Computational Lab 7 : Solving ODEs, part II

Due October 25th, 2019 (@ 17:00)

- Read this document and do its suggested readings to help with the pre-labs and labs.
- This lab's topics revolve around computing solutions to ODEs using various methods to provide constraints on the solution errors, or conserve a physical property of the solution.
- Ask questions if you don't understand something in this background material: maybe we can explain things better, or maybe there are typos.
- Whether or not you are asked to hand in pseudocode, you **need** to strategize and pseudocode **before** you start coding. Writing code should be your last step, not your first step.
- Test your code as you go, **not** when it is finished. The easiest way to test code is with `print('')` statements. Print out values that you set or calculate to make sure they are what you think they are.
- Practice modularity. It is the concept of breaking up your code into pieces that as independent as possible form each other. That way, if anything goes wrong, you can test each piece independently.
- One way to practice modularity is to define external functions for repetitive tasks. An external function is a piece of code that looks like this:

```
def MyFunc(argument):
    '''A header that  explains the function
    INPUT:
    argument [float] is the angle in rad
    OUTPUT:
    res [float] is twice the argument'''
    res = 2.*argument
    return res
```

  In this lab, **you will implement your own integrators** and likely a wrapper to integrate your ODEs. You should be able write the integrator to take any set of ODEs and solve them without changing the integrator code. You will lose 'code' marks if there is excessive copy-pasting of code!

Physics Background

Some of the discussion below is from the text *Computational Physics* by Giordano and Nakanishi.

Molecular dynamics : For elements such as argon, the interactions at large distances (relative to the atom) are dominated by Van der Waals forces caused by transient electric dipole moments between pairs of atoms. This potential varies as r^{-6} and is attractive. When the atoms get close, a repulsive force increases as their electron clouds begin to overlap. This is commonly approximated by a term that varies as r^{-12} and is repulsive. Adding these 2 potentials yields the 'Lennard-Jones potential'

$$V(r) = 4\epsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right] \tag{1}$$

If the argon molecules are not subjected to any other forces the force between any two particles can be calculated from the *interaction potential* $V(r)$ given the separation r between the particles.

Time-independent Schrödinger equation (for Q3). We've treated this system using other methods in previous work. In Q3 you are asked to adapt the shooting code used in Exercise 8.9 to a couple of new cases. Instead of an infinite potential barrier, you are solving the problem in a harmonic or anharmonic well. To solve it, Newman suggests that you set the boundaries of the problem far from the origin $x = 0$ and that at those boundaries you set ψ to zero. For the harmonic potential, you can find exact solutions at http://hyperphysics.phy-astr.gsu.edu/hbase/quantum/hosc5.html to check your numerical solutions. These solutions show that the wave function rapidly decays away from the origin, justifying Newman's suggestion.

Belousov–Zhabotinsky reaction: The *Belousov–Zhabotinsky reaction* is a chemical mixture which, when heated, undergoes a series of reactions that cause the chemical concentrations in the mixture to oscillate between two extremes. You can add an indicator dye to the reaction which changes color depending on the concentrations and watch the mixture switch back and forth between two different colors for as long as you go on heating the mixture. (Videos of the reaction can be found on YouTube, e.g. https://www.youtube.com/watch?v=PpyKSRo8Iec).
Phuysicist Ilya Prigonine formulated a mathematical model of this type of chemical oscillator, which called the ""Brusselator". The equations for this reaction are

$$\begin{aligned} \frac{dx}{dt} &= 1-(b+1)x+ax^2y, \\ \frac{dy}{dt} &= bx-ax^2y. \end{aligned} \tag{2}$$

Here x and y represent concentrations of chemicals and a and b are positive constants. These are *not the same* `a` *and* `b` *constants found in the bulirsch.py found in the textbook.*

Computational Background

Runge-Kutta: The Runge–Kutta fourth order method (RK4) is a widely used algorithm for solving systems of ODEs. In general it is much more accurate than the Euler or Euler–Cromer methods for ODEs used in initial value problems.
The RK4 method involves using the model function to predict the next timestep in the ODE (as in the Euler method) but using those predictions *not* as final value, but as a way to improve the prediction using intermediate values to better approximate the ODE evolution.
As discussed in the text, this method involves calculating the RHS vector of $d\vec{r}/dt = \vec{f}(\vec{r},t)$ at various intermediate points between steps. Full implementation requires coding the following 5 lines which are iterated over t values:

$$\vec{k}_1 = h\vec{f}(\vec{r},t) \tag{3}$$
$$\vec{k}_2 = hf(\vec{r}+\frac{1}{2}\vec{k}_1, t+\frac{1}{2}h) \tag{4}$$
$$\vec{k}_3 = hf(\vec{r}+\frac{1}{2}\vec{k}_2, t+\frac{1}{2}h) \tag{5}$$
$$\vec{k}_4 = hf(\vec{r}+\vec{k}_3, t+h) \tag{6}$$
$$\vec{r}(t+h) = \vec{r}(t)+\frac{1}{6}(\vec{k}_1+2\vec{k}_2+2\vec{k}_3+\vec{k}_4) \tag{7}$$

Verlet algorithm: For equations that follow a structure similar to $F = ma$ the Verlet method provides an accurate and *conservative* method, useful for ensuring energy or momentum conservation. The algorithm is described in the text on pages 371–373. Starting from $a = F/m$, or equivalently,

$$\frac{d^2\vec{r}}{dt^2} = \vec{f}(\vec{r},t) \tag{8}$$

and denoting the velocity as $\vec{v}$, the Verlet algorithm works as follows.

For the first step only, start with an Euler step,

$$\vec{v}(t+\frac{1}{2}h) = \vec{v}(t) + \frac{1}{2}h\vec{f}(\vec{r}(t),t), \tag{9}$$

then repeatedly apply the equations

$$\vec{r}(t+h) = \vec{r}(t) + h\vec{v}(t+\frac{1}{2}h), \tag{10}$$

$$\vec{k} = h\vec{f}(\vec{r}(t+h), t+h), \tag{11}$$

$$\vec{v}(t+h) = \vec{v}(t+\frac{1}{2}h) + \frac{1}{2}\vec{k}, \tag{12}$$

$$\vec{v}(t+\frac{3}{2}h) = \vec{v}(t+\frac{1}{2}h) + \vec{k}. \tag{13}$$

This last equation for $\vec{v}(t+\frac{3}{2}h)$ becomes $\vec{v}(t+\frac{1}{2}h)$ for the next iteration of Eq. 10. Equation (10), which calculates $\vec{v}(t+h)$, is not required for the timestepping. It is useful if you want to diagnose the velocity at the same time as the $\vec{r}(t+h)$ for the purpose of, for example, calculating the energy.

The shooting method for boundary value problems (and the secant method for finding roots) (for Q3): The shooting method is an application of root finding for boundary value problems. A typical application is to look for correct values of parameters that lead to functions on the domain that match a given set of boundary conditions. You modify your guess until you get one that works. To do so, you typically have to solve a nonlinear equation for a root using methods such as the secant method.
(The secant method is a version of Newton's method where you replace the derivative with its finite-difference approximation (compare Newman (6.96) to (6.104)). You use it instead of Newton's method when you do not have an analytic expression for the function you are trying to find the root of, or its derivative.)
In Example 8.9, the shooting method is used with RK4 to find the ground state energy in a square well potential for the time independent Schrödinger equation. Look over the code and see if you can understand what it's doing: it is integrating a pair of ODEs (see the `solve(E)` function, which calls the function `f(r, x, E)`, which in turn calls the function `V(x)`). The solution to these equations depends on the parameter `E`. In the secant method loop, `E` is adjusted until a root is found for the variable `psi`, which the physics in the problem requires to be zero. Stated another way, the program does the following:

- Initializes `E` with reasonable guesses.
- Finds how `psi` depends on `E` using `solve(E)`.
- Adjusts `E` using the secant method.
- Repeats until `E` has converged to within a target accuracy.

Adaptive RK4 steps: (based on section 8.4 of the textbook) The adaptive Runge-Kutta method works by monitoring the estimated error in the integration and adjusts the step size to keep the error close to the required tolerance. Schematically you would

- Calculate **2** RK4 steps of size h,
- Calculate **1** RK4 steps of size $2h$,
- Estimate the error knowing that

$$x(t+2h) = x_h + 2ch^5, \tag{14}$$

$$x(t+2h) = x_{2h} + c(2h)^5, \tag{15}$$

$$\epsilon = ch^5 = \frac{1}{30}(x_h - x_{2h}), \tag{16}$$

where x_h is the value of x calculated with stepsize h, c is and unknown constant.

- Use this estimate for the error to derive a new step–size h

$$\epsilon' = ch'^5 \tag{17}$$
$$= ch^5\left(\frac{h'}{h}\right)^5 \tag{18}$$
$$= \frac{1}{30}(x_h - x_{2h})\left(\frac{h'}{h}\right)^5. \tag{19}$$

  Rearranging for h' we get

$$h' = h\left(\frac{30h\epsilon'}{|x_h - x_{2h}|}\right)^{1/4}. \tag{20}$$

  (If the error per unit interval δ is needed, then $h'\delta = \epsilon'$)

- Update the step size for the calculation and repeat. If the ratio of $h'/h < 1$ our new step is smaller than the current step, meaning we missed the target accuracy and **should go back to time t and repeat the calculation**.

Recursion in the Bulirsch–Stoer method: One way to simplify the Bulirsch–Stoer equation is to make use of recursion. Write a user–defined function called, say `step(r,t,H)` that takes as arguments the vector $\mathbf{r} = (\mathbf{x}, \mathbf{y})$ at time t with an interval length H and returns the vector at time $t + H$. This `step` function should use the 'modified midpoint extrapolation' (section 8.5.5) until the calculation converges to the desired accuracy or you exceed a threshold for the number of steps. If you exceeded the number of steps you should break the problem into smaller steps $H/2$ and try again with two calculations

```
r1 = step(r,t,H/2)
r2 = step(r1,t+H/2,H/2)
```

Correctly written, this function will call itself to sub–divide the domain as needed to improve the accuracy of your solution

Questions

1. **[30%] Molecular dynamics of argon atoms**

   Consider a 2-dimensional molecular dynamics simulation of $N = 2$ molecules under the influence of only the Lennard-Jones potential (Eq. 1 above).

   (a) Find expressions for the x and y components of the acceleration of one particle due to the presence of the other particle. You can use $\sigma = 1$ and $\epsilon = 1$ and also assume units where the masses of the molecules are 1.0.

   NOTHING TO SUBMIT

   (b) Write pseudocode and then a program that updates the position of the 2 particles using the Verlet method. Use a time step of $dt = 0.01$ and run the simulation for 100 time steps. Set all initial velocities to 0.0. Plot the trajectories of both particles on the same plot for the following sets of initial conditions:

   1. $\mathbf{r_1} = [4,4]$, $\mathbf{r_2} = [5.2,4]$
   2. $\mathbf{r_1} = [4.9,4]$, $\mathbf{r_2} = [5.1,4]$
   3. $\mathbf{r_1} = [2,3]$, $\mathbf{r_2} = [7,6]$

   Ensure that you are getting the proper behaviour in each case (e.g. repulsion when the particles are close, attraction when they are far). NOTE: In your trajectory plots, don't use solid lines to join the points, instead use a marker like a dot. You can do this in your plot command as follows:

   ```
   plot(x,y,'.')
   ```

SUBMIT YOUR PSEUDOCODE AND CODE. SUBMIT THE PLOTS FOR EACH SET OF INITIAL CONDITION

(c) One or more of the above initial conditions leads to oscillatory motion for both the particles. Which case is it, and can you explain why? Hint: think about energy conservation.

SUBMIT YOUR WRITTEN ANSWER

2. **[30%] Time independent Schrödinger equation.**

Consider the one-dimensional, time-independent Schrödinger equation in a harmonic (i.e., quadratic) potential $V(x) = V_0 x^2/a^2$, where V_0 and a are constants.

(a) Write down the Schrödinger equation for this problem and convert it from a second-order equation to two first-order ones, as in Example 8.9. Write a program, or modify the one from Example 8.9, to find the energies of the ground state and the first two excited states for these equations when m is the electron mass, $V_0 = 50\,\text{eV}$, and $a = 10^{-11}\,\text{m}$. Note that in theory the wavefunction goes all the way out to $x = \pm\infty$, but you can get good answers by using a large but finite interval. Try using $x = -10a$ to $+10a$, with the wavefunction $\psi = 0$ at both boundaries. (In effect, you are putting the harmonic oscillator in a box with impenetrable walls.) The wavefunction is real everywhere, so you don't need to use complex variables, and you can use evenly spaced points for the solution—there is no need to use an adaptive method for this problem.

The quantum harmonic oscillator is known to have energy states that are equally spaced. Check that this is true, to the precision of your calculation, for your answers. (Hint: The ground state has energy in the range 100 to 200 eV.)

SUBMIT YOUR CODE, AND ENERGY LEVELS OF THE FIRST 3 STATES.

(b) Now modify your program to calculate the first three energies for the anharmonic oscillator with $V(x) = V_0 x^4/a^4$, with the same parameter values.

SUBMIT ANY NEW CODE, AND ENERGY LEVELS OF THE FIRST 3 STATES.

(c) Modify your program further to calculate the properly normalized wavefunctions of the anharmonic oscillator for the three states and make a plot of them, all on the same axes, as a function of x over a modest range near the origin—say $x = -5a$ to $x = 5a$.

To normalize the wavefunctions you will have to evaluate the integral $\int_{-\infty}^{\infty} |\psi(x)|^2\,dx$ and then rescale ψ appropriately to ensure that the area under the square of each of the wavefunctions is 1. Either the trapezoidal rule or Simpson's rule will give you a reasonable value for the integral. Note, however, that you may find a few very large values at the end of the array holding the wavefunction. Where do these large values come from? Are they real, or spurious?

One simple way to deal with the large values is to make use of the fact that the system is symmetric about its midpoint and calculate the integral of the wavefunction over only the left-hand half of the system, then double the result. This neatly misses out the large values.

SUBMIT THE PLOTS OF THE WAVEFUNCTIONS FOR BOTH THE HARMONIC AND ANHARMONIC OSCILLATORS.

3. **[40%] Oscillating chemical reactions** (Exercise 8.18, with notes from 8.17)

(a) Write a program to solve the equations for the Belousov–Zhabotinsky reaction in equation 2, using the adaptive Bulirsch–Stoer method, to an accuracy of $\delta = 10^{-10}$ per unit time in both x and y.

SUBMIT YOUR CODE

(b) Calculate a solution from $t = 0$ to $t = 20$ with $a = 1$ and $b = 4$. Initialize your code with a single time interval of size $H \approx 20$ to ensure you cover the entire time period in the integration. Allow a maximum of $n = 8$ modified midpoint steps in an interval before you divide in half and try again.

NOTHING TO SUBMIT

(c) Plot the solutions for x and y, both on the same graph, and add points to the line–plot to show where your calculations occurred. You should find that the points are significantly closer together where the variables are changing rapidly.

SUBMIT YOUR PLOT

You don't need to try the adaptive RK4 method if the adaptive Bulirsch–Stoer is working, but I tried both with the same tolerance and found similar results from both. However the RK4 method took 1572 steps (in 0.21 seconds) and the Bulirsch–Stoer method took 147 steps (in 0.2 seconds). The plots look the same but there is slightly more 'noise' in the RK4 method. — CL

Question:	1	2	3	Total
Points:	30	30	40	100