LAB SESSION 2
Plots and histograms

CECAM School 2019

June 11, 2019

1 gnuplot

Gnuplot is an open source software for plotting and data visualization.
It might not be the fanciest graphing software available, although it is very customisable, but it is a very powerful tool for rapid and efficient data analysis, especially when used in coordination with scripting languages for data manupulation such as bash and awk.

Exercise 1

Here are some basic commands for a start:

- open gnuplot in your Unix shell
- try plotting some functions of variable `x`, e.g.: `plot x**2 + 1` for a parabola
- define function
- plot 2 Gaussians with different labels
- test
- change color, lw, lt
- zoom on the console, autoscale
- set range, set range restore, range only on one side
- p [:][:]
- set term, set out (eps, png), test
- gnuplot script (from shell with < a.gp, from shell with «-EOF or from gnuplot with command load)
- plot data series from file *linear_series.dat* with points, with lines, with linepoints, with errorbars
- fit
- use stdout as gnuplot input

2 Histograms

Suppose I have a set of data in a 1- or N-dimensional space and I would like to understand how they are distributed along these 1 or N coordinates. The natural way to do it is to build a histogram: divide each space direction into intervals and form this way a spacial grid, that delimits portions of space called bins; then assign every datapoint to a bin and at the end of this process count for each bin how many points have fallen into it. The result is a discrete map defined on a N dimensional grid that can also be the starting point to give a statistically significant estimate of the underlying distribution from which data are sampled. In fact, the assumption that all data are sampled from the same distribution allows to estimate the error (standard deviation) of each bin count n_i of bin i as: $\sigma_i = \sqrt{n_i}$.

2.1 Building a histogram

2.1.1 Exercise: a first histogram

Plot data from file *time-series.dat* using first two columns: this is a typical time series in the presence of a rare transition with two metastable states.

Now build a histogram to see how data in column 2 are distributed. You should write a small code that reads from standard input a column of values and prints on standard output a 2-column file containing bin centers and corresponding count of data in that bin. You should try doing it in C++, since this piece of code will be used in future lab sessions.

Ingredients for your code are:

- read from stdin all values and store them into a vector;
- fix min and max values of histograms and number of bins, then compute the mesh (bin width) of your histogram;
- allocate a vector of *nbins* integers for bin counts and initialize all elements to 0;
- go through every datapoint, check which bin it is in and increment count for the corresponding bin;
- print to stdout for each bin the values of bin center and bin count

By redirecting stdout to a file you can easily plot your histogram in gnuplot, e.g.:

```
p 'histo.dat' u 1:2 w boxes fill p 2 notitle, " u 1:2:(sqrt($2)) w yerr t 'counts'
```

If you have more time:

2.1.2 Bonus Exercise: choose histogram domain automatically

Instead of setting the minimum and maximum values of your histogram's domain manually, you could add to your code some lines to look for the minimum and maximum value in your data and parametrize your histogram based on these.

2.1.3 Bonus Exercise: avoid zero counts

If the number of bins you set is too high some bins could end up being empty at the end of the count. This could give problems e.g. when taking the logarithm of histogram counts (useful e.g. in free energy calculation as you will see). Moreover if bins in your histogram are little populated the statistical error on the counts is comparable to the value of the count itself, making statistical inference based on it less roubust: it is in general preferable to have a mesh which is a bit sparser but with lower relative error on the counts. You can try to modify your code so that the number of bins is chosen adaptively:

- start from a very high default number of bins `nbinmax`
- build a histogram with this first mesh
- check that all bin counts are nonzero
- if there is at least a bin whose count is 0 then set the number of bins to a fraction of the previous (e.g.: `nbinmax *= 0.9`)

2.2 Estimating probability density function of data

Suppose we have a set of data that are all independently sampled from the same distribution: what in statistics is called a set of independent identically distributed random variables. How can we give an estimate of their probability density function starting from a counts histogram?
If we divide the number of counts n_i^* by the total number of data in the histogram N we obtain the so-called **frequency histogram** (or height-normalized histogram):

$$p_i^* = n_i^*/N$$

which is nothing more than the frequentist probability of a datapoint falling in bin i. Since $\sum_i n_i^* = N$ it is clear that $\{p_i^*\}_i$ is a discrete probability distribution and is normalized to $1 = \sum_i p_i^*$. p_i^* is a statistical estimator of the integral of the probability distribution on the bin domain:

$$p_i^* \sim \int_{x_i^{min}}^{x_i^{max}} f(x)\ \mathrm{d}x$$

Therefore a possible estimate of the underlying probability density function of our data is a stepwise function built as:

$$f^*(x) = p_i^*/\Delta_i \quad \forall\ x_i^{min} < x < x_i^{max}$$

where Δ_i is the width of bin i, so that the subtended area for of f^* over each bin i is equal to p_i^*. The histogram built with all the $f_i^* = p_i^*/\Delta_i$ is the so-called **frequency density histogram** (or area-normalized histogram). Notice that once the histogram is area-normalized it is no more necessary to keep all bins equally wide and the shape of the estimated p.d.f. will remain the same up to discretizarion error. It can therefore make sense to increase bin width where data are rare, though keeping a tight mesh where they are abundant not to lose accuracy.

2.2.1 Exercise: estimating p.d.f. with linear interpolation

- modify your previous code so that it also prints the frequency and the frequency density histograms with the correct error;
- plot in gnuplot to visualize: of course the shape of the histogram is not changed, since all bin widths are the same
- a first improvement for estimating the p.d.f. instead of with a stepwise function is a linear interpolation, i.e. a function $f^*_{lin}(x)$ that connects all the f^*_i with lines; for the first (last) bin use a virtual 0-th ($nbins + 1$-th) bin with value $f^*_0 = 0$ ($f^*_{nbins+1} = 0$) ; $f^*_{lin}(x)$ can be considered identically 0 before (after) this point;
- on standard output now print x, a column with $f^*_{lin}(x)$ and one with its derivative sampled on a grid 10 times denser than the bins; you can decide whether to compute $f^*_{lin}(x)$ and its derivative directly in the main when printing or you can define outside the main a real (`double`) function that gives the value of $f^*_{lin}(x)$ for each real value of x e.g taking as arguments x and a vector with all the f^*_i's;

2.2.2 Bonus Exercise: histogram on a multidimensional space

- plot data from file *time-series.dat* using column 1 and 3; how is it different from the previously considered time series?
- build a histogram from column 3;
- the file is actually a 2 dimensional sample, you can get an idea about how data are distributed by plotting column 2 vs column 3 (a suggestion is to plot one point every 10); it looks like there are 3 peaks, so there is not only one single rare transition, but 3 in our system; marginalization to only one dimension gives only very partial information on the system;
- modify your code to take 2 columns as input end build a 2-dimensional count histogram
- you can visualize the histogram as heatmap

2.2.3 Exercise: estimating p.d.f. parametrically

When we expect the p.d.f. underlying our data to have a particular functional form we can estimate it by fitting it to the frequency density varying a suitable set of parameters, as in the following exercise.

- compute the frequency density histogram of data from file *sample.dat* and print it in output;
- plot the histogram using gnuplot and verify qualitatively that data are normally distributed;
- define in gnuplot a Gaussian function with the suitable free parameters;
- fit this functional form to the frequency density histogram (do not forget errorbars);
- a Gaussian distribution is fully parametrized by its mean and standard deviation; compute these two quantities from the data using awk and see how they compare to gnuplot fit;

2.2.4 Bonus Exercise: cumulative probability density function