Matlab Assignment 1

Due Date: November 4, 2011

Important Instructions

- Students are allowed to work in groups of 2 **ONLY**.
- ONLY ONE student from each group should submit the assignment.
- Assignments should be uploaded on Moodle.
- The name of the zipped folder to be uploaded should consist of the **IDs** of the group members.

Example: 200600000-200600001.zip

- The uploaded zipped folder should include one folder for each exercise, named Ex1, Ex2,...containing Matlab files and word documents for test cases.
- Extra points would be added for checking special cases in the exercises: (ex: validity of the input)
- You are **not allowed** to use non trivial built-in functions (i.e. functions that do the job for you)!
- Your Assignment will be graded **ZERO**:
  - In all cases of PLAGIARISM (BOTH PROVIDERS AND CHEATERS).
  - In case your MATLAB Quiz grade is less then half of the Assignment grade.
- If you need assistance pass by MATLAB-CLINIC during the GA's office hours (or by appointment)

Exercise 1 : Conversion Methods

1. Write a MATLAB function: **function [E8 , F8] = Convert2to8(E2, F2)**
   which takes as input two binary vectors E2 and F2 that are respectively the integral and fractional parts of a positive binary number b, converts them to octals and outputs the results as 2 vectors E8 and F8 that are respectively the integral and fractional parts of a positive octal number o.

2. Write a MATLAB function: **function [E10 F10] = Convert8to10(E8, F8)**
   which takes as input two octal vectors E8 and F8 that represent respectively the integral and fractional parts of a positive octal number o, converts them to base 10 and outputs the results as 2 decimal numbers, E10 and F10 that represent respectively the integral and fractional parts of the positive decimal number d using Nested Polynomial Evaluation. At the end, this function should also display d as a decimal number.

3. Test each one of the 2 functions above for 3 different test cases and save the results in a word document.(consider different lengths for all input vectors).

Exercise 2 : Successors and Rounding Procedures

Let $x = +mx \times 10^{ex}$ be a positive decimal number in $F(10, p, -20, +20)$, written in normalized floating point form, with $-20 \leq ex < +20$, and $p < 15$.

1. Write a MATLAB function : **function [my, ey] = GetSuccessor(mx, ex, p)** which takes as inputs:

   - mx : the mantissa of x in standard normalised floating point notation
   - ex : the exponent of x
   - p : the precision of the floating point system to which x belongs

   Let y be the successor of x in $F(10, p, -20, +20)$. This function should output:

   - my : the mantissa of y displayed with a precision p (the non significant digits of the fractional part need not be displayed)
     HINT : first compute my, then use **num2str(my,p)** to output my in the required format
   - ey : the exponent of y

2. Let $m = +m_1.m_2m_3...m_p$ be a positive decimal number whose integral part is m_1, and whose fractional part is $0.m_2m_3...m_p$.

   Write a MATLAB function : **function [m] = ConvertVectortoDecimal(M)** which takes as input a vector M of length p whose i^{th} component is the decimal digit m_i, for $i = 1, ..., p$, and whose output is the decimal number m represented by M.
   Use **" format long g"** to display m in double precision, discarding the non significant zeros of the fractional part .

3. Write a MATLAB function : **function [mz, ez] = Round(Mx, ex, n, t)** which takes as inputs:

   - Mx : a vector of length p whose components represent the mantissa mx of the decimal number $x \in F(10, p, -20, +20)$
   - ex : the exponent of x

- n : a positive integer less then or equal to p ($n \leq p$) , representing the precision required to reach
- t : a parameter taking the values 1 or 2

This function should compute z: the representative of x in $F(10, n, -20, +20)$ by rounding x to the closest if $t = 1$ or by chopping if $t = 2$, and output

- mz : the mantissa of z displayed with a precision n.
  HINT : first compute mz, then use **num2str(my,n)** to output mz in the required format (the non significant zeros of the fractional part will be discarded)
- ez: the exponent of z

At the end your function should also **display** z in normalized floating point representation in $F(10, n, -20, +20)$.

4. Test each one of the 2 functions above for 3 different test cases and save the results in a word document.

Remark: Call for previous functions when needed.

Exercise 3 : Root finding methods

The aim of this exercise is to approximate π by computing the root of $f(x) = sin(x)$ in the interval $(3, 4)$, based on Newton's method. For that purpose:

1. Write 2 MATLAB functions

   function[sinx]= mysin(x, p)

   function[cosx]= mycos(x, p)

   that both input:

   - a variable x representing some angle in **radians**
   - a precision p

   Using Taylor's series expansion, these functions should compute respectively the *sine* and *cosine* of x, up to some tolerance $Tol = 0.5 * 10^{(-p+1)}$, and output respectively:

   - the values of $sinx$ or $cosx$ in $F(10, p, -20, +20)$
     HINT : first compute $sinx$ and $cosx$, then use **num2str(. , p)**

   Note that:
   $sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} \dots$
   $cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots$

   N.B. Do not use the MATLAB built in function for the factorials.

   Test each of the functions above for $x = \pi/3, \pi/4$ and $\pi/6$ with $p = 14$ and save your results in a word document.

2. Write a MATLAB **function [root,k]= myNewton(f, df, a, b, p, kmax)** that takes as inputs:

- a function f and its derivative df (as function handles)
- 2 real numbers a and b, where (a, b) is the interval locating the root of f
- a precision p
- a maximum number of iterations $kmax$

Based on Newton's method, this function should output:

- root : the approximation to the root of f up to p decimal figures
  HINT : first compute *root*, then use **num2str(root , p)**
- k : the number of iterations used to reach the required precision p

$Tol = 0.5 * 10^{(-p+1)}$ is the relative error bound to the computed root

Test your function for 2 different functions f and save your results in a word document.

3. Write a MATLAB **function [mypi, errpi, k]= mypiNewton(p, kmax)** that takes as inputs p and $kmax$ as defined in the previous question.
   Applying **Newton's method** on the interval $(3, 4)$ and using the functions **mysin** and **mycos** programmed in part 1, this function should output:

   - $mypi$: the approximation to π up to p decimal figures
   - $errpi$: the absolute error $|mypi - \pi|$ where π is considered in double precision
   - k: the number of iterations used in Newton's method to reach the precision p

   HINT: Note that after calling the functions **myNewton, mysin and mycos** , their outputs should be converted back to numbers using the command str2num(.)

   Test this function for $kmax = 20$ and successively for $p = 2, 3, 7, 10, 15$. Save your numerical results in a word document.