Parabolic Equation in One Space Dimension

This project is due at 11 PM on Tuesday, May 17.

You are to implement and test a solver for a scalar one dimensional reaction diffusion equation. This project uses the Galerkin method based on continuous piecewise linear functions that you developed in the previous project. In this project you will add step doubling with local extrapolation to get a second-order correct scheme. You will also use adaptive time stepping to make the solution process more robust.

In this project you are to aim at making a "package" that approximates the solutions of problems of this class with minimal changes to low level code. In particular, there should be a class that is called `parabolicProblem` that contains the information that defines the differential problem and its spatial discretization, and a class called `simTime` that contains the information used in the adaptive time stepping process. These classes are described in some detail in the description of project 3 and in the write-up on time step control in the course readings.

You should have a function that initializes the state of the approximate solution (based on the information in a parabolicProblem object) and initializes the simTime object that will guide your time integration.

Advancing Time

The principal innovation that you are making here is the addition of a function `advance` that moves the state of the system forward to a given point in time using the logic for adaptive time steps and the approximate backward difference method from project 3. This routine should use step doubling with local extrapolation to compute trial solutions. The error indicator should be the max norm in space of the difference between S an D, where S is produced by one step of backward difference with step `dt` and D is produced by two steps of backward difference with step `dt/2`.

For each accepted step, `advance` should append to a list several pieces of information including

1. the `dt` used to advance time and $log_{10}(dt)$,
2. the value of the error indicator,

3. the steps since a step rejection,

4. the values of U(a,t) and U(b,t) after updating the solution.

These are useful in debugging the adaptive process and showing the function of the code. The list should be initialized with these items for the solution before any call to `advance`, so that plots of some items versus time can start at the starting time of the simulation. **For each case below, you should include a plot of `dt` versus time.** In addition, for each case you should print a short report that says how many steps were accepted and how many were rejected.

Demonstrating Your Code

These test cases are almost the same as the cases used in project 3. In these cases use $[a, b] = [0, 1]$, a mesh with $N = 30$ intervals and $x[i] = p(i/N)$, where $p(x) = x + 0.9 * x * (1 - x)$. For each case (except as noted) you should use `dtmin = 1e-4`, `dtmax = 1e-1`, `tol = 1e-2`. To make it easy to compare results, on the very first step start with `dt = dtmin`

The plots should have titles and/or captions that indicate what the plot is showing.

Case 0. Use NN boundary conditions with $u_a(t) = u_b(t) = 1$ and initial data $u_0(x) = x$. Take $c(x) = d(x) = 1$ and $f(x, u) = 0$. Plot the solution at $t = 0, 0.05, 0.1, 0.2, 1.0$. The solution should converge to $1 - x$.

Case 1. Use ND boundary conditions with $u_a(t) = 0$, $u_b(t) = 1$ and initial data $u_0(x) = 0$. Take $c(x) = d(x) = 1$ and $f(x, u) = 0$. Plot the curves $u(0, t)$ for $t \in [0, 2]$ and $u(x, 2)$ for $x \in [0, 1]$.

Case 2. Like Case 1, except use $c(x) = d(x) = x$.

Case 3. Like Case 1, except use $c(x) = d(x) = x^2$.

Case 4. Like Case 0, except use $u_a(t) = u_b(t) = 0$, $u_0(x) = 1$, and $f(x, u) = u$. The solution of the differential problem is $u(x, t) = exp(t)$

Case 5. Like Case 2, except use `tol = 1e-3`.

Last modified 5/11/22 – tfd