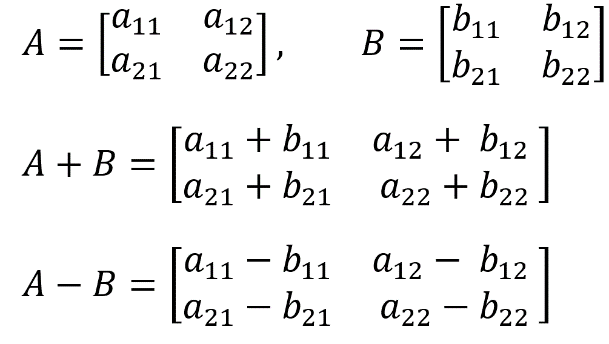
Homework Assignment #1

Due: Oct 16, 2020

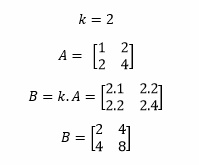
21700242 문선빈

**1. Evaluate the below algebraic expressions. Implement the solution in Python (using NumPy) and submit your scripts using the supplied Jupyter Notebook (homework1.ipynb).**

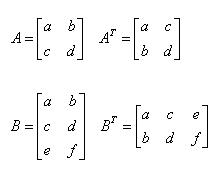
1. Adding and supplement of array calculated below process.



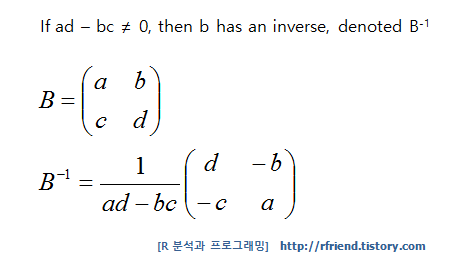
1. Multiple scalar to array calculated below process.



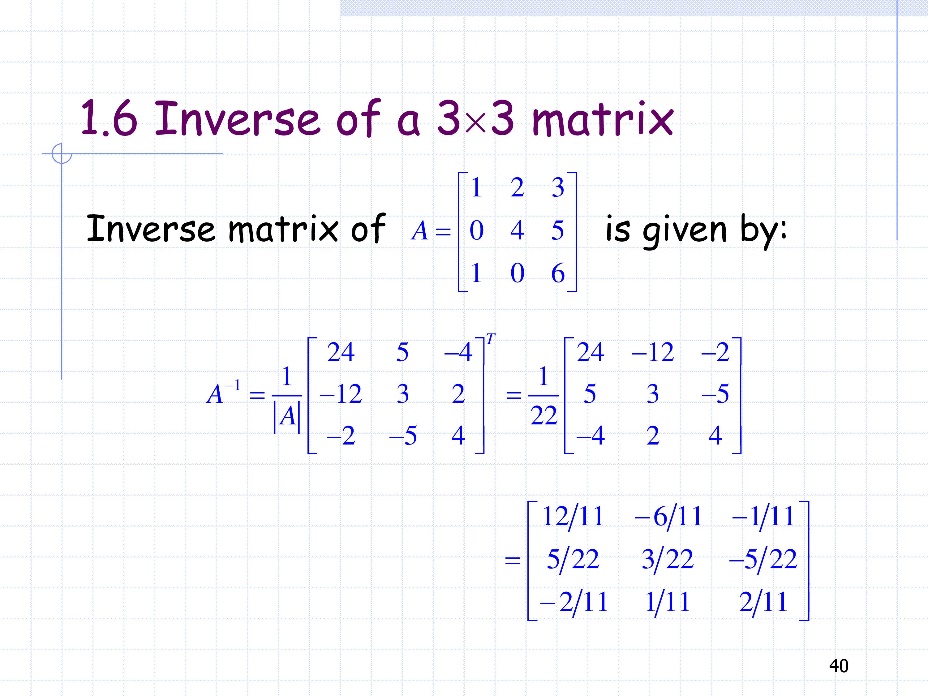
1. Transpose of the array calculated below process.



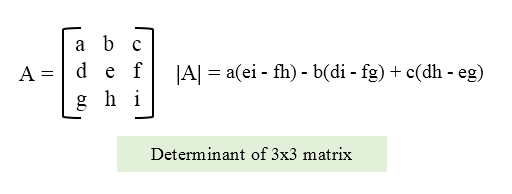
1. Inverse of 2\*2 matrix



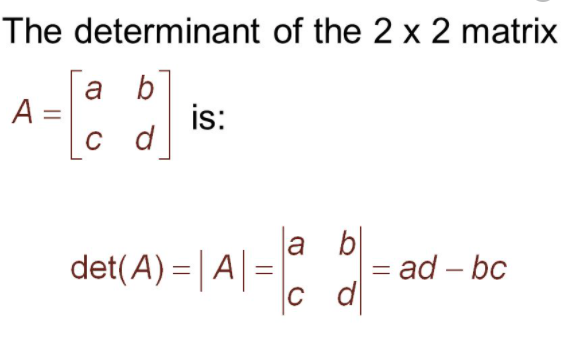
1. Inverse of 3\*3 matrix. C.f) using option (6), to calculate determinant first.



1. Determinant of 3\*3 matrix



1. Determinant of 2\*2 matrix



(a)

Sol) Using (1) option to take the answer

A = np.array([[1, 2, 3], [4, 5, 6]])

B = np.array([[5, 1, 3], [3, 2, 2]])

sol = A+B

print(sol)

[[6 3 6]

[7 7 8]]

(b) Using (1) and (2) option to take the answer

Sol)

# Problem 1 (b)

A = np.array([[3, 1, 3], [1, 2, 2]])

B = np.array([[8, 2, 2], [4, 4, 6]])

sol = 2\*A-0.5\*B

print(sol)

[[2. 1. 5.]

[0. 2. 1.]]

(c) Using (1) and (2) option to take the answer

Sol)

# Problem 1 (c)

A = np.array([[3, 1, 3], [1, 2, 3]])

B = np.array([[1, 5, 3], [5, 4, 1]])

sol = 0.5\*(A+B)

print(sol)

[[2. 3. 3.]

[3. 3. 2.]]

(d)

Sol) Using (1) and (3) option to take the answer. Using transpose method because we have to math the first array’s column and second array’s row number.

A = np.array([[3, 1, 2], [1, 2, 3]])

B = np.array([[1, 1, 0], [2, 0, 1]])

sol = np.matmul(A, np.transpose(B))

print(sol)

[[4 8]

[3 5]]

(e)

Sol) Trace is adding of each diagonal components.

1+4+7 = 12

A = np.array([[1, 3, 2], [6, 4, 5], [9, 8, 7]])

sol = np.trace(A)

print(sol)

12

(f)

Sol) Using (4) to make inverse matrix.

A = np.array([[3, 2], [-7, -5]])

sol = np.linalg.inv(A)

print(sol)

[[ 5. 2.]

[-7. -3.]]

(g)

Sol) Using (5) and (6) to make inverse matrix.

A = np.array([[1, -3, 5], [-1, 4, -5], [-1, 3, -6]])

sol = np.linalg.inv(A)

print(sol)

[[ 9. 3. 5.]

[ 1. 1. 0.]

[-1. -0. -1.]]

(h)

Sol) Using (7) to calculate determinant

A = np.array([[1,3], [4, -5]])

sol = np.linalg.det(A)

print(sol)

-17

**2. Solve the following systems of linear equations. Implement the solution in Python (using NumPy) and submit your scripts using the supplied Jupyter Notebook (homework1.ipynb).**

(a)

Sol)

# 2x + y = 5

# -x + 3y = 1

from sympy import Symbol, solve

x = Symbol('x')

y = Symbol('y')

equation1 = 2\*x + y - 5

equation2 = -x + 3\*y - 1

sol = solve((equation1, equation2), dict=True)

print(sol)

[{x: 2, y: 1}]

(b)

Sol)

# 5x + y - 2z = -4

# 2x + y + 5z = 1

# x - y + 2z = 1

from sympy import Symbol, solve

x = Symbol('x')

y = Symbol('y')

z = Symbol('z')

equation1 = 5\*x + y - 2\*z + 4

equation2 = 2\*x + y + 5\*z - 1

equation3 = x - y + 2\*z - 1

sol = solve((equation1, equation2, equation3), dict=True)

print(sol)

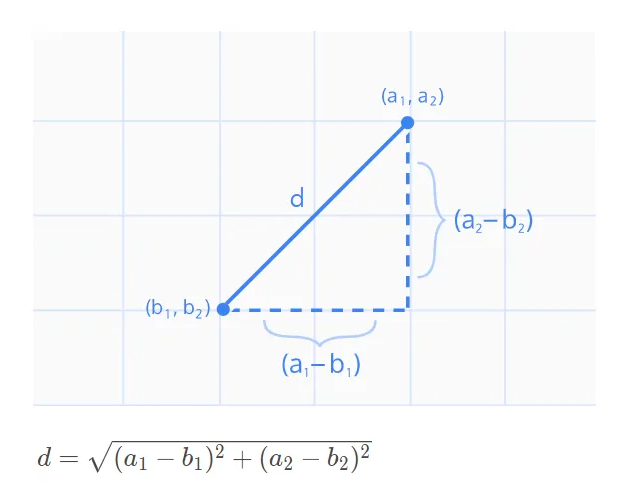
[{x: -1/2, y: -1/2, z: 1/2}]

**3. Clustering with the *k*-Means algorithm**.

(a)

Definition : calculating distance between two points straightly.

The formula is below



Code :

def euclidian\_distance(a, b):

"""

Takes two vectors and returns the Euclidean distance between them

Input: a - input vector a

b - input vector b

Output: the Euclidean distance between vectors a and b

"""

distance = 0

for i in range(len(a)):

distance += (a[i]-b[i])\*\*2

return distance \*\* 0.5

(b)

* What do you need to provide for ‘X’ and ‘K’?

X : input dataset.

K : Number of groups expected to be found in the dataset.

* What do the values ‘max\_iter’ and ‘tol’ do?

Max\_iter : K-Means algorithm is an algorithm that goes around iteration to find the right K. However, the algorithm can also spin an infinite loop without stopping at tolerance, giving it an option called Max\_iter to stop.

Tol : A conditional statement that stops the algorithm, if the error is less than the Tol, the algorithm stops and the answer is obtained.

* What do the output ‘c’ and ‘centroids’ contain?

C : The euclidian distance returns which centroids each value belongs to.

Centroids : Coordinate value of final K

* What do the output ‘log\_centroids’, ‘log\_c’, and ‘log\_sse’ contain?

Log\_centroids : As we record past centroids values, we can see in this data in which order we find the centroids.

Log\_c : It is a value that stores all the past c values.

Log\_sse : It is a value that stores all the past sse values. Sse means total error.

(c)

K = 4

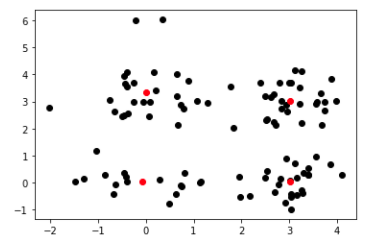
# k = 4

X = generate\_random\_data()

[c, centroids, log\_centroids, log\_c, log\_sse] = kmeans(X, 4, max\_iter=100, tol=0.00005, distance\_metric=euclidian\_distance)

plt.scatter(X[:,0], X[:,1], c='k')

plt.scatter(centroids[:,0], centroids[:,1], c='red')



K = 2

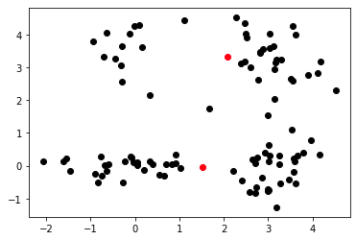
# k = 2

X = generate\_random\_data()

[c, centroids, log\_centroids, log\_c, log\_sse] = kmeans(X, 2, max\_iter=100, tol=0.00005, distance\_metric=euclidian\_distance)

plt.scatter(X[:,0], X[:,1], c='k')

plt.scatter(centroids[:,0], centroids[:,1], c='red')



K=4 is better than K=2. The reason is purpose of the K-Means algorithm is separate similar data into the same group and non-similar data into different groups. K=4 represents purpose of K-Means algorithm than K=2.

(d)

- What does stats.describe(my\_data) do? What kind of information can you find out from the output of stats.describe(my\_data) mean?

scipy.stats ltd.Describe is a function that returns the state of input data.

The values of the output returned in this function are as follows:

Nobs : Number of observations (length of data along *axis*). When ‘omit’ is chosen as nan\_policy, each column is counted separately.

Minmax : Minimum and maximum value of data array.

Mean : Arithmetic mean of data along axis.

Variance : Unbiased variance of the data along axis, denominator is number of observations minus one.

Skewness : Skewness, based on moment calculations with denominator equal to the number of observations, i.e. no degrees of freedom correction.

Kurtosis : Kurtosis (Fisher). The kurtosis is normalized so that it is zero for the normal distribution. No degrees of freedom are used.

\*reference : <https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.describe.html>

In out dataset, we can find all of these parameter.

DescribeResult(nobs=200, minmax=(array([ 0., 18., 15., 1.]), array([ 1., 70., 137., 99.])), mean=array([ 0.56, 38.85, 60.56, 50.2 ]), variance=array([2.47638191e-01, 1.95133166e+02, 6.89835578e+02, 6.66854271e+02]), skewness=array([-0.24174689, 0.48191947, 0.31942368, -0.04686531]), kurtosis=array([-1.94155844, -0.68475139, -0.12589283, -0.83595827]))

Normalization :

I use mean removal and variance scaling. Use numpy

z = (x - mean())/std()

mean :

Code :

# Problem 3 (d)

def normalize(X):

"""

mean-std normalization (standardization)

Takes and applies a normalization on the input matrix X

Input: X - data matrix (numpy array)

Output: X\_normalized - normalized data matrix

"""

N, d = X.shape

info = stats.describe(X)

X\_normalized = np.zeros((N, d))

for i in range(d):

for k in range(N):

X\_normalized[k][i] = (X[k,i] - info.mean[i]) / info.variance[i]

return X\_normalized

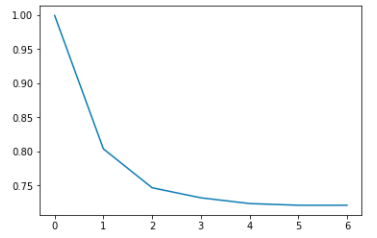
(e)

# Problem 3 (e)

X\_normalized = normalize(my\_data)

[c, centroids, log\_centroids, log\_c, log\_sse] = kmeans(X\_normalized, 5, max\_iter=100, tol=0.00005, distance\_metric=euclidian\_distance)

plt.plot(log\_sse)



The sse has been reduced to the number that passes through each iteration. In particular, after the first iteration, the number decreased rapidly, and it was almost unchanged from iteration 2.

(f)

Code :

K = 7

total\_sse = []

for i in range(1,K):

[c, centroids, log\_centroids, log\_c, log\_sse] = kmeans(X\_normalized, i, max\_iter=100, tol=0.00005, distance\_metric=euclidian\_distance)

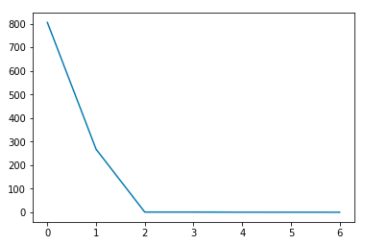
avg = sum(log\_sse)/len(log\_sse)

total\_sse.append(avg)

plt.plot(total\_sse)

#plt.plot(range(1, K), total\_sse)

#plt.show()



I think the best K is when K=2, which is the point that breaks into elbow. The purpose of K-means clustering is to ensure that the value of sse is minimal. It is also necessary to calculate the time when the program is running. Therefore, the point where k=2, which is rapidly decreasing, is the best K value.