1. Consider the four-step method

\[ y_{n+1} = \frac{32}{27} y_n - \frac{5}{27} y_{n-3} + h \left( \frac{2}{3} \dot{y}_{n+1} - \frac{2}{9} \dot{y}_{n-3} \right), \quad n \geq 3, \]

with \( \dot{y}_n = f(t_n, y_n) \). Show that it is a third-order method, and find the leading term in the truncation error, written in the form \( T_n(y) = \frac{c_4}{4!} y^{(4)}(t_n) h^4 + O(h^5) \).

**Solution:** Let us use the following proposition (page 28 of handwritten Ch5.pdf notes on course website):

**Proposition:** A linear \((p + 1)\)-step method is consistent if and only if

\[ \sum_{j=0}^{p} a_j = 1 \quad \text{and} \quad -\sum_{j=0}^{p} j a_j + \sum_{j=0}^{p} b_j = 1. \]

Furthermore, \( \tau(t, x; h) = O(h^{m}) \) for all solutions \( X(t, x) \in C^{m+1}(D) \) is and only if

\[ \sum_{j=0}^{p} (-j)^k a_j + \sum_{j=0}^{p} (-j)^{k-1} b_j = 1, \quad \forall k = 2, \ldots, m. \]

For the four-step method \( p = 3 \), and from the construction of the method we can see that we have \( a_0 = \frac{32}{27}, a_3 = -\frac{5}{27}, a_1 = a_2 = 0, b_{-1} = \frac{2}{3}, b_3 = -\frac{2}{9}, \) and \( b_0 = b_1 = b_2 = 0 \). It is easily verified by direct substitution that

\[
\begin{align*}
(1) \left( \frac{32}{27} \right) &+ (1)(0) + (1)(0) + (1) \left( -\frac{5}{27} \right) + (0) \left( \frac{2}{3} \right) + (0)(0) + (0)(0) + (0) \left( -\frac{2}{9} \right) = 1 \\
(0) \left( \frac{32}{27} \right) &+ (1)(0) + (2)(0) + (3)(0) \left( -\frac{5}{27} \right) + (2) \left( \frac{2}{3} \right) + (0)(0) + (0)(0) + (0)(0) \left( -\frac{2}{9} \right) = 1 \\
(0) \left( \frac{32}{27} \right) &+ (1)(0) + (4)(0) + (9)(0) \left( -\frac{5}{27} \right) + (2) \left( \frac{2}{3} \right) + (0)(0) + (0)(0) + (0)(0) \left( -\frac{2}{9} \right) = 1 \\
(0) \left( \frac{32}{27} \right) &+ (1)(0) + (8)(0) + (27)(0) \left( -\frac{5}{27} \right) + (3) \left( \frac{2}{3} \right) + (0)(0) + (0)(0) + (0)(0) \left( -\frac{2}{9} \right) = 1
\end{align*}
\]

so that these equalities hold for all \( k = 0, 1, \ldots, m \) with \( m = 3 \). We can then conclude that \( \tau(t, x; h) = O(h^{3}) \) for all solutions \( X(t, x) \in C^{4}(D) \) and the method is consequently of third order.

Next, \( c_4 \) in the leading error term is calculated using the formula

\[ c_k = 1 - \sum_{j=0}^{p} (-j)^k a_j - k \sum_{j=-1}^{p} (-j)^{k-1} b_j. \]

With \( k = 4 \), and substituting into this expression we find

\[ c_4 = 1 - \sum_{j=0}^{p} (-j)^4 a_j - 4 \sum_{j=-1}^{p} (-j)^3 b_j = 1 - (-3)^4 \cdot \frac{5}{27} - 4 \left( (1)^3 \cdot \frac{2}{3} + (-3)^3 \cdot -\frac{2}{9} \right) = -\frac{32}{3} \]
2. For the root \( r_0(h\lambda) \) of the midpoint method, take \( h = t/n \) and show that

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

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

**Solution:** Following the hint, we compute:

\[
\ln(r^n) = n \ln(r_0) = n \ln(h\lambda + \sqrt{1 + (h\lambda)^2}) = n(h\lambda + O((h\lambda)^3))
\]

\[
= nh\lambda + O \left( \frac{(nh\lambda)^3}{n^2} \right) = \lambda t + O \left( \frac{(\lambda t)^3}{n^2} \right)
\]

Then \( r^n_0 = \exp(\ln(r^n_0)) = \exp[\lambda t + O((\lambda t)^3/n^2)] \).
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

\[
\begin{align*}
\frac{dy}{dt} &= -5y, \quad 0 < t < 1 \\
y(0) &= 1
\end{align*}
\]

using \( N_s = 25 \) 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.

Solution: We use the following matlab code for `trapezoid2.m`:

```matlab
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;
% 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);
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
```
Now, use this script and the original trapezoid.m on the given initial value problem through the following code:

```matlab
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
```

```matlab
t=t';
it=it/(N-1);
return
```

For trapezoid.m, we reach the desired tolerance in an average of just two iterations per time-step, whereas for trapezoid2.m, we obtain an average of 13.76 iterations per time-step. Below are the graphs of the obtained solutions as well as the differences between the two methods:
We can see that both methods are practically indistinguishable to the true solution when plotted above in the graph on the left. In the graph of the differences, we can see that the difference between the methods has a magnitude of about $10^{-16}$, under the specified tolerance of $10^{-15}$ for the root-finding.
4. For each of the following two initial-value problems with given exact solution

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

(1)

use all three of the methods midpoint, PECE, and trapezoidal, for numbers of steps \(N_s = 10, 100, 1000\). For each of these cases (eighteen 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?

Solution: We use the following Matlab code to apply the three methods and produce the relevant plots:

```matlab
1 f=@(t,y) -y.^2+cos(t)+(1-cos(2*t))/2;  
2 Y=@(t) sin(t);  
3 y0=0;  
4 for i=1:3  
5    Ns=10^i;  
6    [tt,yt]=trapezoid2(f,[0 2*pi],y0,Ns,0);  
7    figure  
8    plot(tt,yt,'-b',tt,Y(tt),'-r')  
9    fgnm=['Trapezoidal Method for Problem (i) with N=', num2str(Ns)];  
10   title(fgnm)  
11    figure  
12    plot(tt,yt-Y(tt),'-b')  
13    fgnm=['Trapezoidal Error for Problem (i) with N=', num2str(Ns)];  
14    title(fgnm)  
15    [tp,yp]=pece2(f,[0 2*pi],y0,Ns,0);  
16    figure  
17    plot(tp,yp,'-b',tt,Y(tt),'-r')  
18    fgnm=['PECE2 Method for Problem (i) with N=', num2str(Ns)];  
19    title(fgnm)  
20    figure  
21    plot(tp,yp-Y(tp),'-b')  
22    fgnm=['PECE2 Error for Problem (i) with N=', num2str(Ns)];  
23    title(fgnm)  
24    [tm,ym]=midpoint(f,[0 2*pi],y0,Ns,0);  
25    figure  
26    plot(tm,ym,'-b',tt,Y(tt),'-r')  
27    fgnm=['Midpoint Method for Problem (i) with N=', num2str(Ns)];  
28    title(fgnm)  
29    figure  
30    plot(tm,ym-Y(tm),'-b')  
31    fgnm=['Midpoint Error for Problem (i) with N=', num2str(Ns)];  
32    title(fgnm)  
33 end  
34  
35 f=@(t,y) -y.^3+cos(t)+(3*sin(t)-sin(3*t))/4;  
36 for i=1:3  
37    Ns=10^i;  
38    [tt,yt]=trapezoid2(f,[0 2*pi],y0,Ns,0);  
39 end
```
Before we give numerical results, we note an important distinction between the two problems. For problem (i), small errors $\delta y$ are propagated as

\[(i) \quad \dot{\delta y} = -2y(\delta y)\]

and thus sometimes grow and sometimes decay depending upon the sign of $y$, whereas for problem (ii) small errors $\delta y$ are propagated as

\[(ii) \quad \dot{\delta y} = -3y^2(\delta y)\]

and thus always decay. We recall that parasitic solutions of the midpoint method grow under such conditions of decay. We now consider numerical results in the following plots.
For Problem (i) with $N=10$ we obtain:

Clearly trapezoidal method is rather poor, but PECE2 is worse, and midpoint very bad.
for $N=100$:

All methods are much better for larger $N$, with trapezoidal and PECE2 quite similar and midpoint a bit worse.
Again, all methods are much better for larger $N$, with trapezoidal and PECE2 quite similar and midpoint a bit worse.
Problem (ii) for $N=10$

Even for this small $N$, trapezoidal method is rather accurate. PECE2 is a bit worse, but midpoint is terrible, with errors of size 1600! We expect midpoint to perform poorly for this problem, because midpoint’s parasitic solution grows exponentially when small errors should be decaying. The huge error here is presumably due to the parasitic solution.
for $N=100$

All methods are much better for larger $N$, with trapezoidal and PECE2 quite similar and midpoint a bit worse. One can still see the effect of the parasitic solution for midpoint in the growing oscillations in the error at long times.
Again, all methods are much better for larger $N$, with trapezoidal and PECE2 quite similar and midpoint a bit worse. For such a large value of $N$ the effect of the parasitic solution is no longer apparent in midpoint over this time range, although it will reappear if one integrates to much longer times with $h = 2\pi/1000$ fixed.
5. This problem concerns the 4th-order Simpson rule multistep method:

\[ y_{n+1} = y_{n-1} + \frac{h}{3}[f(t_{n-1}, y_{n-1}) + 4f(t_n, y_n) + f(t_{n+1}, y_{n+1})] \]

(a) Find the roots \( r_0(h\lambda), r_1(h\lambda) \) of the characteristic equation for this method.

(b) Use the results in part (a) to show that this method is not relatively stable.

**Solution:**

(a) The characteristic equation is

\[ r^{n+1} = r^{n-1} + \frac{h\lambda}{3}(4r^n + r^{n-1}) \implies r^2 = 1 + \frac{h\lambda}{3}(r^2 + 4r + 1) \]

which implies

\[ r_{\pm} = \frac{\frac{4h\lambda}{3} \pm \sqrt{\left(\frac{4h\lambda}{3}\right)^2 + 4\left(1 - \left(\frac{4h\lambda}{3}\right)^2\right)}}{2\left(1 - \frac{h\lambda}{3}\right)} = \frac{2h\lambda}{3} \pm \frac{\sqrt{1 + (h\lambda)^2}}{1 - \frac{h\lambda}{3}} \]

(b) Using the Taylor expansion \( \sqrt{1 + x} = 1 + \frac{1}{2}x + O(x^2) \) and also the geometric series \( \frac{1}{1-x} = 1 + x + O(x^2) \), we have

\[ r_0(h\lambda) = \frac{\sqrt{1 + (h\lambda)^2} + \frac{2h\lambda}{3}}{1 - \frac{h\lambda}{3}} = \left(1 + \frac{2h\lambda}{3} + O((h\lambda)^2)\right)\left(1 + \frac{h\lambda}{3} + O((h\lambda)^2)\right) = 1 + h\lambda + O((h\lambda)^2) \]

\[ r_1(h\lambda) = \frac{-\sqrt{1 + (h\lambda)^2} + \frac{2h\lambda}{3}}{1 - \frac{h\lambda}{3}} = \left(-1 + \frac{2h\lambda}{3} + O((h\lambda)^2)\right)\left(1 + \frac{h\lambda}{3} + O((h\lambda)^2)\right) = -1 + \frac{h\lambda}{3} + O((h\lambda)^2) \]

For small \( |h\lambda|, \text{ with } h\lambda < 0 \):

\[ r_1(h\lambda) < -1, \quad 0 < r_0(h\lambda) < 1 \implies |r_0(h\lambda)| < 1 < |r_1(h\lambda)|. \]

When \( n \to \infty, |r_1(h\lambda)|^n \gg |r_0(h\lambda)|^n \) which implies the method is not relatively stable.