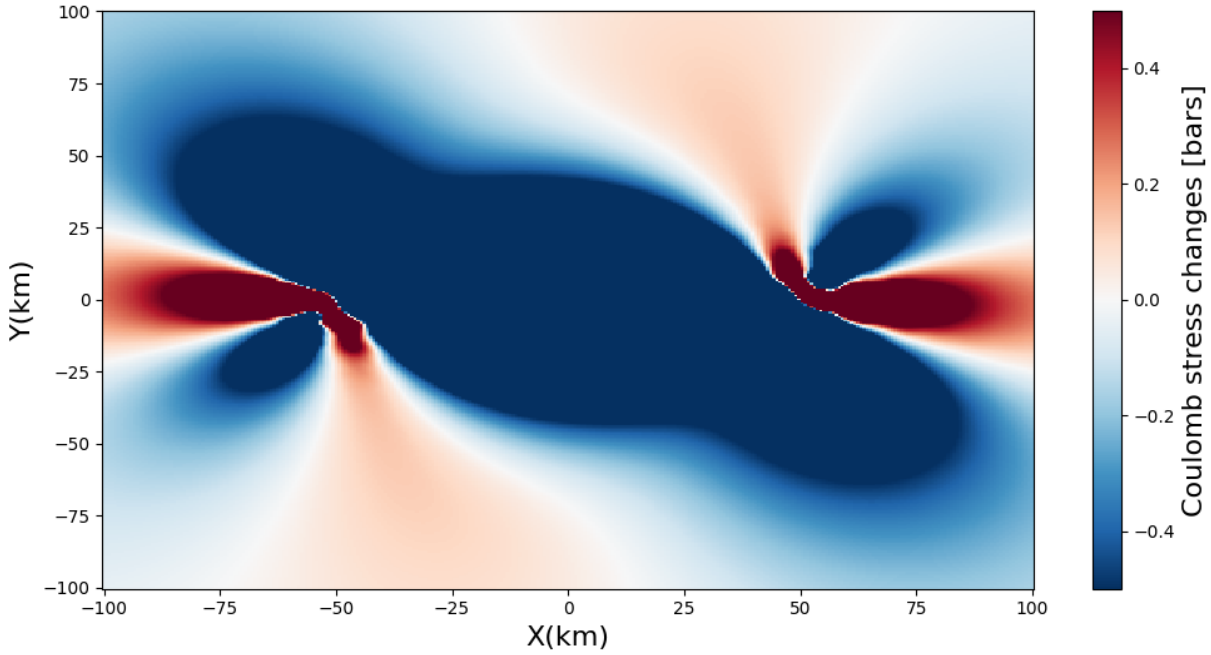


case 4: Coulomb stress changes on varying receiver faults in a Cartesian coordinate system

- If both the reference points of the subpatches from a slip model and the sampling points from a sampling file are referred to a Cartesian coordinate system (say UTM), the keyword argument 'bgeo' should be False,i.e. bgeo=False.
- The source fault and the receiver fault are read from the built-in file (_slipmodelxy.txt), and the built-in sampling file (_samplingvaryingpointsxy.txt), respectively.

In [15]:
```
import coulomb as cs
slipmodel='_slipmodelxy.txt'
samplingfile='_samplingvaryingpointsxy.txt'
coulombfile='temp_coulomb.txt'
cs.AutoCoulomb(slipmodel,samplingfile,coulombfile=coulombfile,bgeo=False,
               ndenser=300,cr=[-0.5,0.5],ct=False)
```

```
processing...
----it's done!----
```



samplingprofile

The 'samplingprofile' function is used to generate a profile with sampling points. After such a file with sampling points on a profile is created, Coulomb stress changes at these sampling points can be calculated using the same scheme as adopted in the case of computing Coulomb stress changes at grids on a horizontal plane with a given depth or on curved surface with varying depths.

samplingprofile(profilename,nL=50,nW=50,extendLW=[0,1,0,1],aftershockfilename="",samplingprofilename="sampling.txt",bdraw=True,figsize='8/6')

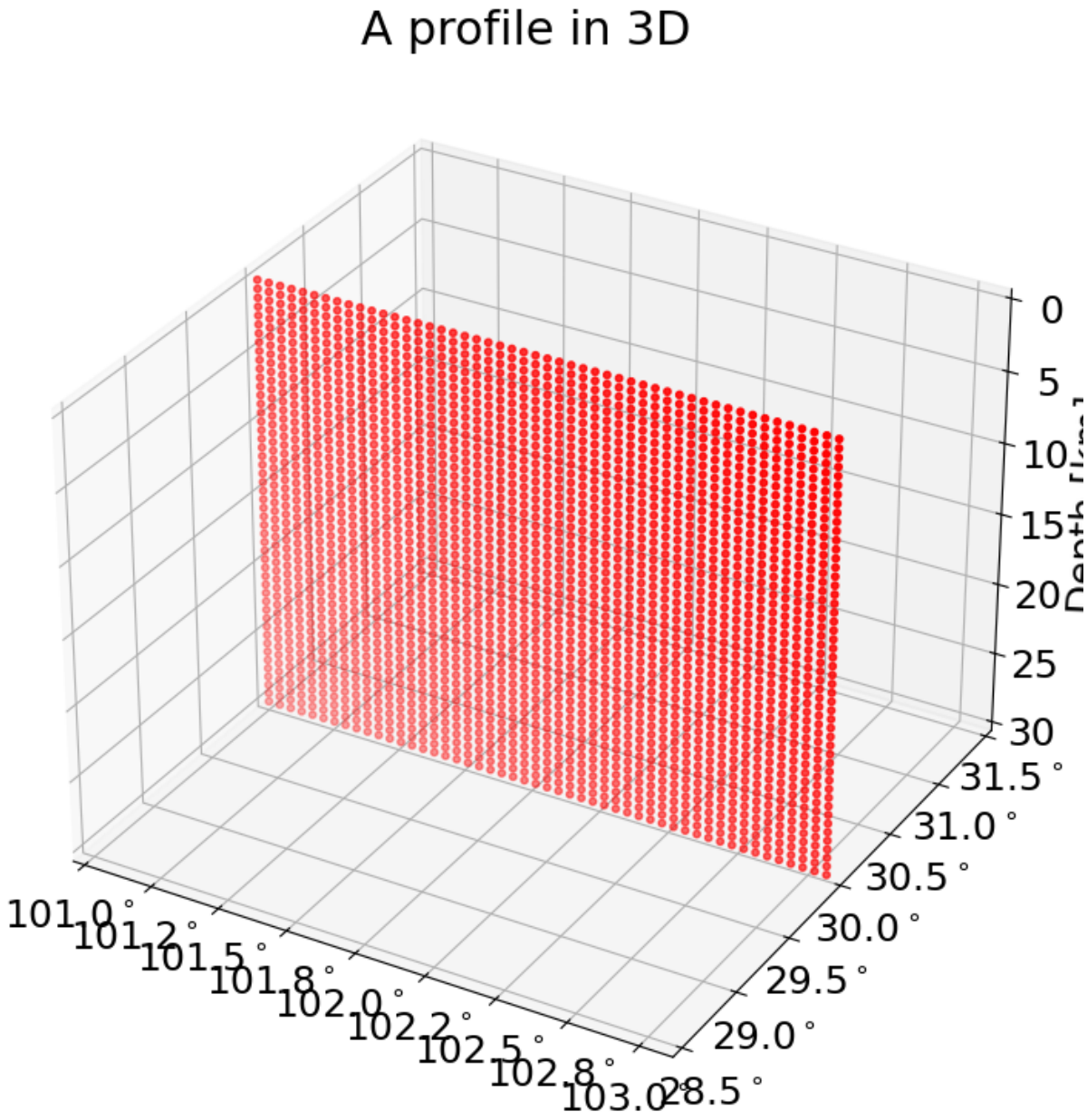
- **Step1: generate sampling points on the profile**

> In this case, the profile model akin to the slip model of a source fault (e.g. _slipmodel.txt) is read from the built-in file (_profile.txt) for generating sampling points on the profile. These sampling points are to be used for computing Coulomb stress changes. When to cacluate Coulomb stress changes, the file with a filename containing the substring of 'grids' is chosen to be the very sampling file for the computation. In addition, the aftershocks are also read from the built-in file (_aftershocks.txt) file. But if not to project aftershocks nearby the profile, it's not necessary to set the argument of 'aftershockfilename' with a value but rather let its value be empty.

In [16]:
```
import coulomb as cs
```

```
file_profile=cs.getabspath('_profile.txt')
print('file_profile=',file_profile)
file_aftershock=cs.getabspath('_aftershocks.txt')
#cs.samplingprofile(file_profile,figsize='12/9',
#aftershockfilename=file_aftershock)
cs.samplingprofile(file_profile,figsize='12/9')
```

```
file_profile= /Users/mac/Downloads/coulomb_dir/coulomb-master/coulomb/data/_p
rofile.txt
/Users/mac/Downloads/coulomb_dir/sampling1.txt is created.
the following files are generated:
./sampling1_profile.txt
./sampling1_profile_grids.txt
split_profile= ./sampling1_profile.txt aftershock= None
```



- **Step 2: Cacluate Coulomb stress changes on the profile**

> - The slip model of a source fault is read from the built-in file (_slipmodel.txt).
> - The sampling file of a receiver fault is generated at the first step.
> - Note the argument of bdraw is set to False for not drawing a Coulomb stress map, because only the generated Coulomb stress file is needed at this step.

In [17…

```
import coulomb as cs
slipmodel='_slipmodel.txt'
samplingfile='sampling1_profile_grids.txt'
coulombfile='temp_coulomb.txt'
cs.AutoCoulomb(slipmodel,samplingfile,coulombfile=coulombfile,
               meridian=102,strike=90,dip=90,rake=0,
               friction=0.4,skempton=0.0,
               bdraw=False)
```

```
processing...
----it's done!----
```

- **Step 3: Combine the sampling file and the Coulomb stress file, and then draw a Coulomb stress map with Coulomb stress changes on the profile**

- 'sampling1_profile.txt' is generated at the first step.
- 'temp_coulomb.txt' is generated at the second step.
- Note the argument of bgeo is set to False, because Coulomb stress changes on the profile are exhibited and the x axis of such a figure is along strike and the y axis is along downdip with a unit of km. In addition, the argument of baxis being False makes the y axis reversed.

In [18…

```
import coulomb as cs
import os
os.system('combined_profilegrids_profileCFS.sh -G sampling1_profile.txt \
          -C temp_coulomb.txt -O combinedCFS.txt' )
data=cs.readtextfile('combinedCFS.txt',headerlines=1)
cs.drawCFS(data[...,0],data[...,1],data[...,4],
           bgeo=False,baxis=False,ndenser=50,
           cr=[-1,1],ct=False,cmap='RdBu_r')
```

```
combinedCFS.txt is generated.
```