Math 504
Introductory Numerical Analysis
Numerical Linear Algebra

Spring 2011

Prof. D Sulsky

Homework 08 Solutions

May 3, 2011

#	*score*
1	
2	
3	
4	
5	
Total	

Daniel Topa
(505) 504-5986
dantopa@gmail.com

Homework #8 – Computer Assignment due 4/28/11

1. The following Census Bureau figures give the population of the United States by year.

Year	Population
1900	75,994,575
1910	91,972,266
1920	105,710,620
1930	122,775,046
1940	131,669,275
1950	150,697,361
1960	179,323,175
1970	203,235,298

   Fit these data in a least squares sense by second and fourth degree polynomials, use the normal equations approach. (You may use matlab, lapack or clapack routines to solve the equations.) What is the 'predicted' population by 1990? Approximately what is the condition number of the system you solved? How did you do the approximation?

2. Repeat problem 1 using QR factorization. You may use matlab, lapack or clapack routines. Extra credit: write your own QR factorization.

3. Fit the data by a function of the form

$$y(t) = c_1 + c_2(t - 1900) + c_3 \exp(d(t - 1900))$$

   This function involves four parameters, c_1, c_2, c_3 and d. The problem is linear in c_1, c_2, c_3, but nonlinear in d. Let y be the 8-vector of population data, $\mathbf{c}$ be the unknown 3-vector of coefficients. Let $A(d)$ be the 8×3 coefficient matrix of $\mathbf{c}$ obtained by plugging in the 8 years of data from the table. Minimize the two norm of $\|A(d)\mathbf{c} - \mathbf{y}\|$ with respect to $\mathbf{c}$ and d. You should use a combination of QR routines and a routine to compute the minimum of a function of one variable. Find the values of $\mathbf{c}$ and d that give the best fit. Predict the 1980 and 1990 population.

4. Make tables summarizing your results and discuss the merits of each approach. What is your best prediction of the 1990 population?

5. Extra Credit: Repeat 2-3 using matlab, lapack or clapack routines for the SVD in place of QR factorization. What are the singular values of the coefficient matrices?

1 Polynomial least squares: normal equations

1. The following Census Bureau figures give the population of the United States by year.

Year	Population
1900	75,994,575
1910	91,972,266
1920	105,710,620
1930	122,775,046
1940	131,669,275
1950	150,697,361
1960	179,323,175
1970	203,235,298

   Fit these data in a least squares sense by second and fourth degree polynomials, use the normal equations approach. (You may use matlab, lapack or clapack routines to solve the equations.) What is the 'predicted' population by 1990? Approximately what is the condition number of the system you solved? How did you do the approximation?

1.1 Polynomial fitting functions

Given a year y_k and a population p_k, for $k = 1, 8$, find least squares fits to the polynomial functions. This implies finding the vector x which solves

$$\min_x \|\mathbf{A}x - p\|_2 . \tag{1.1}$$

In order to guide development of the numeric modules, the problem was first solved in arbitrary precision arithmetic. These results are shown on the following pages.

The system was solved using the pseudoinverse:

$$\mathbf{A}x = p \quad \longrightarrow \quad x = \mathbf{A}^{+}p + \alpha \cdot \mathcal{N}\left(\mathbf{A}^{T}\right). \tag{1.2}$$

Here $\alpha \in \mathbb{C}^{n\times 1}$ is a vectory of arbitrary constants and $\mathcal{N}\left(\mathbf{A}^{T}\right)$ represents the span of the null space of $\mathbf{A}^{T}$. A more formal discussion of this solution is deferred until section (5.1.3).

quadratic fit

create system matrix

```
A = {1, #, #^2} & /@ year;
% // mf
```

$$\begin{pmatrix} 1 & 1900 & 3\,610\,000 \\ 1 & 1910 & 3\,648\,100 \\ 1 & 1920 & 3\,686\,400 \\ 1 & 1930 & 3\,724\,900 \\ 1 & 1940 & 3\,763\,600 \\ 1 & 1950 & 3\,802\,500 \\ 1 & 1960 & 3\,841\,600 \\ 1 & 1970 & 3\,880\,900 \end{pmatrix}$$

solve for polynomial amplitudes

```
soln = PseudoInverse[A].census
% // N
```

$$\left\{\frac{1\,059\,748\,310\,373}{28}, -\frac{4\,277\,690\,402}{105}, \frac{46\,114\,751}{4200}\right\}$$

$\{3.78482 \times 10^{10}, -4.07399 \times 10^{7}, 10\,979.7\}$

define the solution function

```
Clear[y2];
y2[t_] := soln.Table[t^k, {k, 0, 2}]
```

extrapolation

```
y2[1990]
% // N
FortranForm[%]
```

$$\frac{7\,180\,773\,279}{28}$$

2.56456×10^{8}

```
2.564561885357143e8
```

error

```
r = census - y2[year];
r.r // N
```

8.74083×10^{13}

quartic fit

create system matrix

```
dmax = 4;
```

```
A = Table[#^k, {k, 0, dmax}] & /@ year;
% // mf
```

$$\begin{pmatrix} 1 & 1900 & 3\,610\,000 & 6\,859\,000\,000 & 13\,032\,100\,000\,000 \\ 1 & 1910 & 3\,648\,100 & 6\,967\,871\,000 & 13\,308\,633\,610\,000 \\ 1 & 1920 & 3\,686\,400 & 7\,077\,888\,000 & 13\,589\,544\,960\,000 \\ 1 & 1930 & 3\,724\,900 & 7\,189\,057\,000 & 13\,874\,880\,010\,000 \\ 1 & 1940 & 3\,763\,600 & 7\,301\,384\,000 & 14\,164\,684\,960\,000 \\ 1 & 1950 & 3\,802\,500 & 7\,414\,875\,000 & 14\,459\,006\,250\,000 \\ 1 & 1960 & 3\,841\,600 & 7\,529\,536\,000 & 14\,757\,890\,560\,000 \\ 1 & 1970 & 3\,880\,900 & 7\,645\,373\,000 & 15\,061\,384\,810\,000 \end{pmatrix}$$

solve for polynomial amplitudes

```
soln = PseudoInverse[A].census
% // N
```

$$\left\{-\frac{492\,144\,795\,696\,934}{7}, \frac{45\,465\,974\,831\,902}{315}, -\frac{2\,933\,002\,077\,157}{26\,400}, \frac{15\,047\,640\,851}{396\,000}, -\frac{12\,864\,169}{2\,640\,000}\right\}$$

$\{-7.03064\times10^{13}, 1.44336\times10^{11}, -1.11099\times10^{8}, 37\,999.1, -4.87279\}$

define the solution function

```
Clear[fcn];
fcn[t_] := soln.Table[t^k, {k, 0, dmax}]
```

extrapolation

```
fcn[1990]
% // N
FortranForm[%]
```

$$\frac{5\,514\,075\,040}{21}$$

2.62575×10^{8}

```
2.625750019047619e8
```

error

```
r = census - fcn[year];
r.r // N
```

3.53053×10^{13}

1.2 Quadratic fit

The next step was to generate and solve the system using normal equations. These are the routines that will be used to provide both the exact and the double precision answers.

The trial function is this:

$$p(y) = a_0 + a_1 y + a_2 y^2. \tag{1.3}$$

The goal is to find the amplitudes a which best characterize the data in the least squares sense.

1.2.1 The linear system

The linear system is cast in this fashion:

$$\mathbf{A}y = p$$

$$\begin{pmatrix} 1 & y_1 & y_1^2 \\ 1 & y_2 & y_2^2 \\ 1 & y_3 & y_3^2 \\ 1 & y_4 & y_4^2 \\ 1 & y_5 & y_5^2 \\ 1 & y_6 & y_6^2 \\ 1 & y_7 & y_7^2 \\ 1 & y_8 & y_8^2 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \\ p_6 \\ p_7 \\ p_8 \end{pmatrix}. \tag{1.4}$$

1.2.2 The normal equations

The normal equations are formed by premultiplying the above linear system by the transpose of the system matrix. This guarantees that the vector on the right-hand side will be in the range of $\mathbf{A}^T$. The normal equations for the quadratic fit are now:

$$\mathbf{A}^T\mathbf{A}y = \mathbf{A}^T p$$

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ y_1 & y_2 & y_3 & y_4 & y_5 & y_6 & y_7 & y_8 \\ y_1^2 & y_2^2 & y_3^2 & y_4^2 & y_5^2 & y_6^2 & y_7^2 & y_8^2 \end{pmatrix} \begin{pmatrix} 1 & y_1 & y_1^2 \\ 1 & y_2 & y_2^2 \\ 1 & y_3 & y_3^2 \\ 1 & y_4 & y_4^2 \\ 1 & y_5 & y_5^2 \\ 1 & y_6 & y_6^2 \\ 1 & y_7 & y_7^2 \\ 1 & y_8 & y_8^2 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ y_1 & y_2 & y_3 & y_4 & y_5 & y_6 & y_7 & y_8 \\ y_1^2 & y_2^2 & y_3^2 & y_4^2 & y_5^2 & y_6^2 & y_7^2 & y_8^2 \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \\ p_6 \\ p_7 \\ p_8 \end{pmatrix}. \tag{1.5}$$

It's good to write these equations out once a year or so.

Personal preference involves introducing the following vectors:

$$\begin{aligned}
Y_0 &= \left(\begin{array}{cccccccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{array} \right)^T, \\
Y_1 &= \left(\begin{array}{cccccccc} y_1 & y_2 & y_3 & y_4 & y_5 & y_6 & y_7 & y_8 \end{array} \right)^T = y, \\
Y_2 &= \left(\begin{array}{cccccccc} y_1^2 & y_2^2 & y_3^2 & y_4^2 & y_5^2 & y_6^2 & y_7^2 & y_8^2 \end{array} \right)^T = yy.
\end{aligned} \tag{1.6}$$

Notice that this formulation avoids the indeterminate form of 0^0. Also, once the vector Y_0 is constructed, the higher powers are successive Hadamard products with the vector y:

$$\begin{aligned}
Y_1 &= yY_0, \\
Y_2 &= yY_1, \\
\vdots & \quad \vdots \\
Y_n &= yY_{n-1}.
\end{aligned} \tag{1.7}$$

The right hand side can be expressed in a similar fashion:

$$\begin{aligned}
P_0 &= pY_0, \\
P_1 &= pY_1, \\
\vdots & \quad \vdots \\
P_n &= pY_n.
\end{aligned} \tag{1.8}$$

This simplifies the process of finding solutions to systems of arbitrary order.

The normal equations are written in an intuitive manner:

$$\begin{gathered}
\mathbf{A}^T\mathbf{A}y = \mathbf{A}^T p \\
\left(\begin{array}{ccc} Y_0 \cdot Y_0 & Y_0 \cdot Y_1 & Y_0 \cdot Y_2 \\ Y_1 \cdot Y_0 & Y_1 \cdot Y_1 & Y_1 \cdot Y_2 \\ Y_2 \cdot Y_0 & Y_2 \cdot Y_1 & Y_2 \cdot Y_2 \end{array} \right) \left(\begin{array}{c} a_0 \\ a_1 \\ a_2 \end{array} \right) = \left(\begin{array}{c} P_0 \\ P_1 \\ P_2 \end{array} \right).
\end{gathered} \tag{1.9}$$

The element in row r and column c is given by

$$\mathbf{A}_{r,c} = Y_{r-1} \cdot Y_{c-1}. \tag{1.10}$$

Using the basic rules for the Hadamard products

$$\begin{aligned}
Y_0 \cdot Y_0 &= n, \\
Y_\xi \cdot Y_\eta &= \sum_{k=1}^{n} y^{\xi+\eta}.
\end{aligned} \tag{1.11}$$

This shows that the matrix $\mathbf{A}^T\mathbf{A}$ must be symmetric because of the fact that

$$\sum_{k=1}^{n} y^{\xi+\eta} = \sum_{k=1}^{n} y^{\eta+\xi} \tag{1.12}$$

The final form of the normal equations for the quadratic polynomial fit is this:

$$\mathbf{A}^T\mathbf{A}y = \mathbf{A}^T p$$

$$\begin{pmatrix} n & \sum_{k=1}^n y & \sum_{k=1}^n y^2 \\ \sum_{k=1}^n y & \sum_{k=1}^n y^2 & \sum_{k=1}^n y^3 \\ \sum_{k=1}^n y^2 & \sum_{k=1}^n y^3 & \sum_{k=1}^n y^4 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} P_0 \\ P_1 \\ P_2 \end{pmatrix}. \tag{1.13}$$

For the data given in the homework assignment we see the following normal equations for the quadratic polynomial fit:

$$\mathbf{A}^T\mathbf{A}y = \mathbf{A}^T p$$

$$\begin{pmatrix} 8 & 15480 & 29958000 \\ 15480 & 29958000 & 57984984000 \\ 29958000 & 57984984000 & 112248125160000 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} 1061377616 \\ 2061122157250 \\ 4003081813438700 \end{pmatrix} \tag{1.14}$$

The quantity

$$\alpha = \mathbf{A}^T\mathbf{A} \tag{1.15}$$

is sometimes referred to as the curvature matrix[1] and will be needed later to estimate the uncertainty, or error, in the amplitudes a_k.

1.2.3 The solution

The inverse matrix can be written as this:

$$\alpha^{-1} = \left(\mathbf{A}^T\mathbf{A}\right)^{-1} = 1\,680\,000 \begin{pmatrix} 14016787590000 & -14488893000 & 3743700 \\ -14488893000 & 14977300 & -3870 \\ 3743700 & -3870 & 1 \end{pmatrix}. \tag{1.16}$$

The amplitudes for each polynomial mode are now computed directly:

$$\begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix} = \left(\mathbf{A}^T\mathbf{A}\right)^{-1}\mathbf{A}^T p = \frac{1}{4200} \begin{pmatrix} 158962246555950 \\ -171107616080 \\ 46114751 \end{pmatrix} \approx \begin{pmatrix} 1.58962 \times 10^{14} \\ -1.71108 \times 10^{11} \\ 4.61148 \times 10^{7} \end{pmatrix} \tag{1.17}$$

[1] *Data Analysis and Error Reduction for the Physical Sciences, 1e*, Philip Bevington, p. 153. McGraw-Hill, 1969.

1.2.4 Extrapolation

The population extrapolated to the year 1990 via the quadratic fit is found as

$$p(1990) = 2.56456 \times 10^8. \tag{1.18}$$

1.2.5 The residual errors

The residual fit errors are given in this vector equation:

$$r = p - a_0 - a_1 y - a_2 y^2. \tag{1.19}$$

The mean of the residuals is then given by this

$$\overline{r} = \frac{r \cdot Y_0}{Y_0 \cdot Y_0}, \tag{1.20}$$

and the mean of the squares of the residuals is

$$\overline{r^2} = \frac{r \cdot r}{Y_0 \cdot Y_0}. \tag{1.21}$$

The standard deviation, σ, is the square root of the variance. For these data the quantities are given by

$$\begin{aligned} \overline{r} &= 0, \\ \sigma &= \frac{\sqrt{\frac{3671149791700435}{6}}}{7} \approx 3.53368 \times 10^6 \end{aligned} \tag{1.22}$$

A zero mean implies that the residuals are balanced: the amount above the solution curve offsets the amount below the solution curve. The standard deviation calculation produces a criterion for rejecting data points. The data here show a uniform distribution of errors and no outlying points.

1.2.6 Plots

The solution curve is plotted against the data points in figure (1.1). The residual errors are easier to see when plotted on a separate curve in figure (1.2).

1.3 Quartic fit

The trial function is now expanded to a fourth order polynomial:

$$p(y) = a_0 + a_1 y + a_2 y^2 + a_3 y^3 + a_4 y^4. \tag{1.23}$$

Since the process was explored in detail in the last section, we will summarize the results in this section.

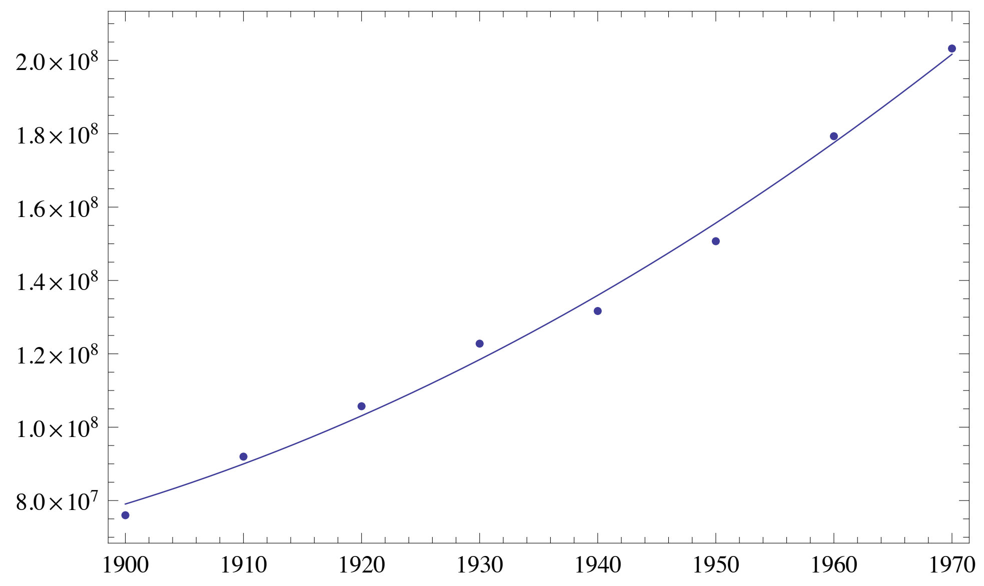


Figure 1.1: The data points and the quadratic solution curve.

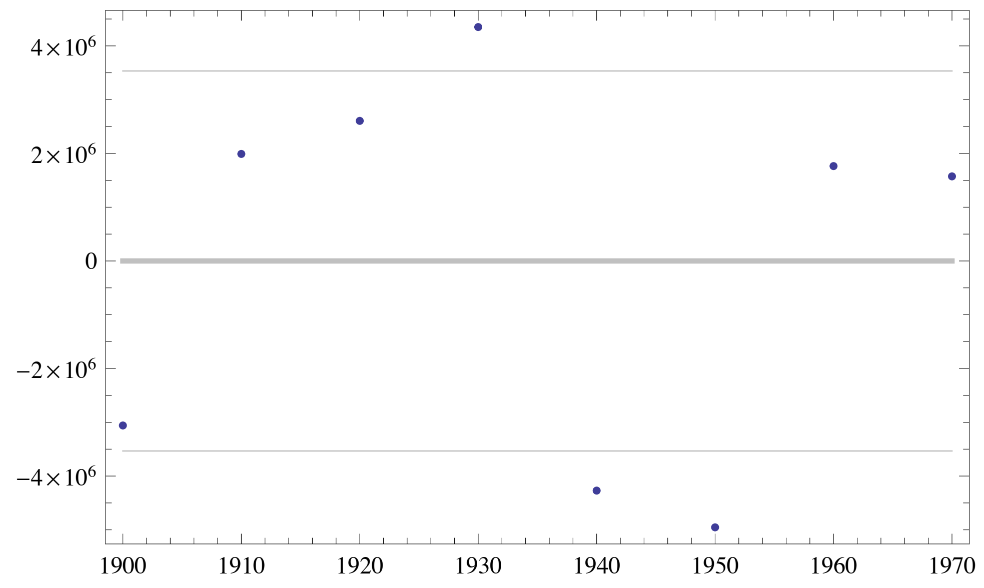


Figure 1.2: The residual errors at each data point for the quadratic solution curve. The thick gray line represents the mean of the residual errors; the thin lines above and below mark $\pm$ one standard deviation.

1.3.1 Intermediate quantities

The matrix products we need are these:

$$\mathbf{A}^T\mathbf{A} = \begin{pmatrix} 8 & 15480 & 29958000 & 57984984000 & 112248125160000 \\ \star & 29958000 & 57984984000 & 112248125160000 & 217321869520800000 \\ \star & \star & 112248125160000 & 217321869520800000 & 420812366627940000000 \\ \star & \star & \star & 420812366627940000000 & 814956097663157040000000 \\ \star & \star & \star & \star & 1578485194416405747600000000 \end{pmatrix},$$

$$\approx \begin{pmatrix} 8. & 15480. & 2.9958 \times 10^{7} & 5.7985 \times 10^{10} & 1.12248 \times 10^{14} \\ \star & 2.9958 \times 10^{7} & 5.7985 \times 10^{10} & 1.12248 \times 10^{14} & 2.17322 \times 10^{17} \\ \star & \star & 1.12248 \times 10^{14} & 2.17322 \times 10^{17} & 4.20812 \times 10^{20} \\ \star & \star & \star & 4.20812 \times 10^{20} & 8.14956 \times 10^{23} \\ \star & \star & \star & \star & 1.57849 \times 10^{27} \end{pmatrix}. \tag{1.24}$$

The data vector transforms in this way:

$$\mathbf{A}^T p = \tag{1.25}$$

$$\left(\mathbf{A}^T\mathbf{A}\right)^{-1} \approx \begin{pmatrix} 1.08502 \times 10^{15} & -2.24328 \times 10^{12} & 1.73914 \times 10^{9} & -599209. & 77.4156 \\ \star & 4.63799 \times 10^{9} & -3.5957 \times 10^{6} & 1238.88 & -0.16006 \\ \star & \star & 2787.65 & -0.960476 & 0.000124091 \\ \star & \star & \star & 0.000330931 & -4.27557 \times 10^{-8} \\ \star & \star & \star & \star & 5.52399 \times 10^{-8} \end{pmatrix}. \tag{1.26}$$

The data vector transforms in this way:

$$\mathbf{A}^T p = \begin{pmatrix} 1061377616 \\ 2061122157250 \\ 4003081813438700 \\ 7775742401827297000 \\ 15105867051563527550000 \end{pmatrix} \tag{1.27}$$

1.3.2 The solution

The amplitudes for each polynomial mode are now computed directly:

$$\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \end{pmatrix} = \left(\mathbf{A}^T\mathbf{A}\right)^{-1} \mathbf{A}^T p \approx \begin{pmatrix} -7.03064 \times 10^{13} \\ 1.44336 \times 10^{11} \\ -1.11099 \times 10^{8} \\ 37999.1 \\ -4.87279 \end{pmatrix} \tag{1.28}$$

1.3.3 Extrapolation

The population extrapolated to the year 1990 is found as

$$p(1990) = 2.62575 \times 10^8. \tag{1.29}$$

1.3.4 The residual errors

For the the quartic fit

$$\begin{aligned} \bar{r} &= 0, \\ \sigma &= \frac{\sqrt{\frac{8155535425638383}{33}}}{7} \approx 2.2458 \times 10^6 \end{aligned} \tag{1.30}$$

1.3.5 Plots

The solution curve is plotted against the data points in figure (1.3). The residual errors are easier to see when plotted on a separate curve in figure (1.4).

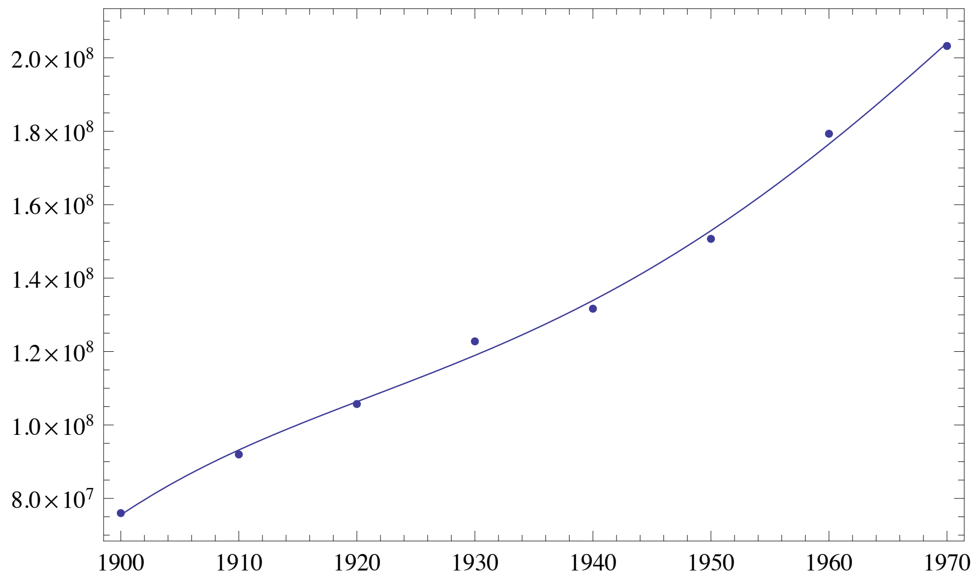


Figure 1.3: The data points and the quadratic solution curve.

1.4 Numeric results

A nice feature of *Mathematica* is that the arbitrary precision computations will switch automatically to IEEE double precision when the first double precision number is encountered. This allows algo-

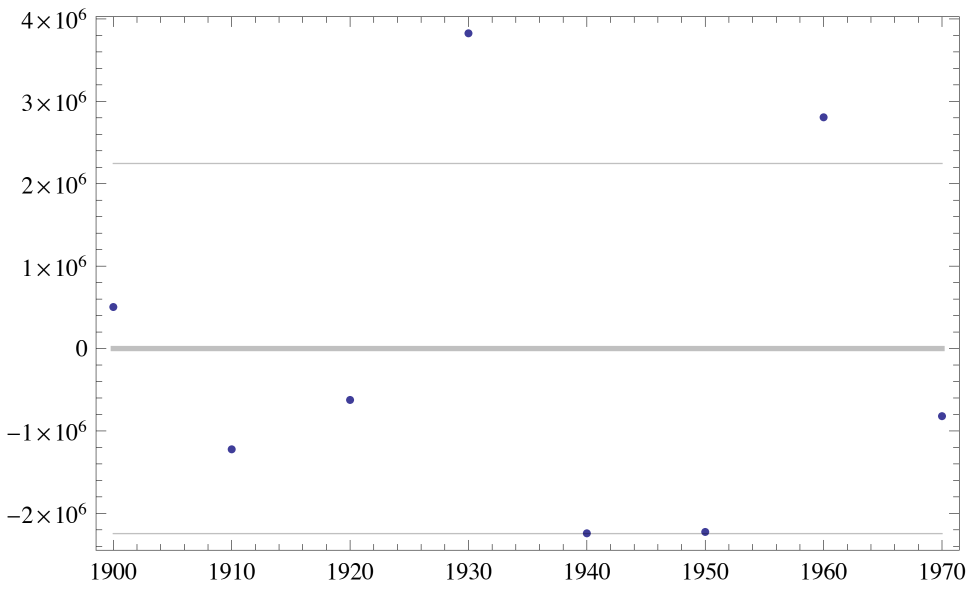


Figure 1.4: The residual errors at each data point for the quartic solution curve. The thick gray line represents the mean of the residual errors; the thin lines above and below mark ± on standard deviation.

rithms to be written and checked exactly and then used to calculate double precision answers.

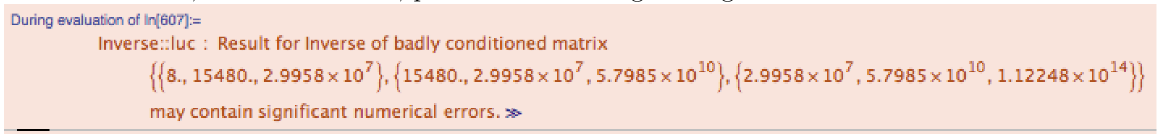
The results for the quadratic fit follow:

$$\boxed{\begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} 3.78482\times10^{10} \\ -4.07399\times10^{7} \\ 10979.7 \end{pmatrix}} \tag{1.31}$$

The projection for 1990 remains the same to six digits:

$$\boxed{p(1990) = 2.56456\times10^{9}.} \tag{1.32}$$

This calculation, while reasonable, produced a warning message:



```
During evaluation of In[607]:=
        Inverse::luc : Result for Inverse of badly conditioned matrix
            {{8., 15480., 2.9958×10^7}, {15480., 2.9958×10^7, 5.7985×10^10}, {2.9958×10^7, 5.7985×10^10, 1.12248×10^14}}
            may contain significant numerical errors. ≫
```

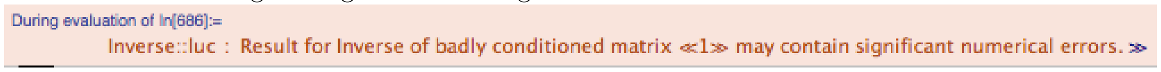
These are the results for the quartic fit:

$$\boxed{\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \end{pmatrix} = \begin{pmatrix} 2.69026 \times 10^{13} \\ -5.66397 \times 10^{10} \\ 4.47077 \times 10^{7} \\ -15682.1 \\ 2.06247 \end{pmatrix}} \tag{1.33}$$

The projection for 1990 becomes meaningless:

$$\boxed{p(1990) = -3.18644 \times 10^{9}.} \tag{1.34}$$

This time the warning message was more urgent:



```
During evaluation of In[686]:=
        Inverse::luc : Result for Inverse of badly conditioned matrix «1» may contain significant numerical errors. »
```

1.5 The condition number

The condition number was computed at each order of fit using the singular values. For any operator norm the condition number for a matrix $\mathbf{A}$is given by

$$\kappa = \frac{\|\mathbf{A}\|}{\|\mathbf{A}^{-1}\|}. \tag{1.35}$$

For the 2−norm, this reduces to the ratio of the largest to the smallest singular values:

$$\kappa = \frac{\|\mathbf{A}\|_2}{\|\mathbf{A}^{-1}\|_2} = \frac{\sigma_1}{\sigma_\rho}. \tag{1.36}$$

where ρ is the matrix rank.

Table (1.37) shows the rapid growth of the condition number showing that the problem is quickly becoming ill-conditioned as the degree of fit increases. These data are plotted in figure (5.1)

degree of fit	condition number
0	1.
1	163434.
2	3.06027×10^{10}
3	6.09688×10^{15}
4	1.3087×10^{21}
5	3.12627×10^{26}
6	8.82999×10^{31}
7	3.4116×10^{37}
8	3.41876×10^{37}

(1.37)

2 Polynomial least squares: QR factorization

1. The following Census Bureau figures give the population of the United States by year.

Year	Population
1900	75,994,575
1910	91,972,266
1920	105,710,620
1930	122,775,046
1940	131,669,275
1950	150,697,361
1960	179,323,175
1970	203,235,298

   Fit these data in a least squares sense by second and fourth degree polynomials, use the normal equations approach. (You may use matlab, lapack or clapack routines to solve the equations.) What is the 'predicted' population by 1990? Approximately what is the condition number of the system you solved? How did you do the approximation?

2. Repeat problem 1 using QR factorization. You may use matlab, lapack or clapack routines. Extra credit: write your own QR factorization.

2.1 QR factorization

The purpose of the factorization is to express any matrix $\mathbf{A} \in \mathbb{C}_{\rho}^{m\times n}$ as the product of a unitary $m \times \rho$ matrix $\mathbf{A}$ and an $\rho \times m$"upper-triangular" matrix $\mathbf{R}$. The quotes on the phrase upper-triangular remind us that the $\mathbf{R}$ matrix is not square in the general case.

The factorization is given by this

$$\mathbf{A} = \mathbf{QR}. \tag{2.1}$$

The target linear system is then reduced to a back-substitution problem:

$$\begin{aligned} \mathbf{A}x &= b \\ (\mathbf{QR})\,x &= b \\ \mathbf{R}x &= \mathbf{Q}^{*}b. \end{aligned} \tag{2.2}$$

The major advantage is that we are resolving $\mathbf{A}$ into a matrix with orthogonal columns which can greatly improve the quality of the solution. The following sections first show the validation computation done with the *Mathematica* routines **QRDecomposition** and **LinearSolve**.

2.1.1 Quadratic fit

Here is the decomposition for the quadratic fit:

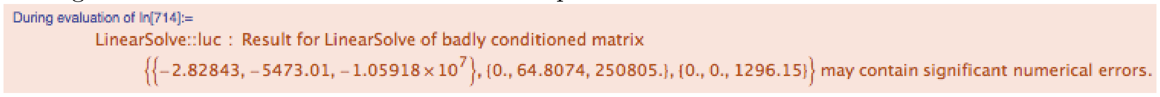
$$\mathbf{Q} = \begin{pmatrix} -0.353553 & -0.540062 & 0.540062 \\ -0.353553 & -0.385758 & 0.0771517 \\ -0.353553 & -0.231455 & -0.231455 \\ -0.353553 & -0.0771517 & -0.385758 \\ -0.353553 & 0.0771517 & -0.385758 \\ -0.353553 & 0.231455 & -0.231455 \\ -0.353553 & 0.385758 & 0.0771517 \\ -0.353553 & 0.540062 & 0.540062 \end{pmatrix}. \tag{2.3}$$

$$\mathbf{R} = \begin{pmatrix} -2.82843 & -5473.01 & -1.05918 \times 10^7 \\ 0. & 64.8074 & 250805. \\ 0. & 0. & 1296.15 \end{pmatrix}. \tag{2.4}$$

Then the back substitution reveals the amplitudes:

$$\boxed{\begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \approx \begin{pmatrix} 3.78482 \times 10^{10} \\ -4.07399 \times 10^7 \\ 10979.7 \end{pmatrix}.} \tag{2.5}$$

During the back-substitution *Mathematica* complained:



```
During evaluation of In[714]:=
        LinearSolve::luc : Result for LinearSolve of badly conditioned matrix
              {{-2.82843, -5473.01, -1.05918×10^7}, {0., 64.8074, 250805.}, {0., 0., 1296.15}} may contain significant numerical errors.
```

The extrapolation is the same

$$\boxed{p(1990) = 2.56456 \times 10^8.} \tag{2.6}$$

2.1.2 Quartic fit

Here is the decomposition for the quartic fit:

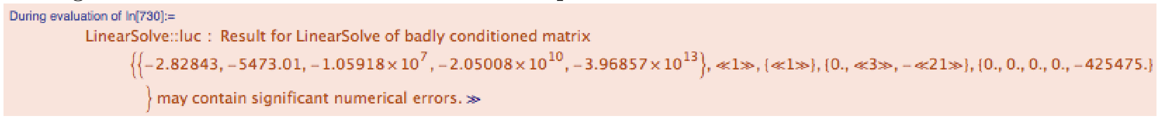
$$\mathbf{Q} = \begin{pmatrix} -0.353553 & -0.540062 & 0.540062 & 0.43082 & -0.282038 \\ -0.353553 & -0.385758 & 0.0771517 & -0.307729 & 0.523785 \\ -0.353553 & -0.231455 & -0.231455 & -0.43082 & 0.120873 \\ -0.353553 & -0.0771517 & -0.385758 & -0.184637 & -0.36262 \\ -0.353553 & 0.0771517 & -0.385758 & 0.184637 & -0.36262 \\ -0.353553 & 0.231455 & -0.231455 & 0.43082 & 0.120873 \\ -0.353553 & 0.385758 & 0.0771517 & 0.307729 & 0.523785 \\ -0.353553 & 0.540062 & 0.540062 & -0.43082 & -0.282038 \end{pmatrix}. \tag{2.7}$$

$$\mathbf{R} = \begin{pmatrix} -2.82843 & -5473.01 & -1.05918\times 10^{7} & -2.05008\times 10^{10} & -3.96857\times 10^{13} \\ 0. & 64.8074 & 250805. & 7.2802\times 10^{8} & 1.8786\times 10^{12} \\ 0. & 0. & 1296.15 & 7.52414\times 10^{6} & 2.91201\times 10^{10} \\ 0. & 0. & 0. & -24372.1 & -1.8864\times 10^{8} \\ 0. & 0. & 0. & 0. & -425475. \end{pmatrix}. \tag{2.8}$$

Then using the back substitution routine we can solve for the amplitudes:

$$\begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{pmatrix} \approx \begin{pmatrix} -7.03064\times 10^{13} \\ 1.44336\times 10^{11} \\ -1.11099\times 10^{8} \\ 37999.1 \\ -4.87279 \end{pmatrix} \tag{2.9}$$

During the back-substitution *Mathematica* complained:



```
During evaluation of In[730]:=
        LinearSolve::luc :  Result for LinearSolve of badly conditioned matrix
              {{-2.82843, -5473.01, -1.05918×10^7, -2.05008×10^10, -3.96857×10^13}, «1», {«1»}, {0., «3», -«21»}, {0., 0., 0., 0., -425475.}
               } may contain significant numerical errors. »
```

The extrapolation is much improved; in fact it is quite reasonable:

$$p(1990) = 2.62575 \times 10^{8}. \tag{2.10}$$

2.2 Extra credit: QR routine

A QR decomposition was written following the presentation in Trefethen and Bau. It uses the Gram-Schmidt method and is listed on the following page. The subsequent listing is the requisite routine to backsolve the triangular system.

modified gs

```
In[340]:= Clear[mgs];
          mgs[A_?MatrixQ] := Module[{Qt, R, v, nrm},
             (* measure the matrix *)
             {row, col} = Dimensions[A];
             (* template matrices *)
             R = IdentityMatrix[col];
             Qt = (A - A)ᵀ;
             (* load the columns of A into v *)
             v = Aᵀ;

             (* primary loop: sweep through columns *)
             For[j = 1, j ≤ col, j ++,
              nrm = Norm[v[[j]]];
              (* normalize the columns of Q *)
              Qt[[j]] = v[[j]]/nrm;
              (* diagonal entries *)
              R[[j, j]] = nrm;

              (* secondary loop: sweep through subsequent columns *)
              For[k = j + 1, k ≤ col, k ++,
               (* amplitude *)
               R[[j, k]] = Qt[[j]]*.v[[k]];
               (* assemble perpendicular vector *)
               v[[k]] -= Simplify[R[[j, k]] Qt[[j]]]
               (* close secondary loop *)
              ]

              (* close primary loop *)
             ];
             (* pass decomposition back as {Q,R} *)
             Return[FullSimplify[{Qtᵀ, R}]];
            ];
```

```
(* Solve a square upper triangular system by back substitution *)
Clear[backsolve];
backsolve[r_?MatrixQ, b_List] := Module[{n, z, ξ},
   n = Length[r];
   (* last element of z *)
   z = {b[[n]] / r[[n, n]]};

   (* sweep up from penultimate row to top row *)
   Do[
    (* ξ is a surrogate for z *)
    ξ = b[[R]];
    (* sweep across columns *)
    Do[
     ξ -= r[[R, C]] z[[C - R]];
     , {C, R + 1, n}];

    (* z entry completed *)
    ξ /= r[[R, R]];

    (* collect terms *)
    PrependTo[z, ξ]
    , {R, n - 1, 1, -1}];

   Return[z];
  ];
```

A few test cases were concocted and run in arbitrary precision to chase out algorithm and coding errors. The test module and the test runs are on the following pages:

test the QR routine with arbitrary precision

In[416]:=
```
Do[
  A = matrices[[k]];

  Print[lf, A // mf, " = target matrix"];

  {q, r} = mgs[A];

  Print["QR decomposition:"];
  Print[q // mf, " = unitary matrix"];
  Print[r // mf, " = upper triangular matrix"];

  B = FullSimplify[q.r];

  Print[B // mf, " = qr"];
  Print[A == B, " = input matches output?"];

  , {k, nmatrices}];
```

$$\begin{pmatrix} 1 & 6 & 0 \\ 2 & 6 & 4 \\ 5 & 4 & 3 \end{pmatrix} = \text{target matrix}$$

QR decomposition:

$$\begin{pmatrix} \frac{1}{\sqrt{30}} & \frac{71}{\sqrt{8970}} & -\frac{11}{\sqrt{299}} \\ \sqrt{\frac{2}{15}} & 2\sqrt{\frac{26}{345}} & \sqrt{\frac{13}{23}} \\ \sqrt{\frac{5}{6}} & -7\sqrt{\frac{5}{1794}} & -\frac{3}{\sqrt{299}} \end{pmatrix} = \text{unitary matrix}$$

$$\begin{pmatrix} \sqrt{30} & 19\sqrt{\frac{2}{15}} & \frac{23}{\sqrt{30}} \\ 0 & \sqrt{\frac{598}{15}} & \frac{103}{\sqrt{8970}} \\ 0 & 0 & \frac{43}{\sqrt{299}} \end{pmatrix} = \text{upper triangular matrix}$$

$$\begin{pmatrix} 1 & 6 & 0 \\ 2 & 6 & 4 \\ 5 & 4 & 3 \end{pmatrix} = \text{qr}$$

True = input matches output?

$$\begin{pmatrix} 0 & 1 & 3 \\ 5 & 3 & 0 \\ 3 & 2 & 3 \end{pmatrix}$$ = target matrix

QR decomposition:

$$\begin{pmatrix} 0 & \sqrt{\frac{34}{35}} & -\frac{1}{\sqrt{35}} \\ \frac{5}{\sqrt{34}} & -\frac{3}{\sqrt{1190}} & -\frac{3}{\sqrt{35}} \\ \frac{3}{\sqrt{34}} & \sqrt{\frac{5}{238}} & \sqrt{\frac{5}{7}} \end{pmatrix}$$ = unitary matrix

$$\begin{pmatrix} \sqrt{34} & \frac{21}{\sqrt{34}} & \frac{9}{\sqrt{34}} \\ 0 & \sqrt{\frac{35}{34}} & \frac{117}{\sqrt{1190}} \\ 0 & 0 & \frac{12}{\sqrt{35}} \end{pmatrix}$$ = upper triangular matrix

$$\begin{pmatrix} 0 & 1 & 3 \\ 5 & 3 & 0 \\ 3 & 2 & 3 \end{pmatrix}$$ = qr

True = input matches output?

$$\begin{pmatrix} 0 & 1 & 3 \\ 5 & 3 & 0 \\ 3 & 2 & 3 \end{pmatrix} = \text{target matrix}$$

QR decomposition:

$$\begin{pmatrix} 0 & \sqrt{\frac{34}{35}} & -\frac{1}{\sqrt{35}} \\ \frac{5}{\sqrt{34}} & -\frac{3}{\sqrt{1190}} & -\frac{3}{\sqrt{35}} \\ \frac{3}{\sqrt{34}} & \sqrt{\frac{5}{238}} & \sqrt{\frac{5}{7}} \end{pmatrix} = \text{unitary matrix}$$

$$\begin{pmatrix} \sqrt{34} & \frac{21}{\sqrt{34}} & \frac{9}{\sqrt{34}} \\ 0 & \sqrt{\frac{35}{34}} & \frac{117}{\sqrt{1190}} \\ 0 & 0 & \frac{12}{\sqrt{35}} \end{pmatrix} = \text{upper triangular matrix}$$

$$\begin{pmatrix} 0 & 1 & 3 \\ 5 & 3 & 0 \\ 3 & 2 & 3 \end{pmatrix} = \text{qr}$$

True = input matches output?

Finally, these same examples were run using double precision arithmetic. The results follow.

- **test the QR routine with double precision**

In[418]:=
```
Do[
  A = matrices[[k]] // N;

  Print[lf, A // mf, " = target matrix"];

  {q, r} = mgs[A];

  Print["QR decomposition:"];
  Print[q // mf, " = unitary matrix"];
  Print[r // mf, " = upper triangular matrix"];

  B = q.r;

  Print[B // mf, " = qr"];
  Print[A - B // mf, " = input - output"];

  , {k, nmatrices}];
```

$$\begin{pmatrix} 1. & 6. & 0. \\ 2. & 6. & 4. \\ 5. & 4. & 3. \end{pmatrix} = \text{target matrix}$$

QR decomposition:

$$\begin{pmatrix} 0.182574 & 0.749656 & -0.636146 \\ 0.365148 & 0.549044 & 0.751809 \\ 0.912871 & -0.369549 & -0.173494 \end{pmatrix} = \text{unitary matrix}$$

$$\begin{pmatrix} 5.47723 & 6.93782 & 4.19921 \\ 0 & 6.31401 & 1.08753 \\ 0 & 0 & 2.48675 \end{pmatrix} = \text{upper triangular matrix}$$

$$\begin{pmatrix} 1. & 6. & 0. \\ 2. & 6. & 4. \\ 5. & 4. & 3. \end{pmatrix} = \text{qr}$$

$$\begin{pmatrix} 1.11022\times 10^{-16} & 0. & 0. \\ 2.22045\times 10^{-16} & 0. & 0. \\ 8.88178\times 10^{-16} & 0. & 0. \end{pmatrix} = \text{input - output}$$

$$\begin{pmatrix} 0. & 1. & 3. \\ 5. & 3. & 0. \\ 3. & 2. & 3. \end{pmatrix} = \text{target matrix}$$

$$\begin{pmatrix} 0. & 1. & 3. \\ 5. & 3. & 0. \\ 3. & 2. & 3. \end{pmatrix} = \text{target matrix}$$

QR decomposition:

$$\begin{pmatrix} 0. & 0.985611 & -0.169031 \\ 0.857493 & -0.0869657 & -0.507093 \\ 0.514496 & 0.144943 & 0.845154 \end{pmatrix} = \text{unitary matrix}$$

$$\begin{pmatrix} 5.83095 & 3.60147 & 1.54349 \\ 0 & 1.0146 & 3.39166 \\ 0 & 0 & 2.02837 \end{pmatrix} = \text{upper triangular matrix}$$

$$\begin{pmatrix} 0. & 1. & 3. \\ 5. & 3. & 0. \\ 3. & 2. & 3. \end{pmatrix} = \text{qr}$$

$$\begin{pmatrix} 0. & 0. & 0. \\ 0. & 0. & 0. \\ 0. & 0. & 0. \end{pmatrix} = \text{input - output}$$

$\begin{pmatrix} 0. & 1. & 3. \\ 5. & 3. & 0. \\ 3. & 2. & 3. \end{pmatrix}$ = target matrix

QR decomposition:

$\begin{pmatrix} 0. & 0.985611 & -0.169031 \\ 0.857493 & -0.0869657 & -0.507093 \\ 0.514496 & 0.144943 & 0.845154 \end{pmatrix}$ = unitary matrix

$\begin{pmatrix} 5.83095 & 3.60147 & 1.54349 \\ 0 & 1.0146 & 3.39166 \\ 0 & 0 & 2.02837 \end{pmatrix}$ = upper triangular matrix

$\begin{pmatrix} 0. & 1. & 3. \\ 5. & 3. & 0. \\ 3. & 2. & 3. \end{pmatrix}$ = qr

$\begin{pmatrix} 0. & 0. & 0. \\ 0. & 0. & 0. \\ 0. & 0. & 0. \end{pmatrix}$ = input - output

2.3 Solution using the new QR decomposition

Now that we have a baseline for accuracy and speed, we can evaluate the performance of the new routines. The results are in full agreement with the *Mathematica* results. The routine is about 20 times slower and it does not provide warning messages.

The results are captured on the following two pages. We do not see full agreement in double precision.

	Mathematica: QRDecomposition, LinearSolve	personal: mgs, backsolve
a_0	37,848,153,941.80905	37,848,153,940.91442
a_1	-40,739,908.59038992	-40,739,908.58946481
a_2	10,979.702619025416	10,979.702618786298
$r \cdot r$	87,408,328,373,882.86	87,408,328,373,815.84
1990	256,456,188.5356598	256,456,188.5350647

Table 2.1: Quadratic fit: a comparison of the *Mathematica* intrinsic routines **QRDecomposition** and **LinearSolve** to the composed routines **mgs** and **backsolve**. The agreement is reasonable, but not full double precision.

	Mathematica: QRDecomposition, LinearSolve	personal: mgs, backsolve
a_0	-70,306,397,416,388.445	-70,305,921,369,758.07
a_1	144,336,423,975.65152	144,335,439,724.154
a_2	-111,098,560.38599531	-111,097,797.30755712
a_3	37,999.09197753073	37,998.82905669359
a_4	-4.872791148562301	-4.872757179203651
$r \cdot r$	35,305,348,015,200.4	35,305,348,084,446,550.0
1990	262,575,002.546875	262,575,188.40625

Table 2.2: Quartic fit: a comparison of the *Mathematica* intrinsic routines **QRDecomposition** and **LinearSolve** to the composed routines **mgs** and **backsolve**. The agreement here is not full single precision.

quadratic fit, QR decomposition using new routines

▪ system

```
In[147]:= A = Table[#^k, {k, 0, dmax}] & /@ year // N;
          % // mf
```

Out[148]//MatrixForm=

$$\begin{pmatrix} 1. & 1900. & 3.61\times10^{6} \\ 1. & 1910. & 3.6481\times10^{6} \\ 1. & 1920. & 3.6864\times10^{6} \\ 1. & 1930. & 3.7249\times10^{6} \\ 1. & 1940. & 3.7636\times10^{6} \\ 1. & 1950. & 3.8025\times10^{6} \\ 1. & 1960. & 3.8416\times10^{6} \\ 1. & 1970. & 3.8809\times10^{6} \end{pmatrix}$$

▪ QR decomposition

```
In[173]:= Clear[qt, r];
          {qt, r} = mgs[A];
          qt.r // mf
```

Out[175]//MatrixForm=

$$\begin{pmatrix} 1. & 1900. & 3.61\times10^{6} \\ 1. & 1910. & 3.6481\times10^{6} \\ 1. & 1920. & 3.6864\times10^{6} \\ 1. & 1930. & 3.7249\times10^{6} \\ 1. & 1940. & 3.7636\times10^{6} \\ 1. & 1950. & 3.8025\times10^{6} \\ 1. & 1960. & 3.8416\times10^{6} \\ 1. & 1970. & 3.8809\times10^{6} \end{pmatrix}$$

▪ amplitudes

```
In[152]:= c = backsolve[r, qt^T.census];
          % // mf
```

Out[153]//MatrixForm=

$$\begin{pmatrix} 3.78482\times10^{10} \\ -4.07399\times10^{7} \\ 10\,979.7 \end{pmatrix}$$

▪ extrapolation

```
In[176]:= Clear[fcn];
          fcn[t_] := c.Table[t^k, {k, 0, dmax}]

          extrap = fcn[1990];
          FortranForm[extrap]
          exact1990 - extrap
```

```
Out[180]= 0.000649601

          2.564561885350647e8
```

quartic fit, QR decomposition using new routines

- **system**

```
In[229]:= A = Table[#^k, {k, 0, dmax}] & /@ year // N;
          % // mf
```

Out[230]//MatrixForm=

$$\begin{pmatrix} 1. & 1900. & 3.61\times10^{6} & 6.859\times10^{9} & 1.30321\times10^{13} \\ 1. & 1910. & 3.6481\times10^{6} & 6.96787\times10^{9} & 1.33086\times10^{13} \\ 1. & 1920. & 3.6864\times10^{6} & 7.07789\times10^{9} & 1.35895\times10^{13} \\ 1. & 1930. & 3.7249\times10^{6} & 7.18906\times10^{9} & 1.38749\times10^{13} \\ 1. & 1940. & 3.7636\times10^{6} & 7.30138\times10^{9} & 1.41647\times10^{13} \\ 1. & 1950. & 3.8025\times10^{6} & 7.41488\times10^{9} & 1.4459\times10^{13} \\ 1. & 1960. & 3.8416\times10^{6} & 7.52954\times10^{9} & 1.47579\times10^{13} \\ 1. & 1970. & 3.8809\times10^{6} & 7.64537\times10^{9} & 1.50614\times10^{13} \end{pmatrix}$$

- **QR decomposition**

```
In[231]:= Clear[qt, r];
          {qt, r} = mgs[A];
          qt.r // mf
```

Out[233]//MatrixForm=

$$\begin{pmatrix} 1. & 1900. & 3.61\times10^{6} & 6.859\times10^{9} & 1.30321\times10^{13} \\ 1. & 1910. & 3.6481\times10^{6} & 6.96787\times10^{9} & 1.33086\times10^{13} \\ 1. & 1920. & 3.6864\times10^{6} & 7.07789\times10^{9} & 1.35895\times10^{13} \\ 1. & 1930. & 3.7249\times10^{6} & 7.18906\times10^{9} & 1.38749\times10^{13} \\ 1. & 1940. & 3.7636\times10^{6} & 7.30138\times10^{9} & 1.41647\times10^{13} \\ 1. & 1950. & 3.8025\times10^{6} & 7.41488\times10^{9} & 1.4459\times10^{13} \\ 1. & 1960. & 3.8416\times10^{6} & 7.52954\times10^{9} & 1.47579\times10^{13} \\ 1. & 1970. & 3.8809\times10^{6} & 7.64537\times10^{9} & 1.50614\times10^{13} \end{pmatrix}$$

- **amplitudes**

```
In[234]:= c = backsolve[r, qt^T.census];
          % // mf
```

Out[235]//MatrixForm=

$$\begin{pmatrix} -7.03059\times10^{13} \\ 1.44335\times10^{11} \\ -1.11098\times10^{8} \\ 37\,998.8 \\ -4.87276 \end{pmatrix}$$

- **extrapolation**

```
In[236]:= Clear[fcn];
          fcn[t_] := c.Table[t^k, {k, 0, dmax}]

          extrap = fcn[1990];
          FortranForm[extrap]
          exact1990 - extrap

Out[240]= -186.501

          2.6257518840625e8
```

3 Linear plus exponential fit functions

3. Fit the data by a function of the form

$$y(t) = c_1 + c_2(t - 1900) + c_3 \exp(d(t - 1900))$$

This function involves four parameters, c_1, c_2, c_3 and d. The problem is linear in c_1, c_2, c_3, but nonlinear in d. Let y be the 8-vector of population data, $\mathbf{c}$ be the unknown 3-vector of coefficients. Let $A(d)$ be the 8×3 coefficient matrix of $\mathbf{c}$ obtained by plugging in the 8 years of data from the table. Minimize the two norm of $\|A(d)\mathbf{c} - \mathbf{y}\|$ with respect to $\mathbf{c}$ and d. You should use a combination of QR routines and a routine to compute the minimum of a function of one variable. Find the values of $\mathbf{c}$ and d that give the best fit. Predict the 1980 and 1990 population.

3.1 Mathematica solution

Start by finding the solution using the full Mathematica toolset. This will provide a fiducial useful for debugging the handcrafted modules.

Define a shifted year variable t

$$t = year - 1900. \tag{3.1}$$

The linear system dictated in the problem statement is this:

$$\begin{array}{cccc} \mathbf{A}(d) & c & = & y \\ \begin{pmatrix} 1 & t_1 & e^{dt_1} \\ 1 & t_2 & e^{dt_2} \\ 1 & t_3 & e^{dt_3} \\ 1 & t_4 & e^{dt_4} \\ 1 & t_5 & e^{dt_5} \\ 1 & t_6 & e^{dt_6} \\ 1 & t_7 & e^{dt_7} \\ 1 & t_8 & e^{dt_8} \end{pmatrix} & \begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix} & = & \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \\ y_8 \end{pmatrix} \end{array} \tag{3.2}$$

This function was sampled for different values of the parameter d and the results are shown in figure (3.1). The first plot found a local minimum near the origin. The second plot zooms in on this minimum.

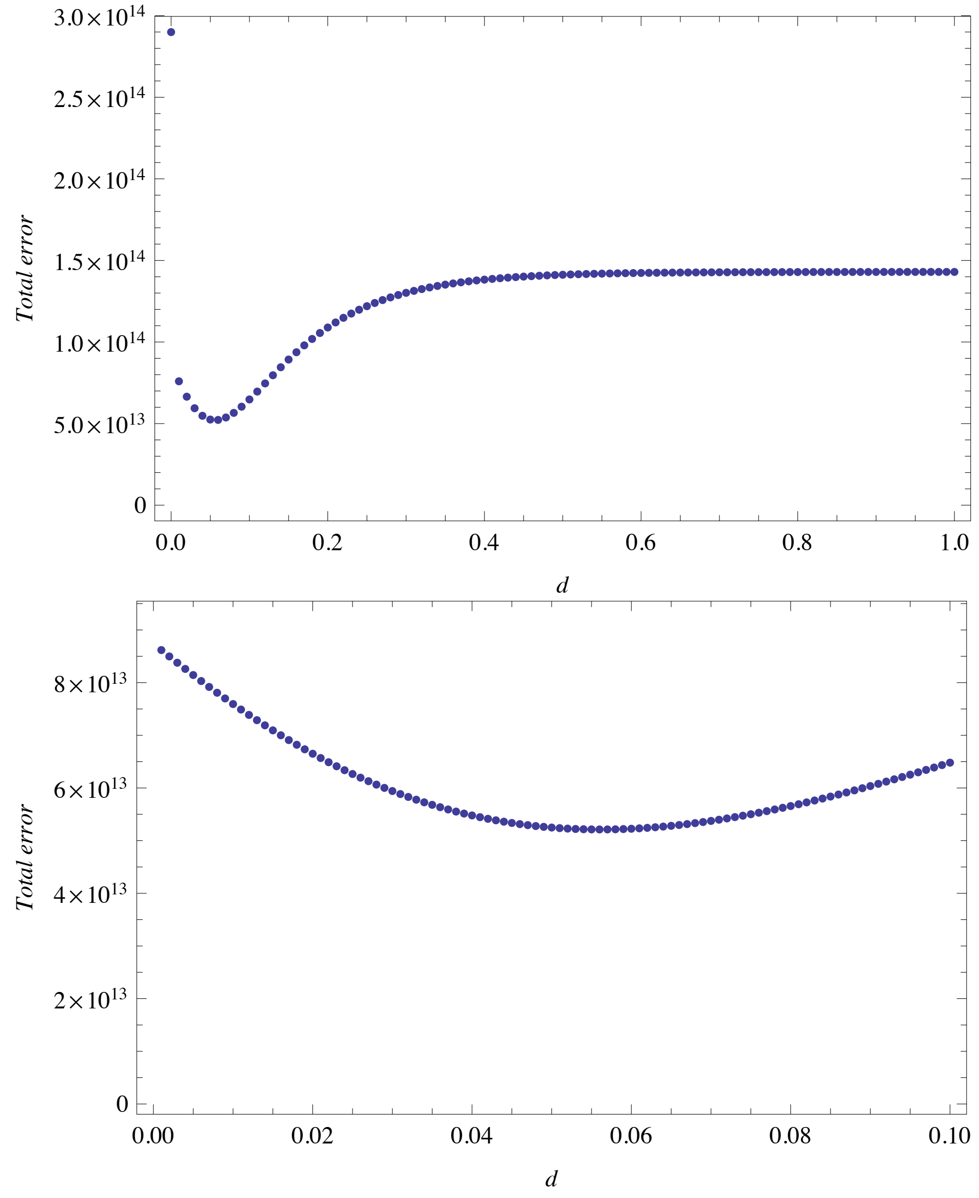
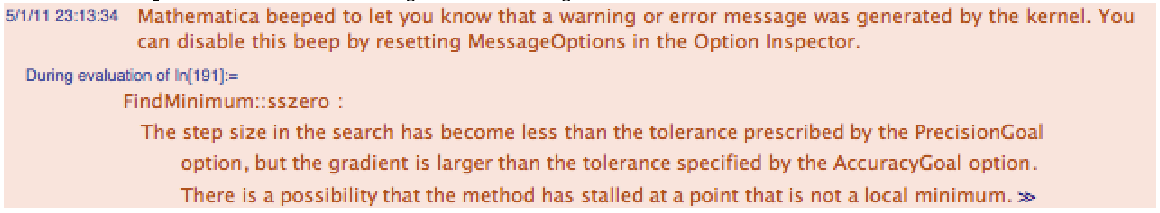


Figure 3.1: A look at the merit function in equation (3.2). The upper curve has a larger domain and shows a unique minimum near the origin. The lower curve zooms in on the minimum.

Next a *Mathematica* module was written to find this minimum and is shown on the following page. This module used the pseudoinverse solution because it is the quickest option for producing the module.

The module produced the following error message:



```
5/1/11 23:13:34  Mathematica beeped to let you know that a warning or error message was generated by the kernel. You
                 can disable this beep by resetting MessageOptions in the Option Inspector.

  During evaluation of In[191]:=
              FindMinimum::sszero :
                The step size in the search has become less than the tolerance prescribed by the PrecisionGoal
                    option, but the gradient is larger than the tolerance specified by the AccuracyGoal option.
                    There is a possibility that the method has stalled at a point that is not a local minimum. >>
```

The last line warns that the solution point may not be a minimum. However, because of the previous plot we see the solution point is the minimum.

The next plot, figure (3.2), shows the solution curve extrapolated out to 1990 against the input data points. We see linear growth at first giving way to exponential growth.

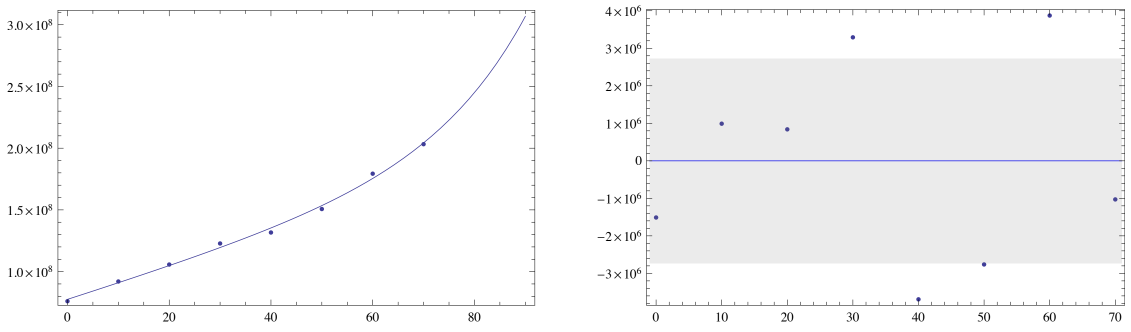


Figure 3.2: The solution curve vs. the data points on the left; residual errors on the right. Near 1990 the solution is a linear function; around 1950 exponential behavior begins to dominate.

The solution using the pseudoinverse and minimization routines in Mathematica is this:

$$\begin{aligned} d &= 0.05613600352531975, \\ a_0 &= 76778509.32757539, \\ a_1 &= 1292986.179127738, \\ a_2 &= 726702.2358835668, \\ r.r &= 5.2122 \times 10^{13}. \end{aligned} \tag{3.3}$$

The extrapolations are then

$$\begin{aligned} 1980 &: 88,261,055.83556682, \\ 1990 &: 89,619,789.81695391. \end{aligned} \tag{3.4}$$

3.2 QR solution

The custom routines **mgs** and **backsolve** were applied to solving the linear plus exponential model. Here we see a valuable lesson from class: a routine that is just a little bit slower can be a significant penalty if it is called thousands or millions of times. In this case, the intrinsic routines needed 0.4 seconds to run. The custom routines took 44 minutes. You don't get many debug cycles when you are waiting 44 minutes per run!

The results for the run are shown on the following page.

Solve nonlinear problem using handcrafted QR and backsolve

```
In[78]:= $tick;
         Clear[c, e, d];
         (* find the minimum *)
         {e, sub} = FindMinimum[efcnqrmgs[ⅆ], {ⅆ, 0.1}]
         tiempo["find minimum qr"];

Out[80]= {5.2122 × 10^13, {ⅆ → 0.056136}}

         CPU time: find minimum qr: 2659.11 sec; 44.3185 min
         elapsed time: 2655.954155 sec; 44.2659 min

In[91]:= (* harvest the solution *)
         d = ⅆ /. sub;
         Print["minimum is at d = ", FortranForm[d]];
         Print["minimum error value = ", e];
         Print["amplitudes = ", a /. sub];
         (* assemble solution function *)
         soln[τ_] = (a /. sub).{1, τ, ⅇ^(d τ)};
         Print["census projected for the year 1980 = ", FortranForm[soln[8]]];
         Print["census projected for the year 1990 = ", FortranForm[soln[9]]];

         minimum is at d = 0.05613600619227794

         minimum error value = 5.2122 × 10^13

         amplitudes = {7.67785 × 10^7, 1.29299 × 10^6, 726 702.}

         census projected for the year 1980 = 8.826105583127841e7

         census projected for the year 1990 = 8.961978982138e7
```

4 Summary and results

4. Make tables summarizing your results and discuss the merits of each approach. What is your best prediction of the 1990 population?

4.1 Demmel's comments

In his book[2] Demmel notes in §3.6 Performance comparison of methods for solving least squares problems, p. 132:

> What is the fastest algorithm for solving dense least squares problems? As discussed in §3.2, solving the normal equations is the fastest, followed by QR and SVD. If $\mathbf{A}$ is is quite well-conditioned, then the normal equations are about as accurate as the other methods, so even thought the normal equations are not numerically stable, they may be used as well. When $\mathbf{A}$ is not well-conditioned but far from rank deficient, we should use QR.

4.2 Comparison of the results

The simplest merit result is the extrapolation. Because we have an exact answer for the extrapolation to 1990, we can quantify the accuracy of the methods. Table (4.1) shows the results for both polynomial fits and four computation strategies.

The obvious best choice for double precision calculations for this problem is the *QR decomposition.* The accuracy is clearly superior to the SVD and the normal equations methods.

4.3 Plots

A series of plots shows the solution curve plotted against the data for each method. The companion plot shows the residual errors. The blue line represents the mean which is ideally zero. The gray box represents the ± one standard deviation for the residuals. One criteria for data rejection is to look at how far the error is from the mean. For these plots, the data are well-behaved.

[2] *Applied Numerical Linear Algebra*, James Demmel, SIAM, 1997

	quadratic fit	quartic fit
exact	2.564561885357143 e8	2.625750019047619 e8
normal	2.564561882009201 e8	-5.5356308086171875 e9
QR	2.564561885356598 e8	2.62575002546875 e8
SVD	2.564562080115585 e8	2.578869665683365 e8

Table 4.1: Results for extrapolation to the year 1980 for each method. The exact answer was computed in arbitrary precision arithmetic. This is allows an accuracy method for the different methods used. The methods are solution by the normal equations, by QR decomposition and by singular values decomposition. The red digits are not significant.

	quadratic fit	quartic fit
exact	16	16
normal	9	0
QR	13	8
SVD	7	2

Table 4.2: Significant digits in the extrapolation to the year 1980 for each method. Clearly the QR decomposition is the best choice. The surprise is that the SVD performance is on par with the normal equations.

	quadratic fit	quartic fit
exact	8.7×10^{13}	3.5×10^{13}
normal	8.7×10^{13}	2.4×10^{20}
QR	8.7×10^{13}	8.4×10^{13}
SVD	8.7×10^{13}	3.5×10^{13}

Table 4.3: Total error for the two fits and the four different methods. These results are surprising. The total error in the quadratic fit was essentially the same all four methods. For the quartic fit, the results were confusing in that the SVD had a lower total error than the QR decomposition despite yielding fewer significant digits in the extrapolation (table (4.2)).

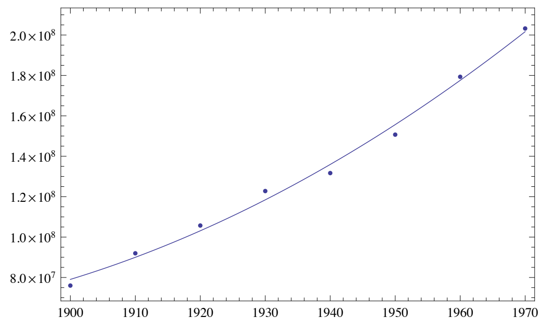
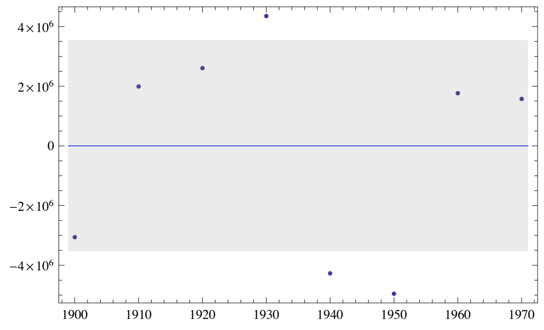
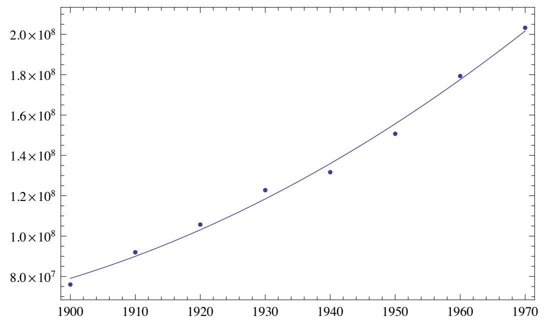
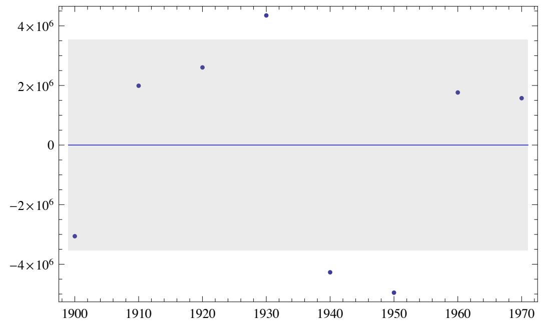
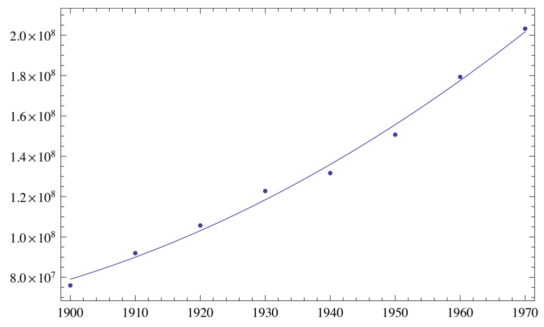
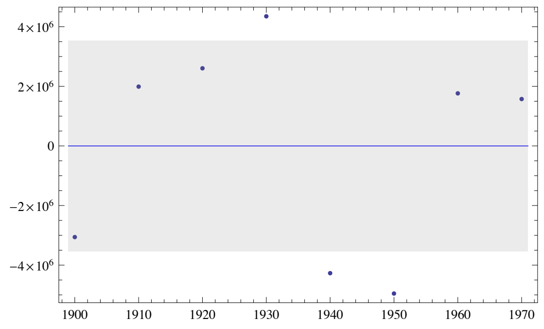
	data vs. fit	residual errors
normal		
QR		
SVD		

Table 4.4: The quadratic fit solution curves and the residual errors.

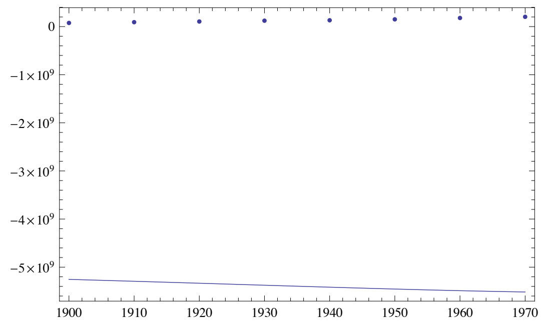
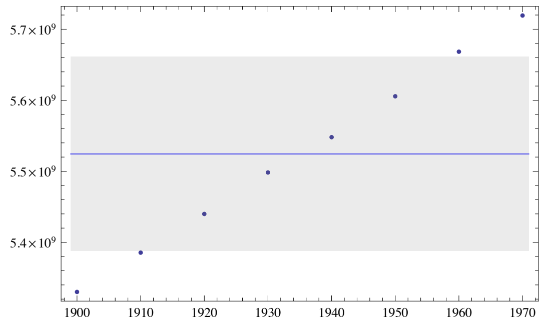
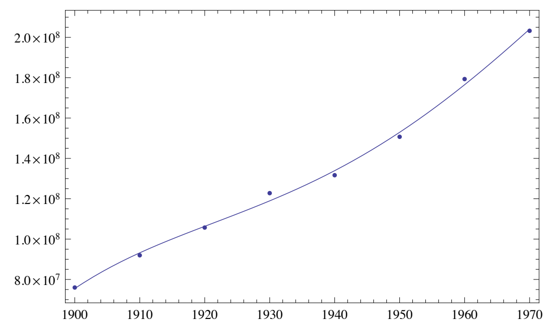
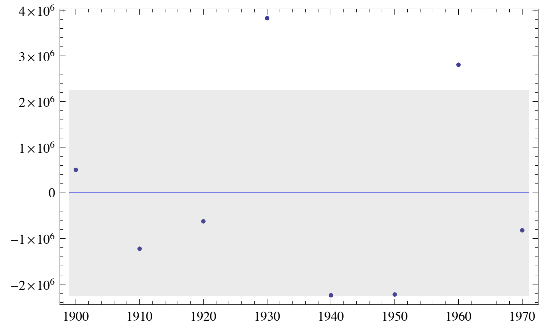
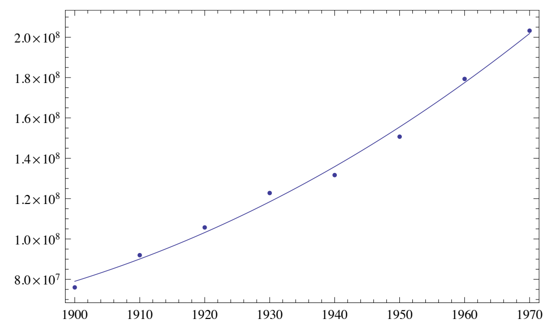
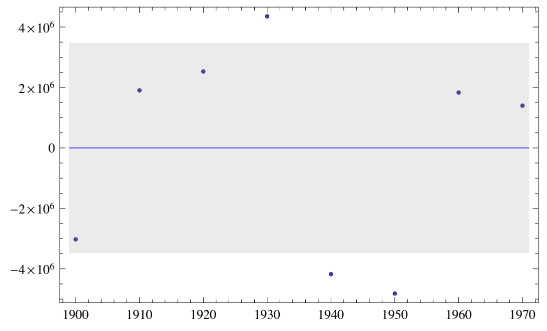
	data vs. fit	residual errors
normal		
QR		
SVD		

Table 4.5: The quartic fit solution curves and the residual errors.

5 Polynomial least squares: singular value decomposition

5. Extra Credit: Repeat 2-3 using matlab, lapack or clapack routines for the SVD in place of QR factorization. What are the singular values of the coefficient matrices?

5.1 The singular value decomposition

The singular value decomposition may be the most powerful decomposition in matrix analysis. It comes at a heavy computational cost of resolving the eigensystem of a matrix product like $\mathbf{A}^T\mathbf{A}$.

However, by resolving the system into orthogonal bases for the domain and codomain the method is robust. The singular values diagnose present the condition number in the $2-$norm and show how "close" target matrix $\mathbf{A}$ is to the nearest matrix of full rank.

Below is a statement of the singular value theorem. This is followed by the application to solving linear systems using the method of least squares.

5.1.1 The singular value decomposition

Given $\mathbf{A} \in \mathbb{C}^{m\times n}_{\rho}$, a complex matrix with m rows and n columns and rank ρ, where $\rho \le \min(m,n)$ then there exists a matrix decomposition of the format

$$\mathbf{A} = \mathbf{Y}\,\Sigma\,\mathbf{X}^{*}. \tag{5.1}$$

1. The unitary matrix $\mathbf{Y}$ represents an orthonormal basis for the *codomain.*
2. The unitary matrix $\mathbf{X}$ represents an orthonormal basis for the *domain.*
3. There are ρ positive *singular values* which are the square roots of the nonzero eigenvalues of the product matrices $\mathbf{A}^{*}\mathbf{A}$ and $\mathbf{A}\,\mathbf{A}^{*}$.
4. The ordered singular values $(\sigma_1 \ge \sigma_2 \ge \cdots \ge \sigma_\rho > 0)$ populate the diagonal $\mathbf{S}$ matrix which is embedded in the Σ matrix.
5. The dimensions of associated matrices are shown here:

$$\mathbf{A}^{m\times n}_{\rho} = \mathbf{Y}^{m\times m}_{m} \left(\begin{array}{c|c} \mathbf{S}^{\rho\times\rho}_{\rho} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} \end{array} \right)^{m\times n}_{\rho} (\mathbf{X}^{*})^{n\times n}_{n}. \tag{5.2}$$

5.1.2 The singular value decomposition and the method of least squares

Consider the most general linear system

$$\mathbf{A}x = b \tag{5.3}$$

The target matrix $\mathbf{A} \in \mathbb{C}_{\rho}^{m\times n}$. The general solution the to the problem is the sum of the particular and homogeneous solutions:

$$x_g = x_p + x_h. \tag{5.4}$$

Using the generalized matrix inverse (or pseudoinverse) $\mathbf{A}^{+}$ the solution to the linear system becomes this

$$\mathbf{A}^{+}\,\mathbf{A}x = \mathbf{A}^{+}\,b \tag{5.5}$$

5.1.3 Using the pseudoinverse

The pseudoinverse is constructed using this recipe:

$$\mathbf{A}^{+} = \mathbf{X}\,\Sigma^{(+)}\,\mathbf{Y}^{*} \tag{5.6}$$

and is the unique matrix which satisfies the four Penrose conditions:

1. $\mathbf{A}^{+}\,\mathbf{A}\,\mathbf{A}^{+} = \mathbf{A}^{+}$
2. $\mathbf{A}\,\mathbf{A}^{+}\,\mathbf{A} = \mathbf{A}$
3. $\left(\mathbf{A}^{+}\,\mathbf{A}\right)^{T} = \mathbf{A}^{+}\,\mathbf{A}$
4. $\left(\mathbf{A}\,\mathbf{A}^{+}\right)^{T} = \mathbf{A}^{+}\,\mathbf{A}$

The pseudoinverse connects to the usual matrix inverse in these cases:

chirality, χ	requirement	equivalence	expression
right	$\rho = m$	$\mathbf{A}^{+} = \mathbf{A}_{\mathrm{R}}^{\text{-1}}$	$\mathbf{A}\mathbf{A}^{+} = \mathbf{I}_m$
left	$\rho = n$	$\mathbf{A}^{+} = \mathbf{A}_{\mathrm{L}}^{\text{-1}}$	$\mathbf{A}^{+}\mathbf{A} = \mathbf{I}_n$
ambichiral	$\rho = m = n$	$\mathbf{A}^{+} = \mathbf{A}^{-1}$	$\mathbf{A}\mathbf{A}^{+} = \mathbf{A}^{+}\mathbf{A} = \mathbf{I}_m$
ampichiral	$\rho < m, \rho < n$	none	$\mathbf{A}\mathbf{A}^{+} \neq \mathbf{I}_m,\ \mathbf{A}^{+}\mathbf{A} \neq \mathbf{I}_n$

Table 5.1: Conditions for when the pseudoinverse is a chiral inverse.

The left-hand side: One of the bonuses we reap from the pseudoinverse is the four fundamental orthogonal projections. Each of these matrices projects onto a specific vector space. The spaces are these:

1. the range $\mathcal{R}\left(\mathbf{A}\right)$,
2. the orthogonal complement to the range, $\mathcal{R}\left(\mathbf{A}\right)^{\perp}$,
3. the null space $\mathcal{N}\left(\mathbf{A}\right)$,
4. the orthogonal complement to the null space, $\mathcal{N}\left(\mathbf{A}\right)^{\perp}$.

Below are these projectors for a matrix $\mathbf{A} \in \mathbb{C}_\rho^{m\times n}$ with pseudoinverse $\mathbf{A}^+$:

$$\begin{array}{rclrcl} \mathbf{P}_{\mathcal{R}(\mathbf{A})} & = & \mathbf{A}\mathbf{A}^+ & \mathbf{P}_{\mathcal{N}\left(\mathbf{A}^{\mathrm{T}}\right)} & = & \mathbf{I}_m - \mathbf{A}\mathbf{A}^+ \\ \mathbf{P}_{\mathcal{R}\left(\mathbf{A}^{\mathrm{T}}\right)} & = & \mathbf{A}^+\mathbf{A} & \mathbf{P}_{\mathcal{N}(\mathbf{A})} & = & \mathbf{I}_n - \mathbf{A}^+\mathbf{A} \end{array} \tag{5.7}$$

Each operator projects vectors onto its range parallel to its nullspace.

The projection relevant here is this

$$\begin{aligned} \mathbf{P}_{\mathcal{R}\left(\mathbf{A}^{\mathrm{T}}\right)} &= \mathbf{A}^+\mathbf{A}, \\ &= \mathbf{X}\,\Sigma^{(+)}\,\Sigma\mathbf{X}^*. \end{aligned} \tag{5.8}$$

The matrix product $\Sigma^{(+)}\,\Sigma$ is a truncated identity which acts like a stencil allowing only the first ρ column vectors of $\mathbf{X}$ into play. These are the vectors which span the image of the codomain.

The right-hand side: The right hand side is the point solution to the least squares problem. We also must account for the homogeneous solution which is composed of null space vectors $\mathcal{N}\left(\mathbf{A}^T\right)$ (shaded columns) from the domain matrix $\mathbf{X}$. The final solution takes this form

$$\boxed{\begin{aligned} x_g &= x_p + x_g \\ &= \mathbf{A}^+\,x + \alpha_1\mathbf{X}_{*,\rho+1} + \alpha_2\mathbf{X}_{*,\rho+2} + \cdots + \alpha_{n-\rho}\mathbf{X}_{*,n} \end{aligned}} \tag{5.9}$$

where α is composed of arbitrary scalars.

5.2 Quadratic fit

For the quadratic fit the target matrix $\mathbf{A} \in \mathbb{R}_3^{8\times 3}$ is given by this:

$$\left(\begin{array}{ccr} 1 & 1900. & 3.61\times 10^6 \\ 1 & 1910. & 3.6481\times 10^6 \\ 1 & 1920. & 3.6864\times 10^6 \\ 1 & 1930. & 3.7249\times 10^6 \\ 1 & 1940. & 3.7636\times 10^6 \\ 1 & 1950. & 3.8025\times 10^6 \\ 1 & 1960. & 3.8416\times 10^6 \\ 1 & 1970. & 3.8809\times 10^6 \end{array}\right) \tag{5.10}$$

5.2.1 Singular value decomposition

The singular value decomposition follows:

$$\mathbf{Y} = \begin{pmatrix} -0.340736 & -0.542669 & -0.545659 & 0.00758321 & 0.0359592 & -0.0379967 & -0.214284 & -0.492904 \\ -0.344332 & -0.393202 & -0.0811783 & 0.0761551 & 0.238024 & 0.370176 & 0.472613 & 0.545334 \\ -0.347947 & -0.24214 & 0.229007 & -0.522739 & -0.512034 & -0.373972 & -0.108553 & 0.284223 \\ -0.351581 & -0.089483 & 0.384897 & 0.792735 & -0.210855 & -0.181223 & -0.118368 & -0.0222906 \\ -0.355233 & 0.0647689 & 0.386492 & -0.219531 & 0.757685 & -0.230612 & -0.184423 & -0.103747 \\ -0.358905 & 0.220616 & 0.233792 & -0.171998 & -0.213082 & 0.764179 & -0.240217 & -0.226267 \\ -0.362596 & 0.378058 & -0.0732033 & -0.0646672 & -0.123157 & -0.196851 & 0.714251 & -0.389851 \\ -0.366305 & 0.537095 & -0.534494 & 0.102462 & 0.0274602 & -0.1137 & -0.321019 & 0.405503 \end{pmatrix}, \tag{5.11}$$

$$\Sigma = \begin{pmatrix} 1.05947 \times 10^7 & 0 & 0 \\ 0 & 64.7746 & 0 \\ 0 & 0 & 0.000346202 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \tag{5.12}$$

$$\mathbf{X} = \begin{pmatrix} -2.66891 \times 10^7 & -0.00103368 & -0.999999 \\ -0.000516579 & -0.999999 & 0.00103368 \\ -1. & 0.000516579 & -2.67087 \times 10^7 \end{pmatrix}. \tag{5.13}$$

The gray shading separates the null space from the range spaces. Here from the domain matrix **X** that the target matrix **A** has full column rank. The **Y** matrix shows that there are five vectors in the null space and that **A** has a column rank deficiency of five.

5.2.2 Pseudoinverse

Finally the pseudoinerse is assembled according to equation (5.6). The transpose is shown due to the width limitations on the page.

$$\left(\mathbf{A}^{+}\right)^{T} = \begin{pmatrix} 1576.13 & -1.62083 & 0.000416667 \\ 234.482 & -0.23631 & 0.0000595238 \\ -661.482 & 0.6875 & -0.000178571 \\ -1111.77 & 1.1506 & -0.000297619 \\ -1116.38 & 1.15298 & -0.000297619 \\ -675.304 & 0.694643 & -0.000178571 \\ 211.446 & -0.224405 & 0.0000595238 \\ 1543.88 & -1.60417 & 0.000416667 \end{pmatrix}. \tag{5.14}$$

As a quick check, we expect the pseudoinverse to also be a left inverse. Do we see an identity matrix?

$$\mathbf{A}^{+}\,\mathbf{A} = \begin{pmatrix} 1. & -4.65661\times10^{-10} & -9.53674\times10^{-7} \\ 3.27294\times10^{-13} & 1. & 0. \\ -1.03379\times10^{-13} & -2.22045\times10^{-16} & 1. \end{pmatrix} \tag{5.15}$$

5.2.3 Solution

The solution vector is the following

$$\begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} 3.78482\times10^{10} \\ -4.07399\times10^{7} \\ 10979.7 \end{pmatrix}. \tag{5.16}$$

The litmus test is the extrapolation:

$$p(1990) = 2.56456\times10^{8} \tag{5.17}$$

The display is in full agreement with answer from arbitrary precision arithmetic.

5.3 Quartic fit

The system matrix is given by this expression

$$\mathbf{A} = \begin{pmatrix} 1 & 1900. & 3.61\times10^{6} & 6.859\times10^{9} & 1.30321\times10^{13} \\ 1 & 1910. & 3.6481\times10^{6} & 6.96787\times10^{9} & 1.33086\times10^{13} \\ 1 & 1920. & 3.6864\times10^{6} & 7.07789\times10^{9} & 1.35895\times10^{13} \\ 1 & 1930. & 3.7249\times10^{6} & 7.18906\times10^{9} & 1.38749\times10^{13} \\ 1 & 1940. & 3.7636\times10^{6} & 7.30138\times10^{9} & 1.41647\times10^{13} \\ 1 & 1950. & 3.8025\times10^{6} & 7.41487\times10^{9} & 1.4459\times10^{13} \\ 1 & 1960. & 3.8416\times10^{6} & 7.52954\times10^{9} & 1.47579\times10^{13} \\ 1 & 1970. & 3.8809\times10^{6} & 7.64537\times10^{9} & 1.50614\times10^{13} \end{pmatrix} \tag{5.18}$$

5.3.1 Singular value decomposition

The singular value decomposition follows:

$$\mathbf{Y} = \begin{pmatrix} -0.328015 & -0.539303 & 0.548353 & 0.436931 & -0.167196 & -0.126854 & -0.224284 & -0.124933 \\ -0.334976 & -0.399375 & 0.0952522 & -0.299108 & 0.22037 & 0.511552 & 0.500932 & 0.261862 \\ -0.342046 & -0.254847 & -0.215504 & -0.429781 & 0.29606 & -0.659008 & -0.192276 & 0.167304 \\ -0.349228 & -0.105648 & -0.379229 & -0.198312 & -0.241835 & 0.185409 & -0.0459014 & -0.767312 \\ -0.356522 & 0.0482972 & -0.391189 & 0.164758 & -0.635196 & 0.0592081 & -0.0830507 & 0.525841 \\ -0.36393 & 0.207062 & -0.246599 & 0.436363 & 0.599323 & 0.290678 & -0.356904 & 0.0489227 \\ -0.371453 & 0.370722 & 0.059375 & 0.313286 & 0.0421002 & -0.386909 & 0.672343 & -0.139276 \\ -0.379092 & 0.539352 & 0.531615 & -0.424123 & -0.113627 & 0.125924 & -0.270858 & 0.0275929 \end{pmatrix}, \tag{5.19}$$

$$\Sigma = \begin{pmatrix} 3.97302\times10^{13} & 0 & 0 & 0 & 0 \\ 0 & 2.42271\times10^{8} & 0 & 0 & 0 \\ 0 & 0 & 1295.38 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0. \end{pmatrix}, \tag{5.20}$$

$$\mathbf{x} = \begin{pmatrix} 6.78480^{-10} & 1.17856^{-10} & -2.66644^{-10} & -4.69222^{-10} & -4.84019^{-10} & -3.05118^{-10} & 7.34649^{-11} & 6.57770^{-10} \\ 6.56321^{-7} & 1.14007^{-7} & -2.57935^{-7} & -4.53897^{-7} & -4.68211^{-7} & -2.95152^{-7} & 7.10655^{-8} & 6.36287^{-7} \\ 0.000423313 & 0.000073532 & -0.000166363 & -0.000292754 & -0.000301986 & -0.000190367 & 0.0000458358 & 0.000410392 \\ -4.35252^{-7} & -7.43440^{-8} & 1.72981^{-7} & 3.02986^{-7} & 3.11892^{-7} & 1.95882^{-7} & -4.88997^{-8} & -4.26350^{-7} \\ 1.11872^{-10} & 1.87884^{-11} & -4.49488^{-11} & -7.83738^{-11} & -8.05104^{-11} & -5.03725^{-11} & 1.30363^{-11} & 1.10722^{-10} \end{pmatrix}. \tag{5.21}$$

The gray shading separates the null space from the range spaces. The singular values provide valuable information. While theoretically the target matrix **A** has full row rank, effectively it is a rank three matrix. One benefit of the SVD is that is diagnoses the rank and tells us that the there are only three indistinguishable basis vectors: the other two are close together. Rounding in double precision eliminated the last two singular values.

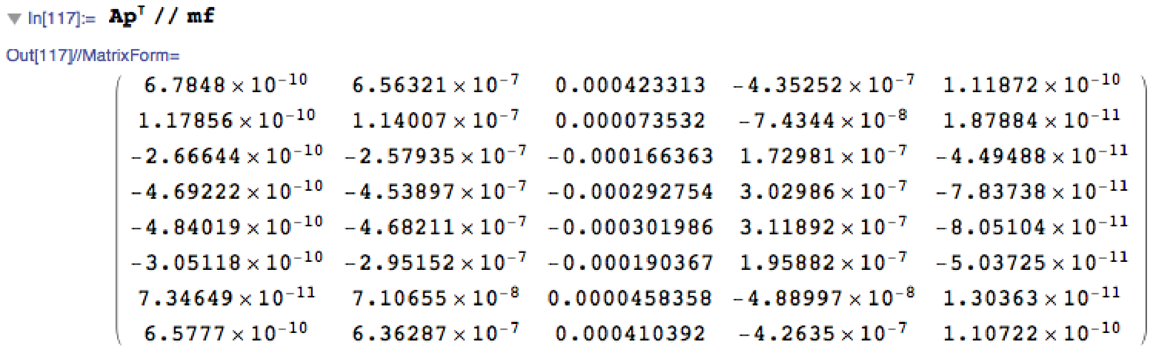
To verify that we have five singular values, *Mathematica* produced the answer symbolically. These symbolic objects were cast into double precision:

$$\begin{aligned} \sigma_1 &= 3.97302 \times 10^{13}, \\ \sigma_2 &= 2.42271 \times 10^{8}, \\ \sigma_3 &= 1295.38, \\ \sigma_4 &= 0.00650868, \\ \sigma_5 &= 3.0358436367713506 \times 10^{-8}, \end{aligned} \tag{5.22}$$

We now see the problem: the singular values span 31 orders of magnitude.

5.3.2 Pseudoinverse

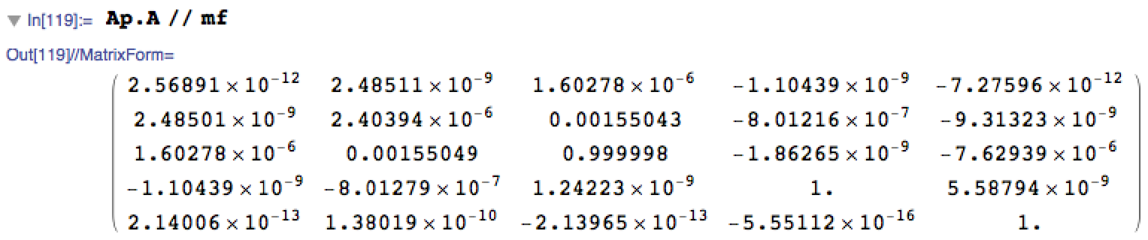
Again the pseudoinerse is assembled according to equation (5.6).



```
In[117]:= ApT // mf

Out[117]//MatrixForm=
 6.7848 × 10^-10    6.56321 × 10^-7    0.000423313    -4.35252 × 10^-7   1.11872 × 10^-10
 1.17856 × 10^-10   1.14007 × 10^-7    0.000073532    -7.4344 × 10^-8    1.87884 × 10^-11
-2.66644 × 10^-10  -2.57935 × 10^-7   -0.000166363    1.72981 × 10^-7   -4.49488 × 10^-11
-4.69222 × 10^-10  -4.53897 × 10^-7   -0.000292754    3.02986 × 10^-7   -7.83738 × 10^-11
-4.84019 × 10^-10  -4.68211 × 10^-7   -0.000301986    3.11892 × 10^-7   -8.05104 × 10^-11
-3.05118 × 10^-10  -2.95152 × 10^-7   -0.000190367    1.95882 × 10^-7   -5.03725 × 10^-11
 7.34649 × 10^-11   7.10655 × 10^-8    0.0000458358  -4.88997 × 10^-8    1.30363 × 10^-11
 6.5777 × 10^-10    6.36287 × 10^-7    0.000410392   -4.2635 × 10^-7    1.10722 × 10^-10
```

As a quick check, we expect the pseudoinverse to also be a left inverse. Do we see an identity matrix? Not to single precision, no.



```
In[119]:= Ap.A // mf
Out[119]//MatrixForm=
 2.56891×10^-12   2.48511×10^-9   1.60278×10^-6   -1.10439×10^-9   -7.27596×10^-12
 2.48501×10^-9    2.40394×10^-6   0.00155043      -8.01216×10^-7   -9.31323×10^-9
 1.60278×10^-6    0.00155049      0.999998        -1.86265×10^-9   -7.62939×10^-6
-1.10439×10^-9   -8.01279×10^-7   1.24223×10^-9    1.              5.58794×10^-9
 2.14006×10^-13   1.38019×10^-10 -2.13965×10^-13  -5.55112×10^-16   1.
```

5.3.3 Solution

The solution vector is the following

$$\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \end{pmatrix} = \begin{pmatrix} 0.0137497 \\ 13.3006 \\ 8578.61 \\ -9.26204 \\ 0.00250447 \end{pmatrix}. \tag{5.23}$$

The litmus test is the extrapolation:

$$p(1990) = 2.57887 \times 10^8 \tag{5.24}$$

The display is in rough agreement with answer from arbitrary precision arithmetic in equation (1.29).

5.4 Linear plus exponential fit

The SVD was applied to solving the linear plus exponential model and the results are shown on the following page. the supporting routines are listed after the results.

Solve nonlinear problem using SVD

```
In[349]:= $tick;
          Clear[c, e, d];
          (* find the minimum *)
          {e, sub} = FindMinimum[efcnsvd[d1], {d1, 0.1}];
          tiempo["find minimum SVD"];

          CPU time: find minimum SVD: 0.396873 sec
          elapsed time: 0.396530 sec

In[353]:= (* harvest the solution *)
          d = d1 /. sub;
          Print["minimum is at d = ", FortranForm[d]];
          Print["minimum error value = ", e];
          Print["amplitudes = ", a /. sub];

          (* assemble solution function *)
          soln[τ_] = (a /. sub).{1, τ, e^(d τ)};
          Print["census projected for the year 1980 = ", FortranForm[soln[8]]];
          Print["census projected for the year 1990 = ", FortranForm[soln[9]]];

          minimum is at d = 0.05613600493561435

          minimum error value = 5.2122 × 10^13

          amplitudes = {7.67785 × 10^7, 1.29299 × 10^6, 726 702.}

          census projected for the year 1980 = 8.82610558332991e7

          census projected for the year 1990 = 8.961978981929444e7
```

supporting routines

- **invert the Σ matrix in the SVD**

```
Clear[Σinvert];
Σinvert[Σ_List] := Module[{r, c, S, λ},
   S = Σ^T;
   {r, c} = Dimensions[S];
   λ = Min[r, c];
   Do[
    If[S[[k, k]] == 0, Break[]];
    S[[k, k]] = 1 / S[[k, k]];
    , {k, λ}];
   Return[S]
  ];
```

minimize the nonlinear error norm using the SVD

```
In[360]:= Clear[efcnsvd];
          efcnsvd[d_] := Module[{e, r, soln, τ},
             (* solve for the amplitudes using the pseudoinverse *)
             (* decomposition *)
             {Y, Σ, X} = SingularValueDecomposition[A[d]];
             (* construct pseudoinverse *)
             Ap = X.Σinvert[Σ].Y^T;
             (* particular solution *)
             a = Ap.census // N;
             (* build the solution function *)
             soln[τ_] = a.{1, τ, e^(d τ)};
             (* compute residual errors at each point *)
             r = census - soln[t];
             (* compute the total error *)
             e = r.r;
             (* return the total error *)
             Return[e];
            ];
```

5.5 Comparison of the results

The simplest merit result is the extrapolation. Because we have an exact answer for the extrapolation to 1990, we can quantify the accuracy of the methods. Table (5.2) shows the results for both polynomial fits and four computation strategies.

The obvious best choice for double precision calculations for this problem is the *singular value*. The accuracy is clearly superior to the QR decomposition.

5.6 Fits to higher orders

As mentioned in the first problem, the formulation lends itself to fits of arbitrary order. A module was written to solve the system using exact arithmetic using the pseudoinverse. This module is shown on the following page.

	1980	1990
exact	88,261,055.8332991	89,619,789.81929444
SVD	88,261,055.8332991	89,619,789.81929444
QR	88,261,055.83127841	89,619,789.82138

Table 5.2: Results for extrapolation to the years 1980 and 1990 for each method. The exact answer was computed in arbitrary precision arithmetic. This is allows an accuracy method for the different methods used.

	linear plus exponential
exact	15
SVD	15
QR	9

Table 5.3: Significant digits in the extrapolation to the years 1980 and 1990 for each method. The SVD routine looks as though it is duplicating the intrinsic *Mathematica* operations. The QR decomposition agrees in a single precision sense.

	linear plus exponential		
exact	$52,122,032,988,875.680 \times 10^{13}$		
SVD	$52,122,032,988,875.766 \times 10^{13}$	QR	$52,122,032,988,875.734 \times 10^{13}$

Table 5.4: Total error for both methods. Despite the differences in the extrapolations, the total errors are remarkably similar.

modules

Solve the linear system Ax = b using the generalized matrix inverse

```
Clear[Axb];
Axb[year_List, census_List, degree_Integer] := Module[{B, a, n, r, x, y, name, ℂ},
   n = Length[year];

   (* tag the output files *)
   name = "degree_" <> pad[degree, 3];

   (* create the polynomial basis functions *)
   B = Join[{1}, Table[x^n, {n, degree}]];

   (* create the system matrix using a pure function *)
   A = Join[{1}, Table[#^n, {n, degree}]] & /@ year;

   (* pseudo inverse *)
   Ap = PseudoInverse[A];

   (* least squares solution *)
   c = Ap.census;

   (* export the amplitudes *)
   Export[dirData <> name <> "_amplitudes.csv", c];

   (* polynomial solution *)
   y[x_] = c.B;

   (* extrapolation *)
   ext = y[x] /. x → 1990;

   (* plot solution curve vs data *)
   gsoln = Show[gdata, Plot[y[x], {x, 1900, 1970}]];
   Export[dirGraph <> name <> "_solution.pdf", gsoln];

   (* vector of residuals *)
   r = c.Join[{1}, Table[#^n, {n, degree}]] & /@ year;
   r -= census;

   (* residual error plot points *)
   err = {year, r}^T;

   (* residual error plot *)
   gerr = ListPlot[err, Frame → True, PlotRange → All];
   Export[dirGraph <> name <> "_residuals.pdf", gerr];

   (* total error *)
   χ2 = r.r // N;

   (* condition number *)
   sigmas = SingularValueList[A];
   ϰ = First[sigmas] / Last[sigmas] // N;

   (* curvature matrix *)
   ℂ = A^T.A;
   a = Diagonal[ℂ];
   Δ = Det[ℂ];

   (* uncertanties *)
s2 = χ2 / (n - (degree + 1));
δc = Sqrt[s2/Δ #] & /@ a;

(* export the error in the amplitudes *)
Export[dirData <> name <> "_amplitudes_error.txt", δc // N];
];
```

$$s2 = \frac{\chi 2}{n - (\text{degree} + 1)};$$

$$\delta c = \sqrt{\frac{s2}{\Delta}\,\#}\ \&\ /@\ a;$$

5.6.1 Collective results

The results are summarized in graphs in figure (5.1). The first graph shows the condition number increasing steadily. By the time we reach the Lagrange interpolant we have no significant figures. The plot beneath shows the decrease of the total error to zero. Without looking at the condition number the zero error may seem attractive. However poor conditioning destroys the answer. The final plot shows the extrapolation to 1990 for the various fit orders. The quartic extrapolation is not shown here because it is negative.

5.6.2 Results for each order

The following tables, (5.5) and (5.6), so the results for each order of fit. We see the solution plotted against the data and the residual errors at each point.

5.7 The moral of the story

We must always challenge ourselves by asking this: The least squares fit is the *best* fit, but is it a *good* fit?

The least squares fit minimizes in the $2-$norm the global error between a model function and a set of data. While the $2-$norm is not the best choice universally, it has properties that make it very enticing:

1. continuity: allows for differentiation;
2. Gauss-Markov theorem:[3] provides the best linear unbiased estimator;
3. popularity: widely used = widely accepted.

Accepting that the minimization will be measured in the $2-$norm, two questions manifest.

1. How well does the model describe the data?
2. How well does the method tolerate the condition of the problem formulation?

We saw that polynomial fits of higher order could reduce the global error to nothing. However, we saw that these fits became worse and worse in terms of extrapolation. Conditioning is vital. This quantifies the erosion in the quality of the answer. The data may not support additional fit parameters. Also as the conditioning grows, we must be more vigilant and aware of the solution method.

[3] *Matrix analysis and applied linear algebra*, C.D. Meyer, SIAM, 2001; p.448.

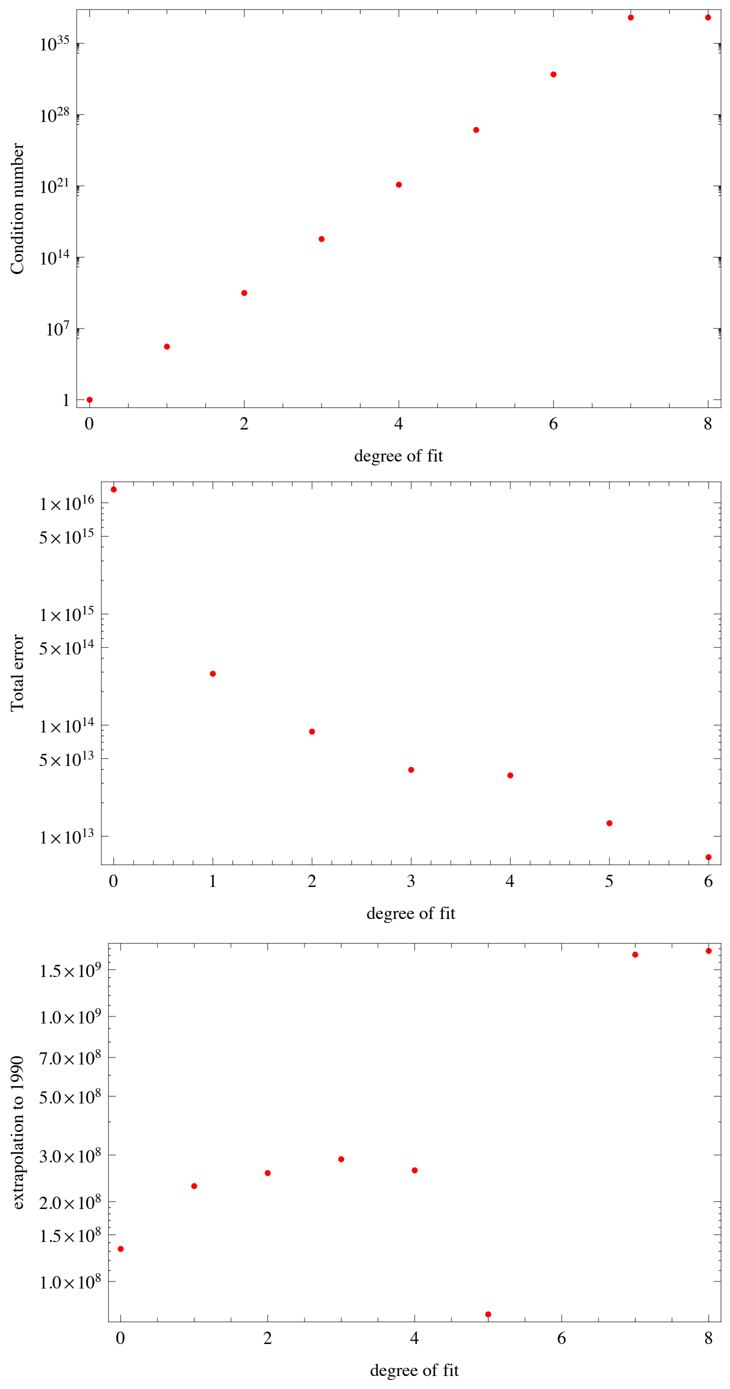


Figure 5.1: A look at fits of increasing orders. We see the condition number (top), the total arrow (middle) and the extrapolation to 1990 (bottom)

solution curve v. data　　　　residuals at each point

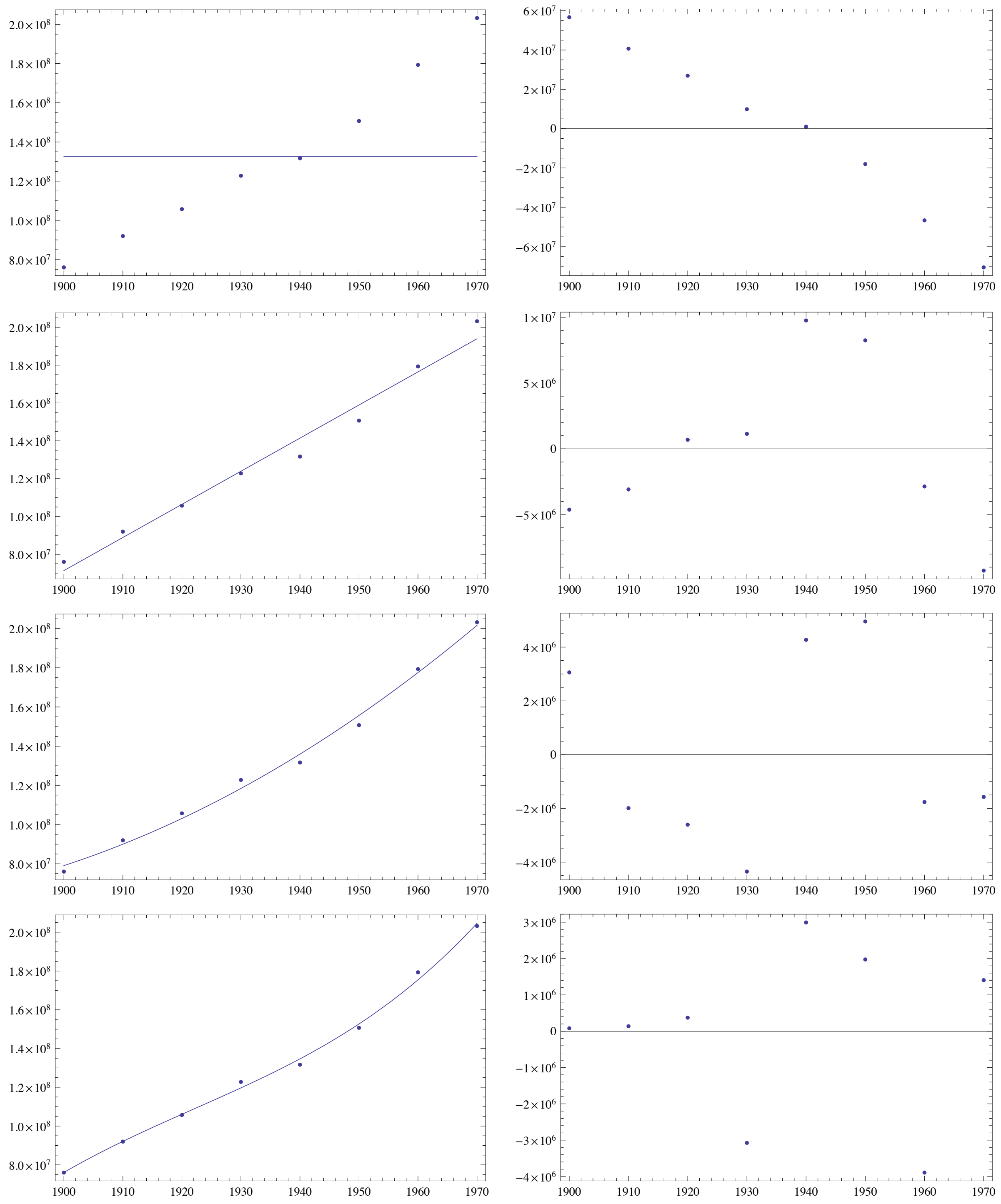


Table 5.5: Lowest order fits. The rows are fit order 0, 1, 2, 3.

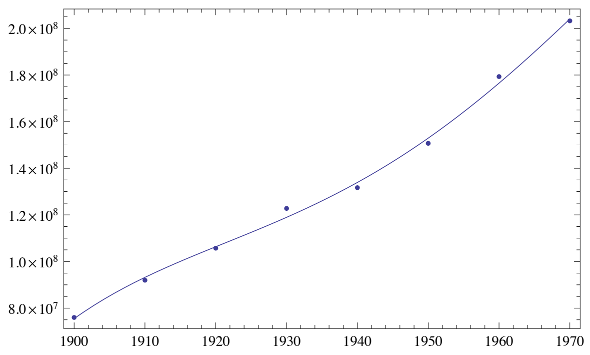
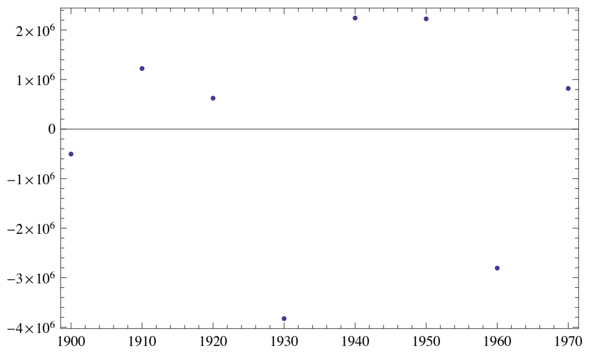
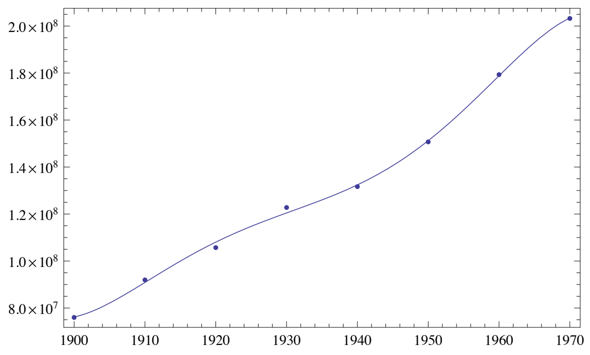
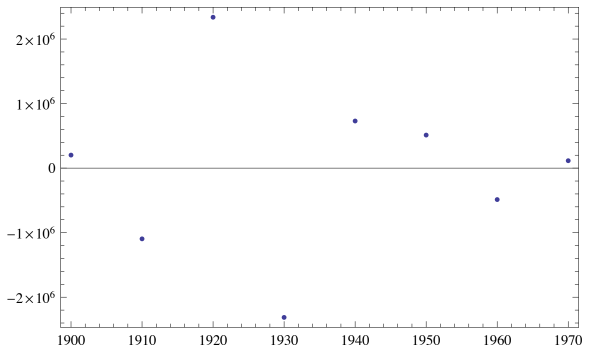
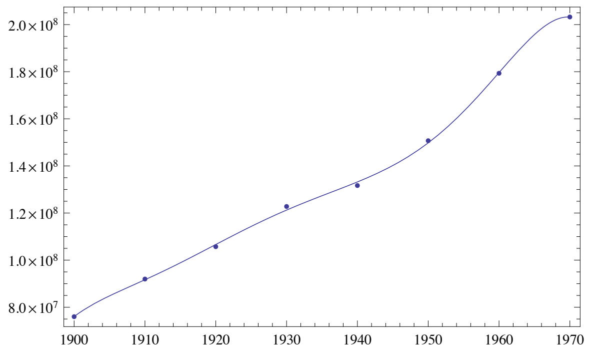
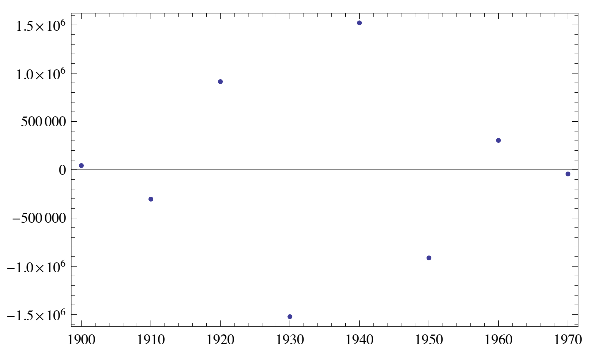
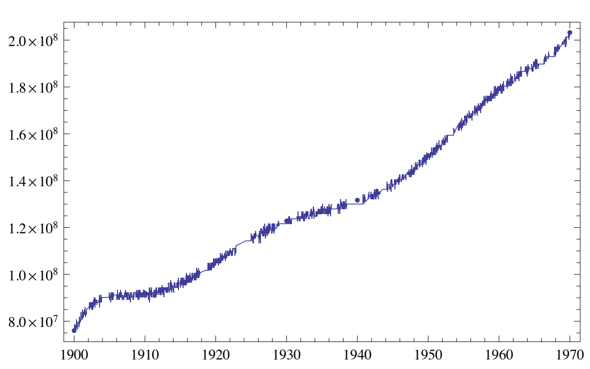
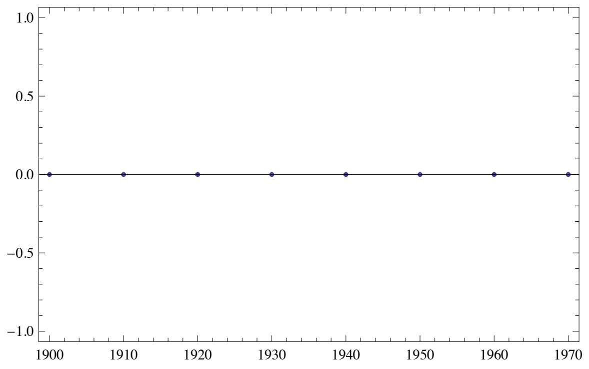
solution curve v. data	residuals at each point
	
	
	
	

Table 5.6: Lowest order fits. The rows are fit order 4, 5, 6, 7.