Computational methods and applications (AMS 147)

Homework 1 - Due Wednesday, Feb 25, 11:59 pm

Please submit to CANVAS a .zip file that includes the following Matlab functions:

`compute_factorial.m`
`compute_Euclidean_norm.m`
`matrix_times_vector.m`
`pi_series.m`

Exercise 1 The factorial of a natural number is defined as

$$n! = n(n-1)(n-2)\cdots 1, \qquad 0! = 1. \tag{1}$$

Write a Matlab function `compute_factorial.m` that takes an integer number as input and returns (1). The function should be in the following form

```
function [b] = compute_factorial(n)
```

Input:
`n`: natural number (possibly including 0)

Output:
`b`: factorial of `n`

Exercise 2 The Euclidean norm of an n-dimensional vector is defined as

$$\|\boldsymbol{x}\| = \sqrt{\sum_{k=1}^{n} x_k^2}. \tag{2}$$

Write a Matlab function `compute_Euclidean_norm.m` that computes the norm (2), for an arbitrary input vector $\boldsymbol{x}$. The function should be in the following form

```
function [z] = compute_Euclidean_norm(x)
```

Input:
`x`: n-dimensional vector (either column or row vector)

Output:
`z`: norm of the vector

Hint: You can use the `for` loop. The number of components of the input vector can be determined by using the Matlab command `length(x)` (see the Matlab documentation). You can compare the output of your function with the Matlab command `norm(x)`.

Exercise 3 Write a Matlab program `matrix_times_vector.m` that computes the product between an n-dimensional square matrix $\boldsymbol{A}$ and an n-dimensional (column) vector $\boldsymbol{x}$. The components of the (column) vector $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$ are defined

$$y_i = \sum_{j=1}^{n} A_{ij} x_j \qquad i = 1, ..., n. \tag{3}$$

The Matlab function should be in the following form

```
function [y] = matrix_times_vector(A,x)
```

Input:
`A`: $n \times n$ matrix
`x`: $n \times 1$ vector

Output:
`y`: $n \times 1$ vector

You are not allowed to use the Matlab expression `A*x` within your function.

Hints: You can use two nested `for` loops to compute the vector $\boldsymbol{y}$ (one loop computes the sum (3) while the other one controls the index i in (3))). The size of the matrix `A` can be determined by using the Matlab command `size(A)` (see the Matlab documentation). You can debug your function by comparing the output with the Matlab expression `A*x`, for simple matrices `A` and vectors `x`.

Exercise 4 The number π can be defined as a limit of various converging series of numbers. Among them

$$\pi = \lim_{n\to\infty} \sum_{k=0}^{n} \frac{1}{16^k}\left(\frac{4}{8k+1} - \frac{2}{8k+4} - \frac{1}{8k+5} - \frac{1}{8k+6}\right) \qquad \text{Simon Plouffe (1995),} \tag{4}$$

$$\pi = \sqrt{6}\sqrt{\lim_{n\to\infty} \sum_{k=1}^{n} \frac{1}{k^2}} \qquad \text{Euler (1735).} \tag{5}$$

Write a Matlab function `pi_series.m` that plots in the same figure the first 10 partial sums of the series (4)-(5), that is the vectors `P(1:10)` and `E(1:10)` with components

$$P_{n+1} = \sum_{k=0}^{n} \frac{1}{16^k}\left(\frac{4}{8k+1} - \frac{2}{8k+4} - \frac{1}{8k+5} - \frac{1}{8k+6}\right) \qquad n = 0, 1, 2, ... \tag{6}$$

$$E_n = \sqrt{6}\sqrt{\sum_{k=1}^{n} \frac{1}{k^2}} \qquad n = 1, 2, ... \tag{7}$$

The function should be in the following form

```
function pi_series()
```

Output:
One figure that includes the plots of `P(1:10)` and `E(1:10)`

<u>*Hints:*</u> Once you have constructed the vectors `P` and `E` with as many components as you like, you can plot them in the same figure by using the sequence of Matlab commands `figure(1)`, `clf`, `plot(P(1:10),'r.')` (plots (6) with red dots), `hold`, `plot(E(1:10),'b.')` (plots (7) with blue dots) (see the Matlab documentation for further details). Which sequence between (6) and (7) converges faster to π?

Extra Credit: At the end of the Matlab function `pi_series.m`, write a code that returns in the command window the smallest integer numbers n_1 and n_2 such that

$$|P_{n_1+1} - \pi| < 10^{-5} \qquad |E_{n_2} - \pi| < 10^{-5}. \tag{8}$$

To determine n_1 and n_2, you can use a `for` loop combined with an `if` statement or, equivalently, a `while` loop.