Phys 512 Assignment 3

Ian Hendricksen

October 8, 2021

All relevant outputs from my script can be found in the file `A3 Print Output.txt`.

1. Following from Numerical Recipes' method for canceling the leading-order error term (5th order), we can implement Runge-Kutta first by taking a full step of length h, and then two half steps of length $h/2$. We then have two evaluations of $y(x+h)$ as

$$\begin{aligned} y(x+h) &= y_h + (2h)^5\phi + 0(h^6) \\ y(x+h) &= y_{h/2} + 2(h^5)\phi + O(h^6), \end{aligned} \tag{1}$$

where y_h is evaluated using Runge-Kutta for one full h step, and $y_{h/2}$ is evaluated using Runge-Kutta for two $h/2$ steps (Eqn. 17.2.1 in text). The fifth-order error terms reflect the manner in which y_h and $y_{h/2}$ are calculated. If we define $\Delta \equiv y_{h/2} - y_h$, we can rewrite 1 as

$$y(x+h) = y_{h/2} + \Delta/15 + O(h^6), \tag{2}$$

and we have thus canceled the leading 5th-order error term. This will be our definition to use in our function `rk4_stepd`. This uses 11 function evaluations every time we want to integrate a new y value. One full step of length h requires 4 function evaluations (i.e. the number of function evaluation of `rk4_step`), and each half step of length $h/2$ requires 4 function evaluations, but we evaluate the same $k1$ for the full step and the first half step, so we don't have to re-evaluate this value.

We evaluate the ODE and initial condition

$$\frac{dy}{dx} = \frac{y}{1+x^2} \quad y(-20) = 1, \tag{3}$$

from x = [-20, 20] using both `rk4_step` and `rk4_stepd`, with 200 steps. The explicit solution is

$$y(x) = c_0 e^{\tan^{-1}(x)} \quad c_0 = \frac{1}{e^{\tan^{-1}(-20)}}. \tag{4}$$

Defining our error estimate as the standard deviation of the difference between the exact solution and each `rk4` evaluation, we find that the error of `rk4_step` is 0.00011776140531974458, and the error of `rk4_stepd` is 0.02890610767690144 (both printed each time the script is run).

Now we want to integrate the ODE using `rk4_stepd` such that we evaluate the function the same number of times as `rk4_step`. For 200 steps, `rk4_step` evaluates the function 800 times. For `rk4_stepd` to evaluate the function 800 times, `rk4_stepd` needs to have 800/11 $\sim$ 73 steps. For 73 steps, the error of `rk4_stepd` is 0.05891211390122789. This suggests that `rk4_step` is more accurate.

We plot the 3 results (200 steps for `rk4_step` and `rk4_stepd`, and 73 steps for `rk4_stepd`) in Figure 1, along with the exact solution. Note that the most significant deviations from the exact solution occur in the area of greatest slope (i.e. the spike in the middle) and at corners.

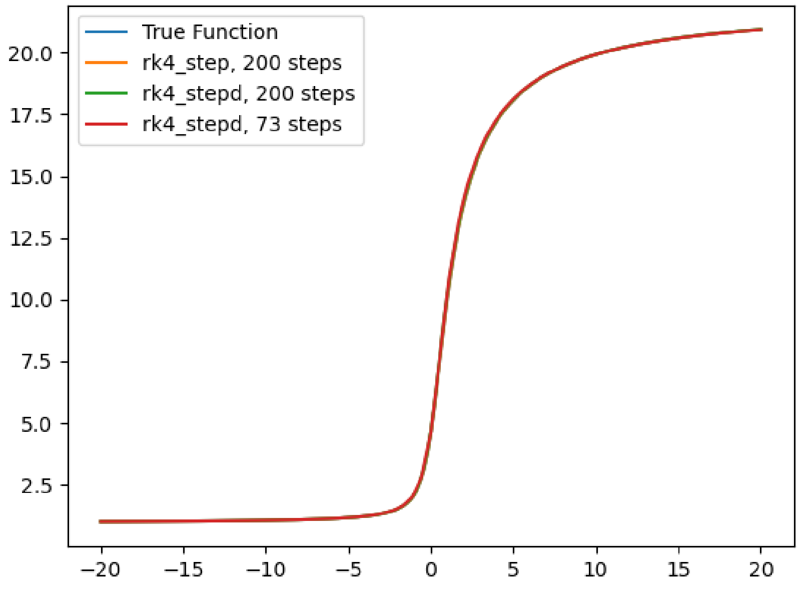

Figure 1: The exact solution plotted with various solutions using RK4 as our ODE solver.

2. (a) The fit is performed with the function `U238_products` I have created, which relies upon SciPy's `integrate.solve_ivp`. I have set *method* = "*Radau*" as our solver since we are dealing with a stiff set of equations due to the large differences in half lives. I set initial amounts to be 1 for U238 and 0 for all other products such that it is normalized. There is a subfunction called `decays` which essentially loops through all the 2-state decay equations for each product. It returns dN/dt for each product, and the loop updates the current and previous dN/dt for each iteration. We then call `solve_ode = integrate.solve_ivp`, where `solve_ode` contains all the integrated amount of each product, and `decays` as an argument and the relevant times, starting point, and method. Finally, `U238_products` returns `solve_ode`.

   (b) Figure 2 plots the ratio of Pb_{206} to U_{238} as a function of time. We observe that the ratio of Pb_{206} to U_{238} is at or nearly 0 until $\sim$70 Gyr, at which point the ratio explodes. The half life of U_{238} is about 4 Gyr, so the time scale at which most of the U_{238} begins to substantially decay into Pb_{206} (as indicated by the explosion in the ratio) makes sense in order of magnitude. This also follows from the fact that the other products' half lives are negligible compared to those of Pb_{206} to U_{238}.

   Figure 3 plots the ratio of Th_{230} to U_{234} as a function of time. The ratio approaches an asymptote at $\sim$0.3 just before 1 Myr.

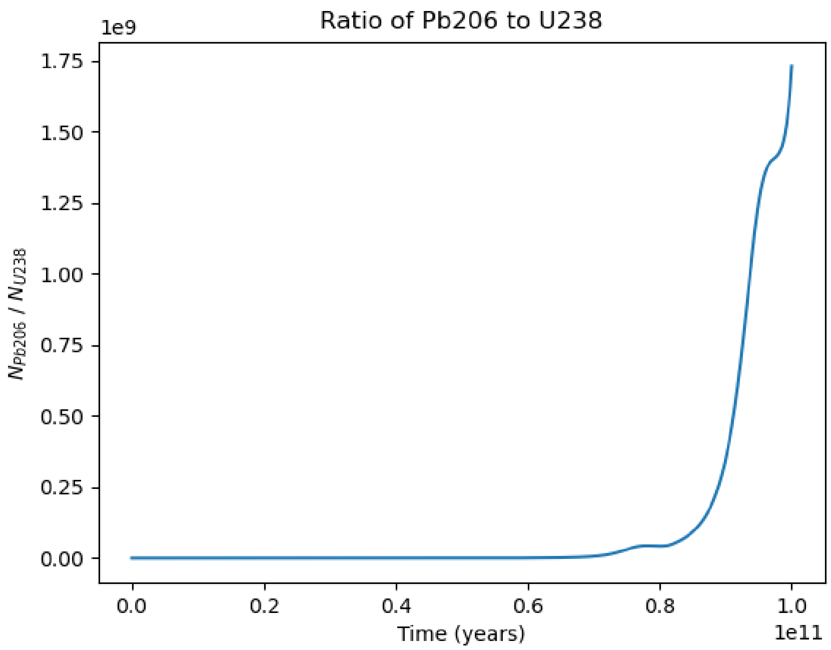

Figure 2: The ratio of Pb_{206} to U_{238} as a function of time.

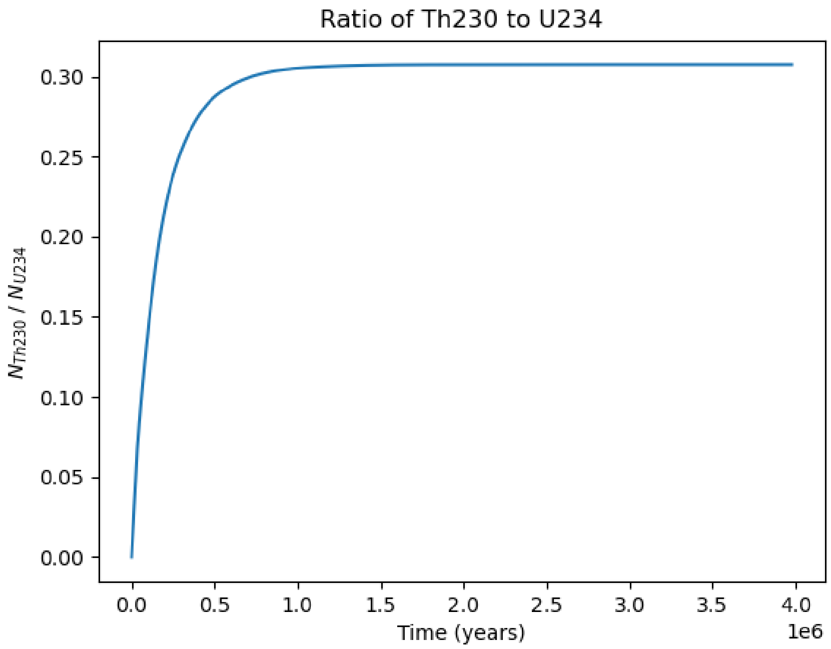

Figure 3: The ratio of Th_{230} to U_{234} as a function of time.

3. (a) A rotationally symmetric paraboloid is described by

$$z = z_0 + a\left((x - x_0)^2 + (y - y_0)^2\right), \tag{5}$$

and we want to fit for x_0, y_0, z_0, and a. However, we see that x_0 and y_0 are not linear in the equation. In order to make this equation linear in its parameters, we must substitute new ones. If we expand 5, we find

$$z = z_0 + a\left(x^2 + y^2 - 2xx_0 - 2yy_0 + x_0^2 + y_0^2\right). \tag{6}$$

Let $b = -2ax_0$, $c = -2ay_0$, and $d = a(x_0^2 + y_0^2) + z_0$; our new equation becomes

$$z = a(x^2 + y^2) + bx + cy + d. \tag{7}$$

(b) The fit is performed with my function `dish_fit`. It performs the fit assuming no noise N, since we are directed to estimate N in (c). The best fit parameters m are contained in `A3 Print Output.txt`.

(c) We form the noise matrix N by assuming that every value of σ_i is the same ($\sigma_i = \sigma\ \forall i$). We then estimate that σ is equal to the standard deviation of the difference between the new z values returned from our fit and the original z values from data, such that

$$\sigma = np.std(abs(z_{new} - z)). \tag{8}$$

Using this, we fill our noise matrix N with σ along the diagonals, and use this to compute the covariance matrix, and finally take the square root of its diagonals to compute the errors on fitted parameters m. These errors are printed, and can be found under `A3 Print Output.txt`, and in the table at the end of this document.

Following from the fact that a parabola centered at (0, 0) can be described by $y = x^2/4f$, where f is the focal length, we can "shift" Equation 7 such that it is centered at (x, z) = (0, 0) (ignoring the y-axis), such that it can be written as

$$z' = a(x')^2, \tag{9}$$

where the primes denote the shifted nature of the equation. We need not consider the y terms here since x and z are sufficient to describe a single parabola along the paraboloid (we could equivalently have written $z' = a(y')^2$ since the terms have the same scale and power). The linear x term and the offset d drop as a result of the shift. The shift preserves the information of the focal length f, such that we can state that

$$f = \frac{1}{4a}. \tag{10}$$

The error on f is computed as

$$\alpha_f = \frac{\alpha_a}{4a^2}, \tag{11}$$

where α_f is the error in the focal length and α_a is the error in a, following from the first-order Taylor expansion of the error in f. The value of f and its associated error is also printed to `A3 Print Output.txt` using the fitted parameter a and its associated error.

The values of the fit parameters m, their associated errors, and the value of f and its associated error are summarized in the following table:

Parameter	Value	Error
a	0.00016670445477399504 mm^{-1}	4.325681536063054e-08 mm^{-1}
b	0.0004535990259629005	8.384731775053653e-05
c	-0.019411558866494028	7.995096913774571e-05
d	-1512.3118166739068 mm	0.20919301898187223 mm
f	1.4996599841253828 m	0.0003891348622025981 m

Table 1: Pretty close to the target value of $f = 1.5$m! Note that the units of a are in mm^{-1}, b and c are unitless, and d is in units of mm.