MATH 934 – HEAT EQUATION PROJECT

INSTRUCTOR: DR. ADAM LARIOS

Consider the heat equation (also called the diffusion equation) in 1D:

$$\begin{cases} \dfrac{\partial u}{\partial t} = \nu \dfrac{\partial^2 u}{\partial x^2}, \\ u(x, t_0) = u_0(x). \end{cases}$$

with periodic boundary conditions on an interval of length L. Here, $\nu > 0$ pronounced "nu" is a constant called the *diffusion coefficient* or the *viscosity*. It has units $[\nu] = (\text{length})^2/\text{time}$, as can be seen from examining the PDE. The function u_0 is the initial data, which we will assume to be a square-integrate function, i.e., $\int_0^L |u_0(x)|^2\,dx < \infty$ (normally, this won't be a big issue in our computations).

If we formally express the solution via its Fourier series, we have:

$$u(x,t) = \sum_k \widehat{u}_k(t) e^{ik\frac{2\pi x}{L}}$$

Formally substituting this into equation (1), we obtain the following relationship between the coefficients:

$$\frac{d}{dt}\widehat{u}_k = -\nu k^2 \widehat{u}_k, \qquad \text{for each } k.$$

Thus, for each k, we have an ODE. Let us consider solving it with an explicit time-stepping method, such as Forward Euler or Runge-Kutta-4. We say in class that in the Euler case, for stability, we must choose a time step respecting the viscous CFL (the Courant-Friedrichs-Lewy condition in the parabolic case). Namely, for a space step size of Δx, we must choose our time step Δt so that

$$\Delta t < \frac{2}{\pi^2}\frac{(\Delta x)^2}{\nu} \approx 0.2\frac{(\Delta x)^2}{\nu}.$$

In general, <u>*any*</u> explicit method for solving the heat equation will have a time-step restriction of the form

$$\Delta t < K\frac{(\Delta x)^2}{\nu},$$

where the dimensionless constant K comes from the numerical scheme. In many popular schemes, usually $0.1 \lesssim K \lesssim 1$. For example, for the 1D heat equation using a centered finite difference scheme and Euler time-stepping, $K = 1/2$, and for the 2D version (with $\Delta x = \Delta y$), $K = 1/4$.

Note: We will see later that the CFL condition for hyperbolic problems such as the transport equation and the wave equation is $\Delta t < K\Delta x/c$, where K is a dimensionless constant, and c is the velocity or wave-speed with units $[c] = \text{length}/\text{time}$.

<u>Task 1</u>: Write a MATLAB code to solve the 1D heat equation using spectral (i.e. Fourier) methods for the spatial component, and Runge-Kutta-4 for the time stepping. Examining the file `FFTderivatives.m` on the webpage may be instructive here. Use the following parameters:

- $L = 2\pi$
- $\nu = 0.01$
- $N = 256$ (i.e., 256 points in space)
- Use a Δt which respects the CFL. (Don't hard-code it, base it off the other parameters.)
- Initial data that is periodic, and has a highly oscillatory part, and a non-so oscillatory part.

Plot the solution in real-time by putting a plot statement in the loop. You must convert back to physical space each time to do this. To see time going by, use this following line right after your `plot` statement:

```
title(sprintf('u(x,%1.3f)',t)); % t = the current time (scalar value)
```

Congratulations! You just solved a PDE using a computer!

<u>**Task 2**</u>**:** Investigate the following questions:

(a) What happens when ν is too big? Too small?
(b) What happens when you violate the CFL, i.e., choose Δt too large?
(c) Can you add a forcing function to the right-hand side? That is, can you solve

$$\begin{cases} \dfrac{\partial u}{\partial t} = \nu \dfrac{\partial^2 u}{\partial x^2} + f(x,t), \\ u(x,t_0) = u_0(x). \end{cases}$$

(Choose f to be a function which is periodic in space.)
(d) Can you modify your program to allow for arbitrary-length intervals?
(e) How many spatial points can your computer handle without taking too much time? Make sure to always choose $N = 2^m$ for some integer m (what happens if you don't?). To stop a computation in Matlab, on your keyboard, push:

`[CTRL]+C`

(Try not to melt your computer.)
(f) What other things can you try with your code?

You don't need to investigate the task 2 questions exhaustively, I just want to you play around with your code, try to break it, find its limits, and see what more it can do.

Have fun, and happy coding!