**CE/CZ 1104 Linear Algebra for Computing**

Lab 2

**Instructions:** There are 3 exercises in this lab with questions for each exercise.

Exercise 1: Computer Graphics – Linear Transformations

In the lectures, we have seen that matrices represent linear transformations such as scaling, translation, rotation and shear. The following table shows the transformations and the corresponding matrices when the input is pixel location is given by the vector . The 1 in the last row of the vector arises from the concept of ‘homogeneous co-ordinates’, which facilitate the transformations to be represented as matrices. You will study homogeneous co-ordinates in the computer graphics course.

|  |  |  |
| --- | --- | --- |
| Transformation type | Transformation matrix | Pixel mapping |
| Identity |  |  |
| Scaling |  |  |
| Rotation |  |  |
| Translation |  |  |
| Horizontal shear |  |  |
| Vertical shear |  |  |

We will implement the above transformations by applying them on the (x, y) co-ordinates of four points represented by the vectors , where the last component of each vector is an ascii character index.

Question 1:

Plot these points. Note the application of Identity transformation in the code.

**Sample code**

**import** **matplotlib.pyplot** **as** **plt**

**import** **numpy** **as** **np**

**import** **string**

*# points a, b and, c*

a, b, c, d = (0, 1, 0), (1, 0, 1), (0, -1, 2), (-1, 0, 3)

*# matrix with row vectors of points*

A = np.array([a, b, c, d])

*# 3x3 Identity transformation matrix*

I = np.eye(3) *#float*

color\_lut = 'rgbc' *#4 colors to represent 4 points*

fig = plt.figure()

ax = plt.gca()

xs = []

ys = []

**for** row **in** A:

output\_row = I @ row

x, y, i = output\_row

xs.append(x)

ys.append(y)

i = int(i) *# convert float to int for indexing*

c = color\_lut[i]

plt.scatter(x, y, color=c)

plt.text(x + 0.15, y, f"{string.ascii\_letters[i]}")

xs.append(xs[0])

ys.append(ys[0])

plt.plot(xs, ys, color="gray", linestyle='dotted')

ax.set\_xticks(np.arange(-2.5, 3, 0.5))

ax.set\_yticks(np.arange(-2.5, 3, 0.5))

plt.grid()

plt.show()

**Expected Output**

Chart, line chart

Description automatically generated

Question 2:

Modify the above code to implement and the display the results of the following transformations: (i) scaling transformation with scale of 2, (ii) rotation transformation with , (iii) translation, (iv) horizontal shear and vertical shear using your own parameters.

**Sample code (scaling)**

*# create the scaling transformation matrix*

T\_s = np.array([[2, 0, 0], [0, 2, 0], [0, 0, 1]])

fig = plt.figure()

ax = plt.gca()

xs\_s = []

ys\_s = []

**for** row **in** A:

output\_row = T\_s @ row

x, y, i = row

x\_s, y\_s, i\_s = output\_row

xs\_s.append(x\_s)

ys\_s.append(y\_s)

i, i\_s = int(i), int(i\_s) *# convert float to int for indexing*

*#c&c\_s are the same in this case, however its good to be explicit*

c, c\_s = color\_lut[i], color\_lut[i\_s]

plt.scatter(x, y, color=c)

plt.scatter(x\_s, y\_s, color=c\_s)

plt.text(x + 0.15, y, f"{string.ascii\_letters[int(i)]}")

plt.text(x\_s + 0.15, y\_s, f"{string.ascii\_letters[int(i\_s)]}'")

xs\_s.append(xs\_s[0])

ys\_s.append(ys\_s[0])

plt.plot(xs, ys, color="gray", linestyle='dotted')

plt.plot(xs\_s, ys\_s, color="gray", linestyle='dotted')

ax.set\_xticks(np.arange(-2.5, 3, 0.5))

ax.set\_yticks(np.arange(-2.5, 3, 0.5))

plt.grid()

plt.show()

**Expected Output**

**Chart, line chart

Description automatically generated**

**Sample code (rotating)**

*# create the rotation transformation matrix*

T\_r = np.array([[0, 1, 0], [-1, 0, 0], [0, 0, 1]])

fig = plt.figure()

ax = plt.gca()

**for** row **in** A:

output\_row = T\_r @ row

x\_r, y\_r, i\_r = output\_row

i\_r = int(i\_r)

c\_r = color\_lut[i\_r]

letter\_r = string.ascii\_letters[i\_r]

plt.scatter(x\_r, y\_r, color=c\_r)

plt.text(x\_r + 0.15, y\_r, f"{letter\_r}'")

plt.plot(xs, ys, color="gray", linestyle='dotted')

ax.set\_xticks(np.arange(-2.5, 3, 0.5))

ax.set\_yticks(np.arange(-2.5, 3, 0.5))

plt.grid()

plt.show()

**Expected Output**

**Chart, line chart

Description automatically generated**

**Sample code (**translating**)**

*#create the translation transformation matrix*

T\_t = np.array([[1, 0, 2], [0, 1, 0], [0, 0, 1]])

fig = plt.figure()

ax = plt.gca()

xs\_s = []

ys\_s = []

**for** row **in** A:

output\_row = T\_t @ row

x, y, i = row

x\_s, y\_s, i\_s = output\_row

xs\_s.append(x\_s)

ys\_s.append(y\_s)

i, i\_s = int(i), int(i\_s)

c, c\_s = color\_lut[i], color\_lut[i\_s]

plt.scatter(x, y, color=c)

plt.scatter(x\_s, y\_s, color=c\_s)

plt.text(x + 0.15, y, f"{string.ascii\_letters[int(i)]}")

plt.text(x\_s + 0.15, y\_s, f"{string.ascii\_letters[int(i\_s)]}'")

xs\_s.append(xs\_s[0])

ys\_s.append(ys\_s[0])

plt.plot(xs, ys, color="gray", linestyle='dotted')

plt.plot(xs\_s, ys\_s, color="gray", linestyle='dotted')

ax.set\_xticks(np.arange(-2, 10, 1))

ax.set\_yticks(np.arange(-2, 3, 1))

plt.grid()

plt.show()

**Expected Output**

Chart, line chart

Description automatically generated

**Sample code (horizontal shear)**

T\_h = np.array([[1, 0.5, 0], [0, 1, 0], [0, 0, 1]])

fig = plt.figure()

ax = plt.gca()

xs\_s = []

ys\_s = []

**for** row **in** A:

output\_row = T\_h @ row

x, y, i = row

x\_s, y\_s, i\_s = output\_row

xs\_s.append(x\_s)

ys\_s.append(y\_s)

i, i\_s = int(i), int(i\_s)

c, c\_s = color\_lut[i], color\_lut[i\_s]

plt.scatter(x, y, color=c)

plt.scatter(x\_s, y\_s, color=c\_s)

plt.text(x + 0.15, y, f"{string.ascii\_letters[int(i)]}")

plt.text(x\_s + 0.15, y\_s, f"{string.ascii\_letters[int(i\_s)]}'")

xs\_s.append(xs\_s[0])

ys\_s.append(ys\_s[0])

plt.plot(xs, ys, color="gray", linestyle='dotted')

plt.plot(xs\_s, ys\_s, color="gray", linestyle='dotted')

ax.set\_xticks(np.arange(-2.5, 3, 0.5))

ax.set\_yticks(np.arange(-2.5, 3, 0.5))

plt.grid()

plt.show()

**Expected Output**

Chart, line chart

Description automatically generated

Question 3:

Modify the code to implement a combination of the rotation and scaling transformations, i.e., a rotation followed by scaling. Note that since the transformations are linear, a combination of transformations is represented simply as a product of the matrices representing the individual transformation.

**Sample code**

*#combined tranformation matrix*

T = T\_s @ T\_r

fig = plt.figure()

ax = plt.gca()

xs\_comb = []

ys\_comb = []

**for** row **in** A:

output\_row = T @ row

x, y, i = row

x\_comb, y\_comb, i\_comb = output\_row

xs\_comb.append(x\_comb)

ys\_comb.append(y\_comb)

i, i\_comb = int(i), int(i\_comb)

c, c\_comb = color\_lut[i], color\_lut[i\_comb]

letter, letter\_comb = string.ascii\_letters[i], string.ascii\_letters[i\_comb]

plt.scatter(x, y, color=c)

plt.scatter(x\_comb, y\_comb, color=c\_comb)

plt.text(x + 0.15 , y, f"{letter}")

plt.text(x\_comb + 0.15, y\_comb, f"{letter\_comb}'")

xs\_comb.append(xs\_comb[0])

ys\_comb.append(ys\_comb[0])

plt.plot(xs, ys, color="gray", linestyle='dotted')

plt.plot(xs\_comb, ys\_comb, color="gray", linestyle='dotted')

ax.set\_xticks(np.arange(-2.5, 3, 0.5))

ax.set\_yticks(np.arange(-2.5, 3, 0.5))

plt.grid()

plt.show()

**Expected Output**

**Chart, line chart

Description automatically generated**

Exercise 2: Web Search – PageRank (not quite, but almost)

Google’s search algorithm to rank web pages, also called PageRank algorithm, is based on the following question: in a particular group of people, who is the most popular? Let us say there are 4 people – Alpha, Bravo, Charlie, Delta – who we will call A, B, C and D. Everyone in the group is asked to list their friends in this group. The outcome is as follows: A lists B and C; B lists A, C and D; C lists A, B and D; D lists A and C. This list can be represented as a 4 x 4 matrix as follows:

*A B C D*

Some people might list every one they ever met and others only list closest friends. This is compensated by normalizing the matrix, i.e., by dividing each list by the number of people in it to obtain the **linking matrix**:

*A B C D*

*L* = .

Next, we associate a non-negative number to each person that reflects that person’s **popularity** and collect all the numbers into a **popularity vector** . Let *a person's popularity be the weighted sum of the popularity of people who reference that person,* for example, . The equations can be written as

, which is the same as .

The above equation can be solved using reduced row echelon form by setting (arbitrarily) to get the values of .

Question 4:

Find the values of from the above data. Ensure that you use fractions in the linking matrix so that the sum of the columns in the matrix is 1.

PageRank essentially looks at a webpage *w* and determines which other web pages *w* links to, thus associating to each webpage *w* a linking matrix of 0's and 1's, and then normalizes it. Rest is just as we did in the popularity problem.

However, there are a few questions to be answered:

 (a) Does the equation  L**r** = **r** above always have a solution?

(b) Will a solution have entries that are nonnegative?

(c) Is the solution unique? If not, we will have conflicting rankings.

To the rescue comes the Perron-Frobenius Theorem:

For any matrix *L* having all entries nonnegative and each column summing to 1, the equation *L****r****=****r*** has a nonnegative solution ***r***.

**Sample code**

**import** **numpy** **as** **np**

*'''define function to transform a matrix to reduced row echelon form'''*

**def** rref(A):

tol = 1e-16

*#A = B.copy()*

rows, cols = A.shape

r = 0

pivots\_pos = []

row\_exchanges = np.arange(rows)

**for** c **in** range(cols):

*## Find the pivot row:*

pivot = np.argmax (np.abs (A[r:rows,c])) + r

m = np.abs(A[pivot, c])

**if** m <= tol:

*## Skip column c, making sure the approximately zero terms are*

*## actually zero.*

A[r:rows, c] = np.zeros(rows-r)

**else**:

*## keep track of bound variables*

pivots\_pos.append((r,c))

**if** pivot != r:

*## Swap current row and pivot row*

A[[pivot, r], c:cols] = A[[r, pivot], c:cols]

row\_exchanges[[pivot,r]] = row\_exchanges[[r,pivot]]

*## Normalize pivot row*

A[r, c:cols] = A[r, c:cols] / A[r, c];

*## Eliminate the current column*

v = A[r, c:cols]

*## Above (before row r):*

**if** r > 0:

ridx\_above = np.arange(r)

A[ridx\_above, c:cols] = A[ridx\_above, c:cols] - np.outer(v, A[ridx\_above, c]).T

*## Below (after row r):*

**if** r < rows-1:

ridx\_below = np.arange(r+1,rows)

A[ridx\_below, c:cols] = A[ridx\_below, c:cols] - np.outer(v, A[ridx\_below, c]).T

r += 1

**if** r == rows:

**break**;

**return** A

\*code continue on next page

*'''program starts here'''*

*# Define the linking matrix*

L = np.array([

[0,1/3,1/3,1/2],

[1/2,0,1/3,0],

[1/2,1/3,0,1/2],

[0,1/3,1/3,0]])

I = np.eye(4)

A = L-I

*#define the popularity matrix r*

r = np.zeros(4)

r[3] = 1 *#set rd = 1*

A[:,3] = A [:,3] \*-1

A\_rref = rref(A\*6) *#transform fractions into interger*

print("Reduced row echelon:**\n**", A\_rref)

r[0:3] = A\_rref[:,-1][0:3]

print("**\n**ra, rb, rc, rd are:",r)

**Expected Output**

Reduced row echelon:

[[1. 0. 0. 1.5 ]

[0. 1. 0. 1.3125]

[0. 0. 1. 1.6875]

[0. 0. 0. 0. ]]

ra, rb, rc, rd are: [1.5 1.3125 1.6875 1. ]

Question 5:

Suppose that we have five websites A, B, C, D, and E. Let's also suppose that the links between the various sites are given by the graph below.  The arrow pointing from C to D means that C's site links to D's, etc

C

A

D

E

B

* Create a linking matrix L which contains the information of which site links to which just as you did in the popularity example. Remember to normalize, and be sure that your input is exact (for example, make sure you enter 1/3 instead of .333 since our columns must sum to 1).
* Find all solutions **x** to the matrix equation (L - I)**x** = **0**.

**Sample code**

*# Define the linking matrix*

L = np.array([

[0,1/2,1/4,1,1/3],

[1/3,0,1/4,0,0],

[1/3,1/2,0,0,1/3],

[1/3,0,1/4,0,1/3],

[0,0,1/4,0,0]])

I = np.eye(5)

A = L-I

*#define the popularity matrix r*

r = np.zeros(5)

r[4] = 1 *#set re = 1*

A[:,4] = A [:,4] \*-1

A\_rref = rref(A\*12) *#transform fractions into interger*

r[0:4] = A\_rref[:,-1][0:4]

print("ra, rb, rc, rd, re are:",r)

**Expected Output**

ra, rb, rc, rd, re are: [6.33333333 3.11111111 4. 3.44444444 1. ]

Exercise 3: Epidemic Dynamics – SIR model

The SIR model is a simple mathematical description of the spread of a disease in a population. It divides the (fixed) population of *N* individuals into four "compartments" which may vary as a function of time, *t*:

* Susceptible *S(t)*:  not yet infected but can acquire the disease the next day;
* Infected *I(t)*:  have the disease;
* Recovered *R(t)*: had the disease, have recovered, and now have immunity to it.
* Deceased *D(t)*: had the disease, and unfortunately, died.

The above information could be put together in a normalized vector, e.g., .

Question 6:

Suppose the progression of the disease over each day is as follows:

* Among susceptible population
  + 5% acquires the disease
  + 95% remains susceptible
* Among infected population
  + 1% dies
  + 10% recovers with immunity
  + 4% recover without immunity (i.e., remain susceptible)
  + 85% remain infected
* 100% of immune and dead people remain in their state

Write a matrix *P* containing the above information. Using the state of the disease given by vector above, what is the state of the disease the next day? Note that .

**Sample code**

**import** **matplotlib.pyplot** **as** **plt**

fig = plt.figure()

ax = plt.gca()

P = np.array([

[0.95,0.04,0,0],

[0.05,0.85,0,0],

[0,0.1,1,0],

[0,0.01,0,1],

])

**Expected Output:** NA (continue with question 7)

Question 7:

Assume . Find the progression of the disease for each day from till . For each of the compartments – *S*, *I*, *R* and *D* – plot the progression from day 1 to day 200 on the same plot.

**Sample code**

x = np.array([[1],[0],[0],[0]])

**for** t **in** range(2,200):

x = np.dot(P,x)

plt.scatter(t,x[0], color="blue",marker=",",s=1)

plt.scatter(t,x[1], color="red",marker=",",s=1)

plt.scatter(t,x[2], color="green",marker=",",s=1)

plt.scatter(t,x[3], color="orange",marker=",",s=1)

plt.text(13, 0.9, "Susceptable",fontsize=11,color = "blue")

plt.text(25, 0.2, "Infected",fontsize= 11,color = "red")

plt.text(150, 0.8, "Recovered",fontsize= 11,color = "green")

plt.text(150, 0.17, "Deceased",fontsize= 11,color = "orange")

ax.set\_xlabel("Time")

ax.set\_ylabel("Xt")

plt.show()

**Expected Output**

A picture containing chart

Description automatically generated