Computational Lab 4 : Solving linear and non–linear systems of equations

Due October 4th, 2019 (@ 17:00)

- Read this document and do its suggested readings to help with the pre-labs and labs.
- The topics of this lab are to put into practice knowledge about numerical errors, and the trapezoid and Simpson's rules for integration.
- 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
```

  For example, in this lab, you will probably use Trapezoidal and Simpson's rules more than once. You may want to write generic functions for these rules (or use the piece of code, provided by the textbook online resources), place them in a separate file called e.g. `functions_lab02.py`, and call and use them in your answer files with:

```
import functions_lab02 as fl2  # make sure file is in same folder
ZehValyou = 4.
ZehDubble = fl2.MyFunc(ZehValyou)
```

Physics Background

Ellipses: The standard equation for an ellipse is

$$\frac{(x-h)^2}{a^2} + \frac{(y-k)^2}{b^2} = 1 \tag{1}$$

where a and b are the semi-major and semi-minor axes, respectively, and (h,k) is the centre of the ellipse.

The general equation of an ellipse is given by

$$Ax^2 + 2Bxy + Cy^2 + 2Dx + 2Fy + G = 0 \tag{2}$$

where A, B, C, D, F, and G are constants. Given enough coordinate values for (x, y), one can solve for the constants using linear algebra.

Kepler's laws: Kepler's first law states that the orbit of a planet about a star must be an ellipse. His second law states that the distance l_2 and velocity v_2 of the planet at its most distant point (aphelion) satisfy $l_2 v_2 = l_1 v_1$, where l_1 and v_1 are the distance and velocity at perihelion. Using Kepler's laws, the following equations can be derived:

$$a = \frac{1}{2}(l_1 + l_2) \tag{3}$$

$$b = \sqrt{l_1 l_2} \tag{4}$$

$$T = \frac{2\pi ab}{l_1 v_1} \tag{5}$$

$$e = \frac{l_2 - l_1}{l_2 + l_1} \tag{6}$$

where a and b are the semi-major and semi-minor axes, respectively, T is the orbital period, and e is the orbital eccentricity.
The ellipse equation parameters can be used to derive the aphelion, perihelion, semi–axes lengths using http://mathworld.wolfram.com/Ellipse.html, starting at equation 15 that you should fit from the data.

Black body radiation An incandescent light bulb usually contains a metal filament that is thin enough so that an electrical current will heat it sufficiently to radiate thermally and visibly. Almost all of the power consumed by the bulb is radiated as blackbody radiation, with only a fraction of the radiation in the visible part of the spectrum.
If the light bulb is assumed to be a perfect black body, then the power radiated per unit wavelength λ obeys

$$I(\lambda) = 2\pi\, Ahc^2 \frac{\lambda^{-5}}{e^{hc/\lambda k_b T} - 1}, \tag{7}$$

where A is the surface area of the filament, h is Planck's constant, c is the speed of light, and k_B is Boltzmann's constant. The total energy radiated between two wavelengths λ_1 and λ_2 is then

$$E(\lambda_1, \lambda_2) = \int_{\lambda_1}^{\lambda_2} I(\lambda)\, d\lambda, \tag{8}$$

and a lightbulb efficiency can be defined as the ratio of desirable light to the total energy emitted by the lightbulb,

$$\eta = \frac{E(\lambda_1, \lambda_2)}{E(0, \infty)} \tag{9}$$

Tungsten current–temperature function For Tungsten filaments the resistance of the wire varies with temperature as

$$R = R_0 \left(1 + \alpha (T - T_0)\right), \tag{10}$$

where $R_0 = 1.1\Omega$ and $\alpha = 4.5 \times 10^{-3} K^{-1}$. This equation can be inverted to give the temperature at a given input voltage and current,

$$T = 300 + \frac{\frac{V/I}{R_0} - 1}{\alpha}, \tag{11}$$

to give a temperature in Kelvins.

Computational Background

Solving constrained linear equations (for Q1): (Following Fitzgibbon, Pilu and Fischer in Fitzgibbon, A.W., Pilu, M., and Fischer R.B., Direct least squares fitting of ellipsees, Proc. of the 13th Internation Conference on Pattern Recognition, pp 253—257, Vienna, 1996).
In fitting an elliptical function to a set of data points we are trying to fit the equation

$$Ax^2 + 2Bxy + Cy^2 + 2Dx + 2Fy + G = 0, \tag{12}$$

for position elements x and y, and (unknown) parameters A, B, C, D, F, G. This equation can be written as

$$f(a, X) = X \cdot a = 0,$$

for the parameters $a = (A, 2B, C, 2D, 2F, G)$ and $X = (x^2, xy, y^2, x, y, 1)$. To fit this function we could minimize the squared distance

$$\delta(a, x) = \sum_{i=1}^{N} f(a, X)^2,$$

for each measurement point i. This minimization can be performed with 'numpy.linalg' or 'scipy.optimize.curve_fit'. However, this equation is trivially fit with the solution $A = B = C = D = F = G = 0$.
To make progress, we need to impose a constraint on the solution to require an ellipse, which can be written as $\gamma(a) = B^2 - 4AC < 0$, and instead of minimizing the squared distance we minimize the constrained function

$$L(a) = \delta(a, X) - \lambda(\gamma(a) - \phi), \tag{13}$$

where $\phi < 0$. Setting the $L(a)$ with respect to a to zero finds the minimum of the function $L(a)$.
Re–writing equation 13 in terms of a and X we get

$$\begin{aligned} L(a) &= \delta(a, X) - \lambda(\gamma(a) - \phi), && (14)\\ &= (Xa)^T(Xa) - \lambda(a^T Ya - \phi), && (15)\\ &= a^T(X^T X)a - \lambda(a^T Ya - \phi), && (16)\\ & && (17) \end{aligned}$$

where Y is the constraint matrix (i.e. $a^T Ya = 4AC - B^2$).
Minimizing the cost function L means finding where the derivative is zero

$$\begin{aligned} \partial L(a)_a &= (X^T X)a - \lambda Ya, && (18)\\ &= 0. && (19)\\ (X^T X)a &= \lambda Ya. && (20) \end{aligned}$$

Replacing $S = X^T X$ for convenience.

$$Sa = \lambda Ya. \tag{21}$$

The last equation should be recognizable as an eigenvalue equation. Re–writing as

$$\frac{1}{\lambda} a = S^{-1} Ya$$

it can be solved with the numpy 'eig' function, using the eigenvector with the largest eigenvalue as the likely solution to the constrained problem.

```
S_inv=...
Y=...
E, V = np.eig(np.dot(S_inv,Y)) # E contains the eigenvalues, V the eigenvectors.
```

The eigenvector decomposition gives you the parameters a which can then be converted to the ellipse parameters A, B, C, D, F, G

X	Y
-38.04	27.71
-35.28	17.27
-25.58	30.68
-28.80	31.50
-30.06	10.53

Table 1: An astronomically suspicious dataset of observations of a comet, in astronomical units (1AU = 1.496×10^{11} metres).

The relaxation method: If you are trying to solve the non–linear equation $x = f(x)$ where there is no simplification that converts it to a linear function then linear methods will not work. One method that works in many cases is to iterate using the equation to update the value of x such that it approaches the correct answer. Modifying the equation to

$$x_{n+1} = f(x_n), \tag{22}$$

then after some iterations N,

$$x_N = f(x_N) + \epsilon, \tag{23}$$

where the value of ϵ can be chosen based on your tolerance for errors in x.
For many functions this convergence happens quickly, but some functions diverge instead (e.g., from the textbook $x = e^{1-x^2}$). In these cases it is sometimes possible to re–arrange the equation to a form that allows convergence ($x = \sqrt{1-\log x}$).

Over–relaxation: For some functions the gradient $f(x)$ is small and the relaxation method is slow. In these cases we can apply an *over-relaxation* scale factor that increases the rate of convergence by artifically increasing the gradient of $f(x)$. A simple form of the over–relaxation method is

$$\begin{aligned} \delta x_n &= f(x_n) - x_n && (24)\\ x_{n+1} &= x_n + (1+\omega)\delta x_n && (25)\\ x_{n+1} &= x_n + (1+\omega)\left(f(x_n) - x_n\right) && (26)\\ x_{n+1} &= (1+\omega) f(x_n) - \omega x_n. && (27) \end{aligned}$$

For a small positive value ω this equation increases the contribution from the future value of x and decreases the contribution from the current value of x, overshooting the *relaxation* method. Alternatively a negative ω would reduce the contribution from the future and increase the contribution from the current value of x.

Questions

1. **[35%] Equation of the orbit of a comet** An astronomer wants to determine the orbit of a comet about the Sun. To do so, the astronomer sets up a Cartesian coordinate system in the plane of the orbit, setting the Sun's location as the origin, and then makes observations of the comet's position in this system five times. The measured coordinates are given in table 1.

   (a) Assuming your code successfully fits the data to an ellipse equation (equation 12) you will have parameters A,B,C,D,F,G from the ellipse equation. Derive, or get from another source (with a source reference!), the equations you will use to convert these ellipse parameters into astronomical parameters of interest. You should be able to calculate perihelion, aphelion, period, eccentricity, semi–major (long) and semi–minor (short) axes.

   SUBMIT YOUR DERIVATIONS OR EQUATIONS WITH REFERENCES

   (b) Write the code to find the equation of the orbit. From this equation, find the perihelion, aphelion, period, and eccentricity. Make sure to document your method for fitting the ellipse, referring to equations as

needed. Plot the orbit, along with the measured points. The estimated minimum speed of the comet is 1350 m/s.

SUBMIT YOUR CODE, EQUATION FOR THE ORBIT, AND PLOT

(c) Before submitting the discovery to the IAU, the astronomer needs to check the list of known comets at https://en.wikipedia.org/wiki/List_of_Halley-type_comets. Has the comet been discovered before? if so, which comet is it?

SUBMIT YOUR ANSWER THE QUESTION AND REASONING.

2. **[35%] The temperature of a light bulb**

(a) Write a Python function to take the temperature T and calculate the efficiency η of a light–bulb for radiation between 380nm and 780nm. Use the function to make a graph of the efficiency from 300K to 10,000K. You should see an extremum in the efficiency curve.

SUBMIT YOUR PYTHON CODE, PLOT, AND WRITTEN ANSWERS.

(b) Calculate the temperature of maximum efficiency of the light bulb to within 1K using an appropriate search method. For this level of accuracy you will need accurate physical constants from 'scipy' and accurate integration and search methods.

SUBMIT YOUR PYTHON CODE, AND TEMPERATURE OF MAXIMUM EFFICIENCY.

(c) How would your answer change if you wanted to make an light bulb that emits in the infra–red (780nm-2250nm) instead of the visible? Is it possible to run the Tungsten bulb at maximum efficiency for either wavelength region?

SUBMIT THE NEW TEMPERATURE OF MAXIMUM EFFICIENCY, AND WRITTEN ANSWERS

3. **[30%] Solving non-linear functions** This question uses the relaxation method and over–relaxation method to solve non–linear equations. Parts of exercises 6.10,6.11, and 6.12 from the textbook are used below, but each question uses essentially the same methods and should not be as much work as the number of sub–parts implies.

(a) Complete exercise 6.10 part (a,b) from the textbook (page 259):

SUBMIT THE CODE AND PLOT

(b) Complete exercise 6.11 parts (b,c,d) from the textbook (page 260).

SUBMIT THE CODE YOU WRITE, AND A BRIEF DISCUSSION OF THE COMPARISON OF THE ITERATIONS TAKEN IN PART (B) COMPARED TO PART (C). FOR PART (D): SUBMIT YOUR WRITTEN ANSWER.

(c) Complete exercise 6.12 parts (a,b,c) from the textbook (page 262).

SUBMIT THE CODE YOU WRITE AND THE DERIVATION FOR PART A,B. MAKE SURE YOUR CODE FAILS GRACEFULLY IF IT DOES NOT CONVERGE AFTER A LARGE NUMBER OF STEPS.

Question:	1	2	3	Total
Points:	35	35	30	100