# Lecture 20

## *Curve fitting II*

## Introduction

In the previous lecture we developed a method to solve the general linear least-squares problem. Given *n* samples $(x_i, y_i)$, the coefficients $c_j$ of a model

$$y=f(x)=\sum_{j=1}^{m} c_j f_j(x) \tag{1}$$

are found which minimize the mean-squared error

$$MSE=\frac{1}{n}\sum_{i=1}^{n}\left[y_i-f(x_i)\right]^2 \tag{2}$$

The *MSE* is a quadratic function of the $c_j$ and best-fit coefficients are the solution to a system of linear equations.

In this lecture we consider the *non-linear least-squares* problem. We have a model of the form

$$y=f(x;c_1,c_2,\ldots,c_m) \tag{3}$$

where the $c_j$ are general *parameters* of the function *f*, not necessarily coefficients. An example is fitting an arbitrary sine wave to data where the model is

$$y=f(x;c_1,c_2,c_3)=c_1\sin(c_2 x+c_3)$$

The mean-squared error

$$MSE(c_1,\ldots,c_m)=\frac{1}{n}\sum_{i=1}^{n}\left[y_i-f(x_i;c_1,\ldots,c_m)\right]^2 \tag{4}$$

will no longer be a quadratic function of the $c_j$, and the best-fit $c_j$ will no longer be given as the solutions of a linear system. Before we consider this general case, however, let's look at a special situation in which a non-linear model can be "linearized."

## Linearization

In some cases it is possible to transform a nonlinear problem into a linear problem. For example, the model

$$y=c_1 e^{c_2 x} \tag{5}$$

is nonlinear in parameter $c_2$. However, taking the logarithm of both sides gives us

$$\ln y=\ln c_1+c_2 x \tag{6}$$

If we define $\hat{y}=\ln y$ and $\hat{c}_1=\ln c_1$ then our model has the linear form

$$\hat{y}=\hat{c}_1+c_2 x \tag{7}$$

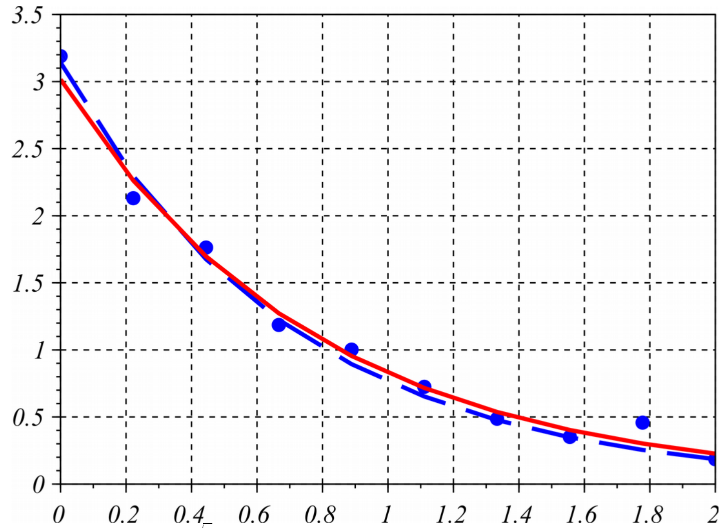


*Fig. 1: Dashed line:* $y=\pi e^{-\sqrt{2}x}$ *. Dots: ten samples with added noise. Solid line: fit of the model* $y=c_1 e^{c_2 x}$ *obtained by fitting a linear model* $\ln y=\hat{c}_1+c_2 x$ *and then calculating* $c_1=e^{\hat{c}_1}$ .

Once we've solved for $\hat{c}_1, c_2$ we can calculate $c_1=e^{\hat{c}_1}$ .

*Example*. Noise was added to ten samples of $y=\pi e^{-\sqrt{2}x}$ , $0\le x\le 2$ . The following code computed the fit of the linearized model.

```
ylog = log(y);
a = (mean(x.*ylog)-mean(x)*mean(ylog))/(mean(x.^2)-mean(x)^2);
b = mean(ylog)-a*mean(x);
c1 = exp(b);
c2 = a;
disp([c1,c2]);

3.0157453  - 1.2939429
```

# Nonlinear least squares

The general least-squares problem is to find the $c_j$ that minimize

$$MSE(c_1,\ldots,c_m)=\frac{1}{n}\sum_{i=1}^{n}\left[y_i-f(x_i;c_1,\ldots,c_m)\right]^2 \tag{8}$$

This is simply the "optimization in *n* dimensions" problem that we dealt with in a previous lecture. We can use any of those techniques, such as Powell's method, to solve this problem. It is convenient, however, to have a "front end" function that forms the *MSE* given the data $(x_i, y_i)$ and the function $f(x;c_1,c_2,\ldots,c_m)$ and passes that to our minimization routine of choice. The function `fitLeastSquares` in the Appendix is an example of such a front end.

The following example illustrates the use of `fitLeastSquares`.

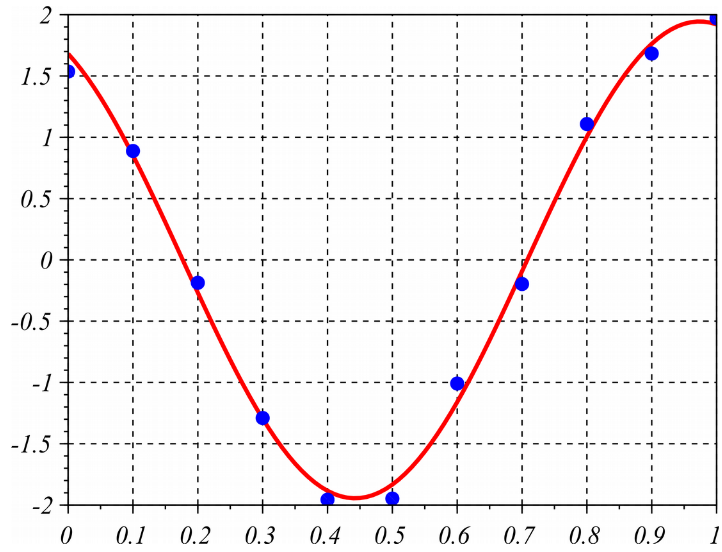


*Fig. 2: Fit of the model* $y=c_1\cos(c_2 x+c_3)$ *to 11 data points.*

*Example.* Eleven data samples in the interval $0 \le x \le 1$ of the function $y=2\cos(6x+0.5)$ were generated. Normally distributed noise with standard deviation 0.1 was added to the *y* data. A fit of the model $y=c_1\cos(c_2 x+c_3)$ gave $c_1=1.94$, $c_2=5.91$, $c_3=0.53$. The fit is shown in Fig. 2 and was generated with the following code.

```
rand('seed',2);
x = [0:0.1:1]';
y = 2*cos(6*x+0.5)+rand(x,'normal')*0.1;
c0 = [1;5;0];

function yf = fMod(x,c)
  yf = c(1)*cos(c(2)*x+c(3));
endfunction

[c,fctMin] = fitLeastSquares(x,y,fMod,c0,0.01,1e-6);
disp(c);

    1.9846917
    5.8614475
    0.5453276
```

# The lsqrsolve (Scilab) and lsqcurvefit (Matlab) functions

The Scilab function `lsqrsolve` solves general least-squares problems. We create a function to calculate the residuals.

The following code solves the problem in the previous example using `lsqrsolve`.

```
rand('seed',2);
x = [0:0.1:1]';
y = 2*cos(6*x+0.5)+rand(x,'normal')*0.1;

function r = residuals(c,m)
  r = zeros(m,1);
  for i=1:m
    r(i) = y(i)-c(1)*cos(c(2)*x(i)+c(3));
  end
endfunction

c0 = [1;5;0];
c = lsqrsolve(c0,residuals,length(x));
disp(c);
```

In Matlab the function `lsqcurvefit` can be used to implement a least-squares fit. The first step is to create a file specifying the model function in terms of the parameter vector `c` and the `x` data. In this example the file is named `fMod.m`

```
function yMod = fMod(c,x)
yMod = c(1)*cos(c(2)*x+c(3));
```

Then, in the main program we pass the function `fMod` as the first argument to `lsqcurvefit`, along with the initial estimate of the parameter vector `c0` and the `x` and `y` data.

```
x = [0:0.1:1]';
y = 2*cos(6*x+0.5)+randn(size(x))*0.1;
c0 = [1;5;0];
c = lsqcurvefit(@fMod,c0,x,y);
disp(c);
```

# Appendix – Scilab code

```
//////////////////////////////////////////////////////////////////////
// fitLeastSquares.sci
// 2014-11-11, Scott Hudson, for pedagogic purposes
// Given n data points x(i),y(i) and a function
// fct(x,c) where c is a vector of m parameters, find c values that
// minimize sum over i (y(i)-fct(x(i),c))^2 using Powell's method.
// c0 is initial guess for parameters. cStep is initial step size
// for parameter search.
//////////////////////////////////////////////////////////////////////
function [c,fctMin] = fitLeastSquares(xData,yData,fct,c0,cStep,tol)
  nData = length(xData);

  function w=fMSE(cTest)
    w = 0;
    for i=1:nData
      w = w+(yData(i)-fct(xData(i),cTest))^2;
    end
    w = w/nData;
  endfunction

  [c,fctMin] = optimPowell(fMSE,c0,cStep,tol);
endfunction
```