PHYS811 Project 2

Marc Farrell

March 26, 2022

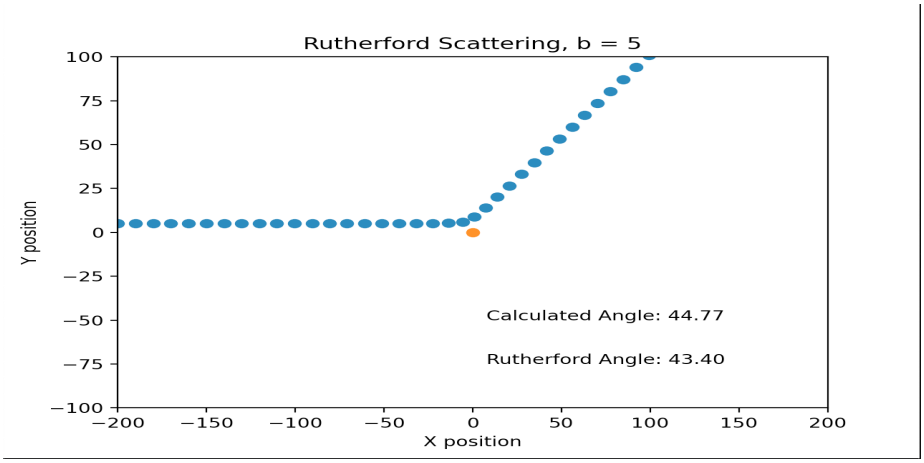
1 Introduction

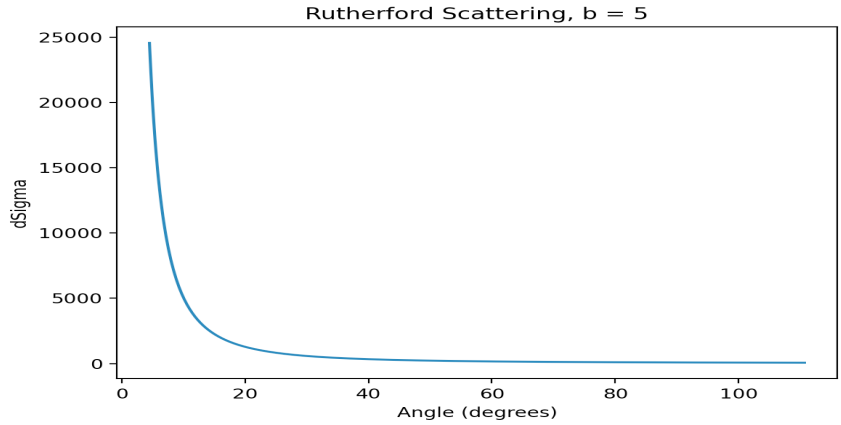
In this report, we explore classical scattering through the perspective of numerical techniques. Specifically, we consider 2-body central potential scattering with a Coulomb potential. In the first part, we have an heavy, immobile target particle. Here, we explore the trajectory of the incoming particle at different y-values, along with the relation between the impact parameter, b, the scattering angle, and the differential cross section. In the second part, we have a light, movable target particle. Once again, we explore the trajectory of the incoming particle along with the relationship of the impact parameter and the energy transferred between the two particles.

In my code, I used the Runge-Kutta 4th order method (RK4) for both parts. As we used a coulomb potential, the equations of motions were written as a system of ordinary differential equations, making the RK4 method. My code is posted at the end of the report, but the structure is as follows: initiate variables, define differential equations, apply RK4, call variables from RK4 and append to lists, graph desired lists against each other. In the code, each section is clearly labeled.

2 Question 1

Here, our goal was to make a plot of scattering in a Coulomb potential, calculate the scattering angle, and plot the differential cross section vs. scattering angle. Below are my two outputed graphs.



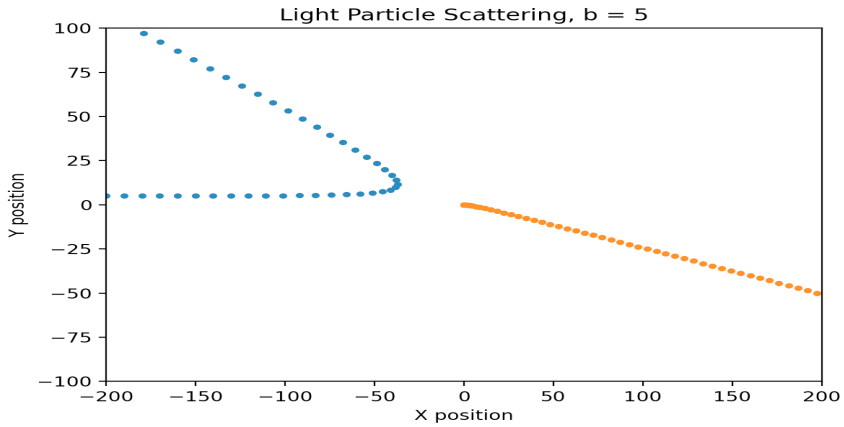


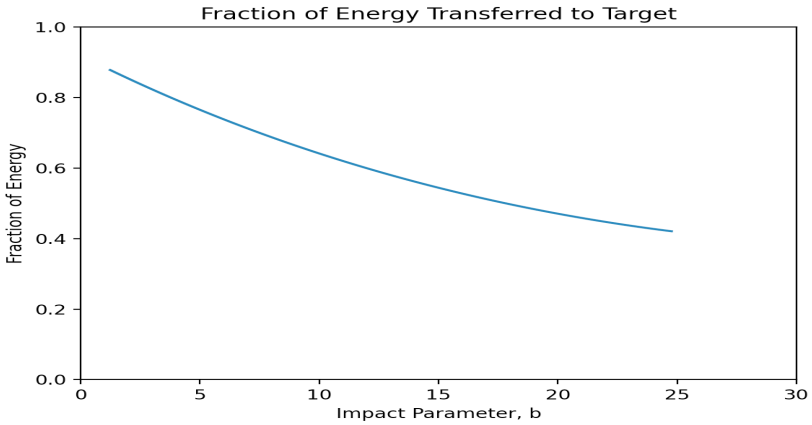
It is clear that both are giving the desired results. In the top graph, we have our immobile target particle at the origin, displayed as an orange dot. The incoming particle is represented by the dotted blue line. As we can see, it matches the expected result nicely. Adaptive steps were not used and I do not believe this problem needs it. We also have our calculated angle compared to the theoretical Rutherford angle – although they are not exactly the same, we see that the RK4 method does a great job at approximating a solution to the equations of motion and getting a scattering angle of reasonable accuracy
In the second graph, we once again see that our results match the expected results. Comparing to the theoretical function for the differential cross section, RK4 has shown its effectiveness as the lines are identical.

3 Question 2

The goal for this part was to change the target particle to a light, movable one and graph the trajectory, differential cross section, and the fraction of energy

transferred from the incoming particle to the target particle. My results are shown in the figures below:





First, we see that the trajectory matches the theoretical plot. Once again, an adaptive step size was not used. Regardless, we see the effectiveness of the RK4 method. In the second graph, we see the relation between the impact parameter and the fraction of energy transferred. It is simple to say that the lower the impact parameter, or the closer the incoming particle is to the target, the more energy should be transferred. An inverse relation. Therefore, we should see what our graph is showing up, validating our findings.

4 Critique

This project was another great lesson in numerical methods, especially the RK4 method. I used it in HW5 where I felt I developed an acceptable understanding of the topic. However, this project was an opportunity to not only solidify what I learned, but challenge myself as well. In that sense, I learned a lot from

revisiting a topic. In studying a topic after a short break from when we covered it in lecture, I was able to digest the information – reestablishing what I already know and making me able to get a deeper understanding. Overall, I thought this was a informative assignment and cannot think of any improvements.

5 Code

1

```
##############################
#Marc Farrell
#PHYS811 HW5
#Dr. Godunov
#3/26/2022
##############################



import numpy as np
import matplotlib.pyplot as plt


#define variables
m1 = 1
m2 = 1
x10 = -200
x20 = 0
y20 = 0
vx0 = 1
vy0 = 0
Z1 = 1
Z2 = 2
ti = 0
tf = 100
N = 100
dt = (tf-ti)/N
dt=10


#Differential Equation
def dx1(vx1):
    return vx1

def dy1(vy1):
    return vy1
```

```
def dvy1(x1, y1):
    return (y1 * Z1 * Z2) / ((x1 ** 2 + y1 ** 2) ** 1.5)

def dvx1(x1, y1):
    return (x1 * Z1 * Z2) / ((x1 ** 2 + y1 ** 2) ** 1.5)



#RK
def next(x10, y10, vx0, vy0):
    #k1
    k1x1 = dt * dx1(vx0)
    k1vx1 = dt * dvx1(x10, y10)
    k1y1 = dt * dy1(vy0)
    k1vy1 = dt * dvy1(x10, y10)

    #k2
    k2x1 = dt * dx1(vx0 + (k1vx1/2.0))
    k2vx1 = dt * dvx1(x10 + (k1x1/2), y10 + (k1y1/2))
    k2y1 = dt * dy1(vy0 + (k1vy1/2))
    k2vy1 = dt * dvy1(x10 + k1x1/2, y10 + k1y1/2)

    #k3
    k3x1 = dt * dx1(vx0 + (k2vx1/2.0))
    k3vx1 = dt * dvx1(x10 + (k2x1/2), y10 + (k2y1/2))
    k3y1 = dt * dy1(vy0 + (k2vy1/2))
    k3vy1 = dt * dvy1(x10 + k2x1/2, y10 + k2y1/2)

    #k4
    k4x1 = dt * dx1(vx0 + k3vx1)
    k4vx1 = dt * dvx1(x10 + k3x1, y10 + k3y1)
    k4y1 = dt * dy1(vy0 + k3vy1)
    k4vy1 = dt * dvy1(x10 + k3x1, y10 + k3y1)

    #new
    xnew = x10 + (k1x1 + 2*(k2x1+k3x1) + k4x1)/6
    ynew = y10 + (k1y1 + 2*(k2y1+k3y1) + k4y1)/6
    vxnew = vx0 + (k1vx1 + 2*(k2vx1+k3vx1) + k4vx1)/6
    vynew = vy0 + (k1vy1 + 2*(k2vy1+k3vy1) + k4vy1)/6

    return xnew, ynew, vxnew, vynew



b = np.linspace(1, 5, 100)
angles = np.zeros(len(b))
```

```
for i in range(len(b)):
    x10 = np.zeros(N) #create lists
    y10 = np.zeros(N)
    vx0 = np.zeros(N)
    vy0 = np.zeros(N)
    x10[0] = -200 #initial values
    y10[0] = b[i]
    vx0[0] = 1
    vy0[0] = 0

    for k in range(N-1):
        x10[k+1], y10[k+1], vx0[k+1], vy0[k+1] = next(x10[k], y10[k], vx0[k], vy

    angles[i] = (180/np.pi) * (np.arccos(vx0[-1]/((vy0[-1]**2 + vx0[-1]**2)**.5)

for l in range(1, len(b)-1):
    db = b[l+1] - b[l-1] /(2 * angles[l] - angles[l-1])



b = np.delete(b, 99)
b = np.delete(b, 0)
angles = np.delete(angles, 99)
angles = np.delete(angles, 0)
section = b/np.sin(angles * (np.pi/180)) * abs(db)

x=[0]
y=[0]

#Plot
plt.plot(x10, y10, 'o')
plt.plot(x,y, 'o')
plt.text(7.8, -50, 'Calculated Angle: 44.77')
plt.text(7.8, -75, 'Rutherford Angle: 43.40')
plt.xlim([-200, 200])
plt.ylim([-100, 100])
plt.xlabel("X position")
plt.title('Rutherford Scattering, b = 5')
plt.ylabel("Y position")
#plt.plot(angles, section)
plt.show()
```

2

```
##############################
#Marc Farrell
#PHYS811 HW5
```

```
#Dr. Godunov
#3/26/2022
##############################


import numpy as np
import matplotlib.pyplot as plt


#define variables
m1 = 1
m2 = 4
k = 1
x10 = -200
y10 = 0
x20 = 0
y20 = 0
vx0 = 1
vy0 = 0
vx20 = 0
vy20 = 0
Z1 = 1
Z2 = 2
ti = 0
tf = 100
N = 100
dt = (tf-ti)/N
dt=10

#Differential
def dx1(vx1):
    return vx1

def dvx1(x1, x2, y1, y2):
    return (k * Z1 * Z2 * (x1-x2))/(m1 * (((x1-x2) ** 2 + (y1-y2) ** 2) ** 1.5))

def dy1(vy1):
    return vy1

def dvy1(x1, x2, y1, y2):
    return (k * Z1 * Z2 * (y1-y2))/(m1 * (((x1-x2) ** 2 + (y1-y2) ** 2) ** 1.5))

def dx2(vx2):
    return vx2
```

```
def dvx2(x1, x2, y1, y2):
    return (k * Z1 * Z2 * (x2-x1))/(m2 * (((x1-x2) ** 2 + (y1-y2) ** 2) ** 1.5))

def dy2(vy2):
    return vy2

def dvy2(x1, x2, y1, y2):
    return (k * Z1 * Z2 * (y2-y1))/(m2 * (((x1-x2) ** 2 + (y1-y2) ** 2) ** 1.5))

#RK
def next(x10, x20, y10, y20, vx0, vy0, vx20, vy20):
    #k1
    k1x1 = dt * dx1(vx0)
    k1x2 = dt * dx2(vx20)
    k1y1 = dt * dy1(vy0)
    k1y2 = dt * dy2(vy20)
    k1vx1 = dt * dvx1(x10, x20, y10, y20)
    k1vx2 = dt * dvx2(x10, x20, y10, y20)
    k1vy1 = dt * dvy1(x10, x20, y10, y20)
    k1vy2 = dt * dvy2(x10, x20, y10, y20)

    #k2
    k2x1 = dt * dx1(vx0 + k1vx1/2)
    k2x2 = dt * dx2(vx20 + k1vx2/2)
    k2y1 = dt * dy1(vy0 + k1vy1/2)
    k2y2 = dt * dy2(vy20 + k1vy2/2)
    k2vx1 = dt * dvx1(x10 + k1x1/2, x20 + k1x2/2, y10 + k1y1/2, y20 + k1y2/2)
    k2vx2 = dt * dvx2(x10 + k1x1/2, x20 + k1x2/2, y10 + k1y1/2, y20 + k1y2/2)
    k2vy1 = dt * dvy1(x10 + k1x1/2, x20 + k1x2/2, y10 + k1y1/2, y20 + k1y2/2)
    k2vy2 = dt * dvy2(x10 + k1x1/2, x20 + k1x2/2, y10 + k1y1/2, y20 + k1y2/2)

    #k3
    k3x1 = dt * dx1(vx0 + k2vx1/2)
    k3x2 = dt * dx2(vx20 + k2vx2/2)
    k3y1 = dt * dy1(vy0 + k2vy1/2)
    k3y2 = dt * dy2(vy20 + k2vy2/2)
    k3vx1 = dt * dvx1(x10 + k2x1/2, x20 + k2x2/2, y10 + k2y1/2, y20 + k2y2/2)
    k3vx2 = dt * dvx2(x10 + k2x1/2, x20 + k2x2/2, y10 + k2y1/2, y20 + k2y2/2)
    k3vy1 = dt * dvy1(x10 + k2x1/2, x20 + k2x2/2, y10 + k2y1/2, y20 + k2y2/2)
    k3vy2 = dt * dvy2(x10 + k2x1/2, x20 + k2x2/2, y10 + k2y1/2, y20 + k2y2/2)

    #k4
    k4x1 = dt * dx1(vx0 + k3vx1)
    k4x2 = dt * dx2(vx20 + k3vx2)
    k4y1 = dt * dy1(vy0 + k3vy1)
    k4y2 = dt * dy2(vy20 + k2vy2)
```

```
    k4vx1 = dt * dvx1(x10 + k2x1, x20 + k2x2, y10 + k2y1, y20 + k2y2)
    k4vx2 = dt * dvx2(x10 + k2x1, x20 + k2x2, y10 + k2y1, y20 + k2y2)
    k4vy1 = dt * dvy1(x10 + k2x1, x20 + k2x2, y10 + k2y1, y20 + k2y2)
    k4vy2 = dt * dvy2(x10 + k2x1, x20 + k2x2, y10 + k2y1, y20 + k2y2)

    #new
    x1new = x10 + (k1x1 + 2*(k2x1+k3x1) +k4x1)/6
    x2new = x20 + (k1x2 + 2*(k2x2+k3x2) +k4x2)/6
    y1new = y10 + (k1y1 + 2*(k2y1+k3y1) +k4y1)/6
    y2new = y20 + (k1y2 + 2*(k2y2+k3y2) +k4y2)/6
    vx1new = vx0 + (k1vx1 + 2*(k2vx1+k3vx1) + k4vx1)/6
    vx2new = vx20 + (k1vx2 + 2*(k2vx2+k3vx2) + k4vx2)/6
    vy1new = vy0 + (k1vy1 + 2*(k2vy1+k3vy1) + k4vy1)/6
    vy2new = vy20 + (k1vy2 + 2*(k2vy2+k3vy2) + k4vy2)/6

    return x1new, x2new, y1new, y2new, vx1new, vx2new, vy1new, vy2new


b = np.linspace(1, 50, 100)
Energy=np.zeros(len(b))
angles = np.zeros(len(b))
for i in range(len(b)):
    x10 = np.zeros(N)
    x20 = np.zeros(N)
    y10 =np.zeros(N)
    y20=np.zeros(N)
    vx0=np.zeros(N)
    vx20=np.zeros(N)
    vy0=np.zeros(N)
    vy20=np.zeros(N)

    x10[0] = -200
    x20[0] = 0
    y10[0] = b[i]
    y20[0] = 0
    vx0[0] = 1
    vx20[0] = 0
    vy0[0] = 0
    vy20[0] = 0

    for k in range(N-1):
        x10[k+1], x20[k+1], y10[k+1], y20[k+1], vx0[k+1], vx20[k+1],vy0[k+1],vy2
    Energy[i] = .5 * m1 * ((vx0[-1]**2)+(vy0[-1]**2))
angles[i] = (180/np.pi) * (np.arccos(vx0[-1]/((vy0[-1]**2 + vx0[-1]**2)**.5)))

for l in range(1, len(b)-1):
```

```
    db = b[l+1] - b[l-1] /(2 * angles[l] - angles[l-1])


b = np.delete(b, 99)
b = np.delete(b, 0)
angles = np.delete(angles, 99)
angles = np.delete(angles, 0)
section = b/np.sin(angles * (np.pi/180)) * abs(db)
Energy = np.delete(Energy, 99)
Energy = np.delete(Energy, 0)

print(Energy)

# plt.plot(x10, y10, '.')
# plt.plot(x20,y20, '.')
plt.xlim([0, 120])
plt.ylim([0, 100])
# plt.xlabel("X position")
# plt.title('Light Particle Scattering, b = 5')
# plt.ylabel("Y position")
plt.plot(angles, section)
# plt.plot(b, Energy)
# plt.title("Fraction of Energy Transferred to Target")
# plt.xlabel("Impact Parameter, b")
# plt.ylabel("Fraction of Energy")
# plt.xlim([0, 30])
# plt.ylim([0, 1])
plt.show()
```