AMSC 607 / CMSC 764 Advanced Numerical Optimization
Fall 2008
UNIT 4: Special Topics
PART 1: Nonlinear Least Squares and Data Fitting
Dianne P. O'Leary

Special Topic 1: Nonlinear Least Squares

The plan:

- A brief review of linear least squares:
  - the problem
  - how to solve it
  - statistical properties (13.3)
- Alternatives to least squares
  - ℓ_1 data fitting
  - min-max data fitting
- Nonlinear data fitting example (13.2)
- Algorithms for nonlinear least squares
  - Gauss-Newton (13.2)
  - Levenberg-Marquardt (13.2)
  - "Varpro"
- errors in both variables: orthogonal distance regression (13.4)

Reference: N&S Chapter 13

Linear Least Squares

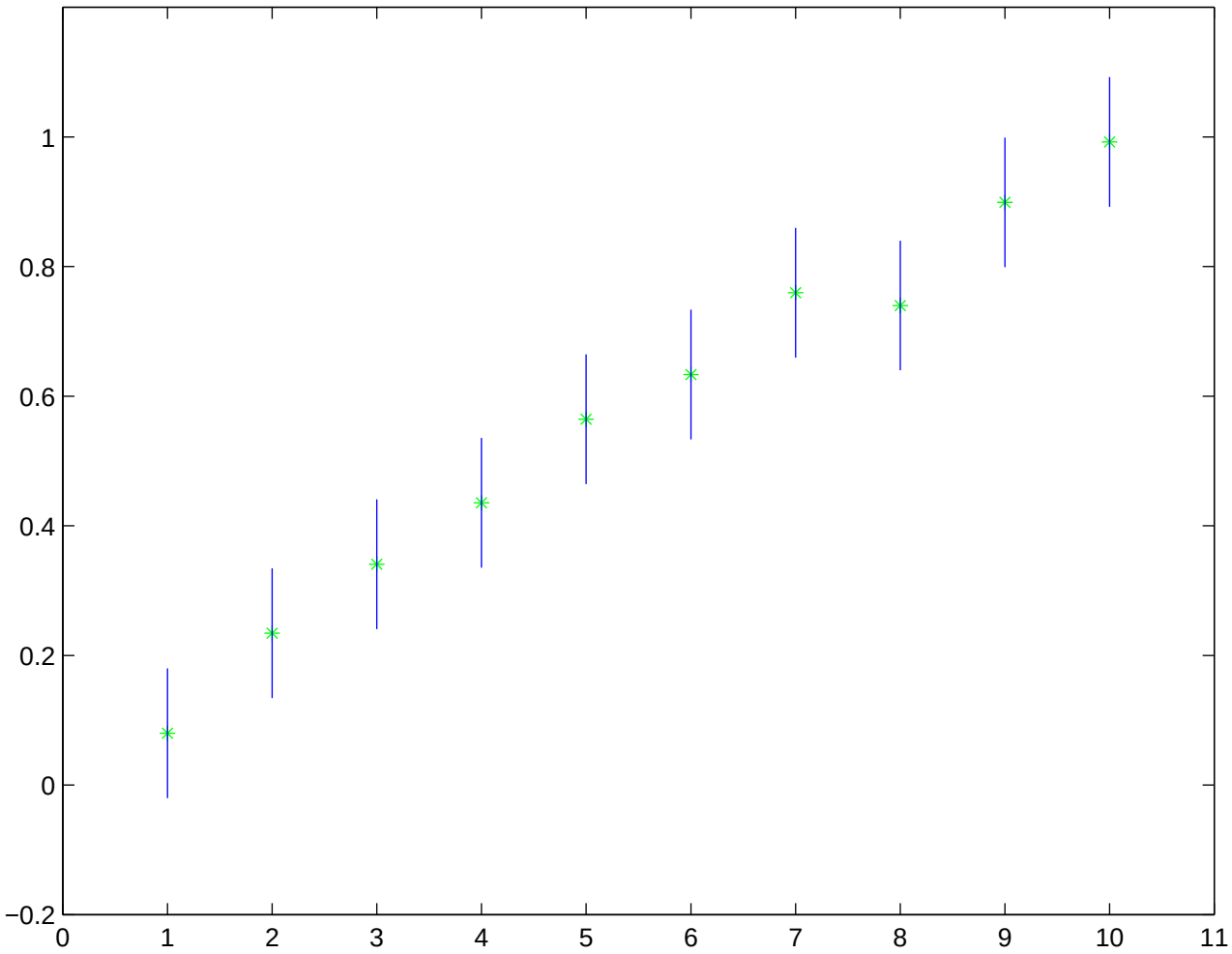
Motivating example

Fit a model to data in order to reduce the effects of noise in the measurements.

Given: a set of basis functions $\phi_1(t), \phi_2(t), \ldots, \phi_n(t)$ that we believe to model the behavior of some function or set of data,

Find: coefficients $x_1, x_2, \ldots, x_n$ so that

$$u(t) = \sum_{j=1}^{n} x_j \phi_j(t)$$


1
0.8
0.6
0.4
0.2
0
−0.2
0
1
2
3
4
5
6
7
8
9
10
11

We measure this difference at a set of m values $t_1, \ldots, t_m$ at which we have measurements b_i:

$$\min_{\mathbf{x}} \|\mathbf{b} - \mathbf{u}\|^2 = \min_{\mathbf{x}} \sum_{i=1}^{m} [b_i - u(t_i)]^2 .$$

Solving least squares problems using the normal equations

Transformation of the problem

Let's see if we can get some insight into our minimization problem.

$$\|\mathbf{b} - \mathbf{u}\|^2 = \sum_{i=1}^{m} [b_i - u(t_i)]^2$$

Now we know that

$$u(t) = \sum_{j=1}^{n} x_j \phi_j(t)$$

so

$$\|\mathbf{b} - \mathbf{u}\|^2 = \sum_{i=1}^{m} [b_i - \sum_{j=1}^{n} x_j \phi_j(t_i)]^2 .$$

If we define the residual vector r so that

$$r_i = b_i - \sum_{j=1}^{n} x_j \phi_j(t_i)$$

then our minimization problem is to minimize $\|\mathbf{r}\|$ over all choices of $\mathbf{x}$.

There is a further very nice simplification. Define the matrix $\mathbf{A}$ to have entries $a_{ij} = \phi_j(t_i)$. Then

$$\mathbf{r} = \mathbf{b} - \mathbf{A}\mathbf{x} ,$$

and we have expressed our problem as a matrix one:

$$\min_{\mathbf{x}} \|\mathbf{b} - \mathbf{A}\mathbf{x}\|^2 .$$

This quantity that we minimize is $\mathbf{b}^T\mathbf{b} - 2\mathbf{x}^T\mathbf{A}^T\mathbf{b} + \mathbf{x}^T\mathbf{A}^T\mathbf{A}\mathbf{x}$, and we will let $\mathbf{B} = \mathbf{A}^T\mathbf{A}$ and $\mathbf{c} = \mathbf{A}^T\mathbf{b}$, reducing the problem to minimizing

$$\mathbf{b}^T\mathbf{b} - 2\mathbf{x}^T\mathbf{c} + \mathbf{x}^T\mathbf{B}\mathbf{x}.$$

Positive semidefiniteness

Since $\mathbf{B} = \mathbf{A}^T\mathbf{A}$, we see that, for any vector $\mathbf{x}$,

$$\mathbf{x}^T\mathbf{B}\mathbf{x} = \mathbf{x}^T\mathbf{A}^T\mathbf{A}\mathbf{x} = \|\mathbf{A}\mathbf{x}\|^2 \geq 0$$

and therefore $\mathbf{B}$ is symmetric positive semi-definite.

Let's solve using calculus. Necessary conditions to have a minimizer are:

1. The first derivative must be zero.

$$-2\mathbf{c} + 2\mathbf{B}\mathbf{x} = \mathbf{0}\,.$$

   Jargon: The n linear equations $\mathbf{B}\mathbf{x} = \mathbf{c}$ are called the normal equations.

2. The second derivative matrix, $\mathbf{B}$ must be positive semi-definite.

If $\mathbf{B}$ is full rank, then the solution exists and is unique. If $\mathbf{B}$ is rank deficient (which happens when $\mathbf{A}$ fails to have linearly independent columns), then it is fair to say that we have chosen a poor set of basis functions ϕ_j.

Note: Unfortunately, real problems often have bad bases.

If $\mathbf{B}$ is full rank, then we can solve the linear system $\mathbf{B}\mathbf{x} = \mathbf{c}$ using the Cholesky decomposition.

Solving least squares problems using orthogonal factorizations

---

The QR factorization

The best tool for solving least squares problems in which $\mathbf{A}$ is a full rank matrix is the QR factorization.

$\mathbf{A} = \hat{\mathbf{Q}}\hat{\mathbf{R}}$, where $\hat{\mathbf{R}}$ is $m \times n$,

$$\hat{\mathbf{R}} = \begin{bmatrix} \mathbf{R} \\ \mathbf{0} \end{bmatrix}$$

$\hat{\mathbf{Q}} = [\mathbf{Q}, \bar{\mathbf{Q}}]$ is $m \times m$ with orthonormal columns.

Let $\mathbf{r} = \mathbf{b} - \mathbf{A}\mathbf{x}$ and let $\mathbf{c} = \hat{\mathbf{Q}}^T\mathbf{b}$ be partitioned into two pieces:

- $\mathbf{c}_1$ of dimension n and
- $\mathbf{c}_2$ of dimension $m - n$.

Then

$$\begin{aligned} \|\mathbf{r}\|^2 &= \|\mathbf{b} - \mathbf{A}\mathbf{x}\|^2 \\ &= \|\mathbf{b} - \hat{\mathbf{Q}}\hat{\mathbf{R}}\mathbf{x}\|^2 \\ &= \|\hat{\mathbf{Q}}^T[\mathbf{b} - \hat{\mathbf{Q}}\hat{\mathbf{R}}\mathbf{x}]\|^2 \\ &= \|\mathbf{c} - \begin{bmatrix} \mathbf{R} \\ \mathbf{0} \end{bmatrix}\mathbf{x}\|^2 \\ &= \|\mathbf{c}_1 - \mathbf{R}\mathbf{x}\|^2 + \|\mathbf{c}_2 - 0\mathbf{x}\|^2\,. \end{aligned}$$

Now, no matter what $\mathbf{x}$ is, the second term remains unchanged.

If we want to minimize this quantity with respect to $\mathbf{x}$, what we need to do is to solve a least squares problem involving $\mathbf{c}_1$ and $\mathbf{R}$. If $\mathbf{R}$ is full rank, then we can make the first term zero, and the norm of the residual $\mathbf{r}$ is simply $\|\mathbf{c}_2\|$.

Algorithm

1. Compute the QR factorization of $\mathbf{A}$.
2. Form $\mathbf{c}_1 = \mathbf{Q}^T\mathbf{b}$.
3. Solve the square, triangular system $\mathbf{Rx} = \mathbf{c}_1$. (Solve it in the least squares sense, if $\mathbf{R}$ is rank deficient.)
4. If $\bar{\mathbf{Q}}$ is available and $\mathbf{R}$ is full rank, then the norm of the residual can be computed as $\|\bar{\mathbf{Q}}^T\mathbf{b}\|$.

   Otherwise the residual is computed as $\mathbf{r} = \mathbf{b} - \mathbf{Ax}$.

Examples

An example: our straight line, revisited

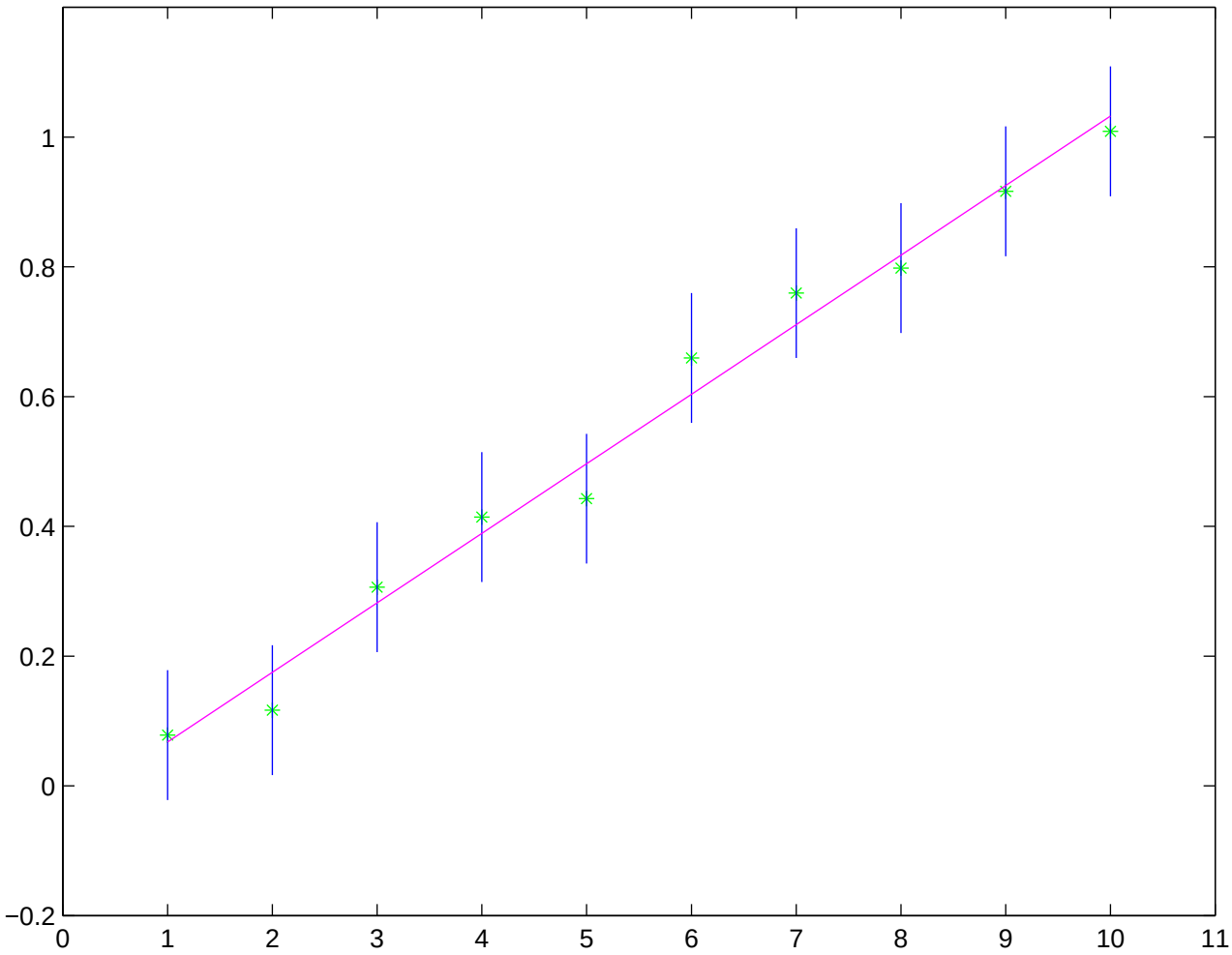
Problem: Fit a model to data in order to reduce the effects of noise in the measurements.

Recall the data from above. Is a straight line a good fit to (t_i, b_i), $i = 1, \ldots, 10$?

Data:

$$\mathbf{A} = \begin{bmatrix} 1 & t_1 \\ . & . \\ . & . \\ . & . \\ 1 & t_{10} \end{bmatrix}, \mathbf{b} = \begin{bmatrix} b_1 \\ . \\ . \\ . \\ b_{10} \end{bmatrix}$$

```
sigma=.05
t = [1:10];
ve = ...
plot(t,ve,'g*')
hold on
for i=1:10,
    plot([t(i),t(i)],[ve(i)+2*sigma,ve(i)-2*sigma])
end
axis([0 11 -.2 1.2])
a = [ones(10,1),t'];
coef = a \ ve';
plot(t,a*coef,'m')
```


1
0.8
0.6
0.4
0.2
0
−0.2
0
1
2
3
4
5
6
7
8
9
10
11

A second example: sensitivity can hurt

Suppose we want to fit a polynomial to some data and suppose that we are ignorant enough to use the power basis.

Our data is taken at $1, 2, \ldots, 30$.

A Matlab program:

```
% Compute the condition numbers of the power-basis matrix
% for various degrees n-1 of the polynomial

t=[1:30]';
m = length(t)
disp('    n        cond(A)')
for n=2:10,
a=ones(m ,1);
for i=2:n,
   a = [a, a(:,i-1).*t];
end
s = sprintf('%5d %15.5e',n,cond(a));
disp(s)
end

pause

% Perform a fit for a polynomial of degree 8
% assuming that the true data is from a polynomial
% with coefficients 1, 2, ..., 9, and some
% random noise (mean zero, standard deviation 1)
% has been added.

n=9
a=ones(m ,1);
for i=2:n,
   a = [a, a(:,i-1).*t];
end

v = a * [1:n]' + rand(m ,1);
format long
coef1 = a \ v;
coef2 = (a'*a) \ (a'*v);
disp('          QR       normal equations')
disp([coef1,coef2])
```

Results:

```
 n         cond(A)
 2     3.65007e+01
 3     1.35936e+03
 4     5.07537e+04
 5     1.93735e+06
 6     7.68136e+07
 7     3.18589e+09
 8     1.38024e+11
 9     6.21454e+12
10     2.88547e+14
```

```
       QR       normal equations
1.0e+02 *

0.02592099356981  -2.35878463268852
0.00880868395348   3.56794884601309
0.03422551695005  -1.66156270129563
0.03923888065992   0.40975832216991
0.05007398826829   0.00699043677858
0.05999594170322   0.06283289495303
0.07000012452822   0.06989408241536
0.07999999802536   0.08000209368768
0.09000000001244   0.08999998301781
```

The problem here is a bad choice of basis functions. A basis for polynomials other than the power basis should be used.

Statistical Properties

We'll consider two alternatives to least squares, but the main advantage of least squares is its statistical properties.

Some statistics

If the errors in the observations y_i are normally distributed with mean zero and known variance, (and we put the variance matrix $\mathbf{\Sigma}^2$ in as weights to the least squares minimization function)

$$\min_{\mathbf{x}} \|\mathbf{Ax} - \mathbf{b}\|_{\mathbf{\Sigma}^{-2}} = \min_{\mathbf{x}} \|\mathbf{\Sigma}^{-1}(\mathbf{Ax} - \mathbf{b})\|_2$$

- Then the least squares solution is the minimum variance unbiased estimator of the true parameters.
- The least squares solution is the maximum likelihood estimator.

- We can compute confidence intervals, centered at the least squares estimate, so that we can make statements like, "If we repeated the experiment 100 times, then 90% of the time we would obtain parameter estimates within these confidence intervals."

This, plus inexpensive algorithms for computing least squares solutions, explains the popularity of this type of modeling.

But if we don't have such simple error properties, or if we don't have estimates of the variances, then it might make sense to consider alternatives.

Alternatives to least squares

Recall two problems that we discussed under SOCP:

$$\min_{\mathbf{x}} \sum_{i=1}^{\ell} \|\mathbf{F}_i\mathbf{x} + \mathbf{g}_i\|_2$$

and

$$\min_{\mathbf{x}} \max_{i} \|\mathbf{F}_i\mathbf{x} + \mathbf{g}_i\|_2$$

We'll consider similar problems now.

ℓ_1 data fitting

Instead of minimizing $\|\mathbf{f}(\mathbf{x})\|_2$, suppose we choose to minimize

$$\|\mathbf{f}(\mathbf{x})\|_1 = \sum_{i=1}^{m} |f_i(\mathbf{x})| \,.$$

This is a linear programming problem:

$$\min_{\mathbf{x},\mathbf{r},\mathbf{w}} \mathbf{e}^T\mathbf{r} + \mathbf{e}^T\mathbf{w}$$

$$\begin{aligned} \mathbf{A}\mathbf{x} - \mathbf{b} &= \mathbf{r} - \mathbf{w} \\ \mathbf{r} &\geq \mathbf{0} \\ \mathbf{w} &\geq \mathbf{0} \end{aligned}$$

This works since a basic solution to this problem will have $r_i = 0$ if $w_i > 0$ and vice-versa.

Min-max data fitting

Instead of minimizing $\|\mathbf{f}(\mathbf{x})\|_2$, suppose we choose to minimize

$$\|\mathbf{f}(\mathbf{x})\|_\infty = \max_{i=1,\ldots,m} |f_i(\mathbf{x})| .$$

This is also a linear programming problem:

$$\min_{\mathbf{x},\mathbf{r},t} \; t$$

$$\mathbf{A}\mathbf{x} - \mathbf{b} = \mathbf{r}$$

$$-t\mathbf{e} \leq \mathbf{r} \leq t\mathbf{e}$$

Nonlinear data fitting example

The problem

$$\min_{\mathbf{x}} F(\mathbf{x})$$

where

$$F(\mathbf{x}) = \frac{1}{2}\sum_{i=1}^{m} [f_i(\mathbf{x})]^2 = \frac{1}{2}\mathbf{f}(\mathbf{x})^T\mathbf{f}(\mathbf{x}) = \frac{1}{2}\|\mathbf{f}(x)\|_2^2 .$$

Example: Suppose we have measured values (t_i, y_i), $i = 1, \ldots, m$, and we believe the model

$$y(t) \approx x_1 e^{x_3 t} + x_2 e^{x_4 t} .$$

Then

$$f_i(x) = y_i - (x_1 e^{x_3 t} + x_2 e^{x_4 t}) .$$

Note:

- This is a little different from the linear problem we considered previously, because the basis functions $e^{x_3 t}$ and $e^{x_4 t}$ now depend on unknown parameters.
- Fitting exponentials is one of the hardest forms of nonlinear least squares computations, because the problem is so ill-conditioned: small changes in the data y_i can cause extraordinarily large changes in the parameters, especially the rate constants x_3 and x_4.

Some mechanics

Let's calculate the gradient and Hessian of the function we are minimizing:

$$\mathbf{g}(\mathbf{x}) = \mathbf{S}(\mathbf{x})^T \mathbf{f}(\mathbf{x})$$

where

$$s_{ij} = \frac{\partial f_i}{\partial x_j},$$

and the Hessian matrix is

$$\mathbf{H}(\mathbf{x}) = \mathbf{S}(\mathbf{x})^T \mathbf{S}(\mathbf{x}) + \sum_{i=1}^{m} f_i(\mathbf{x}) \bigtriangledown^2 f_i(\mathbf{x}).$$

An important observation

$$\mathbf{H}(\mathbf{x}) = \mathbf{S}(\mathbf{x})^T \mathbf{S}(\mathbf{x}) + \sum_{i=1}^{m} f_i(\mathbf{x}) \bigtriangledown^2 f_i(\mathbf{x}).$$

Suppose our model is good. Then the values $f_i(\mathbf{x}^*)$ are small, so

$$\mathbf{H}(\mathbf{x}) \approx \mathbf{S}(\mathbf{x})^T \mathbf{S}(\mathbf{x})$$

when $\mathbf{x}$ is close to $\mathbf{x}^*$.

This is not true if the model is not good, but why would we solve the problem for a bad model?

Gauss-Newton

The Newton direction:

$$\mathbf{H}(\mathbf{x})\mathbf{p} = -\mathbf{g}(\mathbf{x}).$$

where

$$\mathbf{H}(\mathbf{x}) = \mathbf{S}(\mathbf{x})^T \mathbf{S}(\mathbf{x}) + \sum_{i=1}^{m} f_i(\mathbf{x}) \bigtriangledown^2 f_i(\mathbf{x}).$$

The Gauss-Newton idea is to neglect the second term:

$$\mathbf{S}(\mathbf{x})^T \mathbf{S}(\mathbf{x})\mathbf{p} = -\mathbf{g}(\mathbf{x}).$$

So the Gauss-Newton algorithm uses this direction rather than the true Newton direction, but is otherwise the same as Newton's method.

When is the direction bad?

- When the residuals $f_i(\mathbf{x})$ fail to be small.
- when $\mathbf{S}(\mathbf{x})$ fails to have full rank.

In both of these cases, the piece of $\mathbf{H}(\mathbf{x})$ that we are neglecting is important.

Computing the direction

If we compute the Gauss-Newton direction from the formula

$$\mathbf{S}(\mathbf{x})^T\mathbf{S}(\mathbf{x})\mathbf{p} = -\mathbf{g}(\mathbf{x})\,,$$

then the ill-conditioning of the matrix $\mathbf{S}(\mathbf{x})^T\mathbf{S}(\mathbf{x})$ can be a problem, just as in the normal equations for linear least squares.

In fact, recalling that

$$\mathbf{g}(\mathbf{x}) = \mathbf{S}(\mathbf{x})^T\mathbf{f}(\mathbf{x})$$

we see that the Gauss-Newton direction equation really is the solution to a linear least squares problem

$$\min_{\mathbf{p}} \|\mathbf{S}(\mathbf{x})\mathbf{p} + \mathbf{f}(\mathbf{x})\|$$

so we should solve it using a QR factorization of $\mathbf{S}(\mathbf{x})$ instead of factoring $\mathbf{S}(\mathbf{x})^T\mathbf{S}(\mathbf{x})$.

Levenberg-Marquardt

The Newton direction:

$$\mathbf{H}(\mathbf{x})\mathbf{p} = -\mathbf{g}(\mathbf{x})\,.$$

where

$$\mathbf{H}(\mathbf{x}) = \mathbf{S}(\mathbf{x})^T\mathbf{S}(\mathbf{x}) + \sum_{i=1}^{m} f_i(\mathbf{x})\,\bigtriangledown^2 f_i(\mathbf{x})\,.$$

The Levenberg-Marquardt idea is to approximate the second term:

$$\sum_{i=1}^{m} f_i(\mathbf{x})\,\bigtriangledown^2 f_i(\mathbf{x}) \approx \lambda\mathbf{I}\,.$$

(Note: We have seen this idea before, in our discussion of trust regions.)

The only issue is the choice of λ; refer back to the trust region discussion of the issues.

"Varpro"

Why I want to discuss this: One of my colleagues calls Varpro the greatest discovery in numerical analysis of the 20-th century. How can I resist?

- Problems with some linear parameters (like x_1 and x_2 in our expontial fitting example) and some nonlinear parameters (like x_3 and x_4) occur so frequently that it makes sense to develop algorithms to take advantage of this structure in order to make the computation more inexpensive.
- This can also help solve any convergence problems.

The Variable Projection algorithm, or Varpro, is designed to do this.

Credits: The algorithm was developed by Linda Kaufman (*BIT* 15 (1975) p49) after ideas of Gene Golub and Victor Pereyra.

The Varpro algorithm

Let's call the linear parameters **x** and the nonlinear parameters **z**. Then our problem is

$$\min_{\mathbf{x},\mathbf{z}} \frac{1}{2}\mathbf{f}(\mathbf{x},\mathbf{z})^T\mathbf{f}(\mathbf{x},\mathbf{z}).$$

Many people have gotten the idea of alternating between two subproblems:

- Update **z**, using the current values for **x**.
- Update **x**, using the current values for **z**.

But this algorithm does not work very well.

Instead, we can partition our original problem into two subproblems:

- If someone gave us the parameters **z**, we could solve the linear least squares problem for **x**.

  In our example, if someone told us the rate constants x_3 and x_4, then we are just fitting the linear model

  $$x_1 e^{x_3 t} + x_2 e^{x_4 t}$$

  with parameters x_1 and x_2.
- To get the **z** parameters, we need to find the values that best account for the data; i.e., the values for which the component of the data orthogonal to the model is minimized.

In our example, we need to determine x_3 and x_4 so that we solve the problem

$$\min_{\mathbf{z}} \|\mathbf{P}\mathbf{y}\|$$

where $\mathbf{P}$ is the projector onto the space orthogonal to the range of the columns of

$$\begin{bmatrix} e^{x_3 t_1} & e^{x_4 t_1} \\ e^{x_3 t_2} & e^{x_4 t_2} \\ \cdots & \cdots \\ e^{x_3 t_m} & e^{x_4 t_m} \end{bmatrix}.$$

Note that we don't alternate between the two subproblems: we just solve the second one first and then the first one, and we are finished!

Kaufman in her paper works through the linear algebra of the computations. She solves each of the two subproblems using variants on QR factorizations.

Errors in both variables: orthogonal distance regression

Suppose we have our exponential model again, and we have taken our observations (t_i, y_i), but our "clock" for measuring t_i has some uncertainty in it.

So now we need to deal with errors in the variables t_i as well as in the measured values y_i.

It doesn't make sense to minimize the vertical distance between the model and the observations any more.

Picture (Figure 13.2)

Our problem gets more complicated: we need to minimize two things:

- the difference between the model values and the observed values.
- the corrections in the variables t_i

$$t_i + \delta_i \,.$$

This leads us to the problem

$$\min_{\mathbf{x},\boldsymbol{\Delta}} \frac{1}{2}\mathbf{f}(\mathbf{x}, \mathbf{t} + \boldsymbol{\Delta})^T \mathbf{f}(\mathbf{x}, \mathbf{t} + \boldsymbol{\Delta}) + \lambda \boldsymbol{\Delta}^T \boldsymbol{\Delta}\,,$$

where λ is a parameter chosen to balance out the two criteria.

N&S p. 423 provides good references on this topic.

A computational example of Nonlinear Least Squares

Let's fit a sum of two exponentials. I don't have a Varpro implementation, so I'll use the function `lsqnonlin` in Matlab's Optimization Toolbox.

The program:

```
% nonlinear least squares example

global t y

% Original problem: no error in the data

t = 0:.1:10;
y = .2 * exp(-5*t) + .4 * exp(-100*t);

x1 = lsqnonlin('nonlinfnct',[.2 2 1 6])
y1 = x1(1) * exp(-x1(2)*t) + x1(3) * exp(-x1(4)*t);

subplot(2,1,1)
plot(t,y,'b',t,y1,'g');
legend('original data','model')

% Now add a relative error of 1.e-3

y = (1+(1.e-3)*rand(size(y))).*y;
x2 = lsqnonlin('nonlinfnct',[.2 2 1 6])
y2 = x2(1) * exp(-x2(2)*t) + x2(3) * exp(-x2(4)*t);

subplot(2,1,2)
plot(t,y,'b',t,y2,'g');
legend('perturbed data','model')
```

```
function f = nonlinfnct(x)

global t y

f = y - x(1)*exp(-x(2)*t) - x(3)*exp(-x(4)*t);
```

The results:

```
>> clear;nonlinlsex
Optimization terminated successfully:
 Relative function value changing by less than OPTIONS.TolFun

x1 =

    0.1936    4.9513    0.4063   45.8937

Optimization terminated successfully:
 Relative function value changing by less than OPTIONS.TolFun

x2 =

    0.1933    4.9591    0.4075   43.9400
```

Notes:

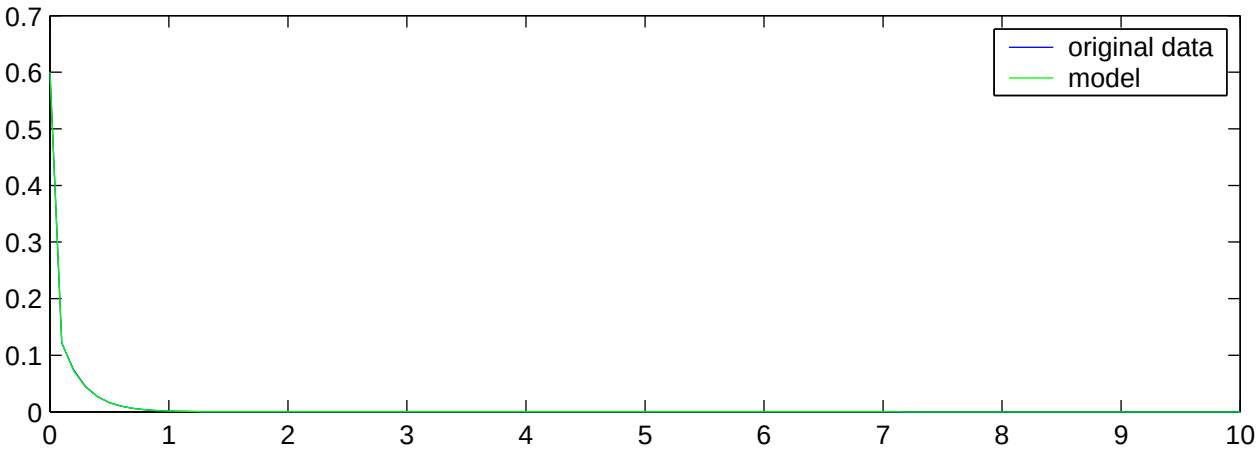
- The computed solutions are far from the true solution of (.2, 5, .4, 100), but the plots don't reveal any problems.
- Note the sensitivity to the initial starting guess:

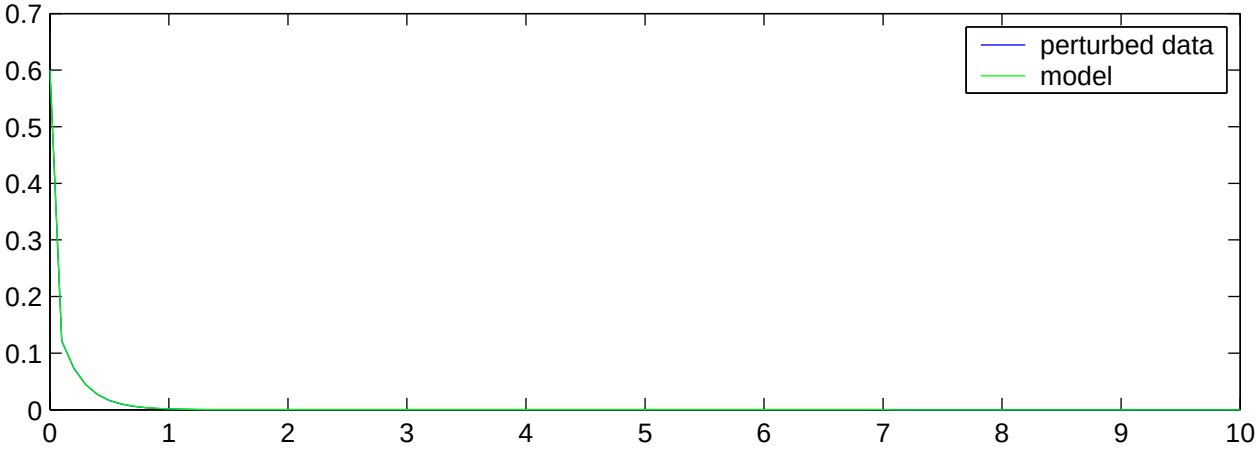
```
>> x1 = lsqnonlin('nonlinfnct',[1 1 1 1])
Optimization terminated successfully:
 Relative function value changing by less than OPTIONS.TolFun

x1 =

    0.2984   13.9262    0.2984   13.9262
```

Final words

- Exponential fitting is dangerous. Don't take the results too seriously unless you do a perturbation analysis, using wiggled data and vastly different starting guesses.
- Sometimes it makes sense to add constraints to the least squares problem.
  - For example, if our parameters are the gray levels in a reconstruction of a picture, then they are constrained to lie between black and white, which perhaps is modeled as 0 to 1.
  - As a second example, if we are trying to determine the amount of two chemical species in a mixture, the amounts must be nonnegative.

  We know lots of algorithms for solving constrained problems!


0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
1
2
3
4
5
6
7
8
9
10
original data
model


0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
1
2
3
4
5
6
7
8
9
10
perturbed data
model