MH2401 — Algorithms and Computing III

Handout for weeks 1–2

AY 2018/19, S1

Aim The aim of this lab is to learn how to work with matrices in Python.

Objectives At the end of this lab, students should be able to:

(a) Explain the order of precedence of arithmetic, relational and logical operators.

(b) Perform matrix/array creation, concatenation, arithmetic operations, and element-wise operations.

(c) Use row-column, linear and logical indexing to extract part(s) of a matrix.

Your task

(a) Complete as many exercises as possible.

(b) If you need help, you are encouraged to ask the lab tutors.

(c) Please put in a serious effort to complete each exercise on your own before consulting the tutors or your friends.

(d) Do NOT submit your work.

(e) When the lab session is over, please submit your feedback by filling an online form (the link is in the end of this handout).

Exercises

Operations

Here we review basic operations and order of precedence in Python. To solve exercises of this section, you can work with the command line Python.

Exercise 1.
Try the following pairs of commands in Python. Why do the pairs of commands produce the same/different outputs?

(a)	`-2 ** 4`	`(-2) ** 4`
(b)	`-2 ** 4 * 3`	`-2 ** (4 * 3)`
(c)	`2 / 3 * 4`	`2 / (3 * 4)`

(d) `2 * 3 / 4`   `2 * (3 / 4)`

(e) `3 > 2 > 1`   `(3 > 2) > 1`

(f) `2 < 1 and 3 < 4 or 4 < 5`   `2 < 1 and (3 < 4 or 4 < 5)`

Exercise 2.
Write down a *single* Python command to evaluate each of the following expressions.
[Hint: use round brackets at appropriate places.]

(a) $2 \times 4^{-7\times5/6} - 1$

(b) 2^{2^3}

Exercise 3.
Try the following commands with different real values for `a`. When `a` is an arbitrary real number, what outputs do the following Python commands produce? Why?
[Hints: try values of `a` that you think would give different answers to each of the inequalities.]

(a) `0 <= a <= 10`

(b) `a * (a > 0)`

(c) `(a > 5) - (a < 5)`

Vector operations

Here we work with vector operations in Python. You will need to import the library *numpy* by typing the command `import numpy as np`. These exercises are harder than exercises in the previous section and it is probably a good idea to use Python scripts here rather than typing commands in the command line.

We will need the following function (it's a good idea to read a manual on it): `arange`. Note that we need to add `np.` to the function's name to specify that it comes from the package *numpy*.

Exercise 4.
Write down a *single* Python command to produce each of the following vectors (loops not allowed) In these vectors, n is a positive integer whose value is given in the Python variable `n`.

(a) $\begin{bmatrix} 2 & 4 & 8 & \cdots & 2^n \end{bmatrix}$

(b) $\begin{bmatrix} 1 & 3 & 6 & \cdots & \frac{n(n+1)}{2} \end{bmatrix}$

Matrix generation and matrix operations

Here we will work with matrices in Python. Still, we need to import the library *numpy* by typing the command `import numpy as np`. We will need the following functions (it's a good idea to read a manual on them): `arange`, `matrix`, `append`, `concatenate`, `transpose`. Note that we need to add `np.` to the function's name to specify that it comes from the package *numpy*.

We will review the differences between *array* and *matrix* in *numpy*. Before going on with the next question, run the following commands in Python:

```
a = np.array([[0, 2, -2],
              [1, 1, 3]])

b = np.array([[1, 2, 3],
              [-1, -2, -4]])

A = np.matrix([[0, 2, -2],
               [1, 1, 3]])

B = np.matrix([[1, 2, 3],
               [-1, -2, -4]])
```

Exercise 5.
First, try running `print(a)` and `print(A)`. Note that the output looks the same. Both variables represent the matrix

$$\begin{bmatrix} 0 & 2 & -2 \\ 1 & 1 & 3 \end{bmatrix}$$

Try the following pairs of commands in Python. Why do the pairs of commands produce the same/different outputs?

(a) `a + b` `A + B`

(b) `a ** 2` `A ** 2`

(c) `a * b.T` `A * B.T`

(d) `a.dot(b.T)` `a @ b.T`

(e) `np.matrix(a).T * np.matrix(b)`
`A.T * B`

Exercise 6.
Write down a *single* Python command to produce each of the following matrices (loops are not allowed). In these matrices, m and n are positive integers whose values are given in the Python variables `m` and `n`, respectively.

(a) $$\begin{bmatrix} 0 & 1 & 2 & \cdots & n \\ 1 & 1 & 1 & \cdots & 1 \\ 2 & 1 & 1 & \cdots & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ m & 1 & 1 & \cdots & 1 \end{bmatrix}$$

(b) $$\begin{bmatrix} 1 & \cdots & j & \cdots & n \\ \vdots & & \vdots & & \vdots \\ i & \cdots & i \times j & \cdots & i \times n \\ \vdots & & \vdots & & \vdots \\ m & \cdots & m \times j & \cdots & m \times n \end{bmatrix}$$

Matrix subsetting

Here we will work with matrices in Python. Still, we need to import the library *numpy* by typing the command `import numpy as np`.

Let's create a 5×6 matrix M in Python by executing the following command:

```
M = np.matrix([[1, -2, 0, -4, 5, 6],
               [7, -8, 0, -10, 11, -12],
               [13, 0, 15, -16, 17, -18],
               [19, -20, 21, -22, 23, -24],
               [25, -26, 0, -28, 29, -30]])
```

Exercise 7.
When `M` is an arbitrary matrix with at least 5 rows and 5 columns, and `i` and `j` are arbitrary integers between 0 and 4, what outputs do the following Python commands produce?

(a) `M[i, j]`

(b) `M[i, :]`

(c) `M[:, j]`

(d) `M[:2, -2:]`

(e) `M[::2, :]`

(f) `M[:, 1::2].shape[0]`

(g) `M[:, 1::2].shape[1]`

(h) `M[-1, :] == 0`

(i) `M[-1, :] = 0`

(j) `np.delete(M, 3, 1)`

Exercise 8.
When `M` is an arbitrary matrix with at least 5 rows and 5 columns, and `i` is an arbitrary integer between 0 and 4, what outputs do the following Python commands produce?

(a) `np.append(M[:2 , 0], M[-2:, -1], axis = 1)`

(b) `M[i]`

(c) `M[np.array([2, 4, 1]), np.array([3, 0, 0])]`

Exercise 9.
Try the following commands with different matrices for `M`. When `M` is an arbitrary matrix, what outputs do the following Python commands produce?
[Hints: use matrices `M` with both positive and negative entries, all positive entries, and all negative entries.]

(a) `M > 0`

(b) `np.where(M == 0)`

(c) `M[M < 0]`

(d) `np.any(M > 0)`

(e) `np.all(M >= 0)`

Exercise 10.
Write a *single* Python command to produce a given matrix `M` with all its negative entries changed to 2401.

Extra Questions

Exercise 11.
Write a *single* Python command to output the value of each of the following functions at the real number x, whose value is given in the Python variable `x`.

[Hint: refer to exercise 4(b).]

(a) $f(x) = \begin{cases} x^2 & \text{if } x \geq 1, \\ 1 & \text{if } x < 1. \end{cases}$

(b) $g(x) = \begin{cases} 0.5 & \text{if } \sin(x) > 0.5, \\ \sin(x) & \text{if } -0.5 \leq \sin(x) \leq 0.5, \\ -0.5 & \text{if } \sin(x) < -0.5. \end{cases}$

For the last question, we need to create a *square* matrix X of size $n \times n$. Below is a Python command that creates the 9×9 matrix X whose $(i.j)$th entry is $\cos i \times \sin j$ rounded to 2 decimal places:

```
n = 9
X = np.round(np.matrix(np.cos(np.arange(1, n+1))).T *
                np.matrix(np.sin(np.arange(1, n+1))) , 2)
```

Exercise 12.
Use linear indexing to perform each of the following exercises with a *single* Python command. Note that "linear indexing" means that you are not supposed to use *numpy* functions, such as `diag`:

(a) Extract the main diagonal of a square matrix `X`. In other words, form the vector `X[0, 0], X[1, 1], ..., X[n-1, n-1].`

(b) Extract the anti-diagonal of a square matrix `X`, i.e., form the vector `X[n-1, 0], X[n-2,1], ..., X[0, n-1].`

(c) Flip the matrix `X` about the anti-diagonal, i.e., form a new matrix whose (i, j) th entry is the same as the $(n-j, n-i)$ th entry of `X`.

Exercise 13.
Create each of the following matrices A of size $n \times n$ using a single Python command (no loops are allowed):

(a) $a_{ij} = \min(i, j)$, i.e.,

$$A = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & 2 & 2 & \cdots & 2 \\ 1 & 2 & 3 & \cdots & 3 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 2 & 3 & \cdots & n \end{bmatrix}$$

(b) The matrix whose entries above the anti-diagonal are 0 and on the anti-diagonal and below it 1. For instance, for $n = 5$, we would have

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Please submit your feedback on this lab. For instance, the course instructors need to know if this handout is sufficient to understand the material and if lab assistants are helpful and their explanations are clear.

- `https://tinyurl.com/mh2401-week1-feedback`