2G03 Homework 6, 2020

Due: Monday Nov 9th (hard deadline Tuesday Nov 10th)

You should submit C++ code and write Makefiles for the exercises marked with `PROGRAM`. You should create a directory called *HW6* in your home directory for these programs. The grader will look there to locate and run your programs.

Arrays – Finite Differences – Interpolation

Exercise 1 `PROGRAM` *Numerical Calculus*

In this exercise we will make use of the PGPLOT library. Please refer to the lecture notes about PGPLOT for more information.

There is a C++ demonstration program that plots a simple function in `/home/2G03/ploty`. This program plots the function $y = cos(x)$ over the range $x = [0, 2\pi]$. You will not need any PGPLOT routines other than the ones used in `ploty.cpp`. The Makefile in that directory also includes all the information required to link a program to make use of the pgplot library. You can use `ploty.cpp` as a basis to do this exercise.

This exercise will make use of finite difference techniques to approximate derivatives and integrals. These techniques are based on Taylor series which were covered in first year mathematics. The idea is that derivatives of smooth functions can be approximated quite accurately using the small finite differences between the values of the function at two or more nearby points.

1.1 The derivative of a function is given as,

$$\frac{d\,f}{d\,x} = \lim_{\Delta x \to 0} \frac{f(x+\Delta x) - f(x)}{\Delta x}.$$

If f is a reasonably smooth function then Δx does not have to be very small to approach the correct limit. A numerical approximation for finite Δx is,

$$\frac{d\,f}{d\,x}\,|_{\text{approx.}} = \frac{f(x+\Delta x) - f(x)}{\Delta x} = \frac{d\,f}{d\,x} + \Delta x \frac{1}{2}\frac{d^2\,f}{d\,x^2} + O(\Delta x^2)$$

The leading error term depends on what the second derivative of the function $\frac{d^2\,f}{d\,x^2}$ is like and scales as $O(\Delta x)$. Code to use this simple approximation at one point is shown below:

```
// derivative of y at x=x[1]
   dydx[1] = (y[2]-y[1])/deltax;
```

Starting with `ploty.cpp`, change the code to plot a different function $f(x) = 1 - x + 3\,x^2 - x^3$ in the interval from $x = 0$ to $x = 1$. Adjust the arguments to `cpgenv` appropriately to ensure the whole curve fits vertically on the plot.

Add code to generate the derivative of $f(x)$ using finite differences. Take special care near the end points so that you aren't using values outside the array bounds.

Plot the derivative of $f(x)$ on the same axes by calling the `cpgline` subroutine. Adjust the arguments to `cpgenv` again to ensure both curves fit vertically on the plot. Use $n = 20$ intervals. Change the color using `cpgsci` so you can tell the difference between the curves.

1.2 The error, e_i, in this approximation is defined as the difference between your approximation and the exact derivative of $f(x)$ at x_i. Calculate the root mean squared ($rmserror = \sqrt{\frac{1}{n}\sum_{i=1}^{n} e_i^2}$) and maximum error over the n intervals. Set up your program so that it reads the number of intervals from the terminal when it is run. Run your program for $n = 20$, $n = 40$ and $n = 80$ and have it print the results for the errors each time using the line:

```
std::cout << "Derivative Error " << DerivativeRmsError << "\n";
```

1.3 How do you expect the *rms* error to change as n gets larger? Do the results of your program match this?

Note that this is a very poor (low order) approximation. For extra credit you can suggest a more accurate expression. The accuracy of any expression can be calculated using Taylor series expansions of the function about a point.

1.4 In general it isn't hard to get an analytical derivative unless the function has unknown form. However, integrals are harder to get analytically and analytical expressions do not exist for integrals of many important functions. Numerical integration is a primary application of scientific computing that is studied in detail in later courses.

An integral of a smooth function may be expressed as the limit of the Riemann sum:

$$\int_0^L f(x)dx = \lim_{\Delta x \to 0} \sum_{n=1}^{N} f(x_n)\Delta x$$

where $N = L/\Delta x$ and $x_n = \Delta x(n - \frac{1}{2})$. This is also easy to approximate numerically. The direct translation of this expression into a numerical algorithm is called the midpoint rule:

$$\int_0^L f(x)dx \mid_{\text{approx.}} = \sum_{n=1}^{N} f(x_n)\Delta x = \int_0^L f(x)dx + \Delta x^2 L \frac{1}{8} < \frac{d^2 f}{d\, x^2} > + O(\Delta x^3)$$

where $x_n = \Delta x(n - \frac{1}{2})$ is the x value at the middle of each interval and $\Delta x = L/n$ as before. The leading error term is much better for the midpoint rule, $O(\Delta x^2)$ and the error is again dependent on the second derivative of f.

Extend your program to estimate the integral of the function $f(x)$ from 0 to x (assuming the integral is zero at $x = 0$) at each point for $n = 20$. Pay special attention to the endpoints and using the appropriate x values to evaluate the function for each interval. When you are testing, have the program list the values for the integral.

```
std::cout << "Total Integral " << Integral << "\n";
```

Plot the integral function on the same axes by calling the `cpgline` subroutine. Use an identical number of intervals $n = 20$ and so forth. Change the color using `cpgci` so you can tell the difference between the curves.

1.5 Calculate the root mean squared ($rmserror = \sqrt{\frac{1}{n}\sum_{i=1}^{n} e_i^2}$) and maximum error over the n intervals. In this case e_i is the difference between your approximation and the exact integral of $f(x)$. Run your program for $n = 20$, $n = 40$ and $n = 80$ and list the results. Use the following line to print the errors.

```
std::cout << "Integral Error " << IntegralRmsError << "\n";
```

1.6 How do you expect the *rms* error to change as n gets larger? Do the results of your program match this?

Note: It is sufficient to submit only the final version of the program with all the modifications. You do not need to submit plots created with pgplot or any graphs of the curves. However, the T.A. should be able to run your program and see the graphs.