First Year Computing (C++ Course)
Dr John Joss Chinn

Week 4.

Decision Making, Bubble Sort, Pivoting, Newton-Raphson Method for Finding Roots

Decision Making

Quite often, in a computer program, you will need to carry out certain tasks based on the value of a number. For instance you may want to group people according to given salary bands. In the simple program, `if1.cpp`, below, if a number, input by the user, is less than five then the program simply prints to the screen `That number is less than 5.` As with `for` and `while` loops, the criteria for the decision is encased between two curly brackets.

```
// if1.cpp
#include<iostream.h>
#include<conio.h>

void main()
{
      int number;

      cout<<"Enter a number between 1 and 10\n\n";
      cin>>number;

      if(number<5)
      {
            cout<<"That number is less than 5\n";
      }
      getch();
}
```

The following program, `if2.cpp,` is a little more complicated. This program again asks the user to type in a number between 1 and 10. This time the program can decide if the number is strictly less than 4, between 4 and 8 inclusive or is strictly greater than 8. (Just a small point, to save paper, this program now runs over to the next page!)

```
// if2.cpp
#include<iostream>
#include<conio.h>

void main()
{
      int number;

      cout<<"Enter a number between 1 and 10\n\n";
      cin>>number;

      if(number<4)
```

```
	{
		cout<<"That number is less than 4\n";
	}
	else if(number>=4 && number<=8)
	{
		cout<<"That number is between 4 and 8\n";
	}
	else
	{
		cout<<"That number is greater than 8\n";
	}
	 getch();
}
```

We can use this decision making methodology to perform use ful tasks. The following program, `max1.cpp`, contains an array of peoples heights in inches. The decision making routine finds the tallest person.

```
	// max.cpp
	#include<iostream>
	#include<conio.h>

	void main()
	{
		float height[]={68,62,72,70,66,68,78,71,69,62};
		float max_h=0;  //initialise to zero
		int i;

		//prints a running comparison
		cout<<"Height\tMaximum\n\n";
		for(i=0;i<10;i++)
		{
			cout<<height[i]<<"\t"<<max_h<<endl;

			if(height[i]>max_h)
			{
				max_h=height[i];
			}
		}
		cout<<"\n\nThe maximum height is "<<max_h<<endl;
		getch();
	}
```

The output looks like that shown in the box here. The first output line (`cout . .`) simply shows what is happening within the program.

```
Height  Maximum

68      0
62      68
72      68
70      72
66      72
68      72
78      72
71      78
69      78
62      78


The maximum height is 78
```

Bubble Sort

The program show below, `bubble1.cpp`, will sort an array of numbers into ascending order, like the lottery. It contains nested loops and a swap routine.

```
//bubble1.cpp
#include<iostream.h>
#include<conio.h>
#define SIZE 6    //One way to define a constant

void main()
{
      int i,j,temp;
      int lotto[]={10, 48, 1, 37, 6, 24};

      cout<<"Here is the array unsorted\n\n";

      //This loop just prints the array to the screen
      for(i=0;i<SIZE;i++)
      {
        cout<<lotto[i]<<"\t";
      }

      /*'Bubble Sort' routine. These nested loops actually do
      the work*/

      for(i=0;i<SIZE;i++)
      {
            for(j=i+1;j<SIZE;j++)
            {
                  if(lotto[i]>lotto[j])
                  {
                    temp=lotto[j];     //store lotto[j]
                    lotto[j]=lotto[i]; //swap
                    lotto[i]=temp;
            /*makes lotto[i] equal to the former lotto[j]*/
                  }
            }
      }

     cout<<"\n\n\nHere is the array sorted out\n\n";

      //this loop prints the sorted array to the screen
      for(i=0;i<SIZE;i++)
      {
        cout<<lotto[i]<<"\t";
      }

      cout<<endl;
      getch();
}
```

The way that this program works, actually the way the nested loops work, is shown in the table over the next page. The comparison is always;

is lotto[i] > lotto[j] ? This has been adapted from “C for Dummies” by Dan Gookin.

I	j	Action	comment	*0*	*1*	*2*	*3*	*4*	*5*
0	1	Is 10 > 48? No		**10**	**48**	1	37	6	24
0	2	Is 10>1? Yes	Swap	**10**	48	**1**	37	6	24
0	3	Is 1>37? No		**1**	48	10	**37**	6	24
0	4	Is 1>6 No		**1**	48	10	37	**6**	24
0	5	Is 1>24 No		**1**	48	10	37	6	**24**
1	2	Is 48>10? Yes	swap	1	**48**	**10**	37	6	24
1	3	Is 10>37? No		1	**10**	48	**37**	6	24
1	4	Is 10>6 Yes	swap	1	**10**	48	37	**6**	24
1	5	Is 6>24? No		1	**6**	48	37	10	**24**
2	3	Is 48>37? Yes	swap	1	6	**48**	**37**	10	24
2	4	Is 37>10? Yes		1	6	**37**	48	**10**	24
2	5	Is 10>24? No		1	6	**10**	48	37	**24**
3	4	Is 48>37? Yes	swap	1	6	10	**48**	**37**	24
3	5	Is 37>24? Yes	swap	1	6	10	**37**	48	**24**
4	5	Is 48>37? Yes	swap	1	6	10	24	**48**	**37**
			Final Result	1	6	10	24	37	48

Pivoting

Last week we looked at a simple program which simply printed out the augmented matrix for a set of simultaneous equations. To solve this set of simultaneous equations by Gaussan Elimination with partial pivoting, one of the things which need to be done is to successively find the largest element in each column and rearrange the rows so that the row with the largest element, in a given column, falls on the diagonal. These elements are then called “pivots”. The program below, `pivot1.cpp`, is a continuation of last weeks work. It simply prints out the original augmented matrix, then it finds the pivot (largest element) in the first column only (this is in the third row in this example, row 2) it then swaps rows 0 and 2 around and then prints out the altered augmented matrix.

```
//pivot1.cpp
#include<iostream>
#include<conio.h>
#include<math.h>

void main()
{
   int i,j,marker;
   float pivot;
   float temp;

   //augmented matrix
   float A[4][5]={ 1, -3, 6,  2,  39,
                  -2,  4,-7, -3, -48,
                   5,  2, 1,  4,  37,
                   3,  2, 6,  7,  67};
```

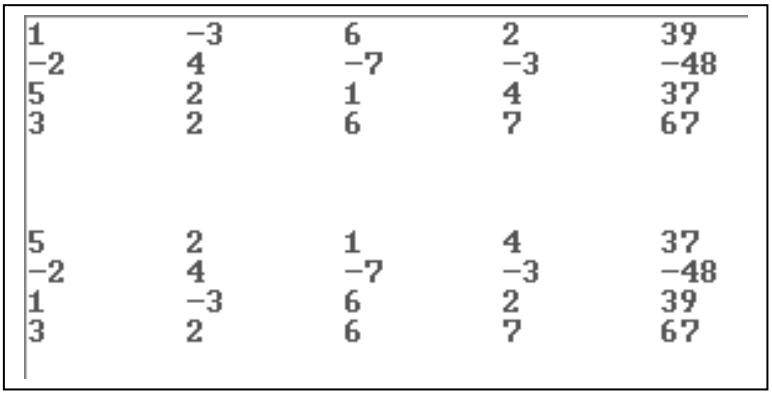
```
   //just print out the augmented matrix
   for(i=0;i<4;i++)
    {
      for(j=0;j<5;j++)
       {
         cout<<A[i][j]<<"\t";
       }
      cout<<endl;
     }
    cout<<"\n\n\n";

 pivot=0;
                  //find the largest A[i][0]
   for(i=0;i<4;i++)
    {
      if(fabs(A[i][0])>fabs(pivot))
       {
         pivot=A[i][0];
         marker=i;
        }
    }
                   //swap row 'marker' with row 1
   for(j=0;j<5;j++)
    {
      temp=A[0][j];
      A[0][j]=A[marker][j];
      A[marker][j]=temp;
     }

    //just print out the augmented matrix
   for(i=0;i<4;i++)
    {
      for(j=0;j<5;j++)
       {
         cout<<A[i][j]<<"\t";
       }
        cout<<endl;
     }

   cout<<endl;
   getch();
}
```

The output looks like this. The original matrix and the modified matrix.



Newton-Raphson

It is probably a good juncture to begin to use our C++ programming skills to understand how Engineering methods may be implemented. The Newton-Raphson method is used to find the solution (the 'roots') of equations of the form:-

(1) $x^3 - 5x^2 - 38x + 168 = 0$
(2) $\tan(x) = x$ $(\tan(x) - x = 0)$
(3) $e^x = 3x$ $(e^x - 3x = 0)$.

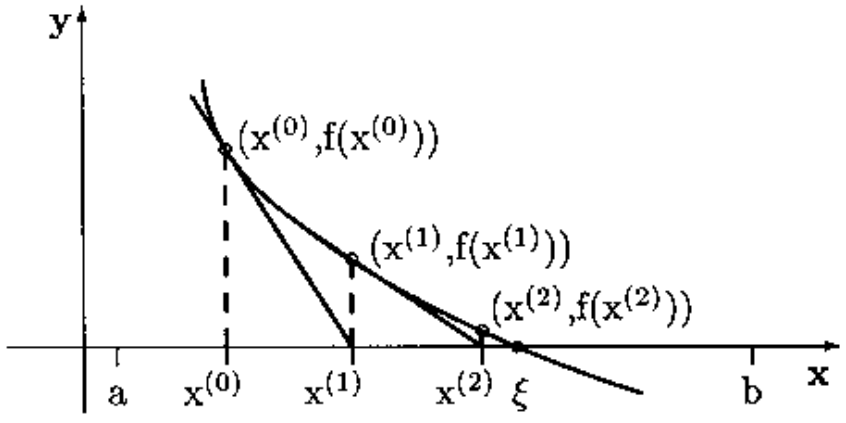
These solutions cannot be solved analytically. One could solve them by trial and error. One might try producing graphs of the functions and observing where the function crosses the x-axis. Finally, one may solve them, to within a specified accuracy, using a numerical method, such as the Newton-Raphson method.

The Newton-Raphson method is an ITERATIVE method. One guesses an initial value for the solution, for the root, and the method then iterates to find an increasingly nearer approximation to the solution. The method works by finding the tangent to the curve of the given function at the guessed value, i.e. x_0, $f(x_0)$, and then uses this to find a better approximation, x_1,$f(x_1)$, and so on.

The method essentially determines the equation to the tangent to the curve f(x), at the point x_0,$f(x_0)$, where x_0 is the initial guessed value for the root. This equation, for the tangent, is of the form $y = mx + c$, i.e. a straight line. This equation is then used to find the next best value, x_1, where the straight line cuts the x-axis. This straight line equation obviously passes through the curve so we have

$$f(x_0) = mx_0 + c,$$

where m is the gradient at the point x_0,$f(x_0)$.



From "Numerical Algorithms with C". Engeln-Müllges and Uhlig.
(The root is ξ and the notation is slightly different than mine.)

By definintion, the gradient to the tangent to the curve is $m = f'(x_0)$. This gives

$$f(x_0) = f'(x_0).x_0 + c.$$

Rearranging this we have

$$c = [f(x_0) - f'(x_0).x_0].$$

Therefore the solution for the equation of the line y = mx + c is

$$y = f'(x_0).x + [f(x_0) - f'(x_0).x_0].$$

Now we want to find x at the point where the curve crosses the x-axis, i.e. where y = 0. This will be at $x = x_1$, the next best value for the root, so that

$$0 = f'(x_0).x_1 + [f(x_0) - f'(x_0).x_0].$$

Rearranging this gives

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

This defines the Newton-Raphson formula for finding the roots of the equation f(x) = 0. We will want to find x_1 using x_0 as an initial guess, we then find x_2 using x_1 and x_3 using x_2, and so on. In general we find x_i using x_{i-1}. The general formula for the Newton-Raphson method ought then to be

$$x_i = x_{i-1} - \frac{f(x_{i-1})}{f'(x_{i-1})}$$

Let us look at an example that we may use to demonstrate the method. Consider the function

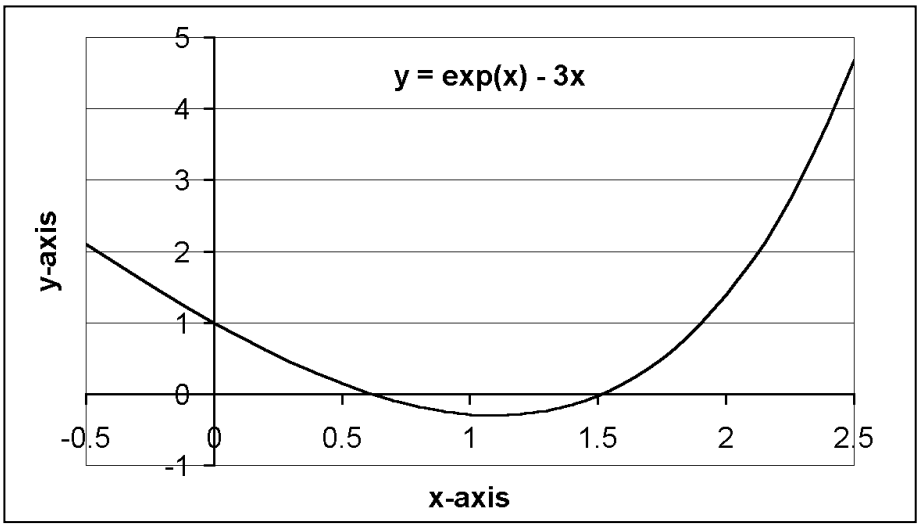
$$f(x) = e^x - 3x = 0.$$

We have

$$f'(x) = e^x - 3.$$

so that

$$\text{from } x_i = x_{i-1} - \frac{f(x_{i-1})}{f'(x_{i-1})} \text{ we get } x_i = x_{i-1} - \frac{e^{x_{i-1}} - 3x_{i-1}}{e^{x_{i-1}} - 3}$$



To put this into a computer program it is a good idea to break it up and have separate variables for the function and its derivative;

```
func=exp(x_old)-3*x_old;
deriv=exp(x_old)-3;
```

The C++ program, `newton1.cpp`, below, shows the method in action.

```
// newton1.cpp
#include<iostream.h>
#include<conio.h>
#include<math.h>

void main()
{
    double error=1e-6;
    double x_old=1e6;   //must initialise it
    double x_new=1;
    double func,deriv;

    while(fabs(x_new-x_old)>error)
     {
       x_old=x_new;
       func=exp(x_old)-3*x_old;
       deriv=exp(x_old)-3;
       x_new=x_old-func/deriv;
       cout<<"x_old\t\t"<<x_old<<"\tx_new\t"<<x_new<<endl;
     }
    cout<<x_new<<endl;
    getch();
}
```

Let us take a look at how this program works. The line `error=1e-6;` defines a sort of tolerance to determine when the root is accurate enough. In other words, when the difference between an old value of x and a new value of x is less than 1e-6 then the program will stop. The `while` loop tests to see if the difference between successive values of x is small enough. It is necessary to initialise `x_old` to some ridiculous value (e.g. 1e6) so that it won't trip out of the loop the first time through. Within the loop itself we set `x_old` to a sensible value, i.e. 1, the same as `x_new` . This is the first guessed value, x_0. So `x_new` is then x_1. The next time through the loop `x_old` takes the value of `x_new` so that the loop therefore calculates x_2 from x_1. The output looks like this

```
x_old   1       x_new   0
x_old   0       x_new   0.5
x_old   0.5     x_new   0.61006
x_old   0.61006 x_new   0.618997
x_old   0.618997        x_new   0.619061
x_old   0.619061        x_new   0.619061
0.619061
Press any key to continue_
```

So the solution of the equation $e^x - 3x = 0$ is x = 0.619061, to six dp.

Tutorial

(1) Type in `if1.cpp` and run it. Try changing the value at which the decision is made from 5 to something else.

(2) Modify `if1.cpp` to `if2.cpp`. Play around with this and change the values on the `if` and `if else` statements.

(3) Type in `max.cpp` and run it. Alter the program to find the minimum persons height.

(4) Type in and run `bubble1.cpp`. Try to understand how it works. Try some numbers of your own and extend it to deal with say ten numbers.

(5) Type in and run pivot1.cpp. Try to understand what it is doing. This is important. Your assignment will be about Gaussian Elimination with Partial Pivoting. Try the following augmented matrix

$$\begin{pmatrix} 5 & 7 & 6 & 5 & 23 \\ 7 & 10 & 8 & 7 & 32 \\ 6 & 8 & 10 & 9 & 33 \\ 5 & 7 & 9 & 10 & 31 \end{pmatrix}$$

(6) Type in and run newton1.cpp. Try the method with different functions. For instance

(a) $x^3 - 5x^2 - 38x + 168 = 0$

(b) $\tan(x) - x = 0$.