# Data treatment in the muon lab

Martin Ljunggren

May 7, 2012

## 1 Data treatment

One thing to notice about the data is that the first channels are empty (0 counts). This is due to a limitation in the electronics so that only values above a certain time can be measured. Naturally, we don't want to take these channels into account since we know that they do not contain any data of interest. Therefore, we simply remove these channels. It is important for the calibration however, that the numbering stays the same. For example, if the number of removed channels is 3 and the original numbering started with channel 1, we must now start counting from channel 4 in order to be able to calibrate with the values we have.

Due to effects of the finite binning, it is also likely that the information in the first channel containing data is misrepresentative. Since we do not know exactly where the low limit of times that we can measure lies, we are likely better off if we discard also this channel. For example, if the considered channel contains counts between times $t$ and $t + \Delta t$ and the shortest time that we can measure lies somewhere in between, the number of counts that we will get in this bin is smaller than the "true" value.

## 2 Chi-square test

When fitting a function to observed data, the least squares technique is a common approach. Given the expression

$$S = \sum \frac{(f(x_i) - y_i)^2}{\sigma^2} \tag{1}$$

the best fit is considered to be the one that minimizes S. For anything else than a linear function $f$, the parameters that minimize f are found numerically. The distance between the fitted function and the data points, $f(x_i) - y_i$, is called a *residual*. Since the data points might be of varying quality it is also reasonable to assign a weight to them. Using the inverse the error as weight should give a more reliable fit. In our case it is common to use the square root of the number of measurements for the data point so it would be $\sigma = \sqrt{(n)}$ where $n$ is the number of counts in a certain channel. If the number of counts in the bin is zero, simply set the error to 1.

If our measurement was perfect, the function would fit perfectly with the data and all residuals and therefore also S, would be zero. Due to experimental errors this will not be the case.

An estimation of the "goodness of fit" can be obtained by using a chi-square test. The chi-square statistic is given by

$$\chi^2 = \sum \frac{(f(x_i) - y_i)^2}{\sigma^2} \tag{2}$$

The so called reduced chi-square $\frac{\chi^2}{N-d}$ where $N$ is the number of data points and d is the number of degrees of freedom (the number of parameters, 3 in this case) in the fit, gives an estimation of the fit quality. Simplified, a good rule of thumb is that if this value is much larger than one, the fit function is probably wrong or the estimated errors are too small. If it is much less than one, the fit is "too good" which for example could mean that the errors are overestimated. Note that this is only an estimate of the fit quality, if it is probable that the measured data follows the assumed distribution. It is not en estimate of the error in the measured parameters (e.g. the muon lifetime).

## 3 Useful Matlab commands

- For-loops are written as

  ```
  for i = 1:10
     x = x+1;
  end
  ```

  where in this example the statement between for and end will be executed 10 times.

- For fitting a polynomial of a certain degree, *polyfit* is a good choice. Given the desired degree and input in the form of two vectors it returns the fit parameters.

  ```
  p = polyfit(x,y,1)
  ```

  returns the parameters of a first degree polynomial fitted to the points contained in x and y, where p(1) is the slope and p(2) the intersect

- To operate on vectors element by element use a dot (.) in front of the normal operator. For example, the code

  ```
  x = [1 2 3 4]
  y = [4 5 6 7]
  m = x.*y
  ```

  produces a vector $m$ = [4 10 18 28]. Without the dot, a normal matrix multiplication will be performed. This is of course not possible in this case.

- To define a function, write

  ```
  func = @(param)  func(param)
  ```

  Example:

```
function = @(param) param(1) + param(2)
function([3 6])
```

This will produce the output 9, i.e. the sum of 3 and 6, when initialized with the parameters 3 and 6.

We can also write

```
x = [1 2 3 4]
func = @(param) param(1)*x + param(2)
m = func([2 1])
```

producing the vector m = [3 5 7 9] given the input parameters 2 and 1.

- To find the parameters that minimize a function, *fminsearch* can be used.

```
minparam = fminsearch(f, ipar)
```

will minimize the function f with using the initial parameters in *ipar* and put the resulting parameters in *minparam*.
Simple example: Fit function $a \cdot x^2 + b$ to data using *fminsearch*:

```
x = [1 1.5 2 2.5 3 3.5 4 4.5 5 5.5 6 6.5 7]                    % define two vectors x and y
y = [3.1 4.35 6.16 8.3 11.2 14.19 18.5 22.15 26.3 32 36.9 44.1 50.8]
func = @(par) sum (( par(1)*x.ˆ2 + par(2) - y(1,:)).ˆ2)       % define function func =
```
$\Sigma(a \cdot x_i^2 + b - y_i)^2$
```
minpar = fminsearch(func, [1 1])                          % minimize func with respect to a
f = minpar(1) * x.ˆ2 + minpar(2)       %fit parameters are returned in vector minpar
hold on
plot (x,f)                                                              % plot the result
plot(x,y,'*')
```