AMTH250

Interpolation, Least Squares

Gary Bunting

September 13, 2011

Contents

1 Interpolation **2**
1.1 Interpolation 2
1.2 Polynomial Interpolation 5
1.3 Piecewise Polynomial Interpolation 10
1.4 Cubic Splines 11
1.5 Piecewise Cubic Hermite Interpolation 12

2 Least Squares **14**
2.1 The General Problem 14
2.2 Linear Least Squares 15
2.3 Over-Determined Linear Systems 16
2.4 Least Squares in Octave 17
2.5 Transformation of Data 21
2.5.1 Power Functions 22
2.5.2 Exponential Functions 25
2.6 A Modelling Example 26

1 Interpolation

1.1 Interpolation

The general problem of interpolation is the following:

Given data

$$(x_i, y_i), \quad i = 1, \ldots n \tag{1}$$

Find a function $f(x)$ such that

$$f(x_i) = y_i, \quad i = 1, \ldots n \tag{2}$$

The equation (2) is called the **interpolation equation** or **interpolation condition**. It says that the function $f(x)$ passes through the data points. A function $f(x)$ satisfying the interpolation condition is called an **interpolating function** for the data.

General Formulation

Assume we have data (x_i, y_i), $i = 1, \ldots n$ and we wish to fit a function $y = f(x)$ to the data. The interpolation conditions are

$$f(x_i) = y_i \qquad i = 1, \ldots, n.$$

Now assume the function $f(x)$ is a linear combination of **basis functions** $f_1(x), \ldots, f_n(x)$

$$f(x) = a_1 f_1(x) + a_2 f_2(x) + \cdots + a_n f_n(x)$$

The problem then is to find the values of a_i, $i = 1, \ldots, n$ so that the interpolation conditions

$$y_i = f(x_i) = a_1 f_1(x_i) + a_2 f_2(x_i) + \cdots + a_n f_n(x_i) \quad i = 1, \ldots, n$$

are satisfied. We are assuming that we have the same number of basis functions as we have data points, so that the interpolation conditions are a system of n linear equations for the n unknowns a_i.

Writing out the interpolation conditions in full gives

$$\begin{aligned}
y_1 &= f_1(x_1)a_1 + f_2(x_1)a_2 + \cdots + f_n(x_1)a_n \\
y_2 &= f_1(x_2)a_1 + f_2(x_2)a_2 + \cdots + f_n(x_2)a_n \\
&\vdots \\
y_n &= f_1(x_n)a_1 + f_2(x_n)a_2 + \cdots + f_n(x_n)a_n
\end{aligned}$$

or, in matrix form

$$\begin{bmatrix} f_1(x_1) & f_2(x_1) & \cdots & f_n(x_1) \\ f_1(x_2) & f_2(x_2) & \cdots & f_n(x_2) \\ \vdots & \vdots & \vdots & \vdots \\ f_1(x_n) & f_2(x_n) & \cdots & f_n(x_n) \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

This shows that the general interpolation problem can be reduced to solving a system of linear equations. The matrix in these equations is called the **basis matrix** or **design matrix**. Each column of the basis matrix consists of one of the basis functions evaluated at all the x data values. The right-hand-side of the system of equations is the vector of y data values. The solution of the linear system gives the coefficients of the basis functions. The general formalism above can be used to solve many interpolation problems.

Example

We will interpolate the data

x	-1	0	1
y	1.4	0.8	1.7

by a function of the form

$$f(x) = a_1 e^{-x} + a_2 + a_3 e^{x}$$

In this case the basis functions are

$$f_1(x) = e^{-x}, \quad f_2(x) = 1, \quad f_3(x) = e^{x}.$$

Our problem is to determine the coefficients a_1, a_2, a_3. It is easily solved in Octave. The basis matrix is

$$\begin{bmatrix} e^{-x_1} & 1 & e^{x_1} \\ e^{-x_2} & 1 & e^{x_2} \\ e^{-x_3} & 1 & e^{x_3} \end{bmatrix} = \begin{bmatrix} e^{1} & 1 & e^{-1} \\ e^{0} & 1 & e^{0} \\ e^{-1} & 1 & e^{1} \end{bmatrix}$$

```
octave:> x = [-1 0 1]';
octave:> aa = [exp(-x) ones(3,1) exp(x)]
aa =
   2.71828   1.00000   0.36788
   1.00000   1.00000   1.00000
   0.36788   1.00000   2.71828
```

The right-hand-side vector is

```
octave:> b =[1.4 0.8 1.7]'
b =
   1.40000
   0.80000
   1.70000
```

and the coefficients of the basis functions are

```
octave:25> a =aa\b
a =
   0.62669
  -0.58101
   0.75432
```

So our interpolating function is (see Figure 1)

$$f(x) = 0.627e^{-x} - 0.581 + 0.754e^{x}$$

```
octave:> x=[-1 0 1]';
octave:> plot(x, b,'or')
octave:> xx = linspace(-1,1,201)';
octave:> yy = [exp(-xx) ones(201,1) exp(xx)]*a;
octave:> hold on
octave:> plot(xx,yy)
```

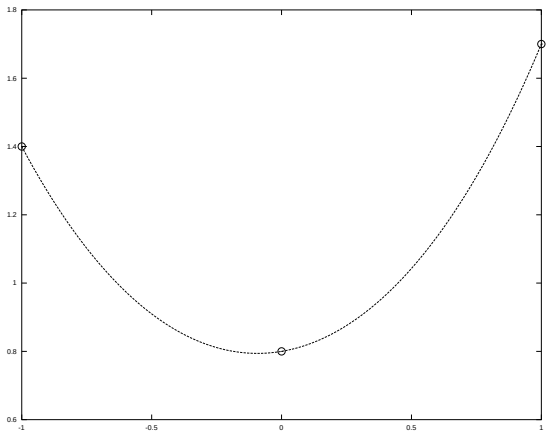

Figure 1: Interpolating function $f(x) = 0.627e^{-x} - 0.581 + 0.754e^{x}$

1.2 Polynomial Interpolation

Example

We will again interpolate the data

x	-1	0	1
y	1.4	0.8	1.7

this time by a quadratic polynomial

$$f(x) = a_1x^2 + a_2x + a_3$$

In this case the basis functions are

$$f_1(x) = x^2, \quad f_2(x) = x, \quad f_3(x) = 1$$

The basis matrix is

$$\begin{bmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ x_3^2 & x_3 & 1 \end{bmatrix} = \begin{bmatrix} 1 & -1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

and we need to solve

$$\begin{bmatrix} 1 & -1 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 1.4 \\ 0.8 \\ 1.7 \end{bmatrix}$$

Solving the problem in Octave:

```
octave:> aa = [1 -1 1; 0 0 1; 1 1 1]
aa =
   1  -1   1
   0   0   1
   1   1   1

octave:> a = aa\b
a =
   0.75000
   0.15000
   0.80000
```

giving the interpolating polynomial (see Figure 2)

$$f(x) = 0.75x^2 + 0.15x + 0.80$$

```
octave:> yy = [xx.^2 xx ones(201,1)]*a;
octave:> plot(x, b,'or')
octave:> hold on
octave:> plot(xx,yy)
```

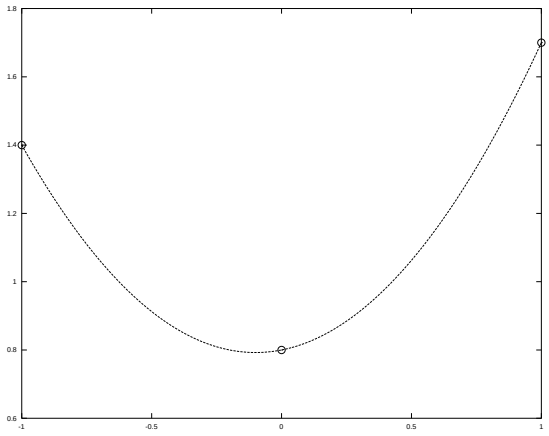

Figure 2: Interpolating polynomial $f(x) = 0.8 + 0.15x + 0.75x^2$

General Formulation

Given any set of data

$$(x_i, y_i), \quad i = 1, \ldots n,$$

there is a *unique* polynomial of degree *at most* $n-1$ which interpolates the data. Note that a polynomial of degree $n-1$ has n coefficients, the same as the number of data points. Writing the interpolating polynomial as

$$p(x) = a_1 x^{n-1} + a_2 x^{n-2} + \cdots + \cdots + a_{n-1}x + a_n$$

the basis functions are

$$f_1(x) = x^{n-1}, \quad f_2(x) = x^{n-2}, \quad \ldots, \quad f_{n-1}(x) = x, \quad f_n(x) = 1$$

For data $x_1, \ldots, x_n$ the design matrix is

$$\mathbf{A} = \begin{bmatrix} x_1^{n-1} & x_1^{n-2} & \ldots & x_1 & 1 \\ x_2^{n-1} & x_2^{n-2} & \ldots & x_2 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_n^{n-1} & x_n^{n-2} & \ldots & x_n & 1 \end{bmatrix}$$

The matrix $\mathbf{A}$ is an example of a **Vandermonde matrix**.

Therefore given data (x_i, y_i), $i = 1, \ldots n$ to find the interpolating polynomial

$$p(x) = a_1 x^{n-1} + a_2 x^{n-2} + \cdots + \cdots + a_{n-1}x + a_n$$

we solve the linear system

$$\begin{bmatrix} x_1^{n-1} & x_1^{n-2} & \dots & x_1 & 1 \\ x_2^{n-1} & x_2^{n-2} & \dots & x_2 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_n^{n-1} & x_n^{n-2} & \dots & x_n & 1 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

to find the coefficients a_i of the interpolating polynomial.

Example

A famous example due to Carl Runge (1901) shows a problem associated with polynomial interpolation. Let

$$f(x) = \frac{1}{1 + 25x^2}$$

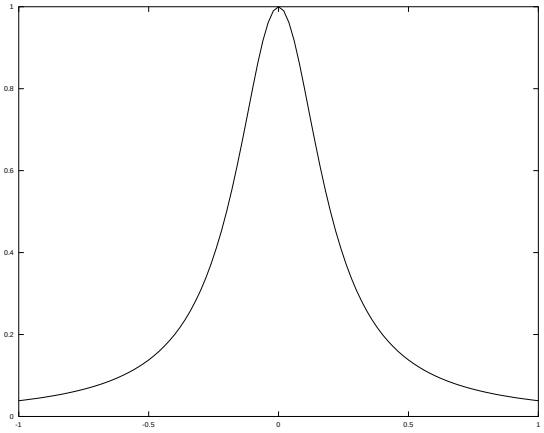

Figure 3: Runge's Function

Evaluate $f(x)$ at $(n+1)$ equally spaced points on $[-1, 1]$ and and let $P_n(x)$ denote the polynomial of degree (at most) n interpolating those function values. The following script computes the interpolation points and interpolating polynomial and then plots $f(x)$ and the interpolating polynomial:

```
% Interpolation Points
xx = linspace(-1, 1, n+1)';
yy =  1 ./ (1 + 25*xx.^2);
% Compute interpolating polynomial
```

```
aa = vander(xx); % Vandermode matrix
p = aa\yy;       % Solve linear system for coeffs of p
% Plot f(x), interpolation points and interpolating polynomial
x = linspace(-1,1,101);
y =  1 ./ (1 + 25*x.^2);
plot(x,y)
hold on
plot(xx,yy,'or')
px  = polyval(p,x);
plot(x, px, 'g')
```

A few of points to note about this script:

1. We need to set a value for `n` before executing the script.
2. The data for interpolation `xx` and `yy` are *column* vectors.
3. We have used two new Octave functions:
   - `vander` to create the Vandermonde matrix.
   - `polyval` to evaluate the interpolating polynomial.

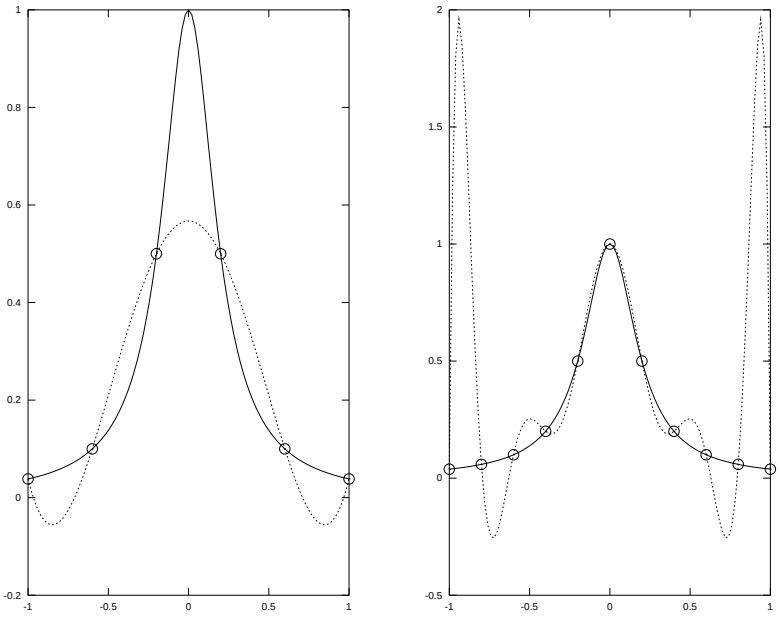

Figure 4: Interpolation by degree 5 and degree 10 polynomials

Figure 4 was produced by the commands:

```
octave:> n=5;
octave:> subplot(1,2,1)
octave:> runge
octave:> n=10;
octave:> subplot(1,2,2)
octave:> runge
```

Runge's example illustrates a problem with polynomial interpolation; the interpolating polynomial can exhibit oscillations not present in the data. These oscillations tend to get worse as the number of data points, and hence the degree of the polynomial, increases. For this reason, high degree polynomial interpolation is rarely used in practice.

Using `polyfit`

The interpolating polynomial can be computed in Octave using `polyfit`, which as we will see later is used more generally for solving least-square problems for polynomials.

We will use the following data which will also be used in other examples in this section:

x	1	2	3	4	5	6
y	16	18	21	17	15	12

Table 1: Data for interpolation problems

```
octave:> x = 1:6;
octave:> y = [16 18 21 17 15 12];
octave:> p = polyfit(x,y,5)
p =
    -0.24167     4.33333   -28.95833    87.66667  -115.80000    69.00000
```

[The last argument to `polyfit` is degree of the interpolating polynomial, in this case 5 because we have 6 data points.]

The interpolating polynomial is

$$p(x) \approx -0.24x^5 + 4.33x^4 - 28.96x^3 + 87.67x^2 - 115.80x + 69.00$$

Note that this polynomial (see Figure 5) shows signs of oscillations not present in the data particularly in the first and last subintervals.

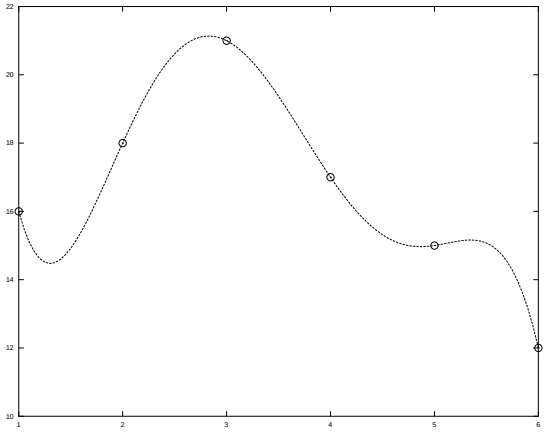

Figure 5: Interpolating Polynomial

1.3 Piecewise Polynomial Interpolation

The most important type for interpolation for practical problems is **piecewise polynomial interpolation**. Given data $(x_i, y_i), i = 1, \ldots, n$ with $x_1 < x_2 < \cdots < x_n$, a different polynomial is used on each subinterval $[x_i, x_{i+1}]$. The abscissa x_i are called variously **nodes**, **knots**, **breakpoints** or **control points**.

Piecewise Linear Interpolation

This is the simplest type of interpolation. On each subinterval the data is interpolated by a linear polynomial, i.e. a straight line, joining the data points. You may have already noticed that this is what Octave does when drawing graphs.

Example

In Octave piecewise polynomial interpolation is done using the pair of functions, `interp1` to compute the interpolating function and `ppval` to evaluate the interpolating function.

We will use the same data for `x` and `y` as in the previous example.

```
octave:> pp = interp1(x, y, 'linear', 'pp');
octave:> xx = linspace(0,7,101);
octave:> plot(x,y,'ro')
octave:> hold on
octave:> plot(xx, ppval(pp, xx))
```

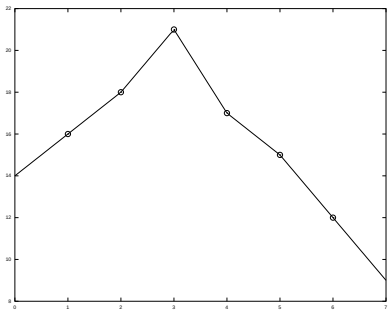

Figure 6: Piecewise Linear Interpolation

1.4 Cubic Splines

Linear interpolation, while having its uses, is not applicable when we want a *smooth* interpolating function. The most important type of interpolation by smooth functions is **cubic spline interpolation**. A modern application of cubic spline interpolation is CAD, computer aided design.

Given knots $x_1 < x_2 < \cdots < x_n$, a cubic spline is a function with the following properties:

1. On each interval $[x_i, x_{i+1}]$ the function is given by cubic polynomial. The function is defined by *different* cubic polynomials on different intervals.

2. At each interior knot $x_2, \ldots x_{n-1}$ the cubic polynomials on adjoining intervals have the property that their values and the values of their first and second derivatives match.

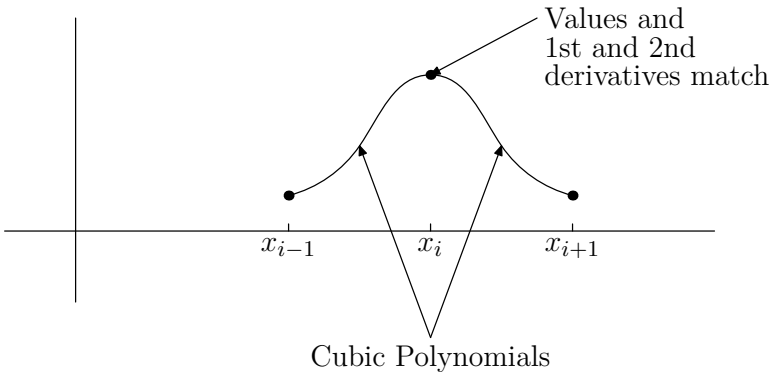

Figure 7: Cubic Spline

The $n-1$ cubic functions are determined by $4(n-1)$ coefficients. The imposed conditions are n interpolation conditions for all n nodes and $3(n-2)$ matching conditions for the $n-2$ interior nodes. This leaves 2 free parameters. There are several ways to determine those. Octave uses the "not-a-knot" condition which matches the third derivatives at the both the second and $n-1^{\text{st}}$ nodes. (This is the same as to require that the cubic polynomial on the first and second interval and on the last and second last interval coincide).

Example

We will use the same data for `x` and `y` as in the previous examples.

```
octave:> pp = interp1(x, y, 'spline', 'pp');
octave:> plot(x,y,'ro')
octave:> hold on
octave:> xx = linspace(0,7,101);
octave:> plot(xx, ppval(pp, xx))
```

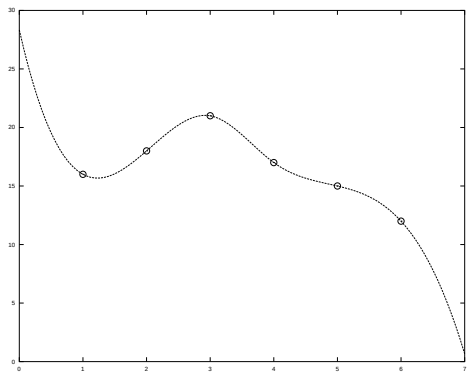

Figure 8: Cubic Spline Interpolation

1.5 Piecewise Cubic Hermite Interpolation

An alternative to cubic spline interpolation is **piecewise cubic Hermite interpolation** or **pchip** interpolation. Again the interpolating function is a cubic polynomial on each subinterval, but the cubic spline condition that the second derivatives match at the knots is dropped. Instead the slope of the interpolating function at the knots is chosen to preserve the "shape" of the data. In particular if the data is monotonic then the interpolating function is monotonic.

Example

We use the same data as before.

```
octave:> pp = interp1(x, y, 'pchip', 'pp');
octave:> plot(x,y,'ro')
octave:> hold on
octave:> xx = linspace(0,7,101);
octave:> plot(xx, ppval(pp, xx))
```

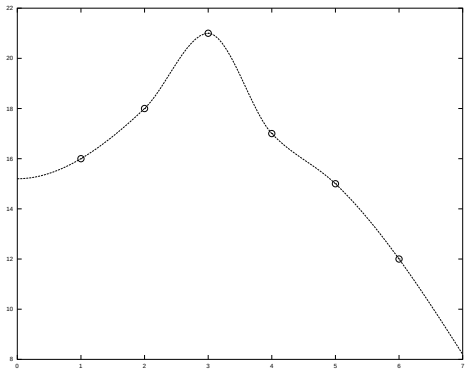

Figure 9: Piecewise Cubic Hermite Interpolation

The cubic spline and cubic Hermite interpolants are compared in Figure 10.

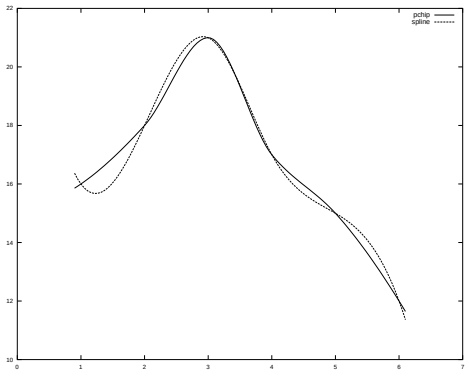

Figure 10: Spline and PCHIP Interpolation

2 Least Squares

In the previous section we looked at interpolation problems where we were given some data (x_i, y_i), $i = 1, \ldots, n$ and we wanted to find a function $y = f(x)$ which interpolated the data so that $y_i = f(x_i)$, $i = 1, \ldots, n$. This approach is only useful when the data are relatively free from error, otherwise the interpolating function will exhibit the same kind of fluctuations that are present in the data.

When we are dealing with data containing random errors the most common approach to fitting a function to data is the method of least squares.

2.1 The General Problem

The general least squares problem can be formulated as follows: given data (x_i, y_i), $i = 1, \ldots, n$ and a function

$$y = f(x, a_1, a_2, \ldots, a_k)$$

depending on the parameters $a_1, \ldots, a_k$, let

$$r_i = y_i - f(x_i, a_1, \ldots, a_k)$$

denote ith **residual**, the distance between the graph of $f(x, a_1, a_2, \ldots, a_k)$ and the data point (x_i, y_i) (see Figure 11).

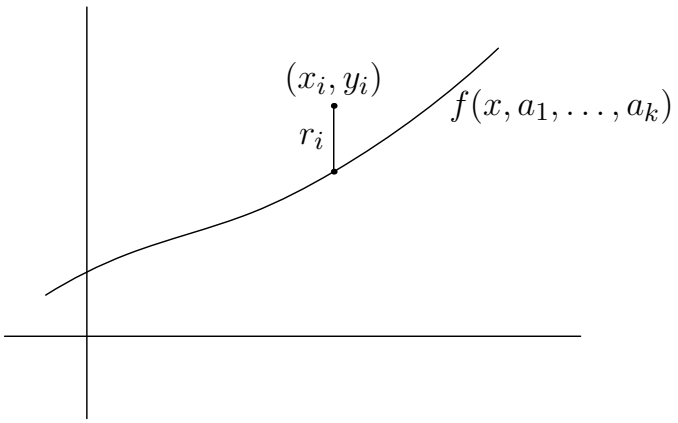

Figure 11: Least squares problem

We want to find values for the parameters $a_1, \ldots, a_k$ which minimizes the sum of squares of the r_i

$$\text{Minimize} \quad S^2 = \sum_{i=1}^{n} r_i^2 = \sum_{i=1}^{n} (y_i - f(x_i, a_1, \ldots, a_k))^2$$

Typically the number of parameters, k, is much smaller than the number of data points, n.

Least squares problems are optimization problems. They are classified as linear or nonlinear, depending on whether the function $f(x, a_1, \ldots, a_k)$ depends linearly or non-linearly on the parameters $a_1, \ldots, a_k$. Nonlinear least squares problems are often difficult to solve and the theory behind them is rather involved.

2.2 Linear Least Squares

The most common data fitting problem is fitting a straight line

$$y = ax + b$$

to data. This is an example of a linear least squares problem with two parameters, in this case a and b, to be determined. More generally, fitting a polynomial of degree k

$$y = a_0 + a_1 x + a_2 x^2 + \cdots + a_k x^k$$

to data is another example of linear least squares. Although y is a nonlinear function of x it is a *linear* function of the parameters $a_0, \ldots, a_k$.

Linear least squares have a number of similarities to the general interpolation problem discussed in the previous section. Like interpolation it can be formulated as a problem in linear algebra.

The Computational Problem

Saying that the function $f(x, a_1, a_2, \ldots, a_k)$ depends linearly on the parameters a_j means that it can be written as a linear combination of some basis functions $f_1(x), \ldots, f_k(x)$:

$$f(x, a_1, a_2, \ldots, a_k) = a_1 f_1(x) + a_2 f_2(x) + \cdots + a_k f_k(x)$$

The least squares approach leads us to minimizing the sum of squares

$$S^2 = \sum_{i=1}^{n} [y_i - (a_1 f_1(x_i) + a_2 f_2(x_i) + \cdots + a_k f_k(x_i))]^2$$

Note that the sum is over the data points, and once the data and basis functions are given, the sum of squares S^2 is a function of the parameters $a_1, \ldots, a_k$ only.

The individual terms inside the square brackets

$$r_i = y_i - (a_1 f_1(x_i) + a_2 f_2(x_i) + \cdots + a_k f_k(x_i))$$

can be written in matrix form

$$\begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} - \begin{bmatrix} f_1(x_1) & f_2(x_1) & \dots & f_k(x_1) \\ f_1(x_2) & f_2(x_2) & \dots & f_k(x_2) \\ \vdots & \vdots & \vdots & \vdots \\ f_1(x_n) & f_2(x_n) & \dots & f_k(x_n) \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_k \end{bmatrix}$$

The matrix in this equation is the same as the **basis** or **design matrix** used in interpolation. Like interpolation our aim is to find the coefficients $a_1, \dots, a_k$, this time to minimize the sum of squares of the residuals r_i. Note that when we have the same number of parameters a_i as data points, the problem reduces to an interpolation problem. In this case we could find values for the parameters so that all the residuals r_i are zero.

2.3 Over-Determined Linear Systems

The formulation of the linear least squares problem given above is closely related to solving linear equations when there are more equations than unknowns. Consider a system of n linear equations in k unknowns:

$$\begin{array}{ccccccccc} b_{11}a_1 & + & b_{12}a_2 & + & \cdots & + & b_{1k}a_k & = & y_1 \\ b_{21}a_1 & + & b_{22}a_2 & + & \cdots & + & b_{2k}a_k & = & y_2 \\ \vdots & & \vdots & & & & \vdots & & \vdots \\ b_{n1}a_1 & + & b_{n2}a_2 & + & \cdots & + & b_{nk}a_k & = & y_n \end{array}$$

In matrix form,

$$\mathbf{Ba} = \mathbf{y}$$

where $\mathbf{B}$ is a given $n \times k$ matrix, $\mathbf{y}$ is a given n vector, and $\mathbf{a}$ is the k vector to be solved for.

When $n > k$ then the system of equations will not, in general, have a solution. For any vector $\mathbf{a}$ the residual is

$$\mathbf{r} = \mathbf{Ba} - \mathbf{y}$$

When we have the same number of equations as unknowns, the residual is zero for the solution vector $\mathbf{a}$. When we have more equations than unknowns, there is no vector $\mathbf{a}$ for which the residual is zero.

What we can do in this case is to try to find a vector $\mathbf{a}$ for which the residual is as small as possible. If we measure the size of the residual by its norm

$$||\mathbf{r}|| = \left[\sum_{i=1}^{m} r_i^2 \right]^{\frac{1}{2}}$$

then minimizing the norm $||\mathbf{r}||$ is equivalent to minimizing the sum of squares

$$S^2 = \sum r_i^2.$$

The linear least squares problem has exactly the same mathematical structure. Given (x_i, y_i), $i = 1, \ldots, n$ and basis functions $f_1(x), \ldots, f_k(x)$ form the basis matrix

$$\mathbf{B} = \begin{bmatrix} f_1(x_1) & f_2(x_1) & \ldots & f_k(x_1) \\ f_1(x_2) & f_2(x_2) & \ldots & f_k(x_2) \\ \vdots & \vdots & \vdots & \vdots \\ f_1(x_n) & f_2(x_n) & \ldots & f_k(x_n) \end{bmatrix}$$

then we want to find the coefficients a_i of the basis functions minimize the norm of the residual

$$\mathbf{r} = \mathbf{Ba} - \mathbf{y}.$$

2.4 Least Squares in Octave

Least squares problems, which as we have seen are equivalent to over-determined linear systems, are solved in Octave with the backslash operator \ the same way as linear equations are solved.

Example

We will look at example of fitting a straight line to data. Our basis functions are $f_1(x) = x$ and $f_2(x) = 1$ and the basis matrix is the Vandermonde matrix

$$\begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \vdots \\ x_n & 1 \end{bmatrix}$$

For our example we will generate some x and y data where

$$y = -3x + 5 + \text{random perturbation}$$

(see Figure 12).

```
octave:> x = (0:10)';
octave:> y = - 3*x1 + 5 + randn(11,1);
```

To fit a straight line to this data we first create the design matrix:

```
octave:> aa = vander(x,2);
```

The second argument to `vander` specifies the number of columns.

Solving with the backslash operator gives

```
octave:> p = (aa1\y)'
p =
  -2.9032   4.8575
```

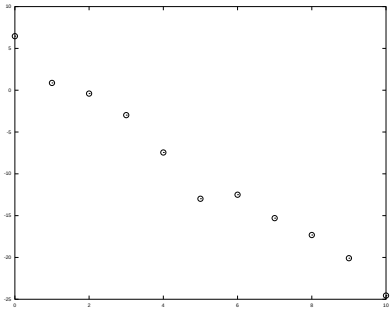

Figure 12: Data for straight line least squares problem

The components of the solution, `p`, are the coefficients of the basis functions $f_1(x) = x$ and $f_2(x) = 1$, so our least squares straight line is

$$y = -2.9032x + 4.8575$$

The same result could have been obtained more easily using `polyfit`

```
octave:> polyfit(x,y,1)
ans =
  -2.9032   4.8575
```

Here the last argument to `polyfit` is the degree of the polynomial, in this case 1.

```
octave:> xx = linspace(0,10,2);
octave:> plot(xx, polyval(p,xx))
```

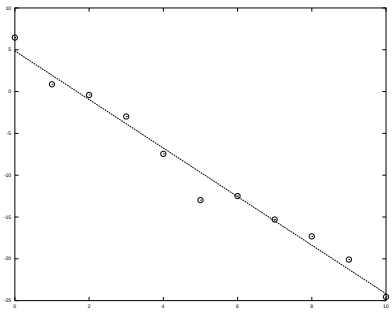

Figure 13: Least squares straight line

The residuals, that is the difference between the data values and the fitted straight line can be easily computed

```
octave:> res = y - polyval(p,x);
octave:> plot(x, res,'or')
octave:> hold on
octave:> plot([0 10],[0 0])
```

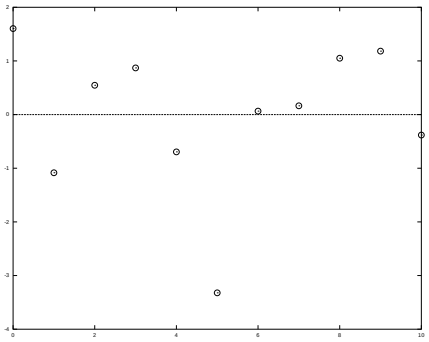

Figure 14: Residuals for least squares straight line

Example

Again we will generate some data, but this time using trigonometric functions:

```
octave:> x = (0:20)';
octave:> y = 4*sin(x)+3*sin(2*x)+2*sin(4*x)+0.5*randn(21,1);
```

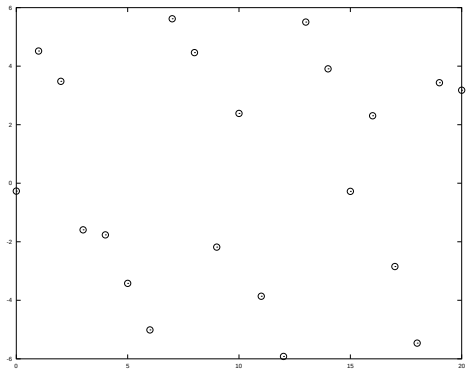

Figure 15: Trigonometric Data

In this problem our basis functions will be

$$f_1(x) = \sin(x) \quad f_2(x) = \sin(2x) \quad f_3(x) = \sin(4x)$$

Our computations follow the same pattern as before; construct the design matrix and then use the \ operator to find the least squares solution.

```
octave:> aa = [sin(x) sin(2*x) sin(4*x)];
octave:> a = aa\y
a =
   3.9323
   2.7317
   2.0234
```

So are fitted function is

$$f(x) = 3.9323\sin(x) + 2.7317\sin(2x) + 2.0234\sin(4x)$$

```
octave:> xx = linspace(0,20,201)';
octave:> yy = [sin(xx) sin(2*xx) sin(4*xx)]*a;
octave:> plot(xx,yy)
```

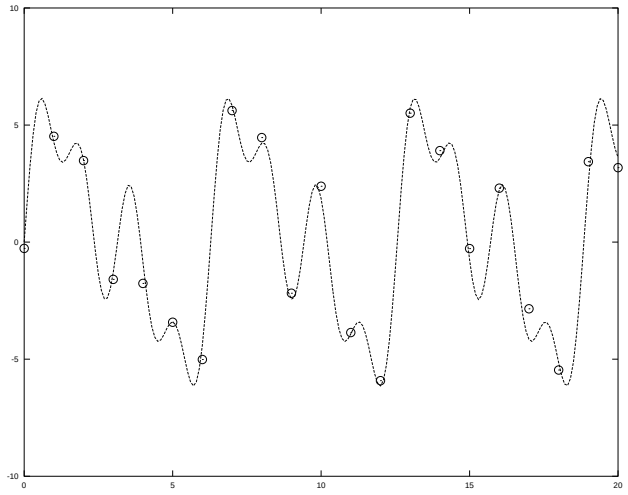

Figure 16: Fitted Trigonometric Function

Using `polyfit`

In general

```
polyfit(x, y, n)
```

finds the least-squares polynomial of degree `n` fitting the data `x` and `y`. When `n = 1` we get the least squares straight line, and when `n` is one less than the number of data points we get the interpolating polynomial.

Computing the least squares polynomial involves computing the design matrix, in this case a Vandermonde matrix, and then solving the resulting overdetermined system of linear equations in the least squares sense. One problem with this is that the Vandermonde matrix is often ill-conditioned. For example, if the x data is $0, \ldots, 20$ then the condition number of the Vandermonde matrix is

```
octave:> x = (0:20)';
octave:> cond(vander(x))
ans =  3.7129e+31
```

Even if we fit a degree 7 polynomial to this data, the condition number is still uncomfortably large

```
octave:> cond(vander(x,7))
ans =  1.6603e+08
```

`polyfit` attempts to mitigate against the effects of the ill-conditioning of the Vandermonde matrix matrix by **rescaling** the x abscissa by

$$x' = \frac{x - \mu}{\sigma}$$

where μ and σ are the mean and standard deviation of the x data.

To see how this works we can perform the computation in Octave

```
octave:> x1 = (x - mean(x))/std(x);
octave:> cond(vander(x1))
ans =  4.3271e+09
octave:> cond(vander(x1,7))
ans =  95.998
```

This is a substantial improvement conditioning.

2.5 Transformation of Data

Recall that the general least squares problem is to find values for the parameters $a_1, \ldots, a_k$ which minimizes the sum of squares of the residuals r_i

$$\text{Minimize} \quad S^2 = \sum_{i=1}^{n} r_i^2 = \sum_{i=1}^{n} (y_i - f(x_i, a_1, \ldots, a_k))^2$$

When $f(x_i, a_1, \ldots, a_k)$ is a *linear* function of the parameters $a_1, \ldots, a_k$ the sum of squares S^2 is *quadratic* function of $a_1, \ldots, a_k$ for which the minimum can be found by linear algebra. When $f(x_i, a_1, \ldots, a_k)$ is a *nonlinear* function of the parameters $a_1, \ldots, a_k$ the mathematical problem is much more difficult.

Some nonlinear problems can be converted to linear problems by transforming the data $(x_,y_i)$. We will look at two very important examples.

2.5.1 Power Functions

If

$$y = ax^p \tag{3}$$

then taking logarithms of both sides gives

$$\log y = p \log x + \log a \tag{4}$$

and $\log y$ is a *linear* function of $\log x$. This has two consequences

1. If we plot $\log y$ vs $\log x$, a **loglog** plot, the result will be a straight line, and

2. If we are given data (x_i, y_i) which we believe are related by a power function, equation (3), (perhaps as the result of looking at a log-log plot of the data), then estimates of the parameters a and p can be found by fitting a straight line to the *transformed* data $(\log x_i, \log y_i)$.

Example

Recall the example of the forward difference approximation to the derivative

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$

A log-log plot of the relative error in the approximation as a function of h gave

We observed that as h decreases the main source of error is, at first, the error is the approximation itself, i.e. truncation error. At $h \approx 10^{-8}$ the error begins to increase. The main source of error now is cancellation and the error increases until at $h = 10^{-16}$ the approximate derivative is zero and completely useless.

In the log-log plot the two individual sources of error, truncation and cancellation, dominate the right and left hand sides of the graph respectively. Further, given that each side of the graph is an approximate straight line, it would appear that truncation error and cancellation error are both, at least approximately, power functions of h.

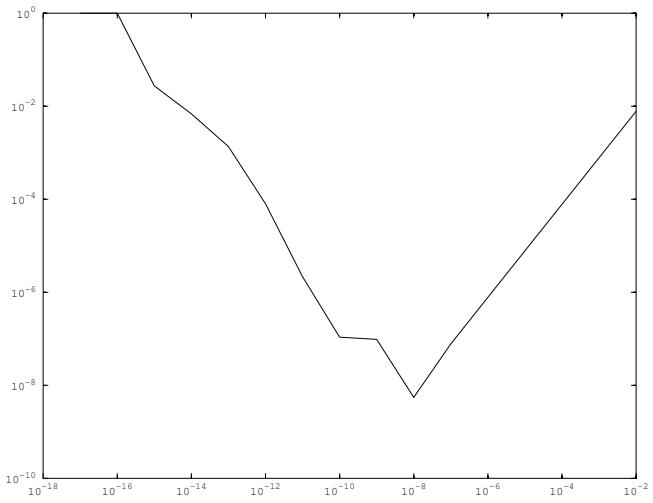

Figure 17: Log-log plot of error as a function of h. Note: h decreases from right to left

Assume that, the individual errors are

$$E_{\text{truc}} = a_t h^{p_t} \qquad E_{\text{canc}} = a_c h^{p_c}$$

so that the total error is

$$E_{\text{tot}} = E_{\text{truc}} + E_{\text{canc}} = a_t h^{p_t} + a_c h^{p_c} \tag{5}$$

Taking logs of equation (5) will not give a linear relation between $\log E_{\text{tot}}$ and $\log h$. Therefore we break the data into two parts, corresponding to truncation error and cancellation error, and determine the parameters appearing in each part separately.

The actual data are in Octave are `h` $= 10^{-2}, 10^{-3}, \ldots, 10^{-17}$ and `err`. We will take truncation error to be associated to data values with $h < 10^{-8}$ and cancellation error to be associated to data values with $h > 10^{-8}$ (and leave out the value $h = 10^{-8}$).

```
octave:> ht = h(1:6);
octave:> errt = err(1:6);
octave:> hc = h(8:16);
octave:> errc = err(8:16);
```

Now we take logs of the data and fit a straight line using `polyfit`

```
octave:> ht = h(1:6);
octave:> errt = err(1:6);
octave:> hc = h(8:16);
```

```
octave:> errc = err(8:16);
octave:> polyfit(log10(ht), log10(errt), 1)
ans =
   1.00050  -0.10662
octave:> polyfit(log10(hc), log10(errc), 1)
ans =
   -0.98474  -16.17182
```

Therefore we have

$$\begin{aligned}
\log_{10} E_{\text{truc}} &= 1.00050 \log_{10} h - 0.10662 \\
E_{\text{truc}} &= 0.78232 h^{1.00050} \\
\log_{10} E_{\text{canc}} &= -0.98474 \log_{10} h - 16.17182 \\
E_{\text{canc}} &= 6.7326 \times 10^{-17} h^{-0.98474}
\end{aligned}$$

with total error

$$E_{\text{tot}} = 0.78232 h^{1.00050} + 6.7326 \times 10^{-17} h^{-0.98474}$$

```
octave:> loglog(h,err,'ro')
octave:> hold on
octave:> hh = logspace(-2,-17,100);
octave:> etot = 0.78232*hh.^1.00050 + 6.7326e-17*hh.^-0.98474;
octave:> loglog(hh,etot)
```

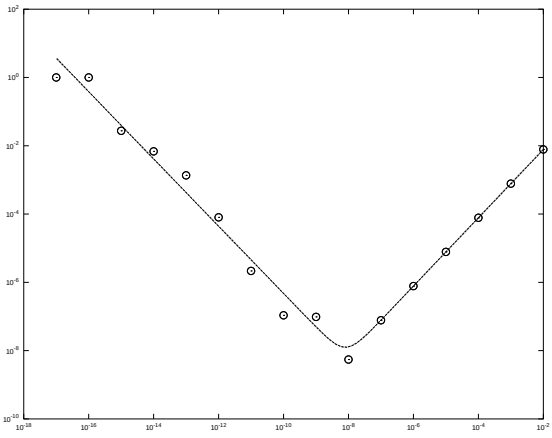

Figure 18: Log-log plot of error as a function of h.

2.5.2 Exponential Functions

Exponential functions of the form

$$y = ae^{kx} \tag{6}$$

are common in applications, e.g. you should be familiar with exponential growth and decay. Taking logarithms of both sides gives

$$\log y = kx + \log a \tag{7}$$

and $\log y$ is a *linear* function of x. Consequently

1. If we plot $\log y$ vs x, a **semilog** plot, the result will be a straight line, and
2. If we are given data (x_i, y_i) which we believe are related by a exponential function, equation (6), then estimates of the parameters a and k can be found by fitting a straight line to the *transformed* data $(x_i, \log y_i)$.

Example

Let us look at the conditioning of the Hilbert matrix $\mathbf{H}_n$ from Assignment 5:

```
octave:> n=1:12;
octave:> chilb = zeros(1,12);
octave:> for k = 1:12
> chilb(k) = cond(hilb(k));
> end
octave:> semilogy(chilb)
```

The semilog plot indicates that the condition number is an exponential function of n

$$\text{cond}(\mathbf{H}_n) \approx ae^{kn}$$

We will determine the parameters a and k.

```
octave:> lch =log(chilb);
octave:> polyfit(n, lch, 1)
ans =
   3.4236  -3.8820
```

Therefore we have

$$k = 3.4236 \quad \ln a = -3.8820$$

and so

$$\text{cond}(\mathbf{H}_n) \approx 0.020609e^{3.4236n}$$

To check our result

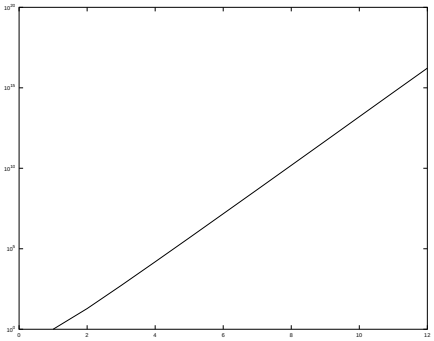

Figure 19: Condition number of $\mathbf{H}_n$, $n = 1, \ldots, 12$.

```
octave:> ch = 0.020609*exp(3.4236*n);
octave:> semilogy(n,chilb,'or')
octave:> hold on
octave:> semilogy(n,ch)
```

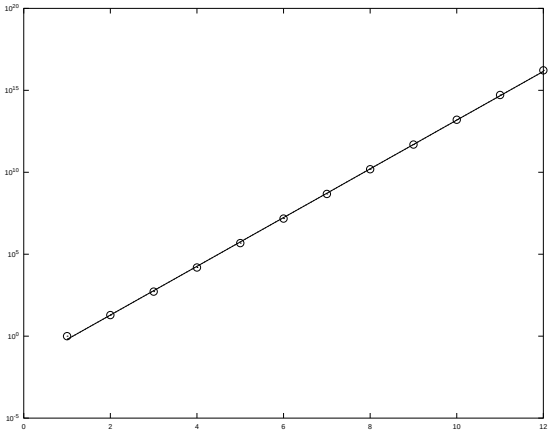

Figure 20: Condition number of $\mathbf{H}_n$.

2.6 A Modelling Example

The data

`ftp://ftp.cmdl.noaa.gov/ccg/co2/trends/co2_mm_mlo.txt`

gives monthly records of atmospheric CO_2 concentrations in parts per million at Mauna Loa, Hawaii from 1958 to the present day.

We read the data into Octave and plot the CO_2 concentration against time. The relevant quantities, time and CO_2 concentration, are in columns 3 and 5 of the data.

```
octave:> co2 = load('co2_mm_mlo.txt');
octave:> t = co2(:,3);
octave:> conc = co2(:,5);
octave:> plot(t,conc);
```

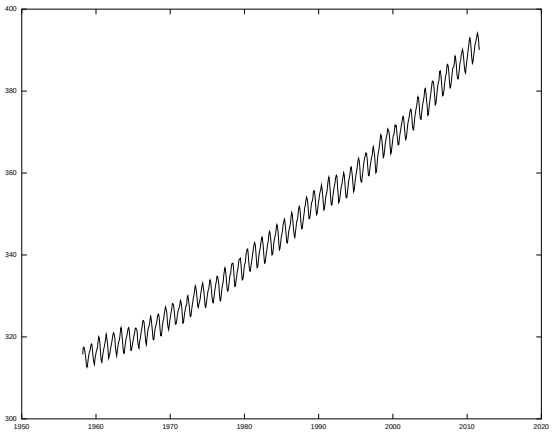

Figure 21: Atmospheric CO_2 concentration (ppm).

The data show a general increase on which is superimposed a periodic component of period 1 year. The yearly cycle is to be expected. The general increasing trend looks as though it might be exponential, but a more likely model is

$$\text{Conc} \approx a + be^{kt}$$

which cannot be linearized by a log transformation. Fitting a model of this form is a genuine nonlinear least squares problem.

We will begin by first looking at the general trend in the data and then looking at the yearly cycle. To get started we will fit a straight line to the data and compare it to the observed data:

```
octave:> p1 = polyfit(t, conc, 1)
p1 =
      1.4536  -2537.9367
octave:> plot(t, conc)
```

```
octave:> hold on
octave:> plot(t, polyval(p1,t), 'r')
octave:> plot(t,conc - polyval(p1,t))
```

The second plot graphs the residuals.

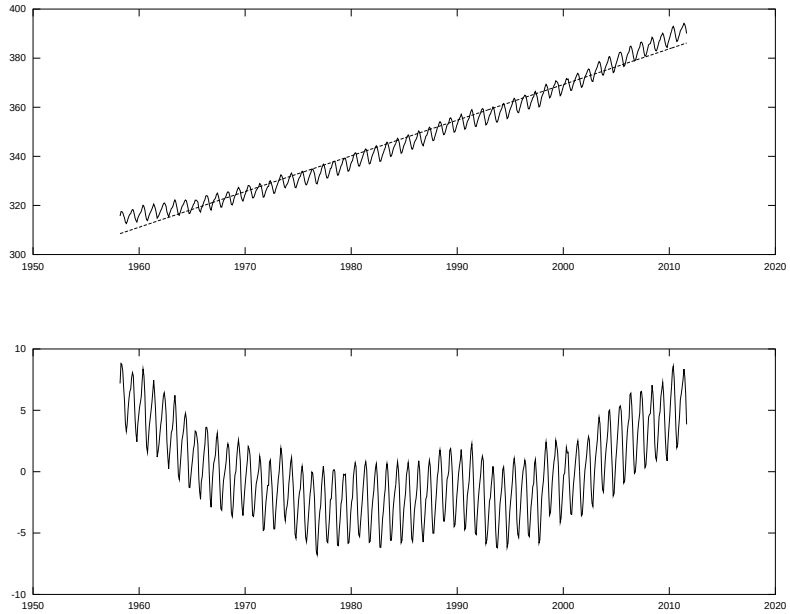

Figure 22: Linear approximation and residuals

The curvature in the residuals indicates that adding a quadratic term might improve the fit. With hindsight at least, this is apparent in the original plot of the data where the trend of the data can be visualized as part of the graph of a quadratic function. While we are at it we can fit higher order polynomials as well.

```
octave:> p2 = polyfit(t, conc, 2);
octave:> p3 = polyfit(t, conc, 3);
octave:> p4 = polyfit(t, conc, 4);
octave:> p5 = polyfit(t, conc, 5);
```

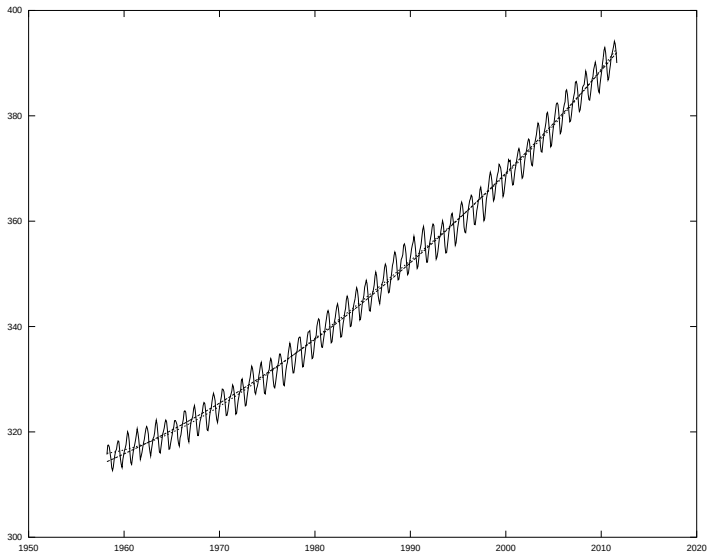

Figure 23: Data and 2nd and 4th degree polynomial fits.

To plot the residuals:

```
octave:> subplot(2,2,1)
octave:> plot(t, conc - polyval(p2,t))
octave:> subplot(2,2,2)
octave:> plot(t, conc - polyval(p3,t))
octave:> subplot(2,2,3)
octave:> plot(t, conc - polyval(p4,t))
octave:> subplot(2,2,4)
octave:> plot(t, conc - polyval(p5,t))
```

A useful measure of the size of the residuals is their norm:

```
octave:42> norm(conc - polyval(p1,t))
ans =  86.356
octave:43> norm(conc - polyval(p2,t))
ans =  55.122
octave:44> norm(conc - polyval(p3,t))
ans =  54.979
octave:45> norm(conc - polyval(p4,t))
ans =  54.056
octave:46> norm(conc - polyval(p5,t))
ans =  54.054
```

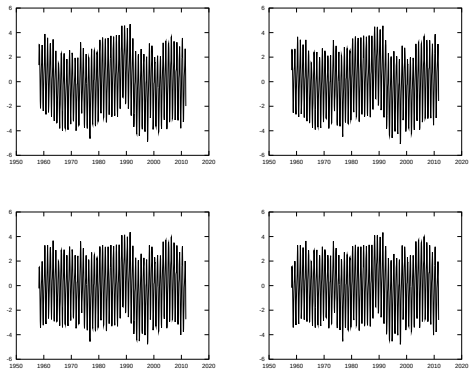

Figure 24: Residuals for degree 2 to 5 polynomial approximations.

We see that there is noticeable decrease from degree 1 to degree 2, then smaller decreases as we increase the degree until degree 5 shows little improvement over degree 4.

Now we include periodic part of the data. The amplitude of the oscillations does not change significantly with time, but the plots of the residuals do not show the shape of the individual oscillations, We look at a random bit of the residual plot for the 4th degree polynomial closely:

```
octave:> plot(t, conc - polyval(p4,t))
octave:> axis([1960 1965 -6 6])
```

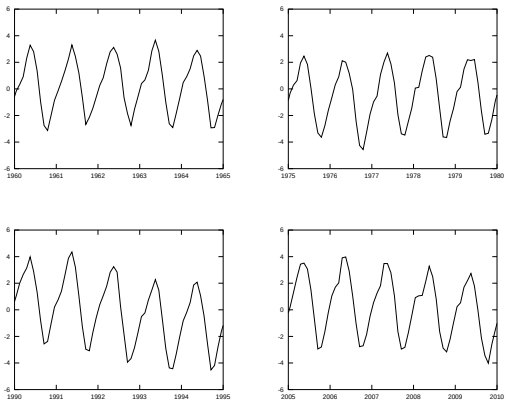

Figure 25: Portions of the residual plot for 4th degree polynomial.

The simplest periodic functions, for given period T are the sinusoidal functions

$$A\sin(2\pi t/T + \phi) = a\sin(2\pi t/T) + b\cos(2\pi t/T)$$

The first form is easier to understand, the three parameters being the period T, the amplitude A and the phase ϕ. The second form is convenient computationally as it is linear in the parameters a and b allowing linear least squares to be used.

Although the shape of the oscillations are not exactly sinusoidal, a sinusoid is a good first approximation. It is assumed that the oscillatory part of the data has period 1 year, so we will add terms

$$a\sin(2\pi t/T) + b\cos(2\pi t/T) \tag{8}$$

to the model with $T = 1$ year.

To add the periodic terms to our model we must construct the design matrix manually. It is important to rescale t to avoid problems with ill-conditioning (for the polynomials `polyfit` did this for us behind the scenes). We use the same rescaling as `polyfit`

$$t' = \frac{t - \mu}{\sigma}$$

where μ and σ are the mean and standard deviation of the t data.

```
octave:> mt = mean(t);
octave:> st =std(t);
octave:> tt = (t - mt)/st;
```

The period of the rescaled data is $T = 1/\sigma$, so the term $k = 2\pi/T = 2\pi\sigma$ in equation (8) is

```
octave:> k = 2*pi*st
k =  97.114
```

Our first model will take a polynomial of degree 4 plus the sinusoidal (8)

$$a_1t^4 + a_2t^3 + a_3t^2 + a_4t + a_5 + a_6\sin kt + a_7\cos kt \tag{9}$$

The design matrix is

```
octave:> aa41 = [vander(tt,5) sin(k*tt) cos(k*tt)];
```

and the coefficients in our model (9) are

```
octave:> a41 = aa41\conc;
```

We evaluate the model (9) at the data points t' by

```
octave:> c41 = polyval(a41(1:5),tt) + aa41(:,6:7)*a41(6:7);
octave:> plot(t,conc)
```

(you should try to understand why the first line, which evaluates the model function, works!)

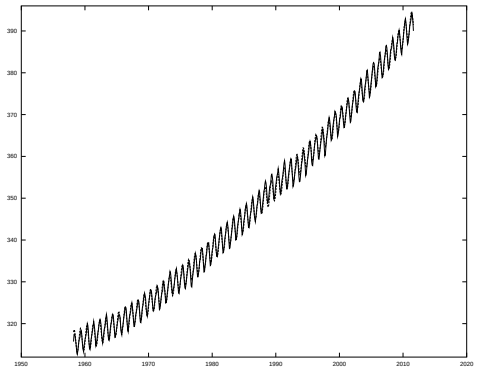

Figure 26: Model (9) fitted to data.

```
octave:> norm(conc-c41)
ans =  20.589
```

Note the reduction in the norm of the residuals.

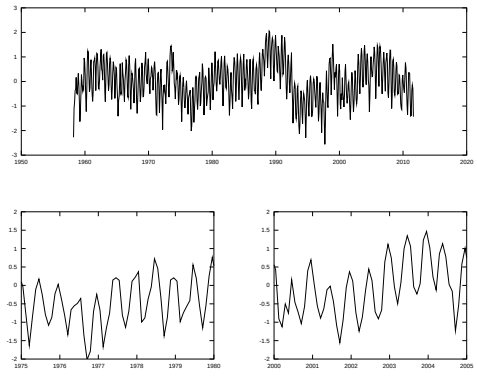

Figure 27: Residuals for model (9).

The oscillations in the residuals now show a period of half a year. This suggests adding terms

$$a\sin(2kT) + b\cos(2kT)$$

to the model. The model is now

$$a_1t^4 + a_2t^3 + a_3t^2 + a_4t + a_5 + a_6\sin kt + a_7\cos kt + a_8\sin 2kt + a_9\cos 2kt \quad (10)$$

```
octave:> aa42 = [vander(tt,5) sin(k*tt) cos(k*tt) sin(2*k*tt) cos(2*k*tt)];
octave:> a42 = aa42\conc;
octave:> c42 = polyval(a42(1:5),tt) + aa42(:,6:9)*a42(6:9);
octave:> norm(conc-c42)
ans =  15.356
```

This gives another substantial reduction in the norm of the residuals.

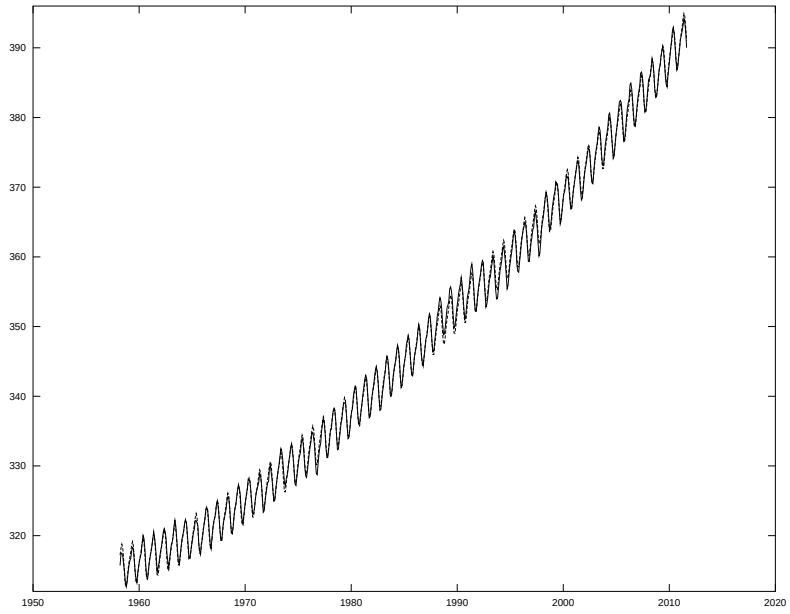

Figure 28: Model (10) fitted to data.

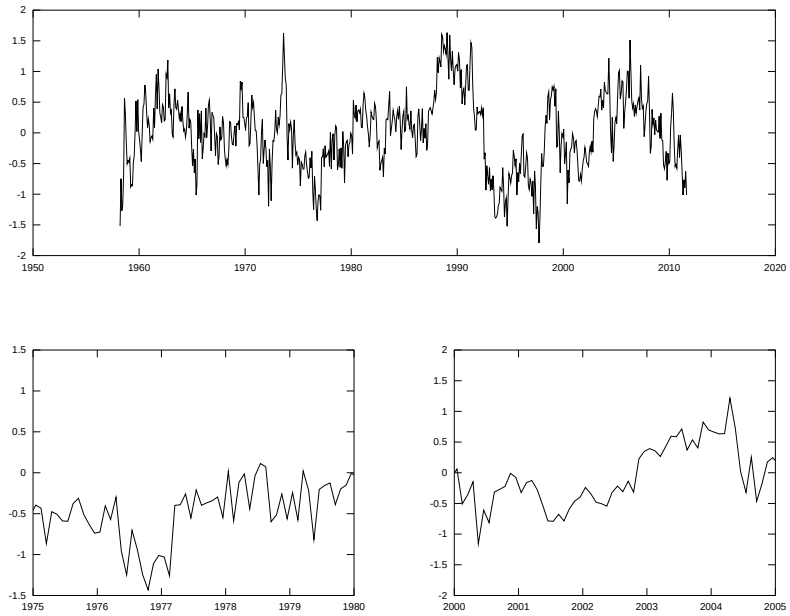

Figure 29: Residuals for model (10).