PHYS-512: Problem Set 3

Simon Briesenick

October 1, 2022

1. A Runge-Kutta integrator of fourth order with step-doubling compares the results with step size of h and $h/2$:

$$f(x+h) = f_1 + h^5\phi + O(h^6) \tag{1}$$

$$f\left(x + 2\frac{h}{2}\right) = f_2 + \left(\frac{h}{2}\right)^5 \phi + O(h^6) \tag{2}$$

   where $\phi \propto f^{(5)}$ is identical in both results, since it is accurate up to sixth order in h. The results are labeled `result1` and `result2`, respectively. An improved estimate can then be obtained by solving the linear combination

$$af(x+h) + bf\left(x + 2\frac{h}{2}\right) \rightarrow a + b = 1 \tag{3}$$

   for (a, b) such that the fifth order error vanishes. This is satisfied for

```
improved_result = a*result1 + b*result2
```

   with

$$(a, b) = \left(-\frac{1}{14}, \frac{15}{14}\right).$$

   The step-doubling integrator, `rk4_stepd` calls the function three times for every one time `rk4_step` calls the function. A comparison between the two for the same number of function calls (i.e `200//3` function calls for the step doubling algorithm) is shown in Fig. 1. As expected, the step-doubling algorithm performs a lot better.

2. All intermediate elements in the chain from ^{238}U to ^{206}Pb are assigned a decay rate, τ_i, as given in the slides. A set of ODEs describes the rate of change of the entire population, X_i, by

$$\frac{dX_i}{dt} = \frac{X_{i-1}}{\tau_{i-1}} - \frac{X_i}{\tau_i} \tag{4}$$

   with X_0 and X_{15} as the populations of ^{238}U and ^{206}Pb, respectively. To make eqn. 4 consistent for the initial and final products, $X_{-1} = 0$ and $\tau_{15} \rightarrow \infty$ are chosen. The ODE solver is then called on this set of ODEs with an initial, pure uranium population of 1. An implicit method works best for these sorts of problems and since this system of equations would be classified as stiff, the `Radau` method is used here.

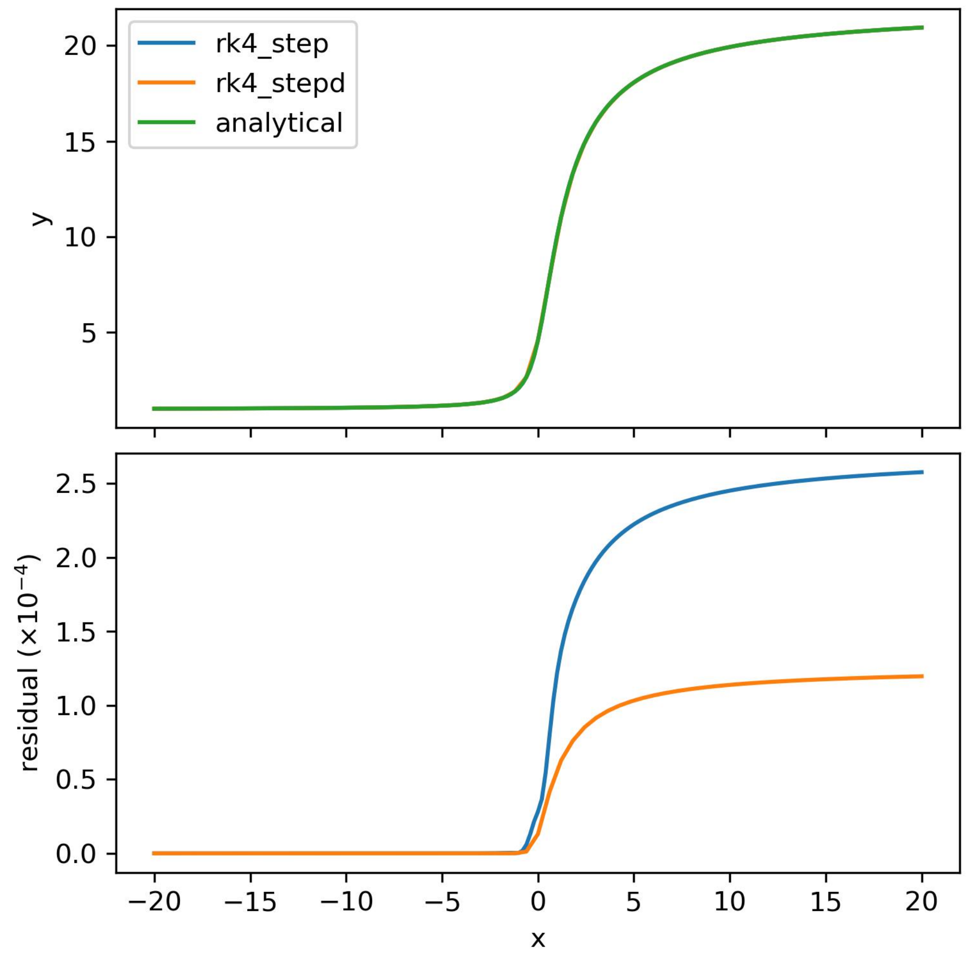

Figure 1: $y(x)$ for the given differential equation with the RK4 integrators and the exact result (above) and the modulus of the residual (below) for the same number of function calls.

```
# tau is a list that stores the decay rates
init = np.zeros(len(tau))
init[0] = 1 # initialize uranium content
# call the ODE solver on written decay fct
# with (t_i, t_f) being initial and final
# times, i.e. (0, 15) in units of tau[0]
# (decay rate in uranium)
soln = integrate.solve_ivp(decay, (t_i, t_f),
            np.asarray(init), method='Radau')
y = soln.y # array of population values
t = soln.t # array of times
```

For plotting purposes, y is then looped over and plotted as function of time, t. The results are shown in Fig. 2. Not surprisingly, in units of τ_0, all of the decaying elements appear almost parallel to the curve of ^{238}U. The population ratio of the two elements can be approximated by solving the differential equation for lead:

$$\frac{X_{15}}{X_0} = e^{t/\tau_0} - 1 \tag{5}$$

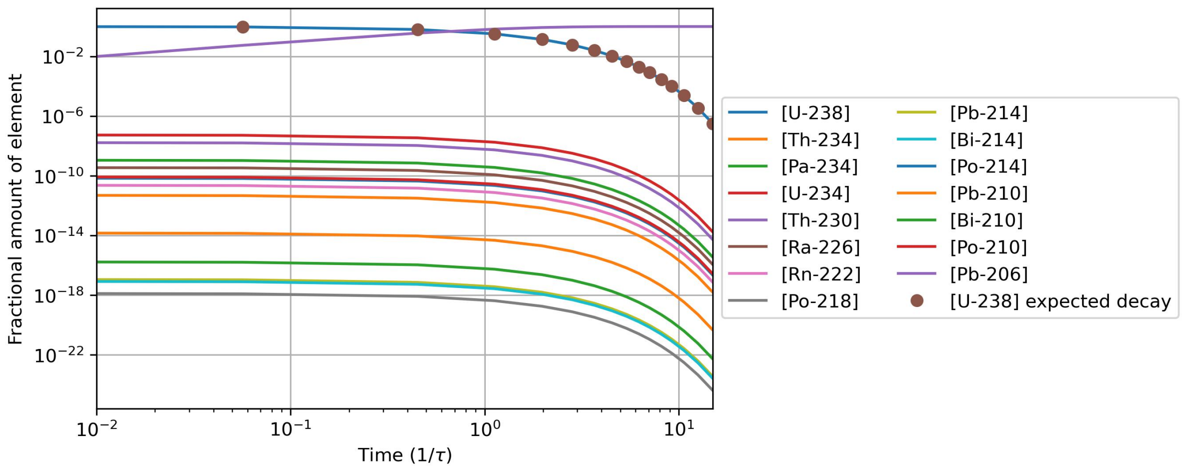

Figure 2: All constituents of the decay chain from ^{238}U $\longrightarrow$ ^{206}Pb shown with fractional amounts in their population w.r.t. ^{238}U. Dotted data shows that ODE solver and theoretical decay model agree. The plot window is enlarged and in units of τ_0 for Uranium.

Fig. 3 shows the population ratio in populations of lead to uranium and compares the result with the approximation, eq. 5. Fig. 4 shows the computed ratio $X_{\text{Th-230}}/X_{\text{U-234}}$. One can observe a steady-state value for long periods of time due to the relative difference in the decay rates of the individual elements. The asymptotic value (for an infinite supply of U-234) is the ratio between the decay rates of the constitutens, i.e.

$$\frac{\tau_{\text{Th-230}}}{\tau_{\text{U-234}}} \approx 0.31 \tag{6}$$

which is also observable in the Figure.

3. (a) The ellipsoid equation can be rewritten in the form

$$z = (z_0 + a(x_0^2 + y_0^2)) - (2x_o a)x - (2y_0 a)y + a(x^2 + y^2). \tag{7}$$

The coefficients can then be lumped together, such that a model vector $\mathtt{m}$ is represented by

$$m = [k_1, k_2, k_3, k_4]^T \tag{8}$$

with $k_1 = (z_0 + ax_0^2 + ay_0^2)$, $k_2 = -2x_0 a$, $k_3 = -2y_0 a$ and $k_4 = a$. Hence, knowing k_4 fixes x_0/y_0, which can subsequently be used to solve for z_0.

(b) The best fit values for k_i are estimated to be

$$\begin{aligned} k_1 &= -1.512 \times 10^3 \\ k_2 &= 4.536 \times 10^{-4} \\ k_3 &= -1.941 \times 10^{-2} \\ k_4 &= 1.667 \times 10^{-4} \end{aligned}$$

The original parameters are then

$$\begin{aligned} a &= 1.667 \times 10^{-4} \\ x_0 &= -1.36 \\ y_0 &= 58.22 \\ z_0 &= -1.51 \times 10^3 \end{aligned}$$

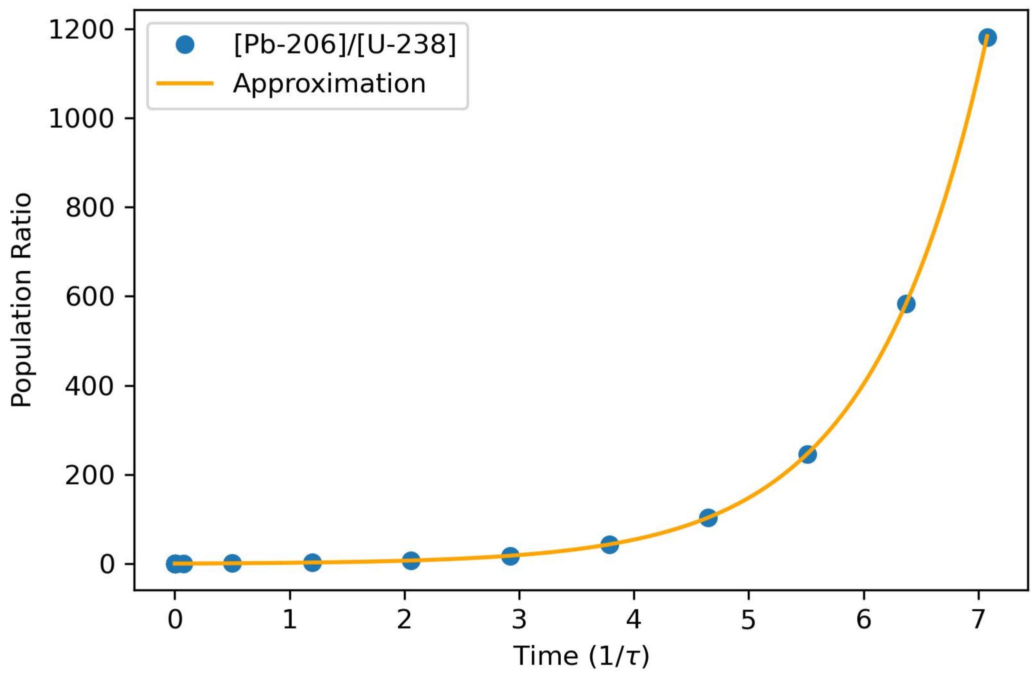

Figure 3: Population ratio between lead and uranium. Line marks approximated values.

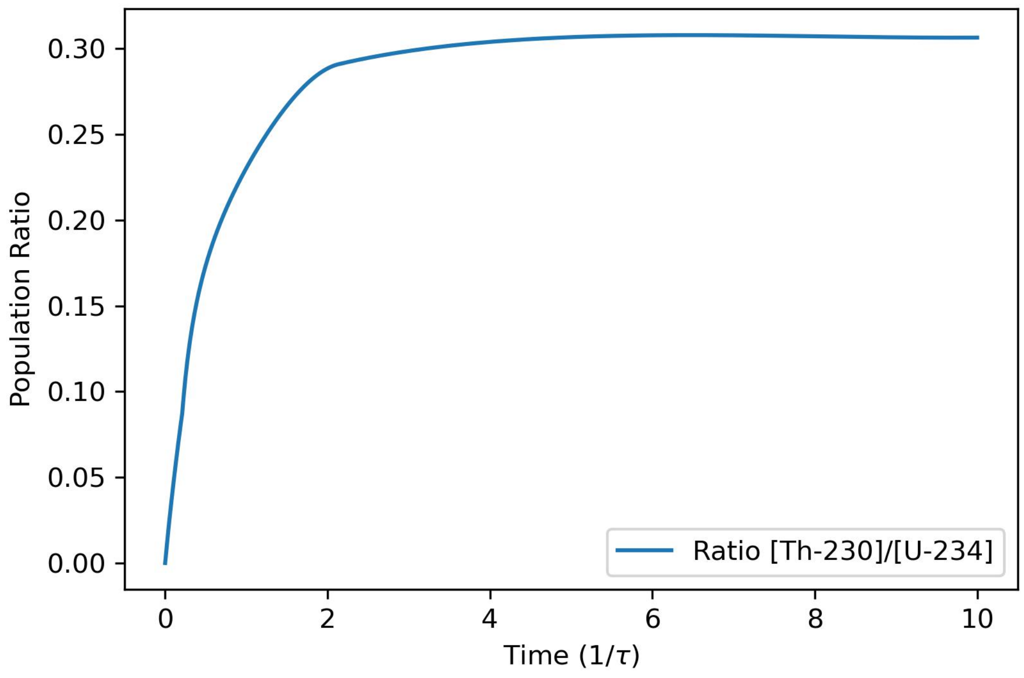

Figure 4: Calculated population ratio in units of τ, where τ is now the decay constant of U-234. Hence the plot window shows a time range on the order of roughly 10^6 years.

Code:

```
import numpy as np
from matplotlib import pyplot as plt

positions = np.loadtxt('dish_zenith.txt')

# define x,y,z values
x = np.zeros(len(positions))
y = np.zeros(len(positions))
z = np.zeros(len(positions))
for i, l in enumerate(positions):
    x[i] = l[0]
    y[i] = l[1]
    z[i] = l[2]

# define model matrix A
# similar to class
A = np.empty([len(x), 4])
A[:,0] = x**0 # first column just 1's
A[:,1] = x # 2nd column x
A[:,2] = y # 3rd column y
A[:,3] = x**2 + y**2 # 4th column sum of squares

# compute matrix products
lhs = A.T@A
rhs = A.T@z
# invert to solve for m
m = np.linalg.inv(lhs)@rhs
pred = A@m
```

(c) The noise in the data can in principle be evaluated using the covariance matrix under the assumption that the fit model closely resembles the true values. Then `n` can be calculated as residual in each data point. The outer product then results in the covariance matrix `N` that then needs to be placed in the matrix product

```
A.T@inv(N)@A.
```

The diagonals will then represent the square of the errors in the model parameters. An estimate can however more easily be obtained by assuming the inverse of the covariance matrix can be written in the form

```
N = np.mean((z-pred)**2)
```

where the product with the unity matrix, $diag(1)$, is implied. It follows that one can write

```
par_errs=np.sqrt(N*np.diag(np.linalg.inv(lhs))).
```

The constant, `N` is pulled out of the matrix equation since it is being multiplied by unity anyway. The root then ensures that one actually obtains the errors in the `m` values, not their squares. Hence, $a = 6.5 \times 10^{-8}$. The focal length in the paraboloid can be estimated from a, using, $f = 1/(4a)$ which results in $1,499.7$mm and is reasonable close to the given focal length value. The uncertainty evaluates to 0.6mm (obtained from the fact that the fractional uncertainty in f and a should be the same).