NUMERICAL METHODS WEEK 4

CURVE FITTING 2

We continue with Curve Fitting. This week general linear regression.

Reading: Capra and Canale, introduction to part 5 and chapter 17.

Learning outcomes:

- Extend the work on Linear Regression to polynomial and multiple variables.
- Combine Python and analytical solutions or other platforms.
- Check your code works correctly, via an external reference.

MATT WATKINS MWATKINS@LINCOLN.AC.UK

FITTING A QUADRATIC FUNCTION

In the case that the largest power of x is x^2 we have

$$y_i(a_0, a_1, a_2; x_i) = a_0 + a_1 x_i + a_2 x_i^2 + e_i$$

and an overall error function

$$S_r(a_0, a_1, a_2) = \sum_{i=0}^{N-1} (y_i - a_0 - a_1 x_i - a_2 x_i^2)^2$$

This leads to a set of equations

$$\begin{aligned}
\frac{\delta S_r(a_0,a_1,a_2)}{\delta a_0} &\implies (n)\, a_0 + \left(\sum x_i\right) a_1 + \left(\sum x_i^2\right) a_2 &&= \sum y_i \\
\frac{\delta S_r(a_0,a_1,a_2)}{\delta a_1} &\implies \left(\sum x_i\right) a_0 + \left(\sum x_i^2\right) a_1 + \left(\sum x_i^3\right) a_2 &&= \sum x_i y_i \\
\frac{\delta S_r(a_0,a_1,a_2)}{\delta a_2} &\implies \left(\sum x_i^2\right) a_0 + \left(\sum x_i^3\right) a_1 + \left(\sum x_i^4\right) a_2 &&= \sum x_i^2 y_i
\end{aligned}$$

- Derive the above equations by differentiating S_r with respect to each of the a_i in turn.
- Write the equations in matrix form $\mathbf{A}x = b$, where x is a column matrix with entries a_0, a_1, a_2 and b is a column matrix with terms that do not depend on the fitting parameters, a_0, a_1 and a_2.
- Solve for a_0, a_1 and a_2.
- Plot your fitted parabola against the data to check the fit.

FITTING A QUADRATIC FUNCTION

This leads to a set of equations

$$
\begin{aligned}
\frac{\delta S_r(a_0, a_1, a_2)}{\delta a_0} &\implies (n)\, a_0 + \left(\sum x_i\right) a_1 + \left(\sum x_i^2\right) a_2 &= \sum y_i \\
\frac{\delta S_r(a_0, a_1, a_2)}{\delta a_1} &\implies \left(\sum x_i\right) a_0 + \left(\sum x_i^2\right) a_1 + \left(\sum x_i^3\right) a_2 &= \sum x_i y_i \\
\frac{\delta S_r(a_0, a_1, a_2)}{\delta a_2} &\implies \left(\sum x_i^2\right) a_0 + \left(\sum x_i^3\right) a_1 + \left(\sum x_i^4\right) a_2 &= \sum x_i^2 y_i
\end{aligned}
$$

We can rewrite these equations as

$$
\begin{pmatrix}
(\sum x_i^0) & (\sum x_i) & (\sum x_i^2) \\
(\sum x_i) & (\sum x_i^2) & (\sum x_i^3) \\
(\sum x_i^2) & (\sum x_i^3) & (\sum x_i^4)
\end{pmatrix}
\begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix}
=
\begin{pmatrix} \sum y_i \\ \sum x_i y_i \\ \sum x_i^2 y_i \end{pmatrix}
$$

Which is of the form $\mathbf{A}x = b$

GENERAL LINEAR LEAST SQUARES

Simple linear, polynomial and multiple linear regression can be generalised to the following linear least-squares model

$$y_i = a_0 z_0(x_i) + a_1 z_1(x_i) + a_2 z_2(x_i) + \cdots + a_{m-1} z_{m-1}(x_i) + e_i$$

can now not polynomials in x but some *predefined* functions of those positions

$z_0(x), z_1(x), \ldots, z_{m-1}(x)$ are m $\mathbf{basis\ functions}$. The predefined basis functions define the model, only depend on the x coordinate. I is called linear least squares as the parameters $a_0, a_1, \ldots, a_{m-1}$ appear linearly. The zs can be highly non-linear in x.

For instance.

$$y_i = a_0 \cdot 1 + a_1 \cos(\omega x_i) + a_2 \sin(\omega x_i)$$

fits this model with $z_0 = 1, z_1 = \cos(\omega x_0)$ and $z_2 = \sin(\omega x_0)$. Where x is a single independent variable and ω is a predefined constant.

GENERAL LINEAR LEAST SQUARES

We can rewrite

$$y_i = a_0 z_0(x_i) + a_1 z_1(x_i) + a_2 z_2(x_i) + \cdots + a_{m-1} z_{m-1}(x_i) + e_i$$

in matrix notation as

$$\boldsymbol{y} = \boldsymbol{Z}\boldsymbol{a} + \boldsymbol{e}$$

where bold lower case indicates a column vector, and bold uppercase indicates a matrix. $\boldsymbol{Z}$ contains the calculated values of the m basis functions at the n measured values of the independent variables:

$$\boldsymbol{Z} = \begin{bmatrix} z_0(x_0) & z_1(x_0) & \cdots & z_{m-1}(x_0) \\ z_0(x_1) & z_1(x_1) & \cdots & z_{m-1}(x_1) \\ \vdots & \vdots & \ddots & \vdots \\ z_0(x_{n-1}) & z_1(x_{n-1}) & \cdots & z_{m-1}(x_{n-1}) \end{bmatrix}$$

The column vector $\boldsymbol{y}$ contains the $n+1$ observed values of the dependent variable

$$\boldsymbol{y}^T = [y_0, y_1, y_2, y_3, \ldots, y_{n-1}]$$

The column vector $\boldsymbol{a}$ contains the $m+1$ unknown parameters of the model

$$\boldsymbol{a}^T = [a_0, a_1, a_2, \ldots, a_{m-1}]$$

The column vector $\boldsymbol{e}$ contains the $n+1$ observed residuals (errors)

$$\boldsymbol{e}^T = [e_0, e_1, e_2, e_3, \ldots, e_{n-1}]$$

GENERAL LINEAR LEAST SQUARES

We can also express error in our model as a sum of the squares much like before:

$$
\begin{aligned}
S_r &= \sum_{i=0}^{n} \left(y_i - \sum_{j=0}^{m} a_j z_{ji} \right)^2 \\
&= \sum e_i^2 = \boldsymbol{e}^T \boldsymbol{e} = (\boldsymbol{y} - \boldsymbol{Z}\boldsymbol{a})^T (\boldsymbol{y} - \boldsymbol{Z}\boldsymbol{a})
\end{aligned}
$$

S_r is minimised by taking partial derivatives wrt $\boldsymbol{a}$, which yields

$$\boldsymbol{Z}^T \boldsymbol{Z}\boldsymbol{a} = \boldsymbol{Z}^T \boldsymbol{y}$$

which is exactly equivalent to the set of simultaneous equations for a_i we found previously when fitting polynomials. More details can of the derivation can be found here (http://fourier.eng.hmc.edu/e176/lectures/NM/node35.html), though the notation is a little different.

This set of equations is of the form $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ and can be solved using gauss elimination or similar method.

> Try to redo the earlier fitting problems in this notation / method.

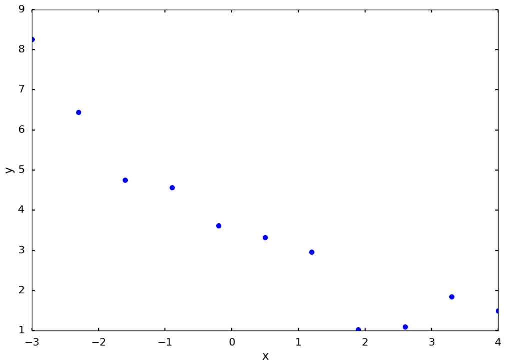
GENERAL LINEAR LEAST SQUARES - EXAMPLE

Suppose we have 11 measurements at

$$x^T = [-3., -2.3, -1.6, -0.9, -0.2, 0.5, 1.2, 1.9, 2.6, 3.3, 4.0]$$

with measured values

$$y^T = [8.26383742, 6.44045188, 4.74903073, 4.5656476, 3.61011683, 3.32743918, 2.9643915, 1.02$$
$$1.49110572]$$



Let us fit it to a model of the form $y_i = a_0 \cdot 1 + a_1 e^{-x_i} + a_2 e^{-2x_i}$

Our $\boldsymbol{Z}$ matrix has 3 columns for the basis functions $z_0(x_i) = 1, z_1(x_i) = e^{-x_i}$ and finally $z_2 = e^{-2x_i}$. It will have 11 rows corresponding to the 11 measurements.

```
Z =
[
[   1.00000000e+00,   2.00855369e+01,   4.03428793e+02],
[   1.00000000e+00,   9.97418245e+00,   9.94843156e+01],
[   1.00000000e+00,   4.95303242e+00,   2.45325302e+01],
[   1.00000000e+00,   2.45960311e+00,   6.04964746e+00],
[   1.00000000e+00,   1.22140276e+00,   1.49182470e+00],
[   1.00000000e+00,   6.06530660e-01,   3.67879441e-01],
[   1.00000000e+00,   3.01194212e-01,   9.07179533e-02],
[   1.00000000e+00,   1.49568619e-01,   2.23707719e-02],
[   1.00000000e+00,   7.42735782e-02,   5.51656442e-03],
[   1.00000000e+00,   3.68831674e-02,   1.36036804e-03],
[   1.00000000e+00,   1.83156389e-02,   3.35462628e-04]]
```

Then we set up the linear equation problem by forming
$\boldsymbol{Z}^T \boldsymbol{Z}$

```
ZTZ =
[
[  1.10000000e+01,   3.98805235e+01,   5.35475292e+02],
[  3.98805235e+01,   5.35475292e+02,   9.23382518e+03],
[  5.35475292e+02,   9.23382518e+03,   1.73292733e+05]]
```

and

$\boldsymbol{Z}^T \boldsymbol{y}$

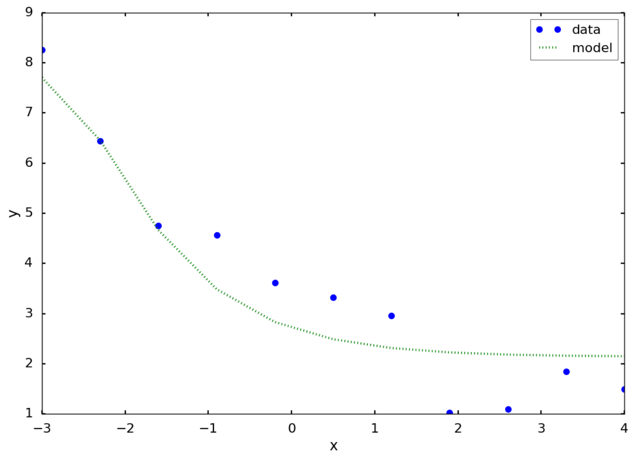
```
ZTy =
[   39.36979777,   272.62352738,  4125.63079852]
```

The solutions of this problem are

```
a = [ 2.13758951,  0.58605735, -0.01537541]
```

This means that our final model for the data is

$$y = 2.13758951 + 0.58605735e^{-x} - 0.01537541e^{-2x}$$



STATISTICAL INTERPRETATION OF LEAST SQUARES

The matrix $(\boldsymbol{Z}^T\boldsymbol{Z})^{-1}$ contains the variance (diagonal elements) and covariances (off-diagonal elements) of the a_i so can be used to estimate the accuracy of the parameter estimation.

We can use the Gauss Jordan method to find $(\boldsymbol{Z}^T\boldsymbol{Z})^{-1}$.

The diagonal elements of $(\boldsymbol{Z}^T\boldsymbol{Z})^{-1}$ can be designated as $z_{i,i}^{-1}$

We will call the standard error of our fitted model to the data
$s_{y/x} = \frac{1}{\sqrt{n-m}}\sqrt{\sum_{i=0}^{i=n-1}[y_i - (a_0 z_0(x_i) + a_1 z_1(x_i) + a_2 z_2(x_i) + \cdots + a_{m-1} z_{m-1}(x_i))]^2}$

The variances of our parameters are given by $\mathrm{var}(a_i) = s^2(a_i) = z_{i,i}^{-1} s_{y/x}^2$

We can now place error limits on our optimal parameters, $a_0, \ldots a_{m-1}$.

If our model is good, the real data should be approximately normally distributed around our model

You can then show that the parameters should have a t-distribution (https://mattatlincoln.github.io/teaching/statistics/lecture_9/#/6/2) with $n-2$ degrees of freedom.

We can put confidence limits on the parameters using
$P(\text{true value of the ith parameter is in the interval } (a_i - t_{c/2,n-2}s(a_i), a_i + t_{c/2,n-2}s(a_i)) = c$
where c is our confidence, for instance 0.95 to be 95% certain the parameter lies within those bounds and t_c are the critical values for the appropriate t distribution (https://www.itl.nist.gov/div898/handbook/eda/section3/eda3672.htm).

Perform a fit to the following data.

```
  x = {         y = {
10.00,          8.953,
16.30,          16.405,
23.00,          22.607,
27.50,          27.769,
31.00,          32.065,
35.60,          35.641,
39.00,          38.617,
41.50,          41.095,
42.90,          43.156,
45.00,          44.872,
46.00,          46.301,
45.50,          47.490,
46.00,          48.479,
49.00,          49.303,
50.00           49.988
}               }
```

use the model form $y_i = a_0 \cdot 1 + a_1 x_i + e_i$

- Calculate an error estimate for the optimal parameters. Note, you can perform the matrix inversion required using Gauss Jordan elimination (https://mattatlincoln.github.io/teaching/numerical_methods/lecture_2/#/7), by by using a solver in Eigen and setting the right hand side equal to the unit matrix.
- Upload your solutions and answers to related questions onto Blackboard (https://blackboard.lincoln.ac.uk/webapps/bbgs-acxiom-bb_bb60/execute/acxiomGateway?

course_id=_127603_1&content_id=_2238111_1&mode=view)

Remember to break the problem down into smaller ones:

- Can you set up the Z matrix?
- Can you find the transpose of the Z matrix?
- Can you multiply the transposed and non-transposed matrices together?
- Can you solve the system of linear equations?

SUMMARY AND FURTHER READING

You should be reading additional material to provide a solid background to what we do in class

Reading: Capra and Canale, introduction to part 5 and chapter 17.

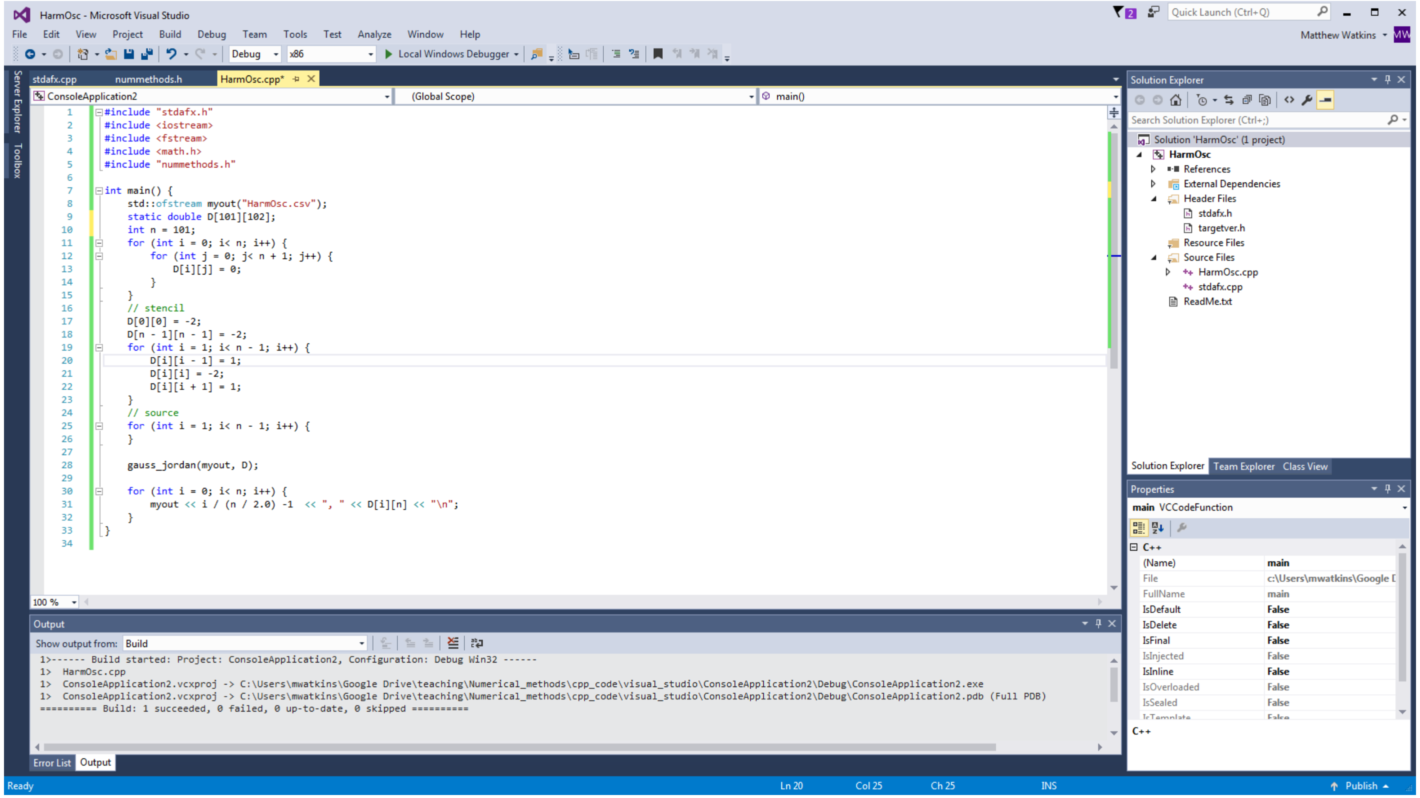
Lots of details inchapters 14 and 15 of Numerical Recipes (http://apps.nrbook.com/c/index.html).

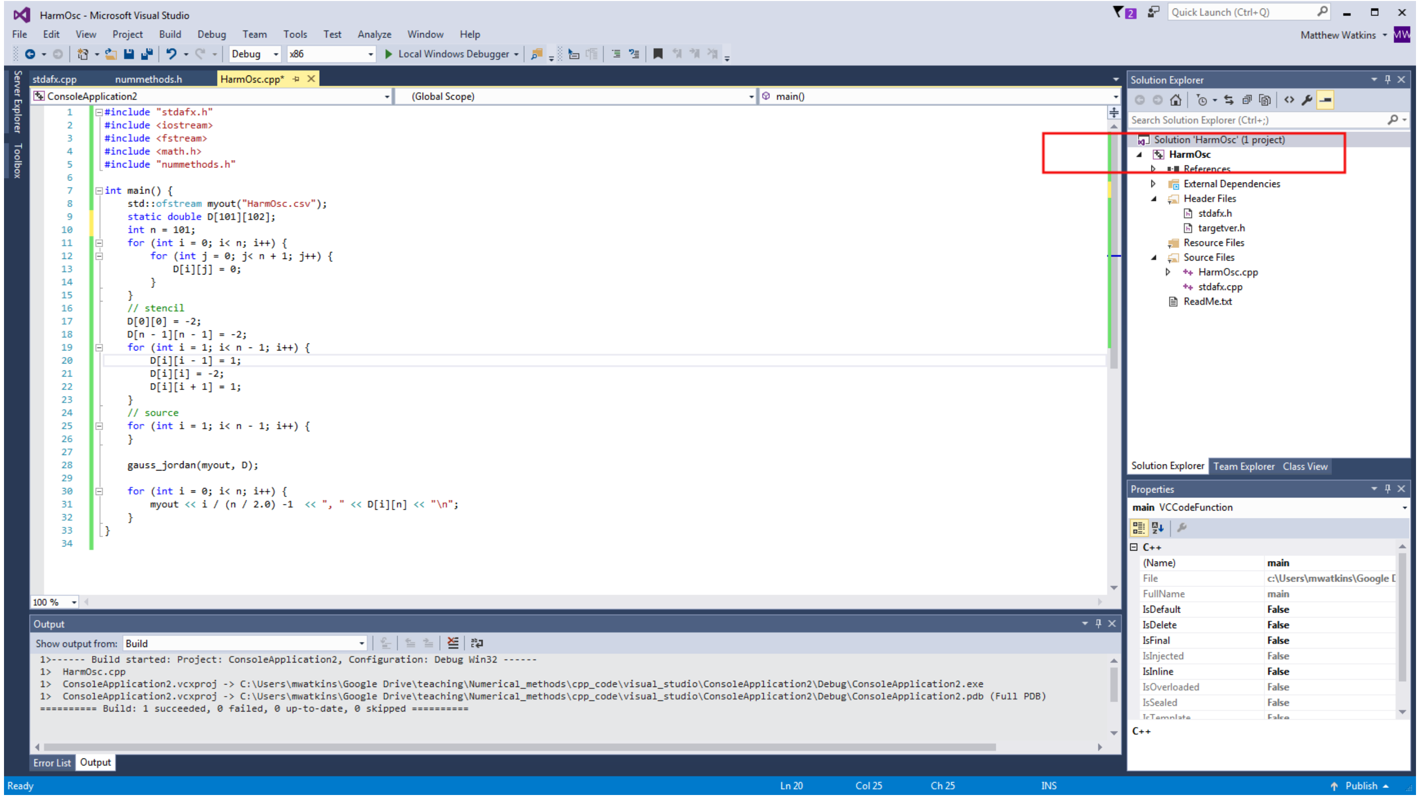
REUSING CODE

You may dislike having lots of code / routines copied around everytime we reuse something

The solution is to **include** other files or, generally, libraries.

We can make our own library file by copying functions other than `main` into a file called `'my_library.h'`

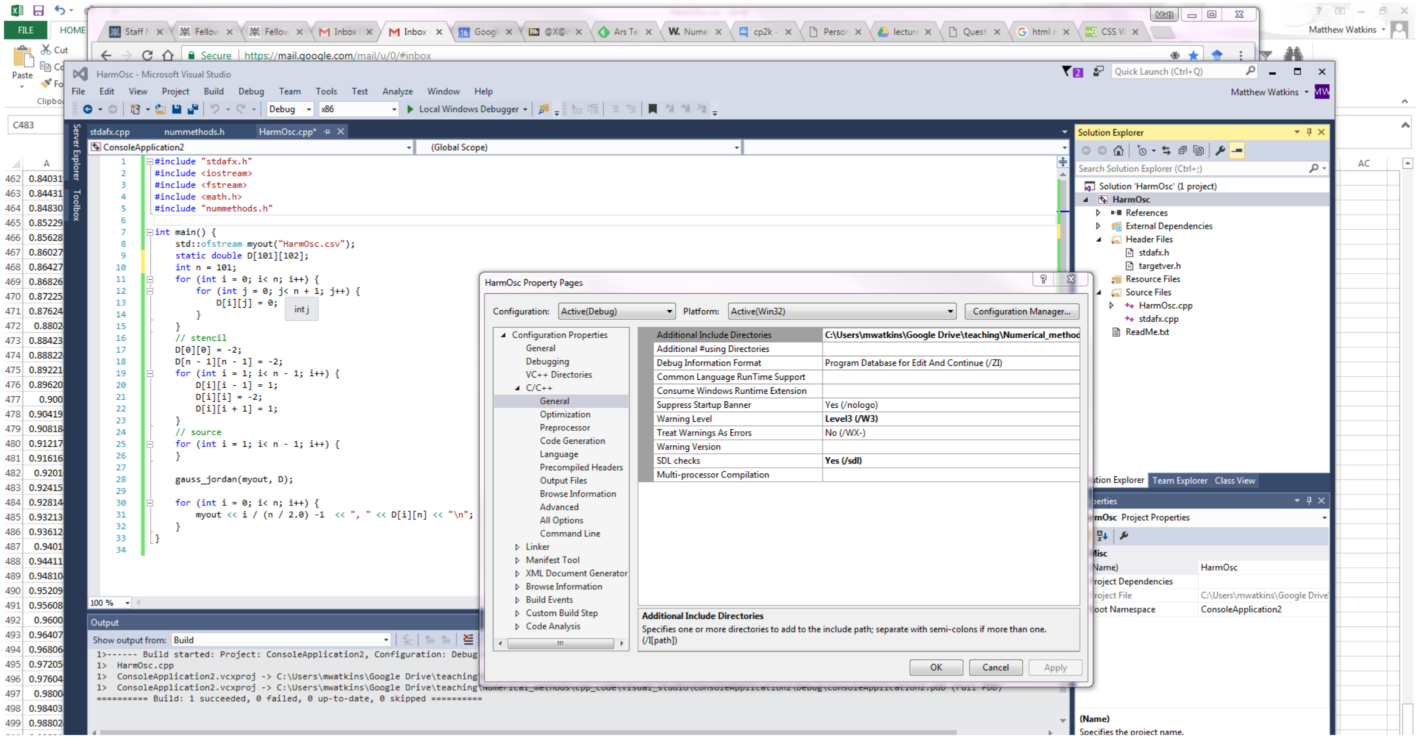

HarmOsc - Microsoft Visual Studio
File Edit View Project Build Debug Team Tools Test Analyze Window Help
Debug
x86
Local Windows Debugger
stdafx.cpp
nummethods.h
HarmOsc.cpp*
ConsoleApplication2
(Global Scope)
main()
#include "stdafx.h"
#include <iostream>
#include <fstream>
#include <math.h>
#include "nummethods.h"
int main() {
std::ofstream myout("HarmOsc.csv");
static double D[101][102];
int n = 101;
for (int i = 0; i< n; i++) {
for (int j = 0; j< n + 1; j++) {
D[i][j] = 0;
}
}
// stencil
D[0][0] = -2;
D[n - 1][n - 1] = -2;
for (int i = 1; i< n - 1; i++) {
D[i][i - 1] = 1;
D[i][i] = -2;
D[i][i + 1] = 1;
}
// source
for (int i = 1; i< n - 1; i++) {
}
gauss_jordan(myout, D);
for (int i = 0; i< n; i++) {
myout << i / (n / 2.0) -1 << ", " << D[i][n] << "\n";
}
}
100 %
Output
Show output from: Build
1>------ Build started: Project: ConsoleApplication2, Configuration: Debug Win32 ------
1> HarmOsc.cpp
1> ConsoleApplication2.vcxproj -> C:\Users\mwatkins\Google Drive\teaching\Numerical_methods\cpp_code\visual_studio\ConsoleApplication2\Debug\ConsoleApplication2.exe
1> ConsoleApplication2.vcxproj -> C:\Users\mwatkins\Google Drive\teaching\Numerical_methods\cpp_code\visual_studio\ConsoleApplication2\Debug\ConsoleApplication2.pdb (Full PDB)
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
Error List
Output
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'HarmOsc' (1 project)
HarmOsc
References
External Dependencies
Header Files
stdafx.h
targetver.h
Resource Files
Source Files
HarmOsc.cpp
stdafx.cpp
ReadMe.txt
Solution Explorer Team Explorer Class View
Properties
main VCCodeFunction
C++
(Name) main
File c:\Users\mwatkins\Google I
FullName main
IsDefault False
IsDelete False
IsFinal False
IsInjected False
IsInline False
IsOverloaded False
IsSealed False
C++
Ready
Ln 20
Col 25
Ch 25
INS
Publish
Quick Launch (Ctrl+Q)
Matthew Watkins



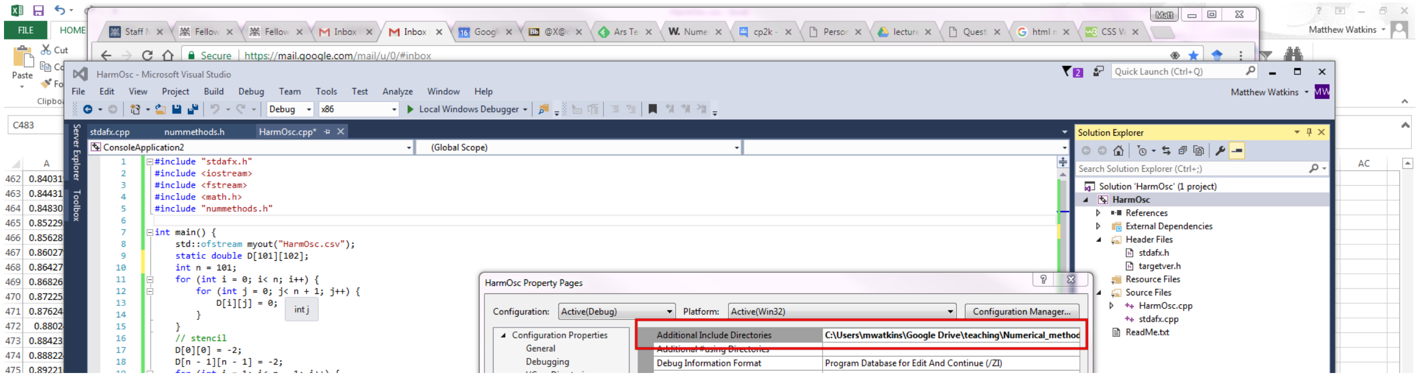
This is my example project. Right click on the **bolded** project name in the pane to right.

Then select `properties`.

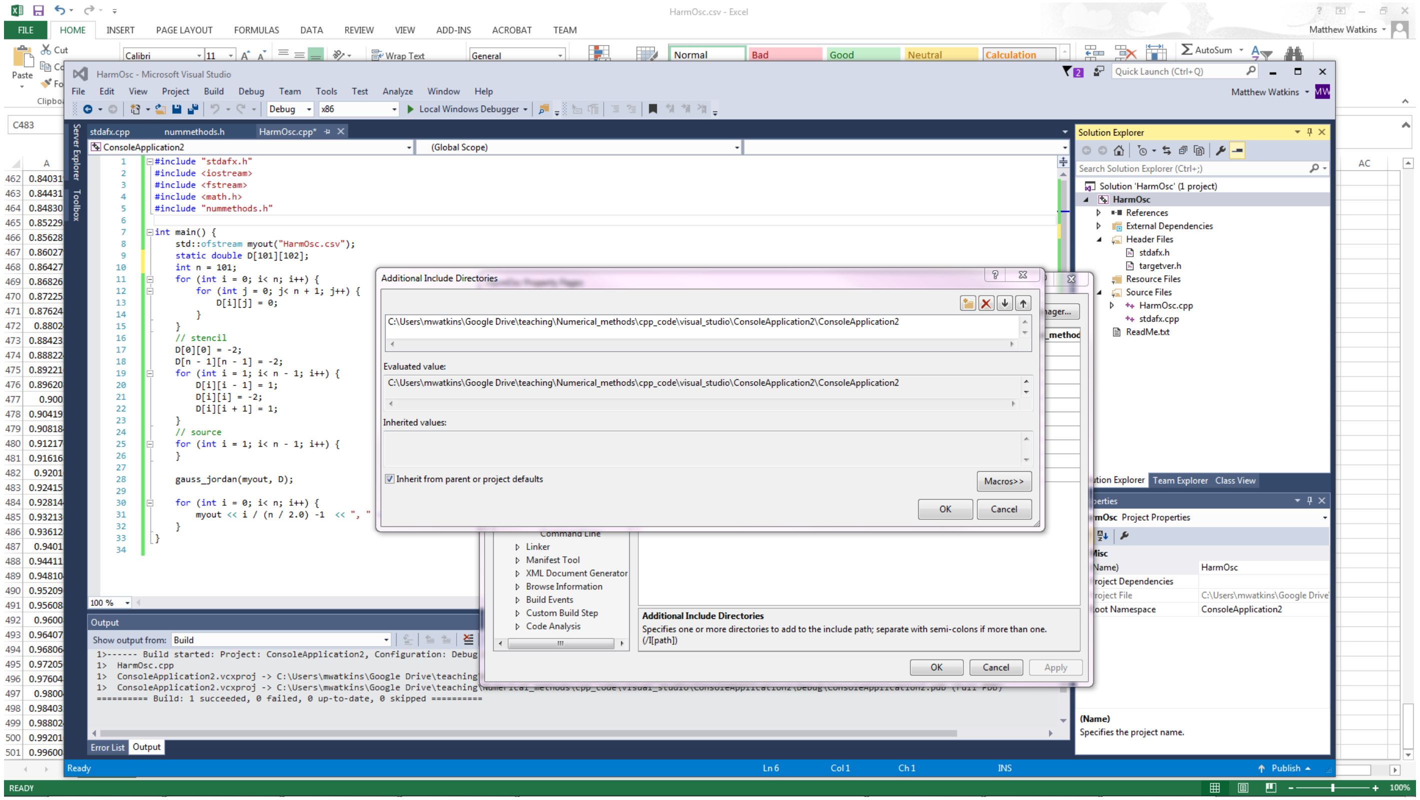
A new window should have appeared, like this:



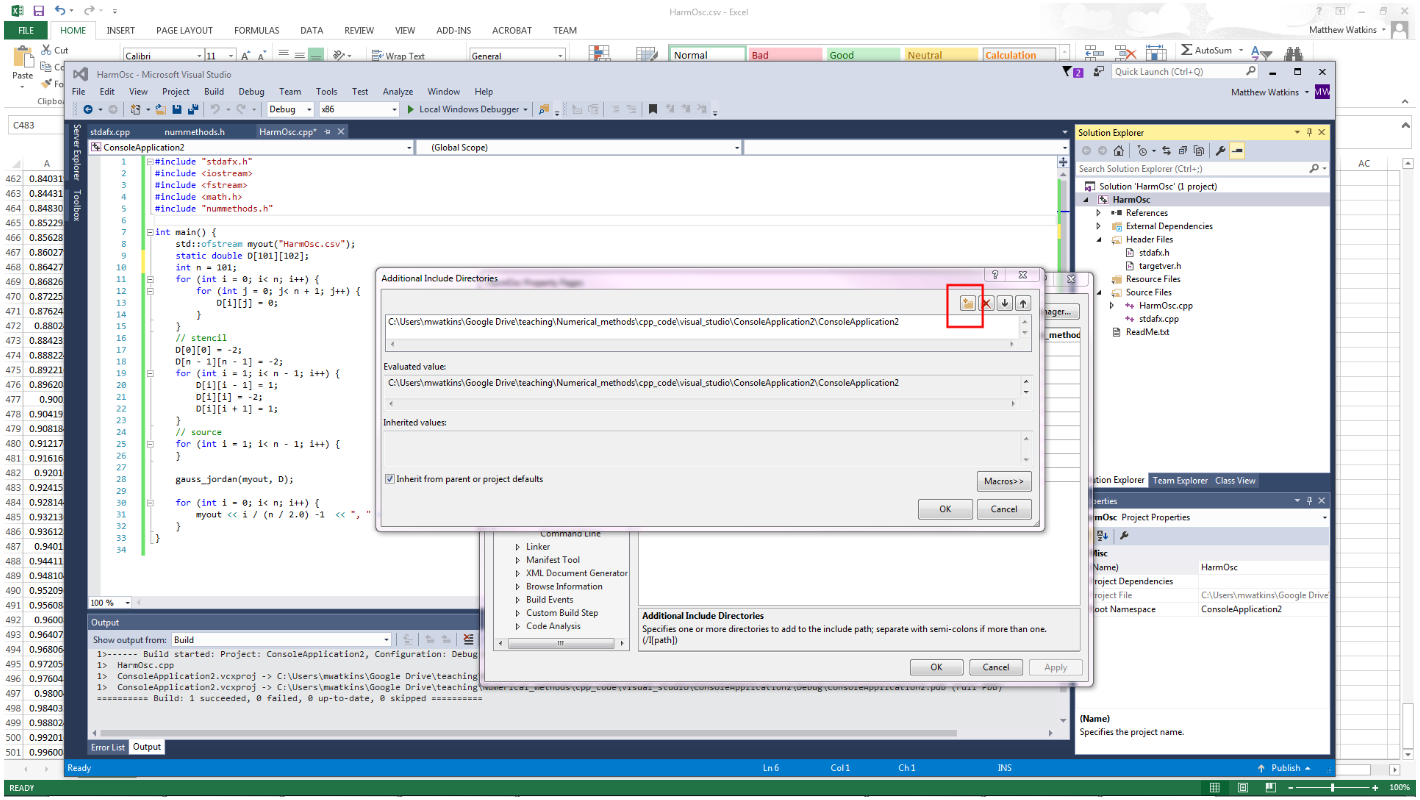
Then click on drop down button in the second column of the the 'Additional Include Directories' row and select edit

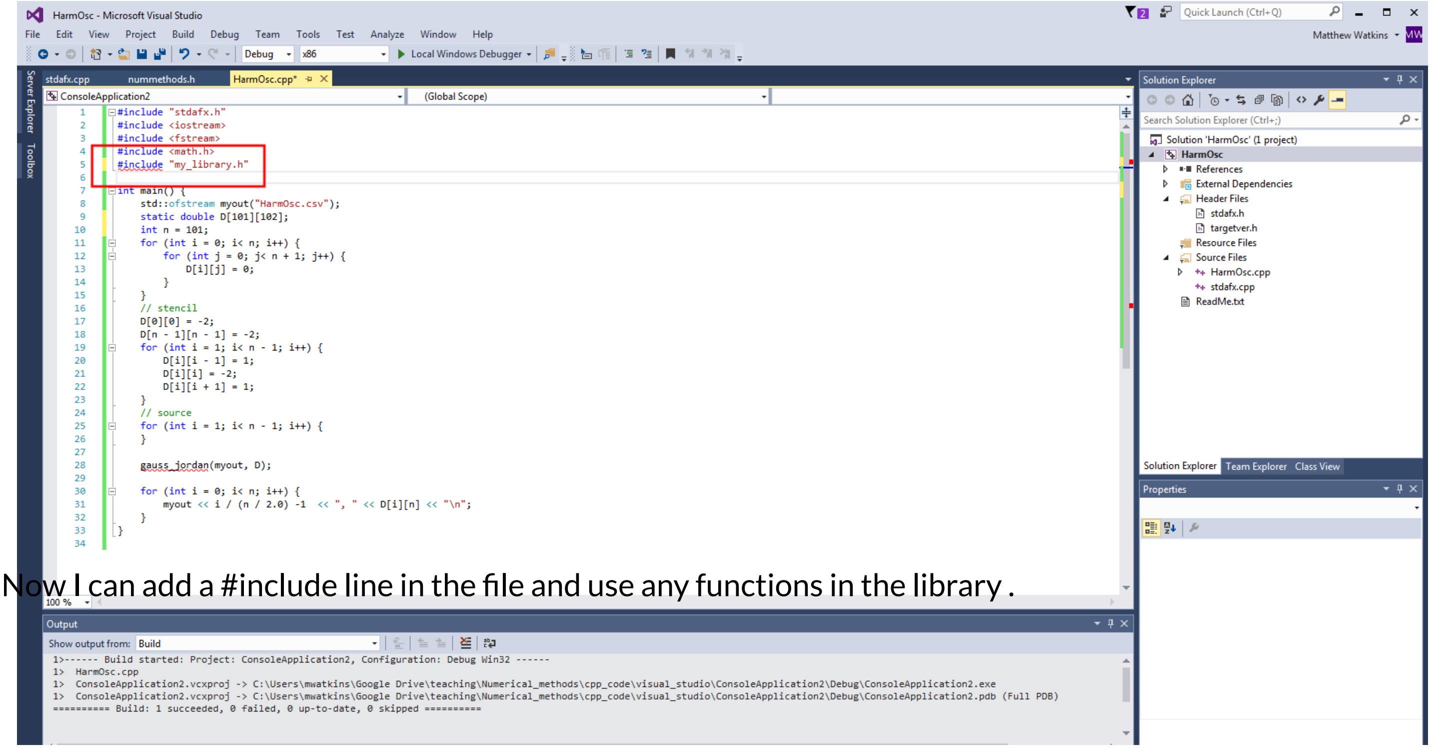


You should see something like this:



Click on the folder icon, and select the directory where you saved your 'my_library.h' file in





Now I can add a #include line in the file and use any functions in the library .

To include the Eigen library,

- save and extract the library (you can find it on the front page of the learning resources tab)
- add the directory containing the extracted Eigen folder to the include path.
- here is an example.

```
#include <iostream>
#include <Eigen/Dense>
using Eigen::Matrix2d;
using namespace std;

int main()
{
  Matrix2d A, b;
  A << 2, -1, -1, 3;
  b << 1, 0, 0, 1;
  cout << "Here is the matrix A:\n" << A << endl;
  cout << "Here is the right hand side b:\n" << b << endl;
```

Here is some code to read in a matrix from a csv file. Assumes the header contains the number of rows then number of columns.

```
#include <iostream>
#include <Eigen/Dense>
#include <fstream>
#include <string>
using Eigen::MatrixXd;
using namespace std;

MatrixXd slurpData()
{
  string nfile, temp;
  int Ncol, Nrow;
  cout << "Insert the name of the file containing the coefficient and the constant terms" <<
"\n";
  cin >> nfile;
  ifstream myFile(nfile);
  // Read the size of the matrix
  if (myFile.is_open()) {
    cout << "\nopened file\n\n";
    getline(myFile, temp, ',');
```

WORKED EXAMPLE OF POLYNOMIAL FIT - USING NEW FANCY MATRICES

- Get the data into a readable form - I suggest using Excel and the 'Text to Columns' function under the 'DATA' tab
- The code to read from a csv file assumes that the first row contains the number of rows, then number of columns
- Save as csv in the same directory as the project you are working on
- Add in the Eigen library
- Write routine(s) to calculate the required sums
- Use Eigen to solve the system of linear equations. See this tutorial (http://eigen.tuxfamily.org/dox/group__TutorialLinearAlgebra.html) for examples. We'll discuss some of the solvers later.

Full code here

```
#include <iostream>
#include <Eigen/Dense>
#include <fstream>
#include <cmath>
#include <string>
using Eigen::MatrixXd;
using Eigen::VectorXd;
using namespace std;

MatrixXd slurpData()
{
  string nfile, temp;
  int Ncol, Nrow;
  cout << "Insert the name of the file containing the coefficient and the constant terms" <<
"\n";
  cin >> nfile;
  ifstream myFile(nfile);
  // Read the size of the matrix
  if (myFile.is_open()) {
```