Lab assignment #7: Ordinary Differential Equations, Pt. II

Instructor: Nicolas Grisouard (nicolas.grisouard@utoronto.ca)

Due Friday, October 30th 2020, 5 pm

First, try to sign up for room 1. If it is full, sign up for room 2 (and ask your partner to join you there if they are in room 1).

Office hours will happen in room 1 for both TAs.

Room 1 (office hours room) Mikhail Schee (Marker, mikhail.schee@mail.utoronto.ca),
URL: `https://gather.town/ZdDgdR4j2gGZIqvc/MS-PHY407`
PWD: `phy407-2020-MS`

Room 2 Alex Cabaj (alex.cabaj@mail.utoronto.ca),
URL: `https://gather.town/2d25ninrKWziTcd4/PHY407AC`
PWD: `phy407-2020-TA-alex`

General Advice

- **Work with a partner!**
- 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, coupled or not, using adaptive-steps Runge-Kutta and Bulirsch-Stoer algorithms for IVPs, and shooting methods for BVPs.
- Ask questions if you don't understand something in this background material: maybe we can explain things better, or maybe there are typos.
- Specific instructions regarding what to hand out are written for each question in the form:

  THIS IS WHAT IS REQUIRED IN THE QUESTION.

  Not all questions require a submission: some are only here to help you. When we do though, we are looking for "C^3" solutions, i.e., solutions that are **C**omplete, **C**lear and **C**oncise.
- An example of **C**larity: make sure to label all of your plot axes and include legends if you have more than one curve on a plot. Use fonts that are large enough. For example, when integrated into your report, the font size on your plots should visually be similar to, or larger than, the font size of the text in your report.
- 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
```

Place these functions in a separate file called e.g. `functions_labNN.py`, and call and use them in your answer files with:

```
import functions_labNN as fl  # make sure file is in same folder
ZehValyou = 4.
ZehDubble = fl.MyFunc(ZehValyou)
```

Computational Background

Varying step size Choosing your step size (h in the RK4 formulas) is always a balance between speed and accuracy. If your right-hand-side vector $\vec{f}$ varies rapidly, it is usually very beneficial to alter the step size during the calculation to accommodate this.

The basic philosophy in choosing a step size is that we want the error at each step to be similar. If the function is changing slowly, then in those regions we can take bigger steps. If the function is changing rapidly, then in those regions we should take smaller steps. A code that does this implements an "adaptive step size" method.

In order to ensure we are keeping the error roughly constant at each step, we need to be able to estimate the error at each step. If the error is larger (smaller) than we want, then we decrease (increase) the step size accordingly. Section 8.4 in the text discusses how to implement an adaptive step size in RK4.

Bulirsch-Stoer In Q2, we will focus on the **Bulirsh-Stoer** method (8.5.5–8.5.6 of the text). It is a combination of the modified midpoint method and Richardson extrapolation. Although it is somewhat more complicated to program than the Runge-Kutte method, it can work significantly better even than the adaptive version of Runge-Kutta, giving more accurate solutions with significantly less work as long as your solutions are relatively smooth. Lectures from week 7 will provide the details on how this method works and the provided program `bulirsch.py` (Example 8.7 of the textbook) gives a demonstration of its use for the nonlinear pendulum.

The shooting method for boundary value problems 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.

As a reminder, 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. We'll talk about the physics of this problem in the Physics Background. But let's look over the code now and see if we 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.

In the Physics background, we'll discuss why `psi` should be zero.

Scipy constants Again, you can use `scipy.constants` if you want to use actual values of the physical constants we will use. If you dig into that package, the amount of information is quite extensive. The constants you would use in this problem would be:

```
import scipy.constants as pc
a = pc.physical_constants['Bohr radius'][0]
E0 = pc.physical_constants['Rydberg constant times hc in eV'][0]
pc.m_e  # electron mass
pc.hbar
pc.e  # elementary charge
pc.epsilon_0
```

Note the `[0]` indexing is for the value. The second index would be the unit, the third index the precision, which we won't care about today.

Again, completely optional, but cool.

Physics Background

The Hydrogen atom The time-independent Schrödinger equation is

$$-\frac{\hbar^2}{2m}\nabla^2\psi(\boldsymbol{x}) + V(\boldsymbol{x})\psi(\boldsymbol{x}) = E\psi(\boldsymbol{x}) \tag{1}$$

Where $\boldsymbol{x}$ is the spatial coordinate. For a central potential $V = V(r)$, we can assume the wavefunction ψ can be separated according to $\psi(\boldsymbol{x}) = R(r)Y_\ell^m(\theta,\phi)$, where we are using standard notation

with Y_ℓ^m being the spherical harmonic of degree m and order ℓ. In this case $R(r)$ satisfies a second order ODE

$$\frac{\mathrm{d}}{\mathrm{d}r}\left(r^2\frac{\mathrm{d}R}{\mathrm{d}r}\right) - \frac{2mr^2}{\hbar^2}\left[V(r) - E\right]R = \ell(\ell+1)R \tag{2}$$

and is subject to certain boundary conditions. For the particular case of a hydrogen atom with an electron interacting electrostatically with a proton, we can write

$$V(r) = -e^2/(4\pi\epsilon_0 r), \tag{3}$$

and in that case the solutions are well known. An online reference for the solution can be found below:

http://hyperphysics.phy-astr.gsu.edu/hbase/quantum/hydwf.html

http://hyperphysics.phy-astr.gsu.edu/hbase/hyde.html

The energy levels are

$$E_n = -E_0/n^2,$$

where $E_0 \approx 13.6\,\mathrm{eV}$ is the ionization energy of hydrogen and $n \in \mathbb{N}^*$. The boundary condition is that $R(r) \to 0$ as $r \to \infty$. The ground state energy with $n = 1$ is about -13.6 eV, the first excited state energy for $n = 2$ is at -3.4 eV, etc. The wave functions corresponding to $n = 1, 2, 3$ and various values of ℓ can be found at the links above. This problem is well known and solved, and so provides a good test case for our home-built shooting method. We will focus only on the radial part $R(r)$ but if you want you can also create plots of the three dimensional wave function of the different energy levels of the hydrogen atom.

Questions

1. **Return of the space garbage (25% of the lab)** Your model of space garbage (Lab #6, Q1) will come in handy again in this question.

   (a) Redo the question but code it up using an adaptive step size approach (you can take the initial h to be $h = 0.01$, i.e., the h of Lab #6). To examine the impact of adaptive step size, overlay your new solution on the one from Lab #6, Q1, which you will have used with $N = 10{,}000$ time steps, by plotting individual points of the adaptive step size algorithm at each time step. For example `plot(x, y, 'k.')` would plot black points. Using a target error-per-timestep $\delta = 10^{-6}$ for the position (x, y) seems to work well. When calculating ρ, use the formula

   $$\rho = \frac{h\delta}{\sqrt{\epsilon_x^2 + \epsilon_y^2}},$$

   which uses the Euclidean error for the position in the (x, y) plane, $\sqrt{\epsilon_x^2 + \epsilon_y^2}$ as described in the text on pp. 359-360 (see the discussion around eqn. 8.54). Do you see the effect of the adaptive time step?

   SUBMIT PLOT AND A SHORT DESCRIPTION OF WHAT YOU SEE.

(b) Once you have convinced yourself that the code is working, compare the clock time it takes to the clock time of the original solution from Lab 6. Compare the $\delta = 10^{-6}$ solution to the time taken for the non-adaptive time step with $h = 0.001$ ($N = 10,000$).

SUBMIT PRINTED OUTPUT OR A SHORT ANSWER.

(c) Provide a plot of the size of the time step as a function of time. Here's a simple way to do this: if your time values are in the list called `tpoints`, you can plot the time step array as follows

```
dtpoints = array(tpoints[1:]) - array(tpoints[:-1])
plot(tpoints[:-1], dtpoints)  # drop the last point in tpoints
```

Optionally, you can drop the first several points in your plot because it takes a little while for the adaptive routine to settle down into predictable behaviour.

Try to relate the time step size to the solution. Under what circumstances do the time steps tend to be relatively short or relatively long?

HAND IN THE CODE, YOUR PLOT(S), AND WRITTEN ANSWERS.

2. **Bulirsch-Stoer method: Earth and Pluto's orbits — Newman Exercise 8.13 (25% of the lab)** This exercise asks you to calculate the orbits of two of the planets using the Bulirsch–Stoer method. It refers to previous exercise 8.12, which uses the Verlet method to compute Earth's orbit, though you do not need to have done it. For you to check that your solution for part (a) makes sense however, I uploaded a solution to Exercise 8.12(a) as `Newman-812a.py`, along with the present document, though we do not ask for any mention of it in your report.

   (a) Do part (a).

   SUBMIT PLOT AND EXPLANATORY NOTES.

   (b) Do part (b).

   SUBMIT PLOT, CODE AND EXPLANATORY NOTES.

3. **The hydrogen atom (50% of the lab)** In this exercise you will use shooting and RK4 to find the bound states of hydrogen for a couple of cases. The calculation will involve finding both the energy eigenvalues and the related eigenfunctions for the radial part of the Schrödinger equation. In your calculation, you will need to set initial conditions on $R(r = 0)$ and an intermediate function, say, $S(r = 0)$. Because the RHS of (2) diverges at $r = 0$, you should carry out your integration starting at $r = h$, where h is the step size, and set $R(h) = 0$ and $S(h)$ to a constant (1 is fine). This will cause the eigenfunction to go to zero at $r = 0$, which, it turns out, is not a good assumption for one of the cases below, but will not affect the energy eigenvalue calculation too much.

   Now do the following:

   (a) Starting from (2), and with reference to Section 8.6.3, Example 8.9, and Newman's code `squarewell.py` online, write out the second order ODE in r as a pair of coupled first order ODEs for R and S, and implement this in Python. Some notes:

- In the secant method loop in Example 8.9, the energy `E` is adjusted until the value `psi` is as close to zero as possible because `psi` represents the wavefunction at the right-hand boundary of the infinite square well. In the hydrogen atom there is no right wall; instead, your domain extends to "infinity", but you can get reasonable answers for a large value of r, and you need to set the wavefunction R to zero there. This is similar to the issues raised in Exercise 8.14 in Newman.
- There are a few adjustable parameters in this exercise: one is the stepsize h and the other is the maximum value of r, which is r_∞. To start with, set $h = 0.002a$ and $r_\infty = 20a$, where $a \approx 5 \times 10^{-11}$ m is the Bohr radius. You can make r_∞ larger and make h smaller to improve the solution. You will need to find out which one makes more of a difference in different situations.
- You will need to set a left boundary condition on R near $r = 0$. For convenience, set $R(h) = 0$, even though this is not required by the mathematics and indeed contradicts the solution for $n = 1$, $\ell = 0$. This is a source of inaccuracy in the code.
- You will also need to set a target energy convergence. This is `e/1000` in `squarewell.py` but you should look at the impact of reducing this.
- Finally, you will need to initialize your eigen energies at some level for the secant method to work. It is easy to miss the energies if you aren't careful. It can be a good idea to bracket the energies, so, for different `n`, you can choose initial values of `E1=-15*e/n**2` and `E2=-13*e/n**2`.

NOTHING TO SUBMIT.

(b) Calculate numerically the ground state energy ($n = 1$) and the first excited state energy ($n = 2$) for $\ell = 0$, and the energy for $n = 2$ and $\ell = 1$ (which is supposed to be the same as the energy for $n = 2$ and $\ell = 0$). These can all be compared to the known solution for the hydrogen atom. Look at the effect of adjusting the various parameters discussed above.

HAND IN WRITTEN ANSWERS.

(c) For the three cases in Part (b), modify your program to plot the normalized eigenfunction R. Normalize by calculating $\int |R(r)|^2 dr$ over the range of r and plot it as a function of r over a finite range (this is similar to Exercise 8.14(c)). You can use either the trapezoidal or Simpson's rule to do the integration.

For some eigenvalues, you might obtain spurious large values of R at the right-hand end (r_∞) of the wave function. By definition, the wave function should go to 0 at the boundary. It does not because we don't have the energy **exactly** right and it's extremely sensitive to the energy value. If the energy is off by even the tiniest bit then the value tends to diverge. This is not a problem for determining the energy itself. But it can give problems when normalizing and potting the wave function.

NOTHING TO SUBMIT.

(d) At the links above you can find explicit solutions for $R(r)$. Scale these solutions so they can be plotted as overlays on top of your numerically calculated solutions. How do

the two solutions compare in terms of overall shape and zero crossings for the wave functions?

HAND IN CODE, PLOTS AND WRITTEN ANSWER.