# LECTURE 7: Smoothing of data and the method of least squares

October 17, 2011

## 1 Introduction

Suppose that the surface tension $S$ of a particular liquid has been measured at eight different temperatures $T$. The surface tension is known to be a linear function of temperature, but the measured values do not fall exactly on a straight line.

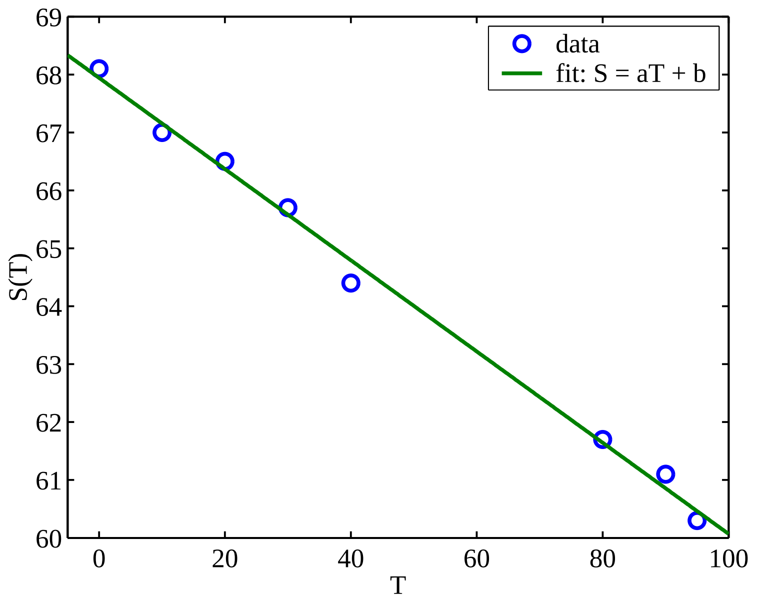

Figure 1: A set of measured data and the corresponding line fit.

Using statistical methods, we can determine the *most probable* values of the constants $a$ and $b$ in the equation

$$S = aT + b$$

# 2 Method of least squares

## 2.1 Linear least squares

An experiment or survey often produces a mass of data which may fall *approximately* on a straight line. There might also be theoretical reasons for assuming that the underlying function is *linear*, and the failure of the points to fall precisely on a straight line is due to experimental errors.

Assuming that the data can be described by the relation

$$y = ax + b$$

we want to determine the coefficients $a$ and $b$ such that the resulting line represents the data as well as possible. For given values of $a$ and $b$, the magnitude of the discrepancy or *error* between the $k$th data point and the line is

$$e = |ax_k + b - y_k|$$

The total absolute error for $m+1$ data points is

$$e_{tot} = \sum_{k=0}^{m} |ax_k + b - y_k|$$

This is a function of $a$ and $b$, and it would be reasonable to choose $a$ and $b$ so that this function assumes its minimum value.

In practice, it is common to minimize a different error function of $a$ and $b$:

$$\varphi(a,b) = \sum_{k=0}^{m} (ax_k + b - y_k)^2$$

This function is suitable because of statistical considerations.

---

If the errors of the data follow a *normal distribution*, then minimization of $\varphi(a,b)$ produces the best estimate of $a$ and $b$.

---

**Finding the minimum of φ**

The following conditions,

$$\frac{\partial\varphi}{\partial a} = 0 \qquad \frac{\partial\varphi}{\partial b} = 0$$

are *necessary* at the minimum of $\varphi(a,b)$. Taking the derivatives of φ gives

$$\begin{cases} \displaystyle\sum_{k=0}^{m} 2(ax_k + b - y_k)x_k = 0 \\ \displaystyle\sum_{k=0}^{m} 2(ax_k + b - y_k) = 0 \end{cases}$$

These are called the *normal equations* and can be written as

$$\begin{cases} \left(\displaystyle\sum_{k=0}^{m} x_k^2\right) a + \left(\displaystyle\sum_{k=0}^{m} x_k\right) b = \displaystyle\sum_{k=0}^{m} y_k x_k \\ \left(\displaystyle\sum_{k=0}^{m} x_k\right) a + (m+1)b = \displaystyle\sum_{k=0}^{m} y_k \end{cases}$$

The explicit solution of these equations is

$$a = \frac{1}{d}\left[(m+1)\sum_{k=0}^{m} y_k x_k - \left(\sum_{k=0}^{m} x_k\right)\left(\sum_{k=0}^{m} y_k\right)\right]$$

$$b = \frac{1}{d}\left[\left(\sum_{k=0}^{m} x_k^2\right)\left(\sum_{k=0}^{m} y_k\right) - \left(\sum_{k=0}^{m} x_k\right)\left(\sum_{k=0}^{m} y_k x_k\right)\right]$$

where

$$d = (m+1)\sum_{k=0}^{m} x_k^2 - \left(\sum_{k=0}^{m} x_k\right)^2$$

**Example.**

For the following table of data we get the following system of two equations:

| $x$ | $y$ |
|---|---|
| 0 | 68.1 |
| 10 | 67.0 |
| 20 | 66.5 |
| 30 | 65.7 |
| 40 | 64.4 |
| 80 | 61.7 |
| 90 | 61.1 |
| 95 | 60.3 |

$$\begin{cases} 26525a + 365b = 22710 \\ 365a + 8b = 514.8 \end{cases}$$

The solution is $a = -0.079$ and $b = 67.94$. The corresponding value of $\varphi(a,b)$ is 0.3 (in the case of a perfect fit, $\varphi(a,b) = 0$).

As a check, we can use Matlab to do the least squares fit and plot the resulting line together with the data points. The following commands can be used:

```
d = load('example1.dat');    %load data
p = polyfit(d(:,1),d(:,2),1) %linear least squares fit
x = 0:0.1:100;               % x range for plot
y = polyval(p,x);            % evaluation of polynomial
plot(d(:,1),d(:,2),'o',x,y)  % plot the fit and data points
xlabel('x')
ylabel('f(x)')
dd = p(1)*d(:,1) + p(2) - d(:,2);
phi = sum(dd.*dd);           % value of phi(a,b)
```

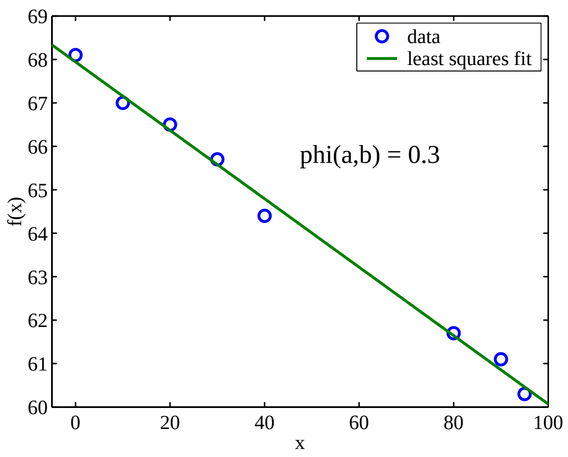

Figure 2: Data set and line fit.

## 2.2 Nonpolynomial example

The method of least squares is not restricted to polynomials or to any specific functional form. Suppose that we want to fit a table of values $(x_k, y_k)$, where $k = 0, 1, \ldots, m$, by a function of the form

$$f = a \ln x + b \cos x + c e^x$$

in the least-squares sense. The unknowns are the three coefficients $a$, $b$ and $c$.

We consider the function

$$\varphi(a,b,c) = \sum_{k=0}^{m} (a \ln x_k + b \cos x_k + c e^{x_k} - y_k)^2$$

and set $\partial\varphi/\partial a = 0$, $\partial\varphi/\partial b = 0$ and $\partial\varphi/\partial c = 0$. This gives us a system of three normal equations.

For the following table of data

| $x$ | $y$ |
|---|---|
| 0.24 | 0.23 |
| 0.65 | -0.26 |
| 0.95 | -1.10 |
| 1.24 | -0.45 |
| 1.73 | 0.27 |
| 2.01 | 0.10 |
| 2.23 | -0.29 |
| 2.52 | 0.24 |
| 2.77 | 0.56 |
| 2.99 | 1.00 |

the solution is $a = -1.04103$, $b = -1.26132$ and $c = 0.03073$. The corresponding value of $\varphi(a,b,c)$ is 0.92557.

We can use the following Matlab commands to carry out the fitting procedure and to plot the results:

```
d = load('example2.dat');      %load data
x = [log(d(:,1)) cos(d(:,1)) exp(d(:,1))];
z = x\d(:,2);                  %solution to YZ = X
dd = z(1)*log(d(:,1)) + z(2)*cos(d(:,1))...
     + z(3)*exp(d(:,1)) - d(:,2);
phi = sum(dd.*dd);             % value of phi(a,b,c)
xi = 0.24:0.1:2.99;            % x range for plot
yi = z(1)*log(xi) + z(2)*cos(xi)+ z(3)*exp(xi);
plot(d(:,1),d(:,2),'o',xi,yi)
xlabel('x')
ylabel('f(x)')
```

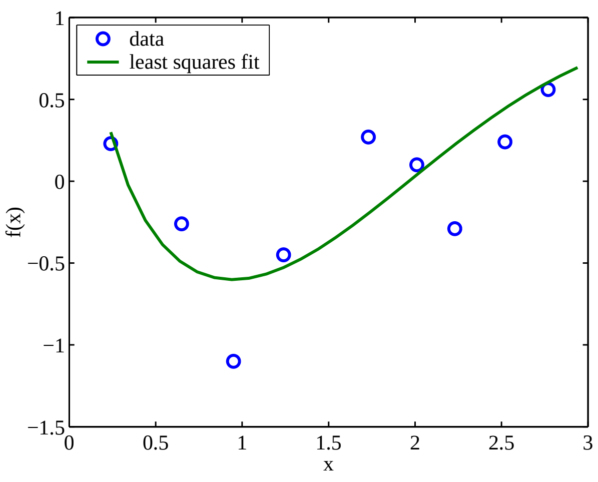

Figure 3: Data set and corresponding nonlinear least-squares fit.

## 2.3 Basis functions

The principle of least squares can be extended to general linear families of functions without involving any new ideas. Suppose that the data are thought to conform to relationship like

$$y = \sum_{j=0}^{n} c_j g_j(x)$$

where the *basis functions* $g_0, g_1, \ldots, g_n$ are known and held fixed. The coefficients $c_0, c_1, \ldots, c_n$ are to be determined according to the principle of least squares.

We define the expression

$$\varphi(c_0, c_1, \ldots, c_n) = \sum_{k=0}^{m} \left[ \sum_{j=0}^{n} c_j g_j(x_k) - y_k \right]^2$$

which is the sum of the squares of the errors associated with each entry $(x_k, y_k)$ in a given table of data. The coefficients are now selected such that this expression is minimized. We write down the following $n$ equations as the necessary conditions for the minimum:

$$\frac{\partial \varphi}{\partial c_i} = 0 \qquad (0 \le i \le n)$$

These partial derivatives are given by

$$\frac{\partial \varphi}{\partial c_i} = \sum_{k=0}^{m} 2 \left[ \sum_{j=0}^{n} c_j g_j(x_k) - y_k \right] g_i(x_k) \qquad (0 \le i \le n)$$

When these are set equal to zero, we obtain the *normal equations*:

$$\sum_{j=0}^{n} \left[ \sum_{k=0}^{m} g_i(x_k) g_j(x_k) \right] c_j = \sum_{k=0}^{m} y_k g_i(x_k) \qquad (0 \le i \le n)$$

The normal equations are linear in $c_0, c_1, \ldots, c_n$, and thus, in principle, they can be solved by the method of Gaussian elimination. In practice, care must be taken in choosing the basis functions $g_i$, since otherwise the normal equations may be very difficult to solve. The set $\{g_0, g_1, \ldots, g_n\}$ should be *linearly independent* which means that no linear combination $\sum_{i=0}^{n} c_i g_i$ can be the zero function.

The basis functions should also be appropriate for the problem at hand. Finally, the set of basis functions should be *well conditioned* for numerical work (see Chapter 5 for more details on ill-conditioned systems of linear equations).

## 3 Orthogonal systems and Chebyshev polynomials

### 3.1 Orthonormal basis functions

We concentrate now on the problem of choosing the set of basis functions $\{g_0, g_1, \ldots, g_n\}$ in order to describe the given data using the equation

$$y = \sum_{j=0}^{n} c_j g_j(x)$$

where the coefficients $c_0, c_1, \ldots, c_n$ are determined according to the principle of least squares.

Once the basis functions have been chosen, the least-squares problem can be interpreted as follows: The set of all functions $g$ that can be expressed as linear combinations of $g_0, g_1, \ldots, g_n$ is a vector space $\mathcal{G}$. In symbols,

$$\mathcal{G} = \left\{ g : \text{ there exist } c_0, c_1, \ldots, c_n \text{ such that } g(x) = \sum_{j=0}^{n} c_j g_j(x) \right\}$$

The function we are looking for in the least-squares problem is thus an element of the vector space $\mathcal{G}$.

The functions $g_0, g_1, \ldots, g_n$ form a *basis for the vector space* $\mathcal{G}$ if they are *not linearly dependent*. A given vector space can have many different bases which may differ drastically in their numerical properties.

**What basis should be chosen for numerical work?**

In the least-squares problem, the main task is to solve the normal equations:

$$\sum_{j=0}^{n} \left[ \sum_{k=0}^{m} g_i(x_k) g_j(x_k) \right] c_j = \sum_{k=0}^{m} y_k g_i(x_k) \qquad (0 \leq i \leq n)$$

The nature of this problem depends on the basis $\{g_0, g_1, \ldots, g_n\}$. We want to solve these equations easily or to be able to solve them accurately.

In an ideal situation, the basis $\{g_0, g_1, \ldots, g_n\}$ has the property of *orthonormality*:

$$\sum_{k=0}^{m} g_i(x_k) g_j(x_k) = \delta_{ij} = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases}$$

In this case, the coefficient matrix is the identity matrix and the normal equations simplify dramatically to

$$c_j = \sum_{k=0}^{m} y_k g_j(x_k) \qquad (0 \leq j \leq n)$$

which is an explicit formula for the coefficients $c_j$.

A procedure known as the *Gram-Schmidt process* can be used to obtain such an orthonormal basis. In certain situations, the effort is justified but often simpler procedures suffice. Next, we give a quick review of the Gram-Schmidt process and then introduce a simpler alternative.

## Gram-Schmidt process

In linear algebra, the Gram-Schmidt process is a method of orthogonalizing a set of vectors in an inner product space, most commonly the Euclidean space $\mathcal{R}_n$.

Orthogonalization in this context means the following: we start with vectors $v1, \ldots, vk$ which are linearly independent and we want to find mutually orthogonal vectors $u1, \ldots, uk$ which generate the same subspace as the vectors $v1, \ldots, vk$.

We denote the inner (dot) product of the two vectors $u$ and $v$ by $(u \cdot v)$. The Gram-Schmidt process works as follows:

$$\begin{aligned} u1 &= v1 \\ u2 &= v2 - [(u1 \cdot v2)/(u1 \cdot u1)]u1 \\ u3 &= v3 - [(u1 \cdot v3)/(u1 \cdot u1)]u1 - [(u2 \cdot v3)/(u2 \cdot u2)]u2 \\ &\vdots \\ uk &= vk - [(u1 \cdot vk)/(u1 \cdot u1)]u1 - [(u2 \cdot vk)/(u2 \cdot u2)]u2 - \ldots \\ &\quad - [(uk-1 \cdot vk)/(uk-1 \cdot uk-1)]uk-1 \end{aligned}$$

To check that these formulas work, first compute $(u1 \cdot u2)$ by substituting the above formula for u2: you will get zero. Then use this to compute $(u1 \cdot u3)$ again by substituting the formula for u3: you will get zero. The general proof proceeds by mathematical induction.

Geometrically, this method proceeds as follows: to compute $ui$, it projects $vi$ orthogonally onto the subspace $\mathcal{U}$ generated by $u1, \ldots, ui-1$, which is the same as the subspace generated by $v1, \ldots, vi-1$. $ui$ is then defined to be the difference between $vi$ and this projection, guaranteed to be orthogonal to all of the vectors in the subspace $\mathcal{U}$.

If one is interested in an orthonormal system $u1, \ldots, uk$ (i.e. the vectors are mutually orthogonal and all have norm 1), then one can divide $ui$ by its norm $(ui \cdot ui)$.

When performing orthogonalization on a computer, the **Householder transformation** is usually preferred over the Gram-Schmidt process since it is numerically more stable, i.e. rounding errors tend to have less serious effects. The Householder transformation is used in a frequently applied algorithm for QR decomposition which means decomposing a matrix $A$ into a much simpler form consisting of an orthogonal matrix $Q$ and an upper triangular matrix $R$. The QR decomposition can be used to easily solve the least-squares problem $Ax = b$.

## Polynomials as basis functions

We now discuss an alternative method for choosing the set of basis functions in such a way that they are suitable for numerical solution of the least-squares problem. Consider the vector space $\mathcal{G}$ that consists of all polynomials of degree $\leq n$. The simplest choice would be to use the following $n+1$ functions as a basis for $\mathcal{G}$:

$$g_0(x) = 1 \quad g_1(x) = x \quad g_2(x) = x^2 \quad \dots \quad g_n(x) = x^n$$

Using this basis, a typical element of the vector space $\mathcal{G}$ has the form

$$g(x) = \sum_{j=0}^{n} c_j g_j(x) = c_0 + c_1 x + c_2 x^2 + \ldots + c_n x^n$$

This basis, however, is almost always a *poor* choice for numerical work. The reason is that the simple polynomials $x^k$ are too much alike. It is difficult to determine the coefficients $c_j$ precisely for a function given as a linear combination of the polynomials $x^k$: $g(x) = \sum_{j=0}^{n} c_j x^j$. Figure 4 shows a plot of $x, x^2, x^3, x^4$ and $x^5$.

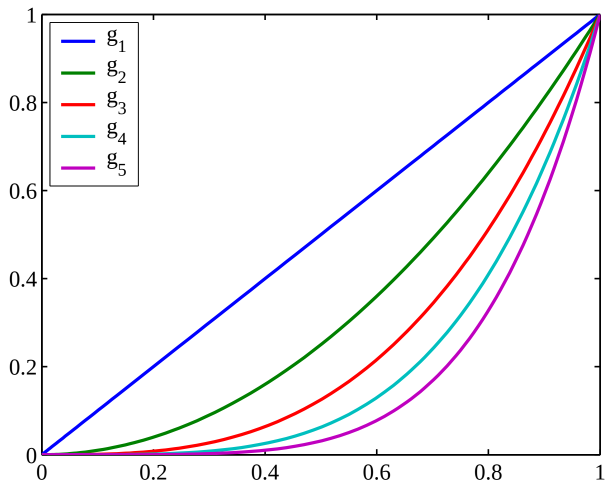

Figure 4: Polynomials $x, x^2, x^3, x^4$ and $x^5$.

## Chebyshev polynomials

For many purposes, a good basis can be obtained using the Chebyshev polynomials. For simplicity, assume that in our least-squares problem all the data points lie within the interval $[-1,1]$. The Chebyshev polynomials for the interval $[-1,1]$ are given by

$$\begin{cases} T_0(x) = 1 \qquad T_1(x) = x \qquad T_2(x) = 2x^2 - 1 \\ T_3(x) = 4x^3 - 3x \qquad T_4(x) = 8x^4 - 8x^2 + 1 \qquad \text{etc.} \end{cases}$$

The formal definition of the Chebyshev polynomials is given by the following recursive formula

$$T_j(x) = 2xT_{j-1}(x) - T_{j-2}(x) \qquad (j \geq 2)$$

with the equations $T_0(x) = 1$ and $T_1(x) = x$.

Alternatively, we can write

$$T_k(x) = \cos(k \arccos x)$$

Figure 5 shows a plot of $T_1, T_2, T_3, T_4$ and $T_5$. Note that these functions are quite different from one another and therefore better suited for numerical work.

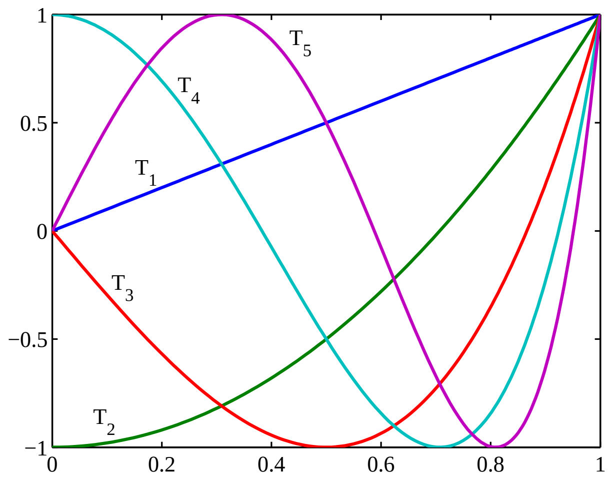

Figure 5: Chebyshev polynomials $T_1, T_2, T_3, T_4$ and $T_5$.

## Nested multiplication algorithm

Linear combinations of Chebyshev polynomials are easy to evaluate because a special nested multiplication algorithm applies. Consider an arbitrary linear combination of $T_0, T_1, T_2, \ldots, T_n$:

$$g(x) = \sum_{j=0}^{n} c_j T_j(x)$$

The following algorithm can be used to compute $g(x)$ for any $x$:

$$\begin{cases} w_{n+2} = w_{n+1} = 0 \\ w_j = c_j + 2xw_{j+1} - w_{j+2} \qquad (j = n, n-1, ..., 0) \\ g(x) = w_0 - xw_1 \end{cases}$$

To see that this really works, we can write down the series for $g(x)$:

$$\begin{aligned}
g(x) &= \sum_{j=0}^{n} c_j T_j(x) \\
&= \sum_{j=0}^{n} (w_j - 2xw_{j+1} + w_{j+2}) T_j \\
&= \sum_{j=0}^{n} w_j T_j - 2x \sum_{j=0}^{n} w_{j+1} T_j + \sum_{j=0}^{n} w_{j+2} T_j \\
&= \sum_{j=0}^{n} w_j T_j - 2x \sum_{j=1}^{n+1} w_j T_{j-1} + \sum_{j=2}^{n+2} w_j T_{j-2} \qquad (w_{n+2} = w_{n+1} = 0) \\
&= \sum_{j=0}^{n} w_j T_j - 2x \sum_{j=1}^{n} w_j T_{j-1} + \sum_{j=2}^{n} w_j T_{j-2} \\
&= w_0 T_0 + w_1 T_1 + \sum_{j=2}^{n} w_j T_j - 2xw_1 T_0 - 2x \sum_{j=2}^{n} w_j T_{j-1} + \sum_{j=2}^{n} w_j T_{j-2} \\
&\quad (T_0 = 1, T_1 = x) \\
&= w_0 + xw_1 - 2xw_1 + \sum_{j=2}^{n} w_j (T_j - 2xT_{j-1} + T_{j-2}) \\
&\quad (T_j = 2xT_{j-1} - T_{j-2}) \\
&= w_0 - xw_1
\end{aligned}$$

The normal equations should now be reasonably well conditioned if the first few Chebyshev polynomials are used to form the basis. This means that Gaussian elimination with pivoting produces an accurate solution to the normal equations.

If the original data do not satisfy $\min\{x_k\} = -1$ and $\max\{x_k\} = 1$ but lie instead in another interval $[a, b]$, then the change of variable

$$x = \frac{1}{2}(b-a)z + \frac{1}{2}(a+b)$$

produces a variable $z$ that traverses $[-1, 1]$ as $x$ traverses $[a, b]$.

## 3.2 Algorithm for polynomial fitting

The following procedure produces a polynomial of degree $\leq (n+1)$ that best fits a given table of values $(x_k, y_k)$ $(0 \leq k \leq m)$. Here $m$ is usually much greater than $n$. The Chebyshev polynomials are used as basis.

**Algorithm.**

1. Find the smallest interval $[a,b]$ that contains all the $x_k$.
   Let $a = \min\{x_k\}$ and $b = \max\{x_k\}$.
2. Make a transformation to the interval $[-1,1]$ by defining
$$z_k = \frac{2x_k - a - b}{b - a}$$
3. Decide the value of $n$ to be used (8 or 10 are large values in this situation).
4. Using Chebyshev polynomials as basis, generate the $(n+1) \times (n+1)$ normal equations (see below for details)
$$\sum_{j=0}^{n} \left[ \sum_{k=0}^{m} T_i(z_k) T_j(z_k) \right] c_j = \sum_{k=0}^{m} y_k T_i(z_k) \qquad (0 \leq i \leq n)$$
5. Use an equation-solving routine to solve the normal equations for the coefficients $c_0, c_1, \ldots, c_n$ in the function
$$g(x) = \sum_{j=0}^{n} c_j T_j(x)$$
6. The polynomial sought is
$$g\left(\frac{2x - a - b}{b - a}\right)$$

The details of step 4 are as follows: Begin by introducing a double-subscribed variable:

$$t_{jk} = T_j(z_k) \qquad (0 \le k \le m, 0 \le j \le n)$$

The matrix $\mathbf{T} = (t_{jk})$ can be computed efficiently by using the recursive definition of the Chebyshev polynomials as in the following pseudocode:

```
real array $(t_{ij})_{0:n\times 0:n}$, $(z_i)_{0:n}$
integer $j, k, m$
for $k = 0$ to $m$ do
    $t_{0k} \leftarrow 1$
    for $j = 2$ to $n$ do
        $t_{jk} \leftarrow 2z_k t_{j-1,k} - t_{j-2,k}$
    end for
end for
```

The normal equations have a coefficient matrix $\mathbf{A} = (a_{ij})_{0:n\times 0:n}$ and a right-hand side $\mathbf{b} = (b_i)_{0:n}$. These are given by

```
real array $(b_i)_{0:n}$, $(t_{ij})_{0:n\times 0:n}$, $(y_i)_{0:n}$
integer $i, j, m, n$
real s
⋮
for $i = 0$ to $n$ do
    $s \leftarrow 0$
    for $k = 0$ to $m$ do
        $s \leftarrow s + y_k t_{ik}$
    end for
    $b_i \leftarrow s$
    for $j = i$ to $n$ do
        $s \leftarrow 0$
        for $k = 0$ to $m$ do
            $s \leftarrow s + t_{ik} t_{jk}$
        end for
        $a_{ij} \leftarrow s$
        $a_{ji} \leftarrow s$
    end for
end for
```

## 3.3 Smoothing of data: Polynomial regression

An important application of the least-squares procedure is smoothing of data; i.e. *removing experimental errors* to whatever degree possible.

If it is known that the data should conform to a specific type of function (e.g. straight line, polynomial, exponential, etc.), then the least-squares procedure can be used to compute any unknown parameters in the function. On the other hand, if the specific form of the function is not the important factor but one only wishes to smooth the data by fitting them to any convenient function, then polynomials of increasing degree can be used to obtain a reasonable balance between a good fit and smoothness.

**Example.**

Suppose that we have the following set of 20 data points.

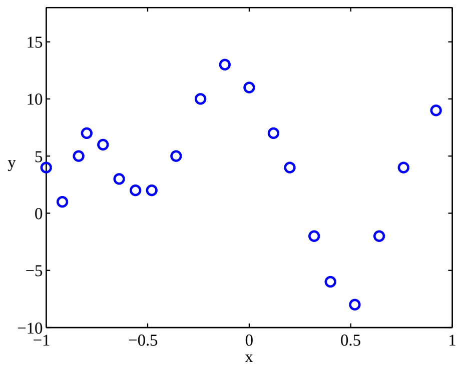

Figure 6: Example data set.

Of course, we can determine a polynomial of degree 19 that passes through all the data points. But if the data points contain experimental errors, then it is much more desirable to find a function that fits the data *approximately* in the least-squares sense. A lower order polynomial can be used for the fitting procedure. In statistical terms, this is called *curvilinear regression*.

Good software libraries contain a procedure for the polynomial fitting of empirical data in the least-squares sense. Such codes are highly optimized to give high precision with minimum computing effort.

For example, the following Matlab commands fit two polynomials of degree 8 and 13 to the given set of 20 data points.

```
d = load('reg_ex1.dat');        %load data
p8 = polyfit(d(:,1),d(:,2),8);  % polynomial of degree 8
p13 = polyfit(d(:,1),d(:,2),13);% polynomial of degree 13
x = -1:0.01:1;                  % x range for plot
y8 = polyval(p8,x);             % evaluation of polynomials
y13 = polyval(p13,x);
plot(d(:,1),d(:,2),'ok',x,y8,'-b',x,y13,'--r')
xlabel('x') ylabel('y')
axis([-1 1 -10 18])
```

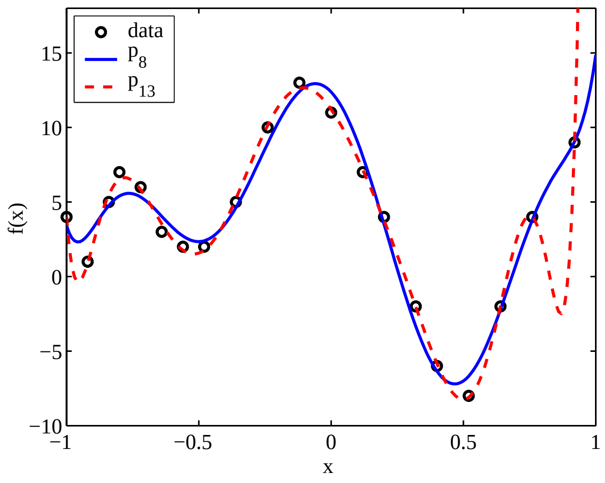

Figure 7: Polynomial fits of degree 8 and 13 to a set of 20 data points.

The lower-degree polynomial produces a much smoother fit without any oscillations. A similar fit can be obtained using the techniques which we have already discussed, e.g. using the Chebyshev polynomials as a basis, we can set up and solve the normal equations to obtain polynomial fits such as those shown in the figure above. This, however, is not a very well optimized approach.

**Polynomial regression**

An efficient procedure for polynomial regression is now explained. This method uses a system of orthogonal polynomials tailormade for the problem in question. We begin with a given table of experimental values:

| $x$ | $x_0$ | $x_1$ | $\dots$ | $x_m$ |
|---|---|---|---|---|
| $y$ | $y_0$ | $y_1$ | $\dots$ | $y_m$ |

The goal is to replace the table by a suitable polynomial of modest degree with the errors in the data suppressed. We do not know what degree of polynomial should be used.

Suppose that the trend of the table is represented by a polynomial

$$p_N(x) = \sum_{i=0}^{N} a_i x^i$$

and that the given tabular values obey the equation

$$y_i = p_N(x_i) + \varepsilon_i \qquad (0 \le i \le m)$$

Here $\varepsilon_i$ represents an observational error present in $y_i$. A further reasonable hypothesis is that these errors are independent random variables that are normally distributed.

We have already discussed a polynomial fitting procedure which can be used to obtain $p_n$ by the method of least squares. Using this approach, we can set up a system of normal equations to determine the coefficients of $p_n$. Once these are known, a quantity called the *variance* can be computed from the formula

$$\sigma_n^2 = \frac{1}{m-n} \sum_{i=0}^{m} [y_i - p_n(x_i)]^2 \qquad (m > n)$$

We know from statistics that if the trend of the table is truly a polynomial of degree $N$ (but infected by noise), then

$$\sigma_0^2 > \sigma_1^2 > \ldots > \sigma_N^2 = \sigma_{N+1}^2 = \sigma_{N+2}^2 = \ldots = \sigma_{m-1}^2$$

For the case in which $N$ is not known, the following strategy can be used: Compute $\sigma_0^2, \sigma_1^2, \ldots$ in succession. As long as these are descending, continue the calculation. When an integer $N$ is reached for which $\sigma_N^2 \approx \sigma_{N+1}^2 \approx \sigma_{N+2}^2 \approx \ldots$, stop and declare $p_N$ to be the polynomial sought.

If we were to compute $\sigma_0^2, \sigma_1^2, \ldots$ directly from the definition of the variance given above, this would involve determining all the polynomials $p_0, p_1, \ldots$. Instead, we can use a more intelligent procedure which avoids determination of all but the one desired polynomial. This procedure is described next.

**Inner product**

We begin by a definition of a quantity called the *inner product* of two functions $f$ and $g$. Assume in the following that the abscissas $x_i$ are distinct and held fixed. If $f$ and $g$ are two functions whose domains include the points $\{x_0, x_1, \ldots, x_m\}$, then the following notation is used for the inner product of $f$ and $g$:

$$\langle f, g \rangle = \sum_{i=0}^{m} f(x_i) g(x_i)$$

Much of the discussion does not depend on the exact form of the inner product but only on its certain properties.

An inner product $\langle\ ,\ \rangle$ has the following properties:

1. $\langle f, g \rangle = \langle g, f \rangle$.
2. $\langle f, f \rangle > 0$ unless $f(x_i) = 0$ for all $i$.
3. $\langle af, g \rangle = a\langle f, g \rangle$ for any $a \in \mathbb{R}$.
4. $\langle f, g + h \rangle = \langle f, g \rangle + \langle f, h \rangle$

A set of functions is said to be *orthogonal* if $\langle f, g \rangle = 0$ for any two different functions $f$ and $g$ in that set.

**Recursive formula for generating an orthogonal set of polynomials**

An orthogonal set of polynomials can be generated recursively by the following formulas:

$$\begin{cases} q_0(x) = 1 \\ q_1(x) = x - \alpha_0 \\ q_{n+1}(x) = xq_n(x) - \alpha_n q_n(x) - \beta_n q_{n-1}(x) \qquad (n \geq 1) \end{cases}$$

where

$$\begin{cases} \alpha_n = \dfrac{\langle xq_n, q_n \rangle}{\langle q_n, q_n \rangle} \\ \\ \beta_n = \dfrac{\langle xq_n, q_{n-1} \rangle}{\langle q_{n-1}, q_{n-1} \rangle} \end{cases}$$

Here $xq_n$ denotes the function whose value at $x$ is $xq_n(x)$.

*Proof.* In order to prove that the definition above leads to an orthogonal system, we can first consider consider the following case,

$$\langle q_1, q_0 \rangle = \langle x - \alpha_0, q_0 \rangle = \langle xq_0 - \alpha_0 q_0, q_0 \rangle = \langle xq_0, q_0 \rangle - \alpha_0 \langle q_0, q_0 \rangle = 0$$

Another similar case is

$$\langle q_2, q_1 \rangle = \langle xq_1 - \alpha_1 q_1 - \beta_1 q_0, q_1 \rangle = \langle xq_1, q_1 \rangle - \alpha_1 \langle q_1, q_1 \rangle - \beta_1 \langle q_0, q_1 \rangle = 0$$

The next step of the proof would be to verify that $\langle q_2, q_0 \rangle = 0$. Finally an inductive argument can be used to complete the proof. (These two remaining steps are left as an exercise for the student).

**Linear combination of orthogonal polynomials**

The system of orthogonal polynomials $\{q_0, q_1, \ldots, q_{m-1}\}$ is a basis for the vector space $\prod_{m-1}$ of all polynomials of degree at most $m-1$. If it is desired to express a given polynomial $p_n$ $(n \leq m-1)$ as a linear combination of $\{q_0, q_1, \ldots, q_n\}$, this can be done as follows.

Set

$$p = \sum_{i=0}^{n} a_i q_i$$

Only one of the summands on the right-hand side, namely $a_n q_n$, contains the highest term $x^n$. The coefficient $a_n$ is chosen in such a way that $a_n x^n$ on the right is equal to a corresponding term in $p$. We write

$$p - a_n x^n = \sum_{i=0}^{n-1} a_i q_i$$

We can now choose $a_{n-1}$ the same way: making the $x^{n-1}$ terms equal on both sides. By continuing this way, we can discover the unique values for all the coefficients $a_i$. This establishes that $\{q_0, q_1, \ldots, q_n\}$ is a basis for $\prod_n$ for $n = 0, 1, 2, \ldots, m-1$.

*Another way of determining the coefficients $a_i$* is to take the inner product of both sides in the equation for the given polynomial. This gives

$$\langle p, q_j \rangle = \sum_{i=0}^{n} a_i \langle q_i, q_j \rangle \qquad (0 \le j \le n)$$

Since the set $q_0, q_1, \ldots, q_n$ is orthogonal, $\langle q_i, q_j \rangle = 0$ for $i \neq j$. Hence,

$$\langle p, q_j \rangle = a_j \langle q_j, q_j \rangle$$

and consequently

$$a_j = \frac{\langle p, q_j \rangle}{\langle q_j, q_j \rangle}$$

**Solution of the least-squares problem**

Now we return to the least-squares problem. Let $F$ be a function that we wish to fit by a polynomial $p_n$ of degree $n$. According to the principle of least squares, we want to find the $p_n$ that minimizes the expression

$$\sum_{i=0}^{m} [F(x_i) - p_n(x_i)]^2$$

The solution is given by

$$p_n(x) = \sum_{i=0}^{n} c_i q_i \qquad c_i = \frac{\langle F, q_i \rangle}{\langle q_i, q_i \rangle}$$

Note especially that $c_i$ does not depend on $n$. This means that the various polynomials $p_0, p_1, \ldots, ..$ can be obtained by *truncating one series*, namely $\sum_{i=0}^{m-1} c_i q_i$.

*Proof.* To see that the solution given above really is the correct solution to our problem, we return to the normal equations. The basis functions are now $q_0, q_1, \ldots, q_n$ and the normal equations are given by

$$\sum_{j=0}^{n} \left[ \sum_{k=0}^{m} q_i(x_k) q_j(x_k) \right] c_j = \sum_{k=0}^{m} y_k q_i(x_k) \qquad (0 \le i \le n)$$

Using the inner product notation, we get

$$\sum_{j=0}^{n} \langle q_i, q_j \rangle c_j = \langle F, q_i \rangle \qquad (0 \le i \le n)$$

where $F$ is some function such that $F(x_k) = y_k$ for $0 \le k \le m$ (these are the given data points).

Next, the orthogonality property, $\langle q_i, q_j \rangle = 0$ when $i \neq j$, is applied. This gives

$$\langle q_i, q_i \rangle c_i = \langle F, q_i \rangle \qquad (0 \le i \le n)$$

Thus we get

$$c_i = \frac{\langle F, q_i \rangle}{\langle q_i, q_i \rangle}$$

which is the same as given above, and thus the solution is correct.

**Algorithm for calculating the variances**

Now we return to the variance numbers $\sigma_0^2, \sigma_1^2, \ldots$. Remember that our task was to find an efficient procedure for calculating these in a successive manner until an integer $N$ is reached for which $\sigma_N^2 \approx \sigma_{N+1}^2 \approx \sigma_{N+2}^2 \approx \ldots$. $p_N$ is then the polynomial we are seeking.

The variance numbers can in fact be computed easily. First, an observation: The set $\{q_0, q_1, \ldots, q_n, F - p_n\}$ is *orthogonal*. To check this, verify that $\langle F - p_n, q_i \rangle = 0$ for $0 \leq i \leq n$. Since $p_n$ is a linear combination of $q_0, q_1, \ldots, q_n$, it follows that

$$\langle F - p_n, p_n \rangle = 0$$

Now recall the definition of the variance $\sigma_n^2$:

$$\sigma_n^2 = \frac{\rho_n}{m-n} \qquad \rho_n = \sum_{i=0}^{m} [y_i - p_n(x_i)]^2$$

These quantities $\rho_n$ can be written in another way, using $F(x_i) = y_i$ and the definition of the inner product:

$$\begin{aligned}
\rho_n &= \sum_{i=0}^{m} [y_i - p_n(x_i)]^2 \\
&= \langle F - p_n, F - p_n \rangle \\
&= \langle F - p_n, F \rangle \qquad\qquad [\text{since} \langle F - p_n, p_n \rangle = 0] \\
&= \langle F, F \rangle - \langle F, p_n \rangle \\
&= \langle F, F \rangle - \sum_{i=0}^{n} c_i \langle F, q_i \rangle \\
&= \langle F, F \rangle - \sum_{i=0}^{n} \frac{\langle F, q_i \rangle^2}{\langle q_i, q_i \rangle}
\end{aligned}$$

The numbers $\rho_0, \rho_1, \ldots$ can now be generated *recursively* by the algorithm

$$\begin{cases}
\rho_0 = \langle F, F \rangle - \dfrac{\langle F, q_0 \rangle^2}{\langle q_0, q_0 \rangle} \\
\rho_n = \rho_{n-1} - \dfrac{\langle F, q_n \rangle^2}{\langle q_n, q_n \rangle} \qquad (n \geq 1)
\end{cases}$$

## 3.4 Summary: Algorithm for polynomial regression

We now have all the necessary ingredients for constructing an efficient algorithm for polynomial regression. As an input, we need a table of data values $\{x_i, F(x_i)\}$ $(0 \leq i \leq m)$ which are assumed to contain experimental errors. The underlying trend of the data is assumed to conform to some functional form. The objective of polynomial regression is to reduce the errors by fitting the data with a polynomial of modest degree. Thus the most important assumption we are making is that the underlying function can be approximated well by a polynomial.

**Constructing a polynomial regression procedure:**

- Write a procedure for calculating inner products. The following segment of pseudocode can be used for this:

```
real array $f_{0:m}$, $g_{0:m}$
sum ← 0
for $i = 0$ to $m$ do
   sum ← $f_i g_i$
end for
```

- Input to the polynomial regression procedure:
  $m$ = number of data points
  $(x_{0:m})$ = array containing the x values of data
  $(F_{0:m})$ = array containing the y values of data
  $(\alpha_{0:m})$ = array for the $\alpha$ values for the basis polynomials
  $(\beta_{0:m})$ = array for the $\beta$ values for the basis polynomials
  $(c_{0:m})$ = array for the coefficients of the sought polynomial $p_n$

- Allocate memory for the following arrays:
  $(\sigma^2_{0:m})$ = values of the variances $\sigma^2_i$
  $(qOld_{0:m})$ = values of the basis polynomial $q_{i-2}$
  $(qPrev_{0:m})$ = values of the basis polynomial $q_{i-1}$
  $(qCurr_{0:m})$ = values of the basis polynomial $q_i$
  $(xq_{0:m})$ = help array for values of $x * q_i(x)$
  (useful in C, not needed in Fortran)

- Two variables are needed for values of $\rho_i$ and $\rho_{i-1}$.

- Calculate the following initial values:
  Basis functions $q_0$ and $q_1$ (stored in arrays $(qOld)$ and $(qPrev)$)
  Value of $\alpha_0$
  Values of $\rho_0$ and $\rho_1$ Values of $\sigma^2_0$ and $\sigma^2_1$
  Values of $c_0$ and $c_1$.

- Set $n = 1$ (degree of polynomial).
- Do a loop to calculate $\rho_2, \rho_3, \ldots$ in succession to obtain $\sigma_2^2, \sigma_3^2, \ldots$. Continue until $\sigma_{n-1}^2 \approx \sigma_n^2$.
- In practice, use two stop criteria within the loop:
  if $\sigma_n^2 > \sigma_{n-1}^2$ or
  if $|\sigma_{n-1}^2 - \sigma_n^2| < \varepsilon$ (where e.g. $\varepsilon = 0.1$).

- In the end, print out the values of $\alpha_i$, $\beta_i$, $\sigma_i^2$ and $c_i$ for $0 \leq i \leq n$.
- Finally, the polynomial $p_n$ can be evaluated at any point $x = t$ in the interval from $min\{x_i\}$ to $max\{x_i\}$ by the formula

  $$p_n(t) = \sum_{i=0}^{n} c_i q_i(t)$$

  where the coefficients $c_i$ are now known.

**Example: Polynomial regression**

Suppose that we have the following set of 20 data points. The trend of data suggests that the underlying function could be a polynomial.

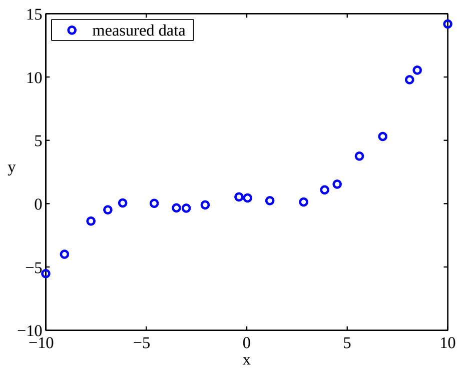

Figure 8: Example data set.

Using a polynomial regression program, we can try to smooth the data by fitting a low-order polynomial. The assumption which we make here is that the fluctuations are indeed due to experimental errors and not true variations of the underlying function.

The program gives the following output:

```
i  alpha     beta      variance   c
0 -0.105193  0.000000 22.009486  1.772073
1  0.005345 36.715295  5.943165  0.667597
2 -0.138884 28.731086  3.722647  0.046650
3  0.195848 28.251617  0.319753  0.010164
```

Thus the resulting polynomial $p_n$ is of degree three ($n = 3$).

Figure 9 shows a graph of the data and the polynomial $p_3$ evaluated at 100 points in the interval $[-10, 10]$.

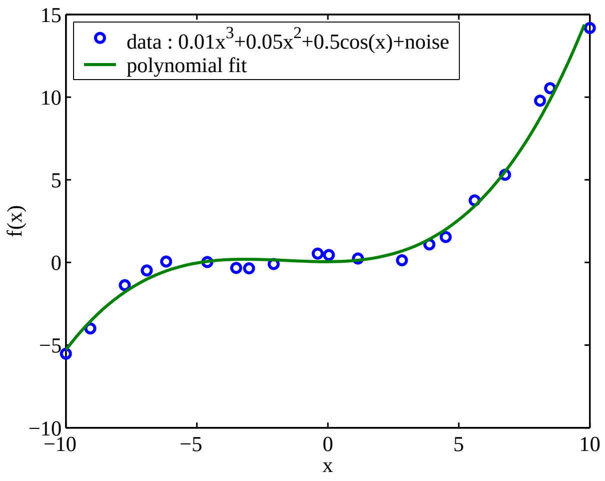

Figure 9: Data points and the polynomial $p_3$ obtained by polynomial regression.

The polynomial seems to be a nice smooth fit which represents the data well. However, if the small fluctuations in the data points are *not* just random noise but due to the true form of the underlying function, then these features are lost in the fitting procedure. Figure 10 shows what the graph of the true underlying function looks like (the data points were obtained by taking the values of this function at various points and adding a small noise term).

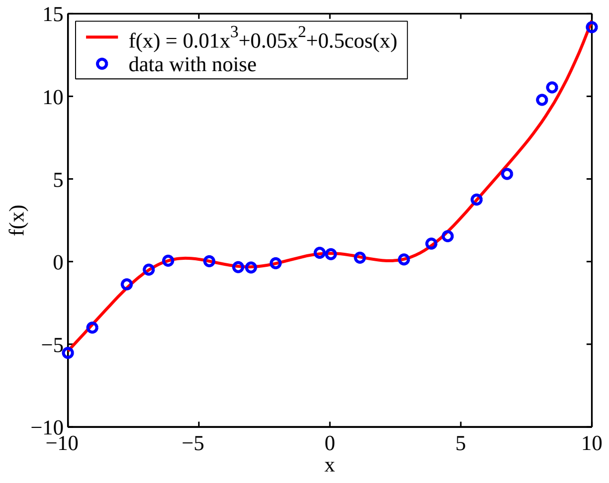

Figure 10: The true underlying function and the data points.

### Example 2: What can go wrong

In the previous example, the overall trend of the data could be described well by a third-order polynomial. The governing assumption is that all deviations in the data points are due to experimental errors (random noise). If the underlying function is in fact more complicated and cannot be well approximated by a low-order polynomial, then the smoothing of the data may result in an approximation which is an oversimplification of the underlying function. (E.g. in the previous example, the oscillations due to the cosine term were lost in the fitting procedure).

The polynomial regression procedure may also fail in the attempt to find a smooth polynomial which would give a good approximation of the data. Figure 11 shows an example of such a case: The trend of the data resembles that of a third order polynomial but the regression program has given a linear fit.

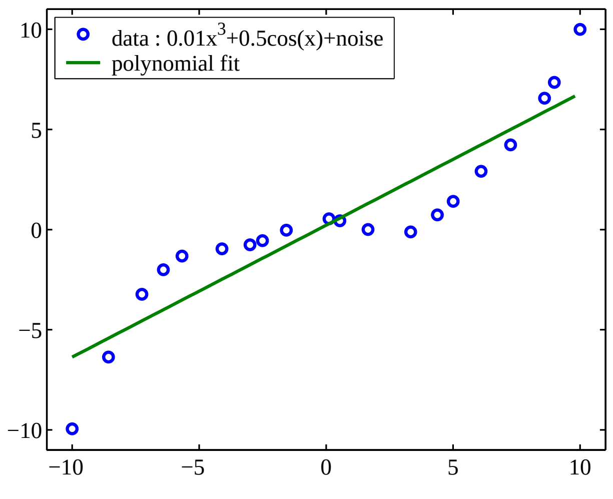

Figure 11: Example of a problematic case.

The reason is that a second-order polynomial does not approximate the data any better (variances are almost equal). So the program stops and does not continue to higher order polynomials. In such a case, we can of course modify the code by hand and calculate the fit using higher order polynomials. We notice that for $n = 3$ we obtain an excellent fit.

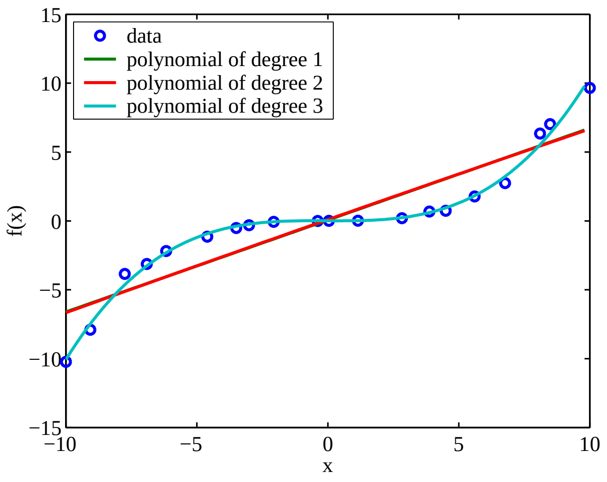

Figure 12: Polynomial fit with $n = 3$.

The corresponding output values are:

```
i  alpha     beta      variance  c
0 -0.105193  0.000000 19.634211 -0.008996
1  0.005345 36.715295  3.484323  0.666793
2 -0.138884 28.731086  3.676964 -0.000892
```

The first and second degree fits are almost equal (the second degree polynomial is very close to a straight line). Figure 13 shows the difference between the first- and second-order polynomials.

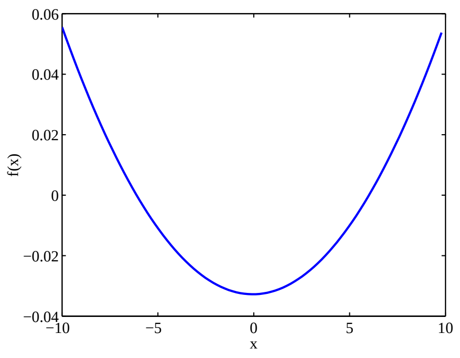

Figure 13: Difference between the first- and second-order polynomials.

# 4 Other examples of the least-squares principle

## 4.1 Solving inconsistent systems of linear equations

As another type of an application of the least-squares principle, we can consider solving an inconsistent system of linear equations of the form

$$\sum_{j=0}^{n} a_{kj}x_j = b_k \qquad (0 \ \leq k \leq m)$$

in which $m > n$. Thus we have $m+1$ equations but only $n+1$ unknowns. In matrix form, the system is written as

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} & \cdots & a_{0n} \\ a_{10} & a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{20} & a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{30} & a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & & \vdots & \\ a_{n0} & a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \\ \vdots & \vdots & \vdots & & \vdots & \\ a_{m0} & a_{m1} & a_{m2} & a_{m3} & \cdots & a_{mn} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_n \\ \vdots \\ b_m \end{bmatrix}$$

If a given set of values $(x_0, x_1, \ldots, x_n)$ is substituted into the equations, we obtain a residual vector. The $k$th element of the vector is termed the *kth residual* and it describes the discrepancy between the two sides of the $k$th equation. Ideally, all the residuals should be zero.

If it is not possible to select $(x_0, x_1, \ldots, x_n)$ such that all the residuals are zero, the system is called *inconsistent* or *incompatible*. In this case, an alternative is to minimize the residuals. This means minimizing the expression

$$\varphi(x_0, x_1, \ldots, x_n) = \sum_{k=0}^{m} \left( \sum_{j=0}^{n} a_{kj}x_j - b_k \right)^2$$

by making an appropriate choice of $(x_0, x_1, \ldots, x_n)$.

We take the partial derivatives with respect to $x_i$ and set them equal to zero. This gives us the normal equations

$$\sum_{j=0}^{n} \left( \sum_{k=0}^{m} a_{ki}a_{kj} \right) x_j = \sum_{k=0}^{m} b_k a_{ki} \qquad (0 \leq i \leq n)$$

This is a linear system of $n+1$ equations involving $n+1$ unknowns
$x_0, x_1, \ldots, x_n$. It can be shown that this system is consistent, provided that the column vectors in the original coefficient array are linearly independent. This system can be solved, for example, by Gaussian elimination. The solution is then an approximate solution of the original system in the least-squares sense.

## 4.2 Weight function

Another important example of different application ways of the least-squares principle occurs in fitting or approximating functions on *intervals* rather than discrete sets.

For example, a given function $f$ defined on an interval $[a,b]$ may have to be approximated by a function like

$$g(x) = \sum_{j=0}^{n} c_j g_j(x)$$

In such a case, we attempt to minimize the expression

$$\varphi(c_0, c_1, \ldots, c_n) = \int_a^b [g(x) - f(x)]^2 dx$$

by choosing the coefficients appropriately.

In some applications, it is desirable to force the functions $f$ and $g$ into better agreement on certain parts of the interval than elsewhere. For this purpose, we can include a positive *weight function* $w(x)$ which can emphasize certain parts of the interval. If all parts are equally important, then $w(x) = 1$. The resulting expression is

$$\varphi(c_0, c_1, \ldots, c_n) = \int_a^b [g(x) - f(x)]^2 w(x) dx$$

The minimum of this expression is again sought by differentiating with respect to $c_i$. The result is the following system of normal equations

$$\sum_{j=0}^{n} \left[ \int_a^b g_i(x) g_j(x) w(x) dx \right] c_j = \int_a^b f(x) g_i(x) w(x) dx \qquad (0 \leq i \leq n)$$

This is a system of $n+1$ equations involving $n+1$ unknowns $c_0, c_1, \ldots, c_n$ and it can be solved by Gaussian elimination.

## 4.3 Nonlinear least squares

As an example of how the least-squares principle is applied in a nonlinear case, suppose that a table of points $(x_k, y_k)$ is to be fitted by a function of the form

$$y = e^{cx}$$

The least-squares approach leads to the minimization of the function

$$\phi(c) = \sum_{k=0}^{m} (e^{cx_k} - y_k)^2$$

Differentiating with respect to $c$ gives

$$\frac{\partial \phi}{\partial c} = \sum_{k=0}^{m} 2(e^{cx_k} - y_k) e^{cx_k} x_k = 0$$

This equation is *nonlinear* in $c$. We could consider using a root-finding method such as the Newton's method or the secant method (introduced in Ch.3). However, this is not a very safe approach because there can be multiple roots in the normal equation. On the other hand, we could think about a direct minimization of the function $\phi$, but this may also turn out to be difficult if there are local minima in $\phi$.

These difficulties are typical in nonlinear least-squares problems. Consequently, other methods of curve fitting are often preferred if the unknown parameters do not occur linearly in the problem.

In this particular case, we can linearize the problem by a change of variables $z = \ln y$ and by fitting to a form

$$z = cx$$

The minimization task involves now the function

$$\phi(c) = \sum_{k=0}^{m} (cx_k - z_k)^2 \qquad (z_k = \ln y_k)$$

This is easily solved and leads to

$$c = \frac{\sum_{k=0}^{m} z_k x_k}{\sum_{k=0}^{m} x_k^2}$$

Note that this value of $c$ is *not* the solution of the original least-squares problem but it may be satisfactory, depending on the application.

**Example.**

Figure 14 shows a set of data which seems to follow a relation of the form

$$y = e^{cx}$$

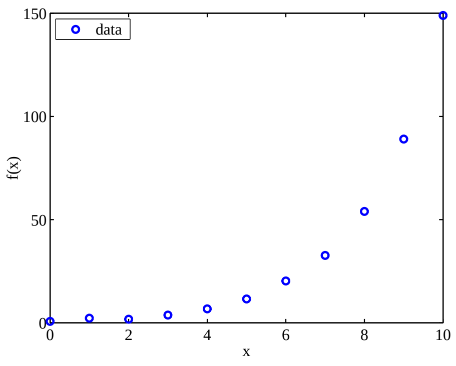

Figure 14: Example data set.

Let us try to linearize the problem by setting $z_i = \ln(y_i)$ and using the least-squares method to fit a straight line through the set $\{x_i, z_i\}$.

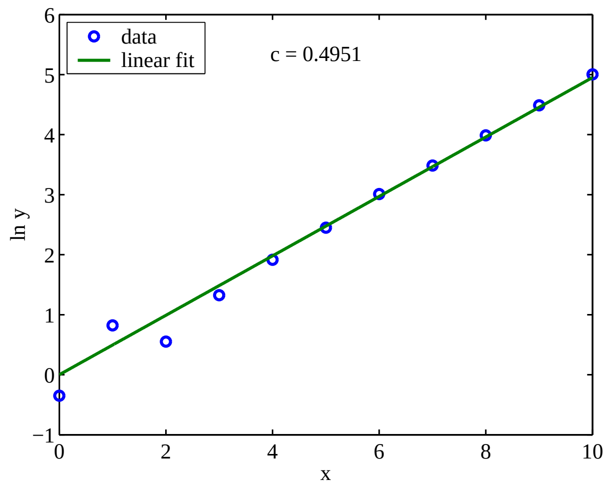

Figure 15: Linearized fit.

The result is shown in Figure 15. The slope of the line is $c = 0.4951$.

This problem is sufficiently simple so that we can also use a nonlinear least-squares method to solve the original problem directly. For example, the Matlab command (nlinfit) can be used for a nonlinear least-squares fit.

Figure 16 shows a comparison of the two fits, one obtained by linearizing the problem and the other by solving the nonlinear problem directly. The two values of the unknown coefficient are: $c_1 = 0.4951$ (from linearization) and $c_2 = 0.4999$ (nonlinear).

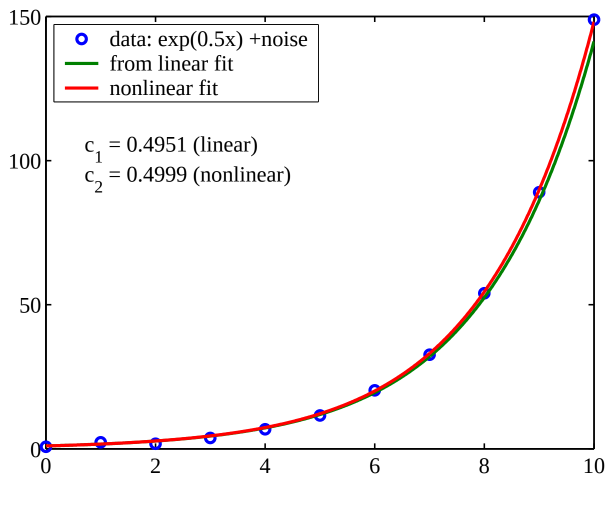

Figure 16: Comparison of the solutions obtained by linearization and a direct nonlinear fit.

The difference between the two curves is not very large so for many applications the solution of the linearized model is sufficient.

Note that the data points are obtained by taking 11 equidistant points from the curve $f(x) = \exp(0.5x)$ and adding random noise (distributed uniformly in $[-1, 1]$). Without any noise in the data, the linearized solution gives exactly $c = 0.5$.

## 4.4 Linear / nonlinear least squares

As a final example, we consider a case which combines elements of linear and nonlinear theory. Suppose that the given data seems to follow a relationship like

$$y = a\sin(bx)$$

Can the least-squares principle be used in this case to find the appropriate values of $a$ and $b$?

Here the parameter $b$ enters the relationship in a nonlinear way which creates some difficulty. According to the principle of least squares, the parameters should be chosen such that they minimize the function

$$\phi(a,b) = \sum_{k=0}^{m} [a\sin(bx_k) - y_k]^2$$

The minimum is again sought by differentiating this expression with respect to $a$ and $b$, thus giving

$$\begin{cases} \sum_{k=0}^{m} 2[a\sin(bx_k) - y_k]\sin(bx_k) & = 0 \\ \sum_{k=0}^{m} 2[a\sin(bx_k) - y_k]ax_k\cos(bx_k) & = 0 \end{cases}$$

If we knew $b$, then $a$ could be obtained easily from either of these equations. The correct value of $b$ is such that the corresponding two values of $a$ obtained from these two equations are equal. So we should solve both of these equations with respect to $a$ and set the results equal. This gives

$$\frac{\sum_{k=0}^{m} y_k \sin bx_k}{\sum_{k=0}^{m} (\sin bx_k)^2} = \frac{\sum_{k=0}^{m} x_k y_k \cos bx_k}{\sum_{k=0}^{m} x_k \sin bx_k \, \cos bx_k}$$

This can now be solved for $b$ using, for example, the bisection or the secant method. Then either side of this equation can be evaluated as $a$.