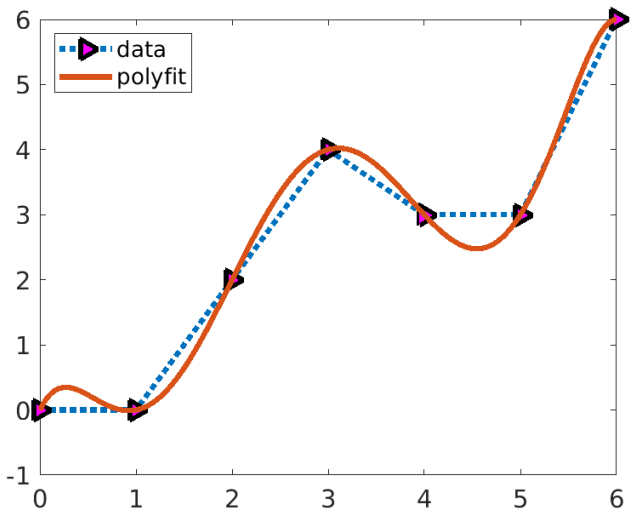
CS319: Scientific Computing (with MATLAB)

# Fitting and Files.

Niall Madden

Week 6: **9am and 4pm**, 15 Feb 2023




Important: you should read:

- Section 7.5 of Learning MATLAB: https://doi-org.nuigalway.idm.oclc.org/10.1137/1.9780898717662
- Sections 13.3 and 11.1 of The MATLAB Guide: https://doi-org.nuigalway.idm.oclc.org/10.1137/1.9781611974669

# In-Class test

We are scheduled to have an in-class test at 4pm on Wednesday of next week (Week 7)...

- Sample question at https://www.niallmadden.ie/2223-CS319/CS319-SampleTest.pdf
- Date time OK? Or when would be better???
- Solutions to the sample questions will be posted on Blackboard in a few days.

# This week, in CS319:

1. 1: Data Fitting with Polynomials
   - Polynomial interpolation
   - polyfit
   - Data
   - Functions
2. 2: Least Squares
   - How does it work?
3. 3. MATLAB Files
   - Load/save
4. Text files
   - fopen() and fclose()
   - fprintf() and fscanf()
5. importdata

You can access these notes, and all others at:

- Bitbucket: https://bitbucket.org/niallmadden/2223-cs319/src/main/https://bitbucket.org/niallmadden/2223-cs319/src/main/
- Web (but only PDFs for slides and labs): https://www.niallmadden.ie/2223-CS319/

# Preview

Much of the focus this week (in lectures, and in Lab 3) is on fitting functions to noisy data.

If we have two points, it is easy to find the equation of the line joining them.

And if you have 10 points, all on the same line, it is also easy to find the equation for the line.

The case of interest is when it *should* be possible to exactly fit a linear polynomial (or some other low-degree polynomial) to some data, but we can't due to errors in the date.

**Least squares** methods are use to find such a polynomial. Their goal is to minimize the different between the data, and the line fitted to it.

We'll try this a approach for data from US population censuses. To work with other data sets, like Ireland's, we'll have to learn how to import data from files.

We finished Week 5 by introducing **polynomial interpolation**. The two problems of interest are:

Data interpolation: given a set of $n+1$ points in 2D, find a polynomial of degree $n$ that goes through them.

Function interpolation: given function find a polynomial that agrees with it at $n$ points.

**Questions:** Why do this?

- To evaluate the polynomial at multiple other points;
- Estimate derivatives;
- Estimate integral;
- ...

**Question:** How can we do this? **Answer:** `polyfit()`

In the following examples, we will construct the polynomial interpolant using the function `polyfit()`.

Syntax: `p = polyfit(x, y, n);`
where `x` and `y` are vectors of the same length, and $n$ is a integer.

If `n = length(x)-1`, then it should interpolate the data. For smaller $n$, a least-squares fit is used (more of that later).

`p` is a vector with $n+1$ entries, where *`p(i)`* is the coefficient of $x^{n+1-i}$ in the polynomial. That is, it represents the polynomial

$$p(1)x^n + p(2)x^{n-1} + \cdots + p(n)x + p(n+1).$$

```
>> x=1:3
x =
     1     2     3
>> y = 1-3*x-2*x.^2
y =
     6    17    34
>> p = polyfit(x,y,2)
p =
   -2.0000   -3.0000    1.0000
```

## Polynomial Data Interpolation

Given the $n+1$ points $(x_0, y_0)$, $(x_1, y_1)$, $\ldots$, $(x_n, y_n)$, we want to find the polynomial $p_n$ such that

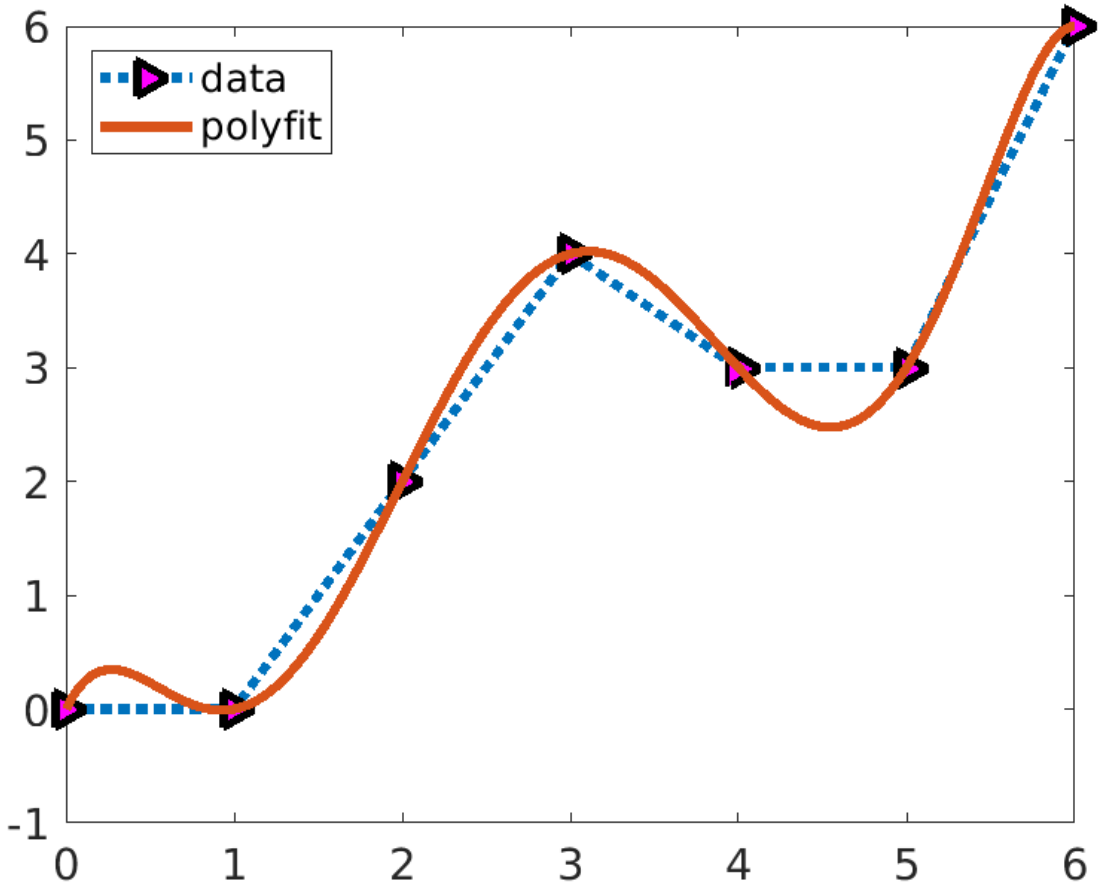
$$p_n(x_0) = y_0, p_n(x_1) = y_1, \quad \ldots, \quad p_n(x_n) = y_n.$$

PolyDataInterp.m

```
x = 0:6;
y = [0, 0, 2, 4, 3, 3, 6];
p = polyfit(x, y, length(x)-1);

X = linspace(x(1), x(end), 1001); % Points for plotting;
Y = polyval(p, X);
plot(x, y, ':>', X, Y, '-', ...
    'LineWidth',3, 'MarkerSize', 10,...
    'MarkerFaceColor', 'magenta', 'MarkerEdgeColor', 'k');
legend('data', 'polyfit', 'location', 'northwest')
set(gca, 'FontSize', 14)
```

PolyDataInterp.m

```
x = 0:6;
y = [0, 0, 2, 4, 3, 3, 6];
p = polyfit(x, y, length(x)-1);
```



## Polynomial Function Interpolation

In practice, this works very similarly to data interpolation: given a set of points $x_0$, $x_1$, ..., $x_n$, and a function $f$, we compute the interpolant to $(x_0, f(x_0))$, $(x_1, f(x_0))$, ..., $(x_n, f(x_n))$.

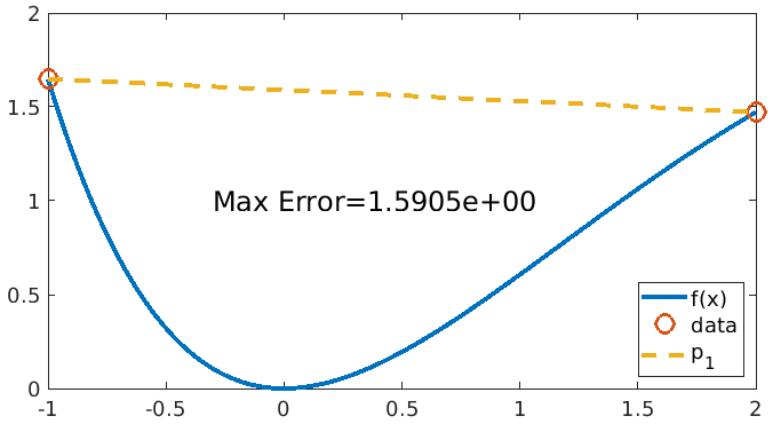
The main differences are

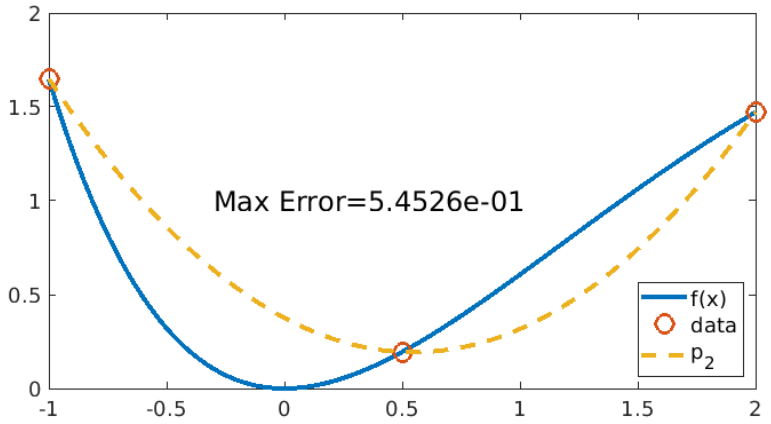
- We are free to choose any points in the function's domain;
- We can vary the number of points easily;
- We can estimate errors: $\|f(x) - p_n(x)\|$.

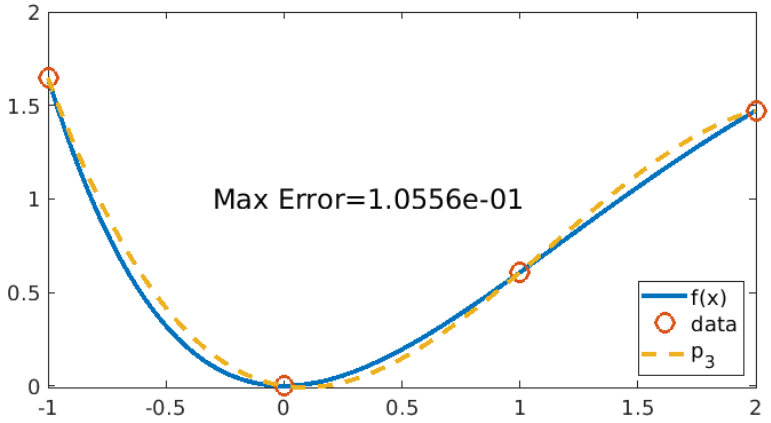
In this example, we'll construct the polynomial interpolant to $f(x) = x^2 e^{-x/2}$, in the interval $[-1, 2]$.

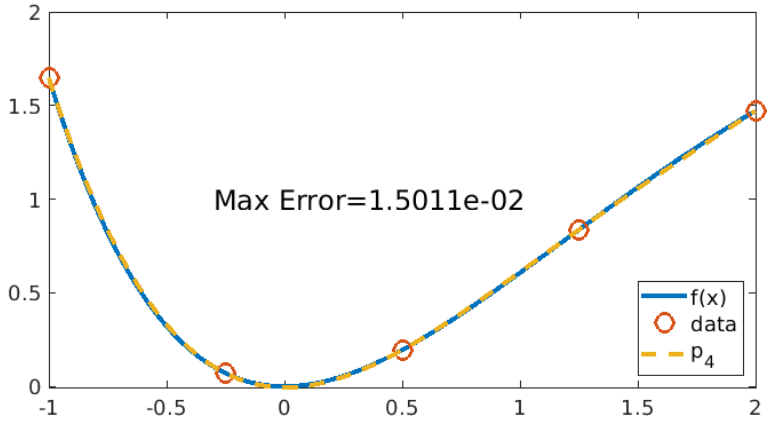
PolyFunctionInterpolation.m

```
f = @(x)(x.^2).*exp(-x/2);
for n=1:4
    xp = linspace(-1,2,n+1);
    p = polyfit(xp, f(xp), n);

    X = linspace(-1,2, 1001); % Points for plotting;
    Y = polyval(p, X);
    Error = norm(f(X)-Y,'inf');
    fprintf('n=%2d, Error=%10.5e\n', n, Error)
    subplot(2,2,n);
    plot(X, f(X), xp, f(xp), 'o', X, Y, '--', ...
        'LineWidth',3, 'MarkerSize', 12);
    leg_str = sprintf('p_{%d}', n);
    legend('f(x)', 'data', leg_str, 'FontSize', 14, ...
        'location', 'southeast')
    Error_str = sprintf('Max Error=%5.4e', Error);
    text(-0.3,1, Error_str, 'FontSize', 18)
    set(gca, 'FontSize', 14)
end
```


Max Error=1.5905e+00
f(x)
data
p1


Max Error=5.4526e-01
f(x)
data
p2


Max Error=1.0556e-01
f(x)
data
p3


Max Error=1.5011e-02
f(x)
data
p4

# 2: Least Squares

In all the previous examples,

- We fitted polynomials of degree $n$ to $n+1$ points, and
- There was no noise in the data.

In many applications, we wish to use low-order polynomials,

- because there are more stable;
- the underlying data is noisy (see Lab 3!).

**Applications:**

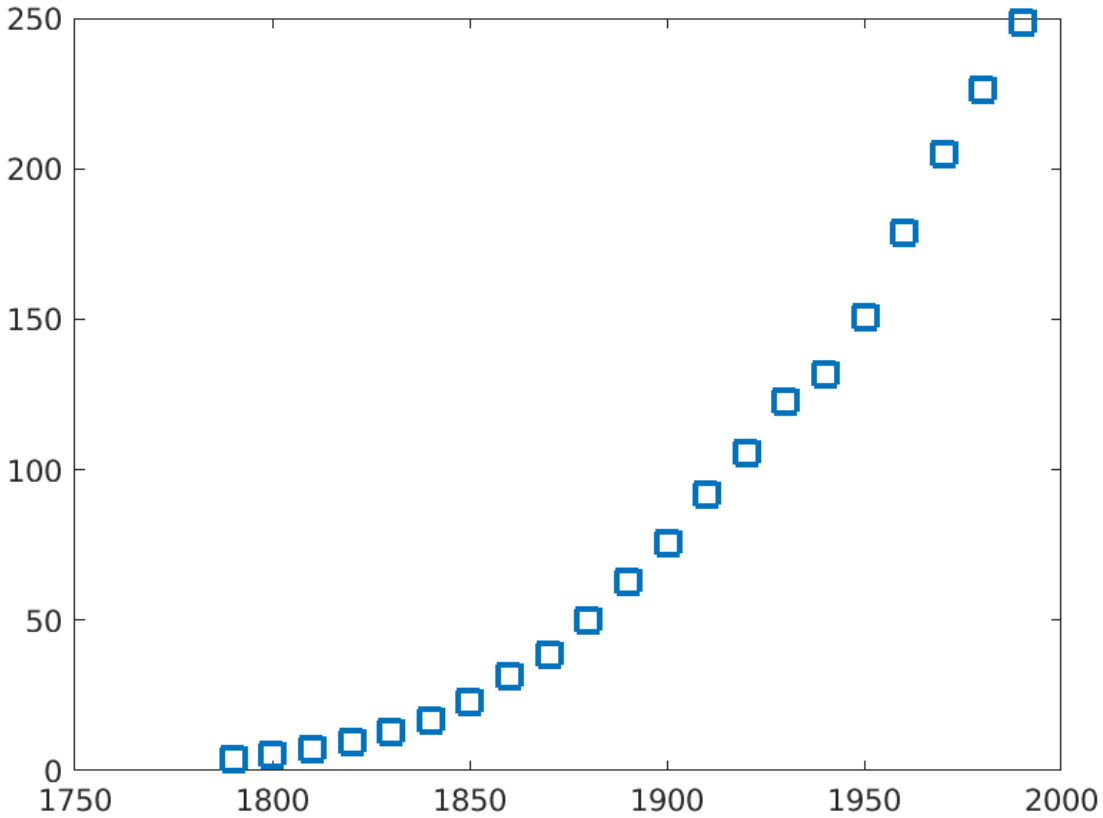
1. Estimating values at non-observed points;
2. Estimating growth rates;
3. **Filtering out noise**.

## Example: US Census Data

1. Construct and plot linear, quadratic and cubic fits to the US census data in *census.mat*.
2. For each, estimate the "root mean squared error" ("$l^2$", or "Euclidian" norm to its friends.
3. For each, what is the estimated rate of growth in 1990?
4. For each, what is the estimated population in 1985 (not a census year)?
5. For each, what is the estimated population in 2000, 2010, and 2020, and how accurate is that?

# 2: Least Squares

```
load census;
plot(cdate, pop, 's', 'MarkerSize', 10, 'LineWidth', 2);
```



# 2: Least Squares

We can construct the linear fit as follows:

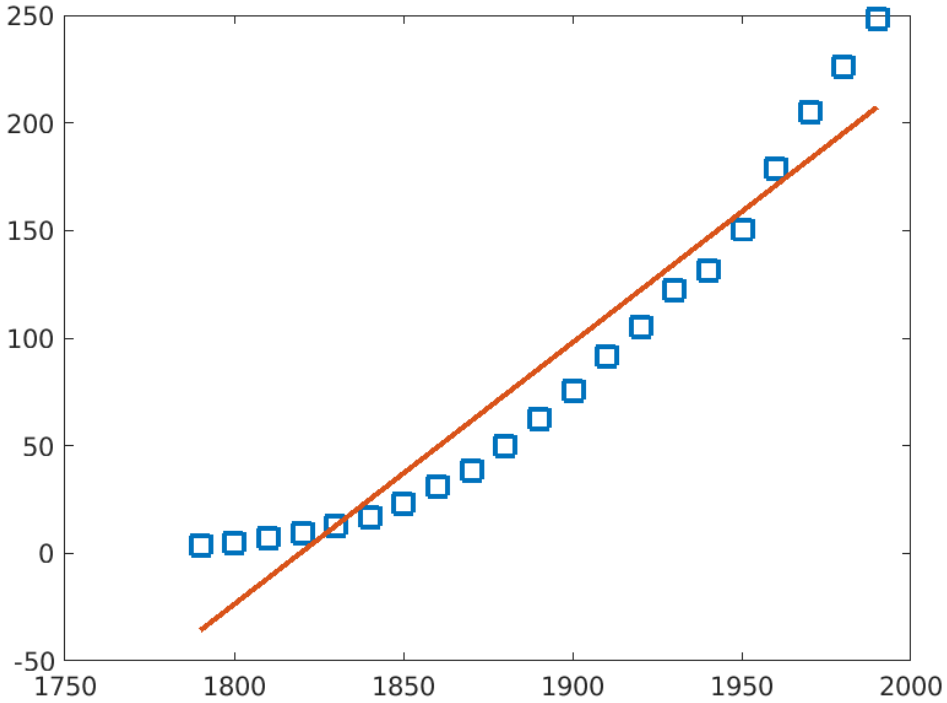
USCensusLeastSquares.m

```
load census;
p1 = polyfit(cdate, pop, 1); %% NOTE: degree=1
t = linspace(1790, 1990,1001);
plot(cdate, pop, 's', t, polyval(p1,t),...
    'LineWidth', 3, 'MarkerSize',10);
Diff1 = norm(pop - polyval(p1,cdate));
dp1 = polyder(p1);
fprintf('p=1, Error=%5.2f, Growth (1990)=%5.2f\n', ...
    Diff1, polyval(dp1, 1990))

Extrap = polyval(p1, 2000:10:2020);
Actual = [281.4, 308.7, 331.5];

fprintf('2000: pop estimate (actual) %.1f (%.1f)\n', ...
    Extrap(1), Actual(1));
fprintf('2010: pop estimate (actual) %.1f (%.1f)\n', ...
    Extrap(2), Actual(2));
fprintf('2020: pop estimate (actual) %.1f (%.1f)\n', ...
    Extrap(3), Actual(3));
```

# 2: Least Squares

Output:

```
p=1, Error=98.78, Growth Estimate (1990)= 1.22
2000: population estimate (actual) 219.5 (281.4)
2010: population estimate (actual) 231.6 (308.7)
2020: population estimate (actual) 243.8 (331.5)
```

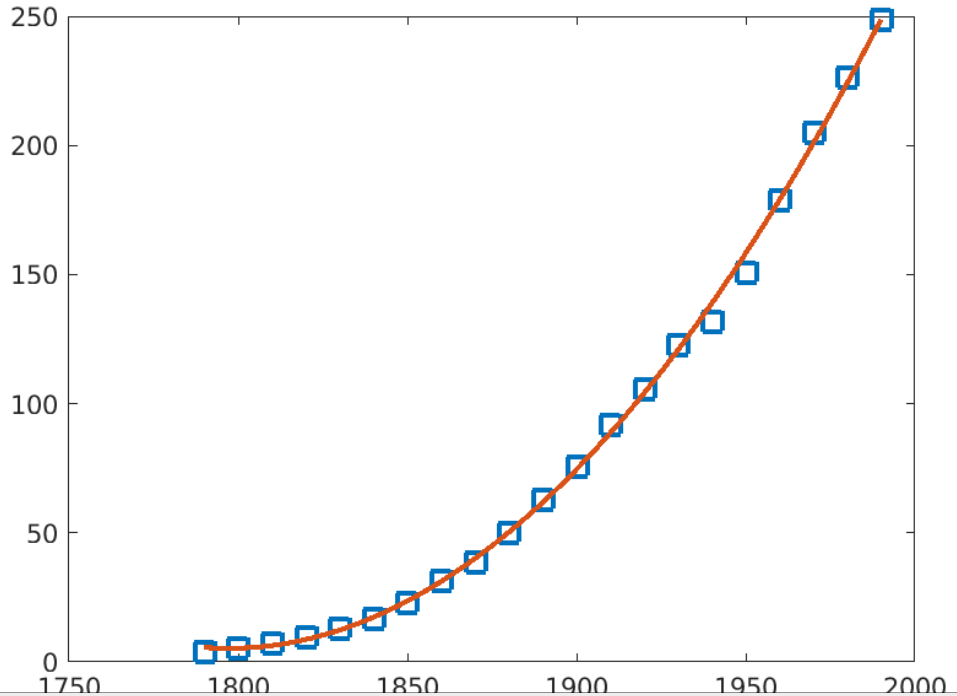


# 2: Least Squares

You can change the `polyfit` line to get a higher-order estimate.

The result for a quadratic would be: Output:

```
p=2, Error=12.61, Growth Estimate (1990)= 2.52
2000: population estimate (actual) 274.6 (281.4)
2010: population estimate (actual) 301.8 (308.7)
2020: population estimate (actual) 330.3 (331.5)
```



# 2: Least Squares

Experiment with higher-order interpolation. Convince yourself that the quadratic fit is most appropriate.

Suppose we want to find the polynomial $p_2 = a_2x^2 + a_1x + a_0$, that fits some data. If it is to fit the point $(x_i, y_i)$, that means

$$a_2x_i^2 + a_1x_i + a_0 = y_i.$$

With 3 unknowns we need 3 equations. So if we have three points is there is an exact solution. The equations would be

$$\begin{aligned} a_2x_1^2 + a_1x_1 + a_0 &= y_1 \\ a_2x_2^2 + a_1x_2 + a_0 &= y_2 \\ a_2x_2^2 + a_1x_3 + a_0 &= y_3. \end{aligned}$$

We could write this as a matrix-vector equation: $Xa = y$, i.e.,

$$\begin{bmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ x_3^2 & x_3 & 1 \end{bmatrix} \begin{bmatrix} a_2 \\ a_1 \\ a_0 \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \end{bmatrix}$$

But in these problems we have many more equations than unknowns:

$$\begin{bmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ x_3^2 & x_3 & 1 \\ \vdots & \vdots & \vdots \\ x_n^2 & x_n & 1 \end{bmatrix} \begin{bmatrix} a_2 \\ a_1 \\ a_0 \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

Now there is no (exact) solution to this problem: there is no vector $a$ for which $Xa = y$.

But we can find a solution that is "better" than the rest. For that we need some way to understand "norms". (See next slide).

[Stuff about norms, hand-written in class].

Recall again the problem is

$$\begin{bmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 2 \\ x_3^2 & x_3 & 3 \\ \vdots & \vdots & \vdots \\ x_n^2 & x_n & n \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

where $n > 3$. There is no solution to $Xa = y$. That is, $\|Xa - y\| > 0$ for all $a$.

The "least squares" solution is the one for which, the residual, $Xa - y$, is as small as possible.

In a linear algebra course, we would prove such a vector exists, and explain how to find it. But here we will just compute it:

```
1 X = [x.^2, x, x.^0];
  a = X\y
```

Try this for the US Census data.

You should take `x=cdate`, `y=pop`. Compare the resulting $a$ with the value of $p$ computed by `polyfit()`.

Try this for the US Census data.

You should take `x=cdate`, `y=pop`. Compare the resulting $a$ with the value of $p$ computed by `polyfit()`.

# 3. MATLAB Files

In the previous section, we accessed US census data just using `load census`. That is because that data comes packaged with MATLAB, since it is frequently used as an example to demonstrate MATLAB capabilities.

In most situations, however, such data is stored in files, and we have to learn how to import it in order to use it.

MATLAB has a bewildering array of ways of creating and reading files.

The simplest involve the `save` and `load` functions. Syntax:

- `save;` Saves all variables in the workspace to a file called *matlab.mat*
- `save('mydata');` saves all variables in the workspace to a file called *mydata.mat*
- `save('mydata', 'p1', 'p2');` saves variables *p1* and *p2* to a file called *mydata.mat*

The format used is a native MATLAB binary format. You can also specific other formats, but I rarely find this to be useful.

To load saved data, use the `load()` function. Syntax is the same.

# Text files

The MATLAB format is mainly useful for saving data between sessions, but not for sharing data between different applications.

For that it is best to use plain text files ("ascii").

The main functions we will use are

- `fopen()` : opens a file, and allows us to read from it, or write to it;
- `fscanf()` : reads data from a file.
- `fprintf()` : write data to a file.
- `fclose()` : close a file.

To read from a file, or write to it, the file must by "open". To open a file, use the `fopen()` function.

We'll use it in two ways:

- `fid1 = fopen('NameOfInFile.txt', 'r')`. This opens a file, in the present folder called `NameOfInFile.txt`. It is opened in `read` mode. This means:
  - It must already exist. If not, it returns the value `-1`.
  - Once open, we can read data from the file, by using the **file identifier** stored in `fid1`.
- *`fid2 = fopen('NameOfOutFile.txt', 'w')`*. This opens a file, in the present folder called *`NameOfOutFile.txt`*, in *`write`* mode. This means:
  - It does not have to already exist. If it does, its contents are discarded.
  - Once open, we can **write** data to the file, using the identifier *`fid2`*.

Once we are finished with the file, we should "close" it: `fclose(fid1)`. Also useful: `fclose(all)`.

The *fprintf()* function for outputting formatted data to the screen can also write to a file. Example (taken from Section 13.3 of Higham+Higham).

```
A = [30 40 60 70];
fid = fopen('myoutput','w');
fprintf(fid,'%g miles/hour = %g kilometers/hour\n', ...
      [A; 8*A/5]);
fclose(fid);
```

Note: this also shows how *fprintf()* operates on a vector.

To read in this data:

```
>> fid = fopen('myoutput','r');
>> X = fscanf(fid,'%g miles/hour = %g kilometers/hour', [2,
     inf])
```

There are many more related functions. See *doc iofun* for a full list.

# importdata

Sometimes, data is stored in a simple, structured manger, and there are specialised functions to loading it.

By far the most common way for sharing tabulated data in plain text is as a Comma Separated Values (CSV) file. See, for example, the file *IrelandPopulation.csv*

```
1841, 6528799, 3222485, 3306314, 103
1851, 5111557, 2494478, 2617079, 105
1861, 4402111, 2169042, 2233069, 103
1871, 4053187, 1992468, 2060719, 103
1881, 3870020, 1912438, 1957582, 102
1891, 3468694, 1728601, 1740093, 101
1901, 3221823, 1610085, 1611738, 100
...
```

Here is some code the load and plot it...

# importdata

To read this, try

```
>> IE = importdata('IrelandPopulation.csv')
```

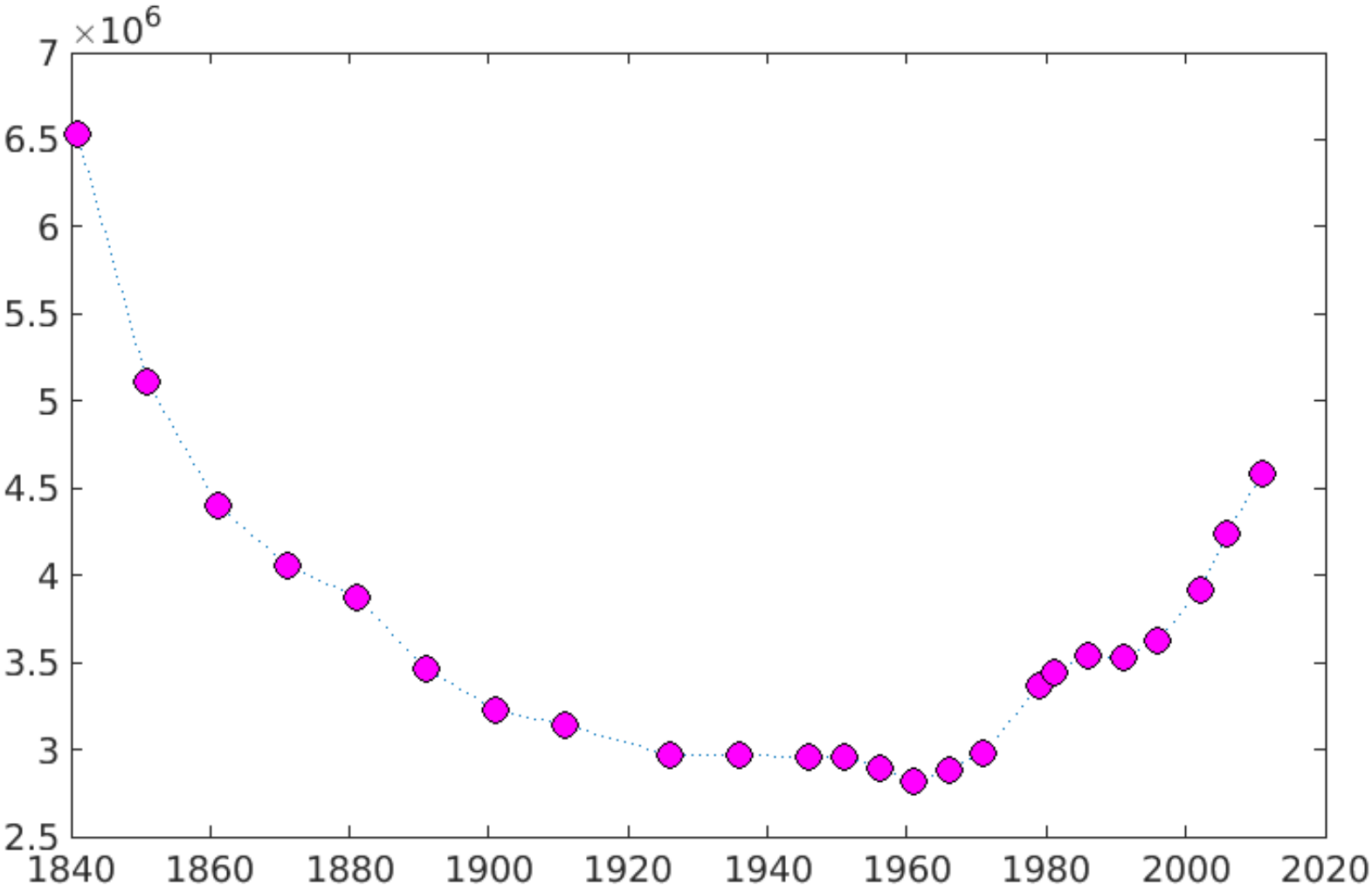
This will create a $24 \times 5$ matrix, *IE* with years in Column 1, population in Column 2, etc.

# importdata

IrelandPopulationData.m

```
clear;

IE = importdata('IrelandPopulation.csv');

Years = IE(:,1); % col 1 has dates
Total = IE(:,2); % Total population
Males = IE(:,3); % Number of males
Females = IE(:,4); % Number of females

figure(1);
plot(Years, Total, ':o', 'MarkerSize',10, ...
'MarkerFaceColor', 'magenta', ...
'MarkerEdgeColor', 'k');

figure(2);
plot(Years, Males, ':o',  ...
   Years, Females, ':v', 'MarkerSize',10, ...
  'LineWidth',3)
legend('Males', 'Females')
```

# importdata


×10^6
7
6.5
6
5.5
5
4.5
4
3.5
3
2.5
1840
1860
1880
1900
1920
1940
1960
1980
2000
2020

## Questions

1. We found the US census data could be well approximated by a quadratic. What polynomial degree(s) would be useful for Irish population data.
2. Other than eye-balling the data, how could we decide the "right" degree of polynomial to use?
3. How could we check if the fit was good for extrapolation?

# importdata

The *importdata()* function is actually a wrapper from several others, including csvread, readmatrix, xlsread, imread...

But it can be more general then these. For example, it can handle CSV files with headers:

```
>> IE = importdata('IrelandPopWithHeaders.csv')
```

```
IE =
  struct with fields:
      data: [24×5 double]
  textdata: {'year'  'total population'  'males'  'females'  'females per 100
  colheaders: {'year'  'total population' 'males'  'females'  'females per 100
```

We'll learn more about cells and structs next week.

To save data, to load into other applications, I find *savematrix()* to be very useful.