UNIVERSITY OF SUSSEX
Scientific Computing
Tutor: Dr. Ilian Iliev, Office: Pev 3 4A5

Problem Sheet 5

1. **Root finding:** Consider the equation $\sin(x) + 3\cos(x) - 2 = 0$ in the interval $(-2, 2)$.

   (a) Find the number and approximate bracketing values of the roots of that equation in this interval, using e.g. plotting.

   (b) Use Ridder's method and the results in (a) to find the the roots with nine significant digits.

   (c) Use Newton-Raphson's method and the results in (a) to find the the roots with nine significant digits.

   (d) Do the same using the in-build function `scipy.optimize.fsolve`.

   Solution:

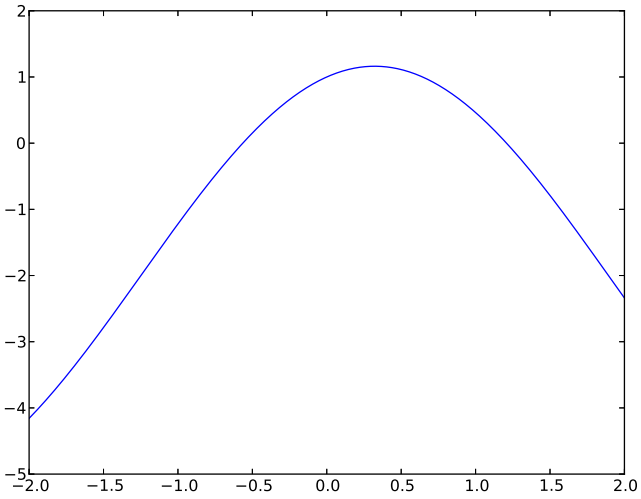
   (a) After importing the required modules, we first set up the vectors for plotting and then produce the plot in the standard way:

```
import numpy as np
import pylab as pl
import scipy as sci
from scipy import optimize
from ridder import *
from newtonRaphson import *

x=np.linspace(-2,2,1000)
y=np.sin(x)+3*np.cos(x)-2
pl.plot(x,y)

pl.show()
```

   This produces the plot in Figure 1.
   Based on the plot we expect 2 roots, one in the interval $(-1, 0)$, and one in $(1, 1.5)$.
   Next, we set up the functions corresponding to the equation to be solved and its derivative (needed for the Newton-Raphson method below), as follows:




```
def f(z):
    return np.sin(z)+3.*np.cos(z)-2.

def df(z):
    return np.cos(z)-3.*np.sin(z)
```

(b) Given the bracketing intervals we found in (a), we call Ridder's solver as:

```
root1 = ridder(f,-1.,0.)
root2 = ridder(f,1.,1.5)

print 'Ridder method results ',root1,root2
```

This yields:

```
Ridder method results  -0.564326569397 1.20782767819
```

for the two roots. We note that the default precision of the code (1e-9) is exactly what is needed here, so we do not need to give it a new value.

(c) Doing the same as above, but for the Newton-Raphson method:

```
root1 = newtonRaphson(f,df,-1.,0.)
root2 = newtonRaphson(f,df,1.,1.5)
```

(note that, as required by the Newton-Raphson method, we now need to provide the derivative, as well). We find:

```
Newton-Raphson method results  -0.564326569397 1.20782767819
```

(d) Finally, applying the in-build routine for equation-solving, we do:

```
a=sci.optimize.fsolve(f,-1.)
b=sci.optimize.fsolve(f,1.)
print 'fsolve results ',a,b
```

(note that fsolve only requires a rough estimate of each root as second argument). This yields:

```
fsolve results  [-0.56432657] [ 1.20782768]
```

Clearly, all three methods give the same values for the three roots within the required precision.

2. **Systems of equations:** The trajectory of a satellite orbiting the Earth is:

$$R = \frac{C}{1 + e\sin(\theta + \alpha)} \tag{1}$$

where (R, θ) are the polar coordinates of the satellite, and C, e, and α are constants (e is known as the eccentricity of the orbit, $C = a(1-e^2)$, where a is the orbit's semi-major axis, and α is the phase). If the satellite was observed at the three positions listed in the table, determine the smallest R of the

θ	-30°	0°	30°
R (km)	6870	6728	6615

trajectory (this is called pericentre of the orbit) and the corresponding value of θ. To solve the equations use both the provided code `newtonRaphson2.py` and the in-build routine `scipy.optimize.fsolve`.

Solution: After importing the required modules, we first set up our equations as a vector-valued function, using the values for R and θ in the table:

```
import numpy as np
import pylab as pl
import scipy as sci
from scipy import optimize
from newtonRaphson2 import *

def f(z):
    return np.array([6870.-z[0]/(1+z[1]*np.sin(z[2]-np.pi/6.)),
6728.-z[0]/(1+z[1]*np.sin(z[2])),6615.-z[0]/(1+z[1]*np.sin(z[2]+np.pi/6.))])
```

Note that in the equations we converted all angles to radians, as this is the standard unit for angles.

We also need some rough starting estimate for the three parameters:

```
x0=np.array([6800.,0.5,0.])
```

Finally, we call the Newton-Raphson and the in-build routines:

```
a=newtonRaphson2(f,x0)

print 'Newton-Raphson solution is: C=',a[0],' e=',a[1],' alpha=',a[2]

b=sci.optimize.fsolve(f,x0)

print 'fsolve solution is: C=',b[0],' e=',b[1],' alpha=',b[2]
```

Both return the three roots of the system, which correspond to the values of the parameters we are seeking.

This yields the solutions:

```
Newton-Raphson solution is: C= 6819.29379321  e= 0.0405989590577
        alpha= 0.34078399795
fsolve solution is: C= 6819.29379321  e= 0.0405989590577
        alpha= 0.340783997955
```

Clearly, both codes find the same roots within the requested precision.

Finally, we can find the orbit radius at the pericentre by noting that this corresponds to the point where the sine function equals 1, so $R_{min} = C/(1+e)$, reached at angle $\theta = \pi/2 - \alpha$:

```
print 'Pericentre is at R=',b[0]/(1.+b[1]),' at angle ',-a[2]+np.pi/2.,
          ' radians'
```

which gives:

```
Pericentre is at R= 6553.23910701  at angle  1.23001232884  radians
```