University of Sussex – Scientific Computing

Assignment 1

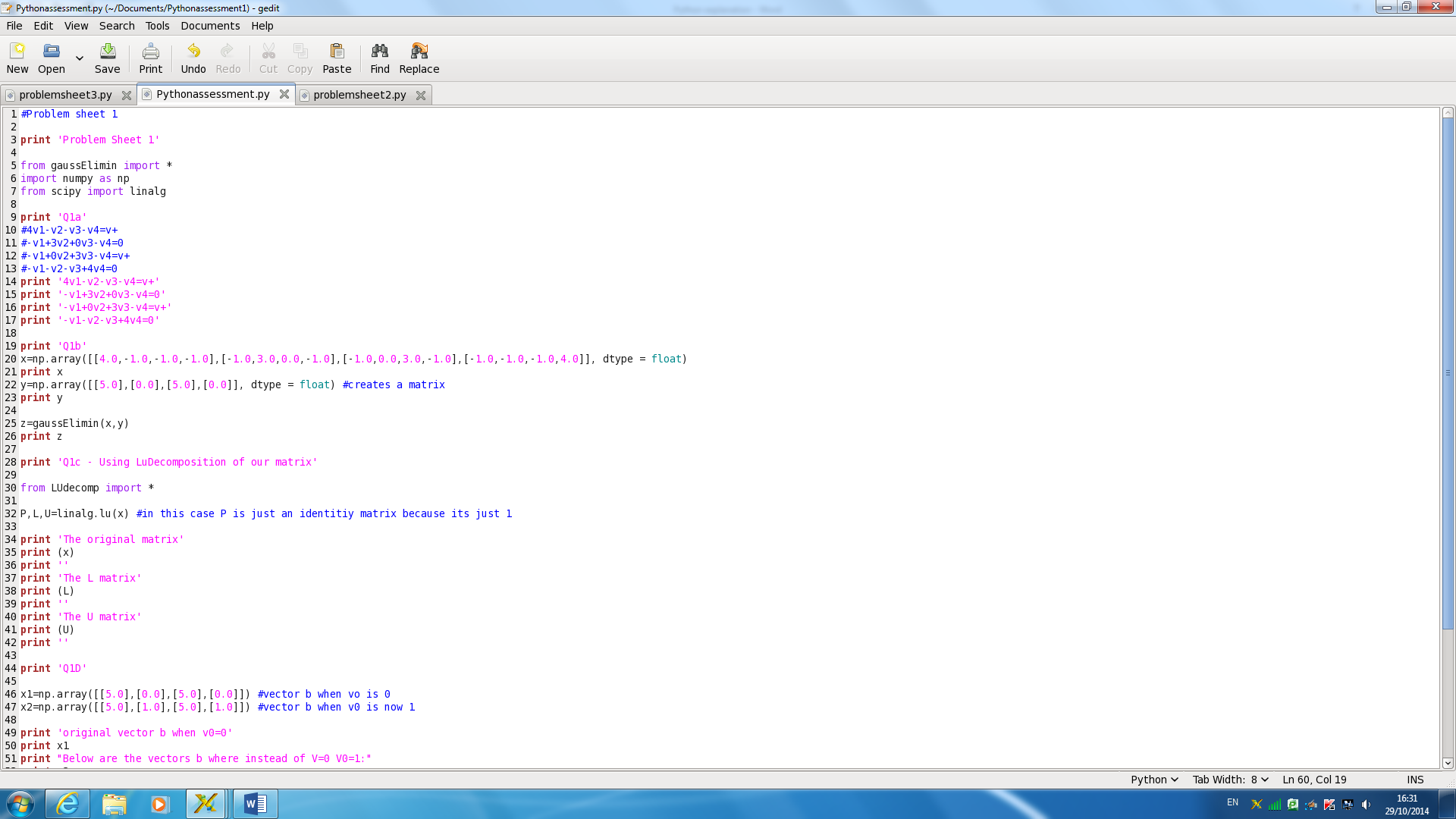
Problem Sheet 1



**Solutions-**



1a) Ohms law and Kirchhoff’s current law were used to calculate the voltages at the other three junctions.

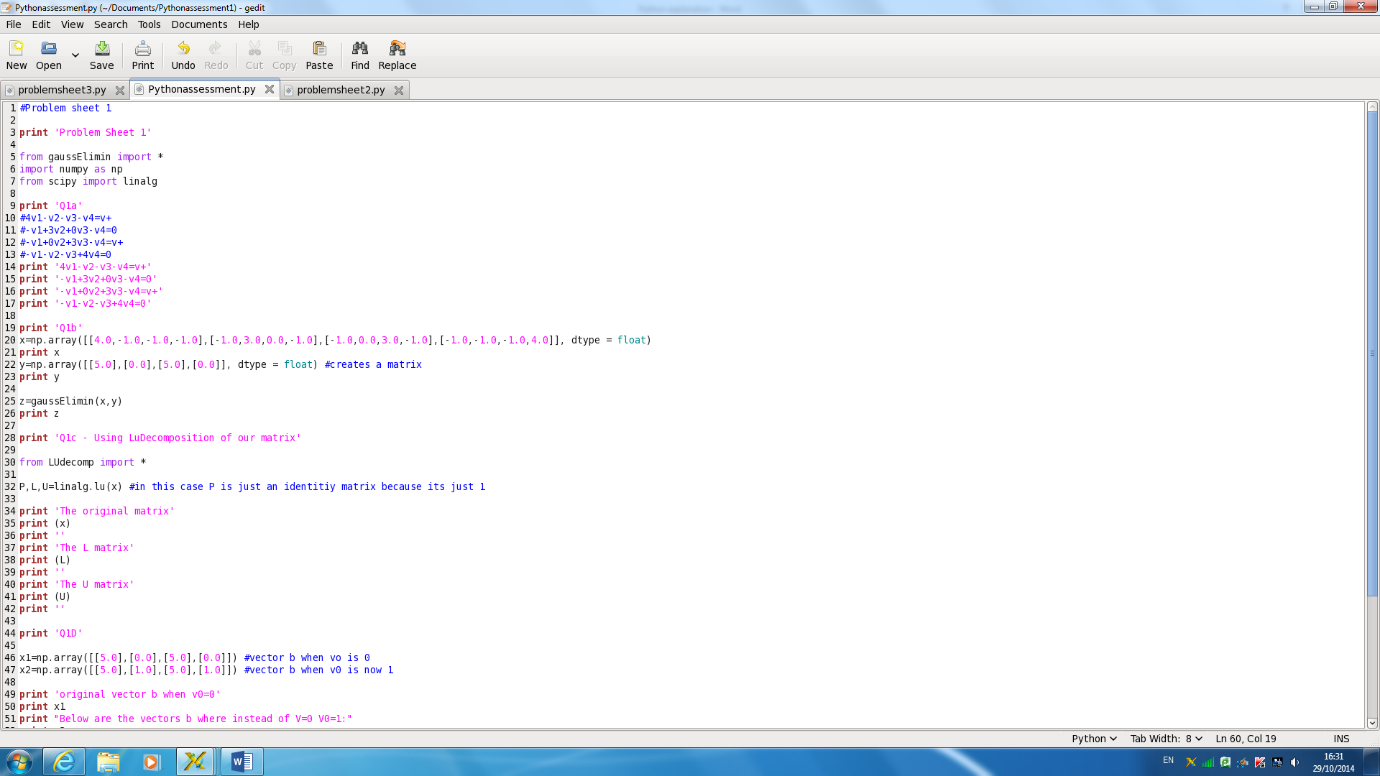
1b)

In line 22 and 24, I have formed a matrix by using an array. I then used Gaussian elimation to solve for the unknown voltages. Gaussian elimation uses the first equation to remove the x1 term from all other equations by subtracting λ=aj1/a11 times the first equation from all the jth equation (j>1).

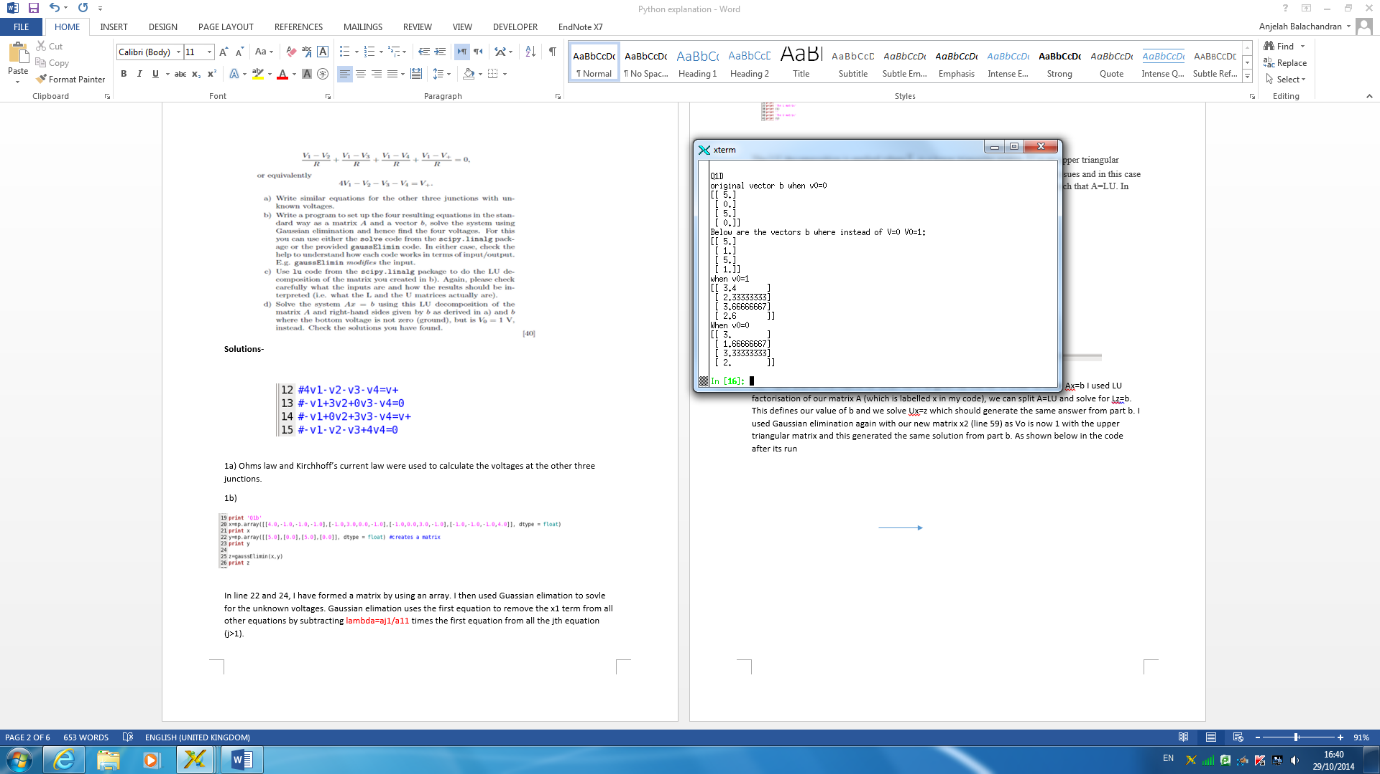
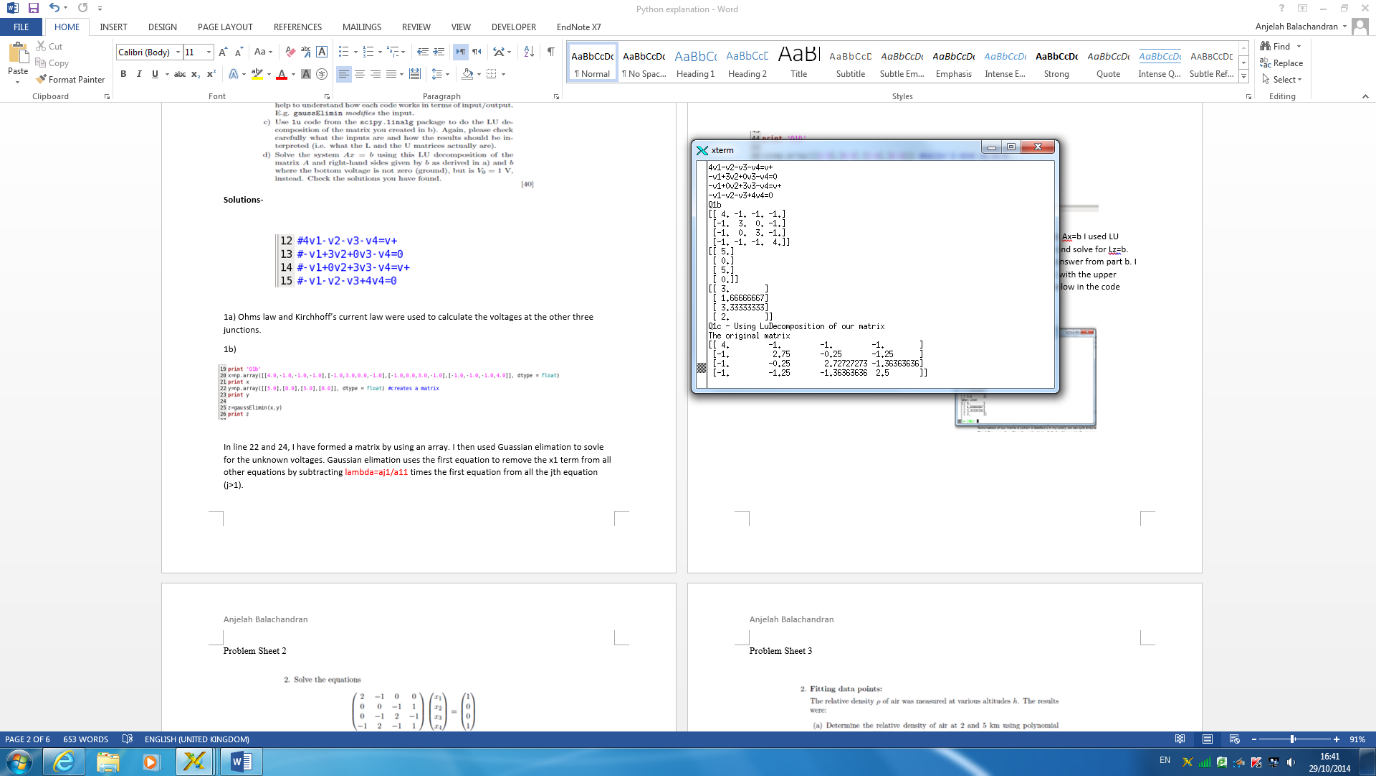
1c)

The LU decomposition is applied where L is a lower triangular matrix, U is an upper triangular matrix and P is a permutation matrix. P is needed to resolve certain singularity issues and in this case is the identity matrix. The LUdecomp function factorises any square matrix A such that A=LU. In lines 34 I have used this to split up my matrix x.

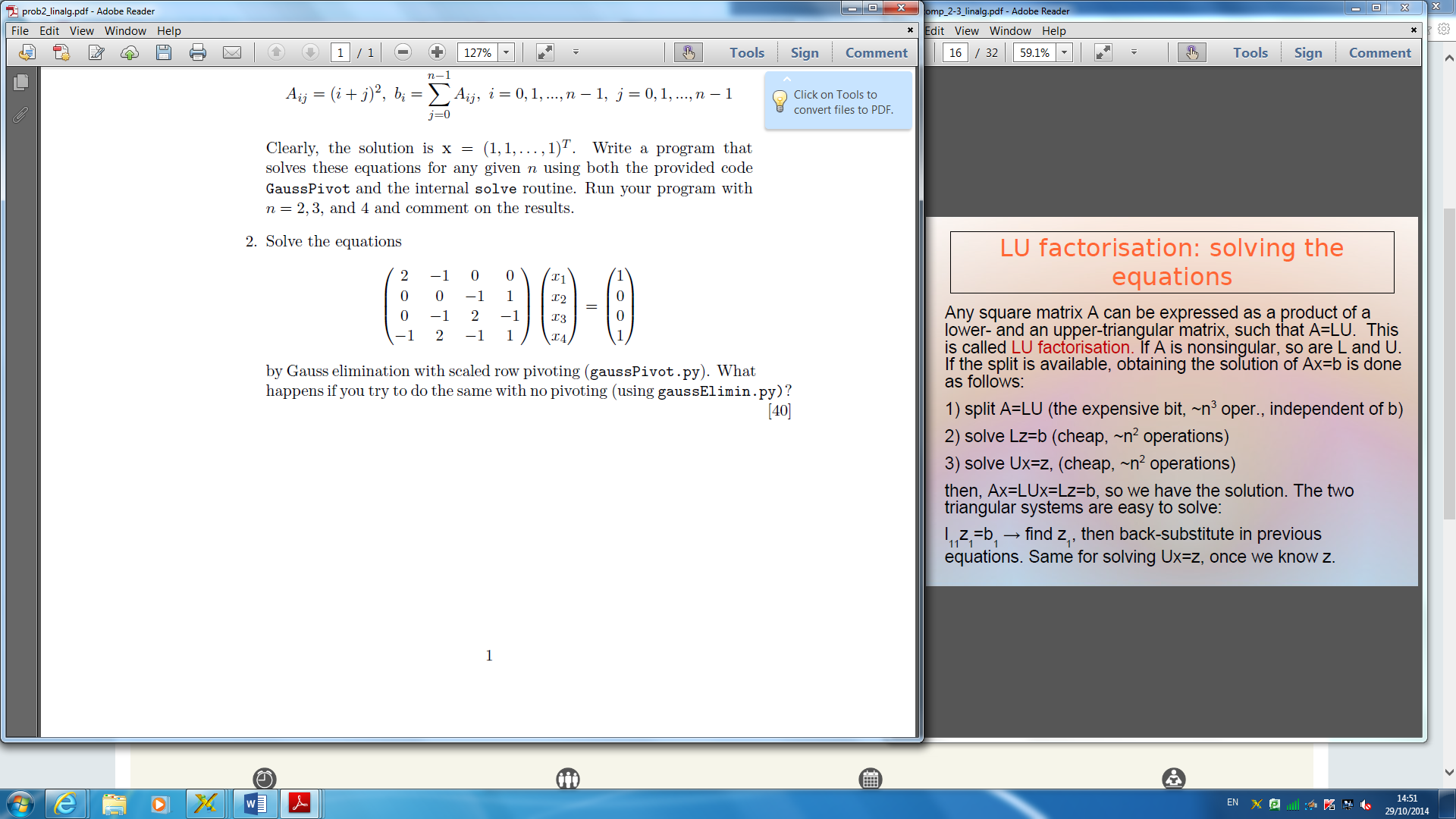
I have then proceeded to print the L and U matrix in lines 40 and 43.



1d) I have altered the matrix for when V0=1 as given by line 49. In order to solve Ax=b I used LU factorisation of our matrix A (which is labelled x in my code), we can split A=LU and solve for Lz=b. This defines our value of b and we solve Ux=z which should generate the same answer from part b. I used Gaussian elimination again with our new matrix x2 (line 59) as Vo is now 1 with the upper triangular matrix and this generated a new solution since V0=1. As shown below in the code after its run.

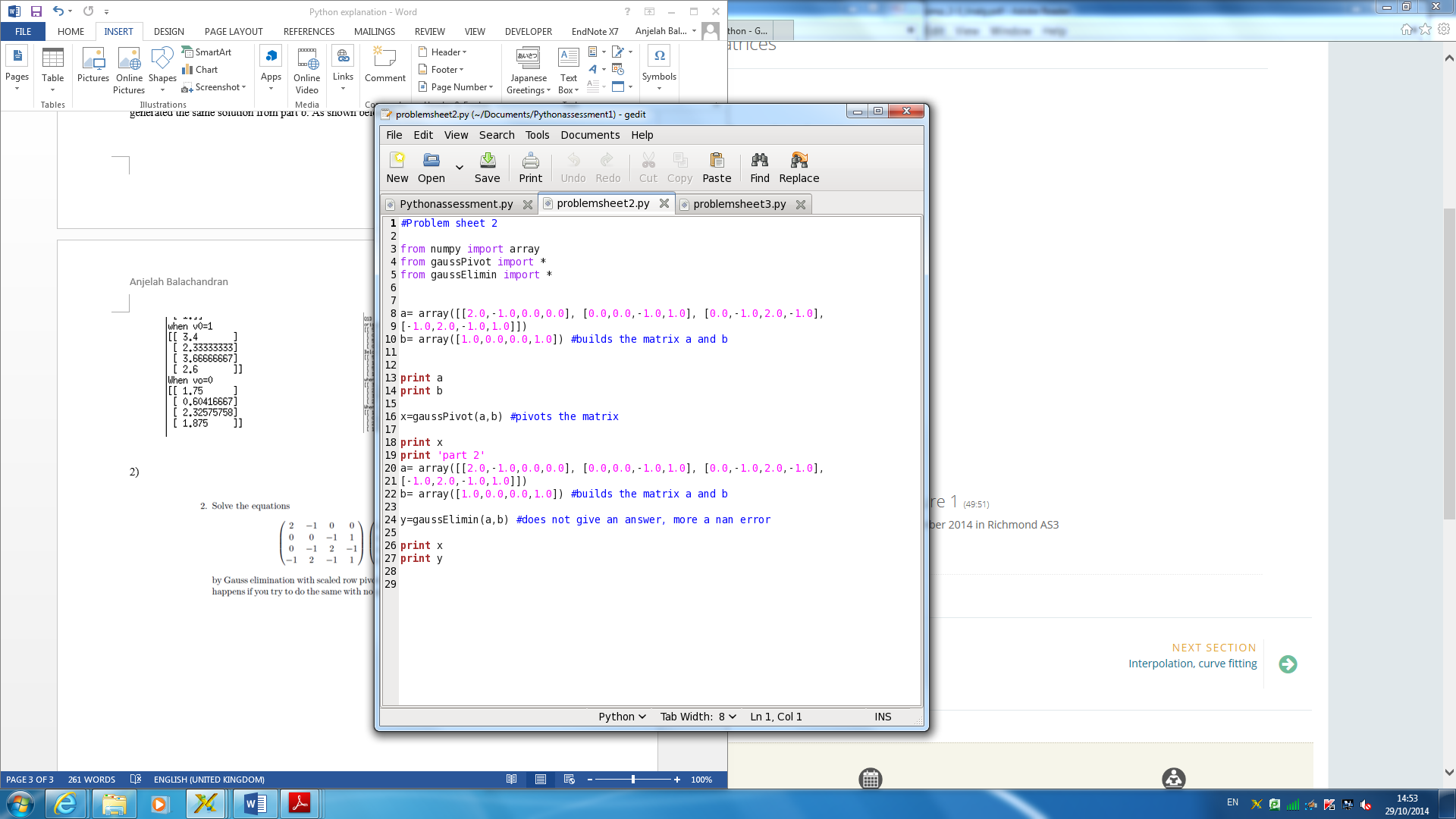


1b 1d

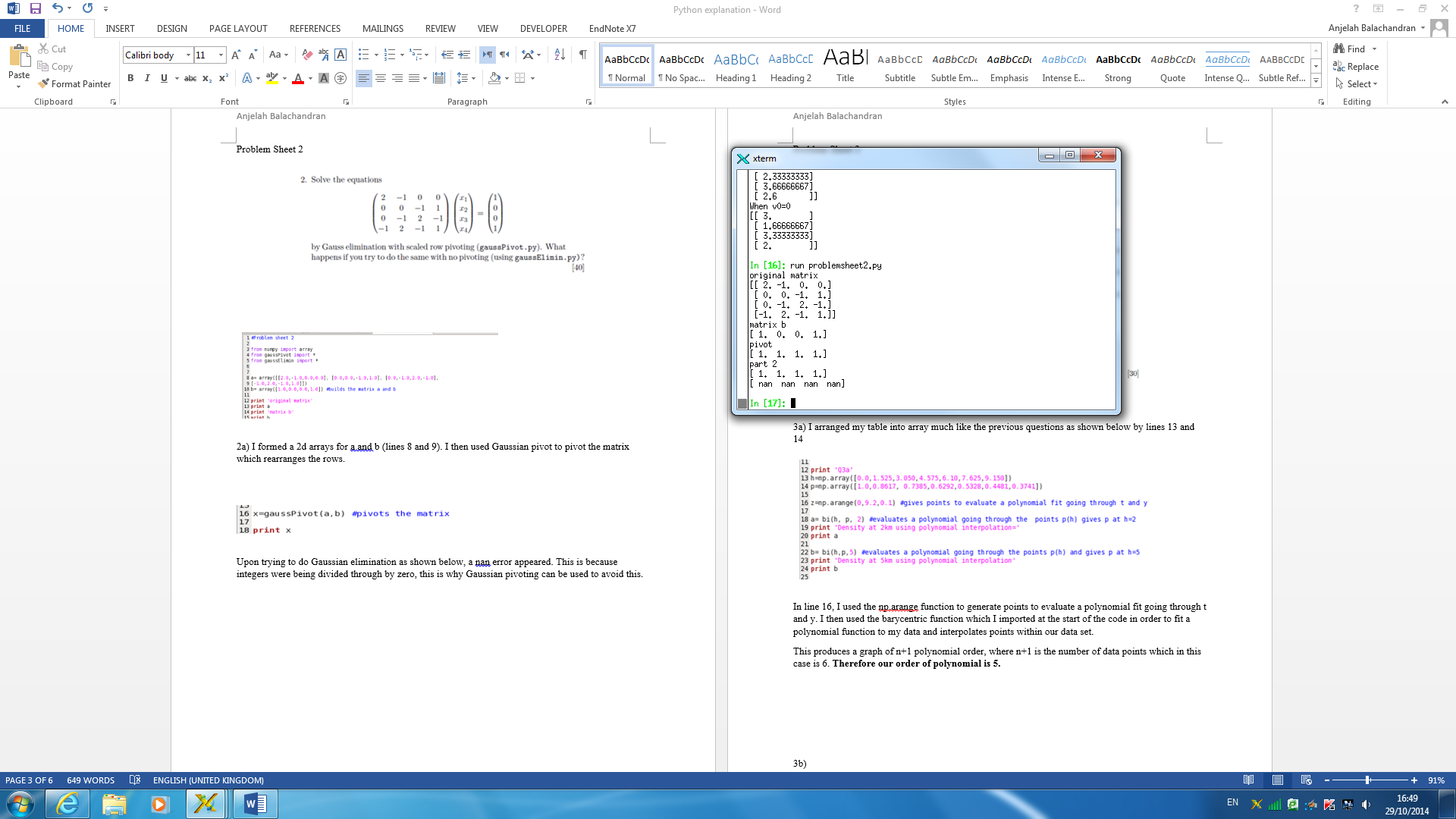
Problem Sheet 2

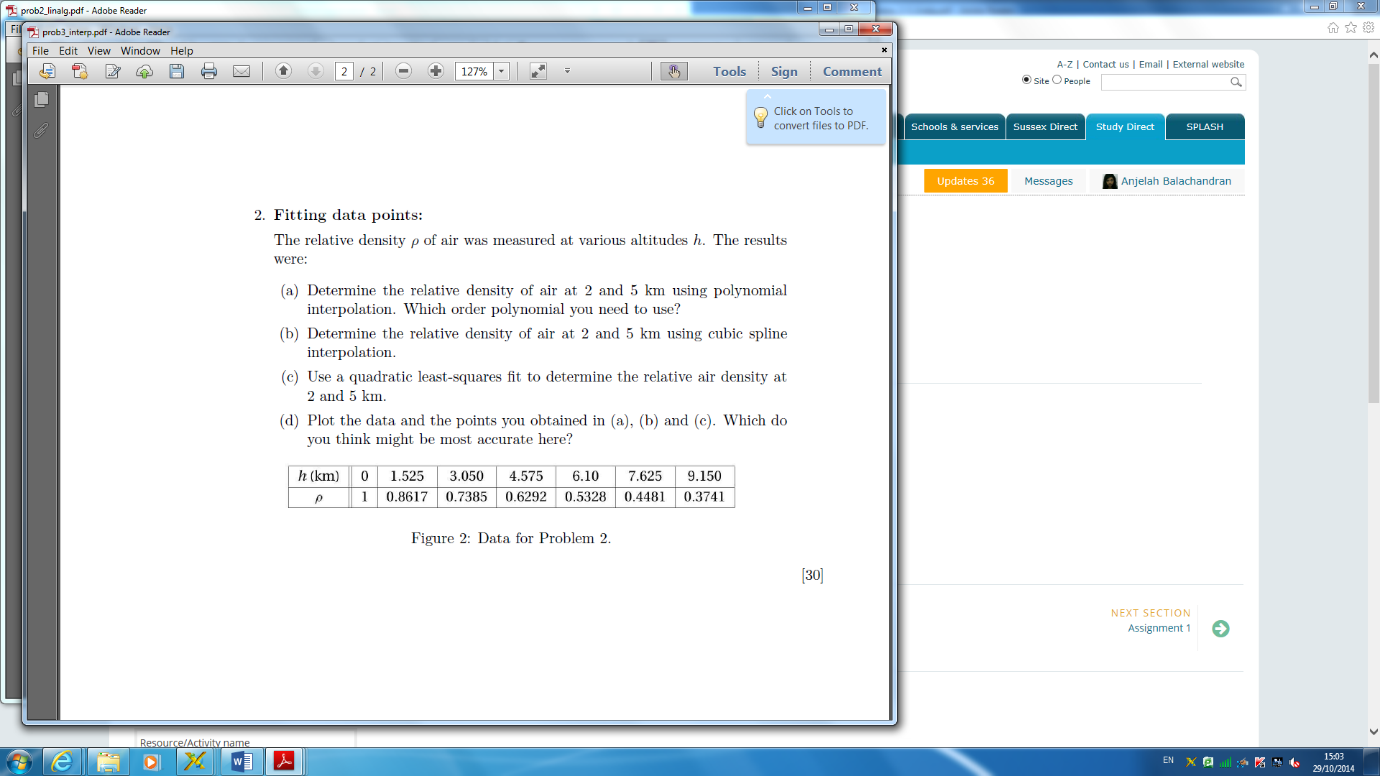


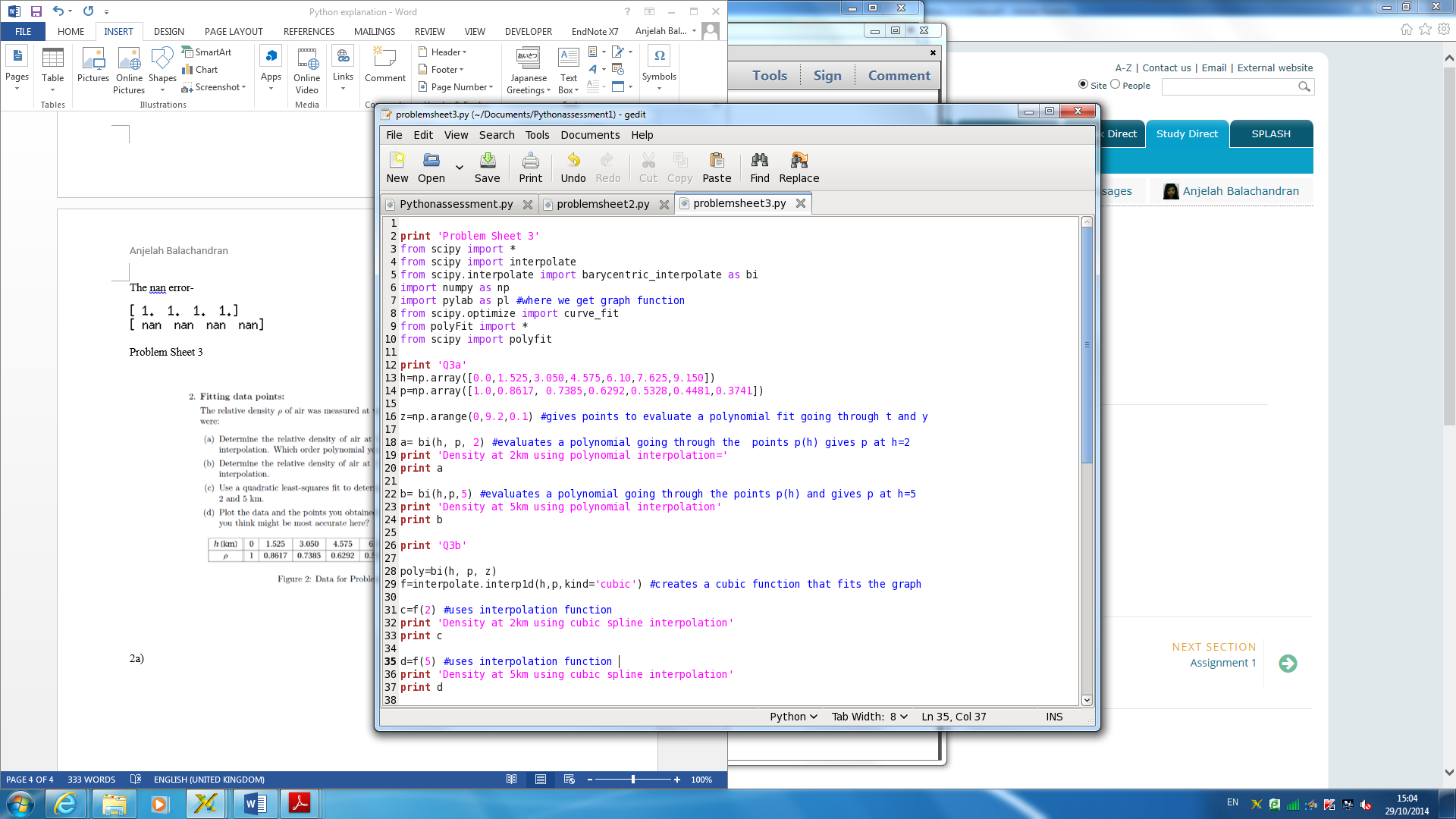
2a) I formed a 2d arrays for the matrices (lines 8 and 9). I then used Gaussian pivot to pivot the matrix which rearranges the rows.



Upon trying to do Gaussian elimination as shown below, a nan error appeared. This is because integers were being divided through by zero, this is why Gaussian pivoting can be used to avoid this.



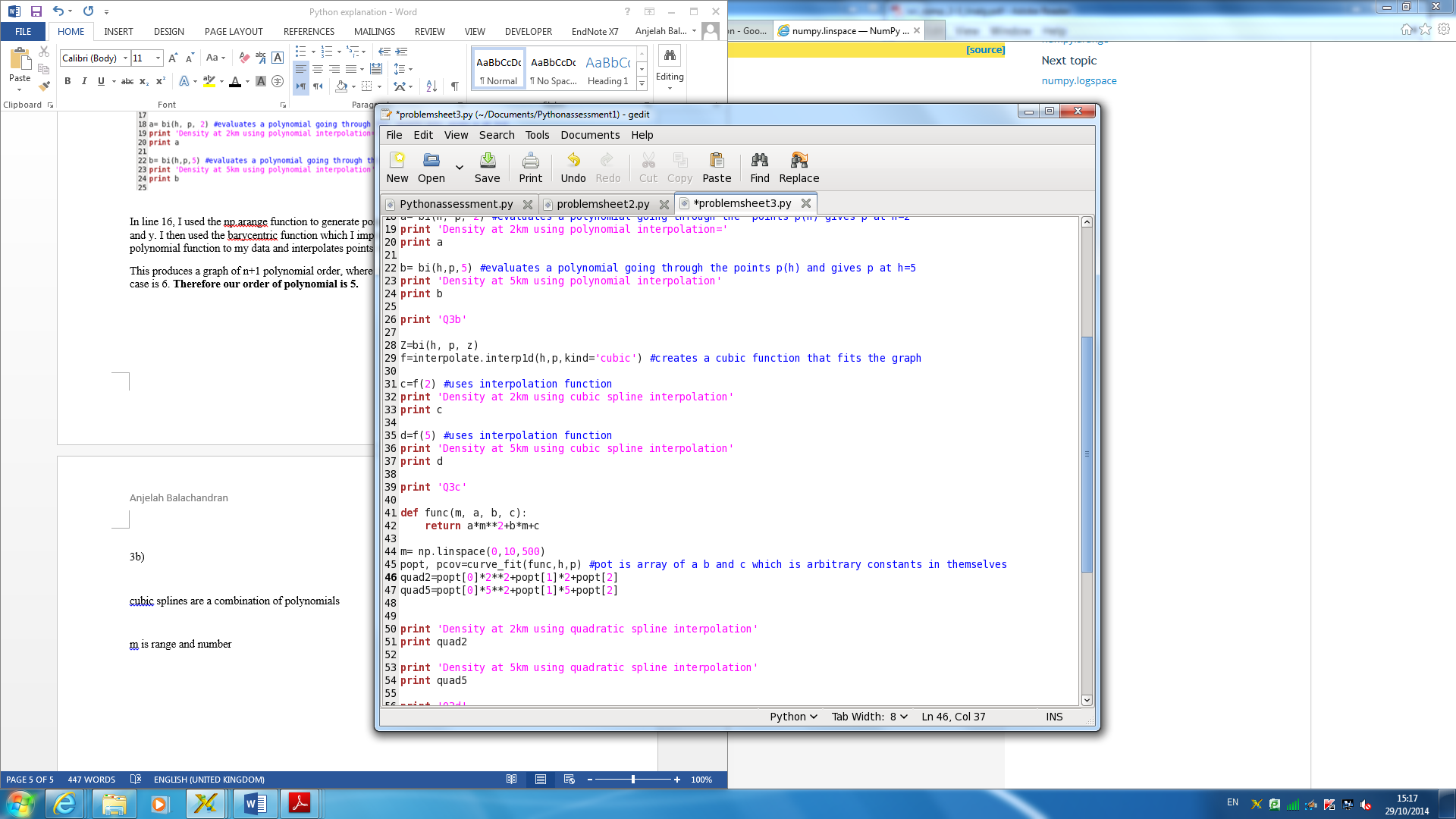
Problem Sheet 3

3a) I arranged my table into array much like the previous questions as shown below by lines 13 and 14.

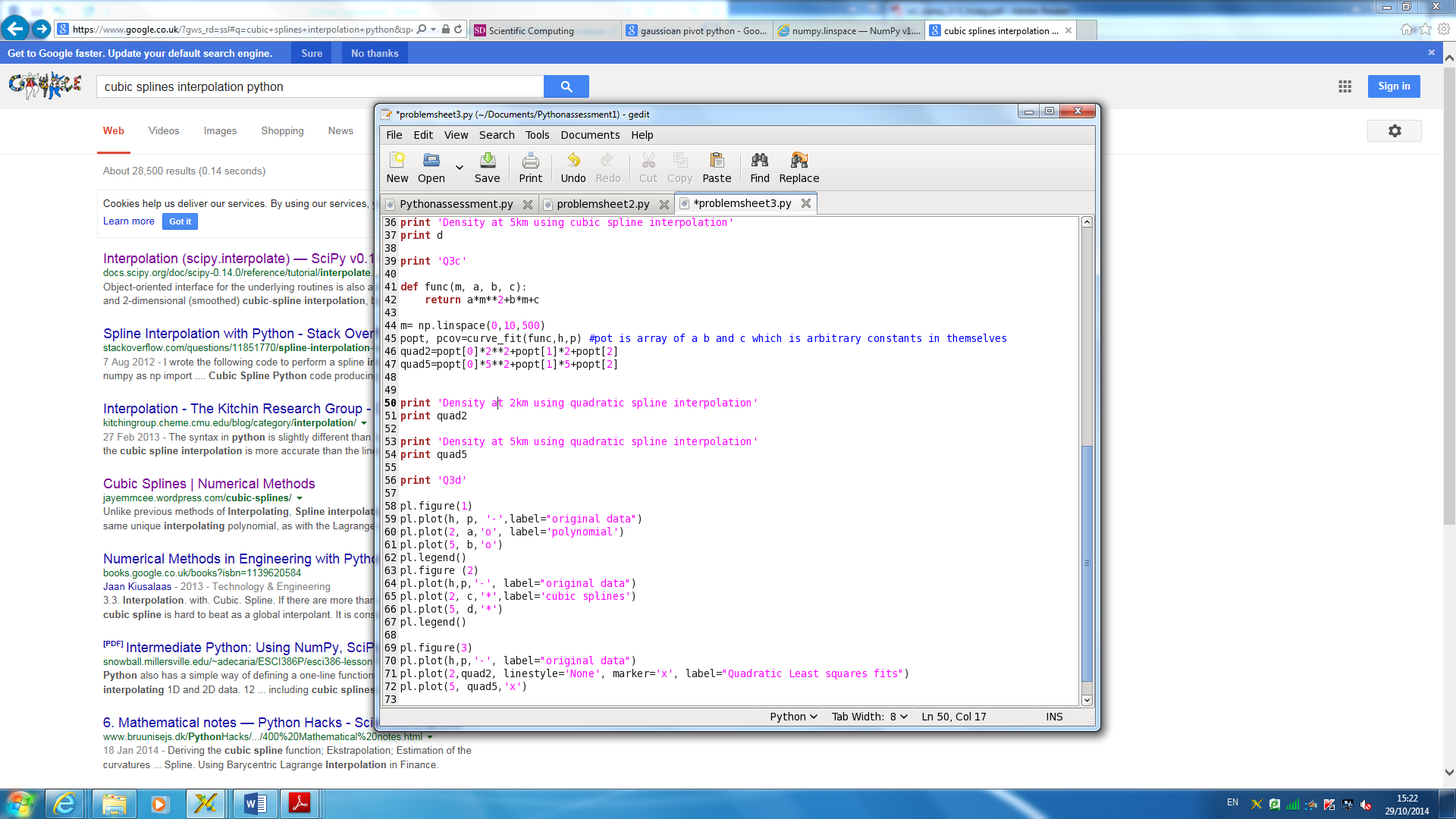
In line 16, I used the np.arange function to generate points to evaluate a polynomial fit going through t and y. I then used the barycentric function which I imported at the start of the code in order to fit a polynomial function to my data athat interpolates points within our data set.

This produces a graph of n+1 polynomial order, where n+1 is the number of data points, which in this case is 7. **Therefore our order of polynomial is 6.**

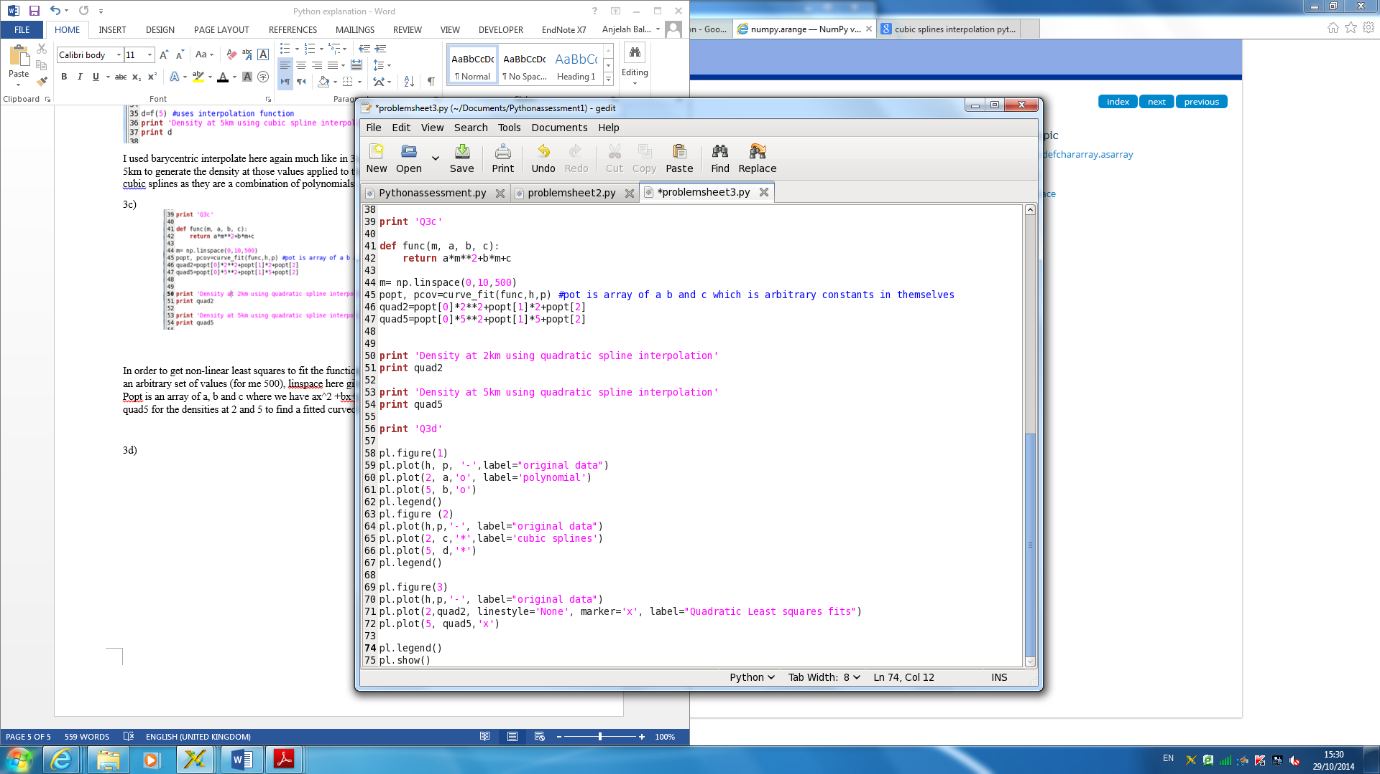
3b)



I used barycentric interpolate here again much like in 3a. I used the interpolate function for 2km and 5km to generate the density at those values applied to the cubic function that fits the graph. We use cubic splines as they are a combination of polynomials.

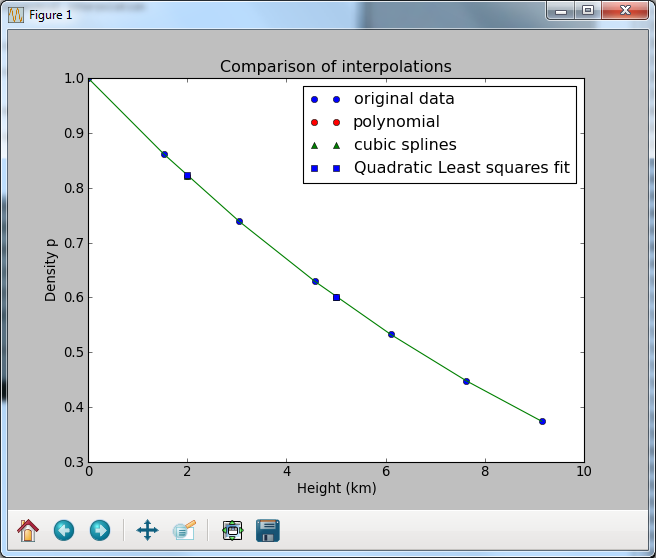
3c)

In order to get non-linear least squares to fit the function I defined the function where m contains set an arbitrary set of values (for me 500), linspace here gives an evenly spaced values within that range. Popt is an array of a, b and c where we have ax^2 +bx+c – i.e the quadratic form. I have set quad2 and quad5 for the densities at 2 and 5 to find a fitted curved function for it.

3d)

I put the results all on one graph so it was easier to compare and analyse. I used polyfit and plot imported from scipy to produce the graph. All three data sets produced the same shape curve. As two of the data points (the polynomial interpolation and cubic spline interpolation were identical), the data points were on top of each other, which is why the polynomial orange point is not there to see.

Graphs produced-



Upon close inspection of the data by zooming in, the quadratic least squares fit appears to be the best approximation for the set of interpolations as the data points are closer to the curve produced whilst the cubic splines and polynomial fit are further away from the line.

