MAT-63506 Scientific Computing

Exercise Set 2 12.–18. 3. 2018

Before doing the exercises read the files “Programming Commands”, “Functions” and “FileIO”.

Exercise 1. Write a function `A = normalize_cols(A, p)` that divides each column of A by its p-norm (`norm`), i.e., replaces column a_j with $a_j/\|a_j\|_p$. The second argument p is optional. If it is not given, it should be set to 2.

Check your function with the 7×7 magic square and $p = 1$, 2, 7.3, 42, and ∞. Store the values in the variables `A1`, ... `A5`.

Exercise 2. Write a function `sqrt_square(x, n)` that first takes the square root of x in a `for`-loop n times and then squares the result n times in another `for`-loop. With exact arithmetic this would leave x unchanged, but with finite-precision floating-point arithmetic this is not the case.. Compute `y2 = sqrt_square(x2, 50)` for `x2 = 0:0.01:10` and plot the error `x2 - y2` with the command `plot(x2, x2 - y2)` (remember to use exactly the names `sqrt_square`, `x2`, and `y2`).

The function should accept an arbitrary array argument (use elementwise squaring) and check that $x \geq 0$ (use `assert` and `all`) and that $n \geq 1$ is an integer (use `assert` and `round`) and if not, print an error message.

Exercise 3. Make a handle to the anonymous function $x \mapsto \ln(x)/(x-1)$ (vectorized) and compute an approximation to its integral from 0 to 1 with the command `integral` setting both the absolute and relative tolerances to 10^{-k} for $k = 1, \ldots, 18$. Store the results into the vector `y3`.

Compare to the exact value $\pi^2/6$ by computing the error `err3 = pi^2/6 - y3`. Plot the tolerance versus the absolute value of the error on a logarithmic scale with `loglog`.

Exercise 4. Write a function `s = file2cellstr(filename)` that reads the text file `filename` and returns the nonempty lines in the cell array `s`.

The argument `filename` is optional. If it is not given, the function should use `uigetfile` to let the user locate the file.

Test your function with the file `ShootInFoot.txt` and store the result in the variable `s`.

HINT: `fopen`, `fclose`, `fgetl`.

Exercise 5. Write a function `cellstr2file(strings, filename)` that writes the strings in the cell array `strings` into the file `filename`, one string per line. If the optional argument `filename` is not given, the function should use `uiputfile` to let the user save the file. The function should check that the first argument `strings` is a cell array of strings (`iscellstr`) and that the second argument is a string (`ischar`).

To test your function load the mat file `'teststrings.mat'` and use the cell array `strings` stored in it. Store the result into the file `ShootInFoot1.txt`.

HINT: `fprintf`, `fopen`, `fclose`.

Exercise 6. Define the functions $C_n : [0,1] \to [0,1]$ recursively as follows: $C_0(x) = x$ and for $n \geq 1$

$$C_n(x) = \begin{cases} \frac{1}{2} C_{n-1}(3x) & \text{if } x \in [0, \frac{1}{3}] \\ \frac{1}{2} & \text{if } x \in (\frac{1}{3}, \frac{2}{3}] \\ \frac{1}{2} + \frac{1}{2} C_{n-1}(3x-2) & \text{if } x \in (\frac{2}{3}, 1]. \end{cases} \tag{1}$$

The Cantor function is defined as the limit $C(x) = \lim_{n\to\infty} C_n(x)$. It has the peculiar property that it is continuous and its derivative is zero almost everywhere and yet it manages to increase from zero to one on the interval $[0, 1]$. The Cantor function is also known as the Devil's Staircase. It is uniformly continuous but not absolutely continuous and forms a counterexample to the fundamental theorem of integral calculus:

$$f(1) - f(0) = \int_0^1 f'(x)\,\mathrm{d}x,$$

since in this case the left hand side is one and the right hand side is zero.

Write a function `cantor(x, n)` that computes the nth approximation to the Cantor function, the function C_n above. The function should check that n is a positive integer and accept an arbitrary array argument x, see the example function `Step2` in `Functions.mlx`. Use a `for`-loop and a local function `cantor_aux(x, n)` that calls itself recursively with a scalar x using the recursion (1).

Test your function with `x6 = 0:0.001:1` and $n = 8$ and store the result into the variable `y6`. Also plot the function with the command `plot(x6, y6)`.