Project #1

ES_APPM 444

Due April 22

For each of the programs described below, you must follow these guidelines:

- All code must be properly commented. See examples in the class notes for guidance.
- The program should read its input parameters from a text file given as an argument to `main`. The parameter list must include all parameters as indicated for each problem. A sample input file will be included for each problem. To assist you, your program should begin like this:

```
int main(int argc, char* argv[])
{
    if (argc != 2) {
        printf("Incorrect usage: only enter the input data file name\n");
        return 0;
    }
    FILE* inputfile = fopen(argv[1], "r");
    if (!inputfile) {
        printf("Unable to open input file\n");
        return 0;
    }
    // start reading input data using function fscanf here
    int N;
    fscanf(inputfile, "%d", &N); // read an integer N for example
    // read rest of parameters here
    fclose(inputfile);

    // ... rest of program ...
```

- Your program should record the time elapsed to run. To do this you will do the following:
  1. Add the line
     `#include <time.h>`
  2. The first line of the `main` program should have the line
     `clock_t start = clock();`
  3. The last line before the return statement in the `main` program should be:
     `printf("Time elapsed: %g seconds\n", (float)(clock()-start)/CLOCKS_PER_SEC);`
- Where requested, your program must write your results at the times given to the data files in binary format. A Matlab script will be provided for you to test and plot your output.

I will be comparing the speed of your code against my own.

A template for the main program incorporating the above requirements, called `main_template.c`, is provided on the Canvas site. You may use that template as a starting point for your own programs.

1. Write a program called `mcquad` to do Monte Carlo quadrature for an integral of the form:

$$\int_0^\infty e^{-x} g(x)\, dx$$

As a reminder of how this is done, suppose you wish to calculate an integral of the form

$$\int_{-\infty}^{\infty} f(x) g(x)\, dx$$

where $f(x)$ is a probability distribution function (pdf). To estimate the integral, take N random samples, $x_1 \dots x_N$, from the distribution given by the pdf $f(x)$, and an estimate for the integral is then given by

$$\int_{-\infty}^{\infty} g(x) f(x)\, dx = \langle g(x) \rangle \approx \frac{1}{N} \sum_{k=1}^{N} g(x_k).$$

To draw a random variable from an exponential distribution, which has pdf

$$f(x) = \begin{cases} e^{-x} & x \geq 0 \\ 0 & x < 0 \end{cases}$$

you can use a transformation. Suppose y is a variable taken from a uniform random distribution in the range $y \in [0, 1]$, then x is a random variable drawn from an exponential distribution by setting $x = -\ln(y)$. A reasonable uniform random number generator is the function `drand48()`, which is declared in the header file `stdlib.h`. Thus, at the top of the file, you will have to add the line

```
#include <stdlib.h>
```

and to get a random value drawn from a uniform $[0, 1]$ sample for a double variable use

```
double x;
x = drand48();
```

Also, you should be aware that `drand48()` will always start with the same seed value. That means that if you run your program twice, you will get exactly the same values, which is not ideal for a random number generator. To try different values, you must seed the generator by calling the function `srand48(1L)` where the expression `1L` means the value of one as a long integer. Change the "1" to a different value to get a different initial seed. This should be called only once at the beginning of your program, *not every time you call the random number generator.*

For this problem, you *must* write the function $f(x)$ as a separate function, it may be in the same file as your main program, or in a separate file. The test case provided in the Matlab code is for the case when $g(x) = \cos x$.

There is only one input parameter for this problem, which is N, the number of sample points to be taken. A sample input file is posted on Canvas called `mcquad.in`. You should output two values to the terminal (i.e. use `printf`), the estimate for the integral, and the sample variance given by the formula:

$$\text{var}\{g\} = \frac{1}{N-1} \sum_{k=1}^{N} (g(x_k) - \langle g(x) \rangle).$$

2. Write a program called `burger` to solve the viscous Burgers equation

$$\begin{aligned} u_t + [f(u)]_x &= \nu u_{xx} \\ f(u) &= \frac{u^2}{2} \\ u(x, 0) &= -\sin \pi x \\ u(-1, t) &= u(1, t) = 0 \end{aligned}$$

on the interval $-1 \le x \le 1$ and for the time interval $0 \le t \le 2$.

Use MacCormack's method to solve this system, which is given by the following steps. Suppose $u_j^n \approx u(x_t, t_n)$, then given u_j^n for all j, you advance in time by doing first an FB step:

$$\hat{u}_j^n = u_j^n + \frac{\Delta t}{\Delta x}\left(f(u_{j+1}^n) - f(u_j^n) + \frac{\nu}{\Delta x}\left((u_{j+1}^n - u_j^n) - (u_j^n - u_{j-1}^n)\right)\right)$$
$$u_j^{n+1} = \frac{1}{2}\left(\hat{u}_j^n + u_j^n + \frac{\Delta t}{\Delta x}\left(f(u_j^n) - f(u_{j-1}^n) + \frac{\nu}{\Delta x}\left((u_{j+1}^n - u_j^n) - (u_j^n - u_{j-1}^n)\right)\right)\right)$$

The next step would be a BF step:

$$\hat{u}_j^{n+1} = u_j^{n+1} + \frac{\Delta t}{\Delta x}\left(f(u_j^{n+1}) - f(u_{j-1}^{n+1}) + \frac{\nu}{\Delta x}\left((u_{j+1}^{n+1} - u_j^{n+1}) - (u_j^{n+1} - u_{j-1}^{n+1})\right)\right)$$
$$u_j^{n+2} = \frac{1}{2}\left(\hat{u}_j^{n+1} + u_j^{n+1} + \frac{\Delta t}{\Delta x}\left(f(u_{j+1}^{n+1}) - f(u_j^{n+1}) + \frac{\nu}{\Delta x}\left((u_{j+1}^{n+1} - u_j^{n+1}) - (u_j^{n+1} - u_{j-1}^{n+1})\right)\right)\right)$$

Here, Δx is the space step size, i.e. $\Delta x = x_{j+1} - x_j$, and Δt is the time step size, $\Delta t = t_{n+1} - t_n$. The program will then alternate between FB and BF steps until the terminal time is reached.

There are three input parameters for this problem, N is the number of grid points in space, Δt the time step size, and T the terminal time. A sample input file is provided on Canvas called `burger.in`.

The output should follow this format:

Variable name	data type	length
N	int	1
x	double	N
$u(x,0)$	double	N
$u(x,0.5)$	double	N
$u(x,1)$	double	N
$u(x,1.5)$	double	N
$u(x,2)$	double	N

See Section 5.2 in the course notes for more information about how to write the data using the function `fwrite(...)`. A matlab program called `plotburger.m` is posted on Canvas and can be used to read the data into Matlab for plotting and testing purposes.

3. Write a program called `burger2` to solve the same problem as in #2 except use a split-step MacCormack/Crank-Nicolson method. The update for this problem proceeds as follows, given u_j^n. First, do an FB step, but without the diffusion term:

$$\hat{u}_j^n = u_j^n + \frac{\Delta t}{\Delta x}\left(f(u_{j+1}^n) - f(u_j^n)\right)$$
$$\hat{u}_j^{n+1} = \frac{1}{2}\left(\hat{u}_j^n + u_j^n + \frac{\Delta t}{\Delta x}\left(f(u_j^n) - f(u_{j-1}^n)\right)\right)$$

This is then followed by a Crank-Nicolson step to solve the diffusion problem:

$$u_j^{n+1} = \hat{u}_j^{n+1} + \frac{\nu \Delta t}{2\Delta x^2}\left(\hat{u}_{j+1}^{n+1} - 2\hat{u}_j^{n+1} + \hat{u}_{j-1}^{n+1} + u_{j+1}^{n+1} - 2u_j^{n+1} + u_{j-1}^{n+1}\right)$$

Next do a BF step followed by another Crank-Nicolson step like this:

$$\begin{aligned}
\hat{u}_j^{n+1} &= u_j^{n+1} + \frac{\Delta t}{\Delta x}\left(f(u_j^{n+1}) - f(u_{j-1}^{n+1})\right) \\
\hat{u}_j^{n+2} &= \frac{1}{2}\left(\hat{u}_j^{n+1} + u_j^{n+1} + \frac{\Delta t}{\Delta x}\left(f(u_{j+1}^{n+1}) - f(u_j^{n+1})\right)\right) \\
u_j^{n+2} &= \hat{u}_j^{n+2} + \frac{\nu \Delta t}{2\Delta x^2}\left(\hat{u}_{j+1}^{n+2} - 2\hat{u}_j^{n+2} + \hat{u}_{j-1}^{n+2} + u_{j+1}^{n+2} - 2u_j^{n+2} + u_{j-1}^{n+2}\right)
\end{aligned}$$

Again, the FB and BF steps should alternate as before.

Note that the Crank-Nicolson step is an implicit step, so it means you'll need to use a suitable linear solver from LAPACK in order to solve the linear system. I recommend using the tridiagonal solver, since the matrix will be a tridiagonal matrix.

The input and output information are the same for this problem as for problem #2. A sample input file is given by the file `burger2.in` and a sample Matlab code is given in `burger2.m`