Comprehensive Guide to MATLAB for Molecular Biology and Neuroscience Researchers - QCB W12: Introduction to Matlab Day2

Workshop Alignment

This guide is designed to align with Day 2 of the MATLAB workshop, focusing on practical programming and data analysis skills essential for researchers in molecular biology and neuroscience. The topics covered include:

1. **User-defined functions**
2. **If/else statements**
3. **Linear regression**
4. **Plotting using MATLAB's GUI**
    - Box plots and violin plots
    - Surface plots
    - Heat maps

The guide thoroughly analyzes and explains every aspect of the provided MATLAB code files, ensuring you gain a deep understanding of MATLAB's capabilities and how to apply them to your research.

Introduction

MATLAB is a high-level programming language and interactive environment for numerical computation, visualization, and programming. It is particularly powerful for matrix computations, data analysis, and visualization—making it an invaluable tool for researchers analyzing complex biological data.

If you're transitioning from R or Python, you'll find that while some concepts are similar, MATLAB has its unique syntax and functionalities. This guide will bridge that gap, helping you become proficient in MATLAB by exploring practical examples relevant to your field.

Table of Contents

    - W12_Day2_student_updated_v1.m
    - importfile_updated_v1.m
    - merge_updated_v1.m
    - createboxplot_updated_v1.m
2. User-defined Functions
3. If/Else Statements
4. Linear Regression
5. Data Visualization
    - Box Plots and Violin Plots
    - Surface Plots
    - Heat Maps
6. Practical Applications
7. Conclusion

Understanding the Code Files

W12_Day2_student_updated_v1.m

This script introduces fundamental MATLAB concepts through practical examples. It covers variable definitions, array and matrix operations, functions, and file input/output operations.

Key Sections:

Variable Definitions:
matlab
Copy code
```
% Define variables
mynumber = 3.34;
myvect = [1; 2; 3];
myarray = [1, 2, 3; 4, 5, 6];
```

-
    - `mynumber` is a scalar.
    - `myvect` is a column vector.
    - `myarray` is a 2x3 matrix.

Array and Matrix Functions:
matlab
Copy code
```
size(myarray) % Display size of the matrix
myrange = 1:2:10; % Generate a range with step size 2
```

-
    - `size()` returns the dimensions of a matrix.
    - The range `1:2:10` generates `[1, 3, 5, 7, 9]`.

Element-wise Operations:
matlab
Copy code
```
result = [1, 2, 3] .^ 2; % Square each element
```

- The `.^` operator performs element-wise exponentiation.

Functions:
matlab
Copy code
```
sinValues = sin(pi * [1, 2, 3]); % Compute sine values
```

- Computes the sine of each element in the array multiplied by π.

Accessing Matrix Elements:
matlab
Copy code
```
a = [1, 2; 3, 4];
disp(a(1, 2));
```

- Accesses the element in the first row, second column of matrix `a`.

Nested For Loops:
matlab
Copy code
```
for ii = 1:size(a, 1)
    for jj = 1:size(a, 2)
        fprintf('a(%d,%d) = %.2f\n', ii, jj, a(ii, jj));
    end
end
```

- Iterates over each element of matrix `a`, printing its value.

File Input/Output:
matlab
Copy code
```
% Writing to a formatted text file
fileName = 'outputData.tab';
fileID = fopen(fileName, 'w');
for ii = 1:size(a, 1)
    fprintf(fileID, '%8.2f\t', a(ii, :));
    fprintf(fileID, '\n');
end
fclose(fileID);
```

```
% Reading from a file
data = readtable('dataFile.txt');
```

-
    - Demonstrates writing data to a file and reading data from a file using `readtable`.

importfile_updated_v1.m

This function imports data from a text file into a MATLAB table, handling potential errors gracefully.

Key Sections:

Function Definition and Default Parameters:
matlab
Copy code
```
function dataTable = importFile(filename, dataLines)
    if nargin < 2
        dataLines = [2, Inf];
    end
    ...
end
```

-
    - `nargin` checks the number of input arguments.
    - `dataLines` specifies which lines to read from the file.

Import Options Configuration:
matlab
Copy code
```
opts = delimitedTextImportOptions("NumVariables", 3);
opts.DataLines = dataLines;
opts.Delimiter = "\t";
...
```

-
    - Configures how the data should be read, including delimiters, variable names, and types.

Reading the Table with Error Handling:
matlab
Copy code
```
try
    dataTable = readtable(filename, opts);
catch ME
```

```
    error('Failed to read file: %s\nError: %s', filename, ME.message);
end
```

-
    - Uses a try-catch block to handle errors during file reading.

merge_updated_v1.m

This script imports and merges data from multiple text files specified in a list, then visualizes the combined data using a box plot.

Key Sections:

Reading the File List:
matlab
Copy code

```
opts = delimitedTextImportOptions("NumVariables", 1, "DataLines", [2,
Inf]);
...
fileList = readtable(fileListPath, opts);
```

-
    - Reads a list of filenames from a specified file.

Merging Data from Multiple Files:
matlab
Copy code

```
mergedData = table();
labels = {};

for ii = 1:height(fileList)
    fileName = fileList.file{ii};
    data = importFile(fileName);
    expr = data.Expression;
    if mean(expr) > -1
        geneName = extractBefore(fileName, ".txt");
        mergedData = [mergedData, table(expr, 'VariableNames',
{geneName})];
        labels = [labels, {geneName}];
    end
end
```

-
    - Iterates through each file, imports data, and appends it to a combined table.
    - Filters data based on the mean expression value.

Visualizing the Merged Data:
matlab
Copy code
```
createBoxPlot(mergedData{:,:}, labels);
```

-
    - Calls the custom function `createBoxPlot` to visualize the data.

createboxplot_updated_v1.m

This function creates a box plot for the provided data with labeled x-axis categories.

Key Sections:

Function Definition and Input Validation:
matlab
Copy code
```
function createBoxPlot(yData, xLabels)
    if nargin < 2 || isempty(yData) || isempty(xLabels)
        error('Both yData and xLabels must be provided and
non-empty.');
    end
    ...
end
```

-
    - Ensures that both data and labels are provided.

Plotting the Box Chart:
matlab
Copy code
```
figureHandle = figure('Name', 'Box Plot', 'NumberTitle', 'off');
axesHandle = axes('Parent', figureHandle);
hold(axesHandle, 'on');

boxchart(yData, 'BoxFaceColor', 'cyan');
xticklabels(xLabels);

title('Box Plot of Given Data');
xlabel('Categories');
ylabel('Values');

hold(axesHandle, 'off');
```

-
    - Creates a figure and axes.
    - Plots the box chart with custom labels and titles.

User-defined Functions

Understanding Functions in MATLAB

- **Definition:** A function is a block of code that performs a specific task.
- **Purpose:** Functions promote code reusability and modular programming.

Syntax and Structure

matlab
Copy code

```
function output = functionName(input1, input2)
    % FUNCTIONNAME Summary of the function's purpose.
    % Detailed explanation and documentation.

    % Function code...
    output = ...; % Compute output based on inputs.
end
```

- **Function Keyword:** Begins the function definition.
- **Input Arguments:** Variables passed to the function.
- **Output Arguments:** Variables returned by the function.
- **Documentation:** Comments explaining the function's purpose and usage.

Practical Example

Creating a function to calculate the mean of an array:

matlab
Copy code

```
function avg = calculateMean(data)
    % CALCULATEMEAN calculates the average of the input array.
    % Input:
    % - data: Numeric array of values.
    % Output:
    % - avg: Mean of the input data.

    if isempty(data)
        error('Input data is empty.');
    end

    avg = sum(data) / numel(data);
end
```

Usage:
matlab
Copy code

```
data = [1, 2, 3, 4, 5];
average = calculateMean(data);
disp(average); % Outputs 3
```

-

Application in Research

- **Custom Analysis Functions:** Create functions to process experimental data, such as normalizing gene expression levels or calculating statistical metrics.
- **Modularity:** Break down complex analyses into smaller, manageable functions.

If/Else Statements

Concept

Conditional statements allow the execution of code based on certain conditions.

Syntax

matlab
Copy code

```
if condition
    % Code executed if condition is true
elseif anotherCondition
    % Code executed if the previous condition is false and this one is
true
else
    % Code executed if all conditions are false
end
```

Practical Example

Classifying data based on a threshold:

matlab
Copy code

```
function category = classifyValue(value)
    % CLASSIFYVALUE categorizes a value based on predefined
thresholds.
```

```
    % Input:
    % - value: Numeric value to classify.
    % Output:
    % - category: String indicating the category.

    if value < 0
        category = 'Negative';
    elseif value == 0
        category = 'Zero';
    else
        category = 'Positive';
    end
end
```

Usage:
matlab
Copy code

```
result = classifyValue(-5); % Returns 'Negative'
```

-

Application in Research

- **Data Filtering:** Remove outliers or select data that meets certain criteria.
- **Decision Making:** Implement logic to handle different experimental conditions.

Linear Regression

Concept

Linear regression models the relationship between a dependent variable and one or more independent variables.

MATLAB Implementation

- **polyfit:** Fits a polynomial to data.
- **polyval:** Evaluates a polynomial.

Practical Example

Performing linear regression on experimental data:

matlab
Copy code

```
% Sample data
```

```
x = [1, 2, 3, 4, 5];
y = [2.3, 4.5, 6.1, 8.2, 9.8];

% Fit a linear model (degree 1 polynomial)
coefficients = polyfit(x, y, 1);

% Evaluate the fitted model
yFit = polyval(coefficients, x);

% Plot the data and the fitted line
figure;
scatter(x, y, 'filled'); % Original data points
hold on;
plot(x, yFit, 'r-', 'LineWidth', 2); % Fitted line
title('Linear Regression');
xlabel('Independent Variable');
ylabel('Dependent Variable');
legend('Data', 'Fitted Line');
hold off;
```

Application in Research

- **Trend Analysis:** Analyze trends in gene expression over time or dosage.
- **Predictive Modeling:** Predict outcomes based on experimental variables.

Data Visualization

Visualizing data is crucial for interpreting results and communicating findings.

Box Plots and Violin Plots

Box Plots

- **Purpose:** Show the distribution of data based on five summary statistics: minimum, first quartile (Q1), median, third quartile (Q3), and maximum.

Example:

matlab
Copy code

```
% Sample data
group1 = randn(100,1);
group2 = randn(100,1) + 1;
```

```
% Combine data
data = [group1, group2];

% Create box plot
figure;
boxchart(data);
xticklabels({'Group 1', 'Group 2'});
title('Comparison of Two Groups');
ylabel('Values');
```

Violin Plots

- **Purpose:** Similar to box plots but also show the kernel probability density.

Note: MATLAB does not have a built-in `violinplot` function, but you can download it from the MATLAB File Exchange.

Example:

matlab
Copy code

```
% After installing the violinplot function
figure;
violinplot(data);
xticklabels({'Group 1', 'Group 2'});
title('Violin Plot of Two Groups');
ylabel('Values');
```

Surface Plots

- **Purpose:** Display three-dimensional data.

Example:

matlab
Copy code

```
% Create grid data
[X, Y] = meshgrid(-5:0.5:5, -5:0.5:5);
Z = sin(sqrt(X.^2 + Y.^2));

% Create surface plot
figure;
surf(X, Y, Z);
title('3D Surface Plot');
```

```
xlabel('X-axis');
ylabel('Y-axis');
zlabel('Z-axis');
```

Heat Maps

- **Purpose:** Represent data values in a matrix format using colors.

Example:

matlab
Copy code

```
% Sample data matrix
dataMatrix = rand(10, 10);

% Create heat map
figure;
heatmap(dataMatrix);
title('Heat Map of Data Matrix');
xlabel('Samples');
ylabel('Features');
```

Application in Research

- **Box/Violin Plots:** Compare distributions of gene expression between different conditions.
- **Surface Plots:** Visualize activation patterns across brain regions.
- **Heat Maps:** Display gene expression levels across samples or time points.

Practical Applications

Merging and Visualizing Gene Expression Data

Using the provided `merge_updated_v1.m` script and the `createBoxPlot_updated_v1.m` function, you can merge gene expression data from multiple files and visualize it.

Steps:

1. **Prepare a File List:**
    - Create a text file (e.g., `fileList.txt`) with the names of data files to merge.

Import and Merge Data:
matlab

Copy code
```
mergedData = mergeDataFiles('fileList.txt');
```

2.
3. **Visualize Data:**
    - The `mergeDataFiles` function automatically calls `createBoxPlot` to visualize the merged data.

Customizing the Box Plot

Changing Colors:
matlab
Copy code
```
boxchart(yData, 'BoxFaceColor', 'green');
```

-
- **Adding Statistical Annotations:**
    - Use additional functions or toolboxes to add significance markers.

Error Handling and Data Validation

- **Ensure Data Quality:**
    - Check for missing or NaN values using `isnan` or `isempty`.
    - Validate data ranges and consistency.

Use Try-Catch Blocks:
matlab
Copy code
```
try
    % Code that may produce an error
catch ME
    % Handle the error
    disp(ME.message);
end
```

-

Conclusion

This guide provides a comprehensive understanding of key MATLAB concepts aligned with Day 2 of the workshop. By thoroughly analyzing the code files and explaining the concepts, you should now be equipped to:

- Create and use user-defined functions.
- Implement conditional logic with `if/else` statements.
- Perform linear regression analysis.
- Visualize data using various plotting techniques.

Next Steps:

- **Practice:** Apply these concepts to your own datasets.
- **Explore Further:** Look into solving differential equations and computational systems biology modeling with MATLAB.
- **Resources:** Utilize MATLAB's documentation and online resources for additional support.

Additional Tips for Researchers Transitioning from R or Python

- **Indexing:** MATLAB uses 1-based indexing (arrays start at index 1).
- **Array Operations:**
  - Use `.*`, `./`, and `.^` for element-wise operations.
  - Matrix operations use `*`, `/`, and `^`.
- **Function Handles:**
  - Anonymous functions can be created using `@` (e.g., `f = @(x) x^2`).
- **Data Structures:**
  - MATLAB has arrays, cell arrays, structures, and tables.
  - Tables are useful for heterogeneous data, similar to data frames in R or pandas DataFrames in Python.
- **Visualization Customization:**
  - MATLAB offers extensive customization options for plots, including labels, titles, legends, and annotations.

By leveraging MATLAB's powerful computational and visualization capabilities, you can enhance your data analysis workflows and gain deeper insights into your research data.

If you have any questions or need further assistance, feel free to reach out!