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. IEEE quadruple-precision floating-point numbers are 128-bit approx-
imations, which in standard layout have the first bit for the sign, the next \( r = 15 \)
bits for the stored exponent \( SE \), and the final \( p = 112 \) bits for the fraction \( F \).
The stored exponent is the true exponent \( E \) plus the bias, \( SE = E + B \), where
\( B = 2^{14} - 1 = 16,383 \). In hexadecimal format, IEEE quadruple-precision numbers
are represented by strings of 32 hexadecimal digits, with the first 4 digits for the sign
and stored exponent, and the remaining 28 digits for the fraction.

Consider the following four IEEE quadruple precision numbers in hexadecimal format:

\[
\begin{align*}
(i) & \quad 4000921fb5442d18469898cc51701b8 \\
(ii) & \quad c0005bf0a8b1457695355fb8ac404e7a \\
(iii) & \quad 4f9f921fb5442d18469898cc51701b8 \\
(iv) & \quad 305f5bf0a8b1457695355fb8ac404e7a
\end{align*}
\]

(a) Find the decimal representations of these numbers in scientific notation to 16
significant figures in Matlab. Notice that it is not enough to simply calculate the
exponent \( E \) and fraction \( F \) and then to calculate \( x = \pm (1 + F) \cdot 2^E \). The problem is
that the exponent \( E \) may have magnitude too large for \( 2^E \) to be represented in IEEE
double precision. Thus, to make the conversion in Matlab, you will need to find an
integer \( E_{10} \) so that \( 2^E = \gamma \times 10^{E_{10}} \) with a correction factor \( 0.1 < \gamma < 10 \).

(b) For each of the four quadruple-precision numbers in hexadecimal format, find the
equivalent IEEE double precision number obtained by binary rounding, in standard
layout and in hexadecimal format. Round the fraction \( F \) to double precision directly
in the hexadecimal representation to 13 digits, using the round-to-even rule. To
find the stored exponent \( SE \) in double precision, you will need to take into account
the different biases for quadruple and double precision. If the exponent \( E \) is too
large, positive or negative, to be represented in IEEE double precision, then give the
hexadecimal representation for INF or 0, respectively.
Problem 2. Consider the following linear system

\[ \begin{align*}
ax + (2 - a)y &= 4 \\
(2 - a)x + ay &= 0
\end{align*} \]

with real \( a \) as input parameter and \( x, y \) as outputs.

(a) For what values of \( a \) is the above problem well-posed? Justify your answer. \textit{Hint}: Find the explicit solutions \( x(a), y(a) \), when those exist.

(b) What is the relative condition number \( K_x(a) \) for calculating \( x(a) \)? For what values of \( a \) is \( x(a) \) ill-conditioned?

(c) What is the relative condition number \( K_y(a) \) for calculating \( y(a) \)? For what values of \( a \) is \( y(a) \) ill-conditioned? Why does the answer differ from that in (b)?

(d) Consider the quantity \( z(a) = x(a) + y(a) \). What is the condition number \( K_z(a) \) for calculating \( z(a) \)? For what \( a \) is \( z(a) \) ill-conditioned?

(e) One algorithm to approximate \( z(a) \) to double precision is obtained by setting

\[ \hat{z}(a) = \hat{x}(a) + \hat{y}(a), \]

where \( \hat{x}(a), \hat{y}(a) \) are values calculated in double precision. Try this in MATLAB by setting

\[ a = 0.999999999999, \]

calculating first \( x \), then \( y \), and adding the results. Record the result you obtain, the relative error compared with the exact result, and the number of significant figures.

(f) What is the source of the error for the approximation in (e)? Explain why both \( \hat{x}(a) \) and \( \hat{y}(a) \) can be accurate to double precision but not \( \hat{z}(a) \).
Problem 3. A modification of Newton’s method which avoids the calculation of \( f'(x) \) is the iteration

\[
x_{n+1} = x_n - \frac{f(x_n)}{D(x_n)}, \quad D(x_n) = \frac{f(x_n + f(x_n)) - f(x_n)}{f(x_n)}.
\]

Unlike Newton’s method which evaluates both \( f \) and \( f' \) once in each iteration, this modified method evaluates \( f \) twice in each iteration.

(a) Assuming \( f'(x_*) \neq 0 \), show that (*) is a second-order method just like Newton. Hint: Write the iteration as \( x_{n+1} = g(x_n) \). Use \( f(x) = (x - x_*)h(x) \) with \( h(x_*) \neq 0 \), and then compute the formula for \( g(x) \) in terms of \( h(x) \). Having done so, apply the general theorem on higher-order fixed-point iterations.

(b) Let \( \tau_f \) denote the typical time to compute \( f \) and \( \tau_{f'} \) the time to compute \( f' \). Calculate the total time \( T_{\text{mod}} \) for this modified Newton method to converge to given tolerance \( \varepsilon \) starting at initial distance \( |x_0 - x_*| \), in terms of \( \tau_f, \tau_{f'} \), and the constant \( K = \log(M\varepsilon)/\log(M|x_0 - x_*|) \). Let \( T_{\text{new}} \) and \( T_{\text{sec}} \) be the corresponding times for Newton and secant method, resp. When is \( T_{\text{mod}} < T_{\text{new}} \)? When is \( T_{\text{mod}} < T_{\text{sec}} \)?

(c) Write a Matlab script to implement the algorithm (*). You can modify the Matlab script \texttt{newton.m} to do so. Make sure that your code is written so that \( f \) is evaluated exactly twice in each iteration and try also to minimize the number of multiplications and divisions in each iteration step.

(d) Use (*), Newton method, and secant method to find the root of the function

\[
f(x) = \exp(\sin(x)) - x
\]

near \( x = 2 \), to tolerance \( \varepsilon = 10^{-15} \). For the modified and original Newton methods take \( x_0 = 2 \) and for the secant method take \( x_0 = 2, x_1 = 2.5 \). How many iterations are required for each method? Try to time carefully each of the methods and estimate the ratios \( T_{\text{mod}}/T_{\text{sec}} \) and \( T_{\text{new}}/T_{\text{mod}} \). Do these results agree with predictions from (b)?
Problem 4. Initial-value problems for ordinary differential equations, of the form
\[ \frac{dx}{dt} = k(x), \quad x(0) = x_0 \]
shall later in the course be solved numerically by the implicit Euler method:
\[ x_{k+1} = x_k + hk(x_{k+1}) \]
for some small time-step \( h \), which yields an approximation \( x_k \approx x(t_k) \) with \( t_k = kh \) for \( k = 0, 1, 2, \ldots \). In this method, the equation (*) defines a root-finding problem which must be solved for \( x_{k+1} \) at each time-step.

(a) Considering as an example the first step \( k = 0 \), the problem (*) may be formulated as a fixed-point iteration \( x^{(n+1)} = g(x^{(n)}) \) with function
\[ g(x) = x_0 + hk(x) \]
and \( x_1 = x^{(*)} \) solving \( x_1 = g(x_1) \). The iteration is started with \( x^{(0)} = x_0 \) as the initial guess. In terms of the constant \( K' = \max_x |k'(x)| \), give a sufficient condition on time-step \( h \) for the iteration to converge. What happens to \( h \) when \( K' \) becomes large?

When the iteration converges, what is the asymptotic linear rate \( \lambda \) of convergence?

(b) Alternatively, equation (*) for \( k = 0 \) defines a root-finding problem with
\[ f(x) = x - x_0 - hk(x), \]
which may be solved for \( f(x_1) = 0 \) by Newton’s method
\[ x^{(n+1)} = x^{(n)} - \frac{f(x^{(n)})}{f'(x^{(n)})} \]
with \( x^{(0)} = x_0 \) as initial guess and \( x^{(*)} = x_1 \). The proof of convergence of Newton’s method shows that one must have, at least,
\[ |x_0 - x_1| < 1/M, \quad M = \frac{\max_x |f''(x)|}{2 \min_x |f'(x)|}. \]
How does \( M \) depend upon \( h \) as \( h \to 0 \)? Is the above inequality easier or harder to satisfy as \( h \) becomes small? When Newton’s method converges, does the convergence rate depend upon the size of \( h \)?

(c) For the specific choice
\[ k(x) = x \sin(x) + \cos(x), \quad x_0 = 5, \quad h = 0.05 \]
apply both the direct iteration of (a) and the Newton method of (b) to solve for \( x_1 \) to tolerance TOL = 10^{-15}. Do the methods converge and, if so, how many iterations are required? Repeat for the case \( x_0 = 50 \). Do your observations agree with the theoretical conclusions in (a),(b)? In particular, calculate the asymptotic linear convergence rate for the given \( k \) and \( x_1 \) using the formula in (a).