Development of Multiple Regression Model to Predict Lower Limb Kinematic Waveforms

Table of Contents

Import gait phases data from spreadsheet

Script for importing data from the following spreadsheet:

```
Workbook: C:\Users\phili\Downloads\Project FYP (Alpha).xlsx
Worksheet: GaitPhases
```

Auto-generated by MATLAB on 05-Jan-2024 22:39:14

Set up the Import Options and import the gait phases data

```
opts = spreadsheetImportOptions("NumVariables", 434);
```

```
% Specify sheet and range
opts.Sheet = "GaitPhases";
opts.DataRange = "A3:PR10";

% Specify column names and types
opts.VariableNames = ["Var1", "GaitPhases", "Var3", "v11", "Var5", "v12", "Var7", "v13", "Var9"
opts.SelectedVariableNames = ["GaitPhases", "v11", "v12", "v13", "v21", "v22", "v23", "v31", "v
opts.VariableTypes = ["char", "string", "char", "double", "char", "double", "char", "double", "

% Specify variable properties
opts = setvaropts(opts, ["Var1", "GaitPhases", "Var3", "Var5", "Var7", "Var9", "Var11", "Var13"
opts = setvaropts(opts, ["Var1", "GaitPhases", "Var3", "Var5", "Var7", "Var9", "Var11", "Var13"

% Import the data
ProjectFYPAlphaS7= readtable("C:\Users\phili\Downloads\Project FYP (Alpha).xlsx", opts, "UseExc
```

Process the imported gait phases data

```
% Assuming 'ProjectFYPAlphaS7' is the table you want to convert
convertedProjectFYPAlphaS7 = table2array(ProjectFYPAlphaS7);

% Extract from the second column to the last column
extractedProjectFYPAlphaS7 = convertedProjectFYPAlphaS7(:, 2:end);
% Round the values to the nearest whole number
gaitPhases = round(str2double(extractedProjectFYPAlphaS7));
```

Import hip data from spreadsheet

Script for importing data from the following spreadsheet:

```
Workbook: C:\Users\phili\Downloads\Project FYP (Alpha).xlsx
Worksheet: Hip
```

Auto-generated by MATLAB on 06-Jan-2024 05:33:17

Set up the Import Options and import the hip data

```
opts = spreadsheetImportOptions("NumVariables", 812);

% Specify sheet and range
opts.Sheet = "Hip";
opts.DataRange = "A4:AEF104";

% Specify column names and types
opts.VariableNames = ["Var1", "VarName2", "C1_T1", "Var4", "Var5", "C1_T2", "Var7", "Var8", "C1
opts.SelectedVariableNames = ["VarName2", "C1_T1", "C1_T2", "C1_T3", "C2_T1", "C2_T2", "C2_T3",
opts.VariableTypes = ["char", "double", "double", "char", "char", "double", "char", "char", "do

% Specify variable properties
opts = setvaropts(opts, ["Var1", "Var4", "Var5", "Var7", "Var8", "Var10", "Var11", "Var12", "Va
opts = setvaropts(opts, ["Var1", "Var4", "Var5", "Var7", "Var8", "Var10", "Var11", "Var12", "Va

% Import the data
```

```
ProjectFYPAlphaS6 = readtable("C:\Users\phili\Downloads\Project FYP (Alpha).xlsx", opts, "UseEx
```

Process the imported hip data

```
% Assuming 'ProjectFYPAlphaS7' is the table you want to convert
convertedProjectFYPAlphaS6 = table2array(ProjectFYPAlphaS6);

% Extract from the second column to the last column
hipAngle = convertedProjectFYPAlphaS6(:, 2:end);

% Iterate through each column
for col = 1:size(hipAngle, 2)
    % Set the last value of the column to be equal to the first value
    hipAngle(end, col) = hipAngle(1, col);
end
```

Import knee data from spreadsheet

Script for importing data from the following spreadsheet:

```
Workbook: C:\Users\phili\Downloads\Project FYP (Alpha).xlsx
Worksheet: Knee
```

Auto-generated by MATLAB on 06-Jan-2024 07:35:47

Set up the Import Options and import the knee data

```
opts = spreadsheetImportOptions("NumVariables", 829);

% Specify sheet and range
opts.Sheet = "Knee";
opts.DataRange = "A4:AEW104";

% Specify column names and types
opts.VariableNames = ["Var1", "VarName2", "VarName3", "Var4", "Var5", "VarName6", "Var7", "Var8
opts.SelectedVariableNames = ["VarName2", "VarName3", "VarName6", "VarName9", "VarName13", "Var
opts.VariableTypes = ["char", "double", "double", "char", "char", "double", "char", "char", "do

% Specify variable properties
opts = setvaropts(opts, ["Var1", "Var4", "Var5", "Var7", "Var8", "Var10", "Var11", "Var12", "Va
opts = setvaropts(opts, ["Var1", "Var4", "Var5", "Var7", "Var8", "Var10", "Var11", "Var12", "Va

% Import the data
ProjectFYPAlphaS14 = readtable("C:\Users\phili\Downloads\Project FYP (Alpha).xlsx", opts, "UseE
```

Process the imported knee data

```
% Assuming 'ProjectFYPAlphaS7' is the table you want to convert
convertedProjectFYPAlphaS14 = table2array(ProjectFYPAlphaS14);

% Extract from the second column to the last column
kneeAngle = convertedProjectFYPAlphaS14(:, 2:end);
```

```
% Iterate through each column
for col = 1:size(kneeAngle, 2)
    % Set the last value of the column to be equal to the first value
    kneeAngle(end, col) = kneeAngle(1, col);
end
```

Import ankle data from spreadsheet

Script for importing data from the following spreadsheet:

```
Workbook: C:\Users\phili\Downloads\Project FYP (Alpha).xlsx
Worksheet: Ankle
```

Auto-generated by MATLAB on 06-Jan-2024 08:54:07

Set up the Import Options and import the ankle data

```
opts = spreadsheetImportOptions("NumVariables", 820);

% Specify sheet and range
opts.Sheet = "Ankle";
opts.DataRange = "A4:AEN104";

% Specify column names and types
opts.VariableNames = ["Var1", "Time_normalized", "AnkleAngle", "Var4", "Var5", "VarName6", "Var
opts.SelectedVariableNames = ["Time_normalized", "AnkleAngle", "VarName6", "VarName9", "VarName
opts.VariableTypes = ["char", "double", "double", "char", "char", "double", "char", "char", "do

% Specify variable properties
opts = setvaropts(opts, ["Var1", "Var4", "Var5", "Var7", "Var8", "Var10", "Var11", "Var12", "Va
opts = setvaropts(opts, ["Var1", "Var4", "Var5", "Var7", "Var8", "Var10", "Var11", "Var12", "Va

% Import the data
ProjectFYPAlphaS17 = readtable("C:\Users\phili\Downloads\Project FYP (Alpha).xlsx", opts, "UseE
```

Process the imported ankle data

```
% Assuming 'ProjectFYPAlphaS17' is the table you want to convert
convertedProjectFYPAlphaS17 = table2array(ProjectFYPAlphaS17);

% Extract from the second column to the last column
ankleAngle = convertedProjectFYPAlphaS17(:, 2:end);
% Assuming ankleAnkle is your 101x216 double matrix

% Iterate through each column
for col = 1:size(ankleAngle, 2)
    % Set the last value of the column to be equal to the first value
    ankleAngle(end, col) = ankleAngle(1, col);
end
```

Clear temporary variables

```
clear opts
```

Processing data for visualization

1. **Prepare Time Points as % of Gait Cycle**

```
%time points as % of gait cycle
timeNormalized=(0:1:100)';
```

1. **Store the Results in Cell Arrays for Each Column**

Hip Angle Data

```
% Assuming 'hipAngle' is a matrix with 216 columns

% Define the number of columns in 'hipAngle'
numColumns = size(hipAngle, 2);

% Initialize a cell array to store the results for each column
dataCellArray_Hip = cell(1, numColumns);

% Iterate through each column of 'hipAngle'
for columnIdx = 1:numColumns
    % Extract the timeNormalized and the corresponding column from 'hipAngle'
    dataColumn = [timeNormalized, hipAngle(:, columnIdx)];

    % Store the result in the cell array
    dataCellArray_Hip{columnIdx} = dataColumn;
end

% Now, dataCellArray contains the desired data for each column
% Access the results using dataCellArray{1}, dataCellArray{2}, ..., dataCellArray{numColumns}
```

Knee Angle Data

```
% Assuming 'ankleAngle' is a matrix with 216 columns

% Define the number of columns in 'kneeAngle'
numColumns = size(kneeAngle, 2);

% Initialize a cell array to store the results for each column
dataCellArray_Knee = cell(1, numColumns);

% Iterate through each column of 'hipAngle'
for columnIdx = 1:numColumns
    % Extract the timeNormalized and the corresponding column from 'hipAngle'
    dataColumn = [timeNormalized, kneeAngle(:, columnIdx)];

    % Store the result in the cell array
    dataCellArray_Knee{columnIdx} = dataColumn;
end

% Now, dataCellArray contains the desired data for each column
% Access the results using dataCellArray{1}, dataCellArray{2}, ..., dataCellArray{numColumns}
```

Ankle Angle Data

```
% Assuming 'ankleAngle' is a matrix with 216 columns

% Define the number of columns in 'ankleAngle'
numColumns = size(ankleAngle, 2);

% Initialize a cell array to store the results for each column
dataCellArray_Ankle = cell(1, numColumns);

% Iterate through each column of 'hipAngle'
for columnIdx = 1:numColumns
    % Extract the timeNormalized and the corresponding column from 'hipAngle'
    dataColumn = [timeNormalized, ankleAngle(:, columnIdx)];

    % Store the result in the cell array
    dataCellArray_Ankle{columnIdx} = dataColumn;
end

% Now, dataCellArray contains the desired data for each column
% Access the results using dataCellArray{1}, dataCellArray{2}, ..., dataCellArray{numColumns}
```

Gait Phases Data

```
% Assuming 'hipAngle' is a matrix with 216 columns

% Define the number of columns in 'hipAngle'
numColumns = size(gaitPhases, 2);

% Initialize a cell array to store the results for each column
dataCellArray_Phases = cell(1, numColumns);

% Iterate through each column of 'hipAngle'
for columnIdx = 1:numColumns
    % Extract the timeNormalized and the corresponding column from 'hipAngle'
    dataColumn = gaitPhases(:, columnIdx);

    % Store the result in the cell array
    dataCellArray_Phases{columnIdx} = dataColumn;
end
```

```
% Assuming 'ankleAngle' is a matrix with 216 columns
% Assuming 'gaitPhases' is a matrix with 216 columns
% Assuming 'timeNormalized' is a vector of appropriate length
% Define the number of columns in 'ankleAngle' and 'gaitPhases'
numColumns_Ankle = size(ankleAngle, 2);
numColumns_Phases = size(gaitPhases, 2);

% Initialize cell arrays to store the results for each column
dataCellArray_Ankle = cell(1, numColumns_Ankle);
dataCellArray_Phases = cell(1, numColumns_Phases);

% Iterate through each column of 'ankleAngle'
for columnIdx = 1:numColumns_Ankle
    % Extract the timeNormalized and the corresponding column from 'ankleAngle'
    dataColumn = [timeNormalized, ankleAngle(:, columnIdx)];

    % Store the result in the cell array
    dataCellArray_Ankle{columnIdx} = dataColumn;
end

% Iterate through each column of 'gaitPhases'
for columnIdx = 1:numColumns_Phases
    % Extract the corresponding column from 'gaitPhases'
    dataColumn = gaitPhases(:, columnIdx);

    % Store the result in the cell array
    dataCellArray_Phases{columnIdx} = dataColumn;
end

% Now, you can use the modified code from the previous response to extract
% values based on your indices and the corresponding data cell arrays.
% Replace 'indexCell' with your actual indices cell array and
% 'dataCell' with the appropriate data cell array (either 'dataCellArray_Ankle'
% or 'dataCellArray_Phases') based on what you want to extract.
```

Keypoints extraction algorithm

Hip extraction

Set up Indices and Data

```
% Assuming indices are stored in a 1x216 cell array called indexCell
indexCell_hip = {... % Your 1x216 cell array with 8x1 strings
 [dataCellArray_Phases{1, 1}], [dataCellArray_Phases{1, 2}], [dataCellArray_Phases{1, 3}], [dat

};

% Assuming data is stored in a 1x216 cell array called dataCell
dataCell_hip = {... % Your 1x216 cell array with 101x2 double matrices
 [dataCellArray_Hip{1, 1}], [dataCellArray_Hip{1, 2}], [dataCellArray_Hip{1, 3}], [dataCellArra

};

% Extract values for each set of indices
```

```
for i = 1:numel(indexCell_hip)
    indices_hip = indexCell_hip{i}; % Double indices
    selected_values_hip = dataCell_hip{i}(indices_hip + 1, :); % +1 because indices start from
    %disp(['Set ', num2str(i), ':']);
   % disp(selected_values_hip);
end
```

Extract Values for Each Set of Indices

```
% Create a cell array to store selected values
selectedValuesCell_hip = cell(size(indexCell_hip));

% Extract values for each set of indices
for i = 1:numel(indexCell_hip)
    indices_hip = indexCell_hip{i}; % Double indices
    selectedValuesCell_hip{i} = dataCell_hip{i}(indices_hip + 1, :); % +1 because indices start
end

% Display the selected values in a table
%disp('Selected Values in Table:');
selectedValuesTable_hip = cell2table(selectedValuesCell_hip, 'VariableNames', cellstr('Set_' +
%disp(selectedValuesTable_hip);
```

```
% Extract the values from row 1, column 1 for each set
firstRowColumnValues_hip = cellfun(@(x) x(1, 1)', selectedValuesCell_hip)';
secondRowColumnValues_hip = cellfun(@(x) x(2, 1)', selectedValuesCell_hip)';
thirdRowColumnValues_hip = cellfun(@(x) x(3, 1)', selectedValuesCell_hip)';
fouthColumnValues_hip = cellfun(@(x) x(4, 1)', selectedValuesCell_hip)';
fithRowColumnValues_hip = cellfun(@(x) x(5, 1)', selectedValuesCell_hip)';
sixRowColumnValues_hip = cellfun(@(x) x(6, 1)', selectedValuesCell_hip)';
seventhRowColumnValues_hip = cellfun(@(x) x(7, 1)', selectedValuesCell_hip)';
eightRowColumnValues_hip = cellfun(@(x) x(8, 1)', selectedValuesCell_hip)';
% Display the extracted values in a table
allRowsColumn1_hip=[firstRowColumnValues_hip secondRowColumnValues_hip thirdRowColumnValues_hip
```

```
% Extract the values from row 1, column 1 for each set
firstRowColumn2Values_hip = cellfun(@(x) x(1, 2)', selectedValuesCell_hip)';
secondRowColumn2Values_hip = cellfun(@(x) x(2, 2)', selectedValuesCell_hip)';
thirdRowColumn2Values_hip = cellfun(@(x) x(3, 2)', selectedValuesCell_hip)';
fouthColumn2Values_hip = cellfun(@(x) x(4, 2)', selectedValuesCell_hip)';
fithRowColumn2Values_hip = cellfun(@(x) x(5, 2)', selectedValuesCell_hip)';
sixRowColumn2Values_hip = cellfun(@(x) x(6, 2)', selectedValuesCell_hip)';
seventhRowColumn2Values_hip = cellfun(@(x) x(7, 2)', selectedValuesCell_hip)';
eightRowColumn2Values_hip = cellfun(@(x) x(8, 2)', selectedValuesCell_hip)';
% Display the extracted values in a table
allRowsColumn2_hip=[firstRowColumn2Values_hip secondRowColumn2Values_hip thirdRowColumn2Values_
```

Knee extraction

```
% Assuming indices are stored in a 1x216 cell array called indexCell
indexCell_knee = {... % Your 1x216 cell array with 8x1 strings
```

```
        [dataCellArray_Phases{1, 1}], [dataCellArray_Phases{1, 2}], [dataCellArray_Phases{1, 3

};

% Assuming data is stored in a 1x216 cell array called dataCell
dataCell_knee = {... % Your 1x216 cell array with 101x2 double matrices
[dataCellArray_Knee{1, 1}], [dataCellArray_Knee{1, 2}], [dataCellArray_Knee{1, 3}], [dataCellAr

};

% Extract values for each set of indices
for i = 1:numel(indexCell_knee)
    indices_knee = indexCell_knee{i}; % Double indices
    selected_values_knee = dataCell_knee{i}(indices_knee + 1, :); % +1 because indices start fr
    %disp(['Set ', num2str(i), ':']);
    %disp(selected_values_knee);
end
```

```
% Create a cell array to store selected values
selectedValuesCell_knee = cell(size(indexCell_knee));

% Extract values for each set of indices
for i = 1:numel(indexCell_knee)
    indices_knee = indexCell_knee{i}; % Double indices
    selectedValuesCell_knee{i} = dataCell_knee{i}(indices_knee + 1, :); % +1 because indices st
end

% Display the selected values in a table
%disp('Selected Values in Table:');
selectedValuesTable_knee = cell2table(selectedValuesCell_knee, 'VariableNames', cellstr('Set_'
%disp(selectedValuesTable_knee);
```

```
% Extract the values from row 1, column 1 for each set
firstRowColumnValues_knee = cellfun(@(x) x(1, 1)', selectedValuesCell_knee)';
secondRowColumnValues_knee = cellfun(@(x) x(2, 1)', selectedValuesCell_knee)';
thirdRowColumnValues_knee = cellfun(@(x) x(3, 1)', selectedValuesCell_knee)';
fouthColumnValues_knee = cellfun(@(x) x(4, 1)', selectedValuesCell_knee)';
fithRowColumnValues_knee = cellfun(@(x) x(5, 1)', selectedValuesCell_knee)';
sixRowColumnValues_knee = cellfun(@(x) x(6, 1)', selectedValuesCell_knee)';
seventhRowColumnValues_knee = cellfun(@(x) x(7, 1)', selectedValuesCell_knee)';
eightRowColumnValues_knee = cellfun(@(x) x(8, 1)', selectedValuesCell_knee)';
% Display the extracted values in a table
allRowsColumn1_knee=[firstRowColumnValues_knee secondRowColumnValues_knee thirdRowColumnValues_
```

```
% Extract the values from row 1, column 1 for each set
firstRowColumn2Values_knee = cellfun(@(x) x(1, 2)', selectedValuesCell_knee)';
secondRowColumn2Values_knee = cellfun(@(x) x(2, 2)', selectedValuesCell_knee)';
thirdRowColumn2Values_knee = cellfun(@(x) x(3, 2)', selectedValuesCell_knee)';
fouthColumn2Values_knee = cellfun(@(x) x(4, 2)', selectedValuesCell_knee)';
fithRowColumn2Values_knee = cellfun(@(x) x(5, 2)', selectedValuesCell_knee)';
sixRowColumn2Values_knee = cellfun(@(x) x(6, 2)', selectedValuesCell_knee)';
```

```
seventhRowColumn2Values_knee = cellfun(@(x) x(7, 2)', selectedValuesCell_knee)';
eightRowColumn2Values_knee = cellfun(@(x) x(8, 2)', selectedValuesCell_knee)';
% Display the extracted values in a table
allRowsColumn2_knee=[firstRowColumn2Values_knee secondRowColumn2Values_knee thirdRowColumn2Valu
```

Ankle extraction

```
% Assuming indices are stored in a 1x216 cell array called indexCell
indexCell_ankle = {... % Your 1x216 cell array with 8x1 strings
    [dataCellArray_Phases{1, 1}], [dataCellArray_Phases{1, 2}], [dataCellArray_Phases{1, 3}], [
};

% Assuming data is stored in a 1x216 cell array called dataCell
dataCell_ankle = {... % Your 1x216 cell array with 101x2 double matrices
[dataCellArray_Ankle{1, 1}], [dataCellArray_Ankle{1, 2}], [dataCellArray_Ankle{1, 3}], [dataCel
};

% Extract values for each set of indices
for i = 1:numel(indexCell_ankle)
    indices_ankle = indexCell_ankle{i}; % Double indices
    selected_values_ankle = dataCell_ankle{i}(indices_ankle + 1, :); % +1 because indices start
    %disp(['Set ', num2str(i), ':']);
    %disp(selected_values_ankle);
end
```

```
% Create a cell array to store selected values
selectedValuesCell_ankle = cell(size(indexCell_ankle));

% Extract values for each set of indices
for i = 1:numel(indexCell_ankle)
    indices_ankle = indexCell_ankle{i}; % Double indices
    selectedValuesCell_ankle{i} = dataCell_ankle{i}(indices_ankle + 1, :); % +1 because indices
end

% Display the selected values in a table
%disp('Selected Values in Table:');
selectedValuesTable_ankle = cell2table(selectedValuesCell_ankle, 'VariableNames', cellstr('Set_
%disp(selectedValuesTable_ankle);
```

```
% Extract the values from row 1, column 1 for each set
firstRowColumnValues_ankle = cellfun(@(x) x(1, 1)', selectedValuesCell_ankle)';
secondRowColumnValues_ankle = cellfun(@(x) x(2, 1)', selectedValuesCell_ankle)';
thirdRowColumnValues_ankle = cellfun(@(x) x(3, 1)', selectedValuesCell_ankle)';
fouthColumnValues_ankle = cellfun(@(x) x(4, 1)', selectedValuesCell_ankle)';
fithRowColumnValues_ankle = cellfun(@(x) x(5, 1)', selectedValuesCell_ankle)';
sixRowColumnValues_ankle = cellfun(@(x) x(6, 1)', selectedValuesCell_ankle)';
seventhRowColumnValues_ankle = cellfun(@(x) x(7, 1)', selectedValuesCell_ankle)';
eightRowColumnValues_ankle = cellfun(@(x) x(8, 1)', selectedValuesCell_ankle)';
% Display the extracted values in a table
allRowsColumn1_ankle=[firstRowColumnValues_ankle secondRowColumnValues_ankle thirdRowColumnValu
```

```
% Extract the values from row 1, column 1 for each set
firstRowColumn2Values_ankle = cellfun(@(x) x(1, 2)', selectedValuesCell_ankle)';
secondRowColumn2Values_ankle = cellfun(@(x) x(2, 2)', selectedValuesCell_ankle)';
thirdRowColumn2Values_ankle = cellfun(@(x) x(3, 2)', selectedValuesCell_ankle)';
fouthColumn2Values_ankle = cellfun(@(x) x(4, 2)', selectedValuesCell_ankle)';
fithRowColumn2Values_ankle = cellfun(@(x) x(5, 2)', selectedValuesCell_ankle)';
sixRowColumn2Values_ankle = cellfun(@(x) x(6, 2)', selectedValuesCell_ankle)';
seventhRowColumn2Values_ankle = cellfun(@(x) x(7, 2)', selectedValuesCell_ankle)';
eightRowColumn2Values_ankle = cellfun(@(x) x(8, 2)', selectedValuesCell_ankle)';
% Display the extracted values in a table
allRowsColumn2_ankle=[firstRowColumn2Values_ankle secondRowColumn2Values_ankle thirdRowColumn2V
```

```
% Extract the first column of data
columnData = allRowsColumn2_ankle(:, 1);

% Calculate quartiles
Q1 = quantile(columnData, 0.25);  % First quartile
Q3 = quantile(columnData, 0.75);  % Third quartile

% Calculate interquartile range (IQR)
IQR = Q3 - Q1;

% Calculate lower and upper bounds
lowerBound = Q1 - 1.5 * IQR;
upperBound = Q3 + 1.5 * IQR;

% Display the results
%disp(['First Quartile (Q1): ', num2str(Q1)]);
%disp(['Third Quartile (Q3): ', num2str(Q3)]);
%disp(['Interquartile Range (IQR): ', num2str(IQR)]);
disp(['Lower Bound: ', num2str(lowerBound)]);
```

```
Lower Bound: -16.2921
```

```
disp(['Upper Bound: ', num2str(upperBound)]);
```

```
Upper Bound: 13.9995
```

Import regression data from spreadsheet

Script for importing data from the following spreadsheet:

```
Workbook: C:\Users\phili\Downloads\Project FYP (Alpha).xlsx
Worksheet: Sheet1
```

Auto-generated by MATLAB on 19-Feb-2024 23:50:47

Set up the Import Options and import the data

```
opts = spreadsheetImportOptions("NumVariables", 52);
```

```
% Specify sheet and range
opts.Sheet = "Sheet1";
opts.DataRange = "A2:AZ217";

% Specify column names and types
opts.VariableNames = ["ws", "bmi", "sex", "age", "HK1_time", "HK2_time", "HK3_time", "HK4_time'
opts.VariableTypes = ["double", "double", "double", "double", "double", "double", "double", "do

% Import the data
ProjectFYPAlphaS20 = readtable("C:\Users\phili\Downloads\Project FYP (Alpha).xlsx", opts, "UseE
```

Clear temporary variables

```
clear opts
```

Regression analysis

This documentation provides a comprehensive overview of the regression analysis performed using MATLAB's Live Script Editor. The analysis involves fitting various regression models to a dataset and calculating predicted values for different response variables.

Data preparation

Loading data

The dataset **`'ProjectFYPAlphaS20'`** is assumed to be loaded into the variable `data`. The dataset contains several predictors and response variables.

```
data=ProjectFYPAlphaS20
```

```
data = 216×52 table
```

...

	ws	bmi	sex	age	HK1_time	HK2_time	HK3_time
1	4.2700	20.6000	1	24	0	11	38
2	4.7000	19.6200	1	23	0	12	38
3	4.8500	20.0400	1	21	0	13	38
4	4.5600	20.0200	1	20	0	13	37
5	3.9100	32.2200	0	18	0	12	37
6	4.6500	24.4900	1	22	0	11	35
7	4.5400	30.9500	1	17	0	7	37
8	4.1500	29.1200	0	20	0	7	36
9	3.9700	18.6600	1	25	0	7	38
10	4.0700	17.7800	1	18	0	11	38
11	3.8400	24.2500	0	18	0	13	39

	ws	bmi	sex	age	HK1_time	HK2_time	HK3_time
12	4.1100	18.0900	1	20	0	13	36
13	4.1400	22.6900	1	20	0	11	39
14	3.9900	22.7700	0	22	0	8	32
15	3.8000	17.4400	0	20	0	11	32
16	3.6800	28.8500	0	23	0	11	32
17	3.7200	31.5200	0	23	0	11	34
18	3.8800	23.0100	1	18	0	9	35
19	3.9700	19.7400	0	18	0	10	31
20	3.6700	22.9400	0	21	0	10	33
21	3.6300	25.7200	0	21	0	10	35
22	3.8100	20.7100	0	19	0	9	33
23	4.0300	17.2100	0	23	0	11	36
24	4.0800	22.0200	1	22	0	10	34
25	4.0900	20.6000	1	24	0	12	29
26	4.5600	19.6200	1	23	0	14	32
27	4.8200	20.0400	1	21	0	10	29
28	4.6500	20.0200	1	20	0	14	32
29	3.7400	32.2200	0	18	0	8	35
30	4.6300	24.4900	1	22	0	12	38
31	4.7000	30.9500	1	17	0	16	34
32	4.1600	29.1200	0	20	0	14	34
33	4.0200	18.6600	1	25	0	12	31
34	4.1900	17.7800	1	18	0	14	30
35	3.9000	24.2500	0	18	0	14	30
36	3.8700	18.0900	1	20	0	14	36
37	4.1800	22.6900	1	20	0	12	38
38	4.0200	22.7700	0	22	0	12	35
39	4.0400	17.4400	0	20	0	14	36
40	3.6600	28.8500	0	23	0	11	39
41	3.7500	31.5200	0	23	0	11	40
42	3.9800	23.0100	1	18	0	16	40
43	3.9700	19.7400	0	18	0	13	31
44	3.6600	22.9400	0	21	0	13	35

	ws	bmi	sex	age	HK1_time	HK2_time	HK3_time
45	3.6700	25.7200	0	21	0	16	38
46	3.8500	20.7100	0	19	0	8	38
47	4.1300	17.2100	0	23	0	6	40
48	4.0300	22.0200	1	22	0	8	33
49	4.0900	20.6000	1	24	0	11	45
50	4.5600	19.6200	1	23	0	9	53
51	4.7400	20.0400	1	21	0	11	45
52	4.6700	20.0200	1	20	0	11	34
53	3.8300	32.2200	0	18	0	12	38
54	4.4900	24.4900	1	22	0	9	39
55	4.4900	30.9500	1	17	0	11	32
56	3.9700	29.1200	0	20	0	13	34
57	3.9800	18.6600	1	25	0	15	33
58	4.1200	17.7800	1	18	0	15	39
59	3.9300	24.2500	0	18	0	14	42
60	4.2500	18.0900	1	20	0	15	47
61	4.0600	22.6900	1	20	0	16	35
62	3.8400	22.7700	0	22	0	14	34
63	4.0400	17.4400	0	20	0	11	30
64	3.7100	28.8500	0	23	0	13	33
65	3.8600	31.5200	0	23	0	13	31
66	3.9600	23.0100	1	18	0	12	33
67	3.8900	19.7400	0	18	0	11	34
68	3.6300	22.9400	0	21	0	13	43
69	3.7000	25.7200	0	21	0	11	32
70	3.8100	20.7100	0	19	0	12	34
71	4.0700	17.2100	0	23	0	12	34
72	4.0600	22.0200	1	22	0	14	33
73	3.1200	20.6000	1	24	0	13	39
74	3.6200	19.6200	1	23	0	13	40
75	3.1600	20.0400	1	21	0	13	40
76	3.3500	20.0200	1	20	0	12	38
77	2.3300	32.2200	0	18	0	13	38

	ws	bmi	sex	age	HK1_time	HK2_time	HK3_time
78	3.3900	24.4900	1	22	0	14	36
79	3.1800	30.9500	1	17	0	11	36
80	2.9300	29.1200	0	20	0	7	37
81	3.4800	18.6600	1	25	0	9	36
82	3.6300	17.7800	1	18	0	11	38
83	3.0300	24.2500	0	18	0	13	40
84	3.1500	18.0900	1	20	0	10	37
85	3.4800	22.6900	1	20	0	11	35
86	2.7600	22.7700	0	22	0	14	40
87	3.1400	17.4400	0	20	0	12	40
88	3.0400	28.8500	0	23	0	14	37
89	3.0500	31.5200	0	23	0	11	38
90	2.9100	23.0100	1	18	0	12	33
91	3.1800	19.7400	0	18	0	10	35
92	3.0300	22.9400	0	21	0	10	41
93	3.0500	25.7200	0	21	0	8	43
94	2.9700	20.7100	0	19	0	3	47
95	3.1900	17.2100	0	23	0	8	48
96	3.0300	22.0200	1	22	0	5	50
97	3.1300	20.6000	1	24	0	4	40
98	3.5900	19.6200	1	23	0	5	41
99	3.3000	20.0400	1	21	0	6	47
100	3.4300	20.0200	1	20	0	13	39

⋮

Defining response variables

A list of response variables is defined. These variables include different metrics such as times and angles for various measurements.

```
% List of response variables
responseVariables = {
  'HK1_time', 'HK2_time', 'HK3_time', 'HK4_time', 'HK5_time', 'HK6_time', 'HK7_time', 'HK8_time
}
```

```
responseVariables = 1×48 cell
'HK1_time'    'HK2_time'    'HK3_time'    'HK4_time'    'HK5_time'    'HK6_time' ···
```

Creating Predictors Matrix

The predictors matrix is created using selected columns from the dataset. These predictors include 'ws', 'bmi', 'sex', and 'age'.

```
% Create predictors matrix
predictors = [data.ws, data.bmi, data.sex, data.age];
% Extract response matrix starting from columns 5 to 88
responseMatrix = table2array(data(:, 5:52));
% Create a table with response matrix and assign column names
responseTable = array2table(responseMatrix, 'VariableNames', responseVariables);
% Initialize a struct to store results
resultsStruct = struct();
% Create a table to store the coefficients
coefficientsTable = table();

% Iterate through each response variable
for i = 1:length(responseVariables)
    % Extract current response variable
    %currentResponse = responseMatrix(:,i);
    currentResponse = responseTable.(responseVariables{i});
    % Fit linear model
    linear = fitlm(predictors, currentResponse);

    % Fit stepwise linear model (if needed)
    stepwise = stepwiselm(predictors, currentResponse);

    % Fit robust linear model using robustfit
    [coef,stats] = robustfit(predictors, currentResponse);
    % Store results in the struct
    resultsStruct.(responseVariables{i}).coef = coef;
    resultsStruct.(responseVariables{i}).stats = stats;
    % Create a table with coefficients and assign variable names
    currentTable = table(coef(1), coef(2), coef(3), coef(4), coef(5), ...
                         'VariableNames', {'Intercept', 'ws', 'bmi', 'sex', 'age'});

    % Add a column for response variable
    currentTable.ResponseVariable = repmat({responseVariables{i}}, height(currentTable), 1);

    % Concatenate with the main coefficients table
    coefficientsTable = [coefficientsTable; currentTable];
    % Add the coefficientsTable to the struct
    resultsStruct.(responseVariables{i}).coefficientsTable = coefficientsTable;
end
```

```
No terms to add to or remove from initial model.
1. Adding x1, FStat = 11.3331, pValue = 0.000902542
1. Adding x3, FStat = 4.7615, pValue = 0.030192
No terms to add to or remove from initial model.
No terms to add to or remove from initial model.
1. Adding x3, FStat = 4.2681, pValue = 0.04004
No terms to add to or remove from initial model.
1. Adding x3, FStat = -214, pValue = 0
2. Adding x1, FStat = Inf, pValue = 0
3. Adding x2, FStat = -212, pValue = 0
```

```
4. Adding x4, FStat = -203.1121, pValue = 0
5. Adding x1:x3, FStat = -107.3973, pValue = 0
6. Adding x1:x4, FStat = -74.3794, pValue = 0
7. Adding x2:x4, FStat = -101.494, pValue = 0
8. Adding x2:x3, FStat = 1053.9908, pValue = 3.6408861e-83
9. Adding x1:x2, FStat = -177.4689, pValue = 0
10. Adding x3:x4, FStat = -18.7009, pValue = 0
No terms to add to or remove from initial model.
1. Adding x1, FStat = 12.4119, pValue = 0.00052195
No terms to add to or remove from initial model.
1. Adding x1, FStat = 9.5746, pValue = 0.0022375
1. Adding x1, FStat = 11.6804, pValue = 0.000757833
2. Adding x3, FStat = 7.1966, pValue = 0.0078859
1. Adding x3, FStat = 5.8543, pValue = 0.016379
1. Adding x1, FStat = 6.509, pValue = 0.011434
No terms to add to or remove from initial model.
No terms to add to or remove from initial model.
1. Adding x1, FStat = 11.3331, pValue = 0.000902542
1. Adding x3, FStat = 4.7615, pValue = 0.030192
No terms to add to or remove from initial model.
No terms to add to or remove from initial model.
1. Adding x3, FStat = 4.2681, pValue = 0.04004
No terms to add to or remove from initial model.
1. Adding x3, FStat = -214, pValue = 0
2. Adding x1, FStat = Inf, pValue = 0
3. Adding x2, FStat = -212, pValue = 0
4. Adding x4, FStat = -203.1121, pValue = 0
5. Adding x1:x3, FStat = -107.3973, pValue = 0
6. Adding x1:x4, FStat = -74.3794, pValue = 0
7. Adding x2:x4, FStat = -101.494, pValue = 0
8. Adding x2:x3, FStat = 1053.9908, pValue = 3.6408861e-83
9. Adding x1:x2, FStat = -177.4689, pValue = 0
10. Adding x3:x4, FStat = -18.7009, pValue = 0
No terms to add to or remove from initial model.
No terms to add to or remove from initial model.
No terms to add to or remove from initial model.
No terms to add to or remove from initial model.
No terms to add to or remove from initial model.
1. Adding x1, FStat = 14.7885, pValue = 0.000158844
No terms to add to or remove from initial model.
No terms to add to or remove from initial model.
No terms to add to or remove from initial model.
1. Adding x1, FStat = 11.3331, pValue = 0.000902542
1. Adding x3, FStat = 4.7615, pValue = 0.030192
No terms to add to or remove from initial model.
No terms to add to or remove from initial model.
1. Adding x3, FStat = 4.2681, pValue = 0.04004
No terms to add to or remove from initial model.
1. Adding x3, FStat = -214, pValue = 0
2. Adding x1, FStat = Inf, pValue = 0
3. Adding x2, FStat = -212, pValue = 0
4. Adding x4, FStat = -203.1121, pValue = 0
5. Adding x1:x3, FStat = -107.3973, pValue = 0
6. Adding x1:x4, FStat = -74.3794, pValue = 0
7. Adding x2:x4, FStat = -101.494, pValue = 0
8. Adding x2:x3, FStat = 1053.9908, pValue = 3.6408861e-83
9. Adding x1:x2, FStat = -177.4689, pValue = 0
10. Adding x3:x4, FStat = -18.7009, pValue = 0
1. Adding x1, FStat = 7.4549, pValue = 0.0068536
1. Adding x1, FStat = 19.3943, pValue = 1.6794e-05
2. Adding x3, FStat = 4.4714, pValue = 0.035629
1. Adding x1, FStat = 35.6566, pValue = 9.80148e-09
1. Adding x1, FStat = 30.0599, pValue = 1.18497e-07
1. Adding x1, FStat = 25.3042, pValue = 1.10747e-06
```

```
2. Adding x3, FStat = 11.9311, pValue = 0.000677869
3. Adding x4, FStat = 6.8706, pValue = 0.0094607
1. Adding x1, FStat = 19.4988, pValue = 1.60049e-05
1. Adding x1, FStat = 11.3328, pValue = 0.000904808
1. Adding x1, FStat = 19.6563, pValue = 1.48729e-05
```

```
    % You can save results or perform additional operations here
```

```
% Example values for ws, bmi, sex, age
ws = 0.39% 0.39 is the mean dimensionless ws; min 0.1, max 0.5
```

```
ws = 0.3900
```

```
bmi = 23.7% 23.70 is the mean bmi; min 18, max 29
```

```
bmi = 23.7000
```

```
sex = 0.5% 0.5; % Assuming 1 for male, 0 for female
```

```
sex = 0.5000
```

```
age = 20.77% 20.77 is the mean age; min 18, max 24
```

```
age = 20.7700
```

```

% Initialize a variable to store the calculated Ki values
calculatedKiValues = zeros(length(responseVariables), 1);

% Iterate through each response variable
for i = 1:length(responseVariables)
    % Extract coefficients for the current response variable
    currentCoefficients = coefficientsTable(strcmp(coefficientsTable.ResponseVariable, response

    % Calculate Ki using the coefficients and example values
    Ki = currentCoefficients.Intercept + ...
         currentCoefficients.ws * ws + ...
         currentCoefficients.bmi * bmi + ...
         currentCoefficients.sex * sex + ...
         currentCoefficients.age * age;

    % Store the calculated Ki value
    calculatedKiValues(i) = Ki;
    % Store the calculated Ki value in the table
    coefficientsTable.CalculatedKi(strcmp(coefficientsTable.ResponseVariable, responseVariables

    % Display the result
    disp(['Calculated Ki for ' responseVariables{i} ': ' num2str(Ki)]);
end
```

```
Calculated Ki for HK1_time: 0
Calculated Ki for HK2_time: 8.8877
Calculated Ki for HK3_time: 38.065
Calculated Ki for HK4_time: 51.3875
Calculated Ki for HK5_time: 60.7383
Calculated Ki for HK6_time: 71.3773
Calculated Ki for HK7_time: 85.1421
Calculated Ki for HK8_time: 100
Calculated Ki for HK1: 15.9476
Calculated Ki for HK2: 14.3809
Calculated Ki for HK3: -8.5236
Calculated Ki for HK4: -12.6917
Calculated Ki for HK5: -3.0429
Calculated Ki for HK6: 10.4805
Calculated Ki for HK7: 19.787
Calculated Ki for HK8: 16.6012
Calculated Ki for KK1_time: 0
Calculated Ki for KK2_time: 8.8877
Calculated Ki for KK3_time: 38.065
Calculated Ki for KK4_time: 51.3875
Calculated Ki for KK5_time: 60.7383
Calculated Ki for KK6_time: 71.3773
Calculated Ki for KK7_time: 85.1421
Calculated Ki for KK8_time: 100
Calculated Ki for KK1: 3.1586
Calculated Ki for KK2: 13.9298
Calculated Ki for KK3: 4.5359
Calculated Ki for KK4: 18.2515
Calculated Ki for KK5: 47.6278
Calculated Ki for KK6: 58.5089
Calculated Ki for KK7: 34.388
Calculated Ki for KK8: 1.7456
Calculated Ki for AK1_time: 0
Calculated Ki for AK2_time: 8.8877
Calculated Ki for AK3_time: 38.065
Calculated Ki for AK4_time: 51.3875
Calculated Ki for AK5_time: 60.7383
Calculated Ki for AK6_time: 71.3773
Calculated Ki for AK7_time: 85.1421
Calculated Ki for AK8_time: 100
Calculated Ki for AK1: -5.3429
Calculated Ki for AK2: -7.0978
Calculated Ki for AK3: -0.51566
Calculated Ki for AK4: -0.45578
Calculated Ki for AK5: -16.084
Calculated Ki for AK6: -14.1655
Calculated Ki for AK7: -3.9055
Calculated Ki for AK8: -8.1895
```

```
% Display the calculated Ki values for all response variables
disp('Calculated Ki values for all response variables:');
```

```
Calculated Ki values for all response variables:
```

```
disp(calculatedKiValues);
```

```
         0
    8.8877
   38.0650
   51.3875
   60.7383
   71.3773
   85.1421
  100.0000
   15.9476
   14.3809
```

```
    -8.5236
   -12.6917
    -3.0429
    10.4805
    19.7870
    16.6012
          0
     8.8877
    38.0650
    51.3875
    60.7383
    71.3773
    85.1421
   100.0000
     3.1586
    13.9298
     4.5359
    18.2515
    47.6278
    58.5089
    34.3880
     1.7456
          0
     8.8877
    38.0650
    51.3875
    60.7383
    71.3773
    85.1421
   100.0000
    -5.3429
    -7.0978
    -0.5157
    -0.4558
   -16.0840
   -14.1655
    -3.9055
    -8.1895
```

```
% Define response variable subsets
AK_time_responseVariables = {'AK1_time', 'AK2_time', 'AK3_time', 'AK4_time', 'AK5_time', 'AK6_t
HK_time_responseVariables = {'HK1_time', 'HK2_time', 'HK3_time', 'HK4_time', 'HK5_time', 'HK6_t
KK_time_responseVariables = {'KK1_time', 'KK2_time', 'KK3_time', 'KK4_time', 'KK5_time', 'KK6_t

AK_angle_responseVariables = {'AK1', 'AK2', 'AK3', 'AK4', 'AK5', 'AK6', 'AK7','AK8'};
HK_angle_responseVariables = {'HK1', 'HK2', 'HK3', 'HK4', 'HK5', 'HK6','HK7','HK8'};
KK_angle_responseVariables = {'KK1', 'KK2', 'KK3', 'KK4', 'KK5', 'KK6', 'KK7', 'KK8'};

AK_velocity_responseVariables = {'AK1_v', 'AK2_v', 'AK3_v', 'AK4_v', 'AK5_v', 'AK6_v', 'AK7_v'}
HK_velocity_responseVariables = {'HK1_v', 'HK2_v', 'HK3_v', 'HK4_v', 'HK5_v', 'HK6_v'};
KK_velocity_responseVariables = {'KK1_v', 'KK2_v', 'KK3_v', 'KK4_v', 'KK5_v', 'KK6_v', 'KK7_v',

AK_acceleration_responseVariables = {'AK1_a', 'AK2_a', 'AK3_a', 'AK4_a', 'AK5_a', 'AK6_a', 'AK7
HK_acceleration_responseVariables = {'HK1_a', 'HK2_a', 'HK3_a', 'HK4_a', 'HK5_a', 'HK6_a'};
KK_acceleration_responseVariables = {'KK1_a', 'KK2_a', 'KK3_a', 'KK4_a', 'KK5_a', 'KK6_a', 'KK7

% Initialize variables to store results
```

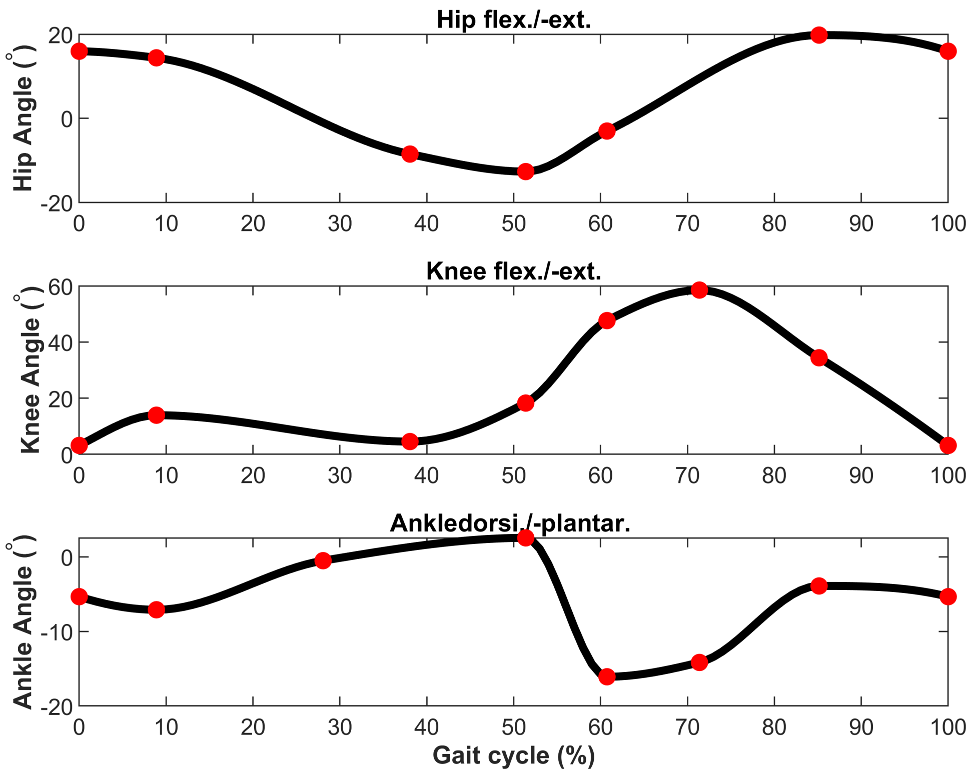
```
AK_time_results = coefficientsTable(ismember(coefficientsTable.ResponseVariable, AK_time_respon
HK_time_results = coefficientsTable(ismember(coefficientsTable.ResponseVariable, HK_time_respon
KK_time_results = coefficientsTable(ismember(coefficientsTable.ResponseVariable, KK_time_respon

AK_angle_results = coefficientsTable(ismember(coefficientsTable.ResponseVariable, AK_angle_resp
HK_angle_results = coefficientsTable(ismember(coefficientsTable.ResponseVariable, HK_angle_resp
KK_angle_results = coefficientsTable(ismember(coefficientsTable.ResponseVariable, KK_angle_resp

AK_velocity_results = coefficientsTable(ismember(coefficientsTable.ResponseVariable, AK_velocit
HK_velocity_results = coefficientsTable(ismember(coefficientsTable.ResponseVariable, HK_velocit
KK_velocity_results = coefficientsTable(ismember(coefficientsTable.ResponseVariable, KK_velocit

AK_acceleration_results = coefficientsTable(ismember(coefficientsTable.ResponseVariable, AK_acc
HK_acceleration_results = coefficientsTable(ismember(coefficientsTable.ResponseVariable, HK_acc
KK_acceleration_results = coefficientsTable(ismember(coefficientsTable.ResponseVariable, KK_acc
```

Reconstruction

```
% Define the struct with three fields: hip, knee, ankle
myStruct = struct('hip', [], 'knee', [], 'ankle', []);
% Initialize the hip field
myStruct.hip = struct(...
    'xSpline', linspace(0,100), ...
    'ySpline', linspace(0,100), ...
    'dySpline', linspace(0,100), ...
    'd2ydx2Spline', linspace(0,100), ...
    'keyEvents', struct(...
        'x', HK_time_results.CalculatedKi, ...
        'y', HK_angle_results.CalculatedKi, ...
        'dydx', HK_velocity_results.CalculatedKi, ...
        'd2ydx2', HK_acceleration_results.CalculatedKi...
    )...
);
% Initialize the knee field
myStruct.knee = struct(...
    'xSpline', linspace(0,100), ...
    'ySpline', linspace(0,100), ...
    'dySpline', linspace(0,100), ...
    'd2ydx2Spline', linspace(0,100), ...
    'keyEvents', struct(...
        'x', KK_time_results.CalculatedKi, ...
        'y', KK_angle_results.CalculatedKi, ...
        'dydx', KK_velocity_results.CalculatedKi, ...
        'd2ydx2', KK_time_results.CalculatedKi...
    )...
);

% Initialize the ankle field
myStruct.ankle = struct(...
    'xSpline', linspace(0,100), ...
    'ySpline', linspace(0,100), ...
    'dySpline', linspace(0,100), ...
    'd2ydx2Spline', linspace(0,100), ...
    'keyEvents', struct(...
        'x', AK_time_results.CalculatedKi, ...
        'y', AK_angle_results.CalculatedKi, ...
        'dydx', AK_velocity_results.CalculatedKi, ...
        'd2ydx2', AK_acceleration_results.CalculatedKi...
    )...
);
```

```
% Remove the 6th value
myStruct.hip.keyEvents.x(6) = [];
myStruct.hip.keyEvents.y(6) = [];

% Define x and y for hip
x_hip = myStruct.hip.keyEvents.x;
y_hip = myStruct.hip.keyEvents.y;

% Iterate through each column
for col = 1:size(y_hip, 2)
    % Set the last value of the column to be equal to the first value
    y_hip(end, col) = y_hip(1, col);
end
% Define x and y for knee
x_knee = myStruct.knee.keyEvents.x;
y_knee = myStruct.knee.keyEvents.y;
for col = 1:size(y_knee, 2)
    % Set the last value of the column to be equal to the first value
    y_knee(end, col) = y_knee(1, col);
end
% Remove the 6th value
% myStruct.ankle.keyEvents.x(6) = [];
% myStruct.ankle.keyEvents.y(6) = [];
% Add 0.05 to the 4th value
myStruct.ankle.keyEvents.y(4) = myStruct.ankle.keyEvents.y(4) + 3;
myStruct.ankle.keyEvents.x(3) = myStruct.ankle.keyEvents.x(3) + (-10);
% Define x and y for ankle
x_ankle = myStruct.ankle.keyEvents.x;
y_ankle = myStruct.ankle.keyEvents.y;
for col = 1:size(y_ankle, 2)
    % Set the last value of the column to be equal to the first value
    y_ankle(end, col) = y_ankle(1, col);
end
% Create a shape-preserving interpolant using pchip for hip
pp_hip = pchip(x_hip, y_hip);
i = 0:1:100;
xi = i;
yi_hip = ppval(pp_hip, xi)';

% Create a shape-preserving interpolant using pchip for knee
pp_knee = pchip(x_knee, y_knee);
yi_knee = ppval(pp_knee, xi)';

% Create a shape-preserving interpolant using pchip for ankle
pp_ankle = pchip(x_ankle, y_ankle);
```

```
yi_ankle = ppval(pp_ankle, xi)';

% Create subplots
figure;

% Subplot for hip
subplot(3,1,1);
plot(xi, yi_hip, 'LineWidth',3,'Color',[0 0 0]);
hold on
plot(x_hip, y_hip, 'MarkerFaceColor',[1 0 0],'Marker','o','LineStyle','none',...
  'Color',[1 0 0]);
hold off
ylabel('Hip Angle (^\circ)','FontWeight','bold','FontName','Arial');
title('Hip flex./-ext.');

% Subplot for knee
subplot(3,1,2);
plot(xi, yi_knee, 'LineWidth',3,'Color',[0 0 0]);
hold on
plot(x_knee, y_knee, 'MarkerFaceColor',[1 0 0],'Marker','o','LineStyle','none',...
  'Color',[1 0 0]);
hold off
ylabel('Knee Angle (^\circ)','FontWeight','bold','FontName','Arial');
title('Knee flex./-ext.');

% Subplot for ankle
subplot(3,1,3);
plot(xi, yi_ankle, 'LineWidth',3,'Color',[0 0 0]);
hold on
plot(x_ankle, y_ankle, 'MarkerFaceColor',[1 0 0],'Marker','o','LineStyle','none',...
  'Color',[1 0 0]);
hold off
ylabel('Ankle Angle (^\circ)','FontWeight','bold','FontName','Arial');
xlabel('Gait cycle (%)','FontWeight','bold','FontName','Arial');
title('Ankledorsi./-plantar. ');
```



```
% x = myStruct.hip.keyEvents.x;
% y = myStruct.hip.keyEvents.y;
% s = spapi(3,x,y);
% i = 0:1:100;
% s = fnval(i,s)';
% % Create figure
% figure;
%
% % Create axes
% axes1 = axes;
% hold(axes1,'on');
%
% hold on
%  %Create plot
% plot(i,s,'LineWidth',3,'Color',[0 0 0]);    %RGB=[0 0 0]
%
% % Create plot
% plot(x,y,'MarkerFaceColor',[1 0 0],'Marker','o','LineStyle','none',...
%   'Color',[1 0 0]);

% % Create ylabel
% %ylabel('Angle (^\circ)','FontWeight','bold','FontName','Arial');
% %ylabel('Velocity (^\circ/gait cycle)','FontWeight','bold','FontName','Arial');
% %ylabel('Acceleration (^\circ/gait cycle^2)','FontWeight','bold','FontName','Arial');
%
% % Create xlabel
% xlabel('Gait cycle (%)','FontWeight','bold','FontName','Arial');
```

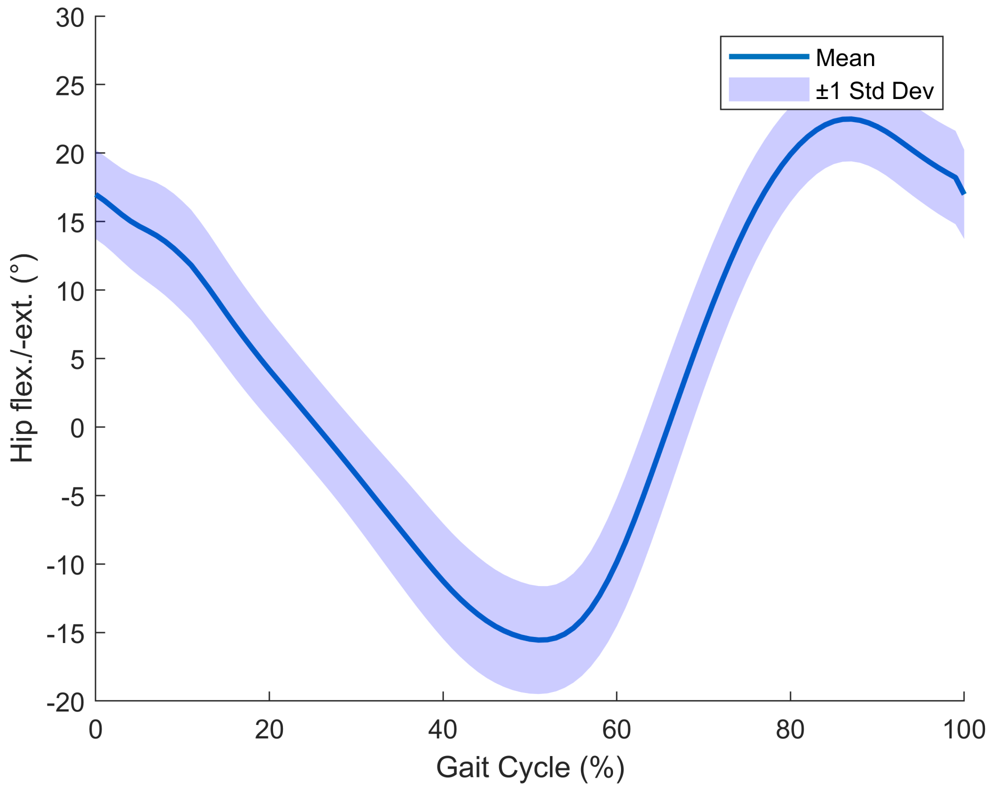
```
%
% % Uncomment the following line to preserve the X-limits of the axes
% % xlim(axes1,[0 100]);
% hold(axes1,'off');
% % Set the remaining axes properties
% set(axes1,'FontName','Arial','FontSize',24,'XColor',[0 0 0],'YColor',...
%     [0 0 0],'ZColor',[0 0 0]);
%
% %Add Legend
% legend
```

Data storage

```
structName.gaitPhases = gaitPhases;
structName.hipAngle = hipAngle;
structName.kneeAngle = kneeAngle;
structName.ankleAngle = ankleAngle;
structName.timeNormalized = timeNormalized;
structName.keyPoints.hipExtraction = {allRowsColumn1_hip allRowsColumn2_hip};
structName.keyPoints.kneeExtraction = {allRowsColumn1_knee allRowsColumn2_knee};
structName.keyPoints.ankleExtraction = {allRowsColumn1_ankle allRowsColumn2_ankle};
structName.regressionData = data;
structName.stepwise = stepwise;
structName.myStruct = myStruct;
```

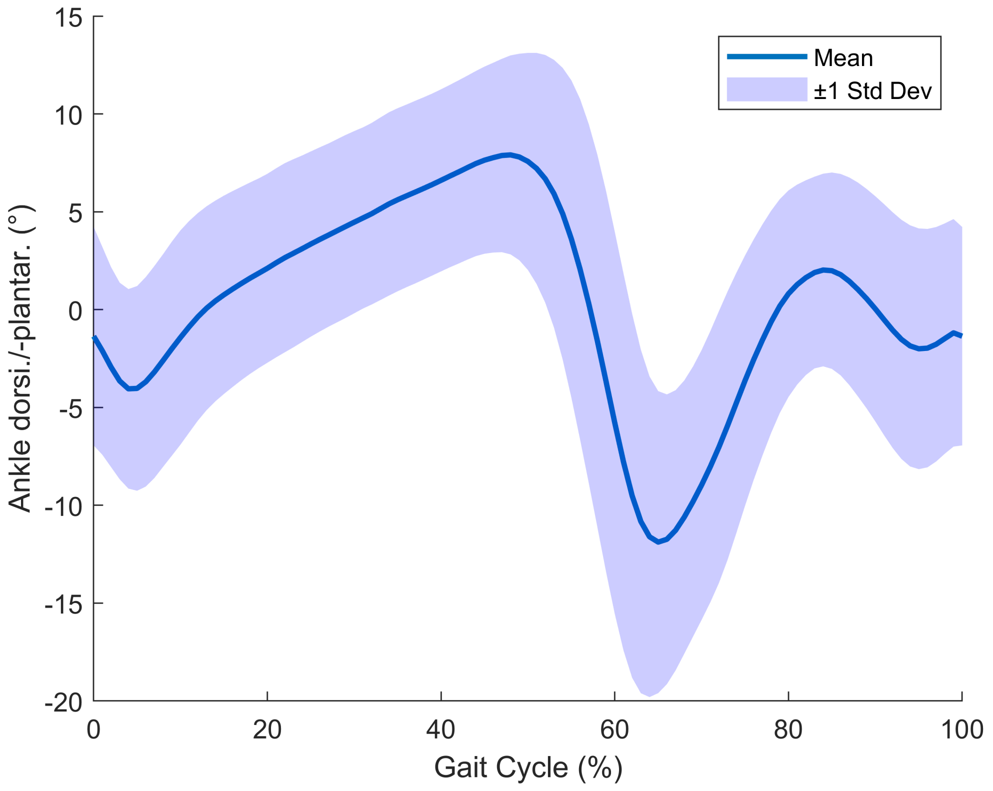
Plots

```
% Assuming the variables in the workspace are named dataMatrix (101x216) and xValues (101x1)
% If they have different names, replace them accordingly

% Calculate the mean and standard deviation across the 216 columns
meanValues = mean(hipAngle, 2); % 101x1 vector
stdValues = std(hipAngle, 0, 2); % 101x1 vector

% Plot the data
figure;
hold on;

% Plot the mean values
plot(timeNormalized, meanValues, 'LineWidth', 2, 'DisplayName', 'Mean');

% Plot the shaded area for ±1 standard deviation
fill([timeNormalized; flipud(timeNormalized)], [meanValues + stdValues; flipud(meanValues - std
    'b', 'FaceAlpha', 0.2, 'EdgeColor', 'none', 'DisplayName', '±1 Std Dev');

hold off;
xlabel('Gait Cycle (%)');
ylabel('Hip flex./-ext. (°)');
%title('Mean and Standard Deviation with Shaded Area');
legend('show');
```



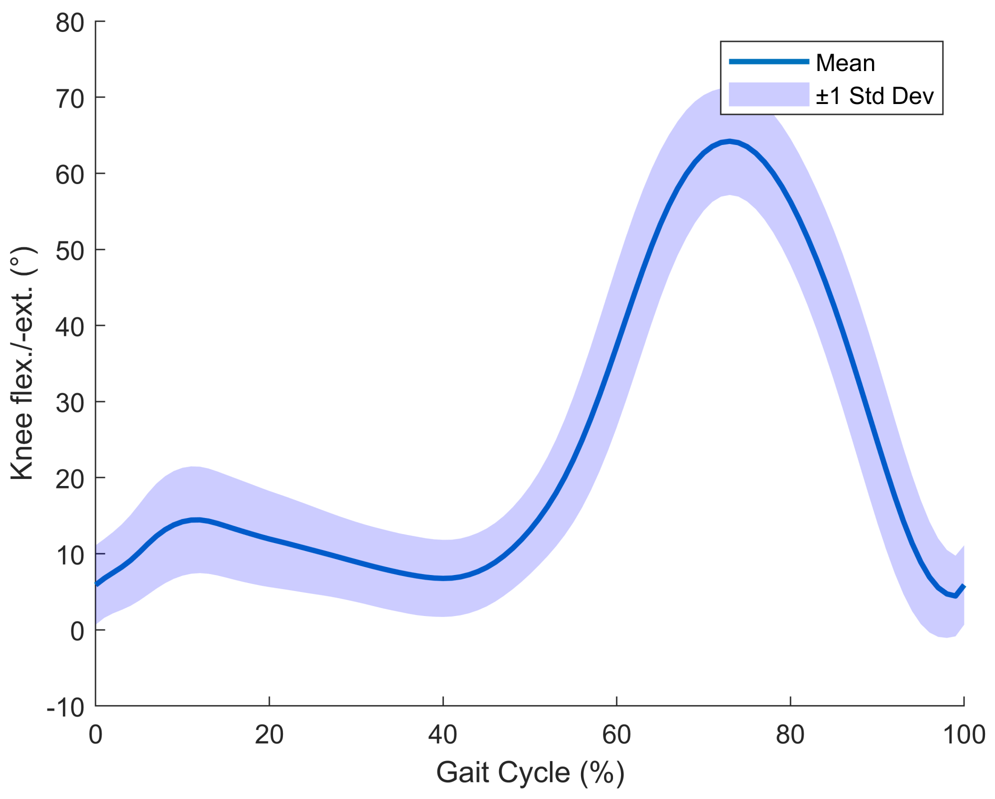
```
%grid on;
```

```
% Assuming the variables in the workspace are named dataMatrix (101x216) and xValues (101x1)
% If they have different names, replace them accordingly

% Calculate the mean and standard deviation across the 216 columns
meanValuesAnkle = mean(ankleAngle, 2); % 101x1 vector
stdValuesAnkle = std(ankleAngle, 0, 2); % 101x1 vector

% Plot the data
figure;
hold on;

% Plot the mean values
plot(timeNormalized, meanValuesAnkle, 'LineWidth', 2, 'DisplayName', 'Mean');

% Plot the shaded area for ±1 standard deviation
fill([timeNormalized; flipud(timeNormalized)], [meanValuesAnkle + stdValuesAnkle; flipud(meanVa
    'b', 'FaceAlpha', 0.2, 'EdgeColor', 'none', 'DisplayName', '±1 Std Dev');

hold off;
xlabel('Gait Cycle (%)');
ylabel('Ankle dorsi./-plantar. (°)');
%title('Mean and Standard Deviation with Shaded Area');
legend('show');
```



```
%grid on;
```

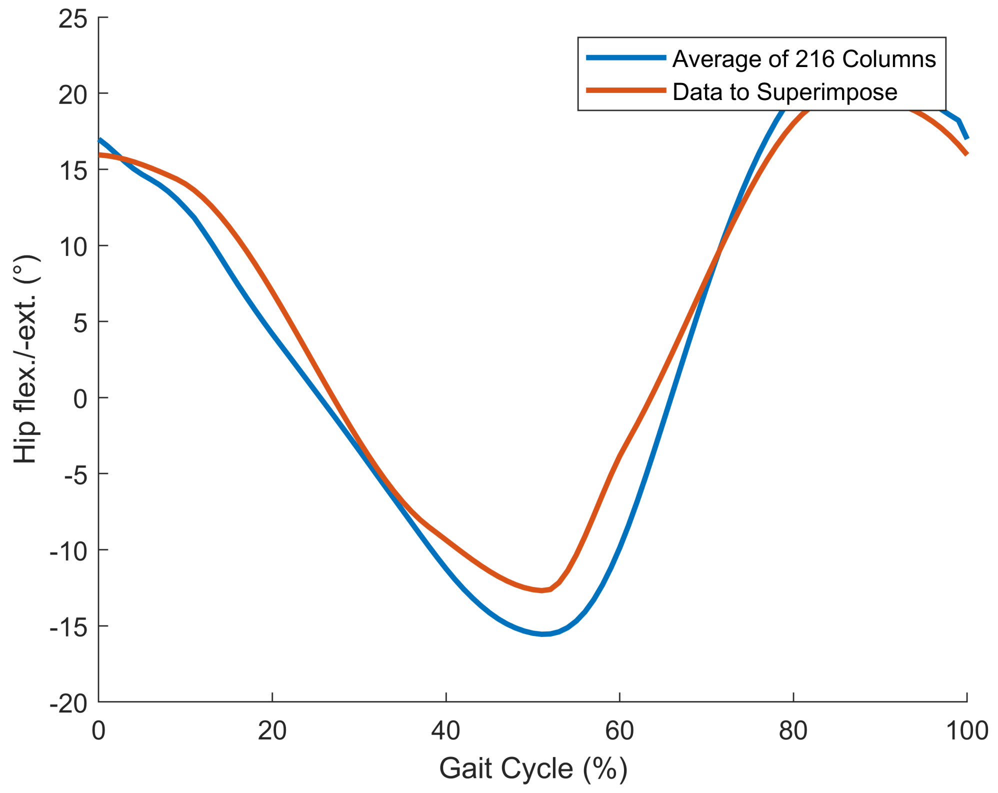
```
% Assuming the variables in the workspace are named dataMatrix (101x216) and xValues (101x1)
% If they have different names, replace them accordingly

% Calculate the mean and standard deviation across the 216 columns
meanValuesKnee = mean(kneeAngle, 2); % 101x1 vector
stdValuesKnee = std(kneeAngle, 0, 2); % 101x1 vector

% Plot the data
figure;
hold on;

% Plot the mean values
plot(timeNormalized, meanValuesKnee, 'LineWidth', 2, 'DisplayName', 'Mean');

% Plot the shaded area for ±1 standard deviation
fill([timeNormalized; flipud(timeNormalized)], [meanValuesKnee + stdValuesKnee; flipud(meanValu
    'b', 'FaceAlpha', 0.2, 'EdgeColor', 'none', 'DisplayName', '±1 Std Dev');

hold off;
xlabel('Gait Cycle (%)');
ylabel('Knee flex./-ext. (°)');
%title('Mean and Standard Deviation with Shaded Area');
legend('show');
```



```
%grid on;
```

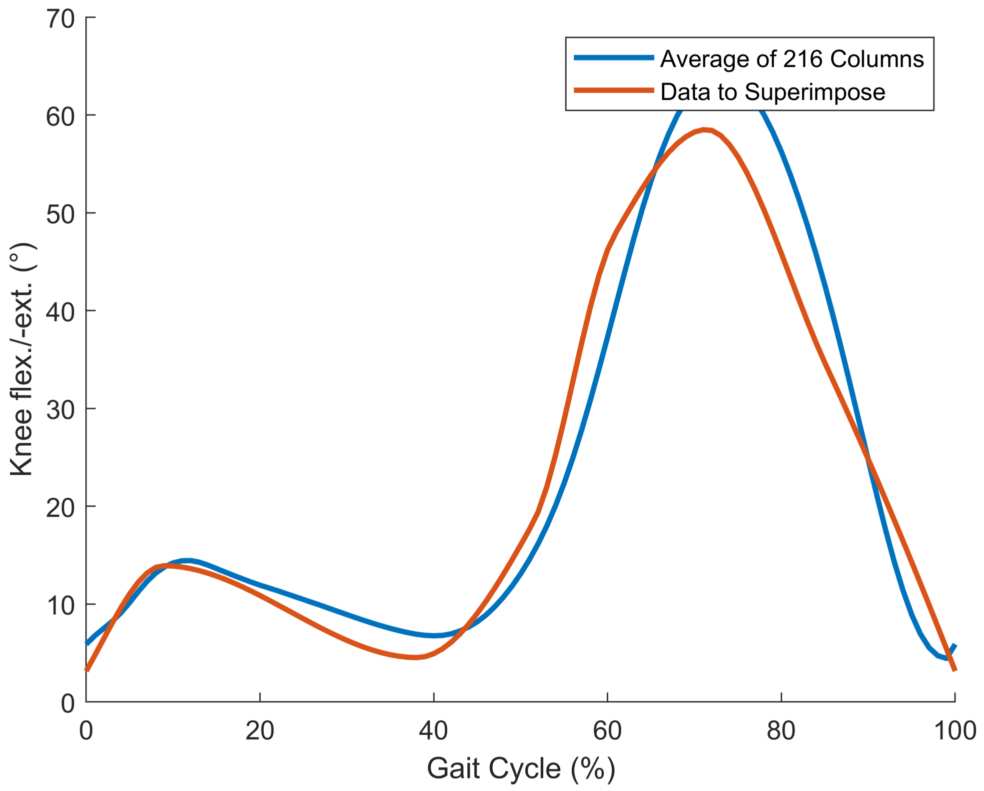
```
% Assuming the variables in the workspace are:
% dataMatrix - 101x216 double matrix
% xValues - 101x1 double vector for x-axis
% dataToSuperimpose - 101x1 double vector to be superimposed on the plot

% Compute the average of the 216 columns
averageDataHip = mean(hipAngle, 2);

% Create the plot
figure;
hold on;

% Plot the average data
plot(timeNormalized, averageDataHip, 'LineWidth', 2, 'DisplayName', 'Average of 216 Columns');

% Superimpose the other data
plot(timeNormalized, yi_hip, 'LineWidth', 2, 'DisplayName', 'Data to Superimpose');

% Add labels and title
xlabel('Gait Cycle (%)');
ylabel('Hip flex./-ext. (°)');
%title('Superimposed Plot of Average Data and Other Data');
legend('show');

% Display the plot
```

```
hold off;
```



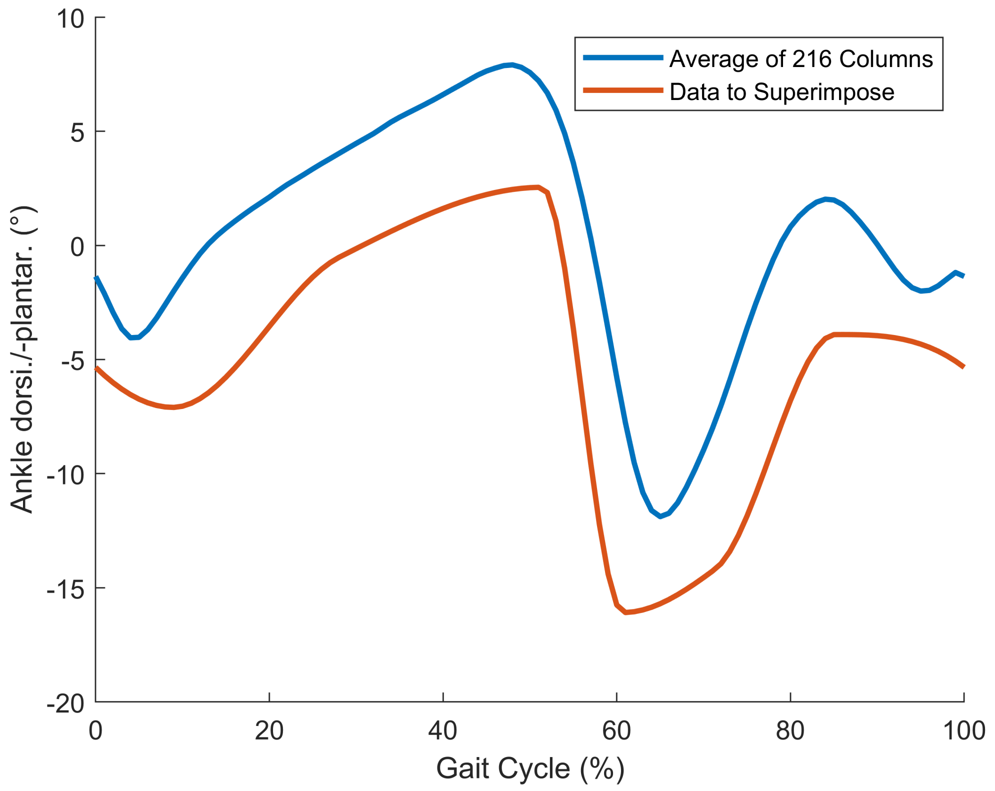
```
% Assuming the variables in the workspace are:
% dataMatrix - 101x216 double matrix
% xValues - 101x1 double vector for x-axis
% dataToSuperimpose - 101x1 double vector to be superimposed on the plot

% Compute the average of the 216 columns
averageDataKnee = mean(kneeAngle, 2);

% Create the plot
figure;
hold on;

% Plot the average data
plot(timeNormalized, averageDataKnee, 'LineWidth', 2, 'DisplayName', 'Average of 216 Columns')

% Superimpose the other data
plot(timeNormalized, yi_knee, 'LineWidth', 2, 'DisplayName', 'Data to Superimpose');

% Add labels and title
xlabel('Gait Cycle (%)');
ylabel('Knee flex./-ext. (°)');
%title('Superimposed Plot of Average Data and Other Data');
legend('show');

% Display the plot
```

```
hold off;
```



```
% Assuming the variables in the workspace are:
% dataMatrix - 101x216 double matrix
% xValues - 101x1 double vector for x-axis
% dataToSuperimpose - 101x1 double vector to be superimposed on the plot

% Compute the average of the 216 columns
averageDataAnkle = mean(ankleAngle, 2);

% Create the plot
figure;
hold on;

% Plot the average data
plot(timeNormalized, averageDataAnkle, 'LineWidth', 2, 'DisplayName', 'Average of 216 Columns'

% Superimpose the other data
plot(timeNormalized, yi_ankle, 'LineWidth', 2, 'DisplayName', 'Data to Superimpose');

% Add labels and title
xlabel('Gait Cycle (%)');
ylabel('Ankle dorsi./-plantar. (°)');
%title('Superimposed Plot of Average Data and Other Data');
legend('show');

% Display the plot
```

```
hold off;
```



Computing r squared and rmse

```
% Assuming the variables in the workspace are:
% averageData - 101x1 double vector (average of the 216 columns)
% dataToSuperimpose - 101x1 double vector to be compared with

% Compute the RMSE
n = length(yi_hip);
rmseHip = sqrt(mean((yi_hip - averageDataHip).^2));

% Compute the R-squared
SS_resHip = sum((yi_hip - averageDataHip).^2);
SS_totHip = sum((yi_hip - mean(yi_hip)).^2);
r_squaredHip = 1 - (SS_resHip / SS_totHip);

% Display the results
fprintf('RMSE: %.4f\n', rmseHip);
```

```
RMSE: 2.5010
```

```
fprintf('R-squared: %.4f\n', r_squaredHip);
```

```
R-squared: 0.9508
```

```
% Assuming the variables in the workspace are:
```

```
% averageData - 101x1 double vector (average of the 216 columns)
% dataToSuperimpose - 101x1 double vector to be compared with

% Compute the RMSE
n = length(yi_knee);
rmseKnee = sqrt(mean((yi_knee - averageDataKnee).^2));

% Compute the R-squared
SS_resKnee = sum((yi_knee - averageDataKnee).^2);
SS_totKnee = sum((yi_knee - mean(yi_knee)).^2);
r_squaredKnee = 1 - (SS_resKnee / SS_totKnee);

% Display the results
fprintf('RMSE: %.4f\n', rmseKnee);
```

```
RMSE: 4.6348
```

```
fprintf('R-squared: %.4f\n', r_squaredKnee);
```

```
R-squared: 0.9366
```

```
% Assuming the variables in the workspace are:
% averageData - 101x1 double vector (average of the 216 columns)
% dataToSuperimpose - 101x1 double vector to be compared with

% Compute the RMSE
n = length(yi_ankle);
rmseAnkle = sqrt(mean((yi_ankle - averageDataAnkle).^2));

% Compute the R-squared
SS_resAnkle = sum((yi_ankle - averageDataAnkle).^2);
SS_totAnkle = sum((yi_ankle - mean(yi_ankle)).^2);
r_squaredAnkle = 1 - (SS_resAnkle / SS_totAnkle);

% Display the results
fprintf('RMSE: %.4f\n', rmseAnkle);
```

```
RMSE: 5.6751
```

```
fprintf('R-squared: %.4f\n', r_squaredAnkle); %THERE IS SOMETHING WRONG WITH THIS VALUE, CHECK
```

```
R-squared: -0.0713
```

Export to excel

```
% Define the directory of Excel file name
excelFileName = 'E:/New/DOCUMENTS/Projects/kone/projects/New Folder/raw data/Test/p11/csv_file

% Load your data (this step can be skipped if data is already loaded)
% hipAngle = ...;
% kneeAngle = ...;
% ankleAngle = ...;

% Define the target sheet numbers (starting from sheet 6)
```

```
sheetNumbers = [6, 7, 8, 9, 10, 11, 12];

% Define the data ranges (top-left cell)
% This example starts writing at A1 for each sheet, adjust as needed
range = 'A1';

% Write the hipAngle data to sheet 6
writematrix(hipAngle, excelFileName, 'Sheet', sheetNumbers(1), 'Range', range);

% Write the kneeAngle data to sheet 7
writematrix(kneeAngle, excelFileName, 'Sheet', sheetNumbers(2), 'Range', range);

% Write the ankleAngle data to sheet 8
writematrix(ankleAngle, excelFileName, 'Sheet', sheetNumbers(3), 'Range', range);

% Write the kneeAngle data to sheet 9
writematrix(yi_hip, excelFileName, 'Sheet', sheetNumbers(4), 'Range', range);

% Write the kneeAngle data to sheet 10
writematrix(yi_knee, excelFileName, 'Sheet', sheetNumbers(5), 'Range', range);

% Write the kneeAngle data to sheet 11
writematrix(yi_ankle, excelFileName, 'Sheet', sheetNumbers(6), 'Range', range);

% Write the kneeAngle data to sheet 12
writematrix(timeNormalized, excelFileName, 'Sheet', sheetNumbers(7), 'Range', range);
```