1. In this first problem we’ll use Halley’s method to compute the Pythagorean sum, \( c = \sqrt{a^2 + b^2} \), without using any square roots.

   (a) Note that \( x = c \) is a root of \( f(x) = x^2 - c^2 \). Show that Halley’s method applied to this equation gives the iteration:

   \[ x_{k+1} = x_k \left( 1 + 2 \frac{c^2 - x_k^2}{c^2 + 3x_k^2} \right). \]

   (b) Now we need to think about an initial guess; that is, we need to pick \( x_0 \) in such a way that the iteration is sure to converge to \( c \). Show that if \( 0 \leq x_0 \leq c \), then \( x_k \leq x_{k+1} \leq c \) for all \( k \). Hints: First establish the following relations:

   \[ x_{k+1} - x_k = 2x_k \left( \frac{c^2 - x_k^2}{c^2 + 3x_k^2} \right) \]

   \[ c - x_{k+1} = \left( \frac{(c - x_k)^3}{c^2 + 3x_k^2} \right) \]

   Then use these to show that \( x_0 \leq x_1 \leq c \), \( x_1 \leq x_2 \leq c \), \( \cdots \).

   (c) In MATLAB, implement a function:

   ```matlab
   function [c, k] = hypotenuse(a, b)
   %
   % [c, k] = hypotenuse(a, b);
   %
   % Given a and b, this computes c = sqrt(a^2 + b^2)
   %
   % Input: scalars a and b
   %
   % Output: c = sqrt(a^2 + b^2)
   %
   % k = number of iterations needed for convergence
   %
   that uses the iteration in part (a) to compute \( c = \sqrt{a^2 + b^2} \).
   
   • In your code, you need to choose an initial guess for \( x_0 \). This should depend on your input \( a \) and \( b \).
   • For a stopping tolerance, use:
     
     \[
     \text{while abs(x2 - c2) > sqrt(eps)*max(abs(a), abs(b))}
     \]
     
     where \( x2 = x_k^2 \) and \( p2 = c^2 \).
2. The Chandrasekhar \( H \)-equation, which is defined as

\[
f(H(\mu)) = H(\mu) - \left( 1 - \frac{c}{2} \int_0^1 \frac{\mu H(\nu)}{\mu + \nu} \, d\nu \right)^{-1} = 0 \tag{1}
\]

is used to solve exit distribution problems in radiative transfer. In this problem we consider approaches to compute approximations of \( H(\mu) \) from equation (1).

(a) We begin by constructing a discrete approximation of this equation.
- First approximate the integral with the composite midpoint rule.\(^1\)

\[
\int_0^1 f(\nu) \, d\nu = \frac{1}{N} \sum_{j=1}^{N} f(\nu_j)
\]

where \( \nu_j = (j - 1/2)/N \) for \( 1 \leq j \leq N \).
- Discretize the variable \( \mu \) in a similar way; \( \mu_i = (i - 1/2)/N \) for \( 1 \leq i \leq N \).
- Show that the above discretization steps lead to the following discrete nonlinear system:

\[
f(x) = 0
\]

where

\[
f(x) = \begin{bmatrix} f_1(x) \\ f_2(x) \\ \vdots \\ f_N(x) \end{bmatrix}, \quad \text{with} \quad x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} \quad \text{and} \quad f_i(x) = x_i - \left( 1 - \frac{c}{2N} \sum_{j=1}^{N} \frac{\mu_i \nu_j}{\mu_i + \nu_j} \right)^{-1}\]

(b) Use MATLAB’s \texttt{fsolve} function to solve \( f(x) = 0 \). Use the simple call:

\[
x = \texttt{fsolve}(\texttt{fun}, x0)
\]

For initial \( x0 \) use a vector of all 0s or a vector of all 1s, \( N = 100 \), and \( c = 0.9 \). (It might be interesting to see if \texttt{fsolve} behaves differently for the two suggested initial guesses.)

(c) Now use \texttt{fsolve}, but construct the Jacobian. Read the \texttt{doc} pages for \texttt{fsolve} to see how to do this. Compare the performance of the two different approaches; that is, not using the explicit Jacobian as in part (b), and using the Jacobian. To compare performance, use data in \texttt{output} from \texttt{fsolve}.

(d) Repeat parts (b) and (c) with \( c = 0.9999 \).

---

\(^1\)We will discuss the midpoint rule, and other numerical integration techniques later in the semester.