PHYS 512 Assignment 3

Michelle Lam, SN: 261005326

October 9, 2021

Problem 1

Refer to code **ps3_Problem1.py.**

Derivative we want to integrate and compare with our analytical solution.

$$\begin{aligned} \frac{dy}{dx} &= \frac{y}{1+x^2} \\ dy &= \int \frac{ydx}{1+x^2} \\ y(x) &= exp(arctan(x))exp(c)) \end{aligned} \tag{1}$$

Inputting our initial values:

$$\begin{aligned} y(-20) &= 1 \\ y(-20) = 1 &= exp(arctan(-20))exp(c) \\ exp(c) &= exp(-arctan(-20)) \\ c &= -arctan(-20) \end{aligned} \tag{2}$$

Therefore our analytical solution is :

$$y(x) = exp(arctan(x)) - arctan(-20)) \tag{3}$$

- combine 2nd order errors, to zero out leading error, to get 3rd order accurate
- take 1 step , take two half size steps, and go from there and take a half size step a 2nd time
- know error should shrink by factor of 4 $(1/2)^{order}$

Regular Runge-Kutta method (used for function rk4_step):

$$\begin{aligned} k_1 &= hf(x, y) \\ k_2 &= hf(x + h/2, y + k_1/2) \\ k_3 &= hf(x + h/2, y + k_2/2) \\ k_4 &= hf(x + h, y + k_3) \end{aligned} \tag{4}$$

For the function, rk4_stepd, the step doubling method was used.

A full step and two half steps are taken:

1. y1 = rk4_step(fun,x,y,h) – full step h
2. y2a =rk4_step(fun,x,y, h/2) – first half step
3. y2b = rk4_step(fun,x+h/2, y2a, h/2) – second half step
4. y2 = y2a+y2b – add to half steps for full step

To cancel our leading error $h^5\phi$:

$$\begin{aligned} y(x+h) &= y_1 + (h)^5\phi + O(h^6) \\ y(x+h) &= y_2 + 2(\frac{h}{2})^5\phi + O(h^6) \\ \frac{16}{15}(y_2 - y_1) &= h^5\phi \\ y(x+h) &= y_2 + \frac{16(y_2 - y_1)}{15 \cdot 2^4} + O(h^6) \\ y(x+h) &= y_2 + \frac{y_2 - y_1}{15} + O(h^6) \end{aligned} \tag{5}$$

Since the step doubling method uses 3 times the amount of function evaluations than the original single step rk4 method, instead of using 200 steps, 200/3 steps were used for rk4_stepd.

The solution for two methods and their respective residuals are plotted below. The mean error in using the two half steps method had a lower error by a factor of ≈ 10, despite the same amount of function evaluations.

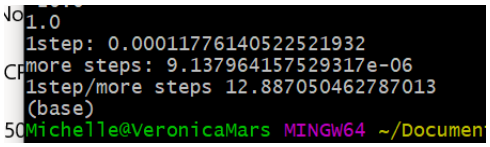

Figure 1: Output of error and comparison from **ps3_Problem1.py**.

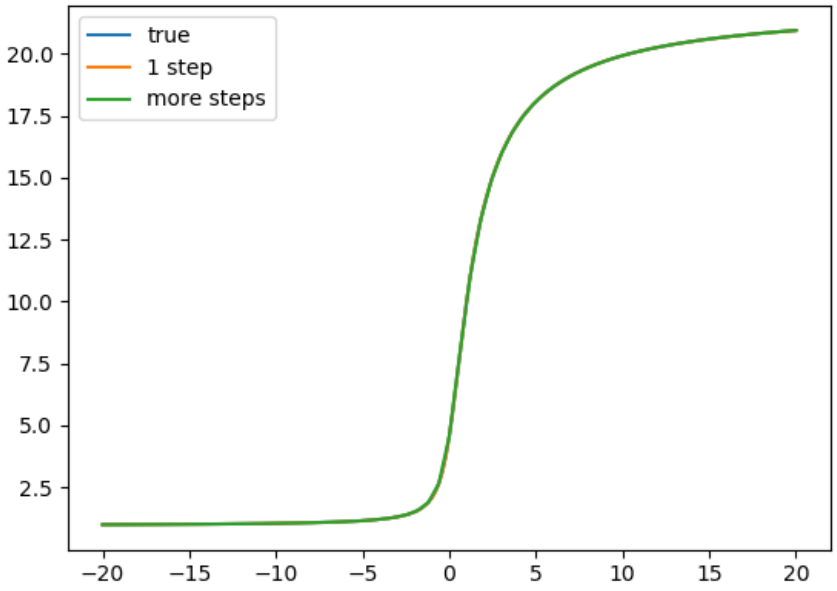

Figure 2: Output from **ps3_Problem1.py**. The analytical solution is labelled 'true', the regular rk4_step method is labelled '1 step' and the step doubling rk4_stepd method is labelled 'more steps'.

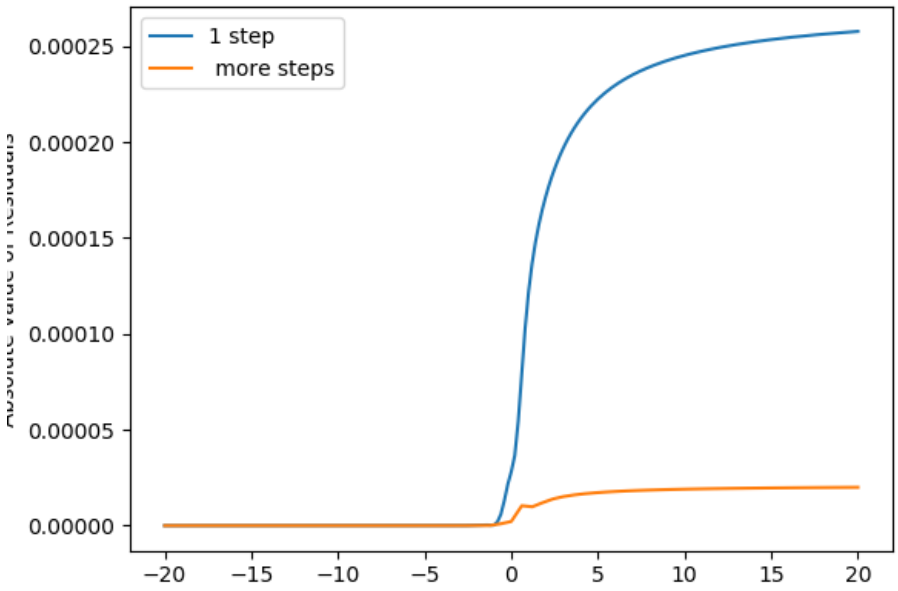

Figure 3: Output from **ps3_Problem1.py**. – calculates the residuals. Note,the analytical solution is labelled 'true', the regular rk4_step method is labelled '1 step' and the step doubling rk4_stepd method is labelled 'more steps'.

Problem 2

Refer to code: **ps3_Problem2.py.**

	Half-Life	Time unit	Emitter
Uranium-238	4,468	billion of years	alpha
Thorium-234	24,10	days	beta -
Protactinium-234	6,70	hours	beta -
Uranium-234	245 500	years	alpha
Thorium-230	75380	years	alpha
Radium-226	1 600	years	alpha
Radon-222	3,8235	days	alpha
Polonium-218	3,10	minutes	alpha
Plomb-214	26,8	minutes	beta -
Bismuth-214	19,9	minutes	beta -
Polonium-214	164,3	microseconds	alpha
Plomb-210	22,3	years	beta
Bismuth-210	5,015	years	beta
Polonium-210	138,376	days	alpha
Plomb-206	Stable		

Figure 4: Decay chain for q2.

Part A

Since the half lives, i.e. decay rates, are very different for each decay, we used an ODE solver from scipy that uses the method 'radau', which is an implicit/stiff ode solver. This method allows us to not have to take very tiny step sizes to account for the fast decay if we're at the part of the decay chain which is slowly decaying, i.e. it's adaptive.

The start of decay chain, first element simply decays:

$$\frac{dU_{238}}{dt} = -U_{238}\lambda_{U238} \tag{6}$$

The in-between elements, receive new material while decaying such as:

$$\frac{dTh_{234}}{dt} = U_{238}\lambda_{U238} - Th_{234}\lambda_{Th234} \tag{7}$$

The last element is stable:

$$\frac{dPb_{206}}{dt} = Pl_{210}\lambda_{Pl210} \tag{8}$$

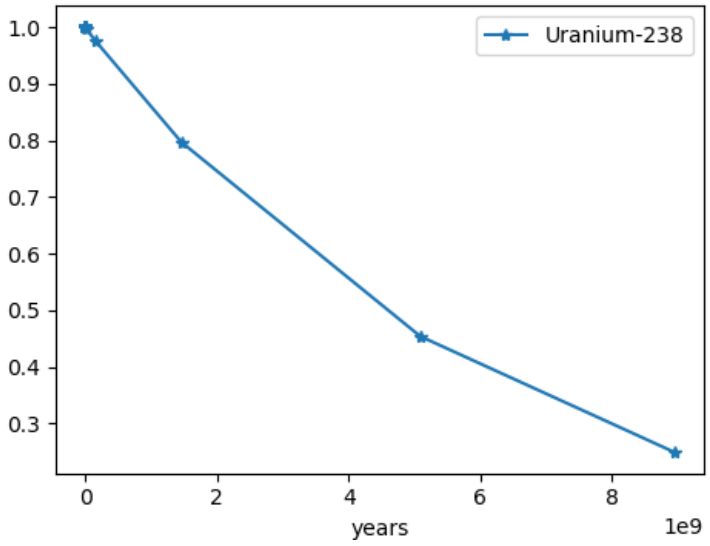

Figure 5: Example of output from program, **ps3_Problem2.py.**

Part B

Using program from Part A for plotting stiff solution to derivative. Analytical solution, Eq. 11, used to compare with our implicit stiff solution for Fig. 6. Run **ps3_Problem2.py** script.
Analytical solution:

$$N_{U238} = N_0 e^{-\lambda_{U238} t} \tag{9}$$

$$N_{Pb206} = 1 - N_0 e^{-\lambda_{U238} t} \tag{10}$$

$$\begin{aligned} \frac{N_{Pb206}}{N_{U238}} &= \frac{1 - N_0 e^{\lambda_{U238} t}}{N_0 e^{-\lambda_{U238} t}} \\ &= \frac{1}{N_0 e^{-\lambda_{U238} t}} - 1 \end{aligned} \tag{11}$$

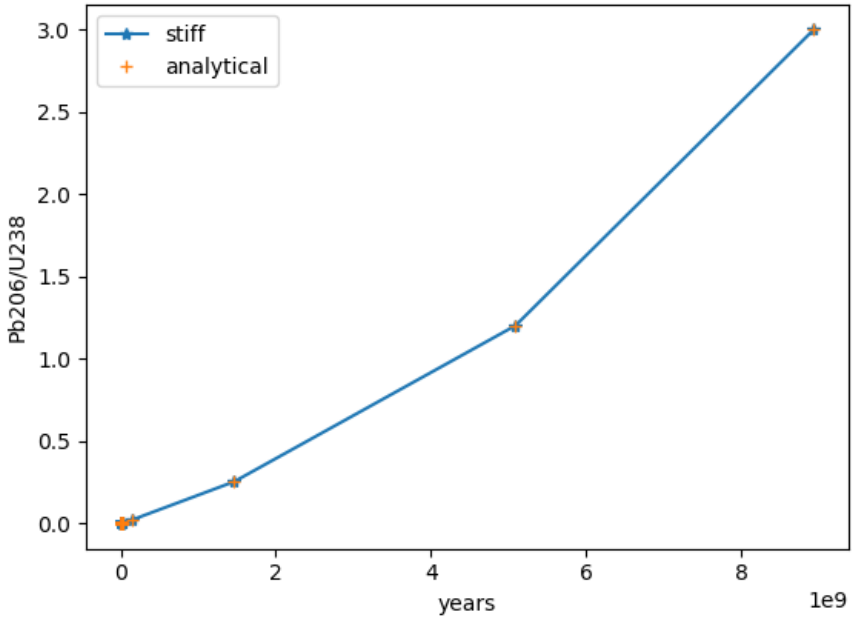

Figure 6: Ratio of Pb206 to U238 graph plotted for both the analytical and stiff solution.

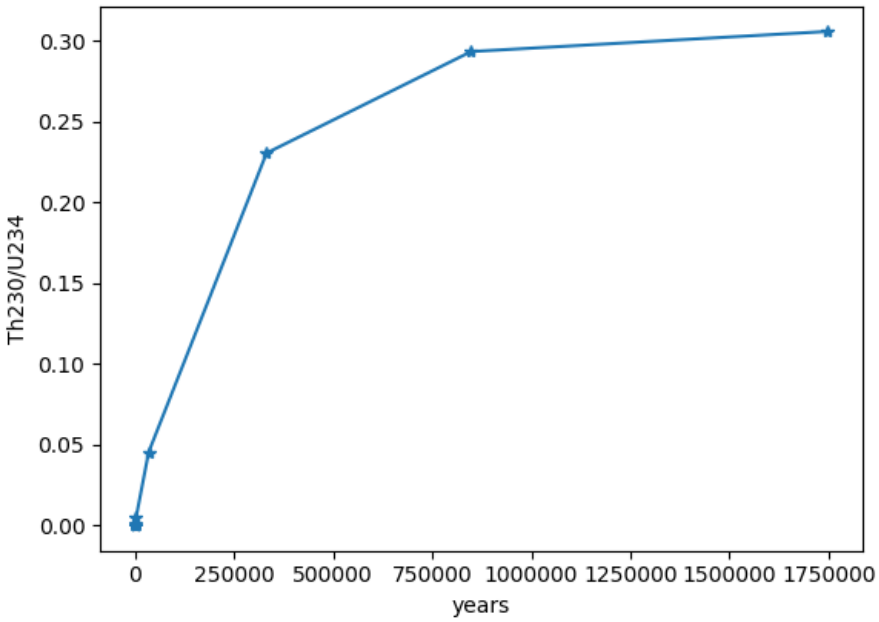

Figure 7: Ratio of Th230 to U234 graph plotted for both the analytical and stiff solution. Zoomed in around interesting part.

Problem 3

Refer to code **ps3_Problem3.py.**

Part A

Linear least squares fitting to fit the data dish_zenith.txt with the equation below.

$$z - z_0 = a((x - x_0)^2 + (y - y_0)^2) \tag{12}$$

Rewrite/Expand equation to linearize:

$$\begin{aligned} z - z_0 &= a(x^2 - 2xx_0 + x_0^2 + y^2 - 2yy_0 + y_0^2) \\ z &= (a)(x^2 + y^2) + (-2x_0a)(x) + (-2y_0a)(y) + (z_0 + ax_0^2 + ay_0^2) \end{aligned} \tag{13}$$

$$\begin{aligned} < d > &= Am \\ z &= Am \end{aligned} \tag{14}$$

Rewriting our linearized equation as a matrix:

$$z = \begin{bmatrix} x_1^2 + y_1^2 & x_1 & y_1 & 1 \\ x_2^2 + y_2^2 & x_2 & y_2 & 1 \\ .. & .. & .. & 1 \\ .. & .. & .. & 1 \\ .. & .. & .. & 1 \\ .. & .. & .. & 1 \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \end{bmatrix} \tag{15}$$

$$z = \begin{bmatrix} x_1^2 + y_1^2 & x_1 & y_1 & 1 \\ x_2^2 + y_2^2 & x_2 & y_2 & 1 \\ .. & .. & .. & 1 \\ .. & .. & .. & 1 \\ .. & .. & .. & 1 \\ .. & .. & .. & 1 \end{bmatrix} \begin{bmatrix} a \\ -2x_0a \\ -2y_0a \\ z_0 + ax_0^2 + ay_0^2 \end{bmatrix} \tag{16}$$

Therefore our old parameters can be found from our new parameters m by:

$$\begin{aligned} a &= m_1 \\ x_0 &= \frac{-m_2}{2a} \\ y_0 &= \frac{-m_3}{2a} \\ z_0 &= m_4 - ax_0^2 - ay_0^2 \end{aligned} \tag{17}$$

Part B

$$m = (A^T N^{-1} A)^{-}1 A^T N^{-1} z \tag{18}$$

Here we assume N = 1. Carrying out the fit, we used Eq. to fit the model parameter and I got:
a = 0.000167,
x0 = -1.36 mm,
y0 = 58.22 mm,
z0 = -1512.88 mm :

```
$ python ps3_Problem3.py
a: 0.00016670445477401347
x0: -1.3604886221976673
y0: 58.22147608157938
z0: -1512.8773767412422
noise in data: 3.7683386487847366
```

Figure 8: output from program, **ps3_Problem3.py.**

Part C

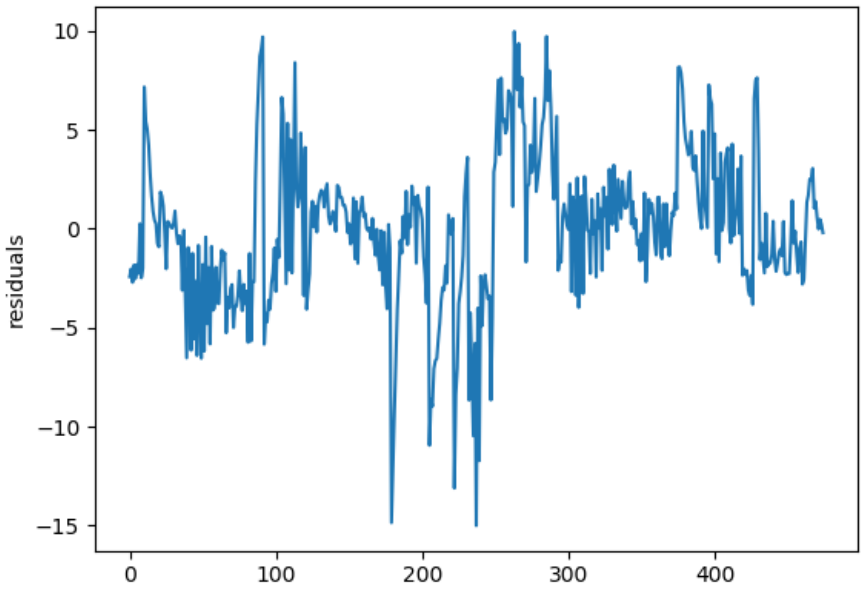

Figure 9: Residuals of modelled z from the actual z values.

The noise was estimated by taking the standard deviation of the above graph. (So no longer assume N=1, instead use standard deviation of noisy data w.r.t our model fit. (though N=1 results are also included in code, if interested)

$$\begin{aligned} N &= diag(\sigma^2) \\ parameter.errors &= (A^T N^{-1} A)^{-1} \end{aligned} \tag{19}$$

```
a: 0.00016670445477401347  +/-  6.451899757263484e-08
x0: -1.3604886221976673  +/-  0.00012506109951270817
y0: 58.22147608157938  +/-  0.00011924956427610148
z0: -1512.8773767412422  +/-  0.31201843620201836
focal length:  1499.659984125217 +/- 0.5804077581892867
(base)
```

Figure 10: Output from code, **ps3_Problem3.py.**

The focal length was calculated like so:

$$\begin{aligned} f &= \frac{1}{4a} \\ \delta f &= \frac{\delta a}{a} f \end{aligned} \tag{20}$$

So the target focal length was 1.5m and my model fitted **1.5 ± 0.6 m**, which agrees with the target.