PHY407 Computational Lab 2
Numerical Errors and Simple Integration Techniques

General Advice

- Read this document and do its suggested readings to help with the pre-labs and labs.
  - Ask questions if you don't understand something in this background material - maybe we can explain things better, or maybe there are typos.
- Carefully check what you are supposed to hand in for grading in the section "Lab Instructions".
- 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.

Computational Background

- **Numerical (roundoff) error (for Q1):** Read Section 4.2 of Newman's textbook, which discusses characteristics of machine error. One important point is that you can treat errors on numerical calculations as random and independent.[1] As a result of this, you can use standard error statistics as in experimental physics to figure out how error propagates through numerical calculations. This results in expressions like (4.7) in the text, describing the error on the sum (series) of N terms $x = x_1 + \ldots + x_N$:

$$\sigma = C\sqrt{N}\sqrt{\overline{x^2}}, \tag{1}$$

where $C \approx 10^{-16}$ is the *error constant* defined on p.130 and the overbar indicates a mean value. This means that the more numbers we include in a given series, the larger the error (by $O(\sqrt{N})$). Even if the mean error is small compared to the individual terms the *fractional error*

$$\frac{\sigma}{\sum_i x_i} = \frac{C}{\sqrt{N}}\frac{\sqrt{\overline{x^2}}}{\overline{x}} \tag{2}$$

can be really large if the mean value $\overline{x}$ is small, as is the case when we sum over large numbers of opposite sign.

[1] This is a little confusing because you will find that errors on a given computer are often *reproducible*: they'll come out the same if you do the calculation the same way on your computer. But there is typically no way to predict what the error is — for the same operations it could be different on different computers, or even if you do a software update on your computer.

- **Solving integrals numerically (for Q2 and Q3).** Read Sections 5.1 - 5.3 of the text, which introduce the Trapezoidal Rule and Simpson's Rule for solving integrals numerically, then explain the errors in these methods and discusses practical issues in choosing your step size for integration. The online resource for the text provides the python program `trapezoidal.py`. Also check out Lecture 2 notes for more discussion of the computational topics for this lab.

- **Using available Python subroutines (for Q2 and Q3).** In Q2 you will write code to calculate the mth order Bessel fucntion $J_m(x)$. I will ask you to compare the output of your routine to the `scipy` routine `scipy.special.jv`. The call `jv(m,x)` calculates the mth order Bessell function $J_m(x)$. Also, in Q3 you will compare your answer to a known result that involves the modified Bessel function $K_n(x)$ (`scipy.special.kn`; see Equation (7)) .

  So why are we doing calculations two ways? Often in this course you will be learning how to do numerical calculations that have already been coded by someone else in the Python open-source community. Another example is the `numpy.dot` routine which you can use for matrix multiplication and which we found was a lot faster than term-by-term matrix multiplication (Lab01). We hope and expect that these "canned" routines are fast and accurate, but we can't count on this. It's always a good idea to test your codes against other people's codes and known results.

  Here's an example of how to test for convergence to a known result. "Hint 1" on p.148 says that

$$\lim_{x\to 0} J_1(x)/x = 0.$$

  Some code to test how `scipy.special.jv` can achieve this limit is reproduced below - try running it yourself:

```
from scipy.special import jv
x = 1.0
for k in range(20):
    print 'x, J_1(x)/x from jv: ', x, jv(1, x)/x
    #make a smaller x
    x /= 2.0
```

- **Integrating over infinite limits (for Q3):** In Q3 we will be dealing with a definite integral over an infinite range [see (5) below]. There are different ways to approach improper integrals like this, but one trick is to change variables to make the integral take place over a finite range. Along the lines of equations (5.73)-(5.75) on p.180 of the textbook, you can use:

$$\int_{-\infty}^{\infty} f(x)\,dx = a\int_{-\pi/2}^{\pi/2} \frac{f(a\tan u)}{\cos^2 u} du, \tag{3}$$

  using the change of variables $x = a\tan u$, where a is a constant with dimensions of x. For example, if x is position then a is a length scale. You can then use standard integration techniques (e.g. Trapezoidal Rule or Simpson's Rule) to calculate the integral.

- **Creating a contour plot (for Q3):** The command `pylab.contour` is useful for plotting a function that depends on two coordinates. To learn about using `contour`, import it and type `help(contour)`. Here's some code that you could use to create the contour plot in Q3:

```
#import plotting commands and arange
from numpy import ...
from pylab import contour, ...

#define coordinate values in r and z
r = arange( ...
nr = len(r)
z = arange( ...
nz = len(z)
#define V array which will be calculated as a function of r and z
V = zeros((nz,nr))
#define contour levels
levs = arange(0.5,5.5, 0.5)#chosen contours of electric potential in V
#...calculations
figure(1)
#plot V, converting r and z to mm. Use contour levels defined by levs,
and make the contours black ('k')
c = contour(r*1e3, z*1e3, V, levels = levs, colors = 'k')
#label the contours - c contains the graphical contour objects
clabel(c)
xlabel('r(mm)')
ylabel('z(mm)')
title('V for distributed static charge')
show()
```

Physics Background

- **Electrostatic potential for a line of charge (for Q3):** Consider an infinite line of charge along the z axis with most of the charge concentrated near the origin. The charge per unit length is

$$\lambda(x) = \frac{Q}{l} e^{-z^2/l^2}, \tag{4}$$

where Q and l are constants.

At the position (r, z), where r is the radial coordinate, the electrostatic potential is given by the integral

$$V(r,z) = \int_{-\infty}^{\infty} \frac{\lambda(z')\,dz'}{4\pi\epsilon_0\sqrt{(z-z')^2+r^2}} = \int_{-\infty}^{\infty} \frac{Qe^{-z'^2/l^2}\,dz'}{4\pi\epsilon_0 l\sqrt{(z-z')^2+r^2}},\ r>0 \tag{5}$$

where the integration accounts for all contributions of the infinite line of charge to to the potential at (r, z). This integral is well behaved away from $r = 0$. To calculate it numerically, it is useful to change coordinates in the integrand as suggested by (3) above: the

transformation to use is $z' = l \tan u$ and this results in:

$$V(r,z) = \int_{-\pi/2}^{\pi/2} \frac{Q e^{-(\tan u)^2}\, du}{4\pi\epsilon_0 \cos^2 u \sqrt{(z - l\tan u)^2 + r^2}}, \tag{6}$$

For $z = 0$, a known result is that (5) becomes

$$V(r, z=0) = \frac{Q}{4\pi\epsilon_0 l} e^{r^2/2a^2} K_0(r^2/2a^2), \tag{7}$$

where $K_0(x)$ is a modified Bessel function (in Python it is implemented as `scipy.special.kn(0,x)`). Comparing your solution to this case will be useful to determine the accuracy of your integral calculation.

Lab Instructions

For grading purposes, only hand in solutions to the following parts. Ensure that your codes are well commented and readable by an outsider:

- Q1: Include the plots, any printed output that is necessary to solve the problem, and your written answers. Do not hand in the code.
- Q2: For Q2a-b, include any printed output that is necessary to solve the problem, and your written answers. Do not hand in the code. For Q2c, hand in the code, printed output, written answers, and plots requested.
- Q3: Hand in the code, printed output, written answers, and plots requested.

Lab Questions

1. **Exploring Numerical Error (30% of lab mark).** The idea of this exercise is to explore the effects of roundoff error for a polynomial calculated a couple of ways. Consider $p(u) = (1-u)^8$ in the vicinity of $u = 1$. Algebraically, this is equivalent to the following expansion

$$q(u) = 1 - 8u + 28u^2 - 56u^3 + 70u^4 - 56u^5 + 28u^6 - 8u^7 + u^8. \tag{8}$$

But numerically p and q are not exactly the same.

   (a) On the same graph, plot $p(u)$ and $q(u)$ very close to $u = 1$, for example picking 500 points in the range $0.98 < u < 1.02$. Which plot appears noisier? Can you explain why?

   (b) Now plot $p(u) - q(u)$ and the histogram of this quantity $p(u) - q(u)$ (for u near 1) using the `hist` function in `pylab`. Do you think there is a connection between the distribution in this histogram and the statistical quantity expressed in (1) above? To check, first calculate the standard deviation of this distribution (you can use the `std` function in `numpy`). Then calculate the estimate obtained by using equation (1), with $C = 10^{-16}$. State how you calculated the other terms in (1). *Hint: We are looking for order of magnitude consistency here, do not worry about $O(30-50\%)$ differences.*

(c) From equation (2) above show that for values of `u` somewhat greater than `0.980` but less than `1.0` the error is around 100%. Verify this by plotting or printing out `abs(p-q)/abs(p)` for `u` starting at `u = 0.980` and increasing slowly up to about `u = 0.984` (it might be different on your computer). This fractional error quantity is noisy and diverges quickly as `u` approaches `1.0` so you might need to plot or print several values to get a good estimate of the values of `u` at which the error approaches 100%.

(d) Of course roundoff error doesn't just apply to series, it also comes up in products and quotients. For the same `u` near 1.0 as in Q1a-b, calculate the standard deviation (error) of the numerically calculated quantity `f = u**8/((u**4)*(u**4))`. This quantity will show a range of values around `1.0`, with roundoff error. You can get a sense of the error by plotting `f-1` versus `u`. Compare this error to the estimate in equation (4.5) on p.131 of the text (don't worry about the factor of $\sqrt{2}$).

2. **Learning and applying numerical integration (40% of the lab).** Q2a-b will help you learn the basics of the Trapezoidal Rule and Simpson's Rule, and Q2c develops a physics application.

(a) On page 142, Example 5.1 uses the Trapezoidal Rule to integrate a simple function. The code `trapezoidal.py` is available online. Do Exercise 5.2 parts (a)-(c) on page 147, which asks you to alter the code so that it uses Simpson's Rule instead of the Trapezoidal Rule and then compare results.

(b) On page 155 you will find Exercise 5.6 which is a follow up to Exercise 5.2. Do this exercise.

(c) On page 148, Exercise 5.4 discusses the diffraction limit of a telescope.

- First, do Exercise 5.4a (create a Bessel-function routine in Python and make the requested plot).
- Then make a second plot where you examine the difference between your Bessel functions and those from `scipy.special.jv`. How well do you reproduce the `scipy` routines with your own Bessel function?
- Now do Exercise 5.4b.

3. **Potential of a line of charge (30% of the lab mark)**

(a) Referring to (6), write a code to calculate $V(r, z)$ using Simpson's rule. To make sure you are doing things correctly, you will need to compare your results to the known result in (7) and adjust the integration method until it falls within a desired accuracy. The relevant scipy modified Bessel function call for (7) has the form `scipy.special.kn(0,x)`. First, overlay your calculations of $V(r, z = 0)$ using (7) and using Simpson's Rule to calculate (6) with `N = 8` over the range $0.25\,\text{mm} < r < 5\,\text{mm}$ in a plot. Set $Q = 10^{-13}\,\text{C}$ and $l = 1\,\text{mm}$. You can find the value of ϵ_0 online, in a book, or as follows:

```
from scipy.constants import epsilon_0
```

I found that even with `N = 8`, I got pretty good agreement between the two calculations. Then plot the difference of these quantities and see if you can bring down the fractional error to about one part in a million by increasing `N`.

(b) With the value of `N` from Q3a, create a plot of the potential field in the domain $-5\,\text{mm} < z < 5\,\text{mm}$ and $0.25\,\text{mm} < r < 5\,\text{mm}$. Make sure to label the contours or otherwise indicate the value of V in volts. Make sure to use enough resolution in r and z that you have captured the gradients (related to the electric field) correctly.