Lab 1

This Lab session is mainly intended to guide you on the possible impact of numerical errors resulting from floating point arithmetic. We'll see ways to (possibly) overcome these issues, and how to optimize your Matlab code.

1. Consider the following expression:

$$f = \frac{x - \sin x}{x^3}, \quad f(0) \approx 0.1\overline{6}$$

   (a) Write a Matlab function, `OriginalExpression(x)`, that gets a scalar x and returns the value of the original expression provided above.
   Note: you can hard-code the expression, or use *function handles* (`f = @(x) expr(x)`);

   (b) Is there any source of error in the original formula? If yes, discuss a possible way to avoid it by reformulating the above expression (*Hint:* Taylor expansion of $\sin x$ may help).
   Write a Matlab function, `ModifiedExpression(x)`, that implements these modifications.

   (c) Create a vector `x` containing at least 100 logarithmically spaced points ranging between 10^{-10} and 10^{-1} (use `logspace` function). Compute the expression with both functions. Use `for/while` loops to iterate over all the values of `x` and store the results in two separate vectors.

   (d) Plot the result with respect to x on the same figure (use `hold on`) and add a `legend`. How does this plot look? Try to replace the `plot` function with `semilogx` and plot the result in a new figure. Why is this strategy better to visualize the data?

   (e) **BONUS**: Modify the function `OriginalExpression(x)` to use Matlab vectorization. i.e. the function should accept a vector `x` and return the vector of computed values. Increase the vector length and compare the computing time using the original and vectorized form of the algorithm. Timing can be done with the `tic` and `toc` commands.

2. **Euclidean Length of a Vector:**

In the following we want to compute the euclidean norm of a vector $v = \begin{bmatrix} a \\ b \end{bmatrix}$, i.e.:

$$p = \sqrt{a^2 + b^2}$$

(a) Create a Matlab function `OriginalNorm(a,b)` which computes the euclidean norm in its original form provided above;

(b) Write a modified version of the previous function, `ScaledNorm(a,b)` which computes $c = \max(|a|, |b|)$, scales a and b with c, and then computes the euclidean norm;

(c) **BONUS** Implement the Morel-Morrison iterative algorithm to compute the euclidean norm (provided below); the function `MorelMorrison(a,b,N)` should accept as an additional input the number of iterations N (it should converge for $N \geq 3$).

(d) Test the original and modified algorithms for `a = b = realmax` and comment on your results (**Note** `realmax` is the largest finite floating point number).

(e) **BONUS**: Modify the function/s above to use Matlab vectorization. i.e. the functions should accept vectors `a` and `b` and return a vector with the computed euclidean norms. Increase the vector length and compare the computing time using the original and vectorized form of the algorithm. Timing can be done with the `tic` and `toc` commands.

E. Newman, F. Brarda
Department of Mathematics, Emory University

Moler-Morrison
Pythagorean Sum Algorithm

Input: scalars a and b

Output: $p = \sqrt{a^2 + b^2}$

$p := \max(|a|, |b|)$

$q := \min(|a|, |b|)$
for $i = 1$ to N
    $r := (q/p)^2$
    $s := r/(4 + r)$
    $p := p + 2sp$
    $q := sq$
next i