Do all four of the following problems. Show all your work. Answers without supporting work may receive no credit.

Students may discuss the exam only with the instructor and the teaching assistant. No discussion of the exam contents, directly or indirectly, is permitted among students or with any third parties. Any book or internet resource may be used, as long as the book or the website are cited, along with the material taken from it.

You may use any numerical software available, unless you are specifically instructed in the problem statement to write your own code. All codes that are written by you should be turned in with the exam, either as paper printouts or preferably as a Matlab script sent by e-mail to the instructor. Numerical results without the code that produced them will receive no credit.

I attest that I have completed this exam without unauthorized assistance from any person, materials, or device:

Full name: ________________________________

Signature: ________________________________

(See the Johns Hopkins Handbook Academic Ethics for Undergraduates).
**Problem 1.** Consider the following two 3-step methods

\[ (a) \quad y_{n+1} = -\frac{1}{2}y_n + y_{n-1} + \frac{1}{2}y_{n-2} + \frac{h}{2}[5f(t_n, y_n) + f(t_{n-2}, y_{n-2})] \]

\[ (b) \quad y_{n+1} = \frac{27}{32}y_n + \frac{5}{32}y_{n-2} + \frac{h}{32}[12f(t_{n+1}, y_{n+1}) + 27f(t_n, y_n) + 3f(t_{n-2}, y_{n-2})] \]

For both of these methods, answer the following questions:

(i) Is the method explicit or is it implicit? Is the method consistent?

(ii) Calculate the local truncation error using the general asymptotic expansion

\[ T_n(y) \sim \sum_j \frac{c_j}{j!} y^{(j)}(t_n)h^j \]

for multistep methods. What is the formal order of convergence of the method?

(iii) Does the method satisfy the root condition? Is the method convergent?

(iv) If the answer to (iii) is yes, then does the method also satisfy the strong root condition? What does this imply about the relative stability of the method?

*Hint:* The cubic polynomial \( \rho(r) \) always has one root \( r_0 = 1 \) so that the other two roots \( r_1, r_2 \) can be obtained by polynomial division and the quadratic formula.
Problem 2. The 3-stage Runge-Kutta scheme

\[ y_{n+1} = y_n + h[\gamma_1 k_1 + \gamma_2 k_2 + \gamma_3 k_3] \]

\[ k_1 = f(t_n, y_n), \quad k_2 = f(t_n + \alpha_2 h, y_n + h\beta_{21} k_1) \]

\[ k_3 = f(t_n + \alpha_3 h, y_n + h\beta_{31} k_1 + h\beta_{32} k_2) \]

with coefficients

\[ \alpha_2 = 1/2, \quad \beta_{21} = 1/2 \]

\[ \alpha_3 = 3/4, \quad \beta_{31} = 0, \quad \beta_{32} = 3/4 \]

\[ \gamma_1 = 2/9, \quad \gamma_2 = 1/3, \quad \gamma_3 = 4/9 \]

was studied in Homework #6, Problem 2 and shown to be 3rd-order.

(a) Calculate the characteristic equation of this method and find its root(s).

(b) Is this method weakly stable? Relatively stable? Explain your answers.

(c) In Homework #6 you wrote a Matlab function rk3.m to implement the above 3rd-order Runge-Kutta scheme. Here we shall compare its performance with the 3-step method from Problem 1(a) of this exam:

\[ y_{n+1} = -\frac{1}{2}y_n + y_{n-1} + \frac{1}{2}y_{n-2} + \frac{h}{2}[5f(t_n, y_n) + f(t_{n-2}, y_{n-2})] \]

Write a Matlab function threestep.m to implement this particular 3-step method, with \( y_1, y_2 \) calculated from \( y_0 \) by the Heun method. You should write the code so that only one function evaluation is required per time step (e.g. by appropriately modifying the Matlab function midpoint.m from the course website.)

(d) Apply both codes to the initial-value problem

\[ \dot{y} = y, \quad t \in [0, 1], \quad y(0) = 1 \]

with exact solution \( Y(t) = e^t \).

Use for both methods the same number of time steps \( N = 100 \), or \( h = 0.01 \). For each method, time the calculation using tic and toc in Matlab. Also, plot together in the same graph the errors for both methods versus time.

Which method is fastest? Which method is most accurate? Explain the results.

(e) Repeat part (d) for the initial-value problem

\[ \dot{y} = -y, \quad t \in [0, 1], \quad y(0) = 1 \]

with exact solution \( Y(t) = e^{-t} \).
Problem 3. Implicit multistep schemes like the trapezoidal method require the solution of a root-finding problem at each time-step. The Newton method is an algorithm to solve this root-finding with quadratic rate of convergence, but it requires that both the vector function $f$ and its derivative $J = \frac{\partial f}{\partial y}$ must be provided, as in the course Matlab code `trapezoid.m`. In this problem we explore instead the use of Steffensen’s method as an alternative to the Newton method in implementing the implicit trapezoidal integrator, which does not require $J = \frac{\partial f}{\partial y}$.

(a) If $g$ is a scalar function with fixed point $x_*$ satisfying $g(x_*) = x_*$, recall that Steffensen’s method finds the fixed point by the iteration $x_{n+1} = \bar{g}(x_n)$ with the improved function

$$\bar{g}(x) = x - \frac{(g(x) - x)^2}{g(g(x)) - 2g(x) + x}.$$ 

If the naive iteration $x_{n+1} = g(x_n)$ is linearly convergent, then what is the order of convergence of Steffensen’s method when $g$ is twice continuously differentiable?

If Steffensen’s method is applied to the function $g(x) = x + f(x)$ to find a scalar root $x_*$ satisfying $f(x_*) = 0$, then the iteration can be written in a form very similar to Newton’s method, as

$$x_{n+1} = x_n - \frac{f(x_n)}{Df(x_n)}$$

where $Df(x_n)$ is a suitable approximation to the derivative $f'(x_n)$. Derive an explicit formula for the approximate derivative $Df(x_n)$ in Steffensen’s method.

(b) Write a code `trapezoid3.m` to implement the trapezoidal method using Steffensen’s method as in part (a) to solve the root-finding at each time-step. You can modify the course code `trapezoid.m` for this purpose, but only consider scalar ODE’s \( \dot{y} = f(t,y) \). (In fact, Steffensen’s method can be generalized to any dimension!) Write your function code to output not only the solution $[t,y]$ but also the vector $\text{it}$ whose components $i_n$ are the number of iterations required in Steffensen’s method to find the trapezoidal approximation $y_n$ at time $t_n$.

(c) In Homework #7, Problem 2 you wrote a code `trapezoid2.m` to implement trapezoidal method by the direct iteration scheme to solve the fixed-point problem at each time-step. Here we shall compare the efficiency of these two implementations for the initial-value problem $\dot{y} = y$, $t \in [0,1]$, $y(0) = 1$ with exact solution $Y(t) = e^t$.

You should rewrite `trapezoid2.m` slightly to output also the vector $\text{it}$ of iterations at each time-step. Using $N = 100$ time-steps for both codes, plot the numerical solutions $y_2$, $y_3$ for the two codes together with the exact solution $Y(t)$, and plot also the absolute difference $\text{abs}(y_2 - y_3)$ versus time $t$. Do the two implementations of trapezoidal method agree to double precision? How many fixed-point iterations were required at each time-step for the two codes? Which implementation required fewer evaluations of function $f(t,y)$ per time-step? Explain.
Problem 4. Consider the initial-value problem

\[ \dot{y} = -e^t (y - \sin t) + \cos t, \quad y(0) = 0 \]

for the time interval \( 0 < t < 2\pi \). The exact solution is \( Y(t) = \sin t \).

(a) Using the Jacobian derivative \( J(t, y) \), explain how the stiffness of this problem changes over time.

(b) First consider solving this problem with Heun’s method for time-step \( h = 2\pi/100 \). At what time \( t^* \) is this \( h \) too large to guarantee absolute stability of Heun’s method? *Hint:* Use the result of Homework #6, Problem 1 for Heun method.

Try to solve the given initial-value problem numerically with Heun’s method for time-step \( h = 2\pi/100 \), using the course code `heun.m`. Plot the numerical solution together with the exact solution and explain your observations.

(c) Now consider solving this problem with the trapezoidal method for time-step \( h = 2\pi/100 \), using the direct iteration

\[ y_{n+1}^{(j+1)} = y_n + \frac{h}{2} [f(t_n, y_n) + f(t_{n+1}, y_{n+1}^{(j)})], \quad j = 0, 1, 2, \ldots \]

to solve the root-finding at each time-step. At what time \( t^* \) is this \( h \) too large to guarantee that the direct iteration is a contraction mapping?

Try to solve the given initial-value problem numerically with trapezoidal method for time-step \( h = 2\pi/100 \), using the code `trapezoid2.m` from Problem 3. Plot the numerical solution together with the exact solution and also plot versus time \( t_n \) the number of fixed-point iterations \( i_n \) required to obtain \( y_n \). Explain your observations.

(d) What is the domain of absolute stability of the trapezoidal method? Is the trapezoidal method \( A \)-stable?

Try to solve the given initial-value problem numerically with trapezoidal method for time-step \( h = 2\pi/100 \), using the code `trapezoid3.m` from Problem 3. Plot the numerical solution together with the exact solution and also plot versus time \( t_n \) the number of fixed-point iterations \( i_n \) required to obtain \( y_n \). Explain your observations.