Computational Physics Weekly Assessments
Week 3 Differential Equations 1

Weekly assessment task & hints

This week you will generate and plot the decay curve for Iodene-133 analytically and numerically. I^{133} has a half-life of 20.8 hours. This task is a little more involved than the last two weeks hence you should put some time in to the assessment before your allocated workshop session – this will help you to get the most from your workshop time. **Study the template solution at the end of this document.**

1. Create a module, named '`cp_3.py`'

2. In the global scope of the module create the following variables, which will be used from within various functions**:**

- `T_HALF` – the half life of the isotope in hours.
- `TAU` – the average lifetime of the isotope in hours, derived from `T_HALF`.

3. Create a function `f(n)` that implements the differential equation for the radioactive decay of 'n' nuclei.

- The function returns the decay rate for n atoms in atoms per hour.
- This function should use the value of the global variable `TAU`.

4. Create a function '`def analytic(N0, ts)`' that takes an initial number of atoms, N0, and a numpy array of times, 'ts' and returns a numpy array of the atom count at each of those times.

- You want an equation that takes N0 and tau and a single time or array of times and calculates the number off atoms at those time(s). This is the analytical solution to radioactive decay presented in the lecture.
- The implementation of this is very similar to assessment 1, part 2
- As with the analytical methods from the last two weeks, this is an equation that you work out 'on paper' and then program in.

5. Create a function '`def solve_euler(N0, dt, n_panels)`' that uses Euler's method to solve the DEQ.

- We are interested in the time interval of 0 <= t < t1. As the initial time is zero we do not explicitly pass it into the function as an argument, unlike last week.
- As with week 2's assessment, divide the time range into a series of `n_panels`. Call the width of a single panel '`dt`'.
- As with week 2, step over each panel applying a timestep to find the new value of N.
- In assessment 2 we just returned the value of the integral. This time we want to look at how this value changes over time – i.e. the decay curve. Allocate a 1d numpy array to store these values, which the function will return.
  E.g. `some_array = numpy.array((n_panels,))`
  After each timestep denoted by i store the current nuclei count at the appropriate point in this array as some_array[i]

6. Create a function '`def solve_heun(N0, dt, n_panels)`' that uses Heun's method to solve the DEQ.

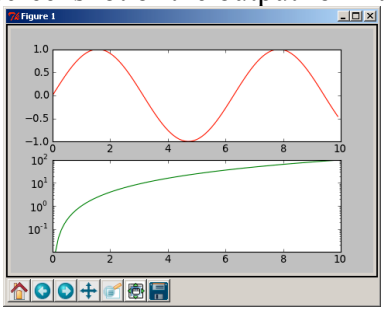
- This function will be very similar to solve_euler, but with a more involved formula (or set of formulas) for making a timestep.
- Finish and test your 'solve_euler' method first – you can then base your Heun code on your tested Euler code.

7. Generate the data to be plotted

- Work in the global scope of your module after the function definitions.
- Create a variable, t1 = 60, the upper time for our integration range.
- Create a variable, N_PANELS = 15, the number of panels we divide our integration range into.
- Create a variable N0 = 1500, the initial number of nuclei at t=0.
- Create an array of N_PANELS timepoints, called 'ts', between 0 and 60 hours
- Call the various functions you have written to generate decay curves over the interval 0 <= t <= t1 and assign these to the variables n_analytic, n_euler and n_heun

8. Plot the data

- This week we are going to plot two graphs on one pyplot figure. This is done with the 'subplot' command.
- See the 'subplot.py' example file attached to the weekly assessment on DUO for more details of using subplots, or see the online matplotlib documentation. The image below is a screenshot of the output form the 'subplot.py' example.



- Divide your figure into two horizontally as in the example above.
- Make the top graph a plot of the decay curves of your three methods against time. Plot the analytic in grey, Euler in red and Heun in blue. Heun's method is sufficiently accurate that the curve should follow a similar trajectory to the analytical solution, so use the additional option `linestyle="--"` when plotting Heun's line to make it dashed – this allows the viewer to see that both curves are coincident.
- Plot the absolute relative error in the two numeric decay curves against time on the bottom figure. Use the formula:

```
error = abs((numeric-analytic)/analytic)
```

9. Answer the question "Why is Heun's method more accurate then Euler's?"

- Place your answer in a variable called 'ANSWER1' in the global scope

10. General issues

- Place the code for parts (7) and (8) above in the global scope
- Place two variables at the start of the code,

    - USER=”your name”
    - USER_ID = “your CIS login”
- Your solution should contain no more than about 80 lines of code and perhaps 10 lines of comments. Excessively long submissions may be penalised.
- A partial example is shown below
- Note the convention that constants, such as T_HALF, are UPPER CASE.

```
from __future__ import division

import numpy
import matplotlib.pyplot as pyplot

USER = 'Emmett Brown'
USER_ID ='DMC-12'

T_HALF = 20.8 # Hours
TAU    = ??? # some func of t_half.  numpy.log(x) works in base e
N0 = 1000     # Initial conditions - number of nuclei
T1 = 60       # integrate over timerange 0 <= t < t1
N_PANELS = 10 # number of panels to divide the time range into

def f(n):
    return ??? # decay rate of n atoms with halflife TAU

def analytic(n0, ts):
    n_analytic = n0 * ??? # some maths involving time

def solve_euler(n0, dt, n_panels):
    # Initialise simulation parameters
    n, t = n0, 0
    # Make an array to hold the counts at each time point in
    n_t = numpy.zeros((n_panels,))
    # Integrate each panel
    for i in range(n_panels):
        n_t[i] = n  # Record current values
        t = i * dt # More accurate than t = t + dt as less rounding errors
        # Mind you, this DEQ doesn't depend on time anyway
        # Calculate next timestep
        n = n + ???  # Euler timestep involving f(n)

    return n_t

def solve_heun(n0, dt, n_panels):
    # A lot like Euler but with a bit more maths
    return n_t

dt = T1 / N_PANELS   # Width of a panel

# Time at the start of each panel - used for plotting & analytical solution
ts = numpy.arange(0, T1, dt)

# Evaluate various methods
n_analytic = analytic(N0, ts)
n_euler    = solve_euler(N0, dt, N_PANELS)
n_heun     = solve_heun(N0, dt, N_PANELS)

# Graphing time
pyplot.figure()
pyplot.subplot(211)   # Top plot - count vs time for methods
???  # Plot number vs time etc.

pyplot.subplot(212)   # Bottom plot - error vs time for numerics
pyplot.semilogy()     # Make y-axis log
err_euler = abs(n_euler-n_analytic)/n_analytic
err_heun  = ???

pyplot.show()
ANSWER1 = ''' would you be prepared if gravity reversed'''
```