3SK3 Project 2

Image Deblur Using LU decomposition

Jiaxian Wang

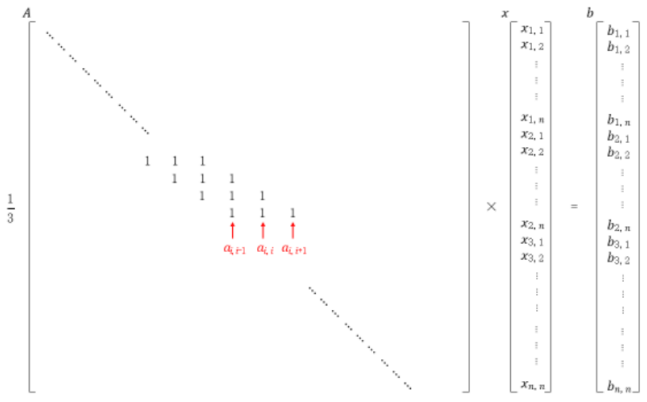
400113480

Starting with the Python platform, it is really confusing that it did not been deblurred on Python as it did on Matlab even though I accomplished the LU decomposition as the function of matrix inversion on both environments. I **strongly suggested** that the lab description declare those questions next time so that students would not waste time on choosing platforms.

In my case at this time, I am looking forward to getting replied on why the two platforms has the difference when I do the deblur. I will upload both LU decomposition and image blurring code in both Python and Matlab based on the algorithms with a little differences in order to improve the calculation performance, and I request consideration on partial bonus and more consideration on my marks for this project since the more effort I have payed.

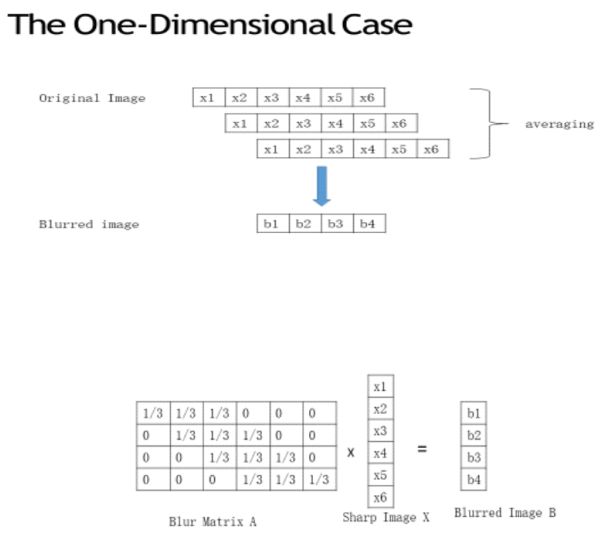
(a)

Starting with the Python code, the first step is blurring the 100 X 100 square image using given matrix A, which has two modes. The first one should be applied is the horizontal motion blurring matrix which is the longitudinal translation of a diagonal matrix of 1 with size of number of elements in original image to indicate the convolution result of the image kernel convolution, as the image below:



Horizontal motion blur

The principle of image blur:



1. Python code:

Image blur:

- The first step is that reading the image and grayscale it:

```
import cv2
import numpy as np
from fractions import Fraction

org_img = 'origin.jpg'
img_read = cv2.imread(org_img)
print(img_read.shape)

print(img_read)

# greyscale the image
img_gray = cv2.cvtColor(img_read, cv2.COLOR_BGR2GRAY)
H, W = img_gray.shape
print(img_gray)
print(img_gray.shape)
```

- Convert the 100 X 100 (or N X N in any case) matrix into a image vector which has the size of (100*100,1), or (N^2,1 in any case)

```
# reshape function (image matrix to vector)
img_vector = np.zeros((H * W, 1))
for i in range(H):
    for j in range(W):
        img_vector[j + i * W] = img_gray[i, j]
print(img_vector)
```

- Get the motion matrix A (N^2, N^2)

```
# Horizontal motion matrix
img_H_motion = np.zeros((H * W, H * W))
for i in range(H * W):
    for j in range(H * W):
        if (j - i == 0) or (j - i == 1) or (i - j == 1):
            img_H_motion[i, j] = 1
print(img_H_motion)
```

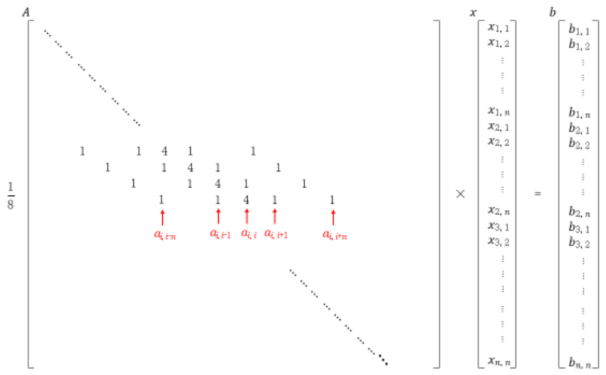
- Do multiplication

```
# multiplication
img_H_blurred_vector = np.zeros((H * W, 1))
img_H_blurred_vector = Fraction(1,3) * np.dot(img_H_motion, img_vector)

print(img_H_blurred_vector)
```

- After done the multiplication, we need to convert the (N^2,1) result matrix Ax = b

```
# convert back to matrix
img_blurred = np.zeros((H, W))
for i in range(H):
    for j in range(W):
        img_blurred[i, j] = img_H_blurred_vector[W * i + j, 0]

print(img_blurred)
cv2.imwrite('Horizontal_motion_blurred_image.jpg', img_blurred)
```

So far, the image blur has already completed and the next is to build the new out of focus image blur matrix, the principle is illustrated by the following image:



Out-of-focus blur

There is only one different step than previous one, which is the motion matrix A.

- Get the motion matrix

```
# focus lost blur matrix
img_FL_motion = np.zeros((H * W, H * W))
for i in range(H * W):
    for j in range(H * W):
        if i == j:
            img_FL_motion[i, j] = 4
        elif (abs(j - i) == 1) or (abs(j - i)) == W:
            img_FL_motion[i, j] = 1

print(img_FL_motion)
```

Result:

Original image (100*100):

Horizontal blurred image in Python (100*100):

Out of focus image in Python (100*100):

We can see the result is what is desired. Next, we need to design the algorithm finding original x matrix after linear process of blurring A*x = b, as we known the b and A.

Image deblur:

- get the image and convert the image to vector

```python
from fractions import Fraction
import cv2
import numpy as np
from copy import copy, deepcopy

img = 'Horizontal_motion_blurred_image.jpg'
blur_img = cv2.imread(img)
img_gray = cv2.cvtColor(blur_img, cv2.COLOR_BGR2GRAY)
print('origin_image = \n', img_gray)

H, W = img_gray.shape

img_vector = np.zeros((H * W, 1))
for i in range(H):
    for j in range(W):
        img_vector[j + i * W] = img_gray[i, j]
print('img vector = \n', img_vector)

```

- stimulated the motion matrix A and permutation matrix P

```python

# LU decomposition

# starting from gauss elimination for A to get U

# with permutation matrix P indicated the row swapping
P = np.eye(H * W, dtype=float)

# get motion matrix A
motion_matrix = np.zeros((H * W, H * W), dtype=float)
for i in range(H * W):
    for j in range(H * W):
        if (j - i == 0) or (j - i == 1) or (i - j == 1):
            motion_matrix[i, j] = 1
print('motion_matrix = \n', motion_matrix)

```

- gauss elimination of A to get U as the first step of LU decomposition

```python
# gauss elimination in A to get U
U = deepcopy(motion_matrix)
for i in range(H * W):
    for j in range(i + 1, H * W):
        # make sure U[i,i] != 0 and swap row in this case
        if U[i, i] == 0:
            # swap row function here with permutation matrix
            swap(U, i, P)
        if U[j, i] != 0:
            # do the subtraction
            fraction = float(U[j, i] / U[i, i])
            for k in range(H * W):
                # subtract the row
                U[j, k] = float(U[j, k]) - fraction * float(U[i, k])

print('U matrix = \n', U)
print('P matrix = \n', P)
P_times_A = np.dot(P, motion_matrix)
print('PA = UL = \n', P_times_A)
```

in this process we can get P matrix in the swapping function and record the rows swapped

The swap function:
swap the current row with the next row with non-zero element in the same column.

```python
def swap(u, r, p):  # check element U[i,i]
    temp = np.zeros((1, H * W), dtype=float)
    temp_p = np.zeros((1, H * W), dtype=float)
    counter = r + 1

    while True:
        if counter == H*W:
            break
        if u[counter, r] == 0:
            counter += 1
            continue
        else:
            break
    # swap U[i] with U[counter]
    for a in range(H * W):
        temp[0, a] = u[r, a]
        temp_p[0, a] = p[r, a]

        u[r, a] = u[counter, a]
        p[r, a] = p[counter, a]

        u[counter, a] = temp[0, a]
        p[counter, a] = temp_p[0, a]
    return u
```

- we need to get L matrix through LU = PA(known)

```python
# get matrix L through LU = PA
# build matrix L
# FORMULA METHOD
#  get matrix L through formula using the swapped A matrix
L_FORMULA = np.eye(H * W, dtype=float)

# gauss elimination without swapping and should get the same answer
U2 = deepcopy(P_times_A)


def get_l(upper, row):
    for r in range(row + 1, H * W):
        L_FORMULA[r, row] = upper[r, row] / upper[row, row]
    return L_FORMULA


for i in range(H * W):
    # Get L matrix here
    get_l(U2, i)
    for j in range(i + 1, H * W):
        # make sure U[i,i] != 0 and swap row in this case
        if U2[j, i] != 0:
            # do the subtraction
            fraction = float(U2[j, i] / U2[i, i])
            for k in range(H * W):
                # subtract the row
                U2[j, k] = U2[j, k] - fraction * U2[i, k]

print('U matrix using swapped PA = \n', U2)
print('L matrix (FORMULA METHOD) = \n', L_FORMULA)
```

After the swapped A matrix, we can directly get elements in L matrix one by one using formula

Comparing with $AX = b$, we have

$$LU = A \text{ and } Ld = b$$

where L is defined as a special lower triangular matrix carrying the elimination information as

$$L = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ \frac{a_{21}}{a_{11}} & 1 & 0 & \cdots & 0 \\ \frac{a_{31}}{a_{11}} & \frac{a'_{32}}{a'_{22}} & 1 & \cdots & 0 \\ \cdots & \cdots & & \ddots & \\ \frac{a_{n1}}{a_{11}} & \frac{a'_{n2}}{a'_{22}} & \cdots & & 1 \end{bmatrix}, \text{ or } l_{ij} = \begin{cases} 0, & i < j \\ 1, & i = j \\ \frac{a_{ij}^{(j-1)}}{a_{jj}^{(j-1)}}, & i > j \end{cases}$$

The other method using calculation of LU (known) = PA (known) will be explained and performed by the **test.py** file in uploaded code file.

- Get D matrix

```
# get swapped D matrix

print('result matrix = \n', img_vector)
B_swapped = np.dot(P, img_vector)
print('swapped result matrix B = \n', B_swapped)

# we know that L*D = B_swapped
D = np.zeros((H * W, 1), dtype=float)


def d_sum(row, lower):
    sum_d = 0.0
    for r in range(row):
        sum_d += lower[row, r] * D[r, 0]
    return sum_d


for i in range(H * W):
    D[i, 0] = B_swapped[i, 0] - d_sum(i, L_FORMULA)

print('matrix D = \n', D)
```

The algorithm in here is that using LD = BP, and the format is D (calculated currently) + sum (known) = B(used)

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -1/2 & 1 & 0 & 0 \\ -5/2 & -4/3 & 1 & 0 \\ 3/2 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ d_4 \end{bmatrix} = \begin{bmatrix} 2 \\ -26 \\ 7 \\ -18 \end{bmatrix}$$

Ex.

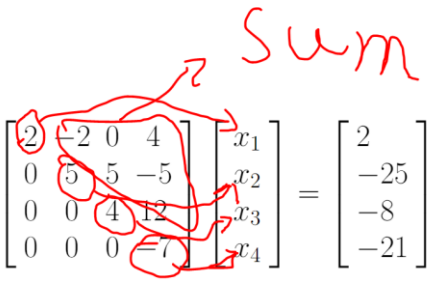
- The same algorithm can also be implemented to get X matrix in final step.

```
# get X matrix sloven
# using UX = D

X = np.zeros((H * W, 1), dtype=float)


def x_sum(row, upper):
    sum_x = 0.0
    for r in range(H * W - row - 1):
        sum_x += upper[row, H * W - 1 - r] * X[H * W - 1 - r, 0]
    return sum_x


for i in range(H * W - 1, -1, -1):
    X[i, 0] = (D[i, 0] - x_sum(i, U2)) / U2[i, i]

print('matrix X = \n', X)
```

The algorithm in here is that using UX = D, and the format is X (calculated currently) + sum (known) = D(used)

Ex.

$$UX = d, X = ?$$



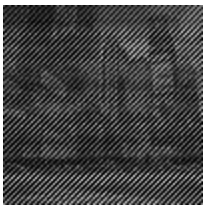
- After the X is calculated, we need to transfer the X vector matrix to the square matrix to form the proper image.

```
# convert back to matrix
img_Deblurred = np.zeros((H, W), dtype=float)
for i in range(H):
    for j in range(W):
        img_Deblurred[i, j] = X[W * i + j, 0]
        if img_Deblurred[i, j] > 255:
            img_Deblurred[i, j] = 255
        elif img_Deblurred[i, j] < 0:
            img_Deblurred[i, j] = 0

print('recovered Horizontal motion image = \n', img_Deblurred)

cv2.imwrite('Horizontal_motion_Deblurred_image.jpg', img_Deblurred)
```

Result:

Original image (100*100):

Horizontal deblurred image in Python (100*100):



Out of focus deblurred image in Python (100*100):

It can be observed that deblurred image is not what we desired comparing to original image and we need to verify them.

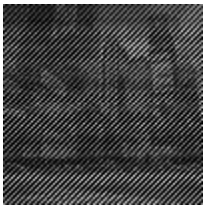
One of the simple methods is that we use the deblurred images solved by our algorithm compare to the **built-in invert matrix function** inside python library.

By setting X = inverse(A) * b, the image outputted should be the proper deblur image solved by the computer.

```
# get motion matrix A
motion_matrix = np.zeros((H * W, H * W), dtype=float)
for i in range(H * W):
    for j in range(H * W):
        if (j - i == 0) or (j - i == 1) or (i - j == 1):
            motion_matrix[i, j] = 1
print('motion_matrix = \n', motion_matrix)

X = np.dot(inv(motion_matrix), img_vector)

# convert back to matrix
img_Deblurred = np.zeros((H, W), dtype=float)
for i in range(H):
    for j in range(W):
        img_Deblurred[i, j] = X[W * i + j, 0]
        if img_Deblurred[i, j] > 255:
            img_Deblurred[i, j] = 255
        elif img_Deblurred[i, j] < 0:
            img_Deblurred[i, j] = 0
```

Result:

Horizontal deblurred image in Python (built-in function) (100*100):



Out of focus deblurred image in Python (built-in function) (100*100):

By verifying, it proofs that my algorithm successfully performs the function of inverse matrix using LU decomposition.

It is really confusing that why Python can not recover the image even if my algorithm is approachable.

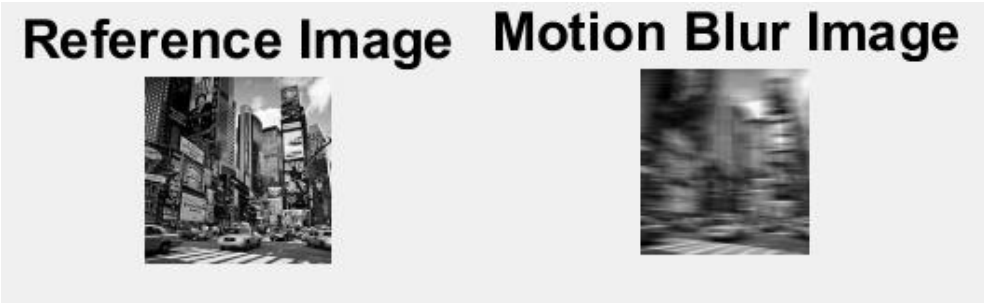
To do more verification, I change the environment to Matlab as I mentioned at the beginning of the report and the process is following.

2. Matlab

The same algorithm here and the code are basically the same with some performance improvement.

- First, based on the given sample code, the blurred image can be found with an boundary method.

  Result:



- It is different with the blur method illustrated on the project description. However, the algorithm still able to be implemented to recover the image.

```
% linear equations: Ax = b
A = [motion_matrix; boundary_matrix];
b = [reshape(img_motion, [], 1); boundary_vector];
tic
img_deblur = reshape(Solving_Linear_Equations_with_LU_decomposition(A, b), size(img_ori));
toc
```

- Use the LU decomposition function here known A and b matrix.
- The first step is to declare the required variables

```
function x = Solving_Linear_Equations_with_LU_decomposition(A, b)
% write your code here
% the output x should be inv(A)*b (or A\b), but you CANNOT use it
% you CANNOT use any high-level function in your code, for exampl
% function here function here function here function here functio
N = length(A);

% LU decomposition
% strating from gauss elimination for A to get U
% with permutation matrix P indicated the row swapping
P = eye(N);
% L matrix
L = eye(N);
% B matrix swapped
B_swapped = b;
% A matrix swapped, PA = A * Permutation
PA = A;
% gauss elimination in A to get U
U = A;
% swapped flag
swapped_flag = 0;
```

- After declaring the variables, we need to get the U and P matrix in the same loop

```
for i = 1:N
    if U(i,i) == 0
        %make sure U[i,i] != 0 and swap row in this case
        for j = i+1:N
            if U(j,i) ~= 0
                % indicated swapped
                swapped_flag = 1;
                % swap row function here with permutation matrix
                temp_U = U(i,:);
                temp_P = P(i,:);


                U(i,:) = U(j,:);
                P(i,:) = P(j,:);


                U(j,:) = temp_U;
                P(j,:) = temp_P;
                break
            end
        end
    end
```

- Get L matrix using direct formula same as the method in Python

```
for column = i+1:N
    if U(column,i) ~= 0
        % do subtraction here calculate fraction
        fraction = U(column,i) / U(i,i);

        % subtracte the row and get L through swapped matrix PA = LU
        L(column,i) = fraction;
        for k = 1:N
            U(column,k) = U(column,k) - fraction * U(i,k);
        end
    end
end
```

- Get A matrix swapped, and B matrix swapped

```
PA = P*A;
B_swapped = P*B_swapped;
```

- Recursion here is used to improve the speed of calculation and efficiency.
  For each swapped A and corresponding b, we need to solve the new x vector for them.
  As the trade off, the memory demand is higher than the Python method, but the speed is faster.

In order to deal with that, we need to clear specific variables for new values to replace the trash values in memory every time we need to do the recursion.

```
if swapped_flag == 1
    % CLEAN UP FOR MEMORY
    U = 0;L = 0;
    P = 0;X = 0;
    x = Solving_Linear_Equations_with_LU_decomposition(PA, B_swapped);
else
```

- The same method to get D and X matrix as Python code

```
% get swapped D matrix
D = zeros(N,1);
% since we know that L*D = B_swapped
for i = 1:N
    D(i) = B_swapped(i) - d_sum(i,L,D);
end
```

```
function sum = d_sum(row,lower,D)
sum_d = 0;
for r = 1:(row-1)
    sum_d = sum_d + lower(row,r)*D(r);
end
sum = sum_d;
end
```

```
% get X matrix sloven
% using UX = D
X = zeros(N,1);

for i = N:-1:1
    X(i) = (D(i) - x_sum(i,U,X,N)) / U(i,i);
end
```

```
function sum = x_sum(row,upper,X,N)
sum_x = 0;
for r = 1:(N-row)
    sum_x = sum_x + upper(row,N-r+1) * X(N-r+1);
end
sum = sum_x;
end
```

Result:

The recovered image in Matlab is the same as the original image, which indicates that the LU decomposition algorithm is approachable.

As the horizontal motion and out of focus blur matrixes are required to write by our own, the same blur methods as Python code does.

```
% out of focus motion matrix
fprintf('Sloving A matrix......\n');
for i = 1:n
    if i+1 < n
        A(i,i+1) = 1/3;
    end

    if i-1 > 0
        A(i,i-1) = 1/3;
    end
    A(i,i) = 1/3;
end
fprintf('sucess!!!\n');
```

- Horizontal blur matrix using single loop for busting up speed

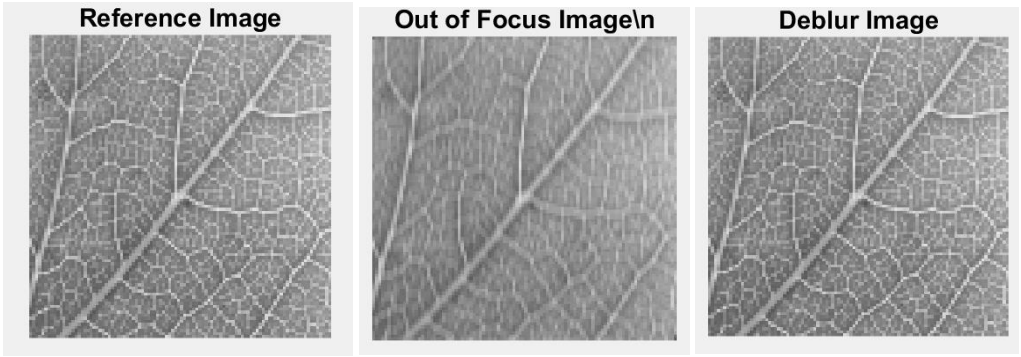
```
% out of focus motion matrix
fprintf('Sloving A matrix......\n');
for i = 1:n
    if (i > row) && (i <= n-row)
        A(i,i+1) = 1/8;A(i,i-1) = 1/8;
        A(i,i) = 1/2;
        A(i,i-row) = 1/8; A(1,1+row) = 1/8;
    end
end
fprintf('sucess!!!\n');
```

- Out of focus blur matrix using single loop for busting up speed

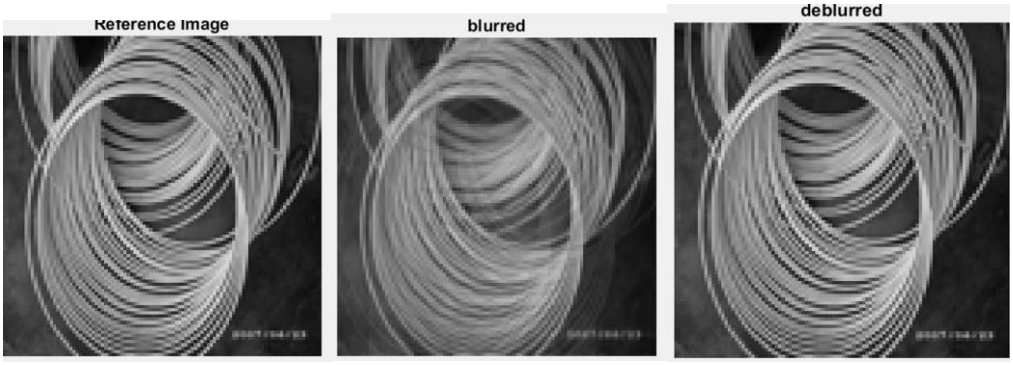
For clearance and easy for observation, the image can be changed.

Result:

Out of focus blur:



Horizontal blur:



(b)

Complexity:

	Python	Matlab
Blur	N^2	N
Get U	N^3	N^2
Get P	0	0
Get L	N^3	N^2
Get D	N^2	N^2
Get X	N^2	N^2

The algorithm is faster and efficient enough on Matlab and can fully recover the image, the run time is varying depend on the computers, can run from 80s-200s as monitored.

(c)

The inside principle of image blurring is that the image kernel does convolution with specific image and can be transformed as the equivalent effect of motion matrix A multiply with the image vector.

If there is some small error in the image kernel, the more the error far away from the desired performance, the more weight of the error are taken from the motion matrix or image kernel, the more effect will be observed in the result.

We can verify that by simply change the motion matrix A inside the loop, changing the if condition to decide where the error can be or change the magnitude of the to see the consequences.