Data reduction and fitting

Eugeniy E. Mikhailov
The College of William & Mary
Lecture 07

Data reduction

- Typical modern experiment generates huge amount of data.
- There is no way for a human to comprehend such enormous amount of data.

\[ y \]
\[ x \]

- To extract important parameters, we need to post-process the data.
- Alternatively, we want to check how our models reflect reality.

Fitting

Someone measured the dependence of an experimental parameter \( y \) on another parameter \( x \). We want to extract the unknown model parameters \( p_1, p_2, p_3, \ldots = \beta \) via fitting (i.e., finding the best \( \hat{\beta} \)) of the model function which depends on \( x \) and \( \beta \): \( f(x, \beta) \).

In general, \( x \) and \( y \) could be vectors i.e., multi-dimensional.

Example

- \( x \) has 2 coordinates: speed of a car and the weight of its load;
- \( y \) has the car fuel consumption and temperature.

For simplicity, we will focus on the one dimensional case for \( x \) and \( y \):
- We are given experimental points \( x_i \rightarrow y_i \);
- Our model \( x_i \rightarrow y_i = f(x_i, \beta) \).
Goodness of the fit

We need to define some way to estimate the goodness of the fit.

Chi-squared test

\[ \chi^2 = \sum_i (y_i - y_{fi})^2 \]

Differences of \((y_i - y_{fi})\) are called residuals.

For a given set of \(\{(x_i, y_i)\}\) and \(f\) the goodness of the fit \(\chi^2\) depends only on parameters vector \(\vec{\beta}\) of the model/fit function.

Our job is simple: find optimal \(\vec{\beta}\) which minimizes \(\chi^2\) using any suitable algorithm. I.e., perform so called the least square fit.

Good fit should have the following properties

- the fit should use the smallest possible fitting parameters set
  - with enough fitting parameters you can make zero residuals fit but this is unphysical since all your data has uncertainties in the measurements
- residuals should be randomly scattered around 0
  - i.e. no visible trends of residuals vs \(x\)
- standard deviation or RMS residual \(= \sqrt{\frac{\sum_i (y_i - y_{fi})^2}{N}}\) should be in order of the \(\Delta y\) (experimental uncertainty for \(y\))
  - the above condition is often overlooked but you should keep your eyes on it. It also can give you actual estimate of the experimental error bars
- fit should be robust: new points must not change parameters much
- Eugeniy's extra: stay away from the high order polynomial fits.
  - line is good, parabola maybe
  - anything else only if there is a deep physical reason for it
  - besides, such fits are usually useless since every new data point usually drastically modifies the fit parameters

Estimation of uncertainty for parameters

- \(\Delta \vec{\beta}\) could be estimated by change of the \(\chi^2\),
- \(\Delta \vec{\beta}: \chi^2(p_1, p_2, p_3, \ldots + \Delta p_i, \ldots) = 2\chi^2(p_1, p_2, p_3, \ldots, \ldots)\)

Practical realization

Have a look at 'fitter.m' where optimization of \(\chi^2\) is done with fminsearch matlab function.
See 'fitter_usage_example.m' for a particular usage example.
Matlab built-ins

- see `fit` from the Matlab curve fitting toolbox
- more cumbersome to start using
- provides parameters uncertainties
- see `lsqcurvefit` from the Matlab optimization toolbox

They are faster since they take an assumption that merit function is quadratic.

Matlab built-in fit usage example

```
%% built in fit function usage example
load initial data file
data=load('data_to_fit.dat');
x=data(:,1); % 1st column is x
y=data(:,2); % 2nd column is y

% define the fitting function with fittype
% notice that it is quite human readable
% Matlab automatically treats x as independent variable
f=fittype(@(A,x0,gamma, x) A ./ (1 +((x-x0)/gamma).^2) )

% let's see did Matlab guess fit parameters right
coeffs = coeffnames(f)

% assign initial guessed parameters
p0=[3,3,1]; % they are in the order of the appearance

% We fit our data here
[fitobject,gof] = fit(x,y, f, 'StartPoint', p0)

% it is good idea to compare fit and data visually
builtin_fit_check(x,y, fitobject)
```

Notes