

MATH 335: Numerical Analysis

Problem Set 19, Solutions

★39 (b) There is an error in the original problem. I didn't want you to iterate

$$z_{k+1} = y_j + hf(t_j, z_k)$$

(which is a quite different solution method from the one used in part (a)). Rather, it ought to have been

$$z_{k+1} = y_j + \frac{h}{2} [f(t_j, y_j) + f(t_{j+1}, z_k)].$$

I have not graded the problem, but the numerical results I give below correspond to this corrected version.

(c) In the table below, I look at the value of the approximate solution at $t = 2$ for various h -values. It was necessary to use more h -values than the problem specified in order to get the ratios of errors to "settle in". It appears halving the stepsize halves the error for both Euler approaches, but cuts the error by 75% for the Trapezoid approach, suggesting the Euler approaches are $O(h)$ and the Trapezoid is $O(h^2)$.

Method	Trial n	h_n	Soln. @ $t = 2$	Error E_n	Ratio $ E_{n-1}/E_n $
Imp. Euler (from class) (solver=Newton)	1	0.2	-0.717620	2.717620	
	2	0.1	-0.639953	2.639953	1.0294
	3	0.05	4.892828	-2.892828	0.9126
	4	0.025	2.494488	-0.494488	5.8501
	5	0.0125	2.202193	-0.202193	2.4456
	6	0.00625	2.093074	-0.093074	2.1724
Imp. Trapezoid (part (a)) (solver=Newton)	1	0.2	2.996380	-0.996380	
	2	0.1	2.157505	-0.157505	6.3260
	3	0.05	2.033445	-0.033445	4.7094
	4	0.025	2.008077	-0.008077	4.1408
	5	0.0125	2.002003	-0.002003	4.0325
	6	0.00625	2.000500	-0.000500	4.0060
Imp. Trapezoid (part (b)) (solver=fixed pt. iter.)	1	0.2	7.496477	-5.496477	
	2	0.1	2.156895	-0.156895	35.0328
	3	0.05	2.033136	-0.033136	4.7349
	4	0.025	2.007709	-0.007709	4.2984
	5	0.0125	2.001650	-0.001650	4.6721
	6	0.00625	2.000277	-0.000277	5.9567

The code used to generate most of these values is given below.

```

% Initialize values
h = 0.0125/2;
t0 = 0;
tfin = 2;
N = tfin / h;
tjs = (t0:h:tfin)';

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This part devoted to the problem of *39(a)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
y1 = [2/5];

% Define relevant functions
function out = f(t, yin)
    out = t.*yin.^2;
end

function out = fy(t, yin)
    out = 2*t.*yin;
end

function root = nrm(f, df, yLatest, tnext, h, tol, maxIt)
    z = yLatest + h*f(tnext - h, yLatest);
    zold = z + 100;    % make z's initially quite far apart so loop will start
    k = 0;
    while ((norm(z - zold) >= tol) & (k < maxIt))
        k++;
        zold = z;
        z = zold - ...
            (yLatest - zold + h*(f(tnext-h, yLatest) + f(tnext, zold)) / 2) ...
            / (h*df(tnext, zold) / 2 - 1);
    end
    root = z;
end

% solution process
for jj = 1:N
    ynew = nrm(@f, @fy, y1(end), tjs(jj+1), h, 10^(-4), 10);

```

```

    y1 = [y1; ynew];
end

ts = (0:.01:2)';
plot(ts, 2./(5 - ts.^2), 'b-')
hold on
plot(tjs, y1, 'r*')
hold off
axis([0 2 0 3])

pause

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This part devoted to *39(b)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
y2 = [2/5];

function root = iterMeth(f, yLatest, tnext, h, tol, maxIt)
    z = yLatest + h*f(tnext - h, yLatest);
    zold = z + 100;    % make z's initially quite far apart so loop will start
    k = 0;
    while ((norm(z - zold) >= tol) & (k < maxIt))
        k++;
        zold = z;
        z = yLatest + h*(f(tnext - h, yLatest) + f(tnext, zold)) / 2;
    end
    root = z;
end

% solution process
for jj = 1:N
    ynew = iterMeth(@f, y2(end), tjs(jj+1), h, 10^(-4), 10);
    y2 = [y2; ynew];
end

ts = (0:.01:2)';
plot(ts, 2./(5 - ts.^2), 'b-')
hold on
plot(tjs, y2, 'r*')
hold off

```

```

axis([0 2 0 3])

pause

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This is the code from class for implicit Euler, the
% code on which the code for parts (a), (b) is based.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
y3 = [2/5];

% solution process
for jj = 1:N
    ynew = nrm(@f, @fy, y3(end), tjs(jj+1), h, 10^(-4), 10);
    y3 = [y3; ynew];
end

ts = (0:.01:2)';
plot(ts, 2./(5 - ts.^2), 'b-')
hold on
plot(tjs, y3, 'r*')
hold off
axis([0 2 0 3])

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Now view errors at t = 2.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
at2 = 2/(5 - 2^2);
%disp('-----')
disp('-----')
disp(' method          h          val.@ t=2          error ')
disp('-----')
fprintf('imp. Trap.-NR    %2.2f %12.6f %12.6f\n',h,y3(end),at2-y3(end))
fprintf('imp. Trap.-iter  %2.2f %12.6f %12.6f\n',h,y1(end),at2-y1(end))
fprintf('imp. Euler-NR     %2.2f %12.6f %12.6f\n',h,y2(end),at2-y2(end))

```

- ★40 (a) i. Here $\partial f/\partial y = -t(1 + t^2) \sin(ty)$ which, for $(t, y) \in D$ (in particular, $t \in [0, 1]$), satisfies

$$\left| \frac{\partial f}{\partial y} \right| = t(1 + t^2) |\sin(ty)| \leq (1)(2)(1) = 2.$$

Thus, by Exercise ★37, part (a), this f is Lipschitz.

- ii. Since $\partial f/\partial y = 0$, this function is Lipschitz.
 iii. If $y_1, y_2 > 1$, then

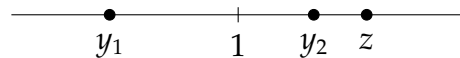
$$|f(t, y_2) - f(t, y_1)| = \left| |y_2 - 1| - |y_1 - 1| \right| = |y_2 - 1 - (y_1 - 1)| = |y_2 - y_1|.$$

If $y_1, y_2 < 1$, then

$$|f(t, y_2) - f(t, y_1)| = \left| |y_2 - 1| - |y_1 - 1| \right| = |1 - y_2 - (1 - y_1)| = |y_2 - y_1|.$$

If the two values are on opposite sides of 1—say, $y_1 < 1$ and $y_2 > 1$ —then set $z = 1 + (1 - y_1) = 2 - y_1$. That is, $z > 1$ is the point just as far away from 1 as y_1 is. Then

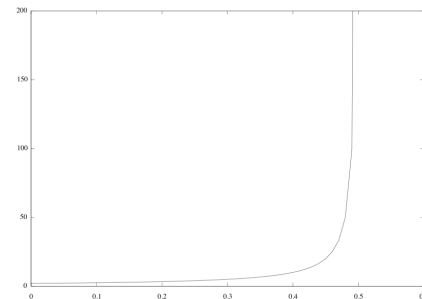
$$\begin{aligned} |f(t, y_2) - f(t, y_1)| &= \left| |y_2 - 1| - |y_1 - 1| \right| \\ &= \left| |y_2 - 1| - |z - 1| \right| \\ &= |y_2 - z| \\ &\leq |y_2 - y_1|. \quad (\text{See picture.}) \end{aligned}$$



In all cases, then $|f(t, y_2) - f(t, y_1)| \leq |y_2 - y_1|$, showing f to be Lipschitz with constant $L = 1$.

- (b) i. With commands like

```
octave:73> function out = f(t, yin)
> out = yin.^2;
> end
octave:75> [y, t] = rk4(@f, 0, 1, 2, 100);
octave:76> plot(t, y, 'k-')
octave:81> axis([0 0.6 0 200])
```



we get the plot at right.

The solution reaches a vertical asymptote (and cannot continue past it) well before the value $t = 1$.

- ii. For $y_1, y_2 \in \mathbb{R}$, we look at the quantity

$$|f(t, y_2) - f(t, y_1)| = |y_2^2 - y_1^2| = |y_2 + y_1| |y_2 - y_1|.$$

Since this is strict equality throughout, we see that there is no *constant* L for which $|f(t, y_2) - f(t, y_1)| \leq L|y_2 - y_1|$ holds. For instance, if we take $y_1 = 0$ and $y_2 = 1$ (just 1 unit apart) we have

$$|f(t, 1) - f(t, 0)| = (1)|1 - 0|.$$

But add 100 to both these numbers, the result is two numbers still only 1 unit apart, but now

$$|f(t, 101) - f(t, 100)| = (201)|1 - 0|.$$

Following this, if we were to want two numbers y_1, y_2 which were 1 unit apart but $|f(t, y_2) - f(t, y_1)| = M|y_2 - y_1|$ (for M arbitrarily large), we roughly get them by taking $y_1 = M/2$ and $y_2 = 1 + M/2$.

- ★41 (a) I do not expect *your* routine to work for inputs $n > 5$. Nevertheless, here is a routine that will:

```
function [wtsAB, wtsAM] = adamsWts ( n )
% function [wtsAB, wtsAM] = adamsWts ( n )
%
% Routine the weights for nth-order Adams-Bashforth and
% Adams-Moulton methods.

q = n - 1;
wtsAB = [];
wtsAM = [];
for j = 0:q
    LABj = 1;
    for k = [0:(j-1) (j+1):q]
        LABj = conv(LABj, [1 k]/(k - j));
    end
    antideriv = polyint(LABj, 0);
    wtsAB = [wtsAB; polyval(antideriv, 1)];
    wtsAM = [wtsAM; -polyval(antideriv, -1)];
end
end
```

- (b) Here is my routine. Notice that I have embedded RK4 steps *inside this routine* instead of calling a separate RK4 routine. Without doing so, I would not be taking advantage of one of the desirable features of multistep methods—the “recycling” of function evaluations.

```
function [y, t] = abm(f, t0, tlast, y0, order, N)
% function [y, t] = abm(f, t0, tlast, y0, order, N)
%
% This routine calculates an approximate solution at mesh points
% t0, t1, ..., tN to the initial-value problem
```

```

%
%          y' = f(t, y),    subject to y(t0) = y0,
%
% using the Adams-Bashforth-Moulton predictor-corrector of specified
% order. We assume evenly-spaced mesh points, with stepsize
%
%          h = (tlast - t0) / N.
%
% INPUTS:
%   f      function handle to the RHS in the ODE giving y'
%   t0     initial "time"
%   tlast  last "time" at which to approximate the solution
%   y0     initial value, the value of the solution at the initial time
%   order  order of the Adams-Bashforth-Moulton method to use
%   N      number of steps to take to go from t0 to tN
%
% OUTPUTS:
%   y      approximate value of solution at mesh points
%   t      mesh points

h = (tlast - t0) / N;
t = [t0:h:tlast]';
y = y0;
prev_f = [];

% Compute first few steps using RK4 method
for jj = 1:min(order-1, N)
    k1 = f(t(jj), y(jj));
    k2 = f(t(jj) + h/2, y(jj) + h*k1/2);
    k3 = f(t(jj) + h/2, y(jj) + h*k2/2);
    k4 = f(t(jj) + h, y(jj) + h*k3);
    y(jj + 1) = y(jj) + h*(k1 + 2*k2 + 2*k3 + k4) / 6;
    prev_f = [k1 prev_f];
end
prev_f = [f(t(jj+1), y(jj+1)) prev_f];

[wtsAB, wtsAM] = adamsWts(order);
for jj = order:N
    y_pred = y(jj) + prev_f * wtsAB * h;
    prev_f = [f(t(jj+1), y_pred) prev_f(1:length(prev_f)-1)];

```

```
    y(jj+1) = y(jj) + prev_f * wtsAM * h;  
    prev_f(1) = f(t(jj+1), y(jj+1));  
end  
end
```