Final Solutions, 553.481/681, May 6, 2024

Problem 1 *[20 points]. This problem studies the Gauss-Lobatto rules for integration, which modify usual Gaussian quadrature by including interval endpoints among the node points in order to enable composite integration.*

(a) The 4-point Gauss-Lobatto rule on the symmetric interval $[-1,1]$ *is given by*

$$\int_{-1}^{+1} f(x)\,dx \doteq w_1 f(-1) + w_2 f(-\xi_1) + w_2 f(\xi_1) + w_1 f(+1)$$

for specified ξ_1, w_1, w_2*. The rule is exact (*$=0$*) for any odd function by the symmetry of the formula. Determine the constants* ξ_1, w_1, w_2 *by requiring that the formula be exact for the first three even powers,* $f(x) = x^i$ *with* $i = 0, 2, 4$ *and by then solving the resulting algebraic equations.*

(b) Use the linear transformation $x \to a + (i + (x+1)/2)h$ *to transform the interval* $[-1,1]$ *into* $[a+ih, a+(i+1)h]$ *for integers* $i = 0, ..., n-1$ *with* $h = (b-a)/n$ *and construct thereby the composite 4-point Gauss-Lobatto rule to calculate a general definite integral* $I = \int_a^b f(x)\,dx$*. Give the resulting formula and write a Matlab function code* `lobattoc.m` *to implement it in the format* `I=lobattoc(f,a,b,n)`*.*

(c) The code in part (b) requires that function $f(x)$ *be evaluated not only at the points* $x_i = a + ih$*,* $i = 0, ..., n$ *but also at* $x_i^{\pm} = x_i + (1 \pm \xi_1)\frac{h}{2}$*,* $i = 0, ..., n-1$ *and is thus about as expensive as composite Simpson with* $3n$ *subintervals. Compare composite Gauss-Lobatto and composite Simpson for the integral*

$$I = \int_0^{\pi} e^x \sin x\,dx = \frac{1}{2}(e^{\pi} + 1)$$

with $n = 10, 30, 90$ *for Gauss-Lobatto and* $n = 30, 90, 270$ *for Simpson. Which is more accurate for the same computational cost?*

Solution: (a) For even $f(x)$, the condition to be imposed is

$$w_1 f(1) + w_2 f(\xi_1) = \int_0^1 f(x)\,dx.$$

Thus,

$$\begin{array}{lll} f(x) = 1 & \Longrightarrow & w_1 + w_2 = 1 \\ f(x) = x^2 & \Longrightarrow & w_1 + w_2\xi_1^2 = \frac{1}{3} \\ f(x) = x^4 & \Longrightarrow & w_1 + w_2\xi_1^4 = \frac{1}{5} \end{array}$$

whose unique solution is easily found to be

$$w_1 = \frac{1}{6}, \quad w_2 = \frac{5}{6}, \quad \xi_1 = \frac{1}{\sqrt{5}}.$$

(b) The composite Gauss-Lobatto rule has the form

$$\hat{I} = \sum_{i=0}^{n-1} [w_1 f(x_i) + w_2 f(x_i^-) + w_2 f(x_i^+) + w_1 f(x_{i+1})] \frac{h}{2}$$

with the notations $x_i = a + ih$, $i = 0, 1, ..., n$ and $x_i^{\pm} = x_i + \frac{1}{2}(1 \pm \xi_1)h$, $i = 0, 1, ..., n-1$ and a Matlab function which implements it is given here:

```
function I=labbatoc(f,a,b,n)

h=(b-a)/n;
x=a:h:b;
xm=x(1:n)+(1-1/sqrt(5))*h/2;
xp=x(1:n)+(1+1/sqrt(5))*h/2;
fx=f(x); fxm=f(xm); fxp=f(xp);

w1=1/6; w2=5/6;
I=0;
for ii=1:n
I=I+(w1*fx(ii)+w2*fxm(ii)+w2*fxp(ii)+w1*fx(ii+1))*h/2;
end

end
```

(c) To make the comparison with composite Simpson, we run the following script

```
for ii=0:1:2

    n=10*3^ii

    Il=lobattoc(@(x) sin(x),0,pi,n)

    errl=Il/2-1

    n=3*n

    Is=simpc(@(x) sin(x),0,pi,n)

    errs=Is/2-1

end
```

with the following output

Gauss-Lobatto

$$n = 10,\ Il = 12.070346377381927,\ errl = 5.053069074278937 \times 10^{-9}$$
$$n = 30,\ Il = 12.070346316473794,\ errl = 6.972422639250908 \times 10^{-12}$$
$$n = 90,\ Il = 12.070346316389745,\ errl = 9.325873406851315 \times 10^{-15},$$

Simpson

$$n = 30,\ Is = 12.070313975530967,\ errs = -2.679364603075385 \times 10^{-6}$$
$$n = 90,\ Is = 12.070345918041660,\ errs = -3.300219919566416 \times 10^{-8}$$
$$n = 270,\ Is = 12.070346311473024,\ errs = -4.073296144824212 \times 10^{-10}$$

For the same computational cost, composite Gauss-Lobatto has superior accuracy compared with composite Simpson.

Problem 2 *[40 points]. (a) An inconvenient feature of the trapezoidal method as implemented in the course code* `trapezoid.m` *is that it uses the Newton-Raphson method to approximate the solution of the nonlinear equation* $\mathbf{F}(\mathbf{y}_{n+1}) = \mathbf{0}$ *for*

$$\mathbf{F}(\mathbf{y}_{n+1}) = \mathbf{y}_{n+1} - \mathbf{y}_n - \frac{h}{2}[\mathbf{f}(t_n, \mathbf{y}_n) + \mathbf{f}(t_{n+1}, \mathbf{y}_{n+1})],$$

and thus requires the $d \times d$ *Jacobian matrix* $\mathbf{J}(t, \mathbf{y}) = \frac{\partial \mathbf{f}}{\partial \mathbf{y}}(t, \mathbf{y})$. *Write a new code* `trapezoidsteff.m` *which uses instead Steffensen's method, a quasi-Newton method employing the approximate derivative matrix*

$$\frac{\partial F_i}{\partial y_j}(\mathbf{y}_{n+1}) \simeq \frac{F_i(\mathbf{y}_{n+1}^j) - F_i(\mathbf{y}_{n+1}^{j+1})}{F_j(\mathbf{y}_{n+1})}, \quad 1 \le i, j \le d$$

with $\mathbf{y}_{n+1}^j = (y_{n+1,1}, ..., y_{n+1,j-1}, y_{n+1,j}^*, ..., y_{n+1,d}^*)$ *and* $\mathbf{y}_{n+1}^* = \mathbf{y}_{n+1} + \mathbf{F}(\mathbf{y}_{n+1})$. *Show for a scalar function* $F(y_{n+1})$ *that this scheme reduces to Steffensen's method to find the fixed point of* $G(y_{n+1}) = y_{n+1} + F(y_{n+1})$ *and thus converges quadratically. Write your code* `trapezoidsteff.m` *to output also the average number of function evaluations per timestep and take care to minimize that number.*

(b) The course code `pece2.m` *implements the 2nd-order PECE method with Euler method as predictor and trapezoidal method as corrector, and with also the first timestep by Euler's method. Write a new code* `pece2mid.m` *instead with midpoint method as predictor* $\mathbf{y}_{n+1}^{\text{mid}}$ *and trapezoidal method as corrector* $\mathbf{y}_{n+1}$, *and with the first timestep by Heun's method* $\mathbf{y}_1 = \mathbf{y}_1^{\text{heun}}$. *How many iterations of the trapezoidal corrector are required to obtain* $\mathbf{y}_{n+1}$ *with the same leading-order truncation error as the full trapezoidal method? Explain your answer. Explain also why the function value* $\mathbf{f}(t_{n+1}, \mathbf{y}_{n+1}^{\text{mid}})$ *rather than* $\mathbf{f}(t_{n+1}, \mathbf{y}_{n+1})$ *can be saved and reused in the next timestep, without modifying the leading-order truncation error.*

(c) Consider the following initial-value problem with given exact solution:

$$y' = y - y^2, \; y(0) = 1/4; \quad Y(t) = \frac{1}{1 + 3e^{-t}}.$$

Use all three of the methods midpoint, PECE2, and trapezoidal as implemented by the course code `midpoint.m` *and the codes in parts (a) and (b), for numbers of steps* $N_s = 30, 300, 3000$ *over the interval* $0 < t < 5$. *For each of these three stepsizes plot together the error of the three methods and compare and discuss the performance of the three methods. Do all three appear to converge? Rate the methods in terms of efficiency, as quantified by the number of function evaluations per time step. Which methods are most accurate, under what circumstances, and why?*

Hint: For the last question, consider the sign of $\frac{\partial f}{\partial y}(t, Y(t))$.

Solution: (a) Note for the scalar case that the approximate derivative is

$$DF(x_{n+1}) = \frac{F(x_{n+1} + F(x_{n+1})) - F(x_{n+1})}{F(x_{n+1})}$$

so that the iteration becomes

$$x_{n+1}^{j+1} = x_{n+1}^{j} - \frac{F(x_{n+1}^{j})}{DF(x_{n+1}^{j})} = x_{n+1}^{j} - \frac{(F(x_{n+1}^{j}))^2}{F(x_{n+1}^{j} + F(x_{n+1}^{j})) - F(x_{n+1}^{j})}.$$

The general iteration for Steffensen's method is

$$x_{n+1}^{j+1} = x_{n+1}^{j} - \frac{(G(x_{n+1}^{j}) - x_{n+1}^{j})^2}{G(G(x_{n+1}^{j})) + x_{n+1}^{j} - 2G(x_{n+1}^{j})}$$

and at least quadratically convergent. However, if $G(x_{n+1}) = x_{n+1} + F(x_{n+1})$, then

$$G(x_{n+1}) - x_{n+1} = F(x_{n+1})$$

and thus

$$\begin{aligned} G(G(x_{n+1})) + x_{n+1} - 2G(x_{n+1}) &= [G(G(x_{n+1})) - G(x_{n+1})] - [G(x_{n+1}) - x_{n+1}] \\ &= F(G(x_{n+1})) - F(x_{n+1}) \\ &= F(x_{n+1} + F(x_{n+1})) - F(x_{n+1}). \end{aligned}$$

It follows that Steffensen's root-finding method is a special case of the general Steffensen's method for fixed-point iteration.

A code which implements trapezoidal method using Steffensen's method for nonlinear root-finding is the following:

```
function [t,y,mfeval] = trapezoidsteff(f,tspan,y_0,N_s)

% solve the ODE dy/dt = f(t,y) by the trapezoidal method
% with N_s steps, using Steffensen's iteration to find the
% fixed point at each time-step.

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

t_0=tspan(1);
t_f=tspan(2);
D=length(y_0);
Fj=zeros(D,1);
DFj=zeros(D,D);

dt = (t_f - t_0)/N_s;
t = t_0:dt:t_f;
N=length(t);

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

```
mfeval=0;


while j < N

yj0=y(j,:)';
ifeval=0;

fj0=feval(f,t(j),yj0);
ifeval=ifeval+1;

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

% Steffensen iteration for update
while norm(yj-yjold)>tol*max(norm(yj),1)
     if k+1>itmax
       break
     end
Fj=yj-yj0-dt*(fj0+feval(f,t(j+1),yj))/2;
ifeval=ifeval+1;
if norm(Fj)<tol*max(norm(yj),1)
   break
end
Fjold=Fj;
wj=yj+Fj;
yjnew=yj;
     for l=D:-1:1
       yjnew(l)=wj(l);
       Fjnew=yjnew-yj0-dt*(fj0+feval(f,t(j+1),yjnew))/2;
       ifeval=ifeval+1;
       DFj(:,l)=(Fjnew-Fjold)/Fj(l);
       Fjold=Fjnew;
     end
yjold=yj;
k=k+1;
yj=yj-DFj\Fj;
end
mfeval=mfeval+ifeval;


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

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

return
```

(b) The midpoint method $\mathbf{y}_{n+1}^{\text{mid}}$ is 2nd-order accurate with local truncation error

$$\mathbf{T}_n^{\text{mid}}(\mathbf{y}) = \mathbf{u}_n(t_{n+1}) - \mathbf{y}_{n+1}^{\text{mid}} = \frac{1}{3}\mathbf{y}^{(3)}(t_n^*)h^3 + O(h^4) = O(h^3)$$

in terms of the local solution $\mathbf{u}_n(t)$. Similarly, the trapezoidal method $\mathbf{y}_{n+1}^{\text{trap}}$ is also 2nd-order accurate but with local truncation error

$$\mathbf{T}_n^{\text{trap}}(\mathbf{y}) = \mathbf{u}_n(t_{n+1}) - \mathbf{y}_{n+1}^{\text{trap}} = -\frac{1}{12}\mathbf{y}^{(3)}(t_n^*)h^3 + O(h^4).$$

By subtraction, it follows that

$$\mathbf{y}_{n+1}^{\text{trap}} - \mathbf{y}_{n+1}^{\text{mid}} = (\mathbf{y}_{n+1}^{\text{trap}} - \mathbf{u}_n(t_{n+1})) - (\mathbf{y}_{n+1}^{\text{mid}} - \mathbf{u}_n(t_{n+1})) = O(h^3).$$

Since $\|\mathbf{y}_{n+1}^{\text{trap}} - \mathbf{y}_{n+1}^{(j)}\|$ decreases by a factor of $hK/2$ for each iteration of

$$\mathbf{y}_{n+1}^{(j+1)} = \mathbf{y}_n + \frac{h}{2}[\mathbf{f}(t, \mathbf{y}_n) + \mathbf{f}(t, \mathbf{y}_{n+1}^{(j)})]$$

it follows that with $\mathbf{y}_{n+1}^{(0)} = \mathbf{y}_{n+1}^{\text{mid}}$ then

$$\mathbf{y}_{n+1}^{\text{trap}} - \mathbf{y}_{n+1}^{(1)} = O(h^4)$$

and

$$\mathbf{u}_n(t_{n+1}) - \mathbf{y}_{n+1}^{(1)} = (\mathbf{u}_n(t_{n+1}) - \mathbf{y}_{n+1}^{\text{trap}}) + (\mathbf{y}_{n+1}^{\text{trap}} - \mathbf{y}_{n+1}^{(1)}) = -\frac{1}{12}\mathbf{y}^{(3)}(t_n^*)h^3 + O(h^4).$$

Hence, the result $\mathbf{y}_{n+1} := \mathbf{y}_{n+1}^{(1)}$ with one iteration of the trapezoidal corrector for the midpoint predictor has leading-order truncation error $\mathbf{T}_n(\mathbf{y}) = \mathbf{u}_n(t_{n+1}) - \mathbf{y}_{n+1}$ which is the same as for the full trapezoidal approximation, $\mathbf{T}_n^{\text{trap}}(\mathbf{y})$.

Furthermore,

$$\mathbf{y}_{n+1}^{(1)} - \mathbf{y}_{n+1}^{(0)} = (\mathbf{y}_{n+1}^{(1)} - \mathbf{u}_n(t_{n+1})) - (\mathbf{y}_{n+1}^{(0)} - \mathbf{u}_n(t_{n+1})) = O(h^3) + O(h^3) = O(h^3)$$

and thus by the differential mean value theorem

$$\mathbf{f}(t_{n+1}, \mathbf{y}_{n+1}^{(1)}) - \mathbf{f}(t_{n+1}, \mathbf{y}_{n+1}^{(0)}) = \frac{\partial \mathbf{f}}{\partial \mathbf{y}}(t_{n+1}, \mathbf{y})(t_{n+1}, y_{n+1}^*) \cdot (\mathbf{y}_{n+1}^{(1)} - \mathbf{y}_{n+1}^{(0)}) = O(h^3).$$

It follows that in the next time-step the midpoint predictor is

$$\begin{aligned} \mathbf{y}_{n+2}^{\text{mid}} &= \mathbf{y}_n + 2h\mathbf{f}(t_{n+1}, \mathbf{y}_{n+1}) \\ &= \mathbf{y}_n^{(1)} + 2h\mathbf{f}(t_{n+1}, \mathbf{y}_{n+1}^{(1)}) \;=\; \mathbf{y}_n^{(1)} + 2h\mathbf{f}(t_{n+1}, \mathbf{y}_{n+1}^{(0)}) + O(h^4) \end{aligned}$$

and thus the use of $\mathbf{f}(t_{n+1}, \mathbf{y}_{n+1}^{(0)})$ rather than $\mathbf{f}(t_{n+1}, \mathbf{y}_{n+1})$ leads to an error only $O(h^4)$, of smaller order than the truncation error of midpoint.

A code which implements PECE2 with midpoint predictor and trapezoid corrector, making the first step by Heun, is given by the following:

```
function [ t,y ] = pece2mid(f,tspan,y_0,N_s)

% solve the ODE dy/dt = f(t,y) by 2nd-order PECE method in N_s steps,
% with midpoint as predictor and trapezoid rule as corrector.

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
yd(1,:)=feval(f,t(1),yj)';
yj = yj + dt*yd(1,:)';
fj0=feval(f,t(2),yj);
y(2,:) = (y(1,:)+dt*fj0.'+yj')/2;
j=2;

while j < N
yj1=y(j-1,:)';
yj0=y(j,:)';
% midpoint predictor
yj = yj1+ 2*dt*fj0;
% trapezoidal corrector
fj=feval(f,t(j+1),yj);
yj = yj0 + dt*(fj0+fj)/2;
% optional second evaluation
% fj0=feval(f,t(j+1),yj);
fj0=fj;
y(j+1,:) = yj';
j = j + 1;
end

t=t';
return
```

(c) A code which makes the comparison of the three methods is here:

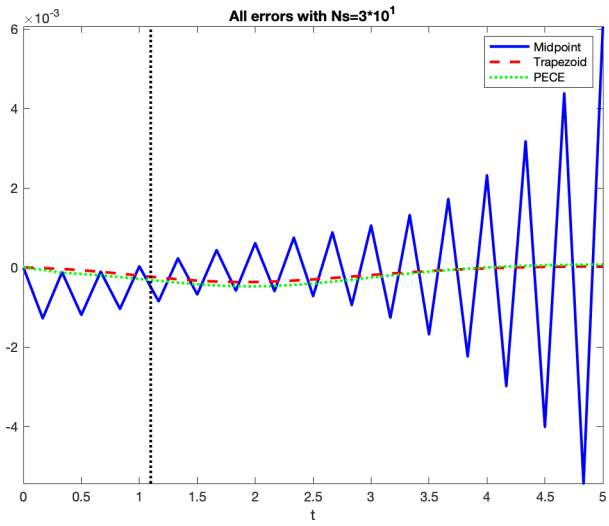
```
f=@(t,y) y-y.^2
Df=@(t,y) 1-2*y
Y= @(t) 1./(1+3*exp(-t))

```

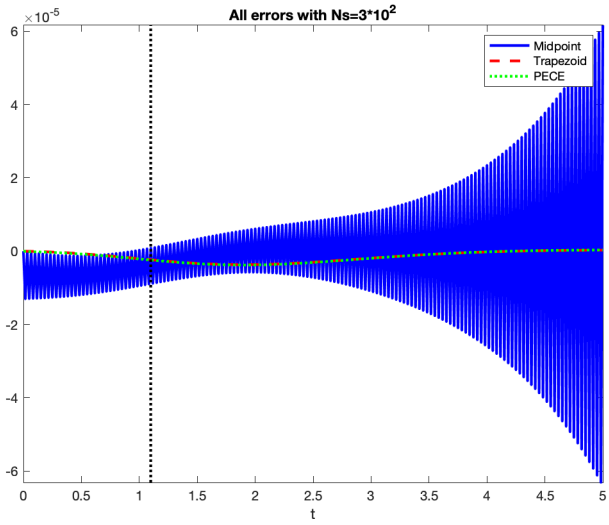
```
5
6  for i=1:3
7
8   Ns=3*10^i;
9
10  [tt,yt,mfeval]=trapezoidsteff(f,[0 5],1/4,Ns,0);
11  mfeval=mfeval
12  [tp,yp]=pece2mid(f,[0 5],1/4,Ns,0);
13  [tm,ym]=midpoint(f,[0 5],1/4,Ns,0);
14
15  figure
16  plot(tm,ym-Y(tm),'-b',tt,yt-Y(tt),'--r',tp,yp-Y(tp),':g','LineWidth',2)
17  low=min(min([ym-Y(tm),yt-Y(tt),yp-Y(tp)]));
18  high=max(max([ym-Y(tm),yt-Y(tt),yp-Y(tp)]));
19  hold on
20  plot(log(3)+0*[low,high],[low,high],':k','LineWidth',2)
21  xlabel('t')
22  axis tight
23  title(sprintf('All errors with Ns=3*10^%d', i))
24  legend('Midpoint','Trapezoid','PECE')
25  fgnm='prob2';
26  for j=1:i
27     fgnm=[fgnm,'i'];
28  end
29  fgnm=[fgnm,'.png'];
30  print(fgnm,'-dpng')
31
32  pause
33
34 end
```

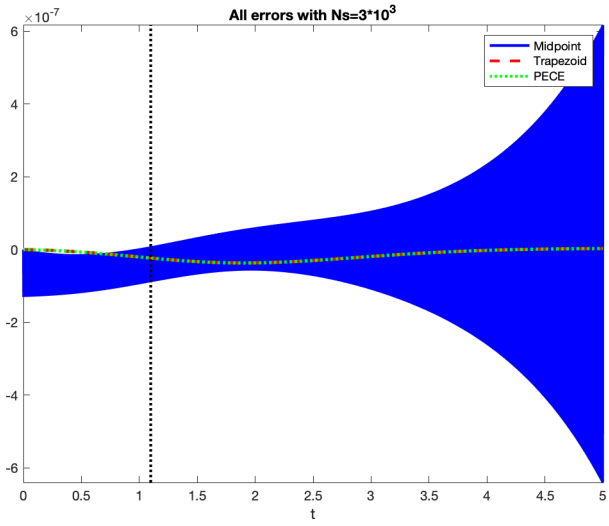
and the output figures are on the following page. By visual inspection, the order of the errors is decreasing in magnitude proportional to h^2 and we can say that all three methods appear to converge quadratically. `midpoint.m` and `pece2mid.m` are both written to make one function evaluation per time-step, but `trapezoidsteff.m` makes a mean number of function evaluations per step which varies with h, as follows:

$$h = 1/6: \ \text{mfeval} = 7.2; \quad h = 1/60: \ \text{mfeval} = 6; \quad h = 1/600: \ \text{mfeval} = 4.$$

The number of function evaluations per step decreases with h, but `trapezoidsteff.m` in all cases is less efficient than the other two methods. In terms of relative error, `trapezoidsteff.m` is most accurate, but `pece2mid.m` has only slightly larger error and in the limit $h \to 0$ the two agree almost exactly. `midpoint.m` is the least accurate, with decreasing error for $0 < t \lesssim 1$ and rapidly increasing error for $1 \lesssim t < 5$. This can be understood by recalling that errors in midpoint grow approximately as for the model problem $\dot{\delta} = \lambda(t)\delta$ with $\lambda(t) = \frac{\partial f}{\partial y}(t, Y(t)) = 1 - 2Y(t)$ and the latter changes sign at $t_* = \ln(3) \doteq 1.0986$ when $Y(t_*) = \frac{1}{2}$. This time is indicated in the plots by the vertical dotted line. For $t > t_*$ where $\lambda(t) < 0$, the midpoint method lacks relative stability and the oscillating parasitic solutions grows.


All errors with Ns=3*10^1
×10^-3
Midpoint
Trapezoid
PECE
t


All errors with Ns=3*10^2
×10^-5
Midpoint
Trapezoid
PECE
t


All errors with Ns=3*10^3
×10^-7
Midpoint
Trapezoid
PECE
t

Problem 3 *[20 points]. For each of the following 3-step methods*

$$
\begin{array}{ll}
(i) & \mathbf{y}_{n+1} = \frac{1}{4}(\mathbf{y}_n + 3\mathbf{y}_{n-2}) + \frac{1}{4}h(9\dot{\mathbf{y}}_n - 2\dot{\mathbf{y}}_{n-1} + 3\dot{\mathbf{y}}_{n-2}) \\
(ii) & \mathbf{y}_{n+1} = \mathbf{y}_{n-2} + \frac{1}{4}h(9\dot{\mathbf{y}}_n + 3\dot{\mathbf{y}}_{n-2}) \\
(iii) & \mathbf{y}_{n+1} = \frac{1}{20}(27\mathbf{y}_n - 7\mathbf{y}_{n-2}) + \frac{h}{10}(6\dot{\mathbf{y}}_{n+1} - 3\dot{\mathbf{y}}_{n-2})
\end{array}
$$

answer the following questions:

(a) Is the method explicit or implicit?

(b) Is the method consistent?

(c) Find the characteristic polynomial $\rho(r) = (1-h\lambda b_{-1})r^{p+1} - \sum_{j=0}^{p}(a_j + h\lambda b_j)r^{p-j}$ *of the method. You do not need to find the* $p+1$ *roots, denoted* $r_0(h\lambda), r_1(h\lambda), ..., r_p(h\lambda)$. *Note for any consistent method there is a root satisfying* $r_0(h\lambda) = 1 + h\lambda + O((h\lambda)^2)$.

(d) Is the method convergent?

To answer this question you may use the following fundamental result: A consistent multistep method is convergent if and only if the root condition is satisfied: $|r_j(0)| \leq 1$ *for all* $j = 0, ..., p$ *and furthermore any root which satisfies* $|r_j(0)| = 1$ *must be simple. Note that the root condition only involves the roots* $r_j(0)$ *for* $h\lambda = 0$, *which you should be able to calculate explicitly.*

(e) If the method is convergent, what is the order of convergence and the leading-order truncation error $\mathbf{T}_n(\mathbf{y})$*?*

(f) If the method is convergent, is it also relatively stable?

To answer (f), you might use the fact that relative stability is implied by the strong root condition: $|r_j(0)| < 1$ *for all* $j \neq 0$. *If you use this condition, you must explain why it implies relative stability. If instead* $|r_j(0)| = 1$ *for some* $j \neq 0$, *then you will need to consider* $r_j(h\lambda)$ *for* $h\lambda \neq 0$. *You may do this numerically, if necessary.*

Solution: It is completely mechanical to determine consistency in (b) and and order of convergence in (e), by checking the corresponding linear conditions

$$d_i := \sum_{j=0}^{p}(-j)^i a_j + i\sum_{j=-1}^{p}(-j)^{i-1}b_j = 1$$

for $i = 0, 1$ in (b) and $i = 2, ..., m$ in (e). Thus, we check those conditions and find the first non-vanishing value $c_{m+1} := 1 - d_{m+1}$ with the following Matlab script

```
p=2;

a=[1,0,3]/4;
b=[9,-2,3]/4;
j=0:p;
```

```
6
7  d=sum(a);
8  i=1;
9  while abs(d-1)<1e-10
10    d=sum((-j).^i.*a+i*(-j).^(i-1).*b);
11    i=i+1;
12 end
13 m=i-2
14 c=1-d
15 [nx,dx]=rat(c,1e-6);
16 frac=[nx,dx]
17 cr=nx./dx;
18 if norm(c-cr)<1e-14
19    c=cr;
20 end
21
22 pause
23
24 a=[0,0,1];
25 b=[9,0,3]/4;
26 j=0:p;
27
28 d=sum(a);
29 i=1;
30 while abs(d-1)<1e-10
31    d=sum((-j).^i.*a+i*(-j).^(i-1).*b);
32    i=i+1;
33 end
34 m=i-2
35 c=1-d
36 [nx,dx]=rat(c,1e-6);
37 frac=[nx,dx]
38 cr=nx./dx;
39 if norm(c-cr)<1e-14
40    c=cr;
41 end
42
43 pause
44
45 a=[27,0,-7]/20;
46 b=[6,0,0,-3]/10;
47 j=0:p;
48 jm=-1:p
49
50 d=sum(a);
51 i=1;
52 while abs(d-1)<1e-10
53    d=sum((-j).^i.*a)+i*sum((-jm).^(i-1).*b);
54    i=i+1;
55 end
56 m=i-2
```

```
c=1-d
[nx,dx]=rat(c,1e-6);
frac=[nx,dx]
cr=nx./dx;
if norm(c-cr)<1e-14
   c=cr;
end
```

We now consider in turn the three given multistep methods:

Method (i): (a) Explicit. (b) Consistent. (c) The characteristic polynomial is

$$r^3 - \frac{1}{4}(1+9h\lambda)r^2 - \frac{1}{2}(h\lambda)r - \frac{3}{4}(1+h\lambda)$$

and for $h\lambda = 0$

$$r^3 - \frac{1}{4}r^2 - \frac{3}{4} = \frac{1}{4}(r-1)(4r^2+3r+3).$$

(d) The three roots for $h\lambda = 0$ are $r_0 = 1,\ r_\pm = \frac{-3\pm i\sqrt{39}}{8}$. Since $|r| \le 1$ for all three, the root condition is satisfied. Because the method is consistent, the root condition is equivalent to convergence. (e) The method is 2nd-order and $c_3 = -\frac{1}{2}$ so that

$$\mathbf{T}_n = \frac{c_3}{3!}\mathbf{y}^{(3)}(t_n)h^3 + O(h^4) = -\frac{1}{12}\mathbf{y}^{(3)}(t_n)h^3 + O(h^4)$$

(f) Since $|r| < 1$ except for the simple root $r = 1$, the strong root condition is satisfied and thus the method is relatively stable.

Because $|r_i| \le 1 - \epsilon < 1 = |r_0|$ for $i > 0$, relative stability follows easily from continuity of the roots $r_i(h\lambda)$ with respect to $h\lambda$. In fact, for sufficiently small $|h\lambda|$,

$$\forall i > 0,\ |r_i(h\lambda)| < 1 - \frac{1}{2}\epsilon < |r_0(h\lambda)|.$$

Method (ii): (a) Explicit. (b) Consistent. (c) The characteristic polynomial is

$$r^3 - \frac{9}{4}h\lambda r^2 - (1 + \frac{3}{4}h\lambda)$$

and for $h\lambda = 0$

$$r^3 - 1.$$

(d) The three roots for $h\lambda = 0$ are the three cube roots of unity $r_0 = 1,\ r_\pm = \frac{-1\pm i\sqrt{3}}{2}$. Since $|r| \le 1$ for all three, the root condition is satisfied. Because the method is consistent, the root condition is equivalent to convergence. (e) The method is 3rd-order and $c_4 = 9$ so that

$$\mathbf{T}_n = \frac{c_4}{4!}\mathbf{y}^{(4)}(t_n)h^4 + O(h^5) = \frac{3}{8}\mathbf{y}^{(4)}(t_n)h^4 + O(h^5)$$

(f) Since $|r| = 1$ for all roots, the strong root condition is not satisfied. We thus calculate $r(h\lambda) = r(0) + r'(0)(h\lambda) + O((h\lambda)^2)$ by differentiating the characteristic polynomial with respect to $h\lambda$ to obtain

$$(3r^2 - \frac{9}{2}(h\lambda)r)r'(h\lambda) - \frac{1}{4}(3 + 9r^2) = 0$$

and thus for $h\lambda = 0$

$$r'(0) = \frac{3 + 9r^2(0)}{12r^2(0)} = \frac{1}{4}(3 + r(0)).$$

Thus, for $r_0 = 1$, one gets $r_0' = 1$ and $r_0(h\lambda) = 1 + h\lambda + O((h\lambda)^2)$, as expected. On the other hand,

$$r_\pm = \frac{1}{2}(-1 \pm i\sqrt{3}) \Longrightarrow r_\pm' = \frac{1}{8}(5 \pm i\sqrt{3}).$$

Since $\mathrm{Re}(r_\pm^* r_\pm') = \frac{-5+3}{16} = -\frac{1}{8} < 0$, it then follows that $|r_0(h\lambda)|$ is increasing for small, real $h\lambda$ whereas $|r_\pm(h\lambda)|$ is decreasing. In that case, taking small $h\lambda < 0$, one gets

$$|r_0(h\lambda)| < 1 < |r_\pm(h\lambda)|,$$

as can be easily verified numerically with the `roots` function in Matlab. Thus, the scheme is not relatively stable.

Method (iii): (a) Implicit. (b) Consistent. (c) The characteristic polynomial is

$$(1 - \frac{3}{5}h\lambda)r^3 - \frac{27}{20}r^2 + \frac{1}{20}(7 + 6h\lambda)$$

and for $h\lambda = 0$

$$r^3 - \frac{27}{20}r^2 + \frac{7}{20} = \frac{1}{20}(r - 1)(20r^2 - 7r - 7).$$

(d) The three roots for $h\lambda = 0$ are $r_0 = 1,\ r_\pm = \frac{7 \pm \sqrt{609}}{40}$. Since $|r| \leq 1$ for all three, the root condition is satisfied. Because the method is consistent, the root condition is equivalent to convergence. (e) The method is 3rd-order and $c_4 = -27/5$ so that

$$\mathbf{T}_n = \frac{c_4}{4!}\mathbf{y}^{(4)}(t_n)h^4 + O(h^5) = -\frac{9}{40}\mathbf{y}^{(4)}(t_n)h^4 + O(h^5)$$

(f) Since $|r| < 1$ except for the simple root $r = 1$, the strong root condition is satisfied and thus the method is relatively stable.