Introduction to Computer Programming in C

Course, June 2012

Short non-assessed exercises

1) Write a programme that defines two integer variables, assigns values to them, adds them up and puts the result in a third integer variable, then prints the result to the screen.

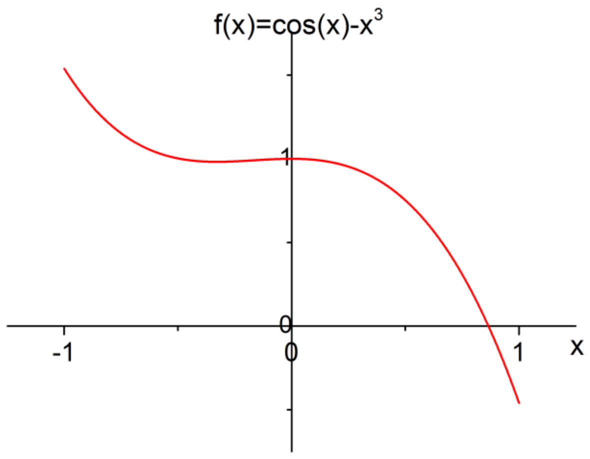
2) Modify the above programme so that it reads the two values from the user using the scanf function.

3) Next, read in two integer numbers, divide the two and then display the whole part of the result and the remainder as well as the complete floating point division with a specified number of digits.

4) Write a loop to display all integers up to 10. Modify it to display only the odd integers.

5) Write a loop to add up all the ODD integers up to 100.

6) Solve the equation cos(x) $-x^3$ =0 using Newton's method.



Here we are solving f(x)=0, with f(x) = cos(x) $-x^3$. The method works by first guessing a solution (say in the range 0 to 1). Call this x_0. Then we proceed iteratively to the solution. The next approximation to the solution is given by

$x_{n+1}=x_n- f(x_n)/[df(x_n)/dx]$

Display the result of each iteration to the screen, like:

```
Itteration  1, x=1.385786648723
Itteration  2,  x=1.018458571221
Itteration  3,  x=0.884288635194
etc….
```

Find the answer to a 12 decimal places. Write code to test when the answer has converged.

Assessed exercise 1

PART A

Write a program to calculate the distribution of numbers produced by the random number function rand().

1) Calculate a random number between 0 and 1. To do this use rand() and divide by RAND_MAX. Remember to use a CAST for RAND_MAX (see lecture notes).

2) Next write some code to 'bin' the data. Create an array where each element represents a bin (use, say, 10 bins to start but make this variable). Increment (increase by one) the array element where the number falls. You need to calculate which bin the random number will fall into. If x is the random number between 0 and 1 and there are 10 bins, then the bin number will be the integer part of x*10 (assuming your array starts at 0).

Repeat this for a large number of times (use a `for(){}` loop).

Calculate the normalised frequency of occurrence in each bin (just the number of bin elements divided by the total number of samples). Then display the results. Check that the sum of all the frequencies from each bin equals 1.0. Are the differences from uniformity reasonable? (Remember for a random distribution the expected error in each count is $N^{1/2}$).

Input the number of samples and the number of bins from the keyboard.

Part B

3) Create a Normal distribution: In part A you will have noticed that the rand function gives a uniform distribution. That is to say that there is an equal probability of the number being any value in the range 0 to the maximum. In some case we need to use a random number generator that has a Gaussian distribution (also called a Normal distribution). Here the probability density (*p*) is given by

$$p = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2} \right)$$

Here μ is the mean value, and σ is the standard deviation. A simple algorithm to do this is

1: Create two random numbers x and y which are between –1 and +1. To do this just double the random number (which is between 0 and 1) and subtract 1.

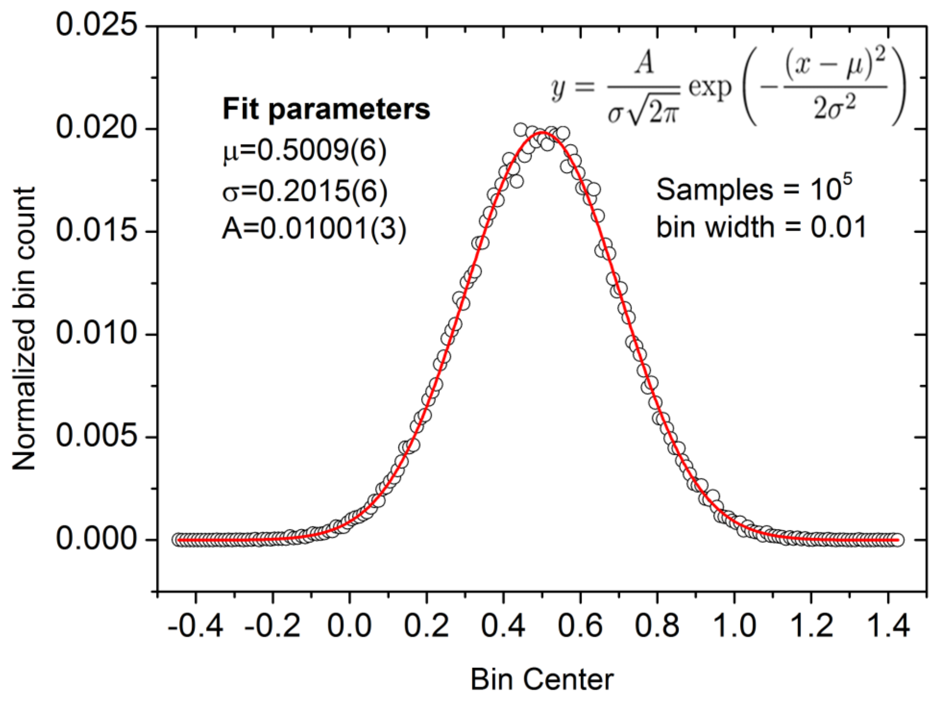
2: calculate: $z = x^2 + y^2$

3: Repeat 1 and 2 until $z < 1.0$ using, for example, a while(){} loop.

4: Now calculate: $w = \sigma x\, (-2\ln(z)/z)^{1/2} + \mu$

w should now be a random number with a Normal distribution with the required standard deviation and mean. For testing purposes, set the mean μ= 0.5 and the standard deviation σ= 0.2

Test that this works by writing a large number of values of *w* into a file, and plot a histogram of the resulting values using standard software (e.g., Origin: use 'statistics\frequency count'): for an example with 10^5 samples see below.



Make this into a function: call it `randGauss` for example and repeat the first part of this exercise for this distribution for large N. Compare the numbers obtained to those expected from the function p above. Note that *p* above is the value of the probability density at a particular *x*, whereas your bin function will give the integral of *p* from *x* to *x* + *dx*, where *dx* is the bin width, if *dx* is small you can approximate the integral to *p*dx*, so the normalised bin count is just *p* times the bin width.

Write the results to a file. Check they are correct by plotting them with plotting software (Origin).

Hand in two programmes: One for part A and a separate one for part B

Assessed exercise 2

Least Squares fitting

On many occasions you will need to fit mathematical functions to data points in order to compare with an underlying theory. The least squares method is common way of doing this.

Mathematical background

The least squares method assumes that the best-fit function of a given type is the one that minimises the sum of the square of the differences (hence the name least squares) for a given set of data. Suppose the data points are (x_i, y_i), $i=1..n$ where x is the independent variable and y the dependent variable. Also suppose that the estimated error on each measured point y_i is σ_i . The fitted function $f(x)$ then has a set of differences from each data point $d_i = y_i - f(x_i)$, $i=1..n$. The least squares method then states that the best fit function minimises

$$S = \sum_{i=1}^{n} \left(\frac{d_i}{\sigma_i} \right)^2 \equiv \sum_{i=1}^{n} \left(\frac{y_i - f(x_i)}{\sigma_i} \right)^2 .$$

To find the minimum, assume that $f(x_i)$ depends on a series of unknown coefficients, C_j, $j=1..m$. Differentiating S with respect to these coefficients results in a series of m simultaneous equations, which can be solved to find the coefficients of the best fit function.

For a straight line (linear) function $f(x) = a + bx$,

$$S = \sum_{i=1}^{n} \left(\frac{y_i - (a + bx_i)}{\sigma_i} \right)^2$$

Here a and b are the unknown coefficients. Taking partial derivatives gives the simultaneous equations

$$\begin{cases} \dfrac{\partial S}{\partial a} \equiv 2 \displaystyle\sum_{i=1}^{n} \left(\frac{y_i - (a + bx_i)}{\sigma_i^2} \right) = 0 \\ \dfrac{\partial S}{\partial b} \equiv 2 \displaystyle\sum_{i=1}^{n} x_i \left(\frac{y_i - (a + bx_i)}{\sigma_i^2} \right) = 0 \end{cases}$$

Expanding these equations gives

$$\begin{cases} \displaystyle\sum \frac{y_i}{\sigma_i^2} = a \sum \frac{1}{\sigma_i^2} + b \sum \frac{x_i}{\sigma_i^2} \\ \displaystyle\sum \frac{x_i y_i}{\sigma_i^2} = a \sum \frac{x_i}{\sigma_i^2} + b \sum \frac{x_i^2}{\sigma_i^2} \end{cases}$$

from which the coefficients can be obtained as follows:

$$a = \frac{\sum \frac{y_i}{\sigma_i^2} \sum \frac{x_i^2}{\sigma_i^2} - \sum \frac{x_i}{\sigma_i^2} \sum \frac{x_i y_i}{\sigma_i^2}}{\sum \frac{1}{\sigma_i^2} \sum \frac{x_i^2}{\sigma_i^2} - \left(\sum \frac{x_i}{\sigma_i^2} \right)^2}$$

$$b = \frac{\sum \frac{1}{\sigma_i^2} \sum \frac{x_i y_i}{\sigma_i^2} - \sum \frac{x_i}{\sigma_i^2} \sum \frac{y_i}{\sigma_i^2}}{\sum \frac{1}{\sigma_i^2} \sum \frac{x_i^2}{\sigma_i^2} - \left(\sum \frac{x_i}{\sigma_i^2} \right)^2}$$

The uncertainty on the coefficients is given by:

$$\sigma_a^2 \equiv \sum \sigma_i^2 \left(\frac{\partial a}{\partial y_i} \right)^2 = \frac{\sum \frac{x_i^2}{\sigma_i^2}}{\sum \frac{1}{\sigma_i^2} \sum \frac{x_i^2}{\sigma_i^2} - \left(\sum \frac{x_i}{\sigma_i^2} \right)^2}$$

$$\sigma_b^2 \equiv \sum \sigma_i^2 \left(\frac{\partial b}{\partial y_i} \right)^2 = \frac{\sum \frac{1}{\sigma_i^2}}{\sum \frac{1}{\sigma_i^2} \sum \frac{x_i^2}{\sigma_i^2} - \left(\sum \frac{x_i}{\sigma_i^2} \right)^2}$$

Notice that the left hand side of this equation is the **square** of the uncertainty σ. Also in this form of the least squares fitting procedure only the errors in the y values are used. There are alternative methods which can take account of the errors in x as well but these are not widely used.

The project

Your aim is to write a programme that can read in a text based data file which contains an unknown number of data points and fit the data to a linear (straight line) form. The format of the data file is x, y, σ_x, σ_y where σ_x, σ_y are the errors on the x and y values. The data should be read in from the file and then the various sums calculated so that the a and b coefficients and their associated error can be computed (note that the error in x is not used in this analysis method). Read the data into an array before performing the sums. You could just calculate the sums without reading it first into an array, but please do it this way to demonstrate that you know how to use multidimensional arrays. (for the more advanced use dynamically created arrays). Finally, make the programme modular so that there is a separate function for reading in the data (takes a filename as an argument and returns an array with the data and the number of datapoints), and a function for calculating the least squares coefficients (takes the arrary of data and the number of points as arguments and returns the coefficients and errors). So the structure of the main part of the code might look something like:

1)Ask user to input filename from keyboard

2) Call function to read in data from file (check the file was actually found and read in correctly)

3) Call function to calculate coefficients

4) Display coefficients

For testing proposes use your program to fit the following data file which can be downloaded from the course web site.

```
LinearTestdata.TXT
LargeDataSet.TXT
```

The expected results for each of the data sets can be found in the file: `TestResults.txt`.