PHY 410
Homework Assignment 4

Han Wen

Person No. 50096432

September 29, 2014

Abstract

The goal of this assignment was to develop a thorough understanding of Integral and root finding as well as the classical scattering problem

Contents

1 Problem 1

1.1 Description

Compare the accuracies and efficiencies of any two quadrature algorithms on the definite integral:

$$\int_0^1 e^x \mathrm{d}s = e - 1$$

Compare the accuracies and convergence rates of any two root-finding algorithms on the functions:

$$f(x) = tan(x), f(x) = tanh(x)$$

1.2 Solution

For part one, I used both Trapezoidal rule and Simpson rule to do the integral. And generated a plot of accuracies vs. number of the steps used. 1 Apparently using simpson gives much higher accuracy. Considering they both used same number of summations during the procedure. Therefore the simpson is more effective.

For part two. I used both simple search and root tangent method. First from basic knowledge we know for tan function periodic roots occur while for tanh the only root is the origin point. For simplicity I am going to test only the origin point root.
tan:
Simple search:
initial guess:-1
initial step: 0.3
accuracy: 0.0000001.
It takes 35 steps to find the root, and the function value is 4.77×10^{-8}.
Root tangent:
initial guess:-1
accuracy:0.0000001
It takes 4 steps to find the root, the function value is -2.32×10^{-10}

tanh:
Simple search:
initial guess:-1
initial step: 0.3
accuracy: 0.0000001
It takes 35 steps to find the root, and the function value is 4.77×10^{-8}
Root tangent:
initial guess:-1
accuracy:0.0000001
It takes 5 steps to find the root, the function value is 2.34×10^{-13}

Therefore we can see the root tangent method gives faster convergence rates and better accuracy.

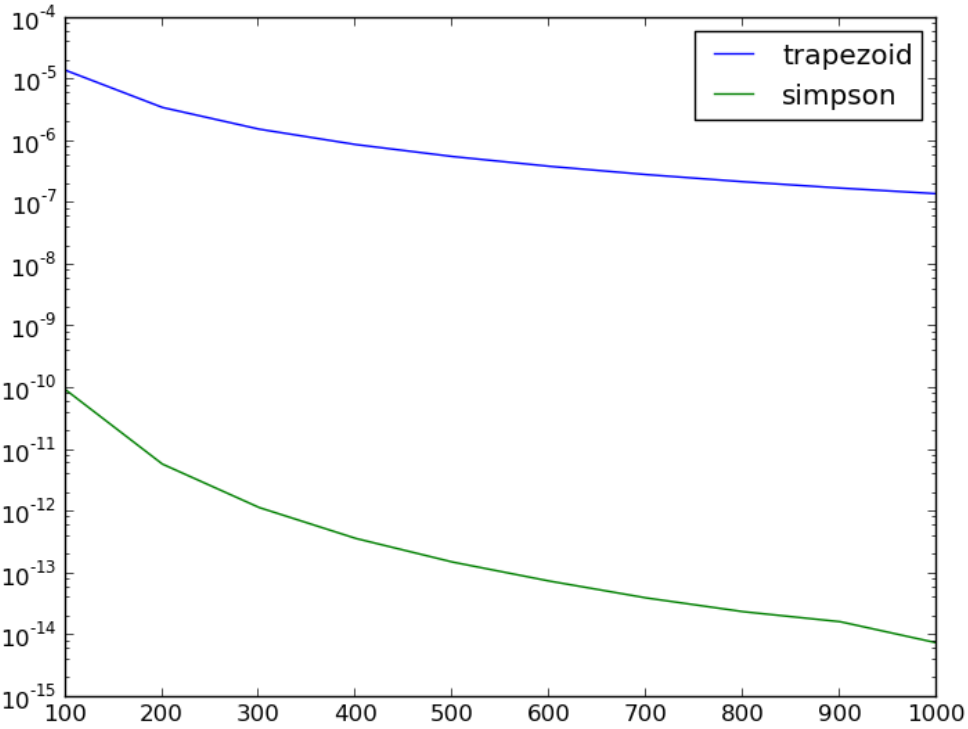

Figure 1: accuracies vs. number of the steps used, problem 1

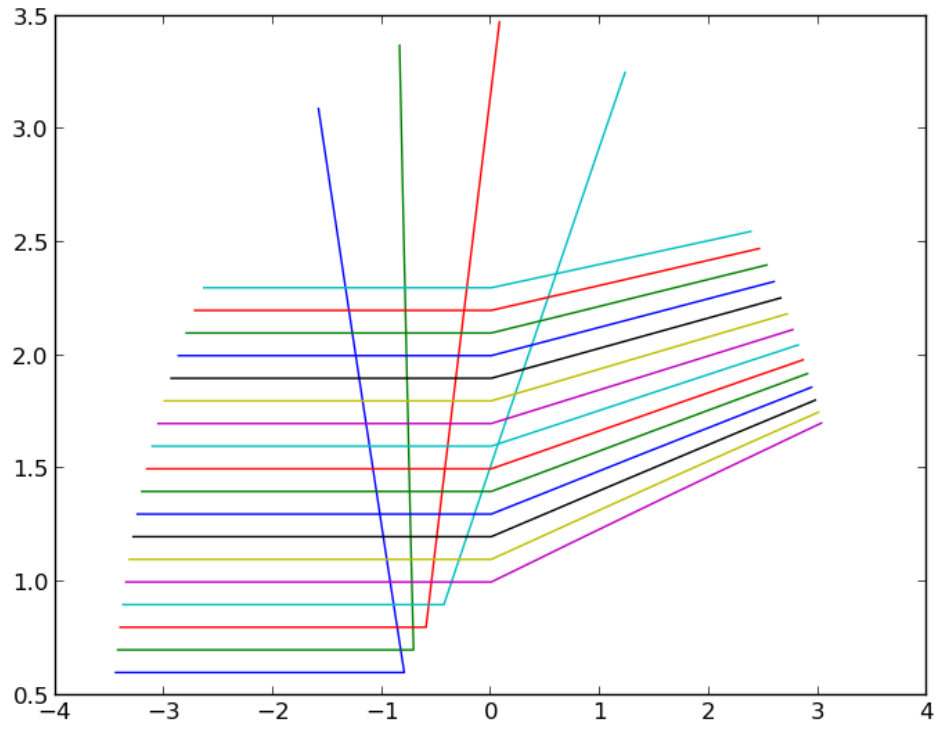

Figure 2: Hard core trajectories with various d, E=0.705

2 Problem 2

2.1 Description

Using the tools from Lecture 12, study classical scattering from a hard sphere and from the Lennard-Jones potential. As discussed in Lecture 12, plot typical trajectories, and compute and plot the differential scattering cross section for the Lennard-Jones case (as per the procedure from class, Slide 16 in Lecture 12). Do this for two or three different values of the energy that you find most interesting. Study the phenomenon of orbiting in the Lennard-Jones case: what is the maximum number of orbits you can generate by carefully tuning the energy and impact parameter?

2.2 Result

Here is the plot of the trajectories of hard-core potential for various d with E=0.705 2.
Here is the plot of the trajectories of Lennard-Jones potential for various d with E=0.705 3.
Here is the plot of differential cross section vs. d for Lennard-Jones potential with E=0.705 4.

Here is the plot of the trajectories of hard-core potential for various d with E=1.705 5.
Here is the plot of the trajectories of Lennard-Jones potential for various d with E=1.705 6.
Here is the plot of differential cross section vs. d for Lennard-Jones potential with E=1.705 7.

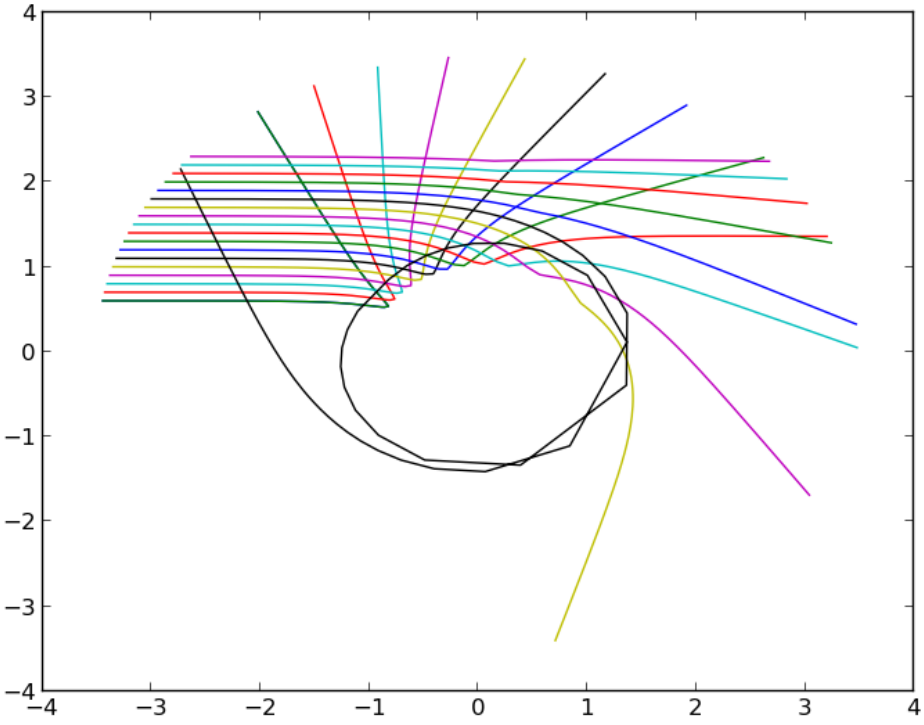

Figure 3: Lennard-Jones trajectories with various d, E=0.705

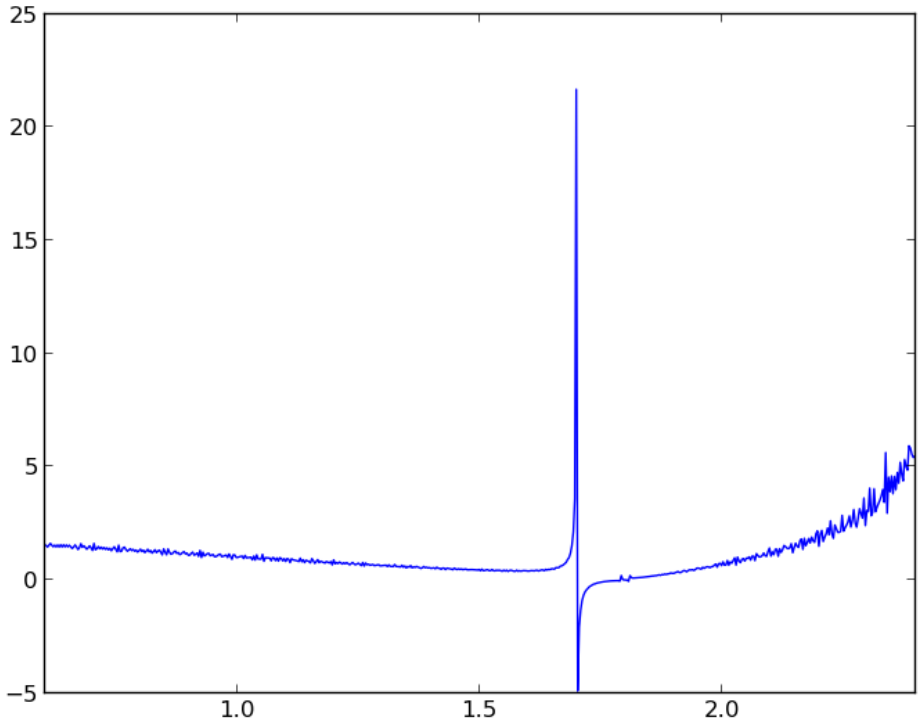

Figure 4: Lennard-Jones differential cross section vs. d, E=0.705

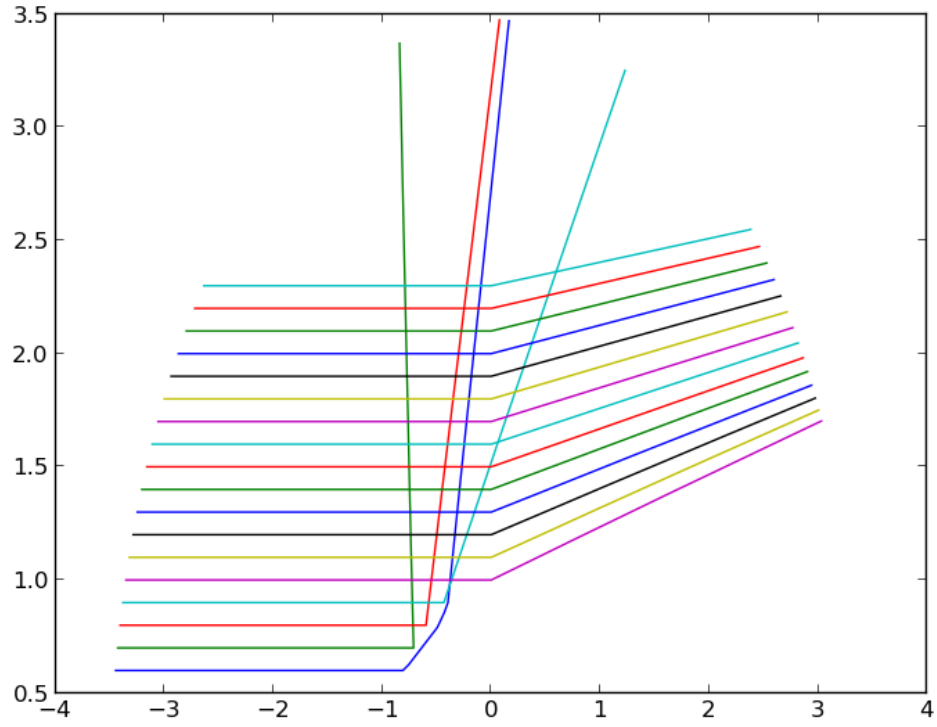

Figure 5: Hard core trajectories with various d, E=1.705

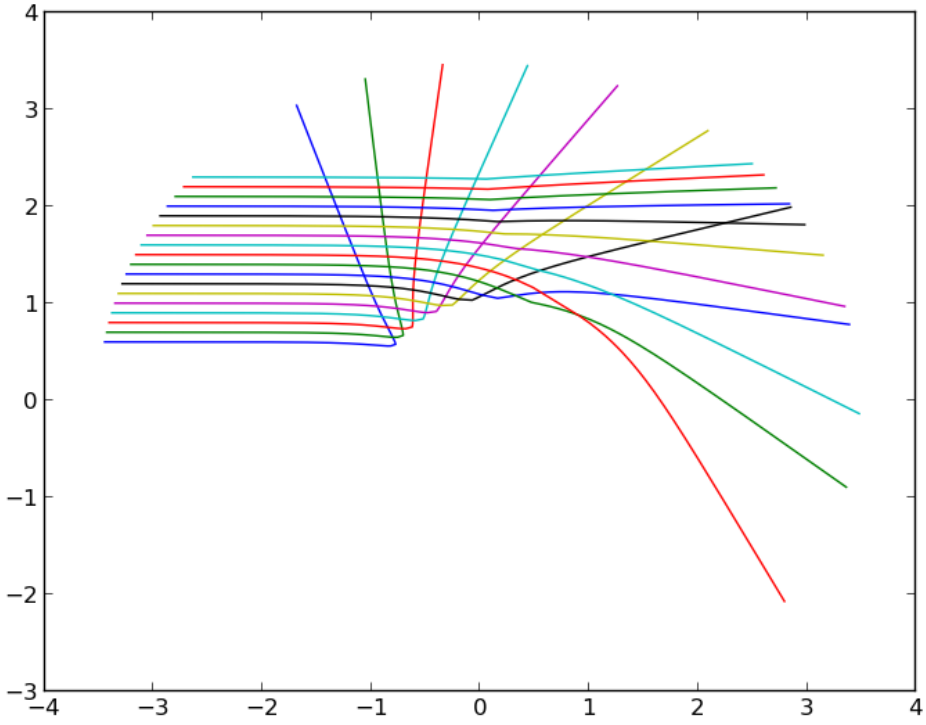

Figure 6: Lennard-Jones trajectories with various d, E=1.705

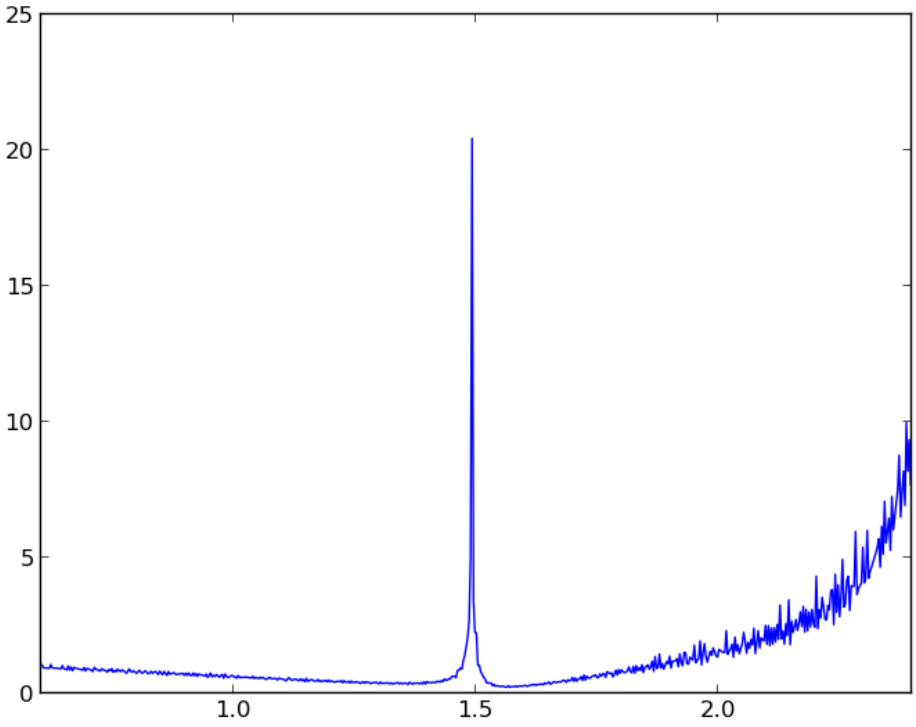

Figure 7: Lennard-Jones differential cross section vs. d, E=1.705

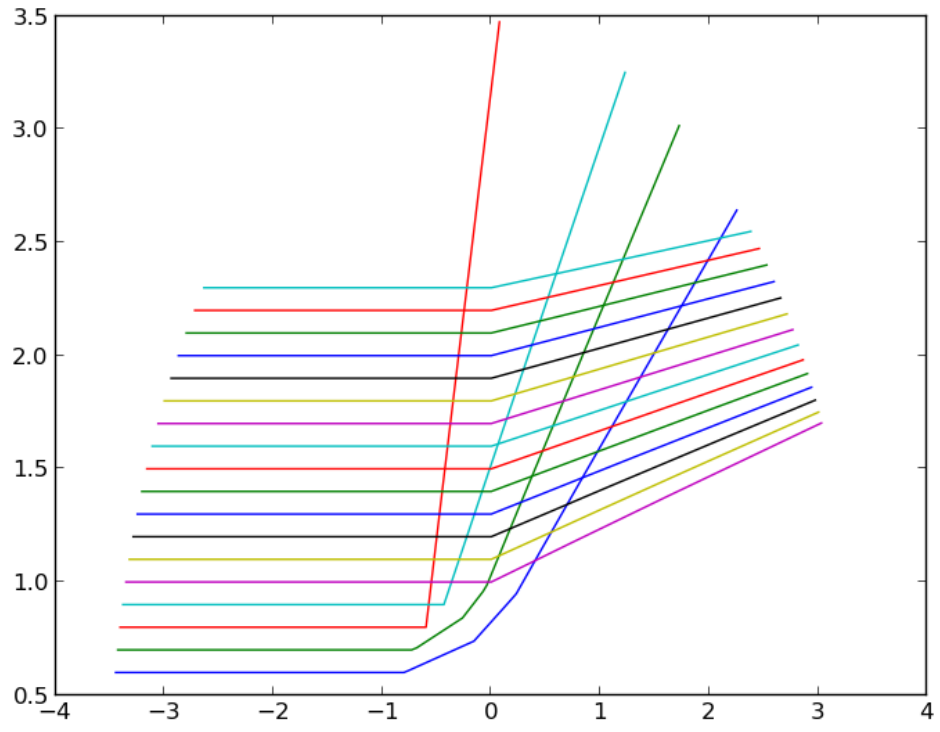

Figure 8: Hard core trajectories with various d, E=2.705

Here is the plot of the trajectories of hard-core potential for various d with E=2.705 8.
Here is the plot of the trajectories of Lennard-Jones potential for various d with E=2.705 9.
Here is the plot of differential cross section vs. d for Lennard-Jones potential with E=2.705 10.

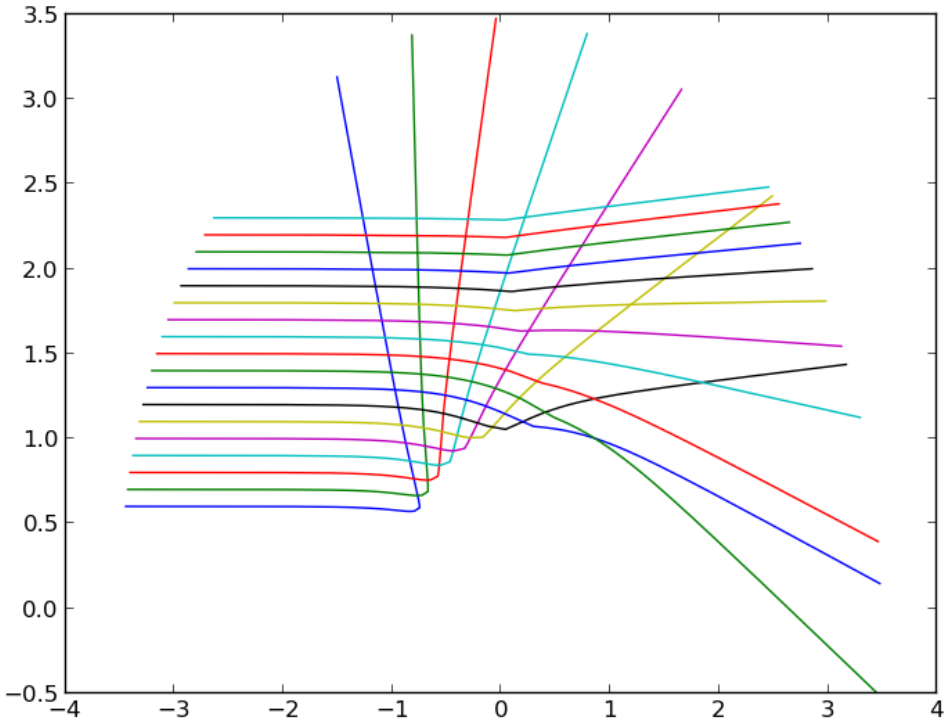

Figure 9: Lennard-Jones trajectories with various d, E=2.705

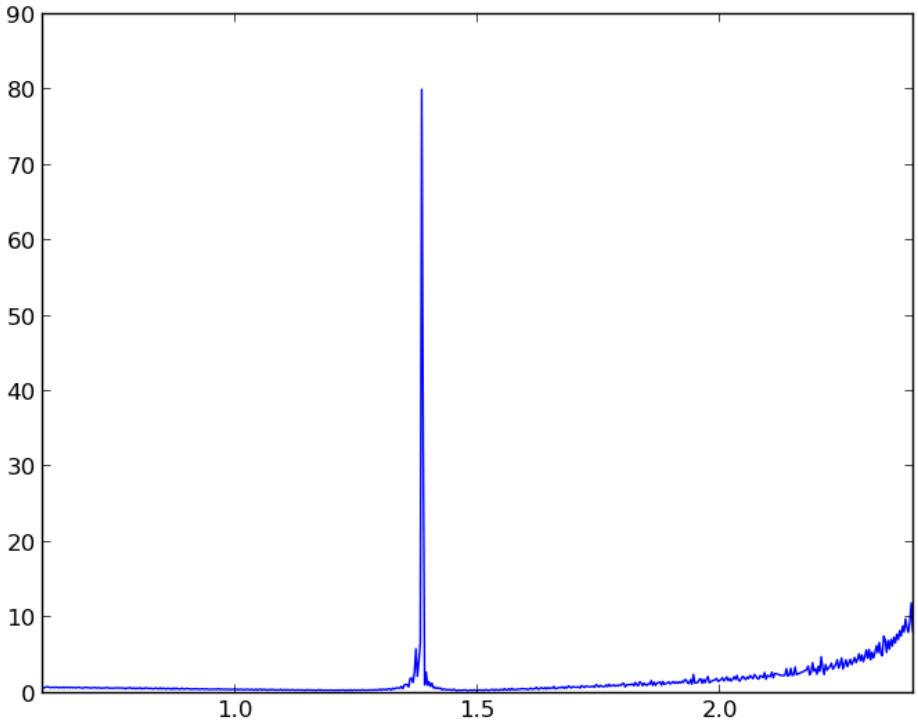

Figure 10: Lennard-Jones differential cross section vs. d, E=2.705

3 Problem 3

3.1 Description

(a) Show that the tangent root-finding algorithm is unstable for starting bigger than a critical value in Problem 1, and find this value. (b) Modify the scattering program to plot trajectories and differential cross sections for the screened Coulomb or Yukawa potential

$$V(r) = V_0 \frac{e^{-r/r_0}}{r}$$

Compare your results in the limit of large r_0 with analytic formulas for Rutherford scattering (as derived in class).

3.2 Result

(a) From the algorithm we can see the reason of existence of the unstable guess is that some value will make the first derivative of the function approach to 0, especially when it thus will cause the value of the first derivative in the next step more approach to 0. In our case, the first function is fine, there will be no so called critical value. Yet for the hyperbolic tangent. We can see for x goes to infinity, the value of tanh(x) goes to plus or minus 1, resulting in its first derivative approaching 0. With an accuracy of 0.0000001, the critical value is about 1.08.
(b)Using the Yukawa potential with E=0.705 and $r_0 = 1.0$, I plot the trajectories of various d: 11, and differential cross section vs. d 12

When r_0 is becoming very large, from the analytical point of vies, the Yukawa potential scattering will general approach Rutherford scattering. However here is a little comments about what we discussed in the afternoon. When talking about the big r_0 behaviour you quoted wikipedia `http://en.wikipedia.org/wiki/Rutherford_scattering`. However, you mistook the b below as the b in the equation, while in fact that b is the nearest location, not related to scattering. The real d is a function of Θ, and thus the $\frac{d\sigma}{d\Omega}$ is not purely a function of d^2 unless when d is quite big where the Θ does not change too much with d, at which time approximately the square rule can apply. And my simulation verified my theory: Here is the diagram when d starts from a rather big value with $r_0 = 10$: 13

Here is the diagram when d starts from a small value: 14

Here is the diagram when d starts from a small value but r_0 very large: 15

Here is the diagram of Rutherford: 16

From above we can see for large r_0 Yukawa scattering behaves like Rutherford, with r_0 increases, the more it approaches. Additionally when d start from a big value the curve behaviour approaches square law.

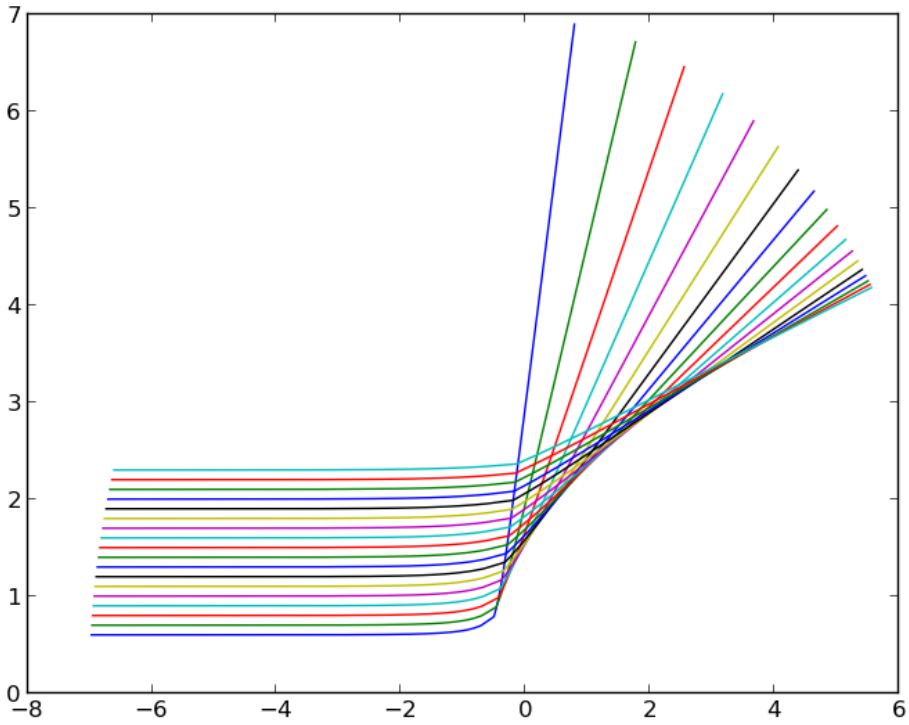

Figure 11: Yukawa trajectories with various d, E=0.705

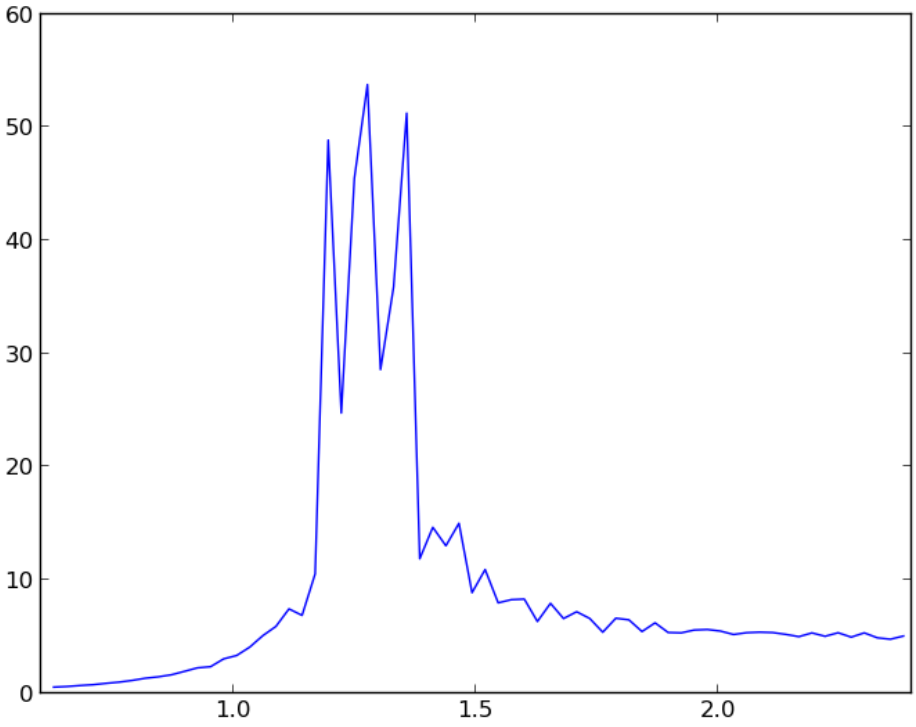

Figure 12: Yukawa differential cross section vs. d, E=0.705

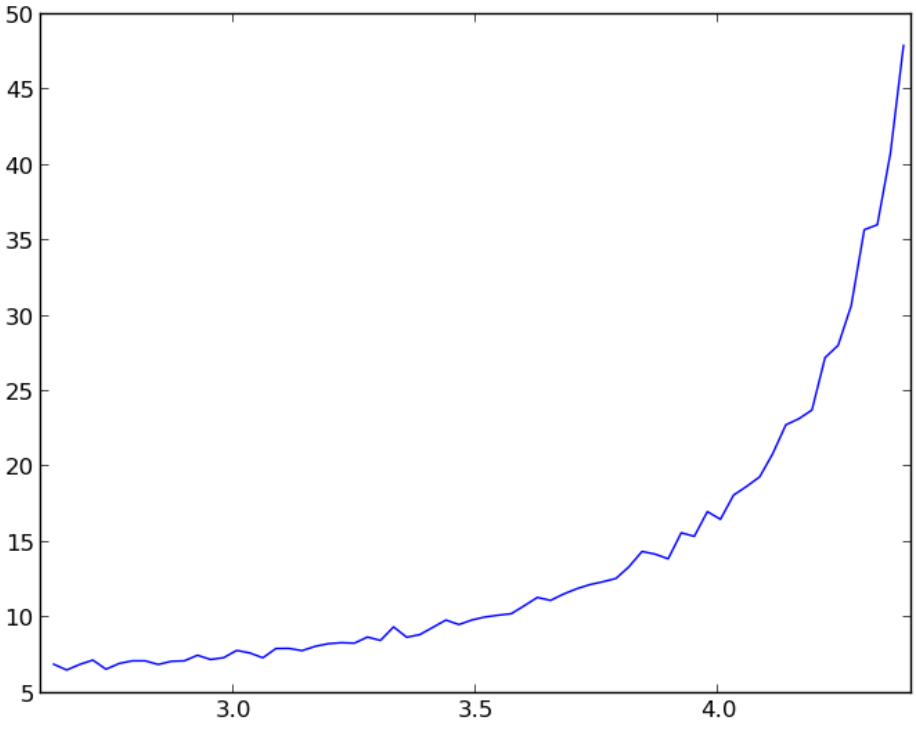

Figure 13: Yukawa differential cross section vs. d, E=0.705,d big

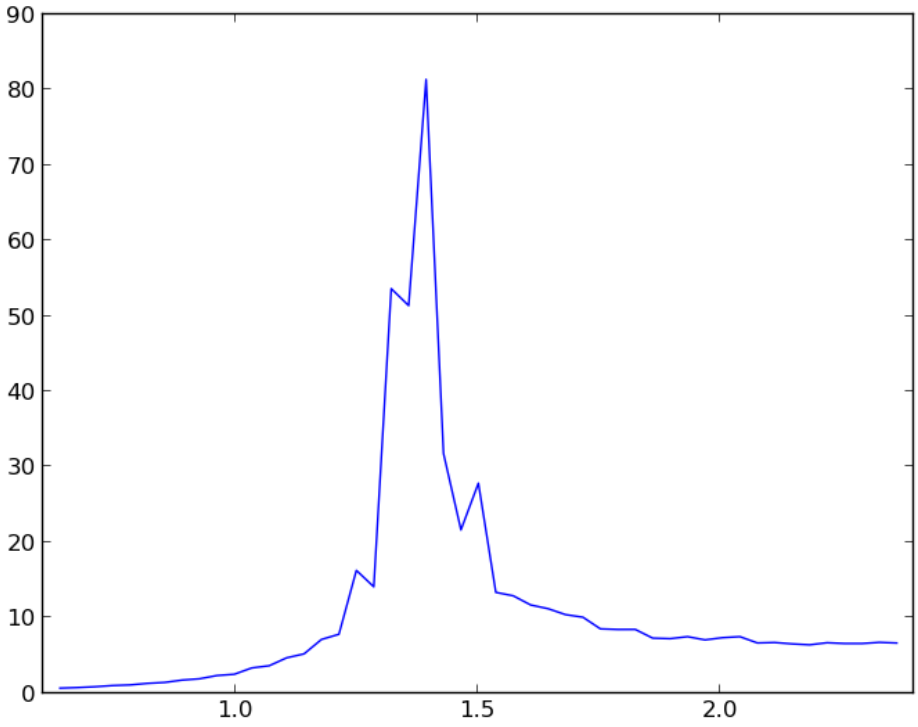

Figure 14: Yukawa differential cross section vs. d, E=0.705,d small

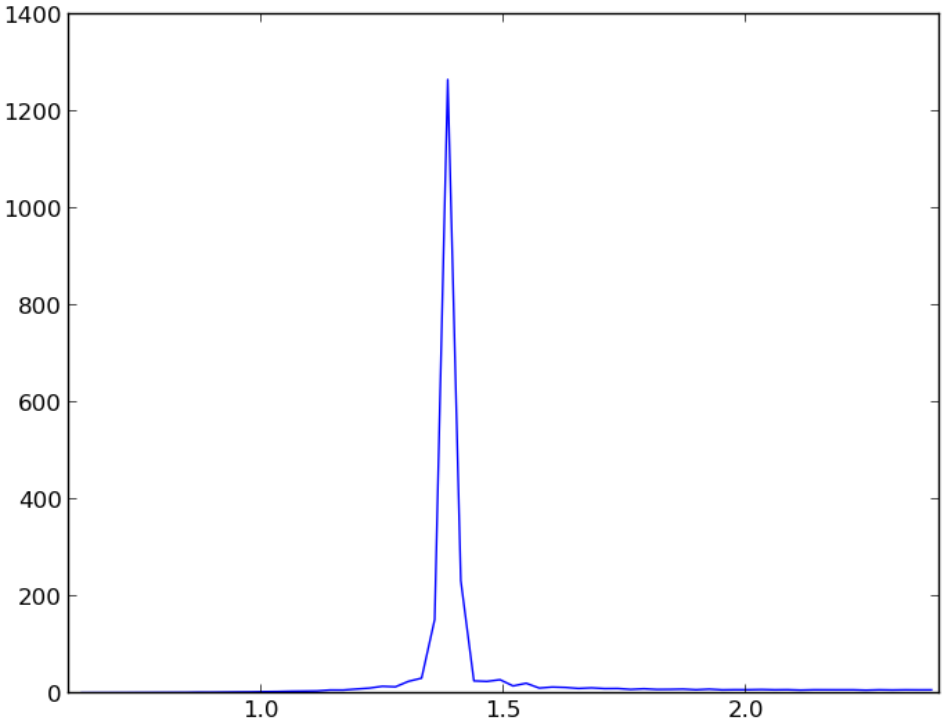

Figure 15: Yukawa differential cross section vs. d, E=0.705,d small, r_0 big

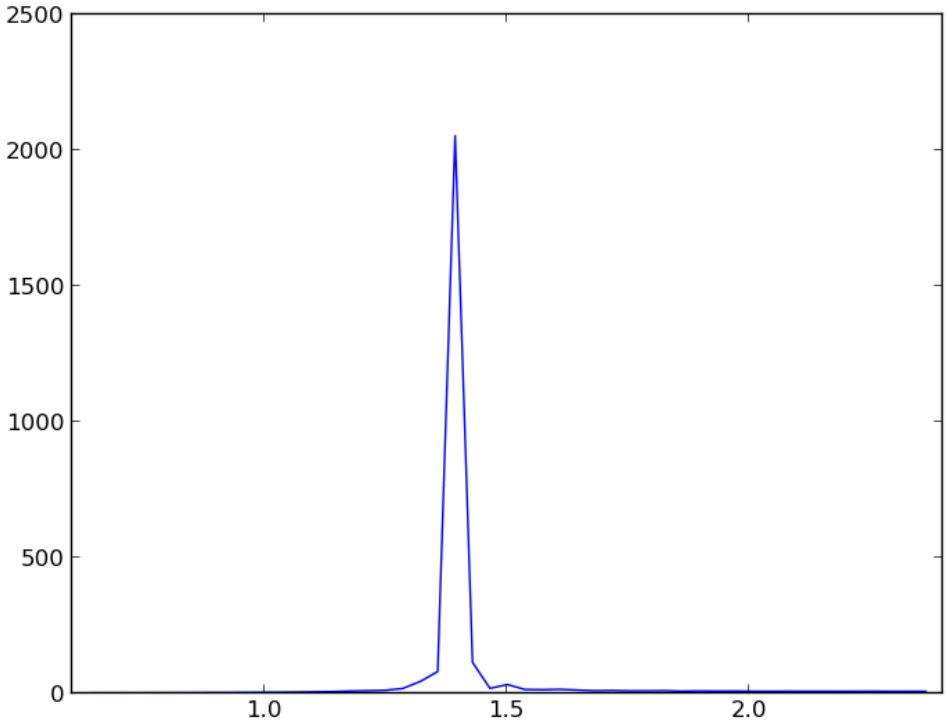

Figure 16: Rutherford

Acknowledgements

I discussed this assignment with my classmates and used material from the cited references, but this writeup is my own.

A Appendix

A.1 python code

The following python code was used to obtain the results in this report:

```
from root_finding import *
from math import *

def fp1 ( x ) :
    return (tan(x))**2+1.0
def fp2 ( x ) :
    return 1.0-(tanh(x))**2



print(" Algorithms for root of tangent")
print(" ----------------------------------------------------------")
print(" 1. Simple search")
x0 = float ( input(" Enter initial guess x_0 : ") )
dx = float ( input(" Enter step dx : ") )
acc = float ( input(" Enter accuracy : ") )
answer = root_simple(tan, x0, dx, acc,1000,True)
print  str ( answer ) + "\n\n"

print(" 2. root tangent")
x0 = float ( input(" Enter initial guess x_0 : ") )
acc = float ( input(" Enter accuracy : ") )
answer = root_tangent(tan, fp1, x0, acc,1000,True)
print  str ( answer ) + "\n\n"


print(" Algorithms for root of tanh")
print(" ----------------------------------------------------------")

print(" 1. Simple search")
x0 = float ( input(" Enter initial guess x_0 : ") )
dx = float ( input(" Enter step dx : ") )
acc = float ( input(" Enter accuracy : ") )
answer = root_simple(tanh, x0, dx, acc,1000,True)
print  str ( answer ) + "\n\n"

print(" 2. root tangent")
x0 = float ( input(" Enter initial guess x_0 : ") )
acc = float ( input(" Enter accuracy : ") )
answer = root_tangent(tanh, fp2, x0, acc,1000,True)
print  str ( answer ) + "\n\n"

from trapezoid import *
```

```
from math import *
from numpy import array
from root_finding import *

# Calculating the deflection angle by
# integrating the differential cross section
# of a scattering amplitude

#########################################################################
#  Important note : All of the distances are in units of the
#                   distance parameter of the potential!
#########################################################################

class lennard_jones :
    '''
    lennard_jones : class to implement the Lennard-Jones potential.
    Initialize with :
        * V0 : potential well depth

    Then execuate with __call__(r), which will return the Lennard-Jones
    potential at r. r must be in units of the potential's minimum, so
    the function is minimized at r = 1.0
    '''
    V0 = 0.0
    def __init__( self, V0) :
        self.V0 = V0

    # Note : r is in units of r0 of L-J potential
    def __call__( self, r ) :
        if r < 0.0 :
            print 'Error! Value for r must be positive!'
            exit(1)
        return 4 *  self.V0 * ( pow( r, -12) - pow(r,-6))

class hard_sphere_potential :
    '''
    hard_sphere_potential : class to implement the hard-scattering ("billiard ball
    Initialize with :
        * V0 : potential "inside" ball

    Then execuate with __call__(r), which will return V0 for r < 1.0, and 0 for r >
    Here, r must be in units of the potential's width.
    '''
    V0 = 0.0
    def __init__( self, V0 ) :
        self.V0 = V0

    def __call__( self, r ) :
```

```
        if r < 0.0 :
            print 'Error! Value for r must be positive!'
            exit(1)
        if r < 1.0 :
            return self.V0
        else :
            return 0

class Yukawa :
    '''
    Yukawa : class to implement the Yukawa potential.
    Initialize with :
        * V0 : potential well depth

    '''
    V0 = 0.0
    def __init__( self, V0, r0) :
        self.V0 = V0
        self.r0 = r0

    # Note : r is in units of r0 of L-J potential
    def __call__( self, r ) :
        if r < 0.0 :
            print 'Error! Value for r must be positive!'
            exit(1)
        return    self.V0 * (exp(-r/self.r0)/r)
        #return    self.V0 /r




class Theta :
    '''
    Theta : class to implement the total scattering amplitude of a particle scatte
            off of a potential V(r). Will utilize dTheta_dr, and integrate it from
            r = -inf to inf. This is done by first finding the root of f_r_min.

            This expects that r is in units of the natural width of the potential.
            So, for a Lennard-Jones potential, r is in units of r0. For a hard sca
            potential, it is in units of the width of the potential box.

    Initialize with :
         * V     : potential to utilize (i.e. an instance of hard_sphere_potential
                 : Must satisfy V(r)
         * E     : energy of incoming particle
         * b     : impact parameter of incoming particle
```

```
        * r_max : projection radius
        * steps : number of steps from r_min to r_max

    Then will compute the trajectory with "trajectory", which returns
    the total deflection ("deflection") along with the (r,theta)
    positions as a tuple of numpy "array" classes:
    [ array([r0,theta0]),
        array([r1,theta1]),
        array([r2,theta2]),
        ...
    ]

    To do this, utilizes :

    - f_r_min :
    implement the computation to find f(r_min), the
    distance at the point of closest approach (PCA).
    Note that r_min must be found numerically using a root-finding
    method of this function.

        Returns : 1 - (b/r)**2 - V(r)/E.


    - dTheta_dr :
    implement d(Theta) / d(r),
    the scattering amplitude of a particle scattering
    off of a potential V(r). Will utilize an instance of "f_r_min" to
    compute the distance at the point of closest approach (PCA).

        Returns : 1.0 / sqrt( abs(f_r_min(r)) ) / pow(r,2.0)


    '''
    V = None    # Must satisfy V.__call__(r)
    E = 0.0
    b = 0.0
    r_max = 0.0
    steps = 0
    dTdr = None
    def __init__( self, V, E, b, r_max, steps ) :
        self.V = V
        self.E = E
        self.b = b
        self.r_max = r_max
        self.steps = steps

    def f_r_min( self, r ) :
            return 1 - pow(self.b / r,2.0) - self.V(r) / self.E
```

```
    def dTheta_dr( self, r ) :
        X = self.f_r_min( r )
        return 1.0 / sqrt( abs(X) ) / pow(r,2.0)

    def trajectory( self ) :
        # Define theta step :
        dtheta = -1.0 * asin( self.b / self.r_max )
        # To return : list of trajectories
        rtheta = [self.r_max, pi + dtheta]
        traj = [ array( rtheta ) ]
        # To return : Total deflection
        deflection = pi - 2*dtheta

        # Find the distance of closest approach with the "root_simple" method
        dr = -1.0 * self.r_max / 100
        r_max = self.r_max
        r_min = root_simple( self.f_r_min, r_max, dr)

        # Integrate to find successive changes in theta :
        dr = (r_max - r_min) / self.steps
        accuracy = 1e-6
        for i in xrange(self.steps) :
            r_upper = traj[i][0]
            r_lower = r_upper - dr
            itheta = traj[i][1]
            dtheta = -self.b * adaptive_trapezoid( self.dTheta_dr, r_lower, r_upper
            rtheta[0] -= dr
            rtheta[1] += dtheta
            traj.append( array( rtheta ) )
            deflection += 2 * dtheta


        # Use symmetry to get the outgoing trajectory points
        for i in range( self.steps-1, 0, -1) :
            rtheta[0] += dr
            dtheta = traj[i][1] - traj[i-1][1]
            rtheta[1] += dtheta
            traj.append( array( rtheta ) )

        return [deflection, traj]


def main() :
    print ' Classical scattering from Lennard-Jones potential'
    E = 0.705
    print ' E = ' + str(E)
    b_min = 0.6
```

```
    db = 0.036
    n_b = 50
#     V = lennard_jones( V0 = 1.0 )
    V = Yukawa( V0 = 1.0, r0=10.0 )
    difffile = open( 'differential_cross_sectionr010.data', 'w' )
    for i in xrange(n_b) :
        b = b_min + db*i
        theta = Theta( V, E, b, 7.0, 100 )
        [deflection,trajs] = theta.trajectory()
        if i==0:
            deflast=deflection
        else:
            dcs=b*abs(db/(deflast-deflection))/sin(deflection)
            s = '{0:8.4f} {1:8.4f}{2:8.4f}\n'.format(b,dcs,deflection)
            difffile.write (s)
            deflast=deflection
#         print ' {0:4.2f} : {1:6.2f}'.format( b, deflection )
#         trajfile = open( 'trajfile_py_' + str(i) + '.data', 'w' )
#         for traj in trajs :
#             s = '{0:8.4f} {1:8.4f}\n'.format( traj[0] * cos( traj[1] ), traj[0] *
#             trajfile.write( s )
#         trajfile.close()

    print 'Finished!'

if __name__ == "__main__" :
    main()

import matplotlib.pyplot as plt

def read_plot(filename):
    file = open( filename, 'r')
    lines = file.readlines()
    x = []
    y = []
    for line in lines :
        words = line.split()
        ix, iy = [float(s) for s in words]
        x.append(float( ix ))
        y.append(float( iy ))
    return x,y

for i in range(18):
    x, y = read_plot("trajfile_Y_py_rmax5r10"+str(i)+".data")
#    print "trajfile_py_"+str(i)+".data"
    plt.plot(x,y)
plt.show()
```