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

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

- 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 $p_1, p_2, p_3, \ldots = \vec{p}$ via fitting (i.e. finding the best $\vec{p}$) of the model function which depends on $x$ and $\vec{p}$: $f(x, \vec{p})$.

In general $x$ and $y$ could be vectors i.e. multi-dimensional.

Example

- $\vec{x}$ 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 $x_i \rightarrow y_f = f(x_i, \vec{p})$
Goodness of the fit

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

Chi-squared test

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

Differences of \((y_i - y_{f_i})\) 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 \(\vec{p}\) of the model/fit function.

Our job is simple: find optimal \(\vec{p}\) 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 \( \sqrt{\frac{1}{N} \sum_{i}^{N} (y_i - y_{fi})^2} \) 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

- $\Delta p_i$ could be estimated by change of the $\chi^2$,
- $\Delta p_i$: $\chi^2(p_1, p_2, p_3, \ldots p_i + \Delta p_i, \ldots) = 2\chi^2(p_1, p_2, p_3, \ldots p_i, \ldots)$
Practical realization

Have a look at 'fitter.m' where optimization of $\chi^2$ is done with \texttt{fminsearch} \texttt{matlab} function. See 'fitter\_usage\_example.m' for a particular usage example.

$$f(x, \vec{\rho}) = \frac{A}{1 + \left( \frac{x-x_0}{\gamma} \right)^2}$$

$\vec{\rho} = [A, x_0, \gamma] = [9.9444, 1.9936, 2.0354]$
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
% [A, x0, gamma] they are in the order of the appearance
% in the above fit function definition
pin=[3,3,1];

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

disp('confidence interval/errorbars for A, x0, and gamma');
ci = confint(fitobject)

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