553.481/681 Numerical Analysis

Homework 7 Solutions
Problem 1. (a) Consider the following implicit 4-step method:

\[ y_{n+1} = \frac{8}{15} y_{n-1} - \frac{1}{15} y_{n-3} + h \left[ \frac{2}{21} y_{n+1} + \frac{32}{21} y_{n} - \frac{2}{21} y_{n-3} \right]. \]

(b) Show this method is consistent & 4th-order and find the leading truncation error.
(c) Show that this method is not relatively stable. Hint: If \( p(x), q(x) \) are polynomials of degree \( n \) and if \( p(x) \) has \( n \) distinct roots \( r_j, j = 1, \ldots, n \), then show that the roots of \( P(x; z) = p(x) + zq(x) \) are given by \( r_j(z) = r_j + zq(r_j)/p'(r_j) + O(z^2), j = 1, \ldots, n. \)

(b) The 4-step method is consistent because the equations

\[ a_0 + a_1 + a_2 + a_3 = 1, \quad -a_1 - 2a_2 - 3a_3 + b - 1 + b_0 + b_1 + b_2 + b_3 = 1 \]

are satisfied, where \( a_1 = 8/7, a_3 = -1/7, b - 1 = 2/7, b_0 = 32/21, b_3 = -2/21 \), and all the other \( a_i \)'s and \( b_i \)'s vanish. To establish order 4 we check that

\[ (-1)^i a_1 + (-2)^i a_2 + (-3)^i a_3 + i [b - 1 + (-1)^{i-1} b_1 + (-2)^{i-1} b_2 + (-3)^{i-1} b_3] = 1, \quad i = 2, 3, 4. \]

We use the relation

\[ c_5 = 1 - \sum_{j=0}^{3} (-j)^5 a_j - 5 \sum_{j=-1}^{3} (-j)^4 b_j = \frac{32}{7}. \]

We thus conclude

\[ T_n(R_5) = \frac{c_5}{5!} f^{(5)}(x_n) h^5 + O(h^6). \]

(c) Let \( y_n = r^n \), with \( \dot{y}_n = \lambda y_n = \lambda r^n \). This yields the characteristic polynomial

\[ \rho(r) - h \lambda \sigma(r) = r^4 - \frac{8}{7} r^2 + \frac{1}{7} - h \lambda \left[ \frac{2}{7} r^4 + \frac{32}{21} r^3 - \frac{2}{21} \right]. \]

Take \( \rho(r) = 0 \) to get \( 7r^4 - 8r^2 + 1 = (r^2 - 1)(7r^2 - 1) = 0 \) and \( r = \pm 1, \pm 1/\sqrt{7} \). Per the hint:

\[ r(\lambda h) = r + \lambda h \sigma(r)/\rho'(r) = r - \lambda h \frac{2r^4/7 + 32r^3/21 - 2/21}{4r^3 - 164/7} + O(h^2 \lambda^2). \]

We find \( r_0(\lambda h) = 1 + h \lambda + O(h^2 \lambda^2) \) and \( r_1(\lambda h) = -1 + 7h \lambda/9 + O(h^2 \lambda^2) \). Evidently \( |r_0(\lambda h)| < |r_1(\lambda h)| \) for small negative \( h \lambda \) and the method is not relatively stable.

To prove the hint observe that differentiating \( P(r_j(z); z) = 0 \) gives \( p'(r_j(z)) r_j'(z) = -q(r_j(z)) - zq'(r_j(z)) r_j'(z) \). When \( z = 0 \) we get \( r_j'(0) = q(r_j)/p'(r_j) \) and thus by Taylor expansion \( r_j(z) = r_j + z r_j'(0) + O(z^2) \), completing the argument.
Problem 2. For root $r_0(h\lambda)$ of the midpoint method, take $h = t/n$ and show that

$$r_0^n = \exp[\lambda t + O((\lambda t)^3/n^2)].$$

*Hint:* Consider $\ln(r_0^n)$ and use $\text{arcsinh}(x) = \ln(x + \sqrt{1 + x^2}) = x - x^3/6 + O(x^5)$.

The characteristic polynomial for the midpoint method is

$$r^2 = 1 + 2(h\lambda)r$$

whose principal root is $r_0 = h\lambda + \sqrt{1 + h^2\lambda^2}$. Then

$$\log r_0^n = n \log \left( h\lambda + \sqrt{1 + h^2\lambda^2} \right) = nh\lambda - \frac{nh^3\lambda^3}{6} + O(nh^5\lambda^5) = \lambda t + O(\lambda^3t^3/n^2).$$

Thus $r_0^n = \exp(\lambda t + O(\lambda^3t^3/n^2))$. 
Problem 3. Write a code `trapezoid2.m` that implements the trapezoidal method, but which uses the naive iteration scheme rather than the Newton method in order to solve for the fixed-point at each time-step. Apply both your code and the original code `trapezoid.m`, using the Newton method, to the initial-value problem

\[
\begin{align*}
\frac{dy}{dt} &= y \ln |y|, \quad 0 < t < 2 \\
y(0) &= 1/e
\end{align*}
\]

using \( N_s = 100 \) steps. Verify that the two codes give the same result, to the requested tolerance \( TOL = 10^{-15} \) in the root-finding iteration. Also, report the average number of iterations per time-step for the two root-finding algorithms. Finally, compare the results of these two codes with the exact solution \( Y(t) = \exp(-e^t) \). *Hint:* You can easily modify `pece2.m` to produce the necessary code for this problem.

The trapezoid method with Newton root-finding averages 2.75 iterations and the trapezoid method via direct iteration requires 9.96 iterations. Code:

```matlab
1 g=@(t,y) y.*log(abs(y))
2 Dg=@(t,y) log(abs(y))+1
3 e=exp(1);
4 Y=@(t) exp(-exp(t))
5
6 [t,y1]=trapezoid(g,Dg,[0 4],1/e,100,0);
7 [t,y2]=trapezoid2(g,[0 4],1/e,100,0);
8 plot(t,Y(t),'-r',t,y1,-b',t,y2,'-g',' LineWidth ',2)
9 legend('exact', 'trap', 'trap2')
10 title('Solution of $\dot{y} = y \ln |y|, \ y(0)=1/e$ ', 'FontSize ',20, 'interpreter','latex')
11
12 figure
13 plot(t,y1-y2,' LineWidth ',2)
14 title('Difference Between Trap and Trap2', 'FontSize ',15)
15 [t,y2]=trapezoid2(g,[0 4],1/e,100,0);

trapezoid2.m:
function [t,y,it] = trapezoid2(f,tspan,y_0,N_s,yes)
% solve the ODE dy/dt = f(t,y) by the trapezoidal method
% with N_s steps, using iteration to find the fixed point
% at each time-step

tol=1e-15;
itmax=100;
it=0;

10 if0=tspan(1);
11 t_f=tspan(2);
12 D=length(y_0);
13 dt = (t_f - t_0)/N_s;
14 t = t_0:dt:t_f;
```
N = length(t);

j = 1;
y(1,:) = y_0(:)';

while j < N
    yj0 = y(j,:);'
    
    % begin fixed-point iteration with forward Euler
    k = 0;
yjold = yj0;
yj = yj0 + dt*feval(f,t(j),yj0);
    
    % fixed-point iteration for update
    while norm(yj-yjold)>tol*max(abs(yj),1)
        if k+1>itmax
            break
        end
        yjold = yj;
fj0 = feval(f,t(j),yj0);
yj = yj0 + dt*(fj0+feval(f,t(j+1),yj))/2;
        k = k+1;
    end
    it = it + k;

    y(j+1,:) = yj';
    j = j + 1;
end

if yes == 1
    for k = 1:D,
        figure
        z = y(:,k);
        plot(t,z)
        xlabel('time t')
ylabel(sprintf('y_{%d}', k))
    end
end

t = t';
it = it/(N-1);

return
Problem 4. For each of these two initial-value problems with given exact solution

(i) \[ \dot{y} = -y^2 + \cos(t) + (1 - \cos(2t))/2, \quad y(0) = 0; \quad Y(t) = \sin(t) \]
(ii) \[ \dot{y} = -y^3 + \cos(t) + (3\sin(t) - \sin(3t))/4, \quad y(0) = 0; \quad Y(t) = \sin(t) \] (1)

use all three of the methods midpoint, PECE, and trapezoidal, for numbers of steps \( N_s = 25, 50, 75, 100 \). For each of these cases (twenty-four in all!) plot the numerical solution and its exact error. Compare and discuss the performance of the three methods. Do all three appear to converge? Which method performs better at finite time-step \( h \), under what circumstances, and why?

All methods appear to converge. However, trapezoidal and PECE2 perform better than midpoint, only moderately better for problem (i) but substantially better for problem (ii). In the second problem, midpoint method suffers from spurious oscillations due to the parasitic solution, which decrease with \( h \) but which degrade the solution relative to trapezoid and midpoint at finite \( h \). Note that infinitesimal errors decay according to the linearized equations, for problem (i)

\[ \dot{\delta} = -2Y(t)\delta \]

and for problem (ii)

\[ \dot{\delta} = -3Y^2(t)\delta. \]

In the latter case, \( \lambda(t) = Y^2(t) < 0 \) for all times \( t \) and midpoint point method lacks relative stability precisely for \( \lambda < 0 \). This helps to explain its poorer performance for problem (ii). Note that PECE2 performs very similarly as does trapezoidal method, although it is more economical in floating point operations.

Code:

```matlab
f=@(t,y) -y.^2+cos(t)+(1-cos(2*t))/2;
Y=@(t) sin(t);
y0 =0;

for i =1:4
    Ns=25*i;
    [tt,yt]=trapezoid2(f,[0 2*pi],y0,Ns,0);
    figure
    plot(tt,yt,'-b',tt,Y(tt),'-r ')    
    fgnm=[' Trapezoidal Method for Problem (i) with N=', num2str(Ns)];
    title(fgnm)
    figure
    plot(tt,yt-Y(tt),'-b ')    
    fgnm=[' Trapezoidal Error for Problem (i) with N=', num2str(Ns)];
    title(fgnm)

    [tp,yp]=pece2(f,[0 2*pi],y0,Ns,0);
    figure
    plot(tp,yp,'-b',tp,Y(tp),'-r ')    
    fgnm=[' PECE2 Method for Problem (i) with N=', num2str(Ns)];
    title(fgnm)
    figure
    plot(tp,yp-Y(tp),'-b ')    
    fgnm=[' PECE2 Error for Problem (i) with N=', num2str(Ns)];
    title(fgnm)

    [tm,ym]=midpoint(f,[0 2*pi],y0,Ns,0);
    figure
    plot(tm,ym,'-b',tm,Y(tm),'-r ')    
    fgnm=[' Midpoint Method for Problem (i) with N=', num2str(Ns)];
    title(fgnm)
    figure
    plot(tm,ym-Y(tm),'-b ')    
    fgnm=[' Midpoint Error for Problem (i) with N=', num2str(Ns)];
    title(fgnm)
end

f=@(t,y) -y.^3+cos(t)+(3*sin(t)-sin(3*t))/4;
for i =1:4
    Ns=25*i;
    [tt,yt]=trapezoid2(f,[0 2*pi],y0,Ns,0);
```

plot(tt, yt, '-b', tt, Y(tt), '-r')
fgnm = ['Trapezoidal Method for Problem (ii) with N= ', num2str(Ns)];
title(fgnm)
figure
plot(tt, yt-Y(tt), '-b')
fgnm = ['Trapezoidal Error for Problem (ii) with N= ', num2str(Ns)];
title(fgnm)

[tp, yp] = pece2(f, [0 2*pi], y0, Ns, 0);
figure
plot(tp, yp, '-b', tp, Y(tp), '-r')
fgnm = ['PECE2 Method for Problem (ii) with N= ', num2str(Ns)];
title(fgnm)
figure
plot(tp, yp-Y(tp), '-b')
fgnm = ['PECE2 Error for Problem (ii) with N= ', num2str(Ns)];
title(fgnm)

[tm, ym] = midpoint(f, [0 2*pi], y0, Ns, 0);
figure
plot(tm, ym, '-b', tm, Y(tm), '-r')
fgnm = ['Midpoint Method for Problem (ii) with N= ', num2str(Ns)];
title(fgnm)
figure
plot(tm, ym-Y(tm), '-b')
fgnm = ['Midpoint Error for Problem (ii) with N= ', num2str(Ns)];
title(fgnm)
Problem 5*. This problem concerns the 3rd-order PECE method with predictor given by the 2nd-order Adams-Bashforth method
\[ y_{n+1} = y_n + \frac{h}{2} [3f(t_n, y_n) - f(t_{n-1}, y_{n-1})], \]  
and corrector given by the 3rd-order Adams-Moulton method
\[ y_{n+1} = y_n + \frac{h}{12} [5f(t_{n+1}, y_{n+1}) + 8f(t_n, y_n) - f(t_{n-1}, y_{n-1})]. \]  

(a) Using the notion of the local solution \( u_n(t) \) which satisfies
\[ \dot{u}_n = f(t, u_n), \quad u_n(t_n) = y_n, \]
explain how many iterations of the AM equation (2) must be made with the prediction of the AB method (1) as initial guess in order to guarantee that the PECE method is 3rd-order. How many iterations are required so that the leading-order truncation error is the same as in the AM method? Justify your answers. Explain why using the Heun method to calculate the first step \( y_1 \) from \( y_0 \) gives a 3rd-order accurate algorithm.

(b) Write a MATLAB code `pece3.m` to implement the simplest scheme in part (a) which employs a single iteration of the AM corrector and the Heun method to generate the first time step. By carefully storing old function values, you should be able to write a code with a single function evaluation per step. Use your code to solve the scalar problem \( \dot{y} = -y(1) \) over the time interval \( 0 < t < 2 \) and verify numerically that the errors indeed scale \( \propto h^3 \) by taking \( h_i = 2^{-i} \) for \( i = 1, 2, ..., 10 \) and comparing the errors for \( i - 1 \) and \( i \).

(c) Write two more MATLAB codes: first, `pece3opt.m`, which implements the same 3rd-order PECE method as (b) but makes a second iteration of the AM corrector equation (2) and thus requires two function evaluations per step; and, second, `AM3.m`, which implements the 3rd-order Adams-Moulton method fully implicitly with the Newton method for root-finding at each time-step. As in (b), use the Heun method to generate the first time step for these two new codes. For the choice \( h = 1/1024 \) compare the results from `pece3.m`, `pece3opt.m`, and `AM3.m` for the same problem as in (b), by plotting the errors \( \delta(t_n) = |y_n - e^{-t_n}| \) versus \( t_n \) for the three methods. Explain your numerical output in terms of the results of (a). Which of the new methods, `pece3opt.m` or `AM3.m`, is most efficient for this problem and why?

(a) Note \( u_n(t_{n+1}) - y_{n+1} = \frac{5}{2} h^3 u_n^{(3)}(t_n)/12 + O(h^4) \) and \( u_n(t_{n+1}) - y_{n+1} = -h^4 u_n^{(4)}(t_n)/24 + O(h^5) \), where AB, AM are respectively the updates of the Adams-Bashford and Adams-Moulton. Then subtracting the above gives \( y_{n+1}^{\text{AM}} - y_{n+1}^{\text{AB}} = \frac{5}{12} h^3 u^{(3)}(t_n)/12 + O(h^4) \). After one fixed-point iteration:
\[ \frac{y_{n+1}^{\text{AM}} - y_{n+1}^{(1)}}{y_{n+1}^{\text{AM}} - y_{n+1}^{(0)}} \leq \frac{5}{12} h \| f(t_{n+1}, y_{n+1}^{(1)}) - f(t_{n+1}, y_{n+1}^{(0)}) \| \leq \frac{5}{12} K h \| y_{n+1}^{(1)} - y_{n+1}^{(0)} \| \leq \frac{5}{12} K h \| y_{n+1}^{\text{AM}} - y_{n+1}^{\text{AB}} \|, \]

We already computed the order of the right-hand side so we can write \( y_{n+1}^{\text{AM}} - y_{n+1}^{(1)} = 5K h^4 u^{(3)}(t_n)/144 + O(h^5) = O(h^4) \). Then \( u_n(t_{n+1}) - y_{n+1}^{(1)} = u_n(t_{n+1}) - y_{n+1}^{\text{AM}} + y_{n+1}^{\text{AM}} - y_{n+1}^{(1)} = O(h^4) \). In other words, one iteration is sufficient for the method to be 3rd-order. One can also iterate twice:
\[ \frac{y_{n+1}^{(2)} - y_{n+1}^{(1)}}{y_{n+1}^{(2)} - y_{n+1}^{(0)}} \leq \frac{5}{12} h \| f(t_{n+1}, y_{n+1}^{(1)}) - f(t_{n+1}, y_{n+1}^{(0)}) \| \leq \frac{5}{12} K h \| y_{n+1}^{(1)} - y_{n+1}^{(0)} \| = O(h^5), \]
so that \( u_n(t_{n+1}) - y_{n+1}^{(2)} = u_n(t_{n+1}) - y_{n+1}^{\text{AM}} + y_{n+1}^{\text{AM}} - y_{n+1}^{(2)} = -h^4 u_n^{(4)}(t_n)/24 + O(h^5) \). Thus two iterations has the same leading-order truncation error as does Adams-Moulton.

Heun method is adequate to generate \( y_1 \), since it has truncation error \( O(h^3) \). Thus, \( y(t_1) = y(t_0) + hF_{\text{Heun}}(t_0, y(t_0); h, f) + O(h^3) \) and compared with \( y_1 := y_0 + hF_{\text{Heun}}(t_0, y(t_0); h, f) \) for \( y_0 = y(t_0) \) it follows that
\[ y(t_1) - y_1 = O(h^3), \]
so we immediately have 3rd-order accuracy.

(b) We verify 3rd-order accuracy for PECE3:

<table>
<thead>
<tr>
<th>N_s</th>
<th>PECE3</th>
<th>exact</th>
<th>error</th>
<th>log2 error-ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>0.12798</td>
<td>0.13534</td>
<td>0.054336</td>
<td>NaN</td>
</tr>
<tr>
<td>8</td>
<td>0.13408</td>
<td>0.13534</td>
<td>0.0092452</td>
<td>2.5551</td>
</tr>
<tr>
<td>16</td>
<td>0.13518</td>
<td>0.13534</td>
<td>0.0011586</td>
<td>2.9963</td>
</tr>
<tr>
<td>32</td>
<td>0.13532</td>
<td>0.13534</td>
<td>0.00014385</td>
<td>3.0097</td>
</tr>
<tr>
<td>64</td>
<td>0.13533</td>
<td>0.13534</td>
<td>1.78999e-05</td>
<td>3.0067</td>
</tr>
<tr>
<td>128</td>
<td>0.13533</td>
<td>0.13534</td>
<td>2.2315e-06</td>
<td>3.0038</td>
</tr>
<tr>
<td>256</td>
<td>0.13534</td>
<td>0.13534</td>
<td>2.7855e-07</td>
<td>3.002</td>
</tr>
<tr>
<td>512</td>
<td>0.13534</td>
<td>0.13534</td>
<td>3.4794e-08</td>
<td>3.001</td>
</tr>
<tr>
<td>1024</td>
<td>0.13534</td>
<td>0.13534</td>
<td>4.3477e-09</td>
<td>3.0005</td>
</tr>
<tr>
<td>2048</td>
<td>0.13534</td>
<td>0.13534</td>
<td>5.4337e-10</td>
<td>3.0002</td>
</tr>
</tbody>
</table>
Code:

```matlab
f=@(t,y) -y;
Y=@(t) exp(-t);
for i=1:10,
    N_s=2^(i+1);
    [t,y] = pece3(f,[0 2],1,N_s,0);
y1(i,:) = y;
S=feval(Y,t(1+N_s));
Y1(i,:)=S;
E1(i)=norm(s-S)/norm(S);
end
format long
time=t(1+N_s);
disp('')
disp('')
disp(['time=', num2str(time)])
disp(['N_s, PECE3, exact, error, log2 error-ratio'])
disp(['4 ', num2str(y1(1,:)), ' ', num2str(Y1(1,:)), ' ', num2str(E1(1)), ' NaN'])
for i=2:10
    num=num2str(2^(i+1));
    for j=1:6-length(num)
        num=[' ', num];
    end
    yst=num2str(y1(i,:));
    for j=1:9-length(yst)
        yst=[' ', yst];
    end
    disp([num, yst, num2str(Y1(i,:)), ' ', num2str(E1(i)),' ', num2str(log2(E1(i-1)/E1(i)))]
end

pece3.m:

function [t,y] = pece3(f,tspan,y_0,N_s,yes)
    t_0=tspan(1);
t_f=tspan(2);
D=length(y_0);
dt = (t_f - t_0)/N_s;
t = t_0:dt:t_f;
N=length(t);
yj=y_0';
y(1,:) = y_0;
j = 1;
    % first step by Heun method
    k1 = feval(f,t(1),yj);
fj1=k1;
k2 = feval(f,t(2),yj+k1*dt);
yj = yj + dt*(k1+k2)/2;
y(2,:) = yj';
fj0=feval(f,t(1),yj);
j=2;
    while j < N
        % 2nd-order AB predictor
        yj0=yj;
```
yj = yj0 + dt * (3*fj0 - fj1)/2;
fj = feval(f,t(j+1),yj);
% 3rd-order AM corrector
yj = yj0 + dt * (5*fj + 8*fj0 - fj1)/12;
fj1 = fj0;
% optional 2nd evaluation
fj0 = feval(f,t(j+1),yj);
fj0 = fj;
j = j + 1;
end
if yes == 1
    for k = 1:D,
        figure
        z = y(:,k);
        plot(t,z)
        xlabel('time t')
        ylabel(sprintf('y_%d', k))
    end
end

t = t';
return

(c) See output below. While PECE3 and PECE3OPT are both 3rd-order methods, the errors for PECE3OPT are essentially identical with those for AM3. This is consistent with the analysis of part (a), where it was found that the truncation errors of PECE3OPT should be asymptotically identical with those of AM3. Since the implicit Adams-Moulton methods have smaller truncation errors and better stability properties than those of the explicit Adams-Bashforth methods, we see that the optional second iteration can secure those good properties without solving the fixed-point problem to machine accuracy. Indeed, we observe that AM3 and PECE3OPT have somewhat smaller errors at long times than does PECE3.

Code:
```matlab
f=@(t,y) -y;
Y=@(t) exp(-t);
for i = 1:10,
    N_s = 2^(i+1);
    [t, y] = pece3(f, [0 2], 1, N_s, 0);
s = y(1+N_s,:);
y1(1,:) = s;
S = feval(Y, t(1+N_s));
y1(1,:) = S;
```
E1(i)= \frac{\text{norm}(s-S)}{\text{norm}(S)};
end

format long

figure
plot(t,exp(-t),'-r',t,y, '-b','LineWidth',2)
xlabel('t','FontSize',15); ylabel('y','FontSize',15);
legend('exact','pece3')
title('Exact vs. PECE3','FontSize',18)
print('fig1f.pdf','-dpdf')

Ns=2048;
[t,yo] = pece3opt(f,[0 2],1,N_s,0);
J=@(t,y) -1;
[t,y3] = AM3(f,J,[0 2],1,N_s,0);
figure
plot(t, abs(y-Y(t)),'-g',t, abs(yo -Y(t)),'-b',t, abs(y3 -Y(t)),'--r','LineWidth',2)
xlabel('t','FontSize',15); ylabel('\delta ','FontSize',15);
title('Error in the Three Methods','FontSize',18)
legend('pece3','pece3opt','AM3')

pece3opt.m:

function [t,y] = pece3opt(f,tspan,y_,0,N_s,0)
% solve the ODE dy/dt = f(t,y) by 3rd-order PECE method in N_s steps,
% with 2nd-order AB predictor and 3rd-order AM corrector

% initial conditions
\begin{align*}
  t_0 &= tspan(1); \\
  t_f &= tspan(2); \\
  N &= length(y_0); \\
  dt &= (t_f - t_0)/N_s; \\
  t &= t_0:dt:t_f; \\
  y_j &= y_0'; \\
  y(1,:) &= y_0; \\
  j &= 1;
\end{align*}

% first step by Heun method
\begin{align*}
  k1 &= \text{feval}(f,t(1),y_j); \\
  f_{j1} &= k1; \\
  k2 &= \text{feval}(f,t(2),y_j+k1*dt); \\
  y_j &= y_j + dt*(k1+k2)/2; \\
  y(2,:) &= y_j; \\
  f_{j0} &= \text{feval}(f,t(1),y_j); \\
  j &= 2;
\end{align*}

% while j < N
\begin{align*}
  \text{while } j < N \\
  \quad k1 &= \text{feval}(f,t(j+1),y_j); \\
  \quad f_{j1} &= k1; \\
  \quad k2 &= \text{feval}(f,t(j+2),y_j+k1*dt); \\
  \quad y_j &= y_j + dt*(k1+k2)/2; \\
  \quad y(j+1,:) &= y_j; \\
  \quad f_{j0} &= \text{feval}(f,t(j+1),y_j); \\
  \quad j &= j + 1;
\end{align*}

if yes==1
  \text{for } k=1:D,
\end{align*}
function [t,y] = AM3(f,J,tspan,y_0,N_s,yes)
% solve the ODE dy/dt = f(t,y) with Jacobian matrix J(t,y)
% by the third-order AM method with N_s steps

tol=1e-15;
itmax=100;
it=0;
t_0=tspan(1);
t_f=tspan(2);
D=length(y_0);
dt = (t_f - t_0)/N_s;
t = t_0:dt:t_f;
N=length(t);
yj=y_0';
y(1,:) = y_0;
j = 1;

% first step by Heun method
k1 = feval(f,t(1),yj);
fj1=k1;
k2 = feval(f,t(2),yj+k1*dt);
yj = yj + dt*(k1+k2)/2;
y(2,:) = yj';
fj0=feval(f,t(1),yj);
yj0=yj;
j=2;

while j < N

% begin Newton iteration with forward Euler
k=0;
yjold=yj0;
yj = yj0 + dt*fj0;

% Newton iteration for update
while norm(yj-yjold)>tol*max(abs(yj),1)
    if k>itmax
        break
    end
    yjold=yj;
    Fj=yj-yj0-dt*(5*feval(f,t(j+1),yj)+8*fj0-fj1)/12;
    DFj=eye(D)-dt*5*feval(J,t(j+1),yj)/12;
k=k+1;
yj=yj-DFj\Fj;
end
it=it+k;
y(j+1,:) = yj';
j = j + 1;
yj0=yj;
fj1=fj0;
fj0=feval(f,t(j),yj0);
end

AM3.m:
if yes==1
    for k=1:D,
        figure
        z=y(:,k);
        plot(t,z)
        xlabel('time t')
        ylabel(sprintf('y_%d', k))
    end
end

end

t=t';
it=it/(N-1)

return