PHYS 260 – Programming Assignment 4

1. **Quadratic formula**

We have seen that rounding error can become significant when subtracting numbers that are nearly equal. To learn how to mitigate these errors, you will create a program that uses the quadratic formula to solve for the roots of a second order polynomial.

(a) Write a program that takes as input three numbers, a, b, and c, and prints out the two solutions to the equation $ax^2 + bx + c = 0$ using the quadratic formula:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Use your program to compute solutions of $0.001x^2 + 1000x + 0.001 = 0$.

(b) There is another way to write the solutions to a quadratic equation. Multiplying the top and bottom of the solution above by $-b \mp \sqrt{b^2 - 4ac}$, show that the solutions can also be written as

$$x = \frac{2c}{-b \mp \sqrt{b^2 - 4ac}}$$

Add further lines to your program to print these values in addition to the earlier ones and again use the program to solve $0.001x^2 + 1000x + 0.001 = 0$. Why is this result different than (a)?

(c) Using what you have learned, write a new program that calculates both roots of a quadratic equation accurately in all cases. Hints: Let x_1 be the positive root and x_2 be the negative root. We can check the accuracy of our results by noticing that $x_1 x_2 = c/a$ and $x_1 + x_2 = -b/a$. Accuracy can be improved by avoiding subtracting numbers that are nearly equal.

2. **Electric potential and field of point charges**

Suppose we have a distribution of charges and we want to calculate the resulting electric field. One way to do this is to first calculate the electric potential V and then take its gradient. The potential is usually easier to work with since it is a scalar. For a point charge q at the origin, the electric potential at a distance r from the origin is $V = q/4\pi\epsilon_0 r$ and the electric field is

$$\vec{E} = -\vec{\nabla}V = -\frac{\partial V}{\partial x}\hat{x} - \frac{\partial V}{\partial y}\hat{y} - \frac{\partial V}{\partial z}\hat{z}$$

(a) You have two charges of ± 1 C, 10 cm apart. Calculate the resulting electric potential on a 1 m × 1 m square plane surrounding the charges and passing through them. Calculate the potential at 1 cm spaced points in a grid and make a visualization of the potential using a density plot. Because $1/r$ diverges at $r = 0$, you will need a way to handle the very large values for V that occur near the charges. A good way to handle this is to define a maximum potential value `Vmax` and use the arguments `vmin = -Vmax` and `vmax = Vmax` for the `imshow` function. Using SI units for all quantities, I found `Vmax = 1e10` yields good results.

(b) Calculate the partial derivatives of the potential with respect to x and y to find the electric field in the xy plane. Use central differences for the interior grid points and forward/backward differences for the grid points on the edge. Make a visualization of the field using streamplot and/or quiver. You may get large arrows near the point charges. To mitigate this, define a maximum electric field `Emax` and rescale the electric field so that $E = \sqrt{E_x^2 + E_y^2}$ does not exceed `Emax` at any grid point. I obtained good results using `Emax = 4e10`. You can add circles to your density plot that represent each charge with `plot(x,y, "ro", ms=8)`. Here, `x` and `y` are the coordinates of the point charge, `"ro"` will draw a red circle, and ms=8 is the relative size of the charge.

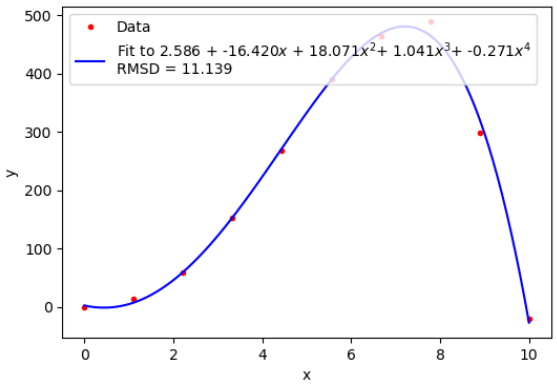
(c) (optional) Modify your code so it plots the potential and field of an arbitrary number of charges. One way to do this is to use a list called, say `qlist`, that holds the (x, y) coordinates and charge of the particle. Then, in your loop over the coordinates of the grid, you will have an inner loop over the charges in `qlist`. Try plotting the potential and field of an electric quadrupole!

3. **Polynomial curve fitting**

Modify your program for polynomial curve fitting so that it calculates the root-mean-squared error ϵ_{rms} of the fit, where

$$\epsilon_{\text{rms}} = \sqrt{\frac{\sum_i \left(y_i - f(x_i)\right)^2}{N}}$$

Here, y_i is the y-value for the ith data point, $f(x_i)$ is the value of the fit function at x_i and N is the number of data points. Have your program generate a plot that graphs the data as points and the fit as a curve. Include axes labels, the value of ϵ and the equation of the fitted line, including the fit parameters to three decimal places. Your code should produce a figure that looks something like the one on the right. To do this, you will need to create a string. To avoid hard-coding this string for every possible polynomial degree, the best way to do this is to create pieces of the string and concatenate them. For example,




```
s = "Fit to %.3f " % a[0]
```

Creates a string with a value of "`Fit to 4.356 `" where 4.35698745 is the value stored in `a[0]`. Notice that the `.3f` causes only 3 decimal places to be displayed. Add to the string with a command:

```
s += "+ %.3f$x$" % a[1]
```

Now `s` will have the value "`Fit to 4.356 + -2.389x`" where `a[1] = -2.3895487325`. The $ signs around x cause matplotlib to display x using TeX markup. You can make the equation appear even more readable with "`$x^2$`". This will be displayed as x^2 (the ^ will superscript what follows). You should put this all in a loop and reference the values of `a` by calling `a[n]` instead of hard-coding explicit values for `n` as in the examples above. You can learn more about writing mathematical expressions here: https://matplotlib.org/users/mathtext.html

Use your program to perform fits up to a 10th degree polynomial to the data in `data.txt` on Canvas. What happens to ϵ_{rms} for the higher-order fits? What polynomial order seems the most reasonable?

4. **Differentiating and integrating noisy data**

There is a file on Canvas called `velocity.txt`, which contains the values of the velocity of a particle in one dimension in units of m/s, measured every 0.1 s. The particle undergoes a constant acceleration of 0.500 m/s^2.

(a) Write a program that loads the data and calculates an array of acceleration values, one for each time value using forward differences for all times except for the last one. Use a backward difference for calculating the acceleration for the last time value. Have your program also calculate an array of position values by using the Trapezoidal Rule, similar to what you did in #1 of Assignment 3. Make a plot of the position, acceleration and velocity on the same graph. What happens to the noise in the velocity data upon differentiation and integration? Explain these results.

(b) Using your array of acceleration values calculate the mean acceleration $\bar{a}$ and standard deviation in the mean acceleration $\sigma_{\bar{a}}$where each of these is given by

$$\bar{a} = \frac{\sum_{i=1}^{N} a_i}{N} \qquad \sigma_{\bar{a}} = \frac{\sigma}{\sqrt{N}} \qquad \sigma = \sqrt{\overline{a^2} - (\bar{a})^2}$$

(c) Because the noise gets worse upon differentiation, your value for $\bar{a}$ has a large uncertainty and doesn't agree very well with the actual value of 0.500 m/s^2. We can obtain better results if we use the position data since the noise gets averaged out upon integration. Modify your program so that it saves a .txt file that contains two columns of data, the first with the time values and the second with the position values. You can do this using:

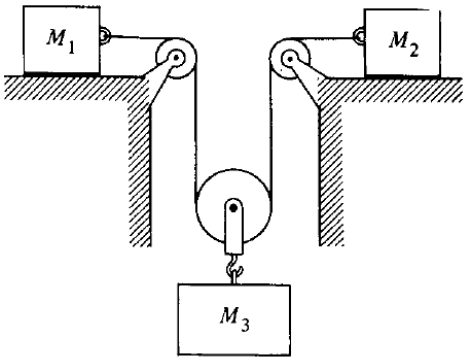
```
savetxt("position.txt", column_stack([t,x]))
```

You will need to import both `savetxt` and `column_stack` from `numpy`. Now import this file into your program for polynomial curve fitting and fit a second order polynomial to this data. Recall that the position of a particle undergoing constant acceleration a is given by

$$x(t) = x_0 + v_0 t + \frac{1}{2} a t^2$$

What is the acceleration of the particle based on your fit parameters?

5. **Pedagogical Machine**



The Pedagogical Machine is depicted to the right. It consists of two masses M_1 and M_2 that lie on a frictionless surface. They are attached to either end of a string of negligible mass of length L that passes around pulleys to a hanging mass M_3. All pulleys are frictionless and of negligible mass. The tension is constant throughout the length of the rope and is denoted by T. The goal of this problem is to solve for the tension in the rope T and the accelerations a_1, a_2, a_3 of the three blocks. Since there are four unknowns, we need four equations to solve the system. Three equations come from Newton's 2nd Law for each block:

1) $F_{\text{net}} = T = M_1 a_1$

2) $F_{\text{net}} = T = M_2 a_2$

3) $F_{\text{net}} = M_3 g - 2T = M_3 a_3$

The fourth equation can be obtained by determining how the accelerations of each block are related. To do this, we can use the fact that the string does not stretch and write the length of the string as:

$L = x_3 - x_1 + x_3 - x_2 = 2x_3 - x_1 - x_2$

where x_1, x_2, and x_3 are the positions of the three blocks. Now we differentiate this expression twice with respect to time to get the fourth equation:

4) $2a_3 - a_1 - a_2 = 0$

Here, we used $d^2L/dt^2 = 0$ since L is a constant and does not change with time.

Write a program to solve this system of equations for the cases below. Put all of this in a loop so that when your program runs, it will calculate each case below. For each case, have your program print the values of a_1, a_2, a_3 and T and a short statement that explains the results.

(a) $M_1 = 2.00$ kg, $M_2 = 5.00$ kg, $M_3 = 1.00$ kg

(b) $M_1 = M_2 = M_3 = 2.00$ kg

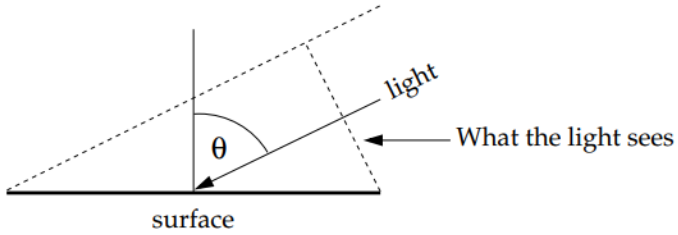
(c) $M_1 = M_2 = M_3 = 4.00$ kg

(d) $M_1 = 0, M_2 = 5.00$ kg, $M_3 = 1.00$ kg

(e) $M_1 = 2.00$ kg, $M_2 = 5.00$ kg, $M_3 = 0$

6. **Image processing and the STM (optional)**

When light strikes a surface, the amount falling per unit area depends not only on the intensity of the light, but also on the angle of incidence. If the light makes an angle θ to the normal, it only "sees" $\cos\theta$ of area per unit of actual area on the surface.



So the intensity of illumination is $a\cos\theta$, if a is the raw intensity of the light. This simple physical law is a central element of 3D computer graphics. It allows us to calculate how light falls on three-dimensional objects and hence how they will look when illuminated from various angles.

Suppose, for instance, that we are looking down on the Earth from above and we see mountains. We know the height of the mounts $w(x,y)$ as a function of position in the plane, so the equation for the Earth's surface is simply $z = w(x,y)$, or equivalently $w(x,y) - z = 0$, and the normal vector $\vec{v}$ to the surface is given by the gradient of $w(x,y) - z$ thus:

$$\vec{v} = \vec{\nabla}[w(x,y) - z] = \begin{pmatrix} \partial/\partial x \\ \partial/\partial y \\ \partial/\partial z \end{pmatrix} [w(x,y) - z] = \begin{pmatrix} \partial w/\partial x \\ \partial w/\partial y \\ -1 \end{pmatrix}$$

Now suppose we have light coming in represented by a vector $\vec{a}$ with magnitude equal to the intensity of the light. Then the dot product of the vectors $\vec{a}$ and $\vec{v}$ is

$$\vec{a} \cdot \vec{v} = |\vec{a}||\vec{v}|\cos\theta$$

where θ is the angle between the vectors. Thus the intensity of illumination of the surface of the mountains is

$$I = \frac{\cos\phi\left(\frac{\partial w}{\partial x}\right) + \sin\phi\left(\frac{\partial w}{\partial y}\right)}{\sqrt{\left(\frac{\partial w}{\partial x}\right)^2 + \left(\frac{\partial w}{\partial y}\right)^2 + 1}}$$

If we can calculate the derivatives of the height $w(x,y)$ and we know ϕ we can calculate the intensity at any point.

(a) On Canvas there is a file called `altitude.txt`, which contains the altitude $w(x,y)$ in meters above sea level (or depth below sea level) of the surface of the Earth, measured on a grid of points (x,y). Write a program that reads this file and stores the data in an array. Then calculate the derivatives $\partial w/\partial x$ and $\partial w/\partial y$ at each grid point. Explain what method you used to calculate them and why. To calculate the derivatives you'll need to know the value of h, the distance in meters between grid points, which is about 30 000 m in this case.

(b) Now, using your values for the derivatives, calculate the intensity for each grid point, with $\phi = 45°$, and make a density plot of the resulting values in which the brightness of each dot depends

on the corresponding intensity value. If you get it working right, the plot should look like a relief map of the world – you should be able to see the continents and mountain ranges in 3D. (Common problems include a map that is upside-down or sideways, or a relief map that is "inside-out", meaning the high regions look low and vice versa. Work with the details of your program until you get a map that looks right to you.)

(c) There is another file in the on-line resources called `stm.txt`, which contains a grid of values from scanning tunneling microscope measurements of the (111) surface of silicon. A scanning tunneling microscope (STM) is a device that measures the shape of surfaces at the atomic level by tracking a sharp tip over the surface and measuring quantum tunneling current as a function of position. The end result is a grid of values that represent the height of the surface as a function of position and the data in the file `stm.txt` contain just such a grid of values. Modify the program you just wrote to visualize the STM data and hence create a 3D picture of what the silicon surface looks like. The value of h for the derivatives in this case is around $h = 2.5$ (in arbitrary units).

For some interesting examples and discussion of lighting effects in computer generated images of astronauts on the moon (including some moon conspiracy theory debunking!), watch this video:

Nvidia Debunks Conspiracy Theories About Moon Landing:
https://www.youtube.com/watch?v=syVP6zDZN7I