INSTITUTO SUPERIOR TÉCNICO

Artificial Intelligence and Decision Systems (IASD)

python problems

version 2.0 — September 2019

1. Make a function that takes as argument a list of numbers, and returns a list with the non-zero numbers of the given list.

2. Make a function that returns `True` or `False` depending on whether its argument n is prime

3. Make a function to return a sequence of the first k primes, where k is the function argument.

4. Make a function with the same functionality as the last one, but more efficient, by using the previously found primes (n is prime if it is not divisible by any prime $k < \sqrt{n}$).

5. Let x and y be two column vectors of the same dimension, represented as lists, for instance

```
x = [1, 2, 3, 4, 5]
y = [6, 7, 8, 9, 0]
```

   Write Python functions to perform the following mathematical operations:

   (a) inner product, that is, $x^T y$

   (b) $x\, y^T$, where the resulting matrix is represented as a list of rows, where each row is a list

   (c) upper triangular Toeplitz matrix, using the above mentioned matrix format

   (d) circular Toeplitz matrix, using the above mentioned matrix format

   Hint 1: Find how all of these cases can be solved in one line of code
   Hint 2: Use list comprehensions

6. Make a program to estimate π using the following Monte Carlo method: consider a circle of radius r inside a square of side $2r$, whose sides are tangent to the circle; randomly draw points inside the square, with a uniform

distribution; since the ratio between the circle area and the square area is $\pi r^2/(2r)^2 = \pi/4$, the probability of each point falling into the circle will be $\pi/4$; by counting the number of points that fall into the circle (over the total amount of points), one can therefore estimate π; the more points are drawn, the more precise the result will be. (Note that the result is independent of the value of r)

7. Make a program to solve Sudoku problems[1] using the following method (called *backtrack search*): given a Sudoku board, first check if all squares are filled with numbers, if yes, return the board, otherwise, choose one unfilled square, and for each number $n = 1, \ldots, 9$ check if n in that square conflicts with the rest of the board; if yes, try another one, otherwise, repeat recursively the process with the new board (*i.e.,* with n in the chosen square); if at any point all 9 numbers are inconsistent, leave the square unfilled and, return failure to the calling function.

[1] See for instance: `http://en.wikipedia.org/wiki/Sudoku`