CMSC 828D: Fundamentals of Computer Vision Homework 3

Instructors : Larry Davis, Ramani Duraiswami,
Daniel DeMenthon, and Yiannis Aloimonos

Solution based on homework submitted by Haiying Liu

1. **Show that in a properly focused imaging system the distance *f'* from the lens to the image plane is equal to (1+*m*) *f*, where *f* is the focal length and m is the magnification. This distance is called the effective focal length. Show that the distance between the image plane and an object must be (*m* + 2 + 1/*m*) *f*.**

Solution: The properly focused imaging system is shown in [Figure 1].

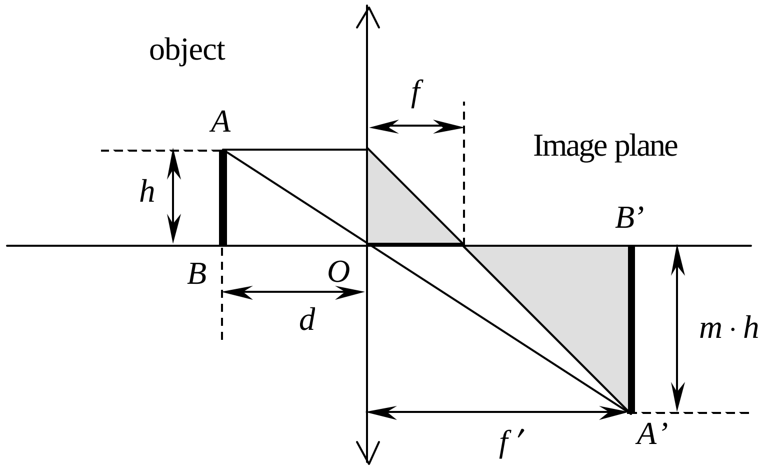

Figure 1: Properly focused imaging system

Suppose the height of the object AB is h. Given that the magnification is m, the height of its image A'B' is mh. The two grayed triangles are similar. Thus, we have:

$$\frac{f'-f}{mh} = \frac{f}{h} = \frac{f'-f+f}{mh+h} = \frac{f'}{h(m+1)}$$
$$\Rightarrow f' = (1+m)f$$

Because the object AB is parallel to the image plane *A'B'*, the triangle *OAB* is similar to the triangle *OA'B'*. Therefore, we have:

$$\frac{d}{h} = \frac{f'}{mh} = \frac{d+f'}{h(m+1)} \Rightarrow d + f' = \frac{f'(m+1)}{m}$$

Substitute f' in the above equation by $(1+m)f$,

$$BB' = d + f' = \frac{(m+1)^2 f}{m}$$
$$= \left(m + 2 + \frac{1}{m} \right) f$$

For unit magnification, i.e. $m = 1$, we have

$$d = \left. \frac{f'}{m} \right|_{m=1} = f' = (1+m) f \big|_{m=1} = 2f$$

The object must be $2f$ far from the lens for unit magnification.

2. What shapes can the perspective image of a sphere have? …

Solution: A point of the sphere has an image on the contour of the image of the sphere if the *line of sight* for this point is tangent to the shape of the object. The locus of the lines of sight tangent to a sphere is a *cone of revolution*. The intersection of this cone of revolution with the image plane is the contour of the image of the sphere. The intersection of a cone and a plane is called a conic. If the sphere is one side of the image plane and the center of projection is on the other side, this conic cannot be a parabola or a hyperbola, and can only be an *ellipse* (a circle being just a special ellipse obtained when the image plane cuts the cone at a 90 degree angle to the cone axis).

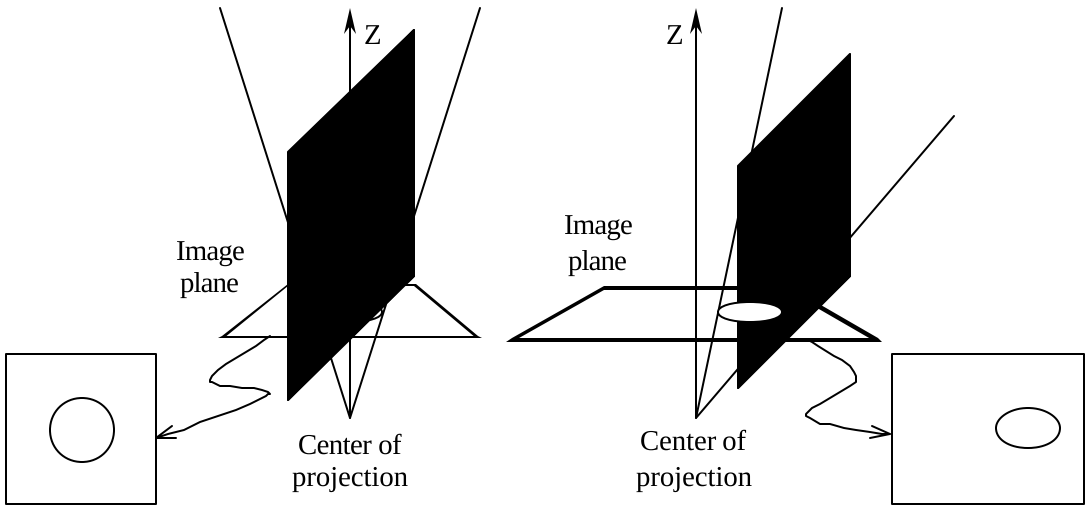

Figure 2: Perspective image of a sphere.

3. Perspective effects can be significant when a wide-angle lens is used, while images obtained with a telephoto lens tends to approximate orthographic projection. Explain why these are only rough rules of thumb.

Solution: The wide-angle lens is a lens with small focus length. The telephoto lens on the other hand has a very long focal length. Therefore, for a given film size or a given CCD chip size, a camera has a field of view with a large angle with a wide-angle lens, can see a large chunk of the world and can see even large objects. With a telephoto lens, a camera has a field of view with a small angle; all the lines of sight are almost parallel to the optical axis. Lines from scene points to the center of projection at large angles from the optical axis don't hit the camera film and large objects close to the camera don't fit in the image.

How can we quantify perspective effects? Consider a road straight in front of you, with telephone poles of height H along the road at intervals D from each other. The length of a phone pole at a distance Z from the camera in the image is f H/Z, and the length of the next one is f H/(Z+D). The ratio of the lengths in the image of the two phone poles is 1+D/Z. D/Z represents the perspective distortion for the two phone poles: their length ratio is 1 in the real world but is "distorted" by D/Z in the image world. The closer the poles are, the smaller Z is, and the larger the distortion. Note that this distortion is independent of the focal length. This is not surprising, since by looking at the equations of perspective projection, we see that the focal length is only a scaling factor, and changing the focal length only changes the scale of things in the image. But with a wide-angle lens the phone poles will fit in the image even if they are close to the camera, so we will be able to take pictures even when there are large perspective distortions. With a telephoto lens only far away phone poles will fit in the image. The rules of thumb above aply if, with each type of lens, we try to position ourselves with respect to an object so that the object always fills out most of the image (which is good photographic practice). If on the other hand we have two phone poles at 100 m that just fit in the image with a telephoto lens, switch to a wide angle lens and don't move at all, we will get an image with the same persective distortion (but at a different scale, with much smaller phone poles).

3. **You see a first image of a cube with a pinhole camera with focal length *f*. Then you rotate the camera around its center of projection to obtain new images.**

(a) Do the lines of sight change? Can you get any new information about the cube by this camera motion? Can you see a new facet that you were not seeing in the first image?

Solution: In a rotation around the center of projection, the center of projection does not change, and the scene of course does not change, so lines from the center of projection to the scene (the lines of sight) do not change. All the information about the cube is contained in the lines of sight. If we are given an image of a scene, we can compute what the scene looks like after such a rotation without additional knowledge from the scene, as illustrated by The sight line of a visible point in space, say *P*, will pass through the center of projection. The correspondent image point (the *p* in the first image and *p'* in the second image) is the intersection point of the sight line and the image plane. If a point in space is occluded in the first image, it will still be occluded after this rotation, since the center of projection remains unchanged. This is illustrated in [Figure 3]. This also explains why we cannot see any new facet that we do not see in the first image after this rotation.

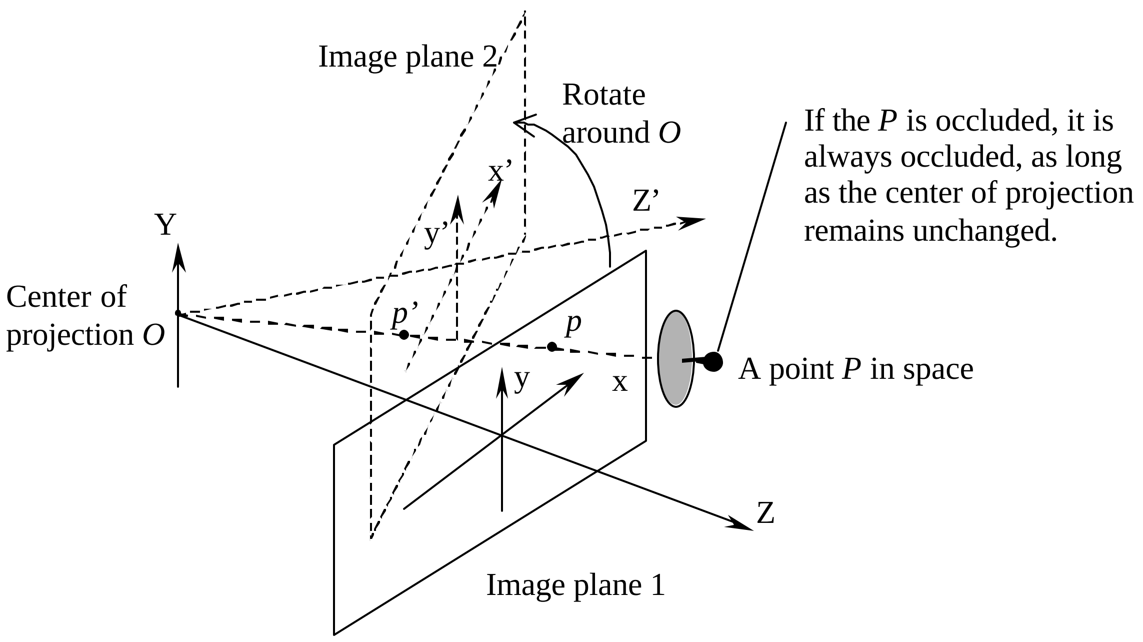

Figure 3: Rotate the camera around its center of projection to get new images.

(b) Assume that a vertex V of the cube is seen as a point with coordinates (x, y) in the first image. The camera is then rotated around a rotation axis going through the center of projection and parallel to the y axis of the image plane, so that the unit vector k perpendicular to the image plane becomes a new vector k'. The unit vector of the y axis is a known vector j. What are the new coordinates of the image of V after the rotation, as functions of x, y, j, k and k'.

Solution: Denote the world coordinate originated at the center of projection C by (X,Y,Z) with unit vectors **i**, **j** and **k** before the rotation, and (X',Y',Z') with unit vectors **i'**, **j'=j** and **k'** after the rotation. Denote the image of the vertex V as a point m with image coordinates (x,y,f) before the rotation, and the image of V as a point m' with coordinates (x',y',f) after the rotation. We have: **Cm** = x **i** + y **j** + f **k, Cm'** = x' **i'** + y' **j** + f **k'**

Since m and m' are the intersection of the same line by two different planes, Cm and Cm' are aligned, and we can write **Cm'** = l **Cm**. Also note that the dot product of Cm' with the vector k' is the z-coordinate of m', which is equal to f. We use that fact to compute l. We obtain **Cm'.k** = l **Cm.k'** = f so that l = f / **Cm.k'**=*f/(*x **i.k'**+ f **k.k')**

Cm' = l **Cm** becomes x' **i'** + y' **j** + f **k'** = f (x **i** + y **j** + f **k)** */(*x **i.k'**+ f **k.k')**

Then one obtains x' by taking the dot product of this equation with **i',** and similarly one obtains y' by multiplying the equation with **j.** We also replace **i** and **i'** by the cross-products **j**×**k** and **j**×**k'.** Also, we use the fact that (**j**×**k**)**. k'** is equal to **j.**(**k**×**k'**)**.**

Therefore,

$$\begin{cases} x' = f\dfrac{x\langle \mathbf{i},\mathbf{i}'\rangle + f\langle \mathbf{k},\mathbf{i}'\rangle}{x\langle \mathbf{i},\mathbf{k}'\rangle + f\langle \mathbf{k},\mathbf{k}'\rangle} = f\dfrac{x\langle \mathbf{k},\mathbf{k}'\rangle - f\ \mathbf{j}.(\mathbf{k}\times\mathbf{k}')}{x\,\mathbf{j}.(\mathbf{k}\times\mathbf{k}') + f\langle \mathbf{k},\mathbf{k}'\rangle} \\ y' = f\dfrac{y}{x\,\mathbf{j}.(\mathbf{k}\times\mathbf{k}') + f\langle \mathbf{k},\mathbf{k}'\rangle} \end{cases}.$$

4. **Using Matlab, and the result of problem 4, create the image you would obtain if the camera that took the image 'flowers.tif' (converted to grayscale) were to be rotated by 10 degrees to the left around its center of projection. (Set the pixels that were not visible in the first image to white. Set the focal length to 250 pixels.**

   Solution: The original image and the image after the camera rotation is shown in [Figure 4]. The Matlab code is listed in Appendix. Linear sub-pixel interpolation is used to get more accurate intensity value. It is assumed that the origin of image coordinates is at the center of the image.

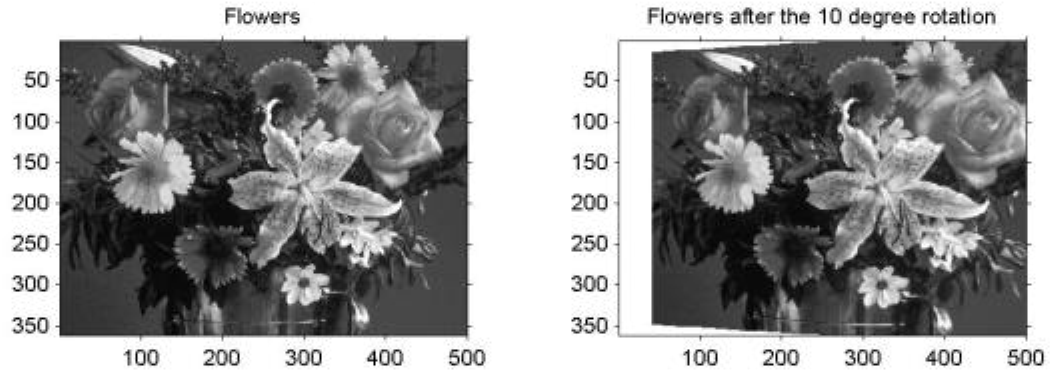

Figure 4: Camera rotate 10° to the left around the camera's center of projection.

5. **Estimate the number of operations …**

   Solution: According to the recursive algorithm (determinant expansion by minors) explained in class, for an $N \times N$ matrix $\mathbf{M} = \lfloor m_{ij} \rfloor$, its determinant can be calculated by:

$$|\mathbf{M}| = \sum_{i=1}^{N}\sum_{j=1}^{N}(-1)^{i+j} m_{ij} |\mathbf{M}_{ij}|$$

   where $\mathbf{M}_{ij}$ is a matrix formed by eliminating row i and column j from $\mathbf{M}$. The above equation has N^2 multiplications and $N^2 - 1$ additions. Recursively using the equation, for each $\mathbf{M}_{ij}$, there are $(N-1)^2$ multiplication and $(N-1)^2 - 1$ addition, and so on. Therefore, there are totally $N^2 \cdot (N-1)^2 \cdot \ldots \cdot 1 = (N!)^2$ multiplications and $(N^2-1)\cdot((N-1)^2-1)\cdot\ldots = O((N!)^2)$ additions. It is much bigger than an algorithm that requires only N^3.

6. **In Matlab, explore the following matrix decompositions and …**

   Solution: The mathematical definition of the decompositions are listed in the following table:

Decomposition	Definition
LU	A procedure for decomposing an $N \times N$ matrix $\mathbf{A}$ into a product of a lower triangular matrix $\mathbf{L}$ and an upper triangular matrix $\mathbf{U}$, such that $\mathbf{A} = \mathbf{LU}$. It is useful in solving linear equations.
Cholesky	Given a symmetric positive definite matrix $\mathbf{A}$, the Cholesky decomposition is an upper triangular matrix $\mathbf{U}$ such that $\mathbf{A} = \mathbf{U}^{\mathrm{T}}\mathbf{U}$. It is useful in solving linear equations.
Eigenvalue	Let *A* be a linear transformation represented by a matrix $\mathbf{A}$. If there is a vector $\mathbf{X} \in \mathbf{R}^n \neq 0$ such that $\mathbf{AX} = l\mathbf{X}$ for some scalar l, then l is called the eigenvalue of $\mathbf{A}$ with corresponding (right) eigenvector $\mathbf{X}$. It reflects the "character" of the matrix.
QR	Given a matrix $\mathbf{A}$, its *QR*-decomposition is of the form $\mathbf{A} = \mathbf{QR}$, where $\mathbf{R}$ is an upper triangular matrix and $\mathbf{Q}$ is an orthogonal matrix, i.e., one satisfying $\mathbf{Q}^{\mathrm{T}}\mathbf{Q} = \mathbf{I}$, where $\mathbf{I}$ is the identity matrix. This matrix decomposition can be used to solve linear systems of equations.
SVD	A decomposition of a matrix $\mathbf{A}$ into the form $\mathbf{A} = \mathbf{U}^*\mathbf{DV}$, where $\mathbf{U}$ is a unitary matrix (i.e. $\mathbf{U}^* = \mathbf{U}^{-1}$), $\mathbf{U}^*$ is its adjoint matrix, and $\mathbf{D}$ is a diagonal matrix whose elements are the singular values of the original matrix. It is also useful for solving linear equations.

The script of examples and results are listed in Appendix.

7. The eigenvalues of a matrix A **can be …**

Solution: Assume $\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$, we have

$$
\begin{aligned}
&|\mathbf{A} - l\,\mathbf{I}| = 0 \\
&\Rightarrow \left| \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} - \begin{bmatrix} l & 0 \\ 0 & l \end{bmatrix} \right| = 0 \\
&\Rightarrow \begin{vmatrix} a_{11} - l & a_{12} \\ a_{21} & a_{22} - l \end{vmatrix} = 0 \\
&\Rightarrow (a_{11} - l)(a_{22} - l) - a_{12}a_{21} = 0 \\
&\Rightarrow l^2 - (a_{11} + a_{22})\,l + (a_{11}a_{22} - a_{12}a_{21}) = 0 \\
&\Rightarrow \begin{cases} l_1 = \dfrac{a_{11} + a_{22} + \sqrt{(a_{11} + a_{22})^2 - 4(a_{11}a_{22} - a_{12}a_{21})}}{2} \\ l_2 = \dfrac{a_{11} + a_{22} - \sqrt{(a_{11} + a_{22})^2 - 4(a_{11}a_{22} - a_{12}a_{21})}}{2} \end{cases}
\end{aligned}
$$

The Matlab function that returns eigenvalues of any 2 by 2 matrix is listed in Appendix, followed by an example.

8. **Read chapter 2.0 through chapter 2.3 of Numerical recipes…**

   Solution: Done.

Appendix:

- **Question 5**

```
function hw3_5
%
% CMSC 828D: Fundamentals of Computer Vision
%                       Homework3
%
% Instructors : Larry Davis, Ramani Duraiswami,
%               Daniel DeMenthon, and Yiannis Aloimonos
% Student     : Haiying Liu
%
% Date        : Sept. 16, 2000
%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%=============================================================================
%= Qustion no.5

% Read image from file.
image = imread('flowers.tif');

% Convert the color image into gray scale image.
image = rgb2gray(image);

% Convert the data type from uint8 to double.
image = double(image);

% Rotate.
% ..Initialize parameters.
f           = 250;
theta       = (-10) * pi / 360;
cos_theta   = cos(theta);
sin_theta   = sin(theta);

image_2     = zeros(size(image));

% Get image size.
nRow        = size(image, 1);
nCol        = size(image, 2);

half_nRow   = nRow ./ 2;
half_nCol   = nCol ./ 2;

% The origin of matrix is at upper left size
% while the origin of the image is at center.
for row2 = 1:nRow
  for col2 = 1:nCol
    yy = row2 - half_nRow;
    xx = col2 - half_nCol;

    x = (xx * f * cos_theta + f * f * sin_theta) / ...
      (f * cos_theta - xx * sin_theta);
    y = (yy * x * sin_theta + yy * f * cos_theta) / f;

    row1 = y + half_nRow;
    col1 = x + half_nCol;

    if 1 <= col1 & col1 <= nCol & 1 <= row1 & row1 <= nRow
      % ..When inside the image 1,
      % ..linearly interpolate to get sub_pixel gray level.
      up    = floor(row1);
      down  = up + 1;
      left  = floor(col1);
      right = left + 1;

      if down <= nRow & right <= nCol
        intensity_1    = image(up, left);
        intensity_2    = image(down, left);
        leftIntensity  = (row1 - up) * ...
          (intensity_2 - intensity_1) + intensity_1;

        intensity_1    = image(up, right);
        intensity_2    = image(down, right);
        rightIntensity = (row1 - up) * ...
          (intensity_2 - intensity_1) + intensity_1;

        intensity      = (col1 - left) * ...
          (rightIntensity - leftIntensity) + leftIntensity;
      else
        intensity      = image(round(row1), round(col2));
      end
    else
      % ..When ouside the image 1,
      % ..set the gray_level to be white.
      intensity = 255;
    end

    image_rotate(row2, col2) = intensity;
  end
end

% Display the result.
figure;

subplot(1, 2, 1);
imshow(image ./ 255);
title('Flowers');
axis on;
axis image;

subplot(1, 2, 2);
imshow(image_rotate ./ 255);
title('Flowers after the 10 degree rotation');
axis on;
axis image;
```

- **Question 7**

```
function hw3_7
%
% CMSC 828D: Fundamentals of Computer Vision
%                        Homework3
%
% Instructors : Larry Davis, Ramani Duraiswami,
%               Daniel DeMenthon, and Yiannis Aloimonos
% Student     : Haiying Liu
%
% Date        : Sept. 16, 2000
%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%==============================================================================
%= Qustion no.7

diary hw3_7.txt

% Generate a random matrix.
matrix             = rand(5)

% Generate a symmetric random matrix.
temp               = rand(5);
temp_l             = tril(temp);
temp_u             = temp_l';
matrix_sym         = temp_l + temp_u
matrix_sym_pos_def = matrix_sym' * matrix_sym

% LU
disp('######');
disp('# LU #');
disp('######');
[L, U] = lu(matrix)

% Cholesky
disp('############');
disp('# Cholesky #');
disp('############');
U = chol(matrix_sym_pos_def)

% Eigenvalue
disp('##############');
disp('# Eigenvalue #');
disp('##############');
eigenvalue = eig(matrix)

% QR
disp('######');
disp('# QR #');
disp('######');
[Q, R] = qr(matrix)

% SVD
disp('#######');
disp('# SVD #');
disp('#######');
[U, S, V] = svd(matrix)

diary off;
```

Example:

```
matrix =

    0.7184    0.3058    0.4562    0.8829    0.4935
    0.9741    0.0802    0.5078    0.7343    0.7427
    0.3277    0.9341    0.6627    0.6582    0.8210
    0.3344    0.1020    0.4170    0.9284    0.1868
    0.7906    0.9477    0.4966    0.7179    0.9243


matrix_sym =

    0.8161    0.6743    0.1632    0.1193    0.4677
    0.6743    1.3561    0.0031    0.8846    0.0224
    0.1632    0.0031    0.0191    0.8012    0.2359
    0.1193    0.8846    0.8012    0.2988    0.2222
    0.4677    0.0224    0.2359    0.2222    0.6882


matrix_sym_pos_def =

    1.3803    1.5811    0.3443    0.9641    0.7837
    1.5811    3.0766    0.8283    1.5518    0.5584
    0.3443    0.8283    0.7245    0.3293    0.4213
    0.9641    1.5518    0.3293    1.5773    0.4839
    0.7837    0.5584    0.4213    0.4839    0.7979

######
# LU #
######

L =

    0.7375    0.2719    0.1321    0.5329    1.0000
    1.0000         0         0         0         0
    0.3364    1.0000         0         0         0
    0.3433    0.0821   -0.5133    1.0000         0
    0.8116    0.9730    1.0000         0         0


U =

    0.9741    0.0802    0.5078    0.7343    0.7427
         0    0.9071    0.4919    0.4112    0.5712
         0         0   -0.3943   -0.2782   -0.2343
         0         0         0    0.4998   -0.2353
         0         0         0         0   -0.0533

############
# Cholesky #
############

U =

    1.1749    1.3457    0.2930    0.8206    0.6671
         0    1.1250    0.3857    0.3977   -0.3016
         0         0    0.6999   -0.0922    0.4889
         0         0         0    0.8586    0.1182
         0         0         0         0    0.0950

##############
# Eigenvalue #
```

```
##############

eigenvalue =

   2.9396
  -0.4482
   0.3927 + 0.1870i
   0.3927 - 0.1870i
   0.0372

######
# QR #
######

Q =

   -0.4728   -0.1229   -0.0322   -0.3427   -0.8018
   -0.6410   -0.4934   -0.0629    0.5393    0.2256
   -0.2156    0.7203   -0.5074    0.3993   -0.1335
   -0.2200   -0.0966   -0.6301   -0.6022    0.4273
   -0.5203    0.4619    0.5836   -0.2638    0.3253


R =

   -1.5195   -0.9129   -1.0342   -1.6078   -1.4084
         0    1.0236    0.3598    0.2452    0.5732
         0         0   -0.3559   -0.5747   -0.0575
         0         0         0   -0.3923    0.2029
         0         0         0         0    0.0427

#######
# SVD #
#######

U =

    0.4222   -0.3202   -0.1009   -0.3332    0.7733
    0.4514   -0.4844    0.5612    0.4655   -0.1731
    0.4748    0.5692   -0.3065    0.5689    0.1816
    0.2982   -0.4350   -0.7136   -0.0518   -0.4583
    0.5511    0.3869    0.2678   -0.5882   -0.3591


S =

    3.1290         0         0         0         0
         0    0.9184         0         0         0
         0         0    0.5655         0         0
         0         0         0    0.2216         0
         0         0         0         0    0.0258


V =

    0.4583   -0.3864    0.6134   -0.3693    0.3578
    0.3712    0.7811   -0.1612   -0.4325    0.1978
    0.3626   -0.0044   -0.2277    0.6670    0.6098
    0.5398   -0.4244   -0.6172   -0.2178   -0.3163
    0.4789    0.2461    0.4061    0.4292   -0.6008
```

- **Question 8**

eig_2by2.m:

```
function [lambda] = eig_2by2(A)
%
% CMSC 828D: Fundamentals of Computer Vision
%                       Homework3
%
% Instructors : Larry Davis, Ramani Duraiswami,
%               Daniel DeMenthon, and Yiannis Aloimonos
% Student     : Haiying Liu
%
% Date        : Sept. 16, 2000
%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%============================================================================
%= Qustion no.8

diary hw3_8.txt

a11 = A(1, 1);
a12 = A(1, 2);
a21 = A(2, 1);
a22 = A(2, 2);

lambda    = zeros(2, 1);
lambda(1) = ((a11 + a22) + sqrt((a11 + a22) .* (a11 + a22) - 4 * (a11 * a22 - a12 *
a21))) / 2;
lambda(2) = ((a11 + a22) - sqrt((a11 + a22) .* (a11 + a22) - 4 * (a11 * a22 - a12 *
a21))) / 2;
```

hw3_8:

```
function hw3_8
%
% CMSC 828D: Fundamentals of Computer Vision
%                       Homework3
%
% Instructors : Larry Davis, Ramani Duraiswami,
%               Daniel DeMenthon, and Yiannis Aloimonos
% Student     : Haiying Liu
%
% Date        : Sept. 16, 2000
%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%============================================================================
%= Qustion no.8

diary hw3_8.txt

% Generate a random matrix.
A = rand(2)

% Compute eigenvalue by eig_2by2.m
disp('###############');
disp('# by eig_2by2 #');
disp('###############');
eig_2by2(A)

% Compute eigenvalue by Matlab function eig.m
```

```
disp('##########');
disp('# by eig #');
disp('##########');
eig(A)

diary off;
```

Example:

```
A =

    0.8959    0.0073
    0.4550    0.8388

###############
# by eig_2by2 #
###############

ans =

    0.9318
    0.8029

##########
# by eig #
##########

ans =

    0.9318
    0.8029
```