Assignment Three

Deadline Friday 29th November 23:00

All work **must** be done in the sub-directory '`assign3`' off your home directory. For both parts of this assignment write well formatted code with neat and consistent indentation. Your code should implement each of the methods as efficiently as possible. Use the '`INTENT`' attribute for dummy variables.

1 Quadrature

We will be looking at composite methods of integration that will numerically calculate integrals of the form,

$$I = \int_a^b f(x)dx. \tag{1}$$

In this handout we look at three composite methods, Midpoint, Trapezoidal and Simpson's. In general, the different composite methods for Quadrature are derived such that they need to be applied over a set number of sub-regions in the region $[a, b]$ for which the curve is to be integrated. Each of these sub-regions will then be integrated using the appropriate method. In order to do the integration each method requires that the sub-regions need to be further divided into intervals. The Midpoint and Trapezoidal methods require one interval per sub-region and Simpson's method requires two intervals per sub-region. Composite methods therefore involve dividing the region $[a, b]$ into n equally spaced intervals, delimited by $\{x_0, x_1, x_2, x_3, ..., x_n\}$, each of width $h = (b - a)/n$. Each interval or in the case of Simpson's two intervals will provide a sub-region under the curve $f(x)$ which is to have its area approximated. The individual areas of the sub-intervals are then summed up to give an approximation to the total integral (1).

1.1 The Composite Midpoint Rule

In the Composite Midpoint Rule the area within each sub-region is calculated by multiplying the function value at the midpoint of the interval by the width of the interval h. These n area approximations are summed up to give a total approximation for the total integral (1).

This gives us the general formula for the Composite Midpoint Rule.

$$\int_a^b f(x)dx \approx h \sum_{i=0}^{n-1} f\left(\frac{x_i + x_{i+1}}{2}\right) \tag{2}$$

1.2 The Composite Trapezoidal Rule

The midpoint rule can be improved by making use of the function values at $f(x_i)$ and $f(x_{i+1})$ to calculate the sub-region area instead of simply the function value of the mid-point of the interval.

This gives us the general formula for the Composite Trapezoidal Rule.

$$\begin{aligned}\int_a^b f(x)dx &\approx \frac{h}{2} \sum_{i=0}^{n-1} [f(x_i) + f(x_{i+1})] \\ &\approx \frac{h}{2}\left[f(x_0) + 2\sum_{i=1}^{n-1} f(x_i) + f(x_n)\right]\end{aligned} \tag{3}$$

1.3 The Composite Simpson's Rule

Simpson's rule is different from the Midpoint and Trapezoidal rules in that it uses two intervals to make an application of the rule to approximate the area of a sub-region. It is for this reason that n, the number of intervals, is always chosen to be even so there is always a whole integer number of sub-regions.

The general formula for the Composite Simpson's Rule,

$$\int_a^b f(x)dx \approx \frac{h}{3}\left[f(x_0) + 2\sum_{i=1}^{(n/2)-1} f(x_{2i}) + 4\sum_{i=1}^{n/2} f(x_{2i-1}) + f(x_n)\right]. \tag{4}$$

Assignment Three (Part I)

1. Write a Fortran module called '`quad_mod`' in the file '`quad_mod.f90`' that could serve as a library of various methods for numerical integration. Your methods should solve numerically integrals of the form,

$$\int_a^b f(x)\,dx. \tag{5}$$

   Assume that the function $f(x)$ is continuous on the region $[a,b]$. The module should contain the following four functions.

   - A '`FUNCTION f(x)`' that returns the value of the function for a given value of the independent variable x.
   - A '`FUNCTION midpoint(a,b,n)`' that uses the Midpoint method and returns the numerical approximation to (5) using n intervals over $[a,b]$.
   - A '`FUNCTION trapezoidal(a,b,n)`' that uses the Trapezoidal method and returns the numerical approximation to (5) using n intervals over $[a,b]$.
   - A '`FUNCTION simpson(a,b,n)`' that uses the Simpson method and returns the numerical approximation to (5) using n intervals over $[a,b]$.

2. Write a main program unit called '`quad`' in the file '`quad.f90`' that '`USES`' your module '`quad_mod`'. Your main program should, in a single run, calculate 'I' for all three methods implemented in your module. Test your code by calculating the following integral

$$I = \int_2^{10} x^3 - 4x\,dx$$

   Solve for $n = 2, 4, 10, 100, 200$. Use a one dimensional array to hold the different values of n and include a '`DO`' loop to step through them. Your code should print out, in a single run, a table formatted as shown below (except with different values). Note the number of decimal places used. Redirect the output from your program into a file called '`quest1.txt`'.

```
           Midpoint   Trapezoidal        Simpson
           --------   -----------        -------


   2     -20.588722    -18.068302     -19.865778
   4     -19.948669    -19.328512     -19.748583
  10     -19.774458    -19.675669     -19.741693
 100     -19.741840    -19.740860     -19.741522
 200     -19.741598    -19.741358     -19.741518
```

2 Backward Substitution

Consider a linear system in matrix form such as $A\mathbf{x} = \mathbf{v}$, which represents the four equations and four unknowns where;

$$A = \begin{pmatrix} 1 & -1 & 2 & -1 \\ 0 & 2 & -1 & 1 \\ 0 & 0 & -1 & -1 \\ 0 & 0 & 0 & 2 \end{pmatrix}, \quad \mathbf{v} = \begin{pmatrix} -8 \\ 6 \\ -4 \\ 4 \end{pmatrix} \quad \text{and} \quad \mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}$$

The x_i where $i \in \{1, 2, 3, 4\}$ are the elements of the unknown vector $\mathbf{x}$. Given a linear system in its upper triangular form we can use backward substitution to calculate the vector $\mathbf{x}$. So, for the above system,

$$\begin{aligned} x_4 &= \frac{4}{2} = 2 \\ x_3 &= \frac{-4 - (-1)x_4}{-1} = 2 \\ x_2 &= \frac{6 - x_4 - (-1)x_3}{2} = 3 \\ x_1 &= \frac{-8 - (-1)x_4 - 2x_3 - (-1)x_2}{1} = -7 \ . \end{aligned}$$

This process of backward substitution gives us our solution vector $\mathbf{x}$,

$$\mathbf{x} = \begin{pmatrix} -7 \\ 3 \\ 2 \\ 2 \end{pmatrix} .$$

Assignment Three (Part II)

1. Write a Fortran module called '`linear_mod`' in the file '`linear_mod.f90`'. The module file should contain the following procedures.
   - '`FUNCTION getvec(m)`' : Inputs an arbitrary vector (1D array) of size '`m`'
   - '`FUNCTION getmat(m,n)`' : Copy this from the class project.
   - '`SUBROUTINE outmat(mat)`' : Copy this from the class project.
   - '`FUNCTION back_sub(mat,vv)`' This function accepts as its arguments an *upper triangular* general square matrix '`mat`' and '`vv`' the right hand side vector $\mathbf{v}$ of the linear system. The function should use backward substitution to calculate and return the solution vector.
2. Write a main program called '`linear`' in the file '`linear.f90`'. This should make use of the above procedures to read in the upper triangular matrix and the right hand side vector $\mathbf{v}$ and solve the linear system to give $\mathbf{x}$. Your main program should also print the upper triangular matrix and the solution vector $\mathbf{x}$ to the screen. To demonstrate your program works, use it to solve the linear system in the example above in section (2) and redirect the output from your code into the file '`quest2.txt`'.

On completion of the assignment you should have the files '`quad.f90`', '`quad_mod.f90`', '`linear.f90`', '`linear_mod.f90`', '`quest1.txt`' and '`quest2.txt`' in the directory '`assign3`' ready to be marked. Do not create any sub-directories in your '`assign3`' directory. You will of course need to edit your '`Makefile`' when switching compilation between the two different codes.