University of Colorado
Department of Computer Science

Numerical Computation CSCI 3656

Spring 2016

Problem Set 13

Issued: 21 April 2016

Due: 28 April 2016

1. *[30 pts]* Modify your code from PS12 to solve the Lorenz equations using the *trapezoidal* method instead.

   Generate two 10,000-point-long trajectories using $h = 0.01$, one from the initial condition $[x, y, z]^T = [1, 1, 1]^T$ and the other from the initial condition $[x, y, z]^T = [1.01, 1.01, 1.01]^T$. Make time-domain plots of the x coordinates and state-space plots for both trajectories.

   Now explore the difference between forward Euler and trapezoidal by generating trajectories from $[x, y, z]^T = [1, 1, 1]^T$ for a range of different time steps (e.g., $h = 0.01$, $h = 0.001$, $h = 0.0001$) with these two solvers and make time-domain plots of their x coordinates. In these plots, keep the overall length the same—length in *time*, that is, not the number (N) of points. (Recall that h is a time interval.) You'll probably want to zoom in on the first second or so of the plot in order to focus this exploration.

   Comment on the similarities and differences between the results produced by these two solvers. Which are better, would you think? Why?

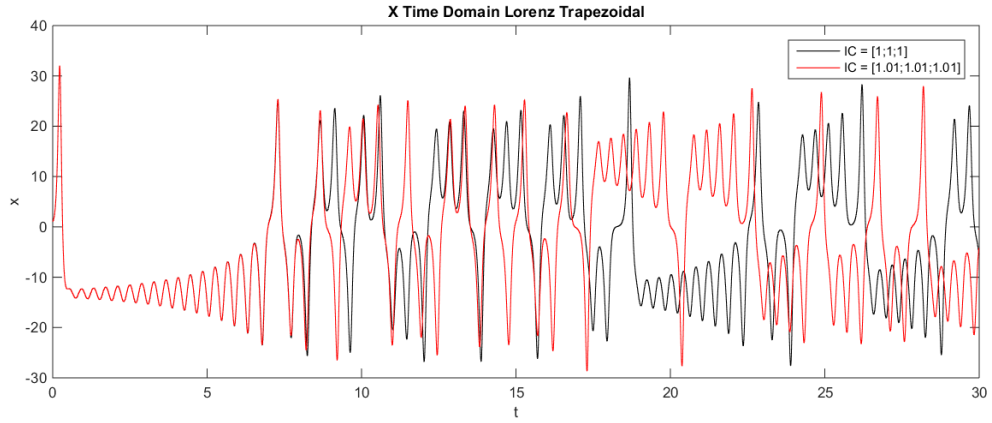
   Please turn in your plots, your thoughts, and a copy of your code.

   Here's some Matlab code for this:

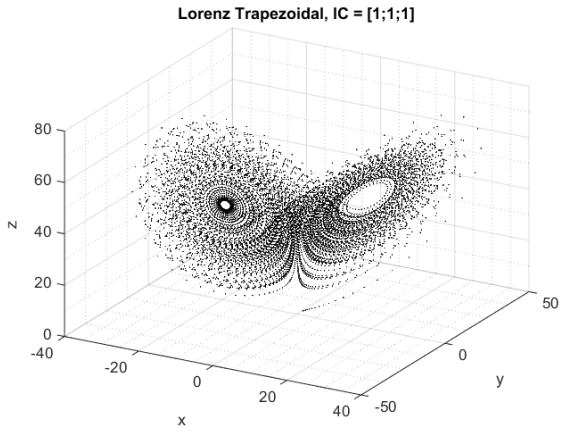
```
function [trajectory] = Lorenz_Trap(current_state,h,N)

    %% this is the system derivative: the box of math that takes values
    %% for the state variables and returns values for their slopes:
    function [slope]=lorenz(where)
        x=where(1,1);
        y=where(2,1);
        z=where(3,1);
        xdot=16*(y-x);
        ydot=45*x - y - x*z;
        zdot=x*y-4*z;
        slope=[xdot,ydot,zdot]';
    end

trajectory=[];
for steps=1:N
  % first take a forward Euler step:
```

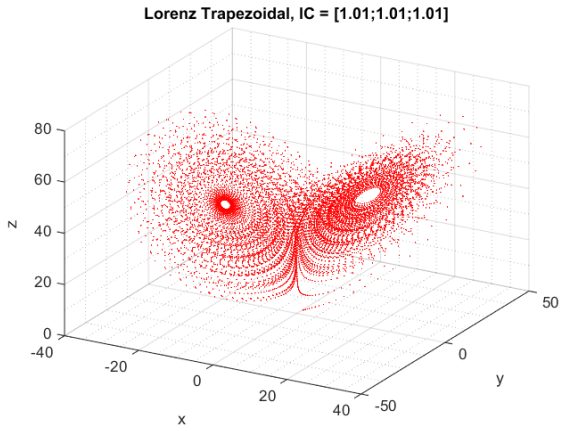
```
FE_stepped_state = current_state + h * lorenz(current_state);
  % average the slope at that point with the slope at the original
  % point, then move forwards along *that* slope:
new_state = current_state + ...
    h/2 * (lorenz(current_state) + lorenz(FE_stepped_state));
  % BTW, if you changed that line to
  %   new_state = current_state + h * lorenz(FE_stepped_state);
  % ... you'd have *backward* Euler
trajectory=[trajectory,new_state];
current_state=new_state;
end
end
```

Here is a time-domain plot of the first 3000 points of the x values from the two initial conditions:



Here are state-space plots for the different initial conditions:

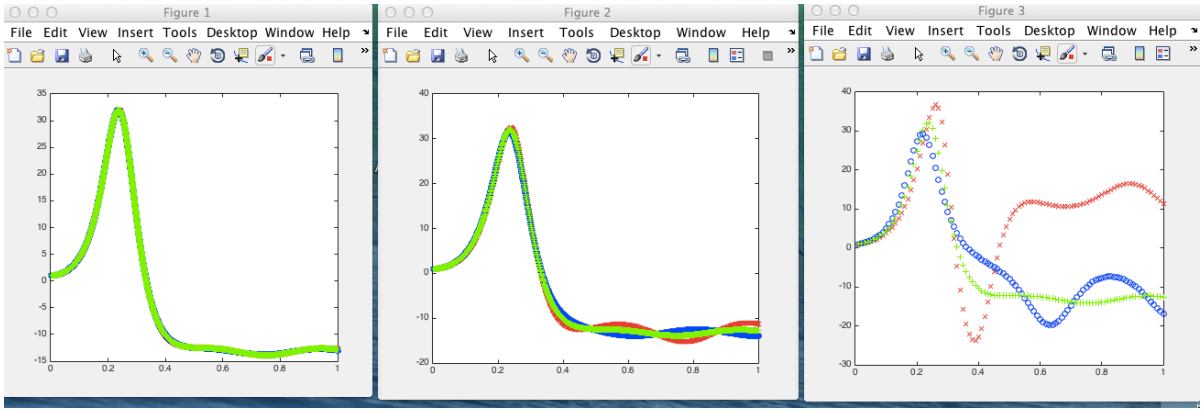




What we were looking for here was that you noticed that the same "sensititive dependence on initial conditions" thing was going on in the trapezoidal results as in the forward Euler results—i.e., that the time course of the trajectories emanating from those two nearby initial conditions were different, as they were in PS12, but that the overall structure of the state-space plot looked pretty similar.

We also wanted you to think a bit about the difference between the trajectories produced by the two methods from the same initial condition, with the same step size, and that you mentioned that the trapezoidal results are better in theory—$O(h^2)$ versus forward Euler's $O(h)$.

To really dig into that difference, you have to zoom in on the trajectories generated with the different time steps. Below are time-domain plots of the first second's worth of the x coordinate of the solution from $[x, y, z] = [1, 1, 1]^T$ with forward Euler (red $\times$s), backward Euler (blue circles), as well as trapezoidal (green +s), for three different time steps: 0.0001, 0.001, and 0.01, from left to right.



For the really small time step, all three solvers do really well. As the time step increases, the Euler methods start to lose it, but trapezoidal still does pretty well.