SUPAC++ Labs 3 and 4

Gary Robertson

26^{th} November 2025

Important Information

This assignment makes up the second half of the labs for this course. In the first half we build up a complex piece of software starting from a simple implementation using functions and user input with the results being simple text output. In this assignment we aim to do something similar, but the steps will not be as explicit. Here, we will focus more on classes and inheritance as well as adapting pre-written code. The content here should be compatible with what we have discussed during the third and fourth lectures.

Try to follow good coding conventions when writing code:

- Comments - Add comments as you go. Helps keep track of what the code is doing, and makes it clearer for anyone looking at it down the line (me, but also future you). Consider also adding your name and the date at the top of the script as a comment.
- Repeated testing - Don't just write the code and then test it at the end. Test as you go to ensure each step is working as expected. Also carefully consider edge-cases which will likely break your code.
- Naming Conventions - Try to be consistent with names you use for variables, and make them clear.
- Indentation - C++ is a whitespace independent language, meaning that you can vary the spaces and indentation in commands without causing a crash (*e.g.* `int x                =                5` will be correctly interpreted) but you should try to keep your code neat to ensure readability.

Submission

Submission of your code for the assignments will be done using git (see instructions at the bottom of the `README.md`). Please try to avoid uploading too many extra files beyond what is needed to complete the exercises. You can also add a `README.md` file explaining how to run your code.

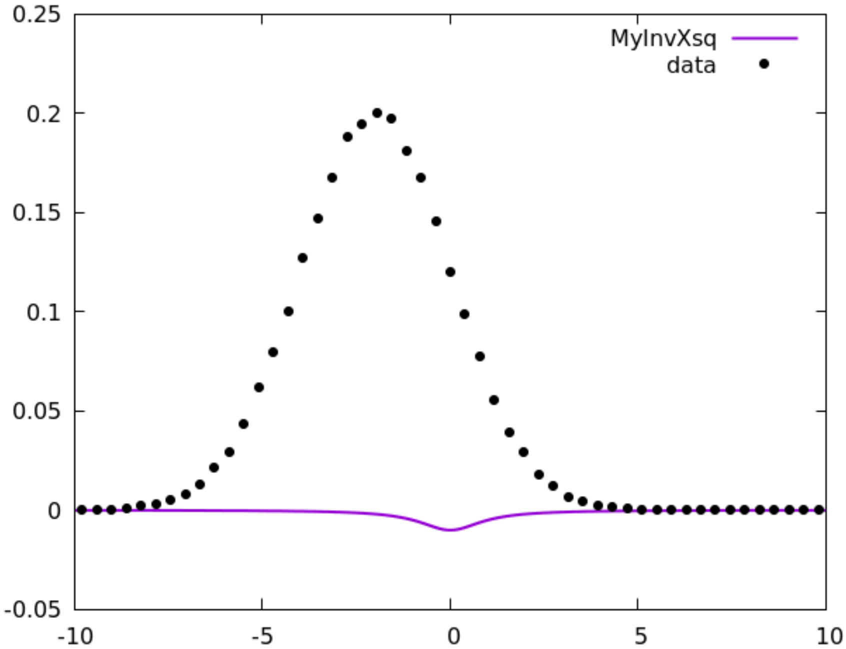
Assignment Exercises

As in the previous assignment, you may find it useful to read through all of the steps to get a feel for the desired functionality of the final code which can be helpful to keep in mind when working through the earlier steps. However, if you find this too overwhelming don't worry - this is why we start simple and slowly add complexity.

In the directory for this assignment you will find a pre-compiled executable called `GenerateRandomData`. If you run this in the terminal it will create a file called `Output/Data/MysteryDataXXXXX.txt` where `XXXXX` is a random combination of numbers. These numbers are 1 dimensional data points following an unknown statistical distribution. The goal of this assignment is to determine what the distribution is that this data has been sampled from, and then use the same function to sample new data points. Rather than having to fit the data directly, you have access to a prewritten base class `FiniteFunctions` which you can use to evaluate and plot finite functions. The default function used is $f(x) = \frac{1}{1+x^2}$.

This assignment is due by the 12th of December

Firstly, write a script to test the default version of the `FiniteFunctions` class. There is no documentation on this class, so you will need to read the code to understand how it works. Without adding anything to the class you should be able to produce a plot of the default function alongside the data points (which can be read in from the `MysteryDataXXXXX.txt` file). N.B. Think about the limits you plot the function in. You can use the provided `Makefile` to compile everything in one go (using the command `make`). Doing this should give you a plot that looks similar to:



It looks like there is a problem with the normalisation of the function! Try to fix it and then remake the plot (Hint: look for any incomplete class methods).

Once the normalisation is fixed you should find that the data still doesn't match the shape of the function very well. We can consider other functional forms to test, adding them as new classes (in a new script) which inherit from the `FiniteFunction` class. Add the following distributions:

1. Normal: $f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$,
2. Cauchy-Lorentz: $f(x) = \frac{1}{\pi\gamma\left[1+\left(\frac{x-x_0}{\gamma}\right)^2\right]}$ where $\gamma > 0$,
3. Crystal Ball: $f(x) = N \cdot \begin{cases} e^{-\frac{(x-\bar{x})^2}{2\sigma^2}} & \text{for } \frac{x-\bar{x}}{\sigma} > -\alpha \\ A \cdot \left(B - \frac{x-\bar{x}}{\sigma}\right)^{-n} & \text{for } \frac{x-\bar{x}}{\sigma} \leq -\alpha \end{cases}$

   where:

   $n > 1$, $\alpha > 0$,

   $A = \left(\frac{n}{|\alpha|}\right)^n \cdot e^{-\frac{|\alpha|^2}{2}}$,

   $B = \frac{n}{|\alpha|} - |\alpha|$,

   $N = \frac{1}{\sigma(C+D)}$,

   $C = \frac{n}{|\alpha|} \cdot \frac{1}{n-1} \cdot e^{-\frac{|\alpha|^2}{2}}$,

   $D = \sqrt{\frac{\pi}{2}}\left(1 + erf\left[\frac{|\alpha|}{\sqrt{2}}\right]\right)$.

You should look carefully at the member declarations in `FiniteFunctions.h` to see which functions can be overridden. The final result should be a single header and C++ file containing the three custom functions and an executable you can use for testing. This should have a main function which calls the constructor for each of the new classes, calculates the integral (using a sensible number of sampling points), prints out information about the function (the range, the chosen parameters, the integral *etc.*) and then produces a plot of the function alongside the mystery data. You can then easily vary the parameters for each function until you believe you have found the distribution that the data was sampled from.

Sampling

Now that you know the distribution the data was sampled from, you can sample it yourself to produce more pseudo-data. This can be accomplished using the metropolis algorithm (note that this method only works on finite functions so you should ensure that the range of your function is wide enough for it to be considered finite) by following the steps below:

1. Generate a random number, x_i, within the range where the function is defined, sampled from a uniform function.
2. Generate a second random number, y, from a normal distribution centred on x_i using an arbitrarily chosen standard deviation.
3. Compute $A = min\left(\frac{f(y)}{f(x_i}, 1\right)$, where f is the function you are sampling from.
4. Generate a random number T where $0 \leq T \leq 1$ and if $T < A$ then accept y.
5. If you accepted y, then set $x_{i+1} = y$, otherwise set $x_{i+1} = x_i$.

Add a method to your classes which samples from each function using the metropolis algorithm. N.B. You should be able to write a single sampling algorithm that works for all functions. As a final check, produce some sampled data and use it to create a plot with the function, the mystery data, and your sampled data. Does the metropolis algorithm work well? Try to adjust the width of the normal distribution used in the sampling to improve the performance.

Deliverables

For the submission you should include (at a minimum):

- `FiniteFunctions.cxx` along with its corresponding header file.
- The script which contains the definitions of the classes which inherit from `FiniteFunctions`, along with its corresponding header file.
- The script which samples new data from the supposed function used to generate the original data (this could be contained within the previous script).
- Optional: A `README` explaining how to compile/run your code.
- Optional: A `Makefile` if you have written one.
- Optional: your attempt at the bonus exercise (see below).

If there are any other scripts you have written that are required to reproduce the results, then you should also include them. You can also include any plots of data that your code produces, but these should be reproducible when running the code from scratch.

Bonus

The following won't be assessed as part of the assignment, but you may find it useful in showing you the versatility of the metropolis algorithm.
Write a script that takes in two arguments at the command line, `radius` and `n_random`, then uses these to generate random numbers. Using these, calculate an estimate of π. Try to estimate with accuracy to 10 decimal places.
Hint: *Consider the area of a square compared to the area of a circle.*