CS 322 Homework 7

out: Thursday 12 April 2007
due: Wednesday 18 April 2007

Ordinary Differential Equations

Problem 1: Problem 7.1 in Moler

Answer: Our state vector is

$$\mathbf{y} = \begin{bmatrix} u \\ v \\ a \\ b \end{bmatrix} \tag{1}$$

where $a = \dot{u}$ and $b = \dot{v}$. Our ODE is then of the form

$$\dot{\mathbf{y}} = \begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{a} \\ \dot{b} \end{bmatrix} = \mathbf{f}(t, \mathbf{y}) = \begin{bmatrix} a \\ b \\ \frac{v}{1+t^2} - \sin\left(\sqrt{a^2+b^2}\right) \\ \frac{-u}{1+t^2} + \cos\left(\sqrt{a^2+b^2}\right) \end{bmatrix} \tag{2}$$

$$\mathbf{y}_0 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{3}$$

Problem 2: Stiff Systems

Take the sample ODE given on Moler p. 205

$$\begin{aligned} \dot{y} &= y^2 - y^3 \\ y(0) &= \delta \\ 0 \leq t &\leq \frac{2}{\delta} \end{aligned}$$

1. Implement three MATLAB functions

   ```
   function y = intFlameForwardEuler(delta, h)
   function y = intFlameMidpoint(delta, h)
   function y = intFlameBackEuler(delta, h)
   ```

   that integrate the above ODE given the parameter δ and the fixed timestep h to step over the specified interval of t using the appropriate fixed-stepsize integrator (Forward Euler, Midpoint, and Backward Euler). Your functions should return a vector $\mathbf{y}$ where $\mathbf{y}(i)$ is the computed value of y at timestep $t_i = ih$.

   Answer:

   ```
   function y = intFlameForwardEuler(delta, h)
   % Forward euler integrator

   tEnd = 2 / delta;
   numSteps = ceil(tEnd / h);
   ts = tEnd / numSteps;

   y = zeros(numSteps, 1);

   yp = delta;

   for i = 1:numSteps
       y(i) = yp + ts*(yp^2 - yp^3);
       yp = y(i);
   end



   function y = intFlameMidpoint(delta, h)
   % Midpoint integrator
   tEnd = 2 / delta;
   numSteps = ceil(tEnd / h);
   ts = tEnd / numSteps;
   ```

```
y = zeros(numSteps, 1);

yp = delta;

for i = 1:numSteps
    k1 = (yp^2 - yp^3);
    yhalf = yp + ts*k1 / 2;
    y(i) = yp + ts*(yhalf^2 - yhalf^3);
    yp = y(i);
end
```

```
function y = intFlameBackEuler(delta, h)
% Backward Euler integrator
tEnd = 2 / delta;
numSteps = ceil(tEnd / h);
ts = tEnd / numSteps;

y = zeros(numSteps, 1);

yp = delta;

for i = 1:numSteps
    coeffs = [-ts ts -1 yp]';
    soln = roots(coeffs);
    soln = soln(imag(soln) == 0);
    % May have multiple solutions.  Find solution
    % closest to current point
    distVec = soln - repmat(yp, 3, 1);
    dist = distVec .* distVec;
    [v, ind] = min(dist);
    y(i) = soln(ind);
    yp = y(i);
end
```

One issue to note is that for large timesteps for Backward Euler, multiple real solutions could exist. In this case, we attempt to do something reasonable to find the one solution we are interested in, and take the solution that is closest to the current value of y_n.

2. Plot the computed value of y for $\delta = 0.0005$ with each of your methods using at least three different choices of timesteps. What do you observe about the solution

computed by each of these methods? What can you say about the order of accuracy (how fast the answer gets better as a function of stepsize h) and stability (stepsizes h for which the answer does not diverge) for each of the three methods on this problem? Submit your plots along with your answer. Your code should be submitted through CMS.

Answer: Code to plot results:

```
function plotODEError(fun, times, fName)

figure();
axes();
n = size(times, 2);
subplot(n, 2, 1);

yt = intFlameMidpoint(0.0005, 1e-3);
tEnd = 2 / 0.0005;
for t = 1:n
    y = fun(0.0005, times(t));

    numSteps = ceil(tEnd / times(t));
    ts = tEnd / numSteps;
    step = ts / 1e-3;
    err = abs(y - yt(step:step:end));
    subplot(n, 2, 2*t-1);
    nStep = size(y, 1);
    tStep = times(t)*(1:nStep);
    plot(tStep, y);
    title(sprintf('Plot of %s with h = %g', fName, times(t)));
    subplot(n, 2, 2*t);
    plot(tStep, err);
    title(sprintf(['Error of %s with h = %g\n']...
                  ['compared to midpoint, h = 1e-3'], ...
                  fName, times(t)));
end
```

See attached graphs. We observe that for Forward Euler, the accuracy seems to improve linearly as a function of the timestep (0.1 is about ten times better than 1). At around $h = 2$, some mild instabilities are displayed, but it is able to stay at the solution within reason. At $h = 3$ the instability becomes obvious — this is very near the edge of the stability region for this problem and integrator, as the solution varies wildly. Finally, at $h = 5$ (and $h = 4$, not shown), the solution explodes.

As for Midpoint, we observe that the accuracy improves roughly as the square of the timestep — $h = 0.1$ is about 100 times more accurate than $h = 1$. The stability boundary moves slightly, as the behavior we were seeing in Foreward Euler at $h = 3$ is not displayed in Midpoint until $h = 3.5$. Still, it is right at the edge of the stable solution, and in fact at $h = 4$ the solution diverges.

Finally, for backward euler, we note that the accuracy improves about as well as forward euler. This is to be expected, as they are the same order of accuracy. The advantage of backward euler, though, is in the timesteps we are allowed to take. Backward euler is able to take increasingly large timesteps without displaying any divergent behavior, even at $h = 100$ (well outside the stability regions of Forward Euler and Midpoint). Even though the error can be considered to be pretty poor in this region, it is always able to eventually track the true solution.

Problem 3: [$\star$] Error analysis

Consider the following ODE integrator, which comes from the Adams family of methods

$$u_{n+1} = u_n + \frac{h}{2}\left(3f(t_n, u_n) - f(t_{n-1}, u_{n-1})\right)$$

Using the methods discussed on Moler p. 215, show that the local discretization error of this method is $O(h^3)$, and so the method is of order 2.

Answer: Assume that we have the exact answer for $u(t_n) = u_n$. Substitute this into the right hand side of the equation:

$$u_{n+1} = u_n + \frac{h}{2}\left(3f(t_n, u_n) - f(t_{n-1}, u_{n-1})\right)$$
$$u_{n+1} = u(t_n) + \frac{h}{2}\left(3f(t_n, u(t_n)) - f(t_{n-1}, u_{n-1})\right)$$

We note that by definition of the ODE, $u'(t) = f(t, u(t))$ and so we can substitute

$$u_{n+1} = u(t_n) + \frac{h}{2}\left(3u'(t_n) - u'(t_n - h)\right)$$

Perform a Taylor expansion on $u'(t_n - h)$ about t_n:

$$u_{n+1} = u(t_n) + \frac{h}{2}\left(3u'(t_n) - \left(u'(t_n) - hu''(t_n) + \frac{h^2}{2}u'''(t_n) + O(h^3)\right)\right)$$
$$u_{n+1} = u(t_n) + \frac{h}{2}\left(2u'(t_n) + hu''(t_n) - \frac{h^2}{2}u'''(t_n) + O(h^3)\right)$$
$$u_{n+1} = u(t_n) + hu'(t_n) + \frac{h^2}{2}u''(t_n) - \frac{h^3}{4}u'''(t_n) + O(h^4)$$

However, we note that performing a Taylor expansion on $u(t_n + h)$ about t_n gives us

$$u(t_n + h) = u(t_n) + hu'(t_n) + \frac{h^2}{2}u''(t_n) + \frac{h^3}{6}u'''(t_n) + O(h^4)$$

Calculating $u(t_n + h) - u_{n+1}$ leaves us with $\frac{-h^3}{12}u'''(t_n) + O(h^4)$, leaving us with an error of at most $O(h^3)$, and so we can conclude that the order of accuracy for this method is $p = 2$.

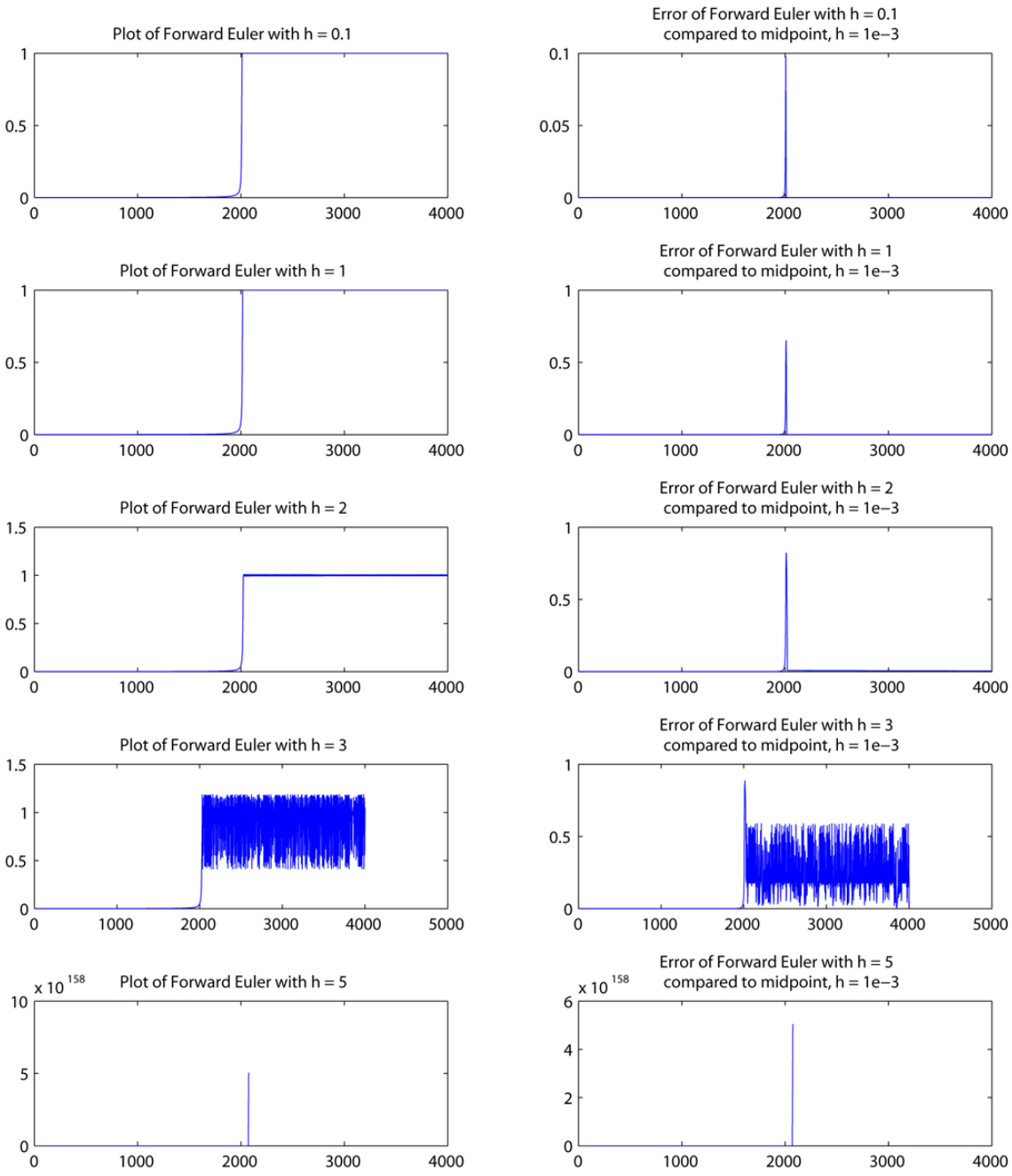

Figure 1: Graphs of Forward Euler and error for various timesteps

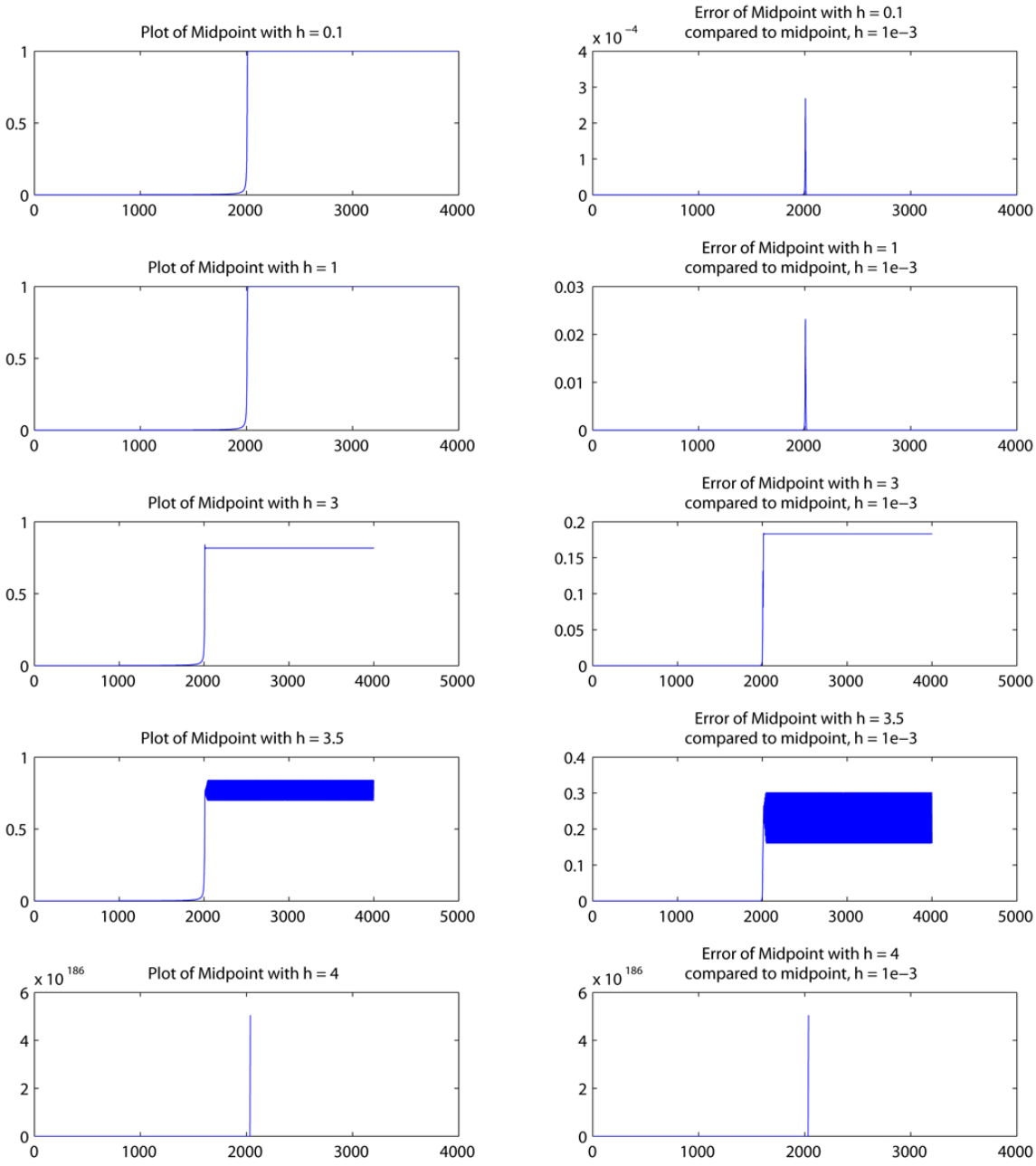

Figure 2: Graphs of Forward Euler and error for various timesteps

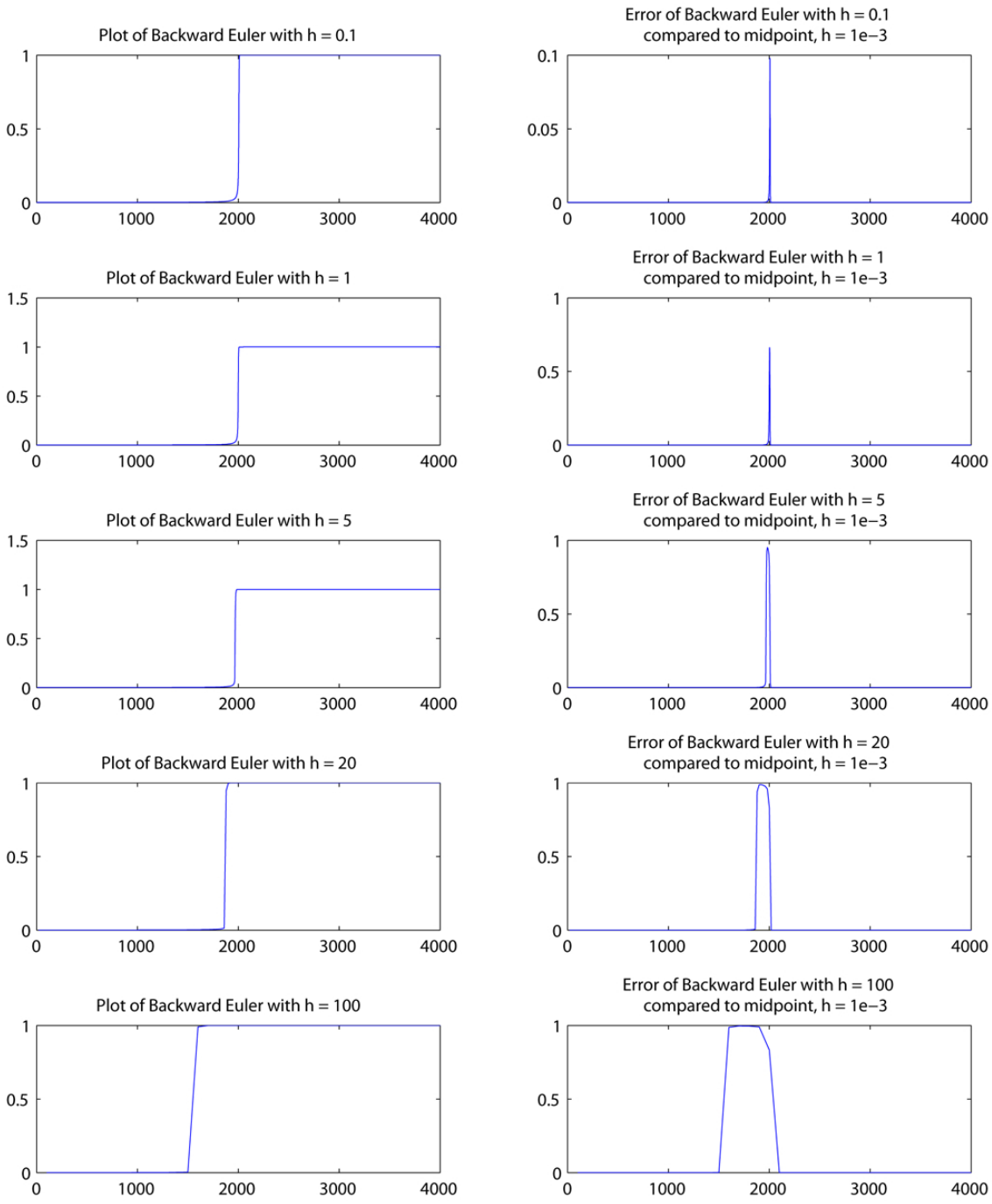

Figure 3: Graphs of Forward Euler and error for various timesteps