PHYS 566 HW2

Matthew Epland
Department of Physics, Duke University, Durham, NC 27707, USA
(Dated: February 3, 2016)

Nuclear decay activity $R(t)$ was simulated numerically using the Euler method in the context of Carbon-14 dating. Three time steps of $\Delta t = 10$, 100, and 100 years were used to model $R(t)$ over 20,000 years. The numerical results were compared to the analytical solution and found to be consistent with $\leq$1% divergence after two half-lives for $\Delta t = 10$ and 100 years, and inconsistent with 9.2% divergence for $\Delta t = 100$ years. For all three time steps the divergence observed was orders of magnitude smaller than the second order term of the analytical solution.

I. INTRODUCTION

Carbon dating is an extremely useful tool for establishing the age of ancient objects in archeology and related fields. From a physics stand point the method is fairly simple, being an application of well understood nuclear decays. In this assignment we numerically modeled the decay of a 10^{-12} kg sample of Carbon-14, which has a half-life of 5700 years, over a period of 20,000 years in order to gain practice applying the Euler method to solve differential equations. Carbon dating is a good case study for doing so as the exact result, a decaying exponential, can by derived analytically without much issue.

II. THEORY

The decay of atomic nuclei can be modeled by assuming the change in number of nuclei per unit time, $\frac{dN}{dt}$, is proportional to the number of nuclei present at the time, $N(t)$.

$$\frac{dN}{dt} = -\frac{1}{\tau}N(t) \tag{1}$$

This simple first-order differential equation (1) can be solved using separation of variables, resulting in the familiar equation for exponential decay (2).

$$N(t) = N_0 e^{-t/\tau} \tag{2}$$

The exponential decay is characterized by initial number of nuclei present, N_0, and the decay constant, τ. We can relate τ to the half-life of the nuclei, $T_{1/2}$, when half of the initial population has decayed by solving (3). Taking natural logs of both sides and rearranging gives the desired result, $T_{1/2} = \tau \ln(2)$.

$$\frac{N_0}{2} = N_0 e^{-T_{1/2}/\tau} \tag{3}$$

In this assignment we are interested in calculating the activity of the decay, $R(t)$, defined in (4).

$$R(t) \equiv -\frac{dN}{dt} = \frac{1}{\tau}N(t) = \frac{N_0}{\tau}e^{-t/\tau} \tag{4}$$

To calculate $N(t)$, and hence $R(t)$, numerically we can use the Euler method; approximating the derivative over a finite time step Δt as (5).

$$\frac{dN}{dt} \approx \frac{N(t+\Delta t) - N(t)}{\Delta t} \tag{5}$$

Substituting (1) in for $\frac{dN}{dt}$ in (5) and solving for $N(t+\Delta t)$ produces an iterative equation (6) which we can use to compute $N(t)$ and $R(t)$ in our program.

$$N(t+\Delta t) = \tau R(t+\Delta t) = \left(1 - \frac{\Delta t}{\tau}\right) N(t) \tag{6}$$

III. RESULTS

As can be seen in Figures 1 and 2, the Euler method numerical results using small time steps $\Delta t = 10$, and 100 years are practically indistinguishable from the exact result when plotted. However, when Δt is increased to 1000 years the numerical result begins to diverge visually.

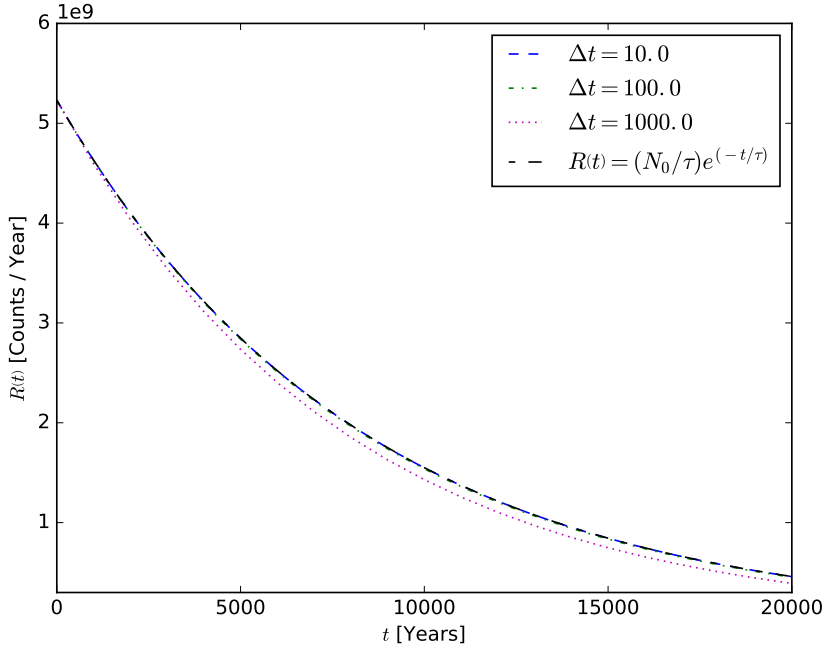

FIG. 1: $R(t)$ calculated numerically with three different time steps, Δt, and the exact solution.

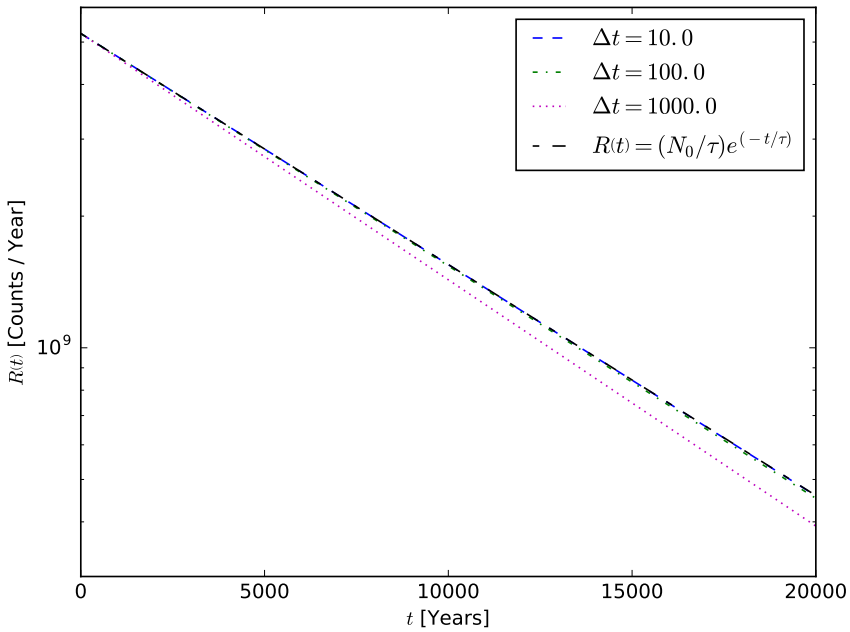

FIG. 2: Semi-log plot of $R(t)$ calculated numerically with three different time steps, Δt, and the exact solution.

Besides just looking at plots, we can quantify how well the numerical results match the exact result by comparing them at specific times. A reasonable time to pick is $t = 2T_{1/2}$ (specified in the assignment itself) as the majority of nuclei have decayed by then, giving the numerical result sufficient time to diverge. Two useful variables for performing the comparison are the raw deviation, $R(t)_{exact} - R(t)_{numerical}$, and percent deviation, $100\% \cdot (R(t)_{exact} - R(t)_{numerical}) / R(t)_{exact}$. We can also compare the deviation to the second order term of the exact result, which can be found by expanding the exponential and identifying the correct term, $(N_0/\tau)(-t/\tau)^2/2$. The modeling code was modified to perform these calculations and print the results to the screen, see below:

```
0
Half-Live is: 5700.0
Decay Constant is: 8223.36
Initial Mass is: 1.00E-12
Atomic Mass is: 14
5 Initial Number (N0) is: 4.30E+13
Stop Time is: 20000.0

Below are the various calculations of deviations from the exact result after two half-lives (11400.0
    years) for each time step.
-------------------------------------------------------------------------------------------------
10

Delta t = 10.0, 1st time step >= 2 T1/2 = 11400.0, exact R(t) = 1.308E+09, numerical R(t) = 1.307E+09,
    deviation = 1.103E+06
% deviation = 0.08432%, 2nd order term = 5.026E+09, deviation/2nd order term = 0.0002194

Delta t = 100.0, 1st time step >= 2 T1/2 = 11400.0, exact R(t) = 1.308E+09, numerical R(t) = 1.297E+09,
    deviation = 1.107E+07
15 % deviation = 0.84620%, 2nd order term = 5.026E+09, deviation/2nd order term = 0.0022016

Delta t = 1000.0, 1st time step >= 2 T1/2 = 12000.0, exact R(t) = 1.216E+09, numerical R(t) = 1.104E+09,
     deviation = 1.120E+08
% deviation = 9.21230%, 2nd order term = 5.569E+09, deviation/2nd order term = 0.0201088

20

Done!
```

output.log

As can be seen from the output, the percent deviation for both $\Delta t = 10$ and 100 years is acceptable at $\leq$1%. However setting $\Delta t = 1000$ years results in a percent deviation of 9.2% which is unacceptable for most applications. In this case using $\Delta t = 10$ years is not computationally prohibitive, so there is no problem producing a high quality numerical result. Interestingly, it appears that the percent deviation and Δt are linearly related, each increasing by an order of magnitude between the three different Δt's. Of course this is only considering three data points and thus is a very weak observation, but it would be an interesting area for future investigation.

Upon taking the ratio of the deviations to the second order term of the exact result, we see that the deviations are orders of magnitude smaller than the second order term for all three time steps. This implies that only going to first order with Euler method was sufficient to model the decays behavior over the range of parameters considered, something hinted at immediately by how well the plots matched visually.

IV. CONCLUSIONS

The nuclear decay of Carbon-14, which must be well understood to facilitate the experimental method of Carbon dating, was successfully modeled numerically via the Euler method, as well as being solved analytically in Section II. Of the three time steps used, the shortest two produced acceptable results within $\leq$1% of the exact result while the longest was unacceptable with a 9.2% deviation from the exact result. In all three cases the deviation from the exact result was much smaller in magnitude than the second order term of the analytic solution.

The Python source code used to produce these results can be found online at `http://github.com/mepland/PHYS_566_Computational_HW/tree/master/hw2/code`, and is included in Section VI.

V. SUPPORTING MATERIAL

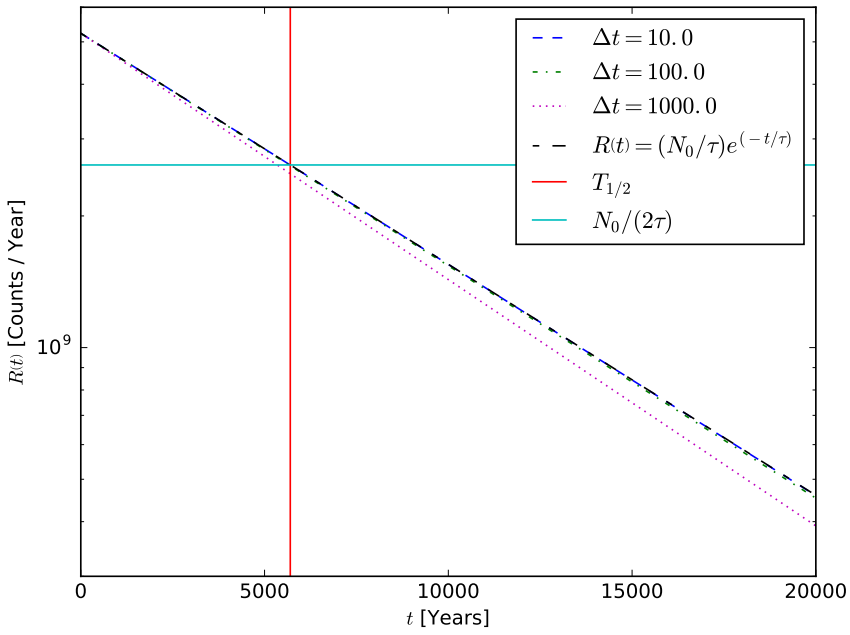

FIG. 3: $R\,(t)$ diagnostics plot generated to verify that $R\left(T_{1/2}\right) = \frac{N_0}{2\tau}$ as it should.

PHY566 Homework #2

Due Date: Feb. 4th, 5:00pm via Sakai

Carbon Dating

The Carbon isotope ${}^{14}_{6}C$ is widely used for dating of ancient artifacts containing biological material. It undergoes β^- decay with a half-life time of $T_{1/2} = 5700$ years. Suppose an ancient artifact originally contained 10^{-12} kg of ${}^{14}_{6}C$ (recall that 1 *mol* of a substance, which has a mass corresponding to the atomic mass-number in *grams*, contains $N_A = 6.022 \cdot 10^{23}$ particles).

a) Derive analytically the relation between half-life time, $T_{1/2}$ and decay constant τ as defined in class. [1 point]

b) write a computer code to numerically calculate the activity of the sample, defined as $R(t) = -dN/dt$, over a duration of 20,000 years. Use numerical time-step widths of 10 and 100 years. Plot the results in appropriate units together with the exact (analytical) solution in the same graph. [6 points]

c) Increase the time-step width to 1,000 years and replot. Is the accuracy of the numerical solution still acceptable? (e.g. what is the percentage deviation from the exact result after 2 half-lifes?) Is the deviation from the exact result as large as you would expect from the neglected 2nd order term? [3 points]

Your homework submission should consist of:

- a document, created in LaTeX, outlining the problem, detailing your solution and discussing your results - the document should include the requested figures. The document should be in pdf format and you should use colors and different marker symbols to enhance the readability of your figures.
- the Python code that you used to solve the problem and generate the figures

All files should be named in the following way:
[PHY566_your name_homework#2_filedescriptor]

VI. CODE

```
#!/usr/bin/env /home/mbe9/Documents/Spring_2016/PHYS_566_Computational_HW/anaconda/bin/python
# #! to the correct python install, not portable!!!

import os
import numpy as np
import matplotlib.pyplot as plt

#########################################################
# Set initial parameters
# Units: Time ~ years, N ~ raw number, mass ~ kg
half_life = 5700.0
initial_mass = 10**-12
stop_time = 20000.0
atomic_mass = 14.0

#########################################################
# Compute initial N0 from initial mass
NA = 6.022*(10**23)
N0 = NA*((initial_mass*1000.0)/atomic_mass)

# Compute decay constant tau from half-life
tau = half_life/np.log(2)

# Print out starting values
print '\nHalf-Live is: %.1f' % half_life
print 'Decay Constant is: %.2f' % tau
print 'Initial Mass is: %2.2E' % initial_mass
print 'Atomic Mass is: %d' % atomic_mass
print 'Initial Number (N0) is: %2.2E' % N0
print 'Stop Time is: %.1f' % stop_time

#########################################################
# Set time steps
dt1 = 10.0
dt2 = 100.0
dt3 = 1000.0

# Produce the desired t ndarrays, from 0.0 to stop_time in steps of dt
# need to add stop_time+dt to get stop_time
t1 = np.arange(0,stop_time+dt1,dt1)
t2 = np.arange(0,stop_time+dt2,dt2)
t3 = np.arange(0,stop_time+dt3,dt3)

# Declare the N ndarrays
N1 = np.zeros(t1.size)
N2 = np.zeros(t2.size)
N3 = np.zeros(t3.size)

# Initialize the N ndarrays to N0
N1[0] = N0
N2[0] = N0
N3[0] = N0

# Fill the ndarrays
for i in range(1,t1.size):
  N1[i] = (1 - (dt1/tau))*N1[i-1]

for i in range(1,t2.size):
  N2[i] = (1 - (dt2/tau))*N2[i-1]

for i in range(1,t3.size):
  N3[i] = (1 - (dt3/tau))*N3[i-1]

#########################################################
# Create the plots
R1_label = '$\Delta t = %.1f$' % dt1
R1_plt, = plt.plot(t1, N1/tau, 'b-', label=R1_label)

R2_label = '$\Delta t = %.1f$' % dt2
R2_plt, = plt.plot(t2, N2/tau, 'g-.', label=R2_label)

R3_label = '$\Delta t = %.1f$' % dt3
R3_plt, = plt.plot(t3, N3/tau, 'm:', label=R3_label)


R_explicit_plt, = plt.plot(t1, (N0/tau)*np.exp(-t1/tau), 'k', label='$R\left(t\\right) = \left(N_{0}/\\
    tau\\right) e^{\left(-t/\\tau\\right)}$')
m_dashes = [4, 8, 8, 8]  # number of points: on, off, on, off
R_explicit_plt.set_dashes(m_dashes)
```

```
80 ##########################################################
   # Format the plot

   # Warning, y axis range is hard coded here!
   plt.axis([0, stop_time, 3*10**8, 6*10**9])
85
   plt.xlabel('$t$ [Years]')
   plt.ylabel('$R\left(t\\right)$ [Counts / Year]')

   plt.legend()
90
   ##########################################################
   # Set output path, hard coded...
   output_path = './output'

95 # Create the output dir, if it already exists don't crash, otherwise raise an exception
   # Adapted from A-B-B's response to http://stackoverflow.com/questions/273192/in-python-check-if-a-
       directory-exists-and-create-it-if-necessary
   # Note in python 3.4+ 'os.makedirs(output_path, exist_ok=True)' would handle all of this...
   try:
       os.makedirs(output_path)
100 except OSError:
       if not os.path.isdir(output_path):
           raise Exception('Problem creating output dir %s !!!\nA file with the same name probably already
       exists, please fix the conflict and run again.' % output_path)

   ##########################################################
105 # Print the plot
   plt.savefig(output_path+'/plot.pdf')
   plt.savefig(output_path+'/plot.png')

   ##########################################################
110 # Make y axis log scale
   plt.yscale('log')

   # Re-Print the plot
   plt.savefig(output_path+'/log_plot.pdf')
115 plt.savefig(output_path+'/log_plot.png')

   ##########################################################
   # Create diagnostic plot to check the half-life is in the correct place...

120 # Add vertical line at T1/2
   plt.axvline(x=half_life, color='r', linestyle='-', label='$T_{1/2}$')

   # Add horizontal line at (N0/2)/tau
   plt.axhline(y=(N0/2)/tau, color='c', linestyle='-', label='$N_{0}/(2$$\\tau)$')
125
   # Redraw the legend
   plt.legend()

   # Print the diagnostic plot
130 plt.savefig(output_path+'/diagnostic_plot.pdf')
   plt.savefig(output_path+'/diagnostic_plot.png')


   ##########################################################
135 # Compute and print out the desired accuracy calculations

   # Find the time step immediately after two half-lives have passed

   # t1
140 i = 0
   while t1[i] < 2*half_life:
     i = i + 1

   t1_i = i
145
   # t2
   i = 0
   while t2[i] < 2*half_life:
     i = i + 1
150
   t2_i = i

   # t3
   i = 0
155 while t3[i] < 2*half_life:
     i = i + 1

   t3_i = i
```

```
############################################################
# Print the results

print '\nBelow are the various calculations of deviations from the exact result after two half-lives
    (%.1f years) for each time step.' % (2*half_life)
print '-------------------------------------------------------------------------------------------'

exact_result = (N0/tau)*np.exp(-t1[t1_i]/tau)
numerical_result =  N1[t1_i]/tau
second_order_term = (N0/tau)*(0.5*((-t1[t1_i]/tau)**2))
deviation = exact_result-numerical_result
percent_deviation = 100*(deviation/exact_result)

print '\nDelta t = %4.1f, 1st time step >= 2 T1/2 = %.1f, exact R(t) = %.3E, numerical R(t) = %.3E,
    deviation = %.3E' % (dt1, dt1*t1_i, exact_result, numerical_result, deviation)
print '%% deviation = %.5f%%, 2nd order term = %.3E, deviation/2nd order term = %.7f' % (
    percent_deviation, second_order_term, deviation/second_order_term)


exact_result = (N0/tau)*np.exp(-t2[t2_i]/tau)
numerical_result =  N2[t2_i]/tau
second_order_term = (N0/tau)*(0.5*((-t2[t2_i]/tau)**2))
deviation = exact_result-numerical_result
percent_deviation = 100*(deviation/exact_result)

print '\nDelta t = %4.1f, 1st time step >= 2 T1/2 = %.1f, exact R(t) = %.3E, numerical R(t) = %.3E,
    deviation = %.3E' % (dt2, dt2*t2_i, exact_result, numerical_result, deviation)
print '%% deviation = %.5f%%, 2nd order term = %.3E, deviation/2nd order term = %.7f' % (
    percent_deviation, second_order_term, deviation/second_order_term)


exact_result = (N0/tau)*np.exp(-t3[t3_i]/tau)
numerical_result =  N3[t3_i]/tau
second_order_term = (N0/tau)*(0.5*((-t3[t3_i]/tau)**2))
deviation = exact_result-numerical_result
percent_deviation = 100*(deviation/exact_result)

print '\nDelta t = %4.1f, 1st time step >= 2 T1/2 = %.1f, exact R(t) = %.3E, numerical R(t) = %.3E,
    deviation = %.3E' % (dt3, dt3*t3_i, exact_result, numerical_result, deviation)
print '%% deviation = %.5f%%, 2nd order term = %.3E, deviation/2nd order term = %.7f' % (
    percent_deviation, second_order_term, deviation/second_order_term)



print '\n\nDone!\n'
```

exp_decay.py