3SK3 Lab3

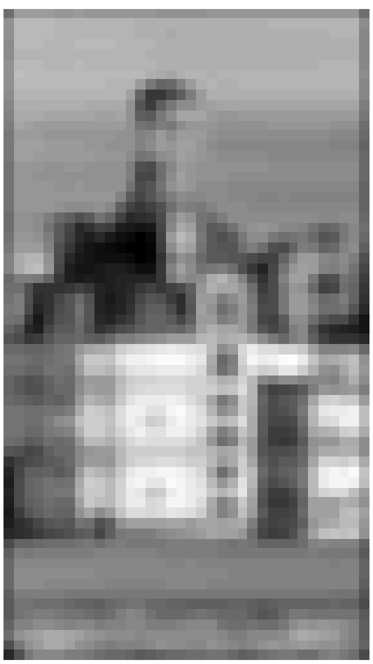
Zhaobo Wang
400188525

a) Solve the image deblurring problem by the LU decomposition method for a given blurring operator A, including both motion blur and out of focus blur.

Blurred Image of cheetah

Deblurred Image of cheetah

Blurred castle



Deblurred castle

Blurred butterfly

Deblurred butterfly

b) Make your program as efficient as possible. What is the number of operations needed to deblur an image?

First I define constructing the convolution matrix, this step involves filling up an m*n by m*n matrix, where m*n is the number of pixels in the image, convolute with kernel. this step would involve O(m*n*kernel_row*kernel*column) operations. LU decomposition algorithm involves find_max_pivot_row and swap_row which simplifies to O(m^3*n^3), Once using L and U, solving Ly = b, Ux = y to O(m^2*n^2)
The overall computational complexity is the sum of these operations:
O(m^2n^2) + O(m^2*n^2) + O(m^3*n^3), m is constant, so we can make m coefficients out.

The dominating term is O(n^3)

So the number of operations needed to deblur an image is O(n^3).

c) Find and understand what happens if there are small errors in the blur kernel matrix A.

Small errors in the blur kernel matrix A will significantly affect the outcome of the deblurring process, If A has small errors, these can be amplified during the inversion or decomposition process, it will lead to inaccuracies in the final deblurred image. For example, LU decomposition involves dividing by elements of A. If A contains errors, this can lead to significant numerical instability. It will also make large errors in the computed L and U matrices.

d) Submit a written project report together with the well-documented software code. The written report must justify your algorithm design, include a detailed complexity analysis, and discuss the effects of errors in the blur kernel matrix A.

Algorithm Design:

Load data to get kernel and given_image
First construct a convolution matrix A
Deblurred function including LU decomposition
Get deblurred image in the end
Well defined Document below

Document Software Code:

```
function project3_code
   % load data
   data = load('castle.mat');

   % load kernel and given image
   mykernel = data.kernel_weights;
   given_image = data.blurred_image;
   % given image size m/n
   [m, n] = size(given_image);
   A = construct_Convolution_Matrix(mykernel, m, n);
   given_image = double(given_image);    % double precision
   my_deblurred_image = my_deblurImage(A, given_image(:), m, n);% this is my
deblurred image function
   figure;
   subplot(1, 2, 1);
   imshow(given_image, []); title('Blurred castle');
   subplot(1, 2, 2);
   imshow(my_deblurred_image, []); title('Deblurred castle');
    % Display two image
end

function A = construct_Convolution_Matrix(blur_kernel, m, n) %define convolution
matrix function
   A = sparse(m*n, m*n); % make A matrix for sparse
   Center_of_Kernel = floor((size(blur_kernel) + 1) / 2); %find the center of the
kernel
   for i = 1:m % loop for the row
       for j = 1:n % loop for the column
           my_index = (j-1)*m + i; % find the index
           for p = 1:size(blur_kernel, 1) % loop kernel row
               for q = 1:size(blur_kernel, 2) % loop kernel column
                   rowOffset = p - Center_of_Kernel(1); % offset
```

```
                    colOffset = q - Center_of_Kernel(2);
                    newRow = i + rowOffset;% new Row is equal to i pos plus offset
                    newCol = j + colOffset;% new Column is equal to j pos plus offset
                    if newRow >= 1 && newRow <= m && newCol >= 1 && newCol <= n % if
new Row and new Column is in the range
                        idx_neighbor = (newCol-1)*m + newRow; % next index is defined
                        A(my_index, idx_neighbor) = blur_kernel(p, q);% A each point
defined
                    end
                end
            end
        end
   end
   A = sparse(A);% using sparse of A
end
function deblurred_image = my_deblurImage(A, blurred_image_v, m, n)
   A_inverse = Lu_inverse(A); % define a inverse by myself
   deblurred_image_v = A_inverse * blurred_image_v; % A inverse * blurred image get
deblurred image
   deblurred_image = reshape(deblurred_image_v, m, n); % reshape the image
end
function A_inverse = Lu_inverse(A) % lu inverse function
   [L, U] = myLU(A); % call lu
   A_inverse = zeros(size(A)); % make a matrix contains many zero elements
   n = size(A, 1);
   for i = 1:n
       b = zeros(n, 1); % each row of b
       b(i) = 1;
       d = sub_forward(L, b); %call sub_forward
       x = sub_backward(U, d); %call sub_backward
       A_inverse(:, i) = x;% A inverse
   end
end
function x = sub_forward(L, b)% define forward substitution function
   n = length(b);
   x = zeros(n, 1);
   for i = 1:n% i from 1 to n in forward
       x(i) = (b(i) - L(i, 1:i-1) * x(1:i-1)) / L(i, i);% get x pos when forward
substitution
   end
end
function x = sub_backward(U, b) % define backward substitution function
   n = length(b);
   x = zeros(n, 1); % x is array when x elements
   for i = n:-1:1 % i from n to 1 to n in backward
       x(i) = (b(i) - U(i, i+1:end) * x(i+1:end)) / U(i, i); % get x pos when forward
substitution
   end
```

```
end
function [L, U] = myLU(A) %define my own lu function
   n = size(A, 1);
   L = eye(n);
   U = zeros(n);
   for j = 1:n % j for loop
       for i = 1:j % i for loop
           U(i, j) = A(i, j) - L(i, 1:i-1) * U(1:i-1, j); %my lu get u decomposition
       end
       for i = j+1:n
           L(i, j) = (A(i, j) - L(i, 1:j-1) * U(1:j-1, j)) / U(j, j); %my lu get l
decomposition
       end
   end
end
```