# Lab 1

**Name: (your name)**

Some remarks before you start:

- You are currently working on a Google Colab notebook, a convenient environment that allows you to write texts, Python codes, and execute the codes. This notebook runs in a temporary virtual machine, which gets reset each time you close the session. You will need to import the below Python packages again each time you start a session.
- The programming language used in this lab is Python, which serves as a tool to implement algorithms rather than be a primary focus by itself. Therefore, technical details of Python that are not crucial to know for problem-solving will be skipped.
- You can double click on any cell to edit it or to learn how to type math objects (matrix, equations,...) or format text (bolden, underline,...) Once inside a cell, you can execute it by pressing **Shift+Enter**.
- Before submitting your report, remove all ouputs by clicking on **Edit→Clear all ouputs**. Then download your report by clicking on **File→Download→.ipynb**. This is the file you will submit on Canvas.

In this lab, you will practice:

- perform row operations on a matrix
- solve a linear system of equations
- solve a linear programming problem

| Problems | Points |
|---|---|
| $1, 4, 6, 9$ | 2 |
| $2, 3, 5$ | 3 |
| $7, 8, 10$ | 4 |
| Readability of your report | 3 |
| Total: 10 | Total: 32 |

## I. Import necessary Python packages

Execute the following code each time you start a session.

```
from numpy import* # import everything from the standard Python numeric package (NumPy)
from sympy import* # import everything from the standard Python symbolic-computing package (SymPy)
from scipy import* # import everything from the standard Python scientific-computing package (SciPy)
```

## II. Matrix and row operations

A *vector* is an array of numbers. You enter an array using the command **array**. Entries are separated from each other by commas.

Enter the vector $a = [1, 2, 4]$ as follows:

```
a=array([1,2,4])
print(a)
```

A *matrix* is a two-dimensional array. You can view a matrix as an array of its rows, each of which is an array itself. For example, you can view the matrix

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

as an array of three entries: $[1, 2, -1]$, $[3, 2, 0]$, $[1, 0, 2]$. Therefore, you can enter this matrix as follows.

```
A=array([[1,2,-1],[3,4,0],[1,0,2]],dtype=float64)
print(A)
```

## Exercise 1

Each entry of an array can be accessed by its index. In Python, the starting index is 0 (not 1). Run the code and write your observation:

```
print(a[0])
print(a[1])
print(a[2])
print(a[3])
```

## Exercise 2

You can multiply a vector by a number or add two vectors of the same length. Explain what each of the following commands does.

```
print(3*a)
b=array([3,5,2])
print(b)
print(a+b)
c=a+2*b
print(c)
```

## Exercise 3

You can perform row operations on a matrix, such as subtracting three times of the first row from the second row and subtracting the first row from the third row:

```
A[1] = A[1] - 3*A[0]
A[2] = A[2] - A[0]
print(A)
```

Continue to do more row operations to turn matrix $A$ into a reduced row echelon form (RREF).

## Exercise 4

You can use the built-in command **rref** to find the RREF of a matrix. Try the following:

```
A=Matrix([[1,2,-1],[3,4,0],[1,0,2]])
A.rref()[0]
```

Note:

- The reason why we write **A.rref()[0]** instead of just **A.rref()** is because the former command gives us exactly what we need (the RREF of matrix $A$), whereas the latter command gives some additional information that we are not interested in.
- In order to use the built-in command **rref**, you need to enter the matrix using **Matrix** (as above) instead of **array**. The reason is a little bit technical: **rref** is built in the SymPy package, not NumPy package. **Matrix** command generates

an object in the SymPy package, whereas **array** command generates an object in the NumPy package.

Use the built-in **rref** command to find the RREF of the following matrix:

$$B = \begin{bmatrix} 2 & -4 & 2 \\ 3 & -4 & 5 \\ 0 & 1 & 1 \\ -3 & 5 & 4 \end{bmatrix}$$

## III. Linear system of equations

### Exercise 5

Consider the following linear system of equations:

$$\begin{aligned} x + y + z &= 4 \\ x + 2y + 4z &= 12 \\ 2x - 3y - z &= 4 \end{aligned}$$

- Write the matrix associated with this system.
- Use the command **rref** to find the RREF of this matrix.
- Solve the system based on the RREF you just found.

### Exercise 6

You can use the built-in command **linalg.solve** to solve a linear system. For example, consider the system

$$\begin{aligned} x + 2y - z &= 4 \\ 3x + 4y &= 5 \\ 2x + z &= 3 \end{aligned}$$

This system has a *coefficient* matrix:

$$A = \begin{bmatrix} 1 & 2 & -1 \\ 3 & 4 & 0 \\ 2 & 0 & 1 \end{bmatrix}$$

and a *right-hand-side* vector:

$$b = \begin{bmatrix} 4 \\ 5 \\ 3 \end{bmatrix}$$

You will enter maxtrix $A$, array $b$, and use the command **linalg.solve** as follows.

```
A=array([[1,2,-1],[3,4,0],[2,0,1]])
b=array([4,5,3])
linalg.solve(A,b)
```

Note: the command **linalg.solve** can successfully solve a system only if it has a unique solution (like in the example above). If the system does not have a solution (inconsistent system) or has infinitely many solutions, **linalg.solve** will throw an error. In that case, you need to find the RREF of the associated matrix and use it to solve the system (as in the Exercise 5).

Solve the linear system in Exercise 5 using **linalg.solve**.

### Exercise 7

Balance the following chemical equation

$$\mathrm{KMnO_4 + HCl \rightarrow KCl + MnCl_2 + H_2O + Cl_2}$$

In other words, find positive intergers $x_1, x_2, \ldots, x_6$ such that

$$x_1\mathrm{KMnO_4} + x_2\mathrm{HCl} = x_3\mathrm{KCl} + x_4\mathrm{MnCl_2} + x_5\mathrm{H_2O} + x_6\mathrm{Cl_2}$$

*Hint:* You will need to write down the system of linear equations, its associated matrix, and find its RREF. You will notice that this system has infinitely many solutions. Just choose one solution in which $x_1, x_2, \ldots, x_6$ are all integers.

## IV. Linear programming

Consider a linear programming problem of minimizing $C = -2x + y$ subject to

$$\begin{aligned} x - y &\leq 3 \\ 2x + 3y &\leq 12 \\ x + y &\geq 2 \\ x, y &\geq 0 \end{aligned}$$

To solve this problem using the **simplex method**, you will first convert it into a standard form: maximizing $P = 2x - y$ subject to

$$\begin{aligned} x - y &\leq 3 \\ 2x + 3y &\leq 12 \\ -x - y &\leq -2 \\ x, y &\geq 0 \end{aligned}$$

Next, you introduce slack variables to turn all inequalities to equalities: maximizing $P = 2x - y$ subject to

$$\begin{aligned} x - y + s_1 &= 3 \\ 2x + 3y + s_2 &= 12 \\ x + y - s_3 &= 2 \\ x, y, s_1, s_2, s_3 &\geq 0 \end{aligned}$$

Notice that the third equation has been multiplied by $(-1)$ to ensure that the right hand side is nonnegative. You rewrite the equation $P = 2x - y$ as

$$-2x + y + P = 0$$

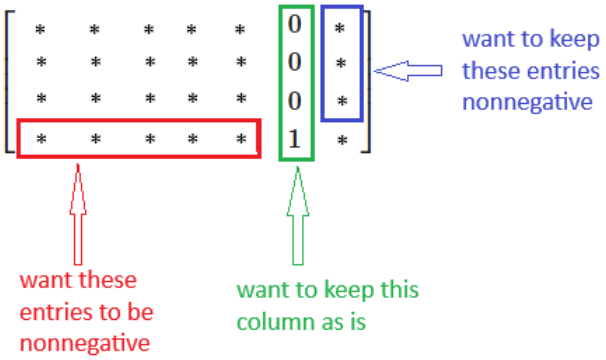
making sure that the coefficient of $P$ is $1$ (not $-1$).

Next, you construct the matrix associated with the system of the above four equations:

$$A = \begin{bmatrix} 1 & -1 & 1 & 0 & 0 & 0 & 3 \\ 2 & 3 & 0 & 1 & 0 & 0 & 12 \\ 1 & 1 & 0 & 0 & -1 & 0 & 2 \\ -2 & 1 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

```
A=array([[1,-1,1,0,0,0,3],[2,3,0,1,0,0,12],[1,1,0,0,-1,0,2],[-2,1,0,0,0,1,0]],dtype=float64)
print(A)
```

Your next goal is to use row operations to turn the matrix into the following form:



The key column is the first column and the pivot element is $1$ on the third row. You then use elementary row operations to turn the key column into a column that has $1$ at the pivot element and $0$'s elsewhere.

```
A[0] = A[0] - A[2]
A[1] = A[1] - 2*A[2]
A[3] = A[3] + 2*A[2]
print(A)
```

The key column is now the fifth column and the pivot element is $1$ on the first row. You then use elementary row operations to turn the key column into a column that has $1$ at the pivot element and $0$'s elsewhere. Note that the operation $R_2 = R_2 + R_4$ is not acceptable because it would change the sixth column, which you do not want to change (see the figure above).

```
A[1] = A[1] - 2*A[0]
A[2] = A[2] + A[0]
A[3] = A[3] + 2*A[0]
print(A)
```

The key column is now the second column and the pivot element is $5$ on the second row. You then use elementary row operations to turn the key column into a column that has $1$ at the pivot element and $0$'s elsewhere.

```
A[1] = 1/5*A[1]
A[0] = A[0] + 2*A[1]
A[2] = A[2] + A[1]
A[3] = A[3] + A[1]
print(A)
```

This matrix is already in the desirable form. It corresponds to the linear system

$$0.2s_1 + 0.4s_2 + s_3 = 3.4$$
$$y - 0.4s_1 + 0.2s_2 = 1.2$$
$$x + 0.6s_1 + 0.2s_2 = 4.2$$
$$1.6s_1 + 0.2s_2 + P = 7.2$$

The maximum value of $P$ is $7.2$ and it is attained when $s_1 = s_2 = 0$. Substituting $s_1 = s_2 = 0$ into the system, you can easily see that $x = 4.2$ and $y = 1.2$. We conclude that the minimum value of $C$ is $-7.2$ and it is attained when $x = 4.2$ and $y = 1.2$.

## Exercise 8

Use the simplex method (as illustrated above) to solve the following linear programming problem:

Maximize $P = x + 4y - z$ subject to

$$x - y \geq -4$$
$$3x - y + z \geq 0$$
$$2x + 2y + z \leq 2$$
$$x, y, z \geq 0$$

## Exercise 9

You can use the built-in command **optimize.linprog** to solve a linear programming problem. To use this command, your problem must be

- a minimizing problem,
- all constraints are of the form $\leq$,
- all variables are nonnegative.

For example, consider a linear programming problem discussed earlier: minimize $C = -2x + y$ subject to

$$\begin{aligned} x - y &\leq 3 \\ 2x + 3y &\leq 12 \\ x + y &\geq 2 \\ x, y &\geq 0 \end{aligned}$$

This problem is already a minimizing problem. Only the third inequality needs to be reversed. You rewrite the problem as: minimize $C = -2x + y$ subject to

$$\begin{aligned} x - y &\leq 3 \\ 2x + 3y &\leq 12 \\ -x - y &\leq -2 \\ x, y &\geq 0 \end{aligned}$$

The array of coefficients in the objective function $-2x + y$ is

$$c = [-2, 1]$$

The matrix of coefficients in the constraints is

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

The array of the right-hand-side of the constraints is

$$b = [3, 12, -2]$$

Now you can use **optimize.linprog** to solve the problem as follows.

```
c = array([-2,1])
A = array([[1,-1],[2,3],[-1,-1]])
b = array([3,12,-2])
opt = optimize.linprog(c,A,b)
print(opt)
```

The output says that the minimum value of the objective function is $-7.2$ and it is attained at $x = 4.2$ and $y = 1.2$.

Use **optimize.linprog** to solve Exercise 8.

## Exercise 10

A candy company makes three types of candy (solid, fruit, and cream-filled) and packages these candies in three different assortments. A box of assortment I contains 4 solid, 4 fruit, 12 cream and sells for $9.40. A box of assortment II contains 12 solid, 4 fruit, 4 cream and sells for $7.60. A box of assortment III contains 8 solid, 8 fruit, 8 cream and sells for $11.00. The manufacturing costs per candy are $0.20 for solid, $0.25 for fruit, and $0.30 for cream. The company can manufacture up to 4800 solid, 4000 fruit, and 5600 cream candies weekly. How many boxes of each type should the company produce in order to maximize profit? What is their maximum profit?

Setup the linear programming model. Then use **optimize.linprog** to solve it.