MATLAB PROJECT 0

# This is an Individual Assignment which has to be completed in MATLAB and submitted as a PDF file through the Assignments Canvas page under Project 0. This Project is worth 10 points; 3 bonus points can be earned on this Project (see Page 14).

**You will use Live Script and do all exercises within the Live Editor. Please see below a link to the video that explains how to use Live Editor and how to create a report (PDF file):**

<https://www.mathworks.com/videos/using-the-live-editor-117940.html?s_tid=srchtitle>

**The final script has to be generated by exporting the Live Script to PDF.**

In this Project, you will learn how to use Live Editor and how to work with matrices in MATLAB. You will review some basic programming techniques. You will also learn how to create functions and work with them in MATLAB.

**Please read carefully instructions for each Project posted on the Canvas page Assignments, Project #, before you begin working on the project.**

Using Live Editor

1) Open MATLAB application.   
Note: If you wish to access MATLAB through UF Apps, please refer to the file “Accessing MATLAB off-campus” located on Canvas under Assignments, Project 0.

2) To begin working on Project 0 in MATLAB, choose New Live Script under the Home tab or New →Live Script under the Live Editor tab in MATLAB. When the file opens up, choose the option “Text” under the Live Editor tab and type the title “Project 0”. Then, type the heading “Exercise 1”. Save the file with the name “MAS3114\_Project0” – MATLAB saves it in the Current Folder with an extension .mlx.

You will do all your work on this Project within the Live Script with exception of creating a function – a function has to be created in a separate file which will have an extension .m (please see Exercise 3 of the current project on the subject of creating a function in a file).

When working in Live Script, **it is strongly recommended to insert Section Break after each exercise and/or after each part of an exercise if the exercise is long and contains multiple parts** – this will allow you to run your script smoothly and, if you need to edit just one exercise/part you can do it by re-running that section (click on **Run Section** or **Run and Advance**). If you would like to re-run the whole Project, you could press **Run** or **Run to End**.

3) If you are typing a command (which you will run), you need to switch to the option “Code” in the Live Editor. If you would like to run a command and get an output but you do not want this output to be printed, you should type a semicolon after the command – MATLAB will run the command and store the output, but the output will not be displayed in the Live Script file.

Important: The **general rule** is that we have to display all required outputs and we should suppress the intermediate outputs (in a “for” loop, for example) unless it is specified otherwise in the instructions.

4) If you would like to type a comment in your live Script, you can either switch to the “Text” option, or, if you are using the “Code” option, you can place the sign % at the beginning of the comment – when you run that section, the part that begins with % will not be executed.

5) Editing the Live Script is comparatively easy. You just need to make corrections and re-run the section. If you have made a change in the Live Script, you will need to Save the file.

NOTE on the Format of Project Instructions:

(1) The sign \*\* at the beginning of an Instructional sentence in a Project indicates that this task has to be perform in MATLAB by an appropriate command or a sequence of commands. This sign is not a MATLAB command – it is used in the Instructions to emphasize that the task has to be executed in MATLAB.

(2) The sign **%** at the beginning of an Instructional sentence indicates that you will need to type a comment in the Live Script.

Important Notes: 1) All exercises in this project have to be completed in the Live Script saved with the name “MAS3114\_Project0” using Live Editor.

2) Each exercise should begin with the line: **Exercise #.**

3) You should also mark down the parts of an exercise if it contains multiple parts, such as, Part1, Part2, (a), (b), and etc. - this will make grading of the Project easier.

The link below to the MATLAB website may help you to get started:

<https://www.mathworks.com/help/matlab/index.html>

Exercise 1: Working with matrices in MATLAB

**Grading**: Completed: 3 points; Not Completed/Partially Completed: 0 points

DESCRIPTION: In this Exercise, you will learn the basics on working with matrices in MATLAB. You have already created the Live Script “MAS3114\_Project0”. You will continue working on Exercises 1 – 4 within this file using Live Editor.

**Part 1**: Creating Matrices

A matrix is a rectangular array, and, in linear algebra the entries will usually be the numbers. The most direct way of creating a matrix is to type the numbers between the brackets using a space or comma to separate different entries in a row and using semicolon or **[Enter]** to create row breaks. For example, if you type

A=[0 3 6 9; 1 4 7 10; 2 5 8 11]

and run the script, it will create a variable A in MATLAB and output it as a 3 by 4 matrix A.

\*\*Type the following matrices in the Live Script:

,  ,  , , 

After you type the matrices as they show above and run the script, the outputs will be displayed. Do not suppress outputs in this exercise; remember, an output will be suppressed if you type a semicolon after the command. Also notice that MATLAB is case sensitive.

You can save the variables in a file by typing, for example, *save* Project0. The variables can be reloaded in a future session by typing *load* Project0. As an alternative, you can re-load variables by re-running that Section of the Live Script where they were created.

Please check out the two small icons in the upper right corner of the Live Editor window: if you choose an option “Show outputs inline”, this part of your project will look similar to the Example below; if you choose “Show outputs on the right”, it will change the display. It might be easier to do work in the Live Scrip when the outputs are shown “on the right”, but, after the Live Script is exported to a PDF, the outputs will be automatically displayed “inline”.

Below is an Example how your Project 0 Live Script may look after you typed the given matrices and ran the Section – an option “Show outputs inline” was chosen here.

Project 0

**Exercise 1**

In this Exercise we will learn the basics on working with matrices in MATLAB.

**Part 1. Creating matrices**

A=[0 3 6 9; 1 4 7 10; 2 5 8 11]

A = 3×4  
 0 3 6 9  
 1 4 7 10  
 2 5 8 11

B=[2 -2; 3 -3;4 -4]

B = 3×2  
 2 -2  
 3 -3  
 4 -4

X=[2;4;6]

X = 3×1  
 2  
 4  
 6

x=[2 3 4 5]

x = 1×4  
 2 3 4 5

y=[1;3;5;7]

y = 4×1  
 1  
 3  
 5  
 7

When *M* is a matrix, the command **size**(*M*) returns a vector whose entries are the number of rows and the number of columns in *M***.**

\*\*Calculate the sizes of the matrices A, B, **X**, **x**, **y** in MATLAB.

**%** Are the matrices **x** and **y** of the same size? Explain please.

\*\*Run the commands size(A,1) and size(A,2).

%Comment on the outputs for these two commands.

Note: Command **help** will provide assistance with any command which name you know, for example, **help size.** The most general command is **help help.**

**Part 2**. Accessing particular matrix entries and changing entries

\*\*Type the following commands in MATLAB. (Notice: a new matrix F is introduced here and it will be modified over the execution of some commands below). Run the section.

**%** Briefly describe the result after executing each of the lines below:

**A, A(1,3), A(:,3), A(2,:), A(:, 1:3)**

**A, A([1 2], [2 4])   
F(:,4)=[-1 1 -4 3], F([1 3], [2 3]) = [1 -3; 2 -5]   
A, F, F([2 3], :) = A([1 3], :)   
F, F(:, [1 2])=F(:, [2 1])**

**F, F(:, 2:4)**

\*\*Create the following commands, run them, and output the results.   
Note: the matrix F will be modified after executing the commands below.

(I) Run the commands that will output **y,** F**,** and a new matrix obtained from F by assigning to the first column of F the vector **y** – the last output will be your new matrix F.

(II) Run the commands that will output F from part (I) and create a new matrix obtained from F by switching rows 2 and 4 – it will be your new matrix F.

(III) Run the commands that will output the matrix F from part (II) and create a new matrix F1 formed by the first two rows of F.

**Part 3**. Pasting blocks together

**\*\*** By using matrices **A, B, X,** and **x**, create the following block matrices in MATLAB and display them:

**A, B, [A B], [B A]   
A, X, [A X]   
A, x, [A ; x]**

**Part 4**. Some special functions for creating matrices:

\*\*(a) Type each of the following commands in MATLAB and Run them:

eye(5), zeros(3,4), zeros(2), ones(3,2), ones(5)

diag([1 2 5 6 7]), diag([1 2 5 6 7],1), diag([1 2 5 6 7],-2)

B, diag(B), diag(diag(B))

A, triu(A), tril(A)

F, triu(F), triu(F,1), triu(F,-1)

F, tril(F), tril(F,-1),tril(F,1)

**%**write a brief comment on each of the output matrices

\*\*Run the following commands:  
magic(5), help magic

hilb(5), help hilb

\*\*(b) Use appropriate MATLAB commands to create and display matrices C, D, and E:

  

**Part 5**. Using the colon (vectorized statement) to create a vector with evenly spaced entries

**\*\*** Execute the following commands in MATLAB:

**V1=1:7, V2=2:0.5:6.5, V3=3:-1:-5**

\*\* Type and run the command needed for creating each of the following vectors in colon notation:

**V**4=[-5 - 4 - 3 -2 -1 0 1] , **V**5=[10 7 4 1 -2] , **V**6=[5 4.5 4 3.5 3 2.5 2],

**V**7: the numbers from 0 to 4 spread by 0.4 apart.

**Part 6**. The format commands

**\*\*** Type each of the following commands in MATLAB and execute them:

R=2.46721652

**format long, R, round(R,5)**

**format rat, R**

**format short, R, round(R,3)**

**%** describe the result for each.

The default mode to display the numbers is ***format short***.

To restore the default mode at any time, type ***format***.

**Note**: We will be using ***format short*** to display numbers throughout the Projects unless it is specified otherwise in the instructions. To suppress extra line feeds, you can type the command   
format compact

**Part 7**. Operations on Matrices­­

The operations of matrix addition and scalar multiplication are defined by + and \* signs, respectively.

Note: all the variables in the commands below were introduced earlier. If you don’t have them saved, you can simply re-run your Live Script to have them re-loaded.

\*\*Execute the following commands in MATLAB:

A, A+A

A, 3\*A

F, magic(4), U=F+magic(4)

x, y, x+y

%Write a comment how the matrix x+y has been created.   
Note: matrices x and y do not have matching sizes, and the output was generated by a special MATLAB operation which is not consistent with the Algebra rule for the matrix addition.

\*\*Execute the commands:

A,A',transpose(A)

% write a brief comment on the outputs.

**Part 8**. Matrix multiplication

**P\*Q** indicates the usual matrix multiplication as it is introduced in the Algebra course.

Note: the number of columns in matrix P has to be equal to the number of rows in Q (the inner dimensions have to match).

The operation  **.\***  is an entry-by-entry multiplication (vs. usual matrix multiplication \*).

\*\* Input the matrices:

P=[1 2 3 4; 1 1 1 1], Q=transpose(P)

\*\* Execute the commands:   
**P\*Q, P.\*P, P.^2**

**%**Explain how the entries of the matrices **P\*Q**, **P.\*P**, and **P.^2** were calculated. (You may need to read the section Matrix Multiplication in our textbook.)

\*\*Try to run the command **P\*P.** See why you are receiving an error message. Delete the command and re-run the Section.

\*\*Create a matrix    
and execute the commands **G^2** and **G\*G**. Verify their equivalence by running two logical commands:

G\*G==G^2, isequal(G\*G,G^2)

**Part 9**. Creating matrices with random entries

The command **rand** creates numbers between 0 and 1 that look very random. The command **randi** creates array of random integers.

**\*\***Run the commands:

**rand(4), rand(3,4), randi(100,2), randi(10,2,4), randi([10 40],2,4)**

**%** Describe the dimensions of each matrix and the distribution set of random numbers for each of the commands.

**\*\***Executethe commands: 5\***rand(3)** that creates 3 by 3 matrix with random values between 0 and 5 and -3+5\***rand(3)** that shifts the range of values from -3 to 2.

\*\*Write and execute commands that create matrices of sizes 3 by 2 with random values in the following intervals:

**random numbers between 2 and 8**

**random integer numbers from 40 to 70**.

Exercise 2: Programming Techniques

**Grading:** out of 2 points

**Part 1**. Logical Operations

\*\*Input the matrices:

M=3\*ones(2), N=[0 3; 3 3]

\*\*Run the following commands:  
M==N, M~=N

isequal(M,N), ~isequal(M,N)

%Comment on the outputs for each of the two lines above. Explain the difference between the logical operations M==N and isequal(M,N).

You will find more information on logical operations under the following link:  
<https://www.mathworks.com/help/matlab/logical-operations.html>

**Part 2**. Conditional Statements  
\*\*Run the following in the Live Editor:

x=[2 3 4 5], y=[1;3;5;7], y', size(x), size(y), size(y')

if isequal(size(x),size(y))

disp('matrices x and y are of the same size')

else

disp('the sizes of x and y are different')

end

\*\*Write a conditional “if …elseif…else” statement in the Live Editor that does the following:

It checks if the matrices x and transpose(y) are equal and, if so, it displays a corresponding message; if x and transpose(y) are not equal, it check if they have the same size and displays a corresponding message; if none of the above is true, it displays a message that matrices x and transpose(y) are neither equal nor they have the same size. Run your code in the Live Script.

\*\*Run the following conditional statements. The matrices M and N were introduced earlier.

% #1

if ~isequal(M, N)

disp('M and N are different')

else

disp('M and N are the same')

end

% #2

if M~=N

disp('M and N are different')

else

disp('M and N are the same')

end

% #3

if M==N

disp('M and N are the same')

else

disp('M and N are different')

end

% Which of the examples (1, 2, or 3) is an incorrect coding for determining whether two matrices are equal? Explain the reason why it gives an incorrect output message.   
(Hint: analyze the outputs for Part 1 of the current exercise.)

**Part 3**. “For Loops” and Vectorized Statements

\*\*Run in the Live Script the code given below.   
Notice that the variable L, first, was created as

L=zeros(5);

then, the entries were re-calculated using double “for loop”. The intermediate outputs in “for loop” were suppressed. The resulting matrix L had been output after the “for loop” terminated.

L=zeros(5);

for i=1:5

for j=1:5

L(i,j)=i+j;

end

end

L

%Explain in detail how the entries of the matrix L were calculated.

\*\*Run the two codes given below. Both generate a matrix whose 3 columns are created by raising vector z elementwise to the consecutive powers from 1 to 3. The first code uses “for loop” and the second – a vectorized statement.

z=transpose(1:5);

ZF=zeros(5,3);

n=size(ZF,2);

for i=1:n

ZF(:,i)=z.^i;

end

ZF

z=transpose(1:5);

ZV=zeros(5,3);

n=size(ZF,2);

i=1:n;

ZV(:,i)=z.^i

%Compare the two codes and analyze the difference between them.   
Hint: you may need temporarily display the intermediate outputs in the “for” loop (remove the semicolon after ZF(:,i)=z.^i) to see the way it works – do not forget to put the semicolon back in after you are done and re-run the script with the intermediate outputs suppressed.

\*\*Generate and display the matrix hilb(5) in three ways: (a) by running H=hilb(5); (b) by using double “for loop” – output the matrix HF, (c) by using a vectorized statement – output the matrix HV.   
Hint: for part (c), see Exercise1 (**Part 5** and the command **x + y** in **Part 7)**; also see *help hilb*  
  
Use the following link for more information on the loops and conditional statements:  
<https://www.mathworks.com/help/matlab/control-flow.html>  
You may also find it useful to visit this link in MATLAB:  
<https://www.mathworks.com/help/matlab/operators-and-elementary-operations.html>

Exercise 3: **Create and Call a Function in MATLAB  
Grading**: out of 2 points   
  
In this exercise, you will learn how to create a function in a file in MATLAB. The first function that you will create has a name **closetozeroroundoff**. We will be working with this function on a regular basis, and you should have it in your Current Folder in MATLAB. This function is needed because of the round-off errors in MATLAB calculations. The rule we employ here is that we will consider an entry of an array equal to zero if the absolute value of that entry is less than 10^(-p), where the integer number p is a parameter that defines a margin for an error – it will be specified for each particular exercise, but, in most cases, we will have p=7. After you have this function created and saved in your Current Folder in MATLAB, you can run it as a usual MATLAB function, such as, *size*, *transpose*, and etc.

\*\*To create a function in a file, click on New → Function in the Live Editor tab. It will open up a file “Untitled” in your Live Editor window. You should delete the preset lines and copy and paste the code given below:

function B=closetozeroroundoff(A,p)

A(abs(A)<10^-p)=0;

B=A;

end

Then, you will need to save the file with the name of the function *closetozeroroundoff*   
The file will have an extension .m

Please note: you will have a separate file for each function that you create.

\*\*Next, type in the Live Script:

format compact

p=1

H=hilb(8)

and run the function   
closetozeroroundoff(H,p)

%Write a comment that explains how the function closetozeroroundoff. worked on the matrix H. Did it modify all entries of H? Which entries of H were replaced with zeros?

\*\*Run the command

type closetozeroroundoff

This command will print the function in your Live Scrip.

Note: in order to run a function, which we created in a file and saved in the Current Folder in MATLAB, we do not need to have it “typed” previously – we use the command type to display the code in the Live Script.

\*\*Create another function in a file called produc. The code is below:

%The function calculates the product of two matrices A and B, when it is

%defined. Each column of the matrix AB is calculated as a product of the

%matrix A and the corresponding column of the matrix B. The output AB is

%compared with the output of the MATLAB built-in function A\*B. Then, we

%demonstrate that the property of invertible matrices holds: the inverse

%of the product of two matrices inv(A\*B) is equal to the product of the

%inverses but in the reverse order: inv(B)\*inv(A.

function [AB,InvAB,InvBInvA] = produc(A,B,p)

%calculating the sizes:

[m,n]=size(A);

[k,q]=size(B);

%making assignments:

AB=[];

InvAB=[];

InvBInvA=[];

%checking if the matrices inner dimensions agree for multiplication

if isequal(n,k)

disp('the matrices dimensions agree for matrix multiplication')

%calculating the product of A and B:

i=1:q;

AB(:,i)=A\*B(:,i);

fprintf('the product of two matrices A and B is\n')

AB

%verifying if the code is correct by running matlab function A\*B

C=zeros(m,q);

fprintf('the output for matlab function A\*B is\n')

C=A\*B;

disp(C)

if isequal(AB,C)

fprintf('the output AB is the same as the output for A\*B\n')

else

disp('check the code!')

return

end

%The next task applies to square matrices A and B of the same size

%which are also invertible. We demonstrate that the following

%property holds: inv(A\*B)=inv(B)\*inv(A)

if m==n && k==q && rank(A)==m && rank(B)==k

disp('A and B are invertible matrices of the same size')

fprintf('the inverse of A\*B is\n')

InvAB=inv(AB)

fprintf('product of inverses in reverse order inv(B)\*inv(A) is\n')

InvBInvA=inv(B)\*inv(A)

if closetozeroroundoff(InvAB-InvBInvA,p)==0

fprintf('inv(A\*B)=inv(B)\*inv(A) holds for the given A and B\n')

fprintf('and within the given precision 10^(-%i)\n',p)

else

fprintf('inv(A\*B)=inv(B)\*inv(A) does not hold for the given\n')

fprintf('A and B within the given precision 10^(-%i)\n',p)

end

end

else

fprintf('the matrices dimensions disagree for matrix multiplication\n')

end

end

**Note on the Format of the Function**: The comment lines in the body of the function, which begin with the sign %, are optional – you may have some similar comments or omit them entirely in your future functions. However, the messages, which are output by the commands disp, fprintf, and etc., have to be present in the code in the way it will be specified in the Instructions.

\*\*Print the function produc in your Live Script using the command

type produc

\*\*Next, type in the Live Script (input the variables):

p=7;

%(a)

A=magic(3), B=hilb(3)

and run the function:

[AB,InvAB,InvBInvA]=produc(A,B,p);

**Note**: Here, we have semicolon placed at the end of the command line to suppress extra outputs for the variables located in the name of the function. The outputs for those variables have been already coded to be displayed. Without semicolon, MATLAB would automatically display them one more time at the end. Placing or omitting semicolon at the end a command that runs the function gives a control over displaying those outputs.

**Important Note**: The outputs for the variables located in the name of the function always have to be assigned and displayed. There may be some other outputs that are required to be displayed. However, all required outputs have to be displayed only once. **Having extra outputs or missing required inputs/outputs will result in a loss of points on the Project**. Empty matrices can be assigned as outputs for the variables; however, empty outputs should be suppressed – they don’t need to be displayed in the Live Script

\*\*Then, run a logical command:

InvAB==InvBInvA

%Compare the output of the last command with the corresponding output of the function produc. Explain the reason why we may need to use the function closetozeroroundoff in our codes to compare the matrices; also, note on the way that function has been used.

\*\*Run the function produc on the variables given below. (You can just copy and paste the text below into your Live Script and Run Section.)

%(b)

A=magic(4), B=ones(4)

[AB,InvAB,InvBInvA]=produc(A,B,p);

%(c)

A=magic(5), B=ones(5,4)

[AB,InvAB,InvBInvA]=produc(A,B,p);

%(d)

A=magic(5), B=ones(4,5)

[AB,InvAB,InvBInvA]=produc(A,B,p);

%(e)

A=magic(5), B=hilb(5)

[AB,InvAB,InvBInvA]=produc(A,B,p);

%Analyze the output message in part (e) which relates to the property of the inverse matrices and compare it with the corresponding output message for part (a). Write a comment about the difference between them.

\*\*Input the new variable

p=6;

and re-run the function produc for the new parameter p on the set of the variables in part (e).

%Write a comment on the output message relating to the property of the inverses with the adjusted parameter p.

Exercise 4: **Working with a Function in Live Editor**

**Grading**: out of 3 points

In this exercise you will calculate the **dot (inner) product** of two vectors, when it is possible, by using three methods. Then, you will use the dot product to find the angle between two non-zero vectors.

\*\*Create a function dotangle in a file. The function begins with:

function [d1,d2,d] = dotangle(u,v)

The input vectors **u** and **v** will be the column vectors. The output variables d1, d2, and d represent the dot product calculated in three different ways.

All of the following has to be included into the body of the function dotangle:

\*\*First, we calculate the numbers of entries of the vectors **u** and **v**, which will be *m* and *n*, respectively, but we do not display them (use semicolon). To find the numbers of entries in **u** and **v**, you could employ, for example, a MATLAB function **length**.

\*\*Next, we will check using a conditional “if” statement whether **u** and **v** have the same numbers of entries.

If the numbers of entries are the same, we code a message:  
fprintf('both vectors have %i entries\n',n)

(this command will also print the number of entries in each vector).

If the numbers of entries are different, we program a message:

disp('the dot product is not defined')

and, in this case, the outputs should be assigned as below:  
d1=[];

d2=[];

d=[];

and the code has to be terminated. To terminate a code, you can compose a conditional statement in a specific way **or** use the command   
return

\*\*When *m* and *n* are the same, the code will continue. We compute, output, and display the numbers d1, d2, and d – these numbers give the value of the dot product calculated in three different ways:   
(a) The formula for d1 uses a definition of the dot product of two vectors **u** and **v** as **transpose(u)\*v**

(b) To compute the number d2, we use a definition of the dot product as a sum of the products of the corresponding entries in the vectors **u** and **v**.

Hint: to calculate d2, you could begin with initializing the sum, d2=0; and, then, employ “for loop” to calculate the required sum.

(c) To compute the number d, we use a MATLAB built-in function:

d=dot(u,v)

\*\*Next, we will verify using a conditional statement, whether all outputs d1, d2, and d are the same. If it is the case, output the message:  
fprintf('the code is correct\n')

otherwise, your code should output

disp('check the code!')

and the program terminates.

\*\*If your code passes the “check point”, that is, all three values d, d1, and d2 are the same, you will continue with determining the angle *theta* (in degrees) between non-zero vectors **u** and **v** – the angle *theta* has to be in the range from 0 to 180 degrees.

Hint: to find the angle between two non-zero vectors **u** and **v**, you may employ another definition of the dot product, namely, , where  is the angle between the vectors **u** and **v** and  (or) is the magnitude of **u** (or **v**), calculated by using the MATLAB function *norm*.

\*\*You will need to calculate and output the angle *theta* in degrees from the formula above (a MATLAB function acosd can be used), but suppress this output (place a semicolon).

\*\*Then, we will code (in the order specified) the following five possibilities and the corresponding output messages for the calculated *theta*:   
(1) If *theta* is within 10^(-5) from 90 degrees, output a message the angle between the vectors is 90 degrees.

(2) If *theta* is within 10^(-5) from 0 degrees, output a message the angle between the vectors is zero.

(3) If *theta* is within of 10^(-5) from 180 degrees, output a message the angle between the vectors is 180 degrees.

If none of the conditions (1)-(3) holds, consider the last two:

(4) If *theta* is less than 90 degrees, output a message angle between vectors is acute and its value theta= (display the calculated *theta*)

(5) If *theta* is greater than 90 degrees, output a message angle between vectors is obtuse and its value theta=(display the calculated *theta*)

Hints: You will need to use a conditional statement to program all 5 possible outcomes.   
For the cases (1)-(3), use closetozeroroundoff function with the parameter p=5.

To display the messages, employ a MATLAB built-in function fprintf.

**This will be the end of your dotangle function**. Save it in the Current Folder in MATLAB (with an extension .m).

\*\*Next, we return to the Live Script and type the command:

type dotangle

This command will have your function printed in the Live Script.

\*\*Then, you will input the first set of variables and run the function:

%(a)

u=randi(15,5,1), v=randi(15,4,1)

[d1,d2,d]=dotangle(u,v);

\*\*You will repeat this step several more times for the sets of variables given below (you can just copy and paste the text below and press Run Section to receive the outputs).

%(b)

u=randi(15,5,1), v=randi(15,5,1)

[d1,d2,d]=dotangle(u,v);

%(c)

u=u, v=-v

[d1,d2,d]=dotangle(u,v);

%(d)

u=u, v=2\*u

[d1,d2,d]=dotangle(u,v);

%(e)

u=u, v=-3\*u

[d1,d2,d]=dotangle(u,v);

%(f)

u=[1;3],v=[-3;1]

[d1,d2,d]=dotangle(u,v);

**This is the END of Exercise 4**

Some Additional Important Comments

bonus! (3 points)

A student can earn 3 bonus points for completing Project 0 perfectly. This includes:

1) All exercises are completed.

2) The file is submitted for grading before the due date.

3) No missing inputs/outputs.  
4) All required inputs, outputs, and messages are displayed.

5) No extra conditions or commands.

6) No extra inputs/outputs.

7) No repetitions in the code.   
8) All necessary comments are present.

9) The formatting is as required.

**Publishing Project0 file**

After you are finished working on the Live Script “MAS3114\_Project0, you will need to export it to a PDF file. To create a PDF file, click on Save under the Live Editor tab and choose an option Export to PDF. Save the PDF file with the same name “MAS3114\_Project0”.

**Important:** You will need to Save the PDF file MAS3114\_Project0 onto your computer and submit it for grading prior to the due date on the Canvas page Assignments → Project 0. If you are using MATLAB through UF Apps and need help on working with the files and saving them onto your computer, please refer to the file

**Frequently Asked Questions on UF Apps (PDF)**

which is located under the Resources on Canvas page Assignments → Project 0.

If you need to combine several PDF file into a single PDF document, you can use the link

<https://combinepdf.com/>

**Very Important**: Before submitting your PDF file for grading, it is important to verify that all required inputs/outputs are presented and displayed properly. Correcting and resubmitting the file may result in a loss of a point or cannot be done at all after the final deadline.

**After the final deadline, Project 0 cannot be submitted or resubmitted. We will grade the LAST submission using the Rubric located on the Canvas page Assignments → Project 0.** **If there is no submission, a zero grade will be assigned for the Project.**