Applied Computational Economics
Problem Set 0
Prologue on Numerical Solutions, Coding and Matlab

The University of Nottingham

2020

Q1 (Find). Create a 101×1 array and fill it with the numbers from 100 to 200. Use the *find* command to get the array index, which contains the number 151.

Q2 (Pseudo-random numbers). Draw a vector of pseudo-random numbers of size 10×1 using the *rand* command.

Q3 (Pseudo-random generator seed). What is a pseudo-random number seed? Run the following sequence of commands to find out.

1. Set the seed to 10 using the *rng* command.
2. Draw a 10×1 vector of pseudo-random numbers using this seed.
3. Draw another 10×1 vector of pseudo-random numbers using this seed.
4. Use the *rng* command to set the seed back to 10.
5. Repeat steps 2 and 3 again. What do you notice?

Q4 (Functions and plots). Generate a 10×1 vector of numbers denoted by $\vec{v}$, in order, from 1 through 9. Create a function that takes this array as an input. Denote the function's output by $\vec{w}$, which is also a 10×1 vector. The output takes each element of $\vec{v}$, (denote the i^{th} entry by $\vec{v}_i$) and transforms it using the equation $w_i = 10\vec{v}_i - 5$ for $i \in \{1, ..., 10\}$. After calling the function, plot the output $\vec{w}$ using the *plot* command. Label the horizontal axis v using the *xlabel* command and the vertical axis w using the *ylabel* command. Create a title for the figure using the *title* command and a legend using the *legend* command.

Q5 (Vectorisation). Consider the multivariate function

$$f(x, y) = xy - y^2.$$

Let's think about maximising this function with respect to y for a fixed value of x. That is — we seek to find a function $y^*(x)$, which maximises $f(x, y)$.

1. Find $y^*(x)$ analytically (using pen and paper).

Now let's think about finding $y^*(x)$ numerically with Matlab. Let's restrict our attention to values of x over the interval $\bar{x} = [1, 1000]$.

2. Discretise the space $\bar{x}$ by creating a array of gridpoints, denoted by $\vec{x}$, over the interval $\bar{x}$. Create this array such that its first entry is 1 and it increases by intervals of 0.1 all the way up to 1000.

Let's restrict our permissible choices of the y variable to be contained to the same grid as x: i.e. $\vec{y} = \vec{x}$.

We'll now solve this problem numerically using two methods — the first using nested *for loops* and the second using *vectorisation*. For the first approach:

3. Use the *size* command to find the dimensionality of the array $\vec{x}$. Denote the answer $|\vec{x}|$. Obviously since $\vec{y} = \vec{x}$, it follows that $|\vec{y}| = |\vec{x}|$.
4. Use two nested *for loops* (over the x and y values) to fill an array of size $|\vec{x}| \times |\vec{y}|$ with all permissible values of $f(x, y)$. Hint: fill in the formula for $f(x, y)$ on the right-side of the equality on line 3 below.

```
1    for i = 1:length(x_vec)
2        for j = 1:length(y_vec)
3            f(i,j) =                              ;
4        end
5
6    end
```

5. Using the *max* command, find the optimal y variable gridpoint from $\vec{y}$, that maximises $f(x, y)$ for a given value of $x \in \vec{x}$. Hint: your use of the *max* command should go on line 5 above.
6. Time the execution of the two nested loops using the *tic* and *toc* commands. Plot the solution for $y^*(x)$.

Now let's think about the vectorisation approach. Vectorising refers to the process of writing your code in terms of matrix operations rather than loops. Since Matlab is designed for processing matrices, this is the desirable approach (when possible).

We can vectorise the code by creating a 2D matrix, which gives all possible values of $f(x, y)$, given our discretised vectors $\vec{x}$ and $\vec{y}$. Let's denote this matrix by F. The idea is that the rows of this matrix will correspond to our possible x values and the columns correspond to our permissible choices of y. Your optimal choice of y for a given x is then the column that offers the highest value of the F matrix for a given row.

7. Create a matrix called X of size $|\vec{x}| \times |\vec{y}|$. Each column, from first row to last, should contain a copy of the $\vec{x}$ array.
8. Create a matrix called Y of size $|\vec{x}| \times |\vec{y}|$, which is simply the transpose of X.
9. Create the F matrix using X and Y, in addition to Matlab's matrix operations.
10. Use the *max* command to find the column of F, which offers the highest value, for a given row.
11. Time the execution of this vectorised code using the *tic* and *toc* commands. How does the speed of this code compare to the nested for loops?