Chapter 1

Fitting and data reduction

1.1 Necessity for data reduction and fitting

Modern day experiments generate large amounts of data, but humans generally cannot operate simultaneously with more than a handful of parameters or ideas. As a result, very large, raw datasets become virtually useless unless there are efficient, consistent, and reliable methods to reduce the data to a more manageable size. Moreover, the sciences are inductive in nature, and ultimately seek to define formulas and equations which simulate or model the reality provided to us in raw data, making data reduction and fitting a crucial aspect of scientific research and inquiry.

In the old days when computers were not available, the most common data reduction methods were calculations of the mean and the standard deviations of a data sample. These methods are still very popular. However, their predictive and modeling powers are very limited.

Ideally, one would like to construct a model which describes (fit) the data samples with only a few free (to modify) parameters. Well-established models which are proved to be true by many generations of scientists are promoted to the status of laws of nature.

Words of wisdom

We should remember that there are no *laws*; there are only hypotheses which explain present observations and are not yet proven wrong with new data in unexplored regions. Laws of classical physics were replaced by quantum mechanics, once enough evidence was collected to highlight discrepancies between models and experiments.

The fitting is the procedure which finds the best values of the free parameters. The process of the model selection is outside of the domain of the fitting algorithm. From a scientific point of view, the model is actually the most important part of the data reduction procedure.

The outcome of the fitting procedure is the set of important parameter values, as well as ability to judge if the selected model is a good one.

1.2 Formal definition for fitting

Suppose someone measured the dependence of an experimental parameter set $\vec{y}$ on another parameter's set $\vec{x}$. 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 $\vec{x}$ and $\vec{p}$: $f(\vec{x}, \vec{p})$. In general $\vec{x}$ and $\vec{y}$ could be vectors, i.e. multi-dimensional.

Example

- $\vec{x}$ dimensions are the speed of a car and the weight of its load;
- $\vec{y}$ components are the car fuel consumption, the engine temperature, and time to the next repair.

For simplicity, **we will focus our discussion on the one dimensional case** by dropping the vector notation for x and y.

Goodness of the fit

Have a look at the fig. 1.1. We can see the data points (x_i, y_i) and a line corresponding to the functional dependence of our model with a given set of fit parameter values, i.e. the fit line. The important points of this fit line are the ones calculated at x_i: $y_{f_i} = f(x_i, \vec{p})$.

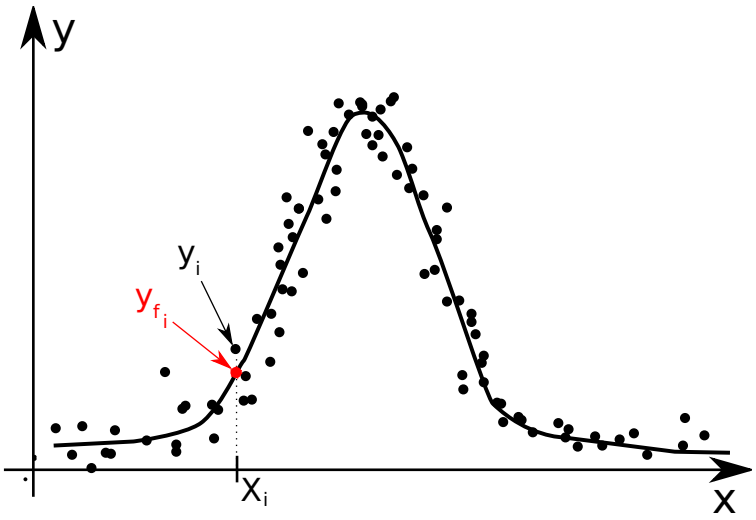

Figure 1.1: Data points and the fit line.

We need to define a formal way to estimate the goodness of the fit. One of the well established methods is the χ^2 (chi squared) test.

χ^2 test

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

Differences of $(y_i - y_{f_i})$ are called *residuals*. Thus, χ^2 is essentially summed squared distances between the data and the corresponding locations of fit curve points.

For a given set of $\{(x_i, y_i)\}$ and the model function f, the goodness of the fit χ^2 depends only on parameters vector $\vec{p}$ of the model/fit function. The job of the fitting algorithm is simple: find the optimal free parameters set $\vec{p}$ which minimizes χ^2 using any suitable algorithm, i.e. perform so-called the *least square fit*. Consequently, fitting algorithms are a subclass of the problem optimization algorithms (discussed in chapter 13).

Luckily, we do not have to worry about the implementation of the fitting algorithm, because MATLAB has the fitting toolbox to do the job. The most useful function in this toolbox is `fit`, which governs the majority of work. The fitting functions work faster than general optimization algorithms, since they can specialize on optimization of a particular quadratic functional dependence of χ^2.

1.3 Fitting example

The above material seems to be quite dry, so let's see fitting at work. Suppose we have a set of data points stored in the data file `'data_to_fit.dat'`[1] and depicted in fig. 1.2. It looks like data of a resonance contour response or a typical spectroscopic experiment.

Our first priority is to choose a model which might describe the data. Yet again, this is not a problem for the computer, but a problem for the person in charge of the data analysis. For this example, we choose the Lorentzian shape to fit the data

$$y = \frac{A}{1 + \left(\frac{x - x_o}{\gamma}\right)^2} \tag{1.1}$$

Here A is the amplitude (height) of the peak, x_0 is the position of the peak, and γ is the peak half width at half maximum level. We decide not to search for an additional background or offset term since data y-values seems to be around zero, far away from the resonance.

[1] The file is available at http://physics.wm.edu/programming_with_MATLAB_book/ch_fitting/data/data_to_fit.dat

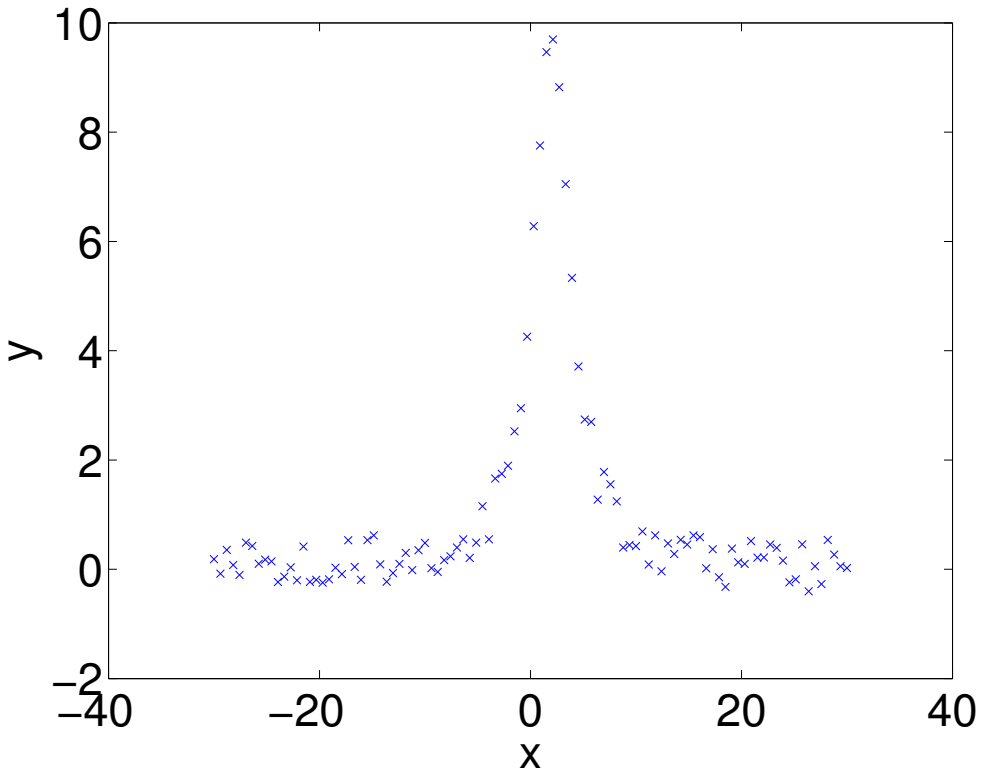

Figure 1.2: Data points

Overall, we have three free parameters to fit the data: A, x_0, and γ. The knowledge of these three parameters is sufficient to describe the experimental data, which results in the amount of data data needing storage to be drastically reduced.

The listing below demonstrates the fitting procedure

Listing 1.1: Lorentzian_fit_example.m (available at http://physics.wm.edu/programming_with_MATLAB_book/ch_fitting/code/Lorentzian_fit_example.m)

```
%% 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) );

%% 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];

%% Finally, we are ready to fit our data
[fitobject] = fit (x,y, f, 'StartPoint', pin)
```

The fitting is actually happening at the last line, where the `fit` function is executed. Everything else is the preparation of the model and data for the `fit` function. The results of the fitting, i.e. the values of the free parameters, are part of the `fitobject`. Let's have a look at them and their confidence bounds

```
fitobject =
     General model:
     fitobject(x) = A./(1+((x-x0)/gamma).^2)
     Coefficients (with 95% confidence bounds):
       A =       9.944  (9.606, 10.28)
       x0 =       1.994  (1.924, 2.063)
       gamma =       2.035  (1.937, 2.133)
```

It is a good idea to visually check the quality of the fit, so we execute

```
%% Let's see how well our fit follows the data
plot(fitobject, x,y, 'fit')
set(gca,'FontSize',24); % adjusting font size
xlabel('x');
ylabel('y');
```

The resulting plot of the data and the fit line is depicted in fig. 1.3.

1.4 Parameter uncertainty estimations

How would one estimate the confidence bounds? Well, the details of the MATLAB algorithm are hidden from us, so we do not know. But one of the possible ways would be the following. Suppose we find the best set of free parameters $\vec{p}$ which results in the smallest possible value of χ^2, then the uncertainty of i-th parameter (Δp_i) can be estimated from the following equation

$$\chi^2(p_1, p_2, p_3, \ldots p_i + \Delta p_i, \ldots) = 2\chi^2(p_1, p_2, p_3, \ldots p_i, \ldots) = 2\chi^2(\vec{p}) \tag{1.2}$$

This is a somewhat simplistic view on this issue, since the free parameters are often coupled. The proper treatment of this problem discussed in [1].

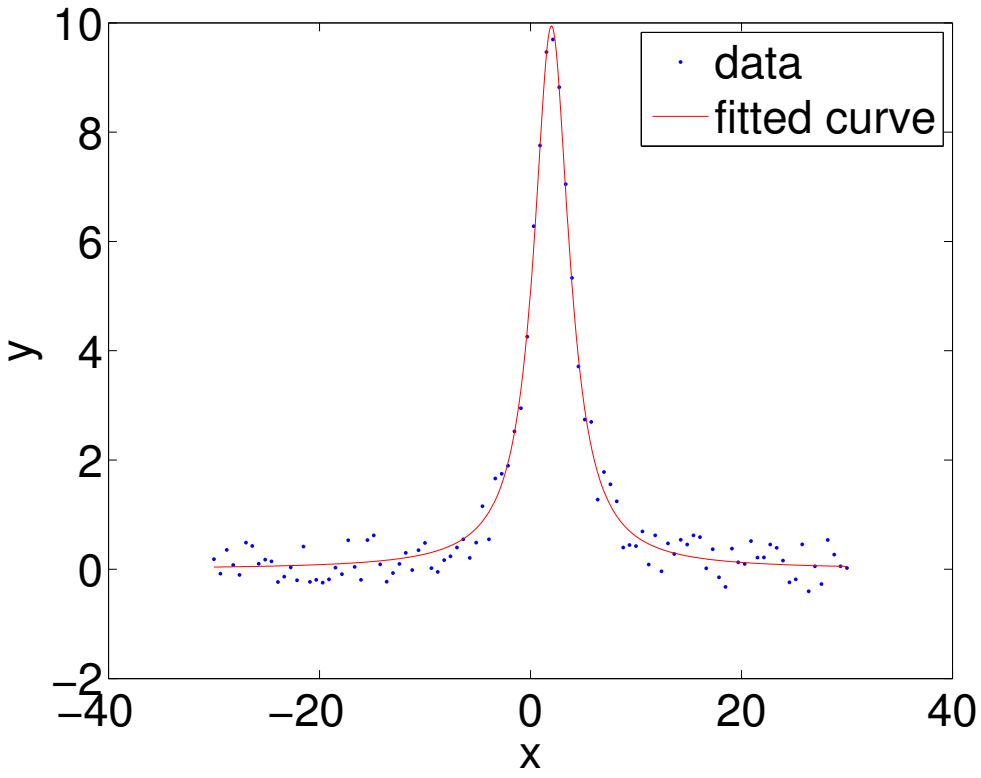

Figure 1.3: Data points and the Lorentzian fit line

1.5 Evaluation of the resulting fit

The visualization of the fit line over the data is a natural step for the fit quality assessment and it should not be skipped, but we need a more formal set of rules.

Good fits should have the following properties:

1. The fit should use the smallest possible set of fitting parameters.

    - With enough fitting parameters you can ~~fit an elephant through the eye of a needle~~ make a fit with zero residuals but this is unphysical, since *the experimental data always has uncertainties* in the measurements.

2. The residuals should be randomly scattered around 0 and have no visible trends.

3. The root mean square of residuals $\sigma = \sqrt{\frac{1}{N}\sum_i^N (y_i - y_{f_i})^2}$ should be similar to the experimental uncertainty (Δy) for y.

Words of wisdom

The above condition is often overlooked but you should keep your eyes on it. If $\sigma \ll \Delta y$ you are probably over fitting, i.e. trying to extract from the data what is not there. Alternatively, you do not know the uncertainties of your apparatus, which is even more dangerous. The σ also can give you an estimate of the experimental error bars if they are unaccessible by other means.

4. Fits should be robust: addition of new experimental points must not change the fitted parameters much.

Words of wisdom

Stay away from the higher ordered polynomial fits.

- a line is good and sometimes a parabola is also good
- use anything else only if there is a deep physical reason for it
- besides, such higher ordered polynomial fits are usually useless, since every new data point addition tends to drastically modify the fit parameters.

Equipped with these rules, we make the plot of residuals

```
%% Let's see how well our fit follows the data
plot(fitobject, x,y, 'residuals')
set(gca,'FontSize',24); % adjusting font size
xlabel('x');
ylabel('y');
```

The resulting plot of the residuals is shown in fig. 1.4. One can see that residuals are randomly scattered around zero and have no visible long term trends. Also, the typical spread of residuals is about 0.5, similar to the data point to point fluctuation, which is especially easy to eyeball on shoulders of the resonance (see fig. 1.2). Thus, at the very least, conditions 2 and 3 listed in the section 1.5 are true. We also used only 3 free parameters, so it is unlikely that we can do any better, since we need to provide parameters which govern height, width, and position of resonance. So condition 1 is also satisfied. The robustness of the fit (condition 4) can be estimated by splitting the data in two sets (for example, choosing every 2nd point for a given set) and then running the fitting procedure for each of them with following comparison of the resulting fit parameters for each of the sets. This exercise is left to the reader.

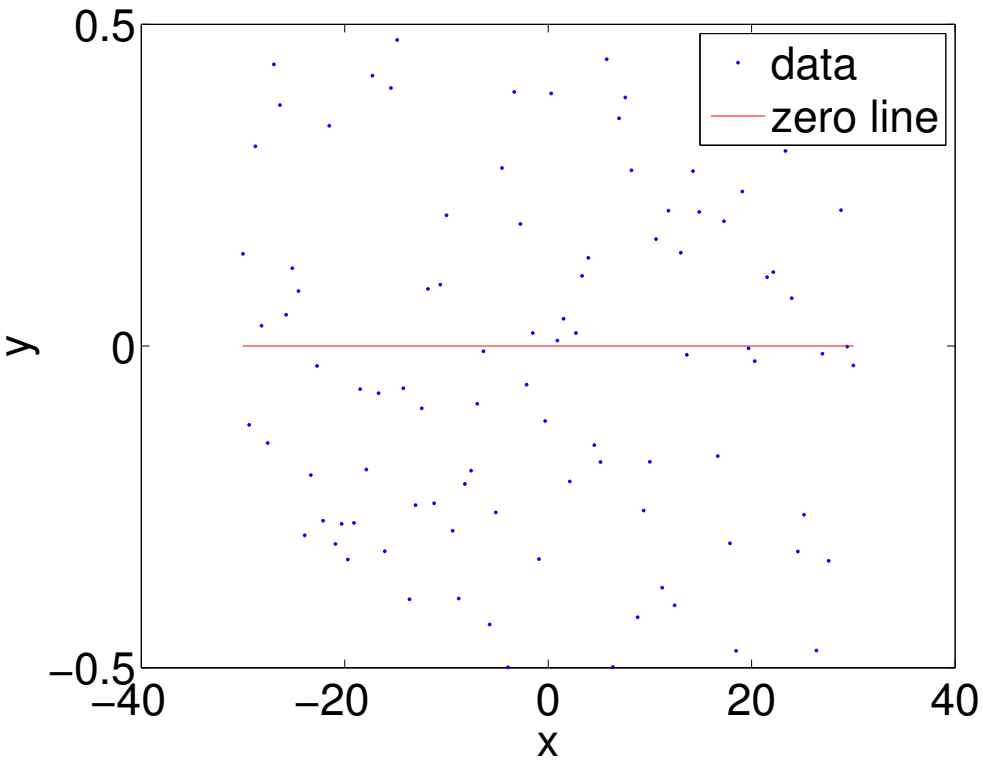

Figure 1.4: Plot of residuals of the Lorentzian fit

1.6 How to find the optimal fit

Fitting in general is not able to find the best parameter set. It finds only the one which guarantees the local optimal. We will talk more about it in the optimization chapter (see chapter 13). Such local optimum might result in an awful fit. As a result, you can often hear that the fitting is an art or even witchcraft. As always, what is actually meant is that the person does not understand the rules under which the fitting algorithm operates and has a futile hope that random tweaks in the initial guess would lead to the success, i.e. that a computer will magically present the proper solution. In this section, we will try to move from the domain of witchcraft to the domain of reliable methods.

The key to success is in the proper choice of the starting guess.

Words of wisdom

It is very naive to hope that the proper initial guess will be a result of a random selection. The fit algorithm, indeed, performs miraculously well even with a bad starting point. However, one needs to know which fit parameter governs a particular characteristic of the fit line. Otherwise, you can look for a good fit for the rest of your life. After all, computers are only supposed to assist us, not to think for us.

Let me first demonstrate how a bad fit guess leads nowhere. I will modify only one parameter: the initial guess `pin=[.1,25,.1]`, in the following listing otherwise identical to the listing 1.1.

Listing 1.2: bad_Lorentzian_fit_example.m (available at http://physics.wm.edu/programming_with_MATLAB_book/ch_fitting/code/bad_Lorentzian_fit_example.m)

```
%% 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) );

%% assign initial guessed parameters
% [A, x0, gamma] they are in the order of the appearance
% in the above fit function definition
pin=[.1,25,.1]; % <---------------- very bad initial guess!

%% Finally, we are ready to fit our data
[fitobject] = fit (x,y, f, 'StartPoint', pin)
```

The resulting fit is shown in fig. 1.5. It is easy to see that the fitted line has no resemblance with the data points overall, except just around $x = 25$ where it passes exactly through 2 data points. The optimization algorithm was locked in the local maximum, which resulted in the bad fit.

Words of wisdom

Usually, the most critical fit parameters are the ones which govern narrow features in the x-space.

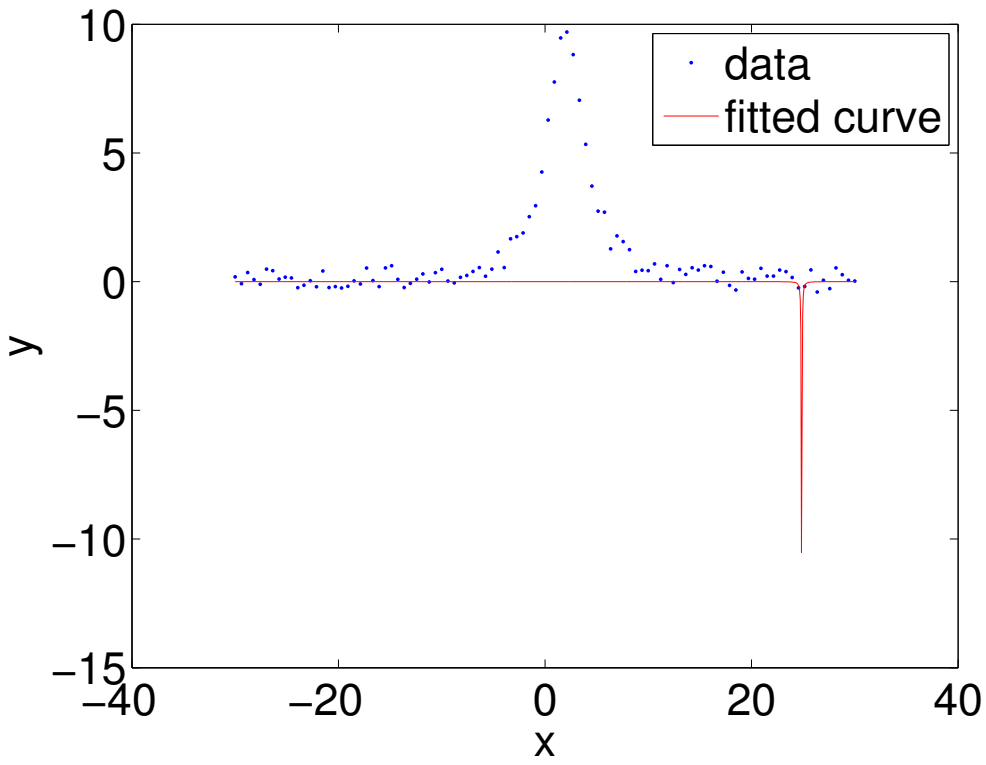

Figure 1.5: Fitting result with the bad starting point

Proper fitting procedure

1. Plot the data.
2. Identify the model/formula which should describe the data. This is outside of the computer's domain.
3. Identify which fit parameter is responsible for a particular fit line feature.
4. Make an *intelligent* guess of the fit parameters based on above understanding.
5. Plot the fit function with this guess and see if it is up to your expectation.
6. Refine your guess and repeat the above steps, until you get a model function curve reasonably close to the data.
7. Finally, ask the computer to do tedious refinements of your guessed parameters, i.e. execute the `fit`.
8. The fit will produce fit parameters with confidence bounds; make sure you like what you see.

As you can see, the most important steps are performed *before* execution of the `fit` command.

1.6.1 Example: Light diffraction on a single slit

The following example should help us see the proper fitting procedure.

Plotting the data

Someone provided us with data which has the sensor response (I values) vs its position (x values) along the diffraction pattern on a screen of the light passed through a single slit. The plot of the data stored in the file `'single_slit_data.dat'`[2] is depicted in fig. 1.6.

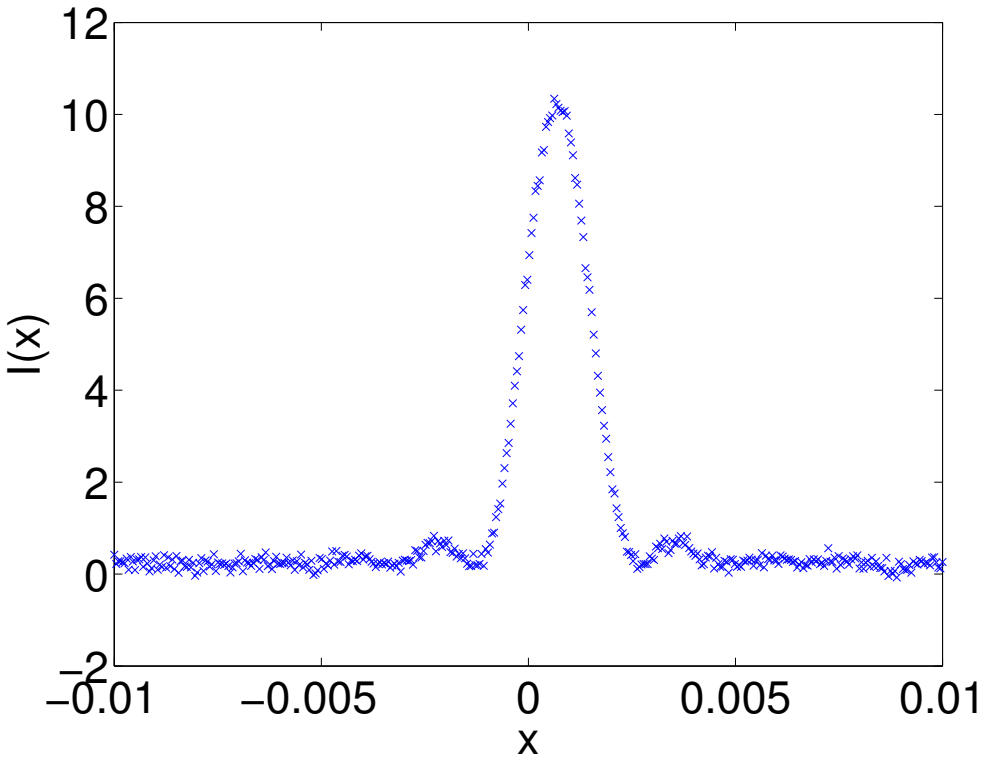

Figure 1.6: Single slit diffraction intensity pattern data

Choosing the fit model

According to the wave theory of light, the detected intensity of light is given by

$$I(x) = I_0 \left(\frac{\sin\left(\frac{\pi d}{l\lambda}(x - x_0)\right)}{\frac{\pi d}{l\lambda}(x - x_0)} \right)^2 \tag{1.3}$$

[2]The file is available at http://physics.wm.edu/programming_with_MATLAB_book/ch_fitting/data/single_slit_data.dat

here d is the slit width, l is distance to the screen, λ is the light wavelength, and x_0 position of the maximum. This, however, assumes that our detector is ideal and there is no background illumination. In reality, we need to take into account the offset due to such background (B), which modifies our equation to a more realistic form

$$I(x) = I_0 \left(\frac{\sin\left(\frac{\pi d}{l\lambda}(x - x_0)\right)}{\frac{\pi d}{l\lambda}(x - x_0)} \right)^2 + B \tag{1.4}$$

The first obstacle for the fitting of our model is in the term

$$\alpha = \frac{\pi d}{l\lambda} \tag{1.5}$$

The d, l, λ are linked, i.e. if someone increases d by 2, we would be able to get the same values of $I(x)$ by increasing either l or λ by 2 to maintain the same ratio. Thus, any fit algorithm will never be able to decouple these 3 parameters from the provided data alone. Luckily, the experimentators, who knew about such things, did a good job and provided us with values for $l = .5$ m and $\lambda = 800 \times 10^{-9}$ m. So by learning the resulting fit parameters, we would be able to tell the slit size d. For now, let's express our formula for intensity with α. The formula looks simpler and requires fewer fit parameters

$$I(x) = I_0 \left(\frac{\sin\left(\alpha(x - x_0)\right)}{\alpha(x - x_0)} \right)^2 + B \tag{1.6}$$

We offload its calculation to the function `single_slit_diffraction` with the following listing

Listing 1.3: single_slit_diffraction.m (available at http://physics.wm.edu/programming_with_MATLAB_book/ch_fitting/code/single_slit_diffraction.m)

```
function [I] = single_slit_diffraction(I0, alpha, B, x0, x)
    % calculates single slit diffraction intensity pattern on a screen
    % I0 - intensity of the maximum
    % B - background level
    % alpha - (pi*d)/(lambda*l), where
    %  d - slit width
    %  lambda - light wavelength
    %  l - distance between the slit and the screen
    % x - distance across the screen
    % x0 - position of the intensity maximum

    xp = alpha*(x-x0);
    I = I0 * ( sin(xp) ./ xp ).^2  + B;
end
```

Making an initial guess for the fit parameters

Now let's work on an initial guess for the fit parameters. The B value is probably the simplest; we can see from eq. (1.6) that, as x goes to very large values, the first oscillatory term drops (it is $\sim 1/x^2$) and equation is dominated by B. On other hand, we see that far edges on fig. 1.6 are above 0 but below 1, so we assign some value in between `B_g=0.5` as an initial guess. According to the above definitions, I_0 is the maximum of intensity, disregarding the small B contribution, so we set I_0 guess as `I0_g = 10`. Similarly, x_0 is the position of the maximum, so our guess for x_0 (`x0_g=.5e-3`) seems to be a reasonable value, since peak is to the right of 0 but to the left of `1e-3`. The α value is the trickiest one. First, we recognize that the squared expression in the parentheses of eq. (1.6) is the expression for the *sinc* function. This function is oscillating because of the sin in the numerator; and the amplitude of the oscillation is decreasing with the growth of x since it sits in the denominator. Importantly, its first location for crossing background line (x_b) is at the point where $\sin(\alpha(x_b - x_0)) = 0$. So $\alpha(x_b - x_0) = \pi$, looking at fig. 1.6, we eyeball that $x_b \approx 0.002$. Thus, a good guess for α is `alpha_g = pi/(2e-3 - x0_g)`.

Plotting data and the model based on the initial guess

We are ready to see if our intelligent guess is any good. Let's plot the model function with our initial guess values

Listing 1.4: plot_single_slit_first_guess_and_data.m (available at http://physics.wm.edu/programming_with_MATLAB_book/ch_fitting/code/plot_single_slit_first_guess_and_data.m)

```
% load initial data file
data=load('single_slit_data.dat');
x=data(:,1); % 1st column is x
y=data(:,2); % 2nd column is y

% _g is for guessed parameters
B_g=0.5;
I0_g=10;
x0_g=.5e-3;
alpha_g = pi/(2e-3 - x0_g);

% we have a liberty to choose x points for the model line
Nx= 1000;
xmodel = linspace(-1e-2, 1e-2, Nx);
ymodel = single_slit_diffraction( I0_g, alpha_g, B_g, x0_g, xmodel);
plot(x,y,'bx', xmodel, ymodel, 'r-');
legend({'data', 'first guess'});
```

```
set(gca,'FontSize',24);
xlabel('x');
ylabel('I(x)');
```

The result is shown in fig. 1.7. It is not a perfect match but pretty close.

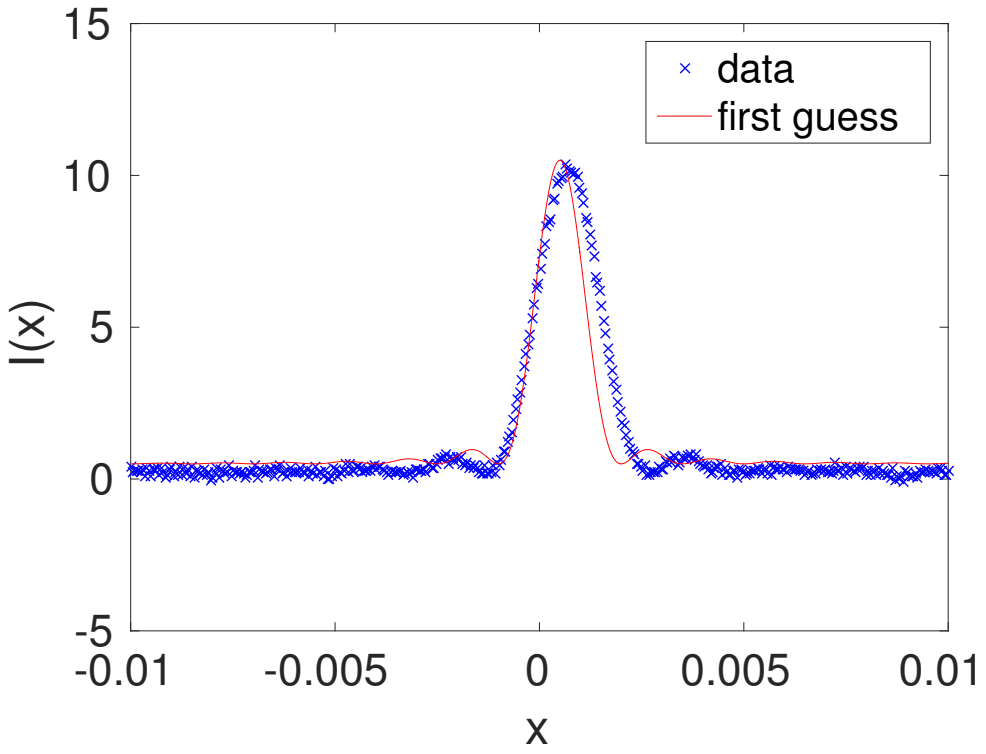

Figure 1.7: Single slit diffraction intensity pattern data and initial guessed fit

Fitting the data

Now, after all of *our* hard work, we are ready to ask MATLAB to do the fitting

Listing 1.5: `fit_single_slit_data.m` (available at http://physics.wm.edu/programming_with_MATLAB_book/ch_fitting/code/fit_single_slit_data.m)

```
% load initial data file
data=load('single_slit_data.dat');
x=data(:,1); % 1st column is x
y=data(:,2); % 2nd column is y

%% defining fit model
```

```
f=fittype(@(I0, alpha, B, x0, x) single_slit_diffraction( I0, alpha, B, x0,
   x) );

%% prepare the initial guess
% _g is for guessed parameters
B_g=0.5;
I0_g=10;
x0_g=.5e-3;
alpha_g = pi/(2e-3 - x0_g);

% pin = [I0, alpha, B, x0] in order of appearance in fittype
pin = [ I0_g, alpha_g, B_g, x0_g ];

%% Finally, we are ready to fit our data
[fitobject] = fit (x,y, f, 'StartPoint', pin)
```

The `fitobject` is listed below

```
fitobject =
     General model:
     fitobject(x) = single_slit_diffraction(I0,alpha,B,x0,x)
     Coefficients (with 95% confidence bounds):
       I0 =       9.999  (9.953, 10.04)
       alpha =        1572  (1565, 1579)
       B =      0.1995  (0.1885, 0.2104)
       x0 =   0.0006987  (0.0006948, 0.0007025)
```

The resulting fit and its residuals are shown in fig. 1.8 and generated with

Listing 1.6: `plot_fit_single_slit_data.m` (available at http://physics.wm.edu/programming_with_MATLAB_book/ch_fitting/code/plot_fit_single_slit_data.m)

```
%% plot the data, resulting fit, and residuals
plot(fitobject, x,y, 'fit','residuals')
xlabel('x');
ylabel('y');
```

The residuals are scattered around zero, which is the sign of the good fit.

Evaluating uncertainties for the fit parameters

The `fitobject` provides parameters' values and confidence bounds, so we can estimate the fit parameters' uncertainties or error bars. We can calculate the width of the slit $d = \alpha l \lambda / \pi$ and its uncertainties below.

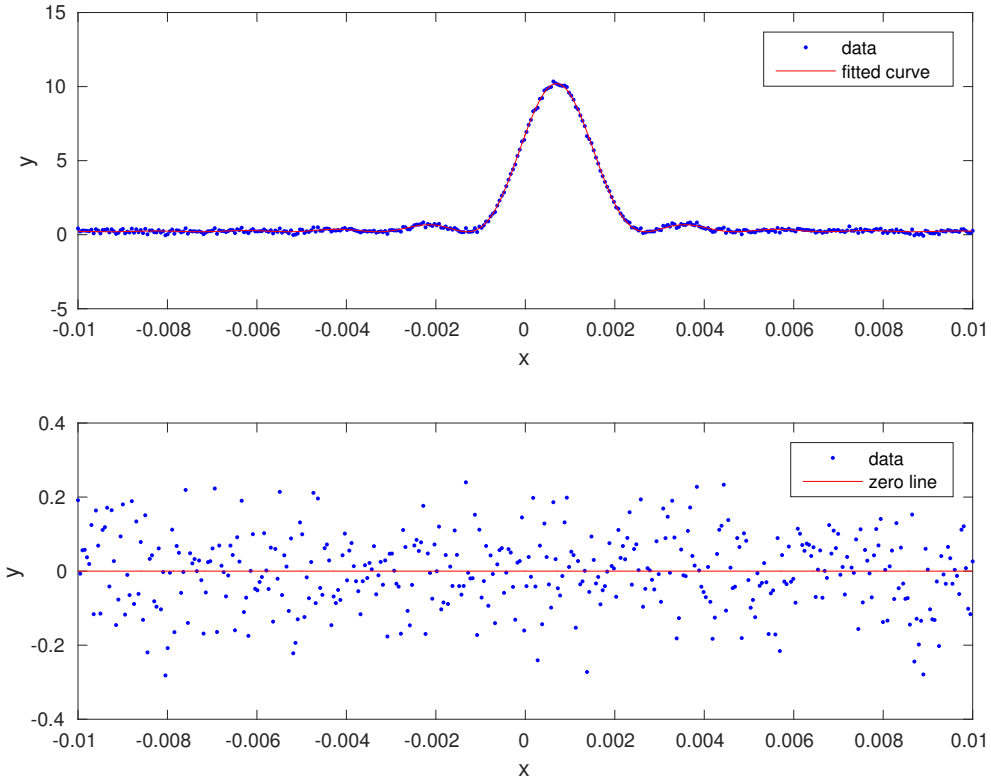

Figure 1.8: Single slit diffraction intensity pattern data and fit, as well as the fit residuals

```
%% assigning values known from the experiment
l = 0.5; % the distance to the screen
lambda = 800e-9; % the wavelength in m

%% reading the fit results
alpha = fitobject.alpha; % note .alpha, fitobject is an object
ci = confint(fitobject); % confidence intervals for all parameters
alpha_col=2; % recall the order of appearance  [I0, alpha, B, x0]
dalpha = (ci(2, alpha_col) - ci(1,alpha_col))/2; % uncertainty of alpha

%% the width related calculations
a = alpha*l*lambda/pi; % the slit width estimate
da = dalpha*l*lambda/pi; % the slit width uncertainty
```

```
a=2.0016e-04
da=9.2565e-07
```

In particular for the slit width, MATLAB provides way too many digits; they make no sense with the above estimation of the uncertainties (`da`). We have to do the *proper rounding* ourselves; the slit width is $a = (2.002 \pm 0.009) \times 10^{-4}$ m. Our error bar is less than 0.5%, which is quite good.

Words of wisdom

Take the fit uncertainties with a ~~grain~~ pound of salt. After all, the algorithm has no knowledge about the experiment. It is possible that your data set was very favorable for the particular value of a given free parameter. Run your experiment again several times and see new estimates of the free parameters, compare to the old values, and then decide about parameters uncertainties.

1.7 Self-Study

- Review the function handle operator `@`: use `help function_handle`.
- Pay attention to error bars/uncertainties; **report them**.
- Use built-in `fittype` to define a fitting function with the following call to `fit` to do the fitting.

Problem 1: Recall one of the problems from section 4.6.

Download data file `'hwOhmLaw.dat'`[3]. It represents the result of someone's attempt to find the resistance of a sample via measuring voltage drop (V), which is the data in the 1st column, and current (I), listed in the 2nd column, passing through the resistor. Judging by the number of samples, it was an automated measurement.

Using Ohm's law $V = RI$ and a linear fit of the data with one free parameter (R) find the resistance (R) of this sample. What are the error bars/uncertainty of this estimate? Does it come close to the one which you obtained via the method used in section 4.6? Do not use the fitting menu available via the GUI interface; use a script or a function to do it.

Problem 2: You are making a speed detector based on the Doppler effect. Your device detects dependence of the signal strength vs. time, which is recorded in the data file `'fit_cos_problem.dat'`[4]. The first column is time and the second is the signal strength.

Fit the data with

$$A\cos(\omega t + \phi)$$

[3]The file is available at http://physics.wm.edu/programming_with_MATLAB_book/ch_fitting/data/hwOhmLaw.dat

[4]The file is available at http://physics.wm.edu/programming_with_MATLAB_book/ch_fitting/data/fit_cos_problem.dat

where A, ω and ϕ are the amplitude, the frequency and the phase of the signal, and t is time.

Find fit parameters (the amplitude, the frequency and the phase of the signal) and their uncertainties.

Problem 3: This is for the physicists among us. Provided that the above radar was using radio frequency, could you estimate the velocity measurement uncertainty? Is it a good detector to measure a car's velocity?

Problem 4: Here's an experiment to do at home. Make a pendulum of variable length (0.1m, 0.2m, 0.3m, and so on up to 1m). Measure how many round trip (back and forth) swings the pendulum does in 20 seconds with each particular length (clearly you will have to round to the nearest integer). Save your observations into the simple text file with 'tab' separated columns. The first column should be the length of the pendulum in meters and the second column should be the number of full swings in 20 seconds.

Write a script which loads this data file and extract acceleration due to gravity (g) from the properly fitted experimental data. Recall that the period of the oscillation of a pendulum with the length L is given by the following formula

$$T = 2\pi\sqrt{\frac{L}{g}}$$

Problem 5: In optics, the propagation of laser beams is often described in the Gaussian beam's formalism. Among other things, it says that the optical beam intensity cross section is described by the Gaussian profile (hence, the name of the beams)

$$I(x) = A\exp\left(-\frac{(x-x_o)^2}{w^2}\right) + B$$

where A is the amplitude, x_o is the position of the maximum intensity, w is the characteristic width of the beam (width at $1/e$ intensity level), and B is the background illumination of the sensor.

Extract the A, x_o, w, and B with their uncertainties from the real experimental data contained in the file `'gaussian_beam.dat'`[5], where the first column is the position (x) in meters and the second column is the beam intensity in arbitrary units.

Is the suggested model describe the experimental data well? Why?

Problem 6: Fit the data from the file `'data_to_fit_with_Lorenz.dat'`[6] with the above Gaussian profile. Is the resulting fit good or not? Why? Compare it to the Lorentzian model (see eq. (1.1)).

[5]The file is available at http://physics.wm.edu/programming_with_MATLAB_book/ch_fitting/data/gaussian_beam.dat

[6]The file is available at http://physics.wm.edu/programming_with_MATLAB_book/ch_fitting/data/data_to_fit_with_Lorenz.dat

Bibliography

[1] P. R. Bevington. *Data reduction and error analysis for the physical sciences*. New York, McGraw-Hill, 1969.