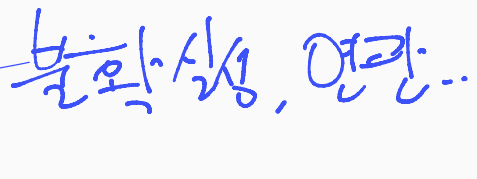
Linear Algebra

Matrix:
Matrix Applications

Automotive Intelligence Lab.

Contents



- Multivariate data covariance matrices
- Geometric transformations via matrix-vector multiplication
- Image feature detection
- Summary
- Code exercises

Multivariate data covariance matrices

Introduction of Covariance and Correlation Matrices

- **How to compute Pearson correlation coefficient?**
  - Vector dot product between two data variables, divided by product of vector norm as learned before.
  - E.g., height and weight

- **What if you have multiple variables more than three variables?**
  - E.g., height, weight, age, weekly exercise
  - You could imagine writing double for loop and applying bivariate correlation formula.
    - Cumbersome and inelegant, therefore not appropriate to linear algebra.

- **In this section…,**
  - We learn how to compute covariance and correlation matrices from multivariate datasets.

Covariance

- **Numerator of the correlation equation**
  - Dot product between two mean-centered variables.

- **Interpreted in same way as correlation**
  - But not bound by ± 1.
    - Because covariance retains scale of data.

- **Normalization prevents covariance from growing larger.**
  - As you sum more data values together.
    - Analogous to dividing by N to transform sum into average.
  - Number of data points is n, normalization factor is $n - 1$.

- **In Eq 1., $\tilde{x}$ to be mean-centered variables x, then covariance is simply Eq 2..**

$$c_{a,b} = (n-1)^{-1} \sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})$$

Eq 1. The equation for covariance

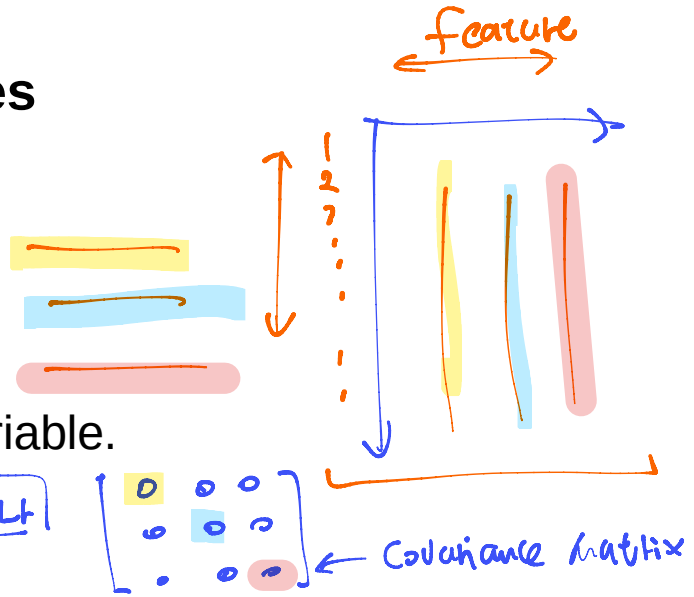
$$c_{a,b} = \tilde{x}^T \tilde{y} / (n-1)$$

Eq 2. The equation for covariance at $\tilde{x}$

Covariance for Multiple Variables

Key insight to implementing Eq 1. for multiple variables

- Matrix multiplication is organized collection of dot products.
  - That between rows of left matrix and columns of right matrix.



Here's what we do.

- Create matrix in which each column corresponds to each variable.
  - Call that matrix $\boldsymbol{X}$ and its sizes: $M \times N$.
  - Variable is data feature.
- Transpose matrix $\boldsymbol{X}$.
  - Rows of $\boldsymbol{X}^T$ correspond to columns of $\boldsymbol{X}$. / n: number of observation, N: dimension of feature
- Assume columns are mean centered and divide by $\boldsymbol{n-1}$.
  - Matrix product $\boldsymbol{X}^T\boldsymbol{X}$ encodes all of pair-wise covariances.
  - The $(i,j)th$ element in the covariance matrix is dot product between data features i and j.

$$c_{a,b} = \tilde{x}^T \tilde{y}/(n-1)$$

Eq 1. The equation for covariance at $\tilde{x}$

$$\boldsymbol{C} = \boldsymbol{X}^T\boldsymbol{X}\frac{1}{n-1}$$

Eq 2. The matrix equation for a covariance matrix

Properties of Covariance Matrix

- **Matrix C is** Symetric
  - Covariance and correlation are symmetric.
    - E.g., Correlation between height and weight is same as correlation between weight and height.

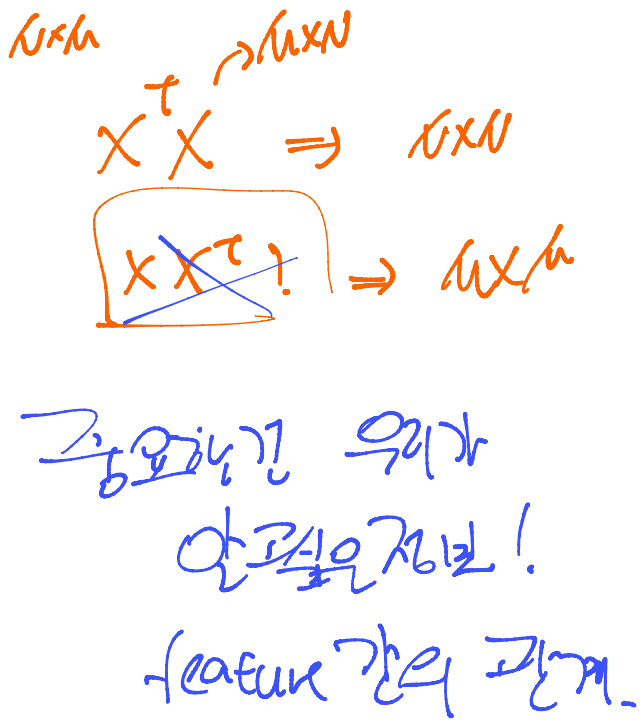
- **What are diagonal elements of C?**
  - Covariances of each variable with itself.
  - In statistics, it is called variance.
    - Quantify dispersion around mean.

$$C = X^T X \frac{1}{n-1}$$

The matrix equation for a covariance matrix

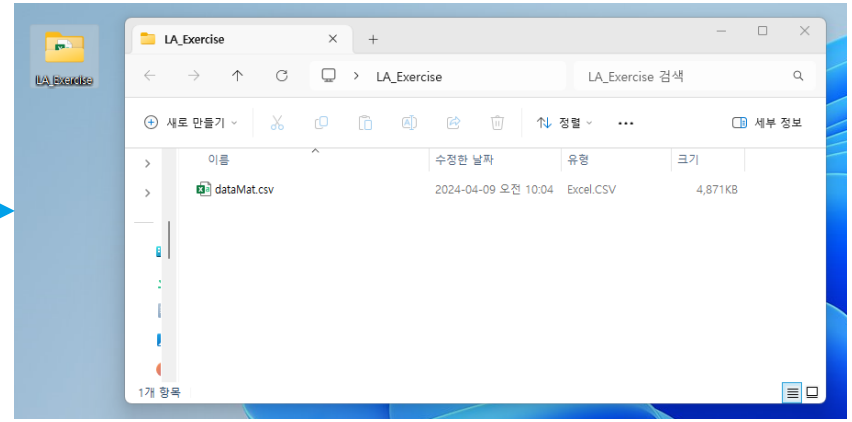
Why Transpose the Left Matrix?

- **Nothing special!**
- **If data matrix is organized as features-by-observations.**
  - Its covariance matrix is $\boldsymbol{XX^T}$.
- **If you are ever uncertain about whether $\boldsymbol{XX^T}$ or $\boldsymbol{X^TX}$.**
  - Think about how to apply rules for matrix multiplication.
    - To produce a features-by-features matrix.
- **Covariance matrices are always features-by-features!**



Code Exercise of Covariance Matrix

- **Create a covariance matrix.**
  - From a publicly available dataset on crime statistics.
    - Includes over a hundred features in various communities around US.
  - To inspect covariance and correlation matrices.
- **Import and some light data processing.**
  - The result is ‘dataMat’.
- **Code Exercise (07_01)**
  - **The directory of csv file is ‘LA_Exercise’.**



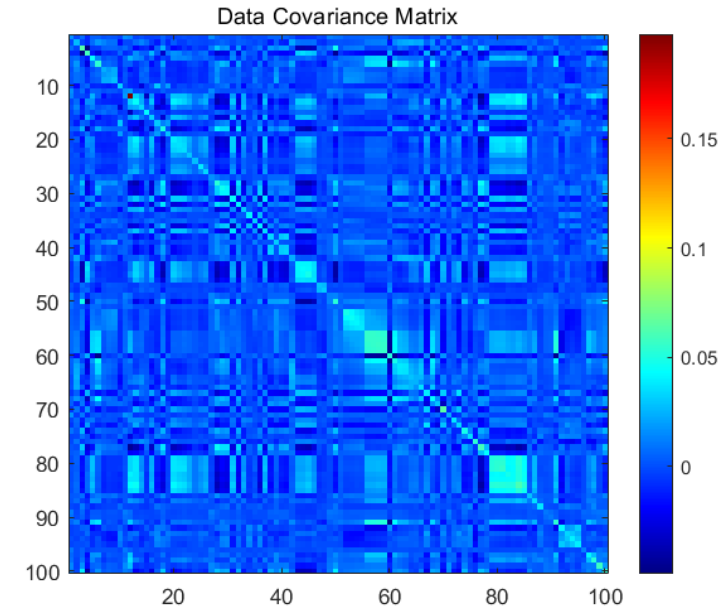

```
% Clear workspace, command window, and close all figures
clc; clear; close all;

% Define the path to the CSV file on the desktop
% The directory is 'Desktop/LA_Exercise'
path_to_home = getenv('HOMEPATH');
path_to_file = append(path_to_home,'/Desktop/LA_Exercise/dataMat.csv');
numericalData = readtable(desktopPath);

% Calculate the mean of each column
columnMeans = mean(numericalData{:,:}, 1);

% Display the means
disp('Column Means:');
disp(columnMeans);

% Mean-center the data
dataMat = numericalData{:,:} - repmat(columnMeans, size(numericalData, 1), 1);
```

```
% Compute the covariance matrix
covarianceMatrix = cov(dataMat);

% Visualize the covariance matrix
figure;
imagesc(covarianceMatrix);
colorbar;
title('Data Covariance Matrix');
colormap jet; % Use a colorful colormap
axis equal;
axis tight;
```

MATLAB code example

Code Result of Covariance Matrix

- **Light colors: Variables that covary positively.**
  - E.g., percentage of divorced males versus number of people in poverty.
- **Dark colors: Variables that covary negatively.**
  - E.g., percentage of divorced males versus median income.
- **Blue colors: Variables that are unrelated to each other.**



Data covariance matrix

Computing a Correlation from a Covariance

- **Scaling by norms of vectors.**

- **This can be translated into matrix equation.**
  - Allow to compute data correlation matrix without for loops.
  - Exercise 1 will walk you through procedure.

- **MATLAB has functions to compute covariance and correlation matrices.**
  - Respectively, cov() and corrcoef().

- **In practice, to use functions is more convenient than to write out code yourself.**
  - But encourage to understand math and mechanisms that those functions implement.

Code Exercise of Correlation from Covariance

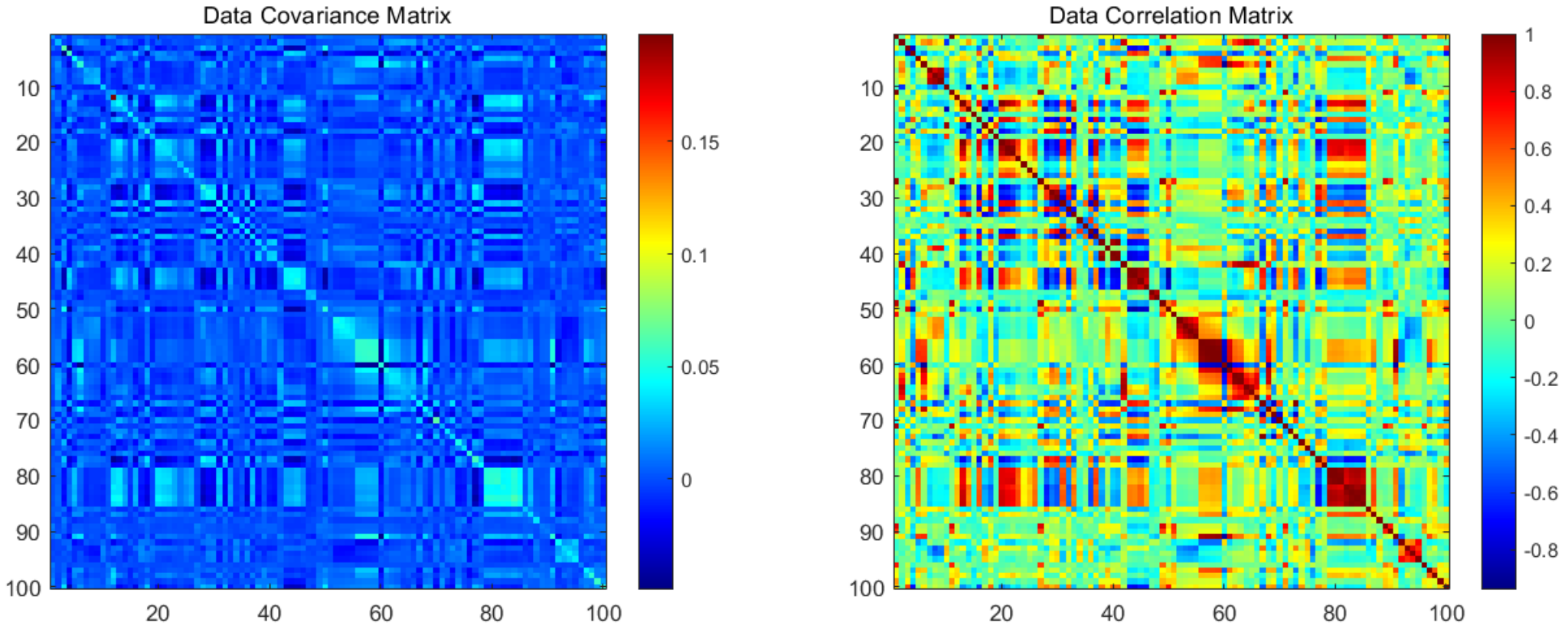
Code Exercise (07_02)

```
% Clear workspace, command window, and close all figures
clc; clear; close all;

% Define the path to the CSV file on the desktop
% The directory is 'Desktop/LA_Exercise'
desktopPath = fullfile(getenv('USERPROFILE'), 'Desktop', 'LA_Exercise', 'dataMat.csv');
numericalData = readtable(desktopPath);

% Calculate the mean of each column
columnMeans = mean(numericalData{:,:}, 1);

% Display the means
disp('Column Means:');
disp(columnMeans);

% Mean-center the data
dataMat = numericalData{:,:} - repmat(columnMeans, size(numericalData, 1), 1);

% Compute the covariance matrix
covarianceMatrix = cov(dataMat);

% Visualize the covariance matrix
figure;
imagesc(covarianceMatrix);
colorbar;
title('Data Covariance Matrix');
colormap jet; % Use a colorful colormap
axis equal;
axis tight;

% Compute and visualize the correlation matrix
correlationMatrix = corrcoef(dataMat);
figure;
imagesc(correlationMatrix);
colorbar;
title('Data Correlation Matrix');
colormap jet; % Use a colorful colormap
axis equal;
axis tight;
```

MATLAB code example of Correlation from Covariance

Code Result of Correlation from Covariance

Code Exercise (07_02)



Result of MATLAB code example of Correlation from Covariance

Geometric transformations via matrix-vector multiplication

Matrix-Vector Multiplication

- **Purpose of matrix-vector multiplication**
  - To apply a geometric transform to a set of coordinates.

- **In this section,**
  - You will see matrix-vector multiplication in 2D static images and in interactive figures.
  - You will learn about pure rotation matrices and about creating interactive figures in MATLAB.

Pure Rotation Matrix

- **Rotate a vector while preserving its length.**
  - E.g., hands of analog clock
    - As time ticks by, hands rotates but don't change in length.

- **Example of orthogonal matrix**
  - Columns of $\boldsymbol{T}$ are Orthogonal
    - Dot product is $cos(\theta)sin(\theta) - sin(\theta)cos(\theta)$.
  - Columns of $\boldsymbol{T}$ are Unit Vectors
    - Trig identity is $cos^2(\theta) + sin^2(\theta) = 1$.

$$\boldsymbol{T} = \begin{bmatrix} cos(\theta) & sin(\theta) \\ -sin(\theta) & cos(\theta) \end{bmatrix}$$

2D rotation matrix

Method to Use Transformation Matrix

- **First, set θ to some angle of clockwise rotation.**

- **Then, multiply matrix T by $2 \times N$ matrix of geometric points.**
  - Each column in that matrix contains (X, Y) coordinates for each of N data points.

- **Example**
  - Setting $\theta = 0$ does not change points' locations.
    - Because $\theta = 0$ means $\mathbf{T} = \mathbf{I}$.
  - Setting $\theta = \pi/2$ rotates points by 90° around origin.

$$\boldsymbol{T} = \begin{bmatrix} cos(\theta) & sin(\theta) \\ -sin(\theta) & cos(\theta) \end{bmatrix}$$

Eq 1. 2D rotation matrix

Visualization Example

- **Consider set of points aligned in a vertical line and effect of multiplying those coordinates by T.**
  - Set $\theta = \pi/5$. How will the results be drawn?

- **Before continuing with this section…,**
  - Inspect online code that generates this figure.
  - Make sure you understand how Eq 1. is translated into code.
  - Take moment to experiment with different rotation angles.
  - Figure out how to make rotation counterclockwise instead of clockwise as Fig 1..
    - Answer: Swap minus signs in sine functions

$$\boldsymbol{T} = \begin{bmatrix} cos(\theta) & sin(\theta) \\ -sin(\theta) & cos(\theta) \end{bmatrix}$$

Eq 1. 2D rotation matrix

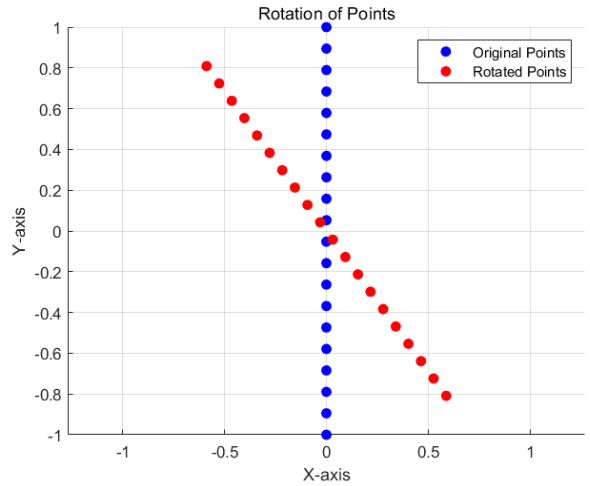


Fig 1. Twirling points around the origin through a pure rotation matrix

Visualization Example: Wobbly Circle

- Wobbly Circle
  - Circles are defined by set of $cos(\theta)$ and $sin(\theta)$ points.
  - Vector of θ angles that range from 0 to 2π

- **Using "Impure rotations" to make Wobbly Circle**
  - Impure rotation: **stretching and rotating, not only rotating**

- **Transformation matrix for Wobbly Circle**
  - Diagonal elements scale x-axis and y-axis coordinates
  - Off-diagonal elements stretch both axes.

- **In Fig 1., value of $\emptyset$ will smoothly transition.**
  - From 1 to 0 and back to 1, following formula $\emptyset = x^2, -1 \leq x \leq 1$.
  - Note that $\boldsymbol{T} = \mathbf{I}$ when $\emptyset = 1$.

$$\boldsymbol{T} = \begin{bmatrix} 1 & 1-\emptyset^2 \\ 0 & 1 \end{bmatrix}$$

2D rotation matrix

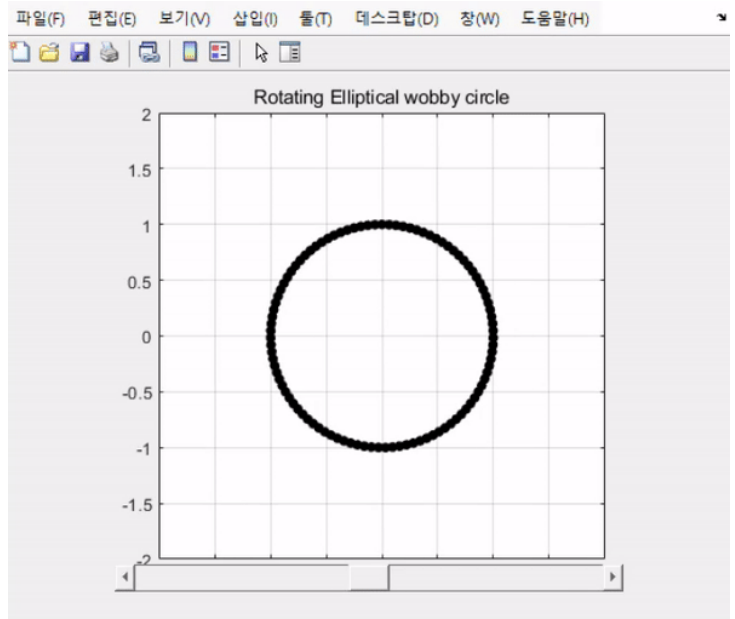


Fig 1.Wobbly Circle

The Code Example of Impure Rotation Transformation: Part 1

- **Set up figure.**
  - Define parameter ø.
  - Create a wobbly circle and plot it.

```
% Clear workspace, command window, and close all figures
clc; clear; close all;

% Define the theta range from 0 to 2*pi with 100 intervals
theta = linspace(0, 2*pi, 100);

% Create an elliptical wobby circle
wobbyCircleX = sin(theta);
wobbyCircleY = cos(theta);

% Prepare the figure
figure;
hPlot = plot(wobbyCircleX, wobbyCircleY, 'ko', 'MarkerSize', 5, 'MarkerFaceColor', 'k');
axis equal;
xlim([-2 2]);
ylim([-2 2]);
grid on;
title('Rotating Elliptical wobby circle');
```

MATLAB code example of Impure Rotation Transformation: Part 1

The Code Example of Impure Rotation Transformation: Part 2

- Generate slide bar to control parameter ∅.

```
% Clear workspace, command window, and close all figures
clc; clear; close all;

% Define the theta range from 0 to 2*pi with 100 intervals
theta = linspace(0, 2*pi, 100);

% Create an elliptical wobby circle
wobbyCircleX = sin(theta);
wobbyCircleY = cos(theta);

% Prepare the figure
figure;
hPlot = plot(wobbyCircleX, wobbyCircleY, 'ko', 'MarkerSize', 5, 'MarkerFaceColor', 'k');
axis equal;
xlim([-2 2]);
ylim([-2 2]);
grid on;
title('Rotating Elliptical wobby circle');

% Create a slider for controlling the rotation angle
hSlider = uicontrol('Style', 'slider', 'Min', -1, 'Max', 1, 'Value', 0, ...
    'Units', 'normalized', 'Position', [0.15 0.05 0.7 0.05], ...
    'Callback', @(src, event) rotateCircle(src, hPlot, wobbyCircleX, wobbyCircleY));
```

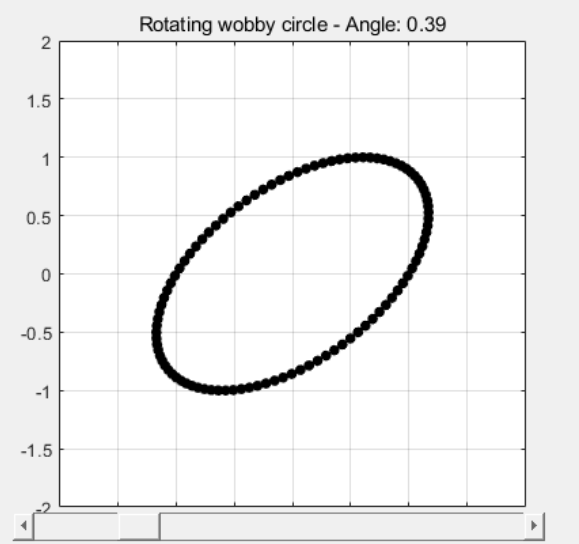
MATLAB code example of Impure Rotation Transformation: Part 2

The Code Example of Impure Rotation Transformation: Part 3

- Define function that updates the shape of circle with the position of bar.
- Code Exercise (07_03)

```
% Clear workspace, command window, and close all figures
clc; clear; close all;

% Define the theta range from 0 to 2*pi with 100 intervals
theta = linspace(0, 2*pi, 100);

% Create an elliptical wobby circle
wobbyCircleX = sin(theta);
wobbyCircleY = cos(theta);

% Prepare the figure
figure;
hPlot = plot(wobbyCircleX, wobbyCircleY, 'ko', 'MarkerSize', 5,
'MarkerFaceColor', 'k');
axis equal;
xlim([-2 2]);
ylim([-2 2]);
grid on;
title('Rotating Elliptical wobby circle');

% Create a slider for controlling the rotation angle
hSlider = uicontrol('Style', 'slider', 'Min', -1, 'Max', 1, 'Value', 0, ...
    'Units', 'normalized', 'Position', [0.15 0.05 0.7 0.05], ...
    'Callback', @(src, event) rotateCircle(src, hPlot, wobbyCircleX,
wobbyCircleY));

% Function to rotate the circle
function rotateCircle(src, hPlot, wobbyCircleX, wobbyCircleY)
    % Get the value of the slider
    x = get(src, 'Value');

    % Update the title with the current angle
    title(sprintf('Rotating wobby circle - x: %.2f', x));

    % Define the rotation matrix
    R = [1 1-x*x; 0 1];

    % Create the rotated wobby circle
    rotatedPoints = R * [wobbyCircleX; wobbyCircleY];

    % Update plot data
    set(hPlot, 'XData', rotatedPoints(1,:), 'YData', rotatedPoints(2,:));
end
```

MATLAB code example of Impure Rotation Transformation: Part 3



Result of code

The Code Example of Impure Rotation Transformation: Part 4

- **Fig 1. show one frame of animation.**
  - You can scroll bar by running code.
  - It does show how matrix multiplication is used in animations.
- **Graphics in CGI movies and video games are slightly more complicated.**
  - Because they use mathematical objects called quaternions.
    - Quaternions: Vectors in $\mathbb{R}^4$ that allow for rotations and translations in 3D.
    - Principle is exactly same.
      - Multiply matrix of geometric coordinates by transformation matrix.

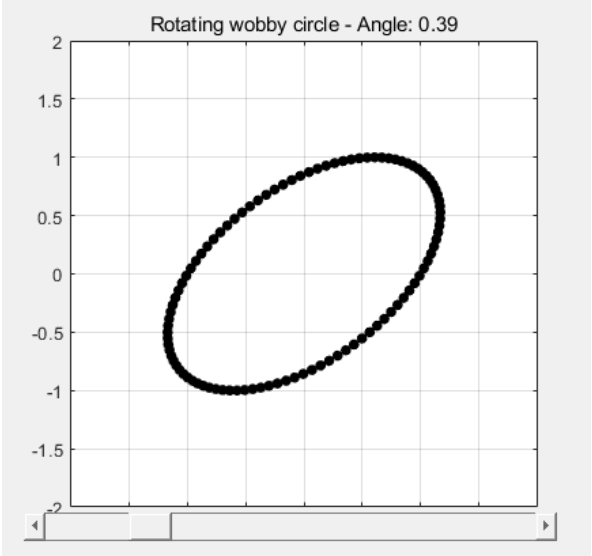


Fig 1. A frame of the movie The Wobby Circle

Additional Things to Consider

- **Change transformation matrix**
  - Setting one of diagonal elements to 0.5 or 2.
- **Change lower-left off-diagonal element**
  - Instead of (or in addition to) upper-right off-diagonal element
- **Parameterize one of diagonal elements**
  - Instead of the off-diagonal element
- **Additional question**
  - Can you figure out how to get circle to wobble to left instead of to right?
    - Answer: Set lower-right element to -1

Image feature detection

Image Filtering

- **Mechanism for image feature detection**
- **Extension of time series filtering**
  - So having gone through chapter 4 will benefit you here.
    - Recall that to filter or detect features in time series signal.
    - Then create time series of dot product
      - Between kernel and overlapping segments of signal.
- **Work same way except in 2D instead of 1D.**
  - Design 2D kernel.
  - Create new image comprising “dot products”.
    - Between kernel and overlapping windows of image.
    - “Dot products”
      - Because operation here is not formally same as vector dot product.

Dot Products

- **Operation here is not formally same as vector dot product.**
  - Computation is same.
    - Element-wise multiplication and sum.
  - However, operation takes place between two matrices.
    - So, implementation is Hadamard multiplication and sum over all matrix elements.
- **Fig 1. illustrates procedure.**
- **Additional details of convolution.**
  - Padding image so that result is same size.
    - You would learn about in image-processing book.
  - Here, focus on linear algebra aspects.

A .* B

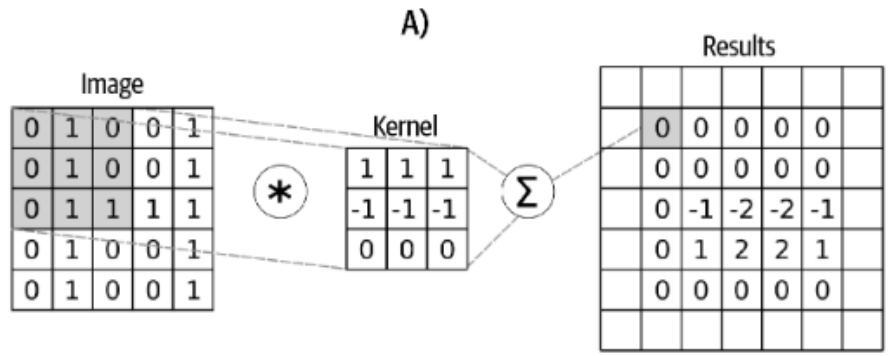
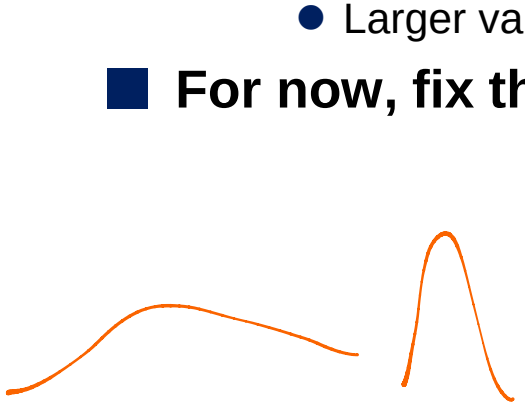


Fig 1. Mechanism of image convolution

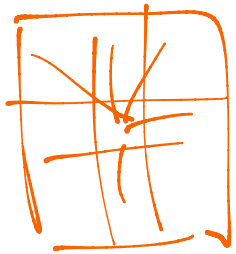
2D Gaussian Kernel

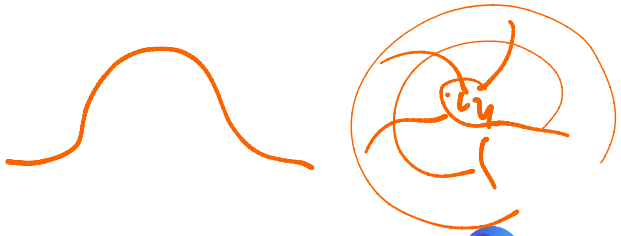
- **A brief introduction to the 2D Gaussian kernel**
- **2D gaussian is given by Eq 1..**
- **A few notes about Eq 1.**
  - exp: The natural exponential
  - $\exp(x)$: Used instead of e^x when exponential term is long.
  - X and Y: 2D grids of x, y coordinates on which to evaluate function.
  - σ: Parameter of function that is often called “shape” or “width”
    - Smaller values make Gaussian narrow
    - Larger values make Gaussian wider
- **For now, fix that parameter to certain values.**



$$G = exp(-(X^2 + Y^2)/\sigma)$$

Eq 1. A 2D Gaussian





2D Gaussian Kernel into Code

- **The X and Y grids**
  - Go from -3 to $+3$ in 21 steps.
- **Width parameter**
  - Hard coded to 20.
- **Normalize**
  - Normalize values in kernel.
  - Sum over entire kernel is 1
    - Preserve original scale of data.
  - When properly normalized.
    - Each pixel in filtered image becomes weighted average of surrounding pixels.
      - Weights defined by the Gaussian.

```
%% Initialize convolution kernel
kernelN = 20;
[X, Y] = meshgrid(linspace(-3, 3, kernelN+1), linspace(-3, 3, kernelN+1));
kernel = exp(-(X.^2 + Y.^2) / kernelN);
kernel = kernel / sum(kernel(:)); % Normalize kernel
```

MATLAB code example of kernel

Method of Smooth a Random-Numbers Matrix

- **Similar to how we smoothed random-numbers time series in chapter 4.**
  - Random-numbers time series in chapter 4
    - Computing the dot product between the kernel and a short snippet of the data of the same length as the kernel is required.
    - This procedure produces one time point in the filtered signal, and then the kernel is moved one time step to the right to compute the dot product with a different (overlapping) signal segment.
    - This procedure is called [ ]
- **In Fig 1., you can see below.**
  - Random-numbers matrix
  - Gaussian kernel
  - Result of convolution

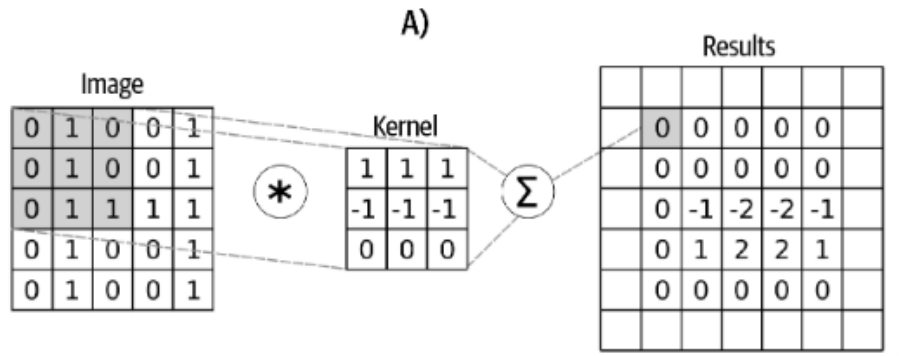


Fig 1. Mechanism of image convolution

Image Convolution into Code

- **Think back to time series convolution.**
  - The idea is same but with extra dimension necessitating extra for loop.

```
% Clear workspace, command window, and close all figures
clc; clear; close all;

% Initialize image size
imgN = 20;
image = randn(imgN, imgN);

% Initialize convolution kernel
kernelN = 20;
[X, Y] = meshgrid(linspace(-3, 3, kernelN+1), linspace(-3, 3, kernelN+1));
kernel = exp(-(X.^2 + Y.^2) / kernelN);
kernel = kernel / sum(kernel(:)); % Normalize kernel

% Prepare for convolution
halfKr = floor(kernelN / 2);
convoutput = zeros(imgN + kernelN, imgN + kernelN);

% Add padding to the image manually (replacing padarray)
imagePad = zeros(size(convoutput));
imagePad(halfKr + 1:end-halfKr, halfKr + 1:end-halfKr) = image;

% Perform convolution for each pixel of the image
for rowi = halfKr + 1 : imgN + halfKr
    for coli = halfKr + 1 : imgN + halfKr
        % Extract a piece of the image
        pieceOfImg = imagePad(rowi - halfKr : rowi + halfKr, coli - halfKr : coli + halfKr);

        % Dot product: element-wise multiplication and sum
        dotprod = sum(sum(pieceOfImg .* kernel));

        % Store the result for this pixel
        convoutput(rowi, coli) = dotprod;
    end
end
```

MATLAB code example of applying convolution

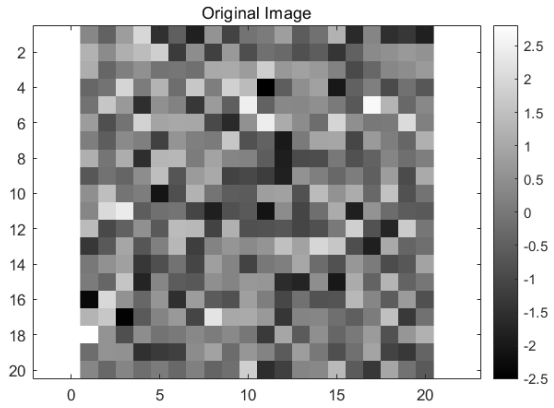
Code Exercise of Convolution

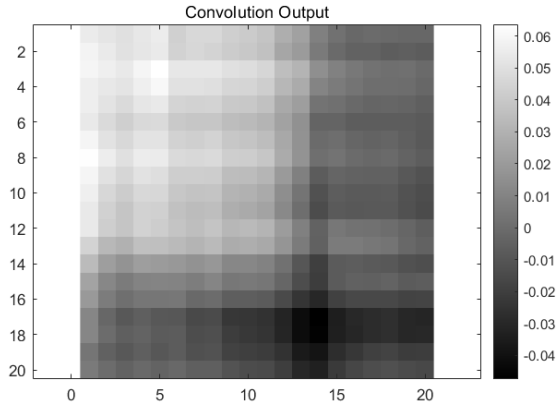
Code Exercise (07_04)

- Convolution with predefined kernel.

```
% Clear workspace, command window, and close all figures
clc; clear; close all;

% Initialize image size
imgN = 20;
image = randn(imgN, imgN);

% Initialize convolution kernel
kernelN = 20;
[X, Y] = meshgrid(linspace(-3, 3, kernelN+1), linspace(-3, 3,
kernelN+1));
kernel = exp(-(X.^2 + Y.^2) / kernelN);
kernel = kernel / sum(kernel(:)); % Normalize kernel

% Prepare for convolution
halfKr = floor(kernelN / 2);
convoutput = zeros(imgN + kernelN, imgN + kernelN);

% Add padding to the image manually (replacing padarray)
imagePad = zeros(size(convoutput));
imagePad(halfKr + 1:end-halfKr, halfKr + 1:end-halfKr) = image;

% Perform convolution for each pixel of the image
for rowi = halfKr + 1 : imgN + halfKr
    for coli = halfKr + 1 : imgN + halfKr
        % Extract a piece of the image
        pieceOfImg = imagePad(rowi - halfKr : rowi + halfKr, coli -
halfKr : coli + halfKr);

        % Dot product: element-wise multiplication and sum
        dotprod = sum(sum(pieceOfImg .* kernel));

        % Store the result for this pixel
        convoutput(rowi, coli) = dotprod;
    end
end
```

```
% Trim the edges
convoutput = convoutput(halfKr + 1:end-halfKr, halfKr + 1:end-
halfKr);

% Visualize the original image
figure;
imagesc(image);
axis equal;
colormap gray;
colorbar;
title('Original Image');

% Visualize the convolution output
figure;
imagesc(convoutput);
axis equal;
colormap gray;
colorbar;
title('Convolution Output');
```





MATLAB code example of Convolution and results

Implementing Convolution

- **In previous method, use double for loop.**
  - Computationally inefficient
- **Method to overcome computationally inefficient**
  - Implement convolution in frequency domain.
    - Faster and with less code
    - Thanks to convolution theorem
      - Convolution in time (or space) domain is equal to multiplication in frequency domain.
- **Recommendation**
  - Use MATLAB conv2() function instead of double for loop.
    - Particularly for large images.

Properties of Real Image

- **Real image is 3D matrix.**
  - Because rows, columns, depth.
    - Depth contains pixel intensity values from red, green, blue color channels.
  - Matrix in $\mathbb{R}^{1675 \times 3000 \times 3}$
- **Formally, that's called tensor.**
  - Because cube, not spreadsheet, of numbers

Smoothing Real Image Converted to Grayscale

- **Use image of the museum in Amsterdam.**
- **For now, reduce it to 2D matrix.**
  - Converting to grayscale as A) of Fig 1..
  - Simplify computations
- **A) of Fig 1 . is Original image and B) of Fig 1. used Gaussian kernel.**
- **A) and B) of Fig 1. used Gaussian kernel.**
  - A): convert to grayscale.
  - B): smooth to image.
- **How many other kernels are available?**
  - Infinite number!

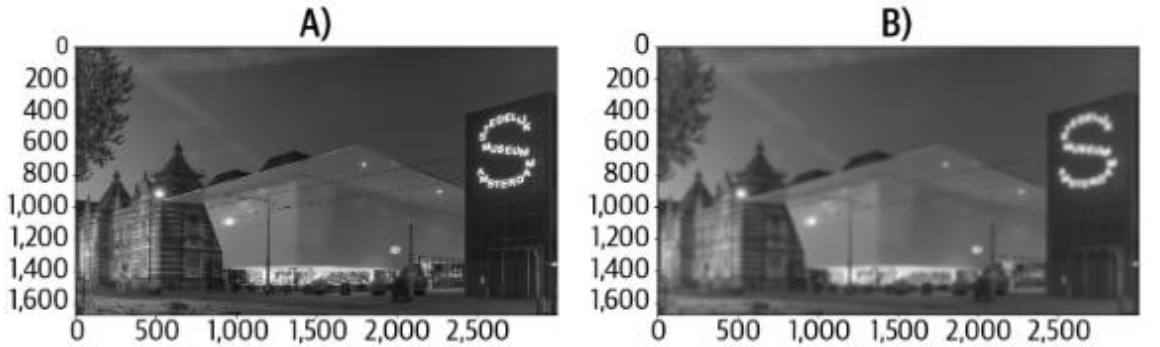


Fig 1. An image of the museum, before and after respectable smoothing

Image Convolution Kernels

- **Major topic in computer vision**
- **The incredible performance of convolutional neural networks**
  - Entirely due to the network's ability to craft optimal filter kernels through learning.

Summary

Summary

- **Important and sophisticated methods in data science and machine learning**
  - Built on simple linear algebra principles.

Code exercises

Covariance and Correlation Matrices Exercises

- In this exercise, you will transform the covariance matrix into a correlation matrix. The procedure involves dividing each matrix element (that is, the covariance between each pair of variables) by the product of the variances of those two variables.
- This is implemented by pre- and post-multiplying the covariance matrix by a diagonal matrix containing inverted standard deviations of each variable (standard deviation is the square root of variance). The standard deviations are inverted because we need to divide by the variances although we will multiply matrices. The reason for pre- and post-multiplying by standard deviations is the special property of pre- and post-multiplying by a diagonal matrix.
- C is the covariance matrix, and S is a diagonal matrix of reciprocated standard deviations per variable (that is, the ith diagonal is $1/\sigma_i$ where is the standard deviation of variable i).
- Your goal in the exercise is to compute the correlation matrix from the covariance matrix, by translating $R = SCS$. into MATLAB code.
- You can then reproduce Fig 1..
- Use dataMat.csv in uploaded zip file.

Covariance and Correlation Matrices Exercises

Result of code and sample code

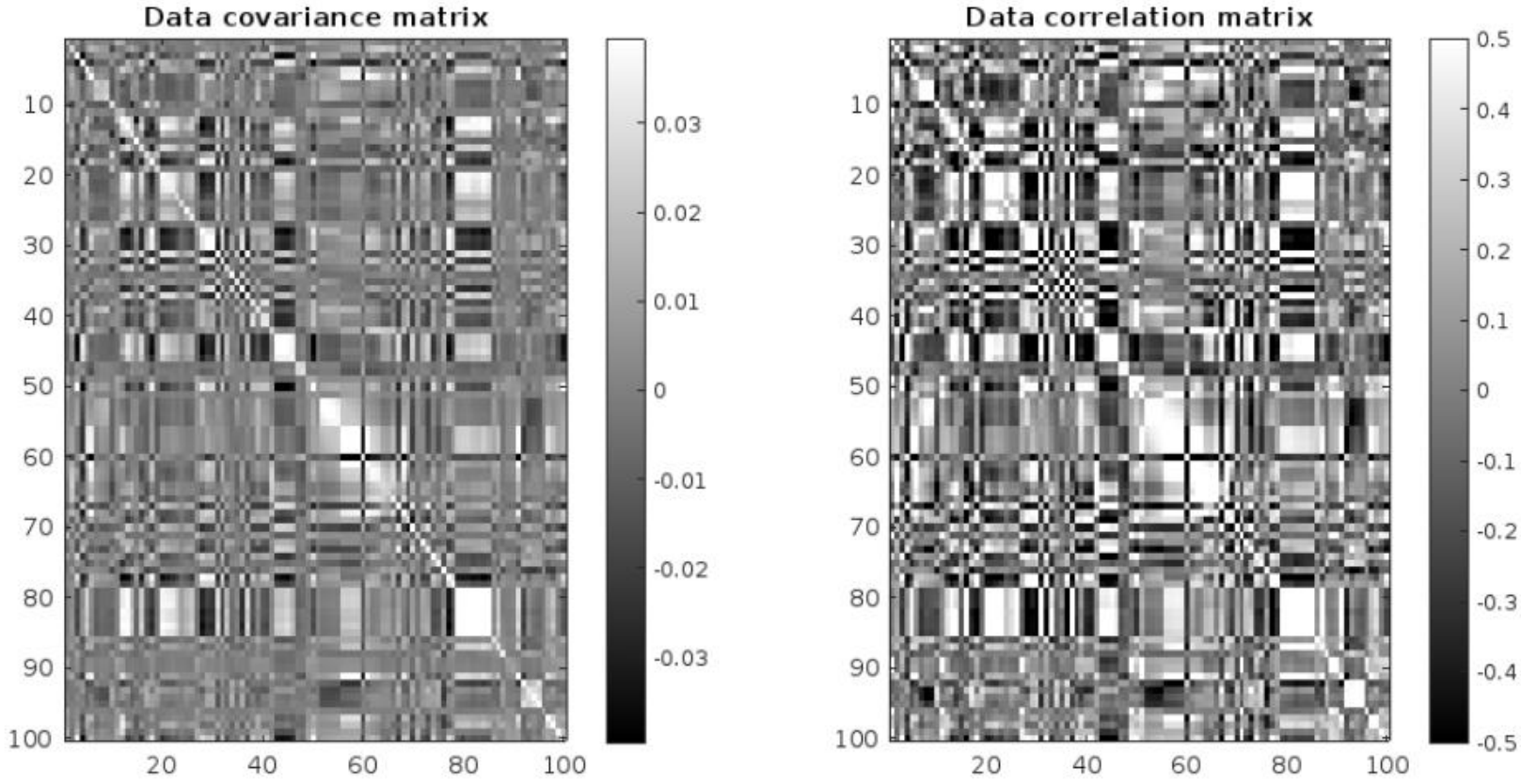


Fig 1. Result of code

Covariance and Correlation Matrices Exercises

Sample code

```
% Clear workspace, command window, and close all figures
clc; clear; close all;

% Define the path to the CSV file
data_path = '/MATLAB Drive/dataMat.csv';

% Read the data from the URL into a table
data = readtable(data_path, 'FileType', 'text', 'Delimiter', ',');
dataMat = table2array(data);

% write here (start)
% Compute the mean of each data feature
datamean = ;

% Mean-center the data
dataMatM = ;

% Confirm that any given feature has mean=0 (or very close...)
disp('Mean of the first feature after centering: ');
disp(mean(dataMatM(:, 1)))

% Compute the covariance matrix
covMat =;

% Diagonal matrix of inverse standard deviations
variances = ; % Extract diagonal elements (variances) from the covariance matrix
standard_devs =;
S = ; % Create diagonal matrix of inverse standard deviations

% Compute the correlation matrix
% R = S*C*S
corrMat = ;

% write here (end)

% Dynamic color scaling
% Just for visualization
clim = max(abs(covMat(:))) * 0.2;

% Visualization of the covariance and correlation matrices
figure('Position', [100, 100, 1300, 600]);

% Plot the covariance matrix
subplot(1, 2, 1); % First subplot
imagesc(covMat, [-clim, clim]); % Display the covariance matrix
colormap('gray'); % Use gray colormap
title('Data covariance matrix', 'FontWeight', 'bold');
colorbar; % Show color scale

% Plot the correlation matrix
subplot(1, 2, 2); % Second subplot
imagesc(corrMat, [-0.5, 0.5]); % Display the correlation matrix
colormap('gray'); % Use gray colormap
title('Data correlation matrix', 'FontWeight', 'bold');
colorbar; % Show color scale
```

Sample code

Geometric Transformations Exercises

- The goal of this exercise is to show points in a circle before and after applying a transformation
- Use Eq 1. and then create a graph that looks like Fig 1..

$$T = \begin{bmatrix} 1 & 0.5 \\ 0 & 0.5 \end{bmatrix}$$

Eq 1. Transformation matrix

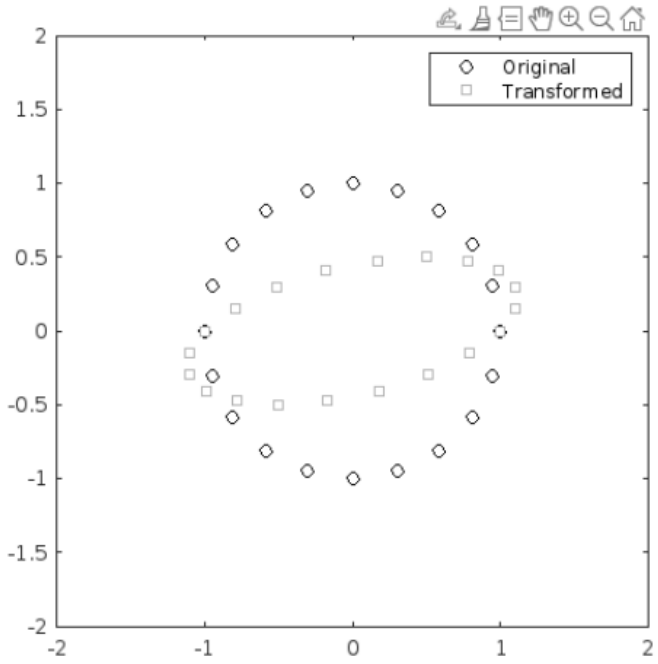


Fig 1. Result of code

Geometric Transformations Exercises

Sample Code

```
% Clear workspace, command window, and close all figures
clc; clear; close all;

% Transformation matrix
T =;

% Define the set of points (a circle)
theta = linspace(0, 2*pi - 2*pi/20, 20);
origPoints = [cos(theta); sin(theta)];

% Apply transformation
transformedPoints = T * origPoints;

% Plot the points
figure('Position', [100, 100, 600, 600]); % Create a figure window
plot(origPoints(1,:), origPoints(2,:), 'ko'); hold on; % Plot original points
plot(transformedPoints(1,:), transformedPoints(2,:), 's', 'Color', [0.7, 0.7,
0.7]); % Plot transformed points

axis square;
xlim([-2, 2]);
ylim([-2, 2]);
legend('Original', 'Transformed'); % Correctly add legend here
```

Sample code

Image Feature Detection Exercises

- Technically, image smoothing is feature extraction, because it involves extracting the smooth features of the signal while dampening the sharp features. Here we will change the filter kernels to solve other image feature detection problems: identifying horizontal and vertical lines.
- The two kernels are shown in Fig 1., as are their effects on the image. You can handcraft the two kernels based on their visual appearance; they are 3×3 and comprise only the numbers -1, 0 and $+1$. Convolve those kernels with the 2D grayscale picture to create the feature maps shown in Fig 1..

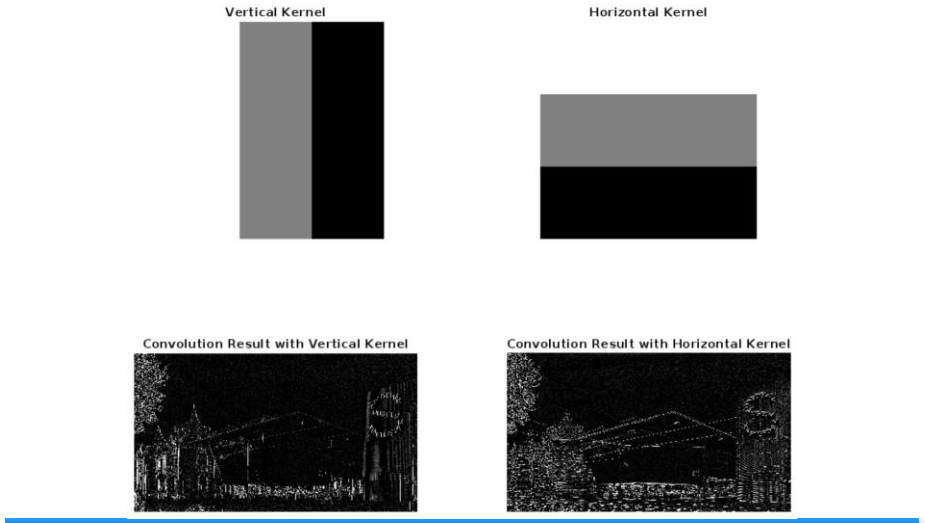


Fig 1. Results of exercise

Image Feature Detection Exercises

Sample Code

```
% Clear workspace, command window, and close all figures
clc; clear; close all;

% Read an image from the web
url = 'https://upload.wikimedia.org/wikipedia/commons/6/61/De_nieuwe_vleugel_van_het
_Stedelijk_Museum_Amsterdam.jpg';
bathtub = webread(url);

% Check the size
disp(size(bathtub));

% Convert the image to 2D for convenience (grayscale)
bathtub2d = rgb2gray(bathtub);
bathtub2d = double(bathtub2d);
bathtub2d = (bathtub2d - min(bathtub2d(:))) / (max(bathtub2d(:)) - min(bathtub2d(:)));

% Create two feature-detection kernels
VK =; % Vertical kernel
HK =; % Horizontal kernel

% Run convolution with the Vertical Kernel
convresVK =;
% Run convolution with the Horizontal Kernel
convresHK =;

% Display kernels and convolution results
figure;
subplot(2,2,1);
imshow(VK, []);
colormap('gray');
title('Vertical Kernel');

subplot(2,2,2);
imshow(HK, []);
colormap('gray');
title('Horizontal Kernel');

subplot(2,2,3);
imshow(convresVK, [], 'Colormap', gray, 'DisplayRange', [0, max(convresVK(:)) * 0.1]);
axis off;
title('Convolution Result with Vertical Kernel');

subplot(2,2,4);
imshow(convresHK, [], 'Colormap', gray, 'DisplayRange', [0, max(convresHK(:)) * 0.1]);
axis off;
title('Convolution Result with Horizontal Kernel');
```

Sample code

THANK YOU
FOR YOUR ATTENTION