# ASTP 720 Homework Assignment 3

Isabella Cox – igc5972@rit.edu

September 23, 2020

---

## Task 1

*As promised, unit testing time! Please write a unit test class (e.g., in a file called test_matrix.py if your Matrix class in in a file called matrix.py) for your Matrix class. I highly recommend you use Python's built-in unit test framework, ,though if you want to use just your own assert commands you may do so. Do not overdo it with these, simply compare for each of the above items that you get a sensible result, i.e., you only need to do one test per item above, so don't make 50 tests of whether you have or have not transposed a matrix or not. The goal is to just learn about the practicalities of unit testing and make it easier for you to debug your Matrix class. For this part of the assignment, you may absolutely use numpy or scipy's built-in functionality for the purpose of checking, though if you want to put in 3x3 matrices since you can do the checks by hand, that is of course fine as well.*

Matrix class is in file: `matrix.py`. Unit test class is in file: `test_matrix.py`

Some notes about what I did:

1. Comparing two arrays (or lists of lists) was throwing an error, so I wrote a new function called `assertArrayEqual` that calls `assertIsNone` on `np.testing.assert_equal_array` that returns None if the two arrays are equal.

2. The determinant returned by my class's function and the built in numpy method vary by $5 \times 10^{-8}$ so was returning an error that they were not equivalent. So I used `assertAlmostEqual` and set the tolerance to 5 decimal places, which resolved the issue.

# Task 2

*For a total number density of N= 1cm$^3$ for convenience, calculate the different number densities as a function of temperature T. Make a plot showing the different number densities vary as a function of T. If you're feeling adventures, try to think about the physicality of this plot and it's limitations (not required).*

The physical system can be represented by the following system of equations in matrix form for the 3×3 case:

$$\begin{pmatrix} (B_{12}+B_{13})\bar{J} & -(A_{21}+B_{21}\bar{J}) & -(A_{31}+B_{31}\bar{J}) \\ -(B_{12}\bar{J}) & (B_{21}\bar{J}+A_{21}+B_{23}\bar{J}) & -(B_{32}\bar{J}+A_{32}) \\ -(B_{13}\bar{J}) & -(B_{23}\bar{J}) & (B_{31}\bar{J}+A_{31}+B_{32}\bar{J}+A_{32}) \end{pmatrix} \begin{pmatrix} n_1 \\ n_2 \\ n_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \quad (1)$$

where $A_{ul}$ is the transition probability (per unit time) for spontaneous emission, $B_{ul}\bar{J}$ is transition probability (per unit time) for stimulated emission and $B_{lu}\bar{J}$ is the transition probability (per unit time) for absorption where $\bar{J}$ is the mean intensity of the radiation field.

$A_{ul}$, $B_{ul}$ and $B_{lu}$ can be related to each other by using Equations 2-3. Equation 2,

$$g_l B_{lu} = g_u B_{ul} \quad (2)$$

connects the two $B$ values, linking them with the degeneracy or a given level, so $g_n = 2n^2$. And equation 3 links $A_{ul}$ to $B_{ul}$ as follows,

$$A_{ul} = \frac{2h\nu^3}{c^2} B_{ul} \quad (3)$$

where $\nu$ is the frequency of the emission line and $h$ is Planck's constant.

For populating the array for the 9 × 9 case, I extracted a pattern from the 3 × 3 case. The "rules" are as follows:

1. $r > c : M_{rc} = -B_{cr}\bar{J}$
2. $r < c : M_{rc} = -(B_{cr}\bar{J} + A_{cr})$
3. $r = c;\ r, c > 1 : M_{rc} = -(\sum_{j=1, j\neq r}^{N} B_{rj}\bar{J} + \sum_{j=1, j\neq r}^{r} A_{rj})$
4. $r = c = 1 : M_{11} = B_{12}\bar{J} + B_{13}\bar{J}$

Okay so I spent sooo many hours trying to populate my matrix and I could not do it!! I am pretty confident though, that if were able to get that part working I will be able to solve for the coefficients. I would probably divide everything by $n1$ and then I would get everything in terms of a fraction and use the final constraining equation ($N = n1 + n2 + ....$ to solve for the various coefficients.

# Task 3

*Write an ODE package that includes Euler's method, Heun's method, and the explicit RK4 method.You can set this up in a number of different ways but to simplify things, it should take an initial guess and a list of times to solve for. Note that this is not the same as the step size. Feel free to assume that the function requires user input such as the step size (or also feel free to set it to be an optional argument, e.g.,make each step size 0.1 or 0.01 or 0.001 of the interval), number of Picard iterations, etc.*

I wrote the three required functions and they are in a file: ode_library.py.

# Task 4

*Test your three functions against the damped pendulum example in scipy's `odeint()` documentation. Of course, your functions do not have to receive input in the same way that `odeint()` does. Make some plots to demonstrate that things are working as they should.*

My functions I wrote for ODE solvers accept a function ($\frac{dy}{dt}$) that can either return a value or a list of two values. The guesses should correspondingly either be a list of one value (one variable case) or two values. Figure **??** shows a plot of $\omega$ and $\theta$ as a function of time. Each panel depicts a different ODE solving method's result in solid lines and in dotted lines, is the result from `odeint()` RK4 matches `odeint()`, Euler's does not - probably because of the fixed step size. I am not sure what is happening with Heun's method for this part, as you can see Heun's method was able to work for Task 5.

# Task 5

*Test your three methods against the following stiff ODE and comment on the stability of each:* $\frac{dy}{dt} = -\lambda(t - cos(t))$

For checking, the solution to this ODE is:

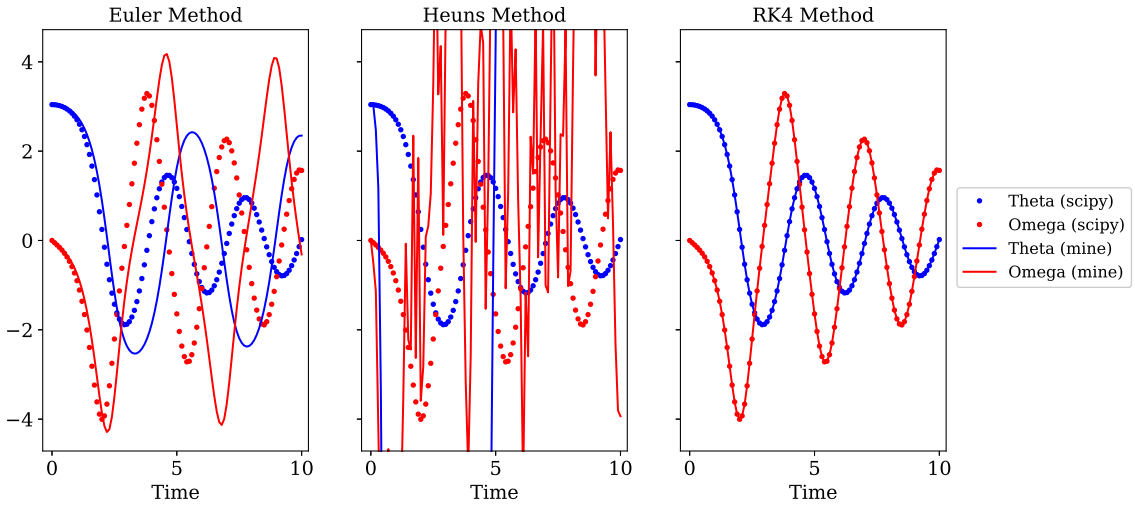


Figure 1: Solution to pendulum ODE is shown for each of the three ODE solvers, one per panel as a solid line with the blue denoting the omega and red denoting the theta. The dotted line in each panel is the plot of the exact solution to the differential equation.

$$y(t) = \frac{\lambda^2}{1+\lambda^2} e^{-\lambda t} + \frac{\lambda}{1+\lambda^2} sin(t) + \frac{\lambda^2}{1+\lambda^2} cos(t), \quad y(0) = 0 \tag{4}$$

and I will choose $\lambda = 12$, so the solution becomes

$$y(t) = \frac{144}{145} e^{-12t} + \frac{12}{145} sin(t) + \frac{144}{145} cos(t), \quad y(0) = 0 \tag{5}$$

Plotting the results from the three methods works out well for the Euler and RK4 cases, but breaks down for the Heun's case. This was using the default step size which is the difference between the current time and the next time entry. To achieve a stable solution, the step size was manually inputted by the user to be a small value. Figure 2 shows the results of the differential equation solutions as found by the three ODE solvers compared to the actual known solution for a $\lambda$ value of 12.

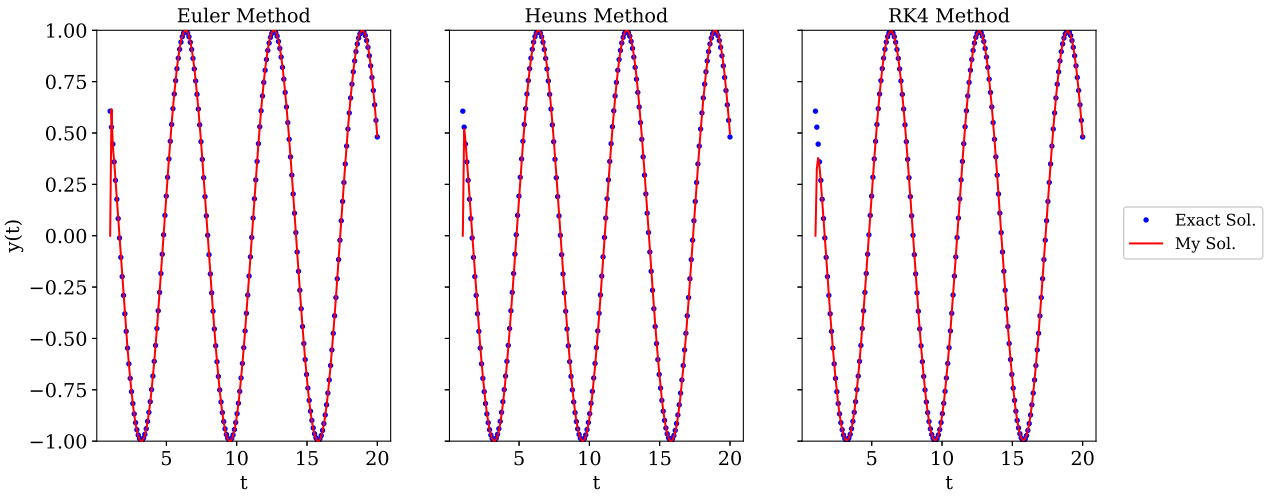


Figure 2: Solution to $\frac{dy}{dt}$ is shown for each of the three ODE solvers, one per panel as a solid line. The dotted line in each panel is the plot of the exact solution to the differential equation.