Approximation

Beyond Simple Linear Least-Squares

Sachin Shanbhag

Department of Scientific Computing
Florida State University,
Tallahassee, FL 32306.

Linear Least Squares

- In linear least squares, we are given $n+1$ data-points

$$(x_0, f_0), (x_1, f_1), \cdots, (x_i, f_i), \cdots (x_n, f_n).$$

- We seek a function $g(x)$, which approximately fits the data.
- In general, this $g(x)$ has the form:

$$g(x) = a_0\phi_0(x) + a_1\phi_1(x) + \cdots + a_m\phi_m(x).$$

- For polynomial interpolation,

$$\{\phi_0, \phi_1, ..., \phi_m\} = \{1, x, ..., x^m\}.$$

- For the Lorenz function ($m = 0$),

$$\{\phi_0\} = \left\{\frac{1}{1+x^2}\right\}$$

Linear Least Squares

Setting $g(x_i) = f_i$ gives us $n+1$ linear equations of the form:

$$\begin{bmatrix} \phi_0(x_0) & \phi_1(x_0) & \phi_2(x_0) & \ldots & \phi_m(x_0) \\ \phi_0(x_1) & \phi_1(x_1) & \phi_2(x_1) & \ldots & \phi_m(x_1) \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ \phi_0(x_n) & \phi_1(x_n) & \phi_2(x_n) & \ldots & \phi_m(x_n) \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \ldots \\ a_m \end{bmatrix} = \begin{bmatrix} f_0 \\ f_1 \\ \ldots \\ f_n \end{bmatrix}$$

This can be written as $\mathbf{A}\mathbf{a} = \mathbf{f}$, where $\mathbf{A} = \mathbf{A}(\{x_i\})$.

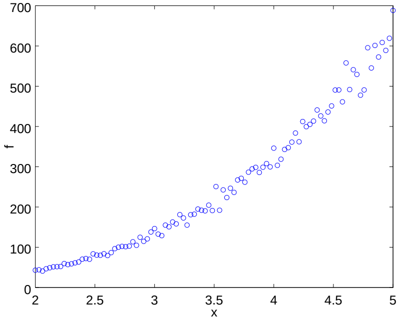
The least-squares solution is obtained by solving the $m+1$ normal equations:

$$\mathbf{A}^{\mathbf{T}}\mathbf{A}\hat{\mathbf{a}} = \mathbf{A}^{\mathbf{T}}\mathbf{f}$$

Sometimes, we can rephrase a problem and use linear least squares in situations where it may not necessarily be the obvious choice.

Linear Least Squares

- Consider fitting a function $g(x) = a_0 x^{a_1}$ to datapoints $\{x_i, f_i\}$ as shown in the figure below.



$$f(x) = 5x^3 + \text{noise}$$

- Note that I cannot write $g(x) = a_0\phi_0(x) + a_1\phi_1(x)$.
- I cannot formulate it as a LLS (linear least squares) problem.

Linear Least Squares

- However, consider a simple logarithmic transformation applied to the data:

$$X_i = \log x_i, \quad F_i = \log f_i,$$

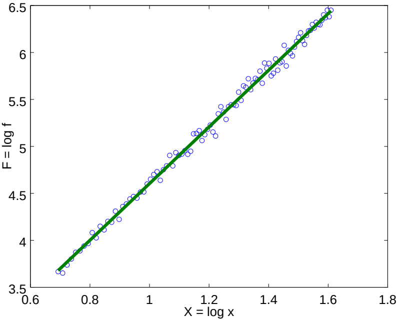
  and the function

$$\begin{aligned} G(X) &= \log g(x) \\ &= \log(a_0 x^{a_1}) \\ &= \log(a_0) + a_1 \log x \\ &= A_0 + a_1 X \end{aligned}$$

  where $A_0 = \log(a_0)$.
- In this rephrased problem, we have data $\{X_i, F_i\}$, and fitting function $G(X)$ which is linear in the parameters A_0 and a_1.

Linear Least Squares

We can solve the LLS problem and get:

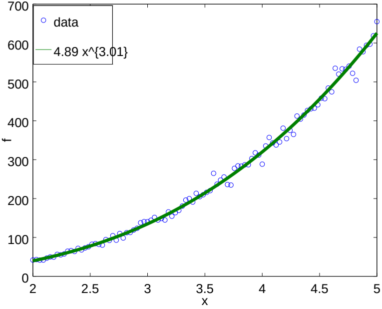


$$A_0 = 1.5878, \quad a_1 = 3.0165.$$

Therefore $a_0 = \exp(A_0) = 4.8928$.

Linear Least Squares

Finally, plotting, the original data and the fitting equation



$$g(x) = 4.8928x^{3.0165}$$

Nonlinear Least Squares

Many fitting functions are nonlinear, and they cannot be transformed to construct a LLS problem

Some simple examples:

(i) $g(x) = a_0 x^{a_1} + a_2$

(ii) $g(x) = a_0 x^{a_1} + a_2 x^{a_3}$

(iii) $g(x) = a_0 \sin(a_1 x + a_2)$

(iv) $g(x) = a_0 \exp(a_1 x) + a_2 \exp(a_3 x)$

In such cases, we have to solve the full nonlinear LS problem.

Nonlinear Least Squares

We go back to basics; recall that the "residual" corresponding to the i^{th} data-point/equation is

$$r_i(x_i, \mathbf{a}) = g(x_i, \mathbf{a}) - f_i$$

The squared error is the sum of the squares of all the $n+1$ residuals:

$$\begin{aligned} \epsilon^2(\mathbf{a}) &= ||\mathbf{r}(\mathbf{a})||_2^2 \\ &= \sum_{i=0}^{n} r_i^2(x_i, \mathbf{a}) \\ &= \sum_{i=0}^{n} (g(x_i, \mathbf{a}) - f_i)^2 \end{aligned}$$

Nonlinear Least Squares

In principle, since we can pose the problem of minimizing $\epsilon^2(\mathbf{a})$ as a multidimensional optimization problem.

The "multiple" dimensions come from the $m+1$ components of $\mathbf{a}$.

We may then be able to use Newton's method, BFGS, conjugate gradient, or any such method.

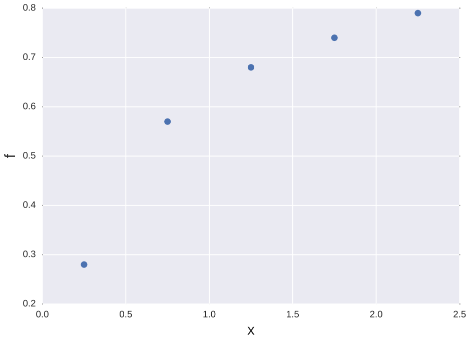
In general, nonlinear LS (NLLS) problems are "harder" than LLS; multiple optima exist, and iterative solutions are required.

Let us consider a concrete example:

Example

Consider fitting a function $g(x; a_0, a_1) = a_0(1 - e^{-a_1 x})$ to the data:

$\mathbf{x}$	0.25	0.75	1.25	1.75	2.25
$\mathbf{f}$	0.28	0.57	0.68	0.74	0.79



Note that $g(x, \mathbf{a})$ is nonlinear in $\mathbf{a}$; we cannot write it out as $g(x, \mathbf{a}) = a_0\phi_0(x) + a_1\phi_1(x)$.

Example

In this problem, we have $n+1=5$ data-points/equations, and $m+1=2$ fitting parameters

The residuals are:

$$\begin{aligned} r_i(x_i, \mathbf{a}) &= g(x_i, \mathbf{a}) - f_i \\ r_i(a_0, a_1) &= a_0(1 - e^{-a_1 x_i}) - f_i, \quad i = 0, ..., 4 \end{aligned}$$

The squared error is:

$$\epsilon^2(a_0, a_1) = \sum_{i=0}^{4} r_i^2(a_0, a_1).$$

Let us use BFGS to minimize $\epsilon^2(a_0, a_1)$, with an initial guess of $a_0 = a_1 = 1$.

Example

We need to derive an expression for the gradient of $\epsilon^2(a_0, a_1)$:

$$\nabla \epsilon^2(a_0, a_1) = \begin{bmatrix} \frac{\partial \epsilon^2}{\partial a_0} \\ \frac{\partial \epsilon^2}{\partial a_1} \end{bmatrix}$$

For $k = 0$ and $k = 1$, we have

$$\begin{aligned} \frac{\partial \epsilon^2}{\partial a_k} &= \sum_{i=0}^{4} \frac{\partial r_i^2}{\partial a_k} \\ &= \sum_{i=0}^{4} 2 r_i \frac{\partial r_i}{\partial g} \frac{\partial g}{\partial a_k} \\ &= \sum_{i=0}^{4} 2 r_i \frac{\partial g(x_i, \mathbf{a})}{\partial a_k} \end{aligned}$$

Example

Thus, we need the derivatives of $g(x_i, \mathbf{a})$ with respect to a_0 and a_1.

$$\frac{\partial g(x_i, \mathbf{a})}{\partial a_0} = 1 - e^{-a_1 x_i},$$

and

$$\frac{\partial g(x_i, \mathbf{a})}{\partial a_1} = a_0 x e^{-a_1 x_i}$$

We can now assemble all of these parts into our BFGS program.

Matlab Code:

The function $\epsilon^2(\mathbf{a})$ is the sum of the squared residuals

```
% epsilon2 = sum of residual squares
function Z = f(a)

  x = [0.25 0.75 1.25 1.75 2.25];
  f = [0.28 0.57 0.68 0.74 0.79];
  Z = 0.;

  for i = 1:numel(x)
    gi = a(1) * ( 1 - exp(-a(2)*x(i)) );
    ri = gi - f(i);
    Z = Z + ri * ri;
  end

end
```

Matlab Code:

The gradient of the function $\epsilon^2(\mathbf{a})$.

```
% gradient(epsilon2)
function Z = gradf(a)

  x = [0.25 0.75 1.25 1.75 2.25];
  f = [0.28 0.57 0.68 0.74 0.79];
  Z = [0.;0.];

  for i = 1:numel(x)

    gi = a(1) * ( 1 - exp(-a(2)*x(i)) );
    ri = gi - f(i);

    dgida1 = 1 - exp(-a(2)*x(i));
    dgida2 = a(1)*x(i)*exp(-a(2)*x(i));

    Z = Z + 2 * ri * [dgida1; dgida2];

  end
```

Matlab Code: Calling Routine

Using the old BFGS function, without any changes.

```
% BFGS: use as [x f n] = bfgs_nlls([1;1], 1e-3);
function [xopt fopt nopt] = bfgs_nlls(x0, tol)

  x = x0;       n = length(x);
  B = eye(n);   k = 0;

  while(norm(gradf(x)) > tol) % Need to make gradf ~ 0
    df      = gradf(x);
    s       = B\(-df);
    x       = x + s;
    y       = gradf(x) - df;
    B       = B + (y*y')/(y'*s) - (B*s*s'*B)/(s'*B*s);
    k       = k + 1;
  end

  xopt = x;  fopt = f(x);  nopt = k;

end
```

Example

With an initial guess of [1; 1] tolerance of 10^{-3}:

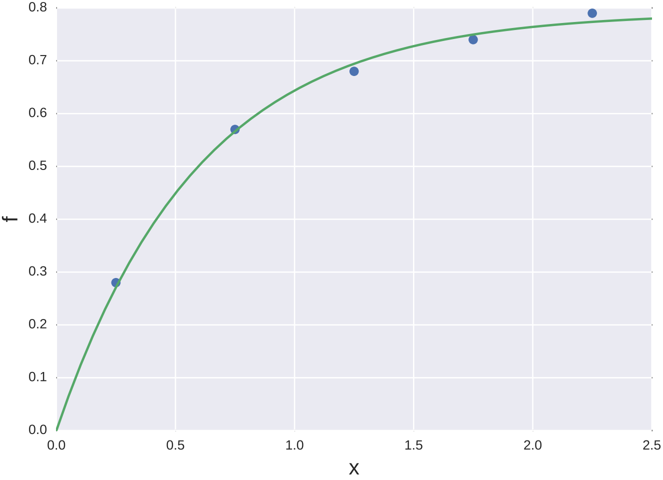
We get the final solution as:

$$\mathbf{a}^* = \begin{bmatrix} 0.7919 \\ 1.6751 \end{bmatrix}$$

after 10 iterations.

The least squared error is $\epsilon^2(\mathbf{a}^*) = 6.62 \times 10^{-4}$

Example Solution



Perspective

- For nonlinear least squares, there are specialized algorithms like Gauss-Newton and Levenberg-Marquardt
- These algorithms exploit the form of the function $\epsilon^2(\mathbf{a})$
- They linearize $r_i(\mathbf{a})$, which leads to a linear LS problem during each iteration
- For linear LS, we said, solve the normal equations,

$$\mathbf{A}^T\mathbf{A}\mathbf{a} = \mathbf{A}^T\mathbf{f}.$$

- If A is ill-conditioned to begin with, multiplying by $\mathbf{A}^T$ can make the problem worse off
- Such problems are typically solved by QR factorization, or some variant thereof.