Advanced Methods on Computational Physics

Dr. Movahed

Exercise Set 3

Pooria Dabbaghi

98416029

General Notes:

Each directory px is for each section of the exercise and contains the source code "px.cpp" and required data files to run and a "resultx.zip" that contains executable file of each program ("main.out") and other harvested results from running it. All programs use armadillo library for linear algebra and matplot library for c++ that is a c++ interface to python matplot library; Therefor to build the programs from source the system should have armadillo package and python-dev tools installed and "matplotlibcpp.h" file in the current directory or the gcc path; then programs can be compiled with the command

```
g++ -g px.cpp -o main.out -O2 -L/path/to/larmadillo -larmadillo -I/path/to/python2.7 -lpython2.7
```

programs p2 and p3 call the system's gnuplot command to plot some data from file.

1. To fit the curve $y = ax^H$ to the data from fitinput.txt, X^2 considered to be the error that need to be minimized

$$X^2 = \frac{1}{2N}\sum_{i=1}^{N}(ax_i^H - y_i)^2$$

The best parameters of ax^H to fit to data are the parameters that minimize X^2 therefor mathematically

$$\frac{\partial X^2}{\partial a} = \frac{1}{N}\sum_{i}^{N}(ax_i^H - y_i)x_i^H = 0 \quad (1)$$

$$\frac{\partial X^2}{\partial H} = \frac{1}{N}\sum_{i}^{N}(ax_i^H - y_i)ax_i^H \ln(x_i) = 0 \quad (2)$$

Although a_{best} can be obtained from (1) , we're not able to obtain a straightforward equation for H_{best}. So a phase space has been created with $-10 \le a \le 10$ and $-3 \le H \le 3$ with $\Delta a = 0.1$ and $\Delta H = 0.01$ and for each a *and* H the value of X^2 been calculated and concretely for X^2_{min} , a_{best} and H_{best} has been derived.

To compute the errors of a and H from error propagation for the function $f(x, y)$ the error is

$$\sigma_{mf}^2 = \sigma_{mx}^2 < \left(\frac{\partial f(x,y)}{\partial x}\right)^2 > + \sigma_{my}^2 < \left(\frac{\partial f(x,y)}{\partial y}\right)^2 >$$

So for a and H error propagation from x and y is

$$\sigma_{ma}^2 = \sigma_{mx}^2 < \left(\frac{\partial a}{\partial x}\right)^2 > + \sigma_{my}^2 < \left(\frac{\partial a}{\partial y}\right)^2 >$$
$$\sigma_{mH}^2 = \sigma_{mx}^2 < \left(\frac{\partial H}{\partial x}\right)^2 > + \sigma_{my}^2 < \left(\frac{\partial H}{\partial y}\right)^2 >$$

Since $y = ax^H$

$$a = \frac{y}{x^H} \Rightarrow \begin{matrix} \frac{\partial a}{\partial x} = -\frac{yH}{x^{H+1}} \\ \frac{\partial a}{\partial y} = \frac{1}{x^H} \end{matrix}$$

$$H = \frac{\ln\frac{y}{a}}{\ln x} \Rightarrow \begin{matrix} \frac{\partial H}{\partial x} = -\frac{\ln\frac{y}{a}}{x\,(\ln x)^2} \\ \frac{\partial H}{\partial y} = \frac{1}{y \ln x} \end{matrix}$$

Then using these relations standard deviation of H_{best} and a_{best} has been calculated in p1.cpp and saves $a_{best}, H_{best}, error(a_{best})\ and\ error(H_{best})$ to file 'sph.txt' and saves the diagram in 'fitinput.png'.

2. The armadillo library has functions that generate random numbers with defined distributions. Function $randu(n,m)$ generates a $n \times m$ matrix of random elements with uniform distribution and function $randn(n,m)$ generates a $n \times m$ matrix of random elements with normal distribution (Gaussian distribution). Therefor in program p2 two vectors with 1000 elements has been produced using this two functions and PDF of these two vectors have been calculated with two methods using simple kernel function. Function $slowpdf()$ takes these two vectors and according to minimum and maximum of the vectors creates a class vector with $xmin{:}\ dx{:}\ xmax$ and loops through each vector and counts the number of elements that belong to each class. Function $fastpdf()$ instead of classify data with dx works with integers and converts data to integers by $k_i = int(\frac{\xi_i}{dx})$ and then creates a map that has integer classes between $k_{min}\ and\ k_{max}$ as keys and number of members of that class as values. Then by looping through k_i it counts the number of members of each class. P2 saves the calculated PDF of gaussian distributed random in 'gaussian.txt' and PDF of uniformly distributed random in 'uniform.txt' and then saves the plot of $fastpdf()$ function in 'randompdf-fast.png' and plot of $slowpdf()$ function to 'randompdf-slow.png'.

3. The algorithm to compute PDF here is pretty much the same as p2.cpp bust here functions take dx and $name$ to compute different PDF for each dx; The outputs are "dxi.txt" that contains x classes and pdf for each dx ($dx_1 = 0.1, dx_2 = 0.01\ and\ dx_3 = 0.001$) and "dxi.png" that is the plot of PDFs and "all.png" that is the plot of all dx1, dx2, dx3.

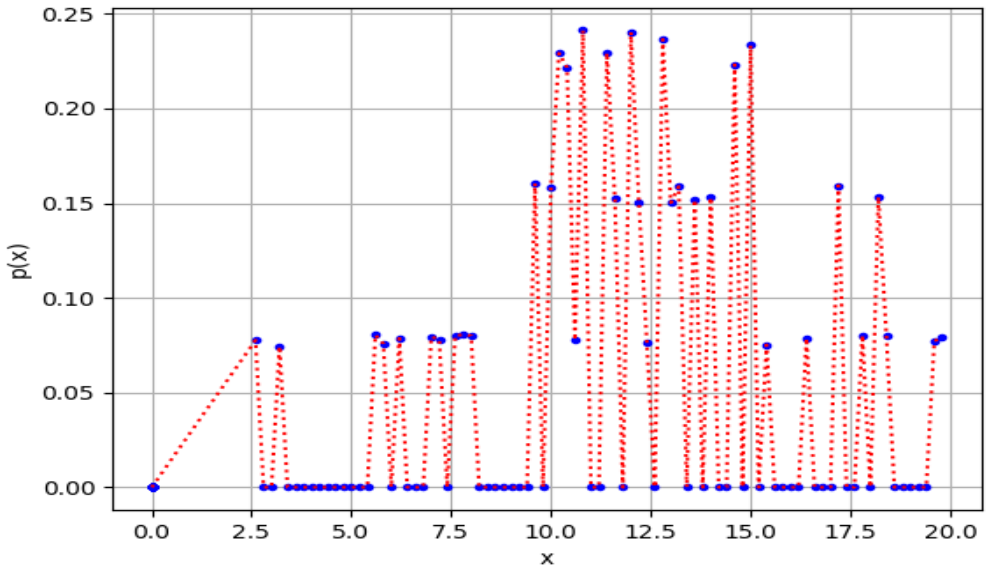
4. Like p3 but with "marks.txt" and dx=0.2 and instead of counting the number of members $w(\frac{\xi_i - k_i \Delta x}{\Delta x})$ is added to the value of the n; Then PDF has been computed by using $p(x) = \frac{n(x)}{N\Delta x}$ and saved in "marksPDF.txt" and the plot is save in "marks GWF PDF.png" .

5. Like p3 PDF has been computed using Top-Hat window function and then smoothed by using the relation

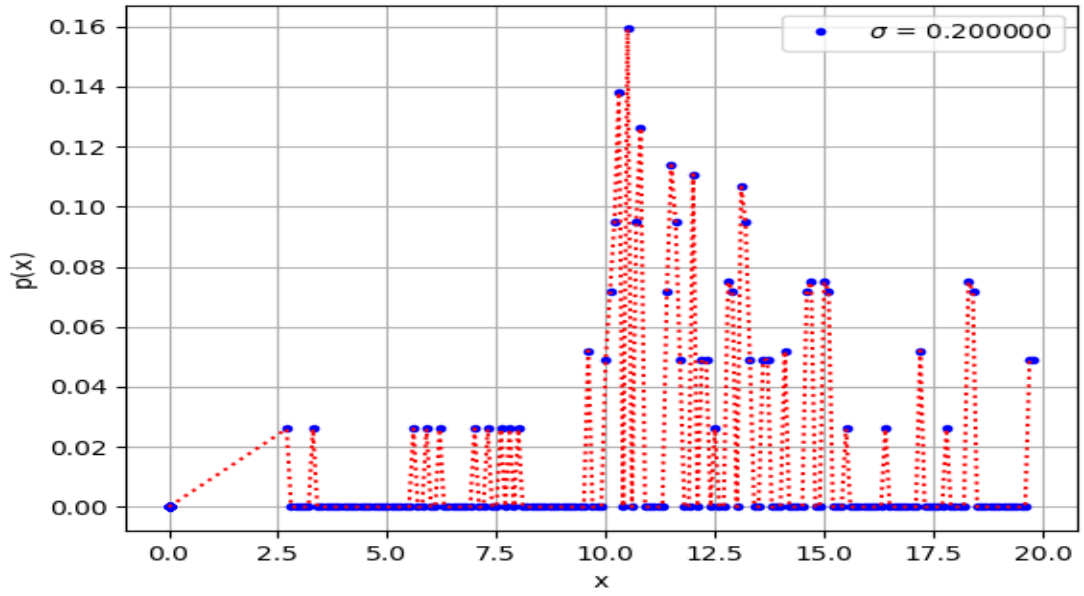
$$\tilde{p}(x) = \sum_{x'=-\infty}^{\infty} w(x - x')p(x')$$

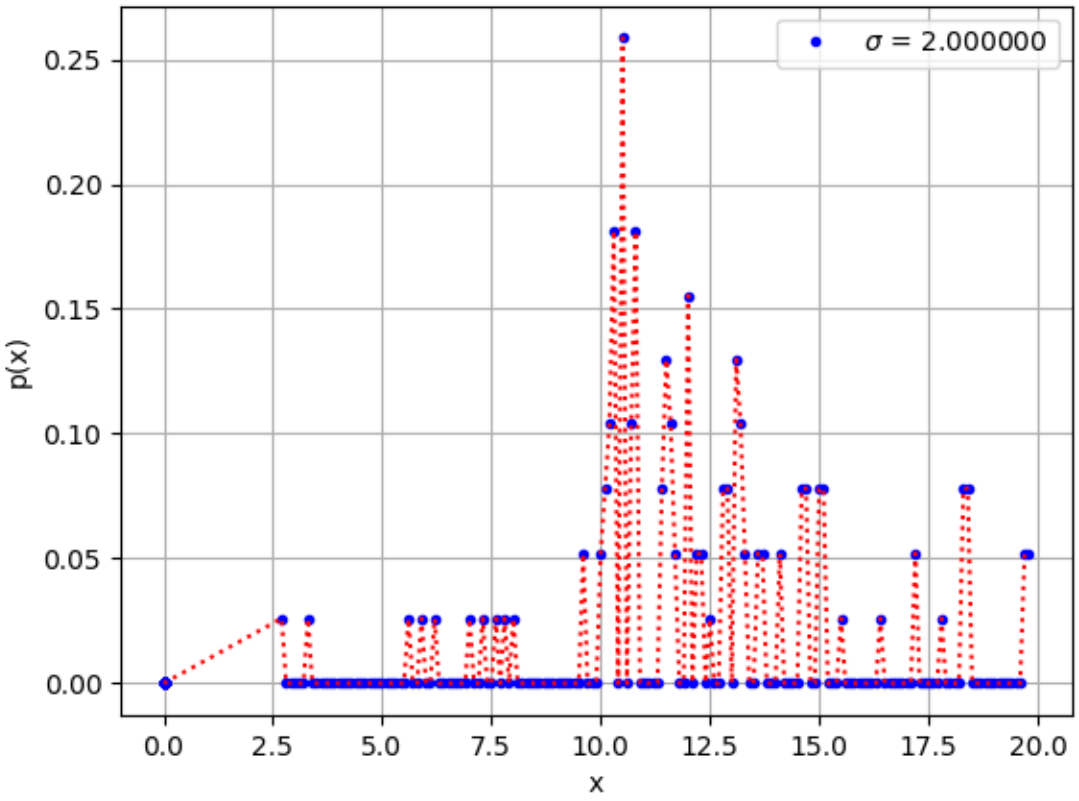
and the algorithm explained in the class;

p4 result plot



p5 result plots





As shown the p5 results are smoother (less variant) than p4 result; and between p5 results we find out that as σ gets larger $p(x)$ gets smaller.