

This programming assignment may seem really long. However, we are giving a very detailed explanation to help you become familiar with MATLAB®'s languageM and the way we will be building a small library of functions.

This particular exercise does everything for you if you follow along with the video and type in the code that is discussed there. In future exercises you will do more of the programming yourself.

Preliminaries

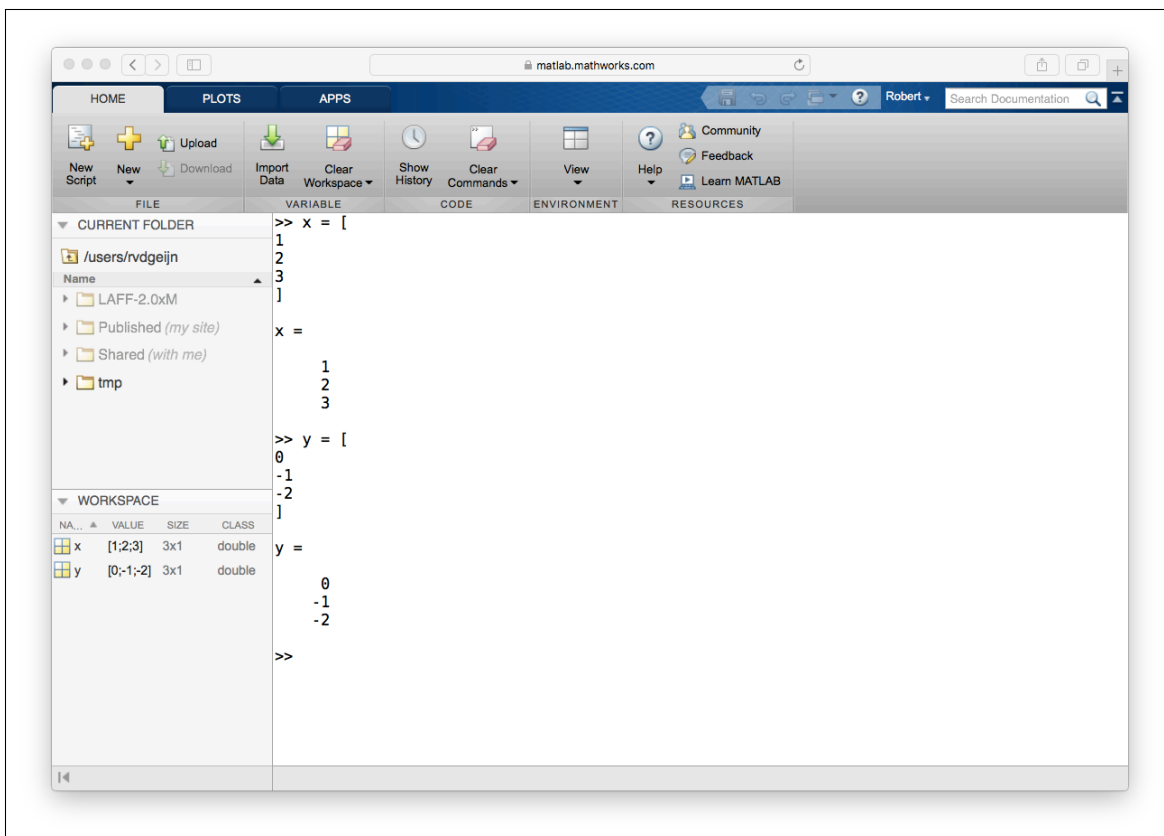
Start MATLAB® and in the Command Window start by creating vectors

$$x = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} \quad \text{and} \quad y = \begin{pmatrix} 0 \\ -1 \\ -2 \end{pmatrix}.$$

Type

```
1 >> x = [  
2 1  
3 2  
4 3  
5 ]  
6 >> y = [  
7 0  
8 -1  
9 -2  
10 ]
```

into the Command Window:



One could instead type

```
x = [1;2;3];  
y = [0;-1;-2];
```

This still creates column vectors. The “;”s separate rows in x and y) and the “;” at the end of the line suppresses the printing of the result. However, we usually like to emphasize that x and y are column vectors by entering them as columns.

At this point, you may want to have a look at what is in variables x and y :

```
1 >> x  
2  
3 x =  
4  
5     1  
6     2  
7     3  
8  
9 >> y  
10  
11 y =  
12  
13     0  
14    -1  
15    -2
```

Now, copying one vector into another is easy with MATLAB®:

```
1 >> x_old = x  
2  
3 x_old =  
4  
5     1  
6     2  
7     3  
8  
9 >> y_old = y;  
10 >>
```

Notice how if one adds “;” after a command, the result of that command is *not* printed.

We have created new variables x_old and y_old because this will make it convenient later to restore the contents of x and y .

Indexing into a vector

In our course we index starting at 0:

$$x = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} = \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix}$$

so that in this example $\chi_0 = 1$, $\chi_1 = 2$, and $\chi_2 = 3$. MATLAB[®] instead starts indexing at 1:

```
1 >> x( 1 )
2
3 ans =
4
5     1
```

Mathematicians typically start indexing at 1 while computer scientists typically start at 0. The famous computer scientist Edsger W. Dijkstra used to wear a T-shirt that declared “Counting starts at zero”. Get used to it! What you will find is that later this week we introduce a way of referencing parts of vectors and matrices that mostly avoids the whole indexing of individual elements issues.

The point of most of our exercises that use MATLAB[®] is to link coding algorithms to the abstractions that we use. An abstraction on the math side is the notion of the vector, x , as an ordered list of numbers that we call elements and that we index starting at 0: $\chi_0, \dots, \chi_{n-1}$. The corresponding abstraction in MATLAB[®] is the array x that we index into starting at 1: $x(1)$, $x(2)$, $x(3)$.

On the math side, we can assign the elements of a vector x to another vector y . To do so, the sizes of vectors x and y have to be equal and then assigning $y := x$ (y becomes x) means that

$$\begin{aligned}\psi_0 &:= \chi_0 \\ \psi_0 &:= \chi_0 \\ &\vdots \\ \psi_{n-1} &:= \chi_{n-1}\end{aligned}$$

After the assignment $y := x$, the equalities

$$\begin{aligned}\psi_0 &= \chi_0 \\ \psi_0 &= \chi_0 \\ &\vdots \\ \psi_{n-1} &= \chi_{n-1}\end{aligned}$$

hold. Another way of saying this uses the “for all” *quantifier* (universal quantifier):

$$\forall_{i=0}^{n-1} (\psi_i = \chi_i),$$

which one should read as “ ψ_i equals χ_i for all indexes i from 0 to $n - 1$.” This is shorthand for

$$\psi_0 = \chi_0 \quad \text{and} \quad \psi_1 = \chi_1 \quad \text{and} \quad \dots \quad \text{and} \quad \psi_{n-1} = \chi_{n-1}.$$

We can code the assignment of one vector to another as

```
1 >> y = x;
```

In this case, the output array y is automatically made of the same size as x . If we want to more explicitly assign individual elements, we could instead execute

```
1 >> y( 1 ) = x( 1 );
2 >> y( 2 ) = x( 2 );
3 >> y( 3 ) = x( 3 );
```

Using a for loop

Simple assignment works fine if x and y have only a few elements. But if they are long and/or you don't know how many elements they have, you may want to write a *loop*:

```
1 >> for i=1:3
2   y( i ) = x( i )
3 end
4
5 y =
6
7     1
8    -1
9    -2
10
11
12 y =
13
14     1
15     2
16    -2
17
18
19 y =
20
21     1
22     2
23     3
```

The “for all” abstraction in the math translates to a `for` loop in MATLAB[®]. The array `y` is printed every time. This can be avoided by executing

```
1 >> for i=1:3
2   y( i ) = x( i );
3 end
4 >> y
5
6 y =
7
8     1
9     2
10    3
```

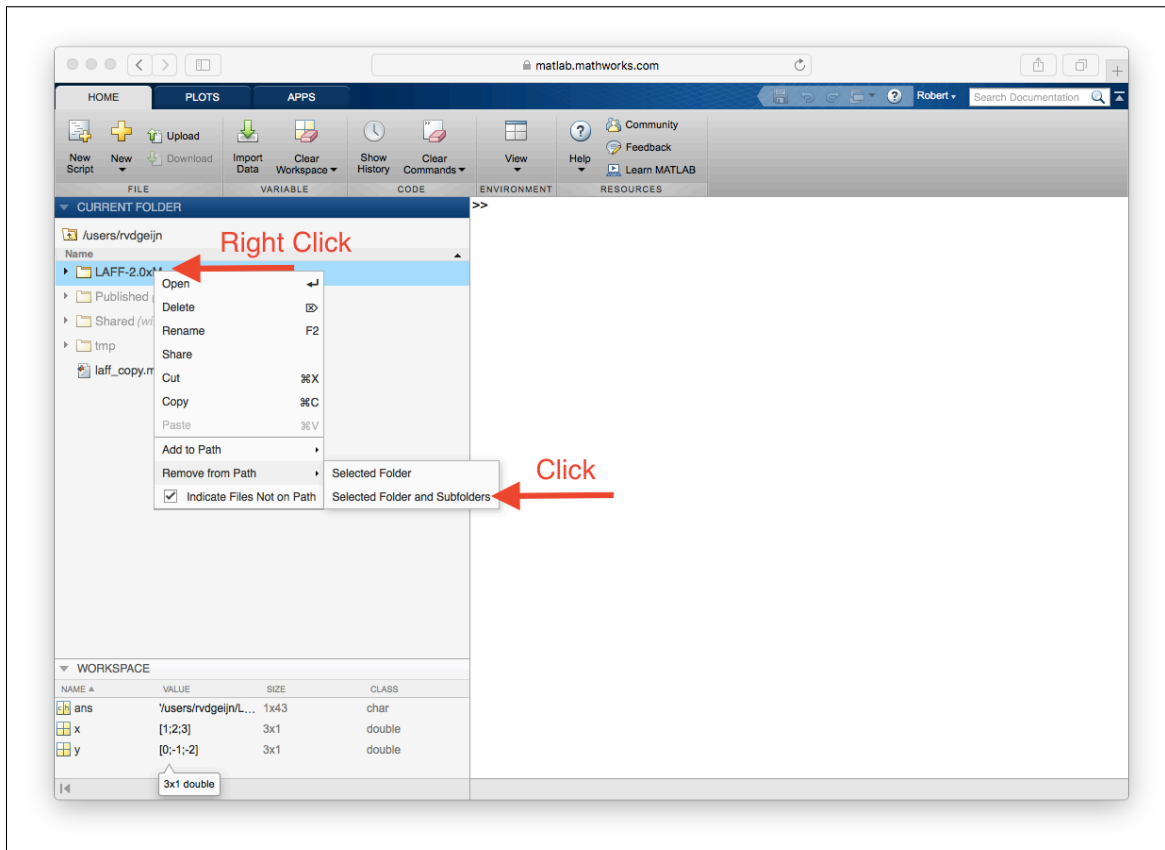
instead (notice the “;”s).

Notice how the “for all” abstraction $\forall_{i=0}^{n-1}$ translates into the MATLAB[®] loop

```
1 for i=1:n
2     ...
3 end
```

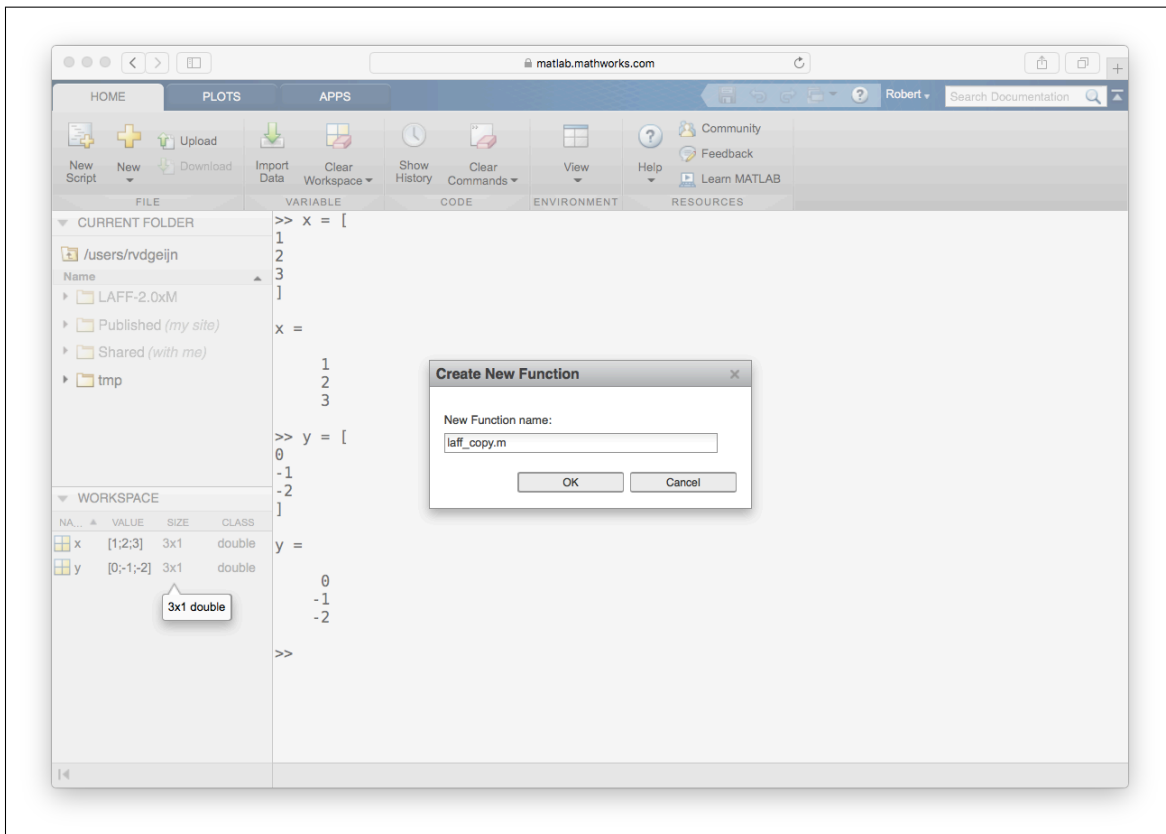
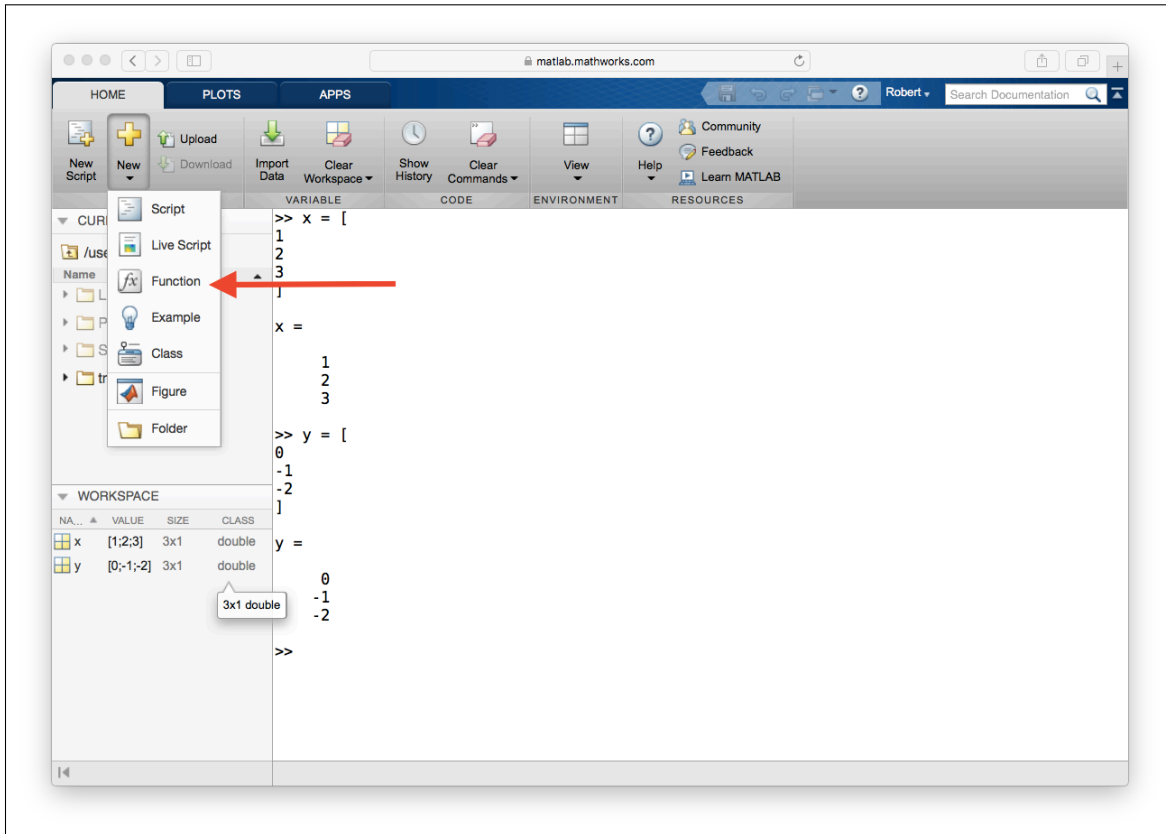
Functions

IMPORTANT. MATLAB has a whole bunch of “intrinsic” functions in various directories. If you are curious, you can type “path” in the Command Window to see all the directories in which there are files that are on the “path” meaning that they are “known” when you work in the Command Window. Now, when you unzipped LAFF-2.0xM, that directory and all its subdirectories were also placed on the path. Unfortunately, some of the names of functions will conflict with functions that you will be writing. For this reason, you will want to start by first removing LAFF-2.0xM from the path. You do this by right-clicking on the directory LAFF-2.0xM and then choosing the indicated option:

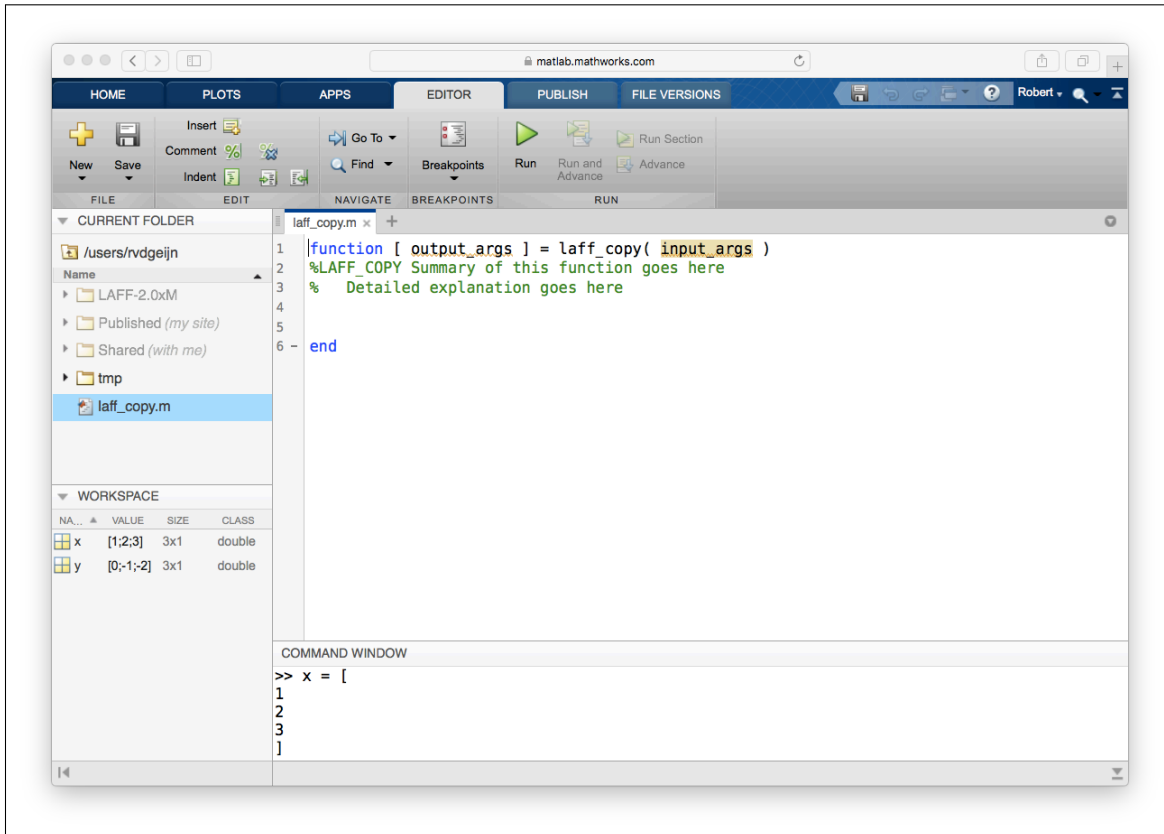


You will notice that LAFF-2.0xM turns gray. Any directory or file that is gray is NOT on the path.

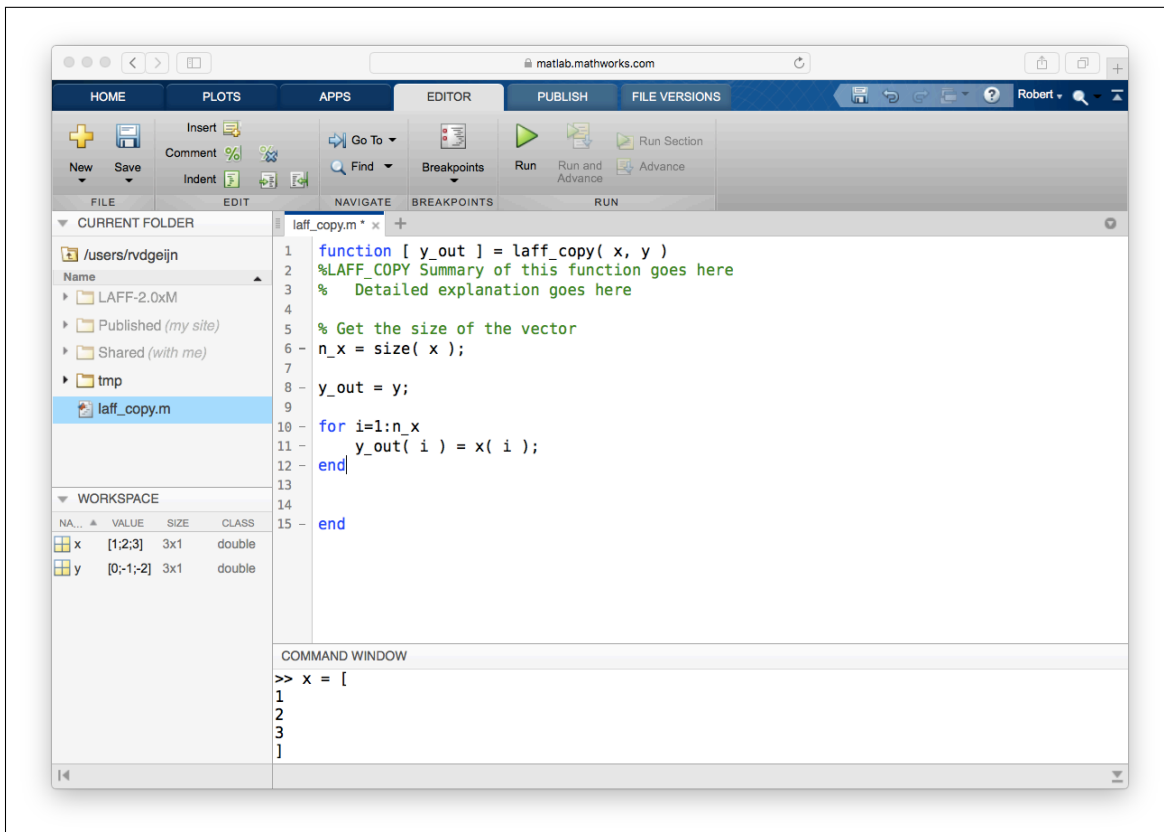
Let us pretend that MATLAB[®] does not have the built-in (intrinsic) ability to copy one array to another by executing $y = x$. It sure would be inconvenient to always write a loop every time we wanted to copy one array to another. For this reason, programming languages typically include the ability to create functions. To do so, create a new function in MATLAB[®]:



yielding

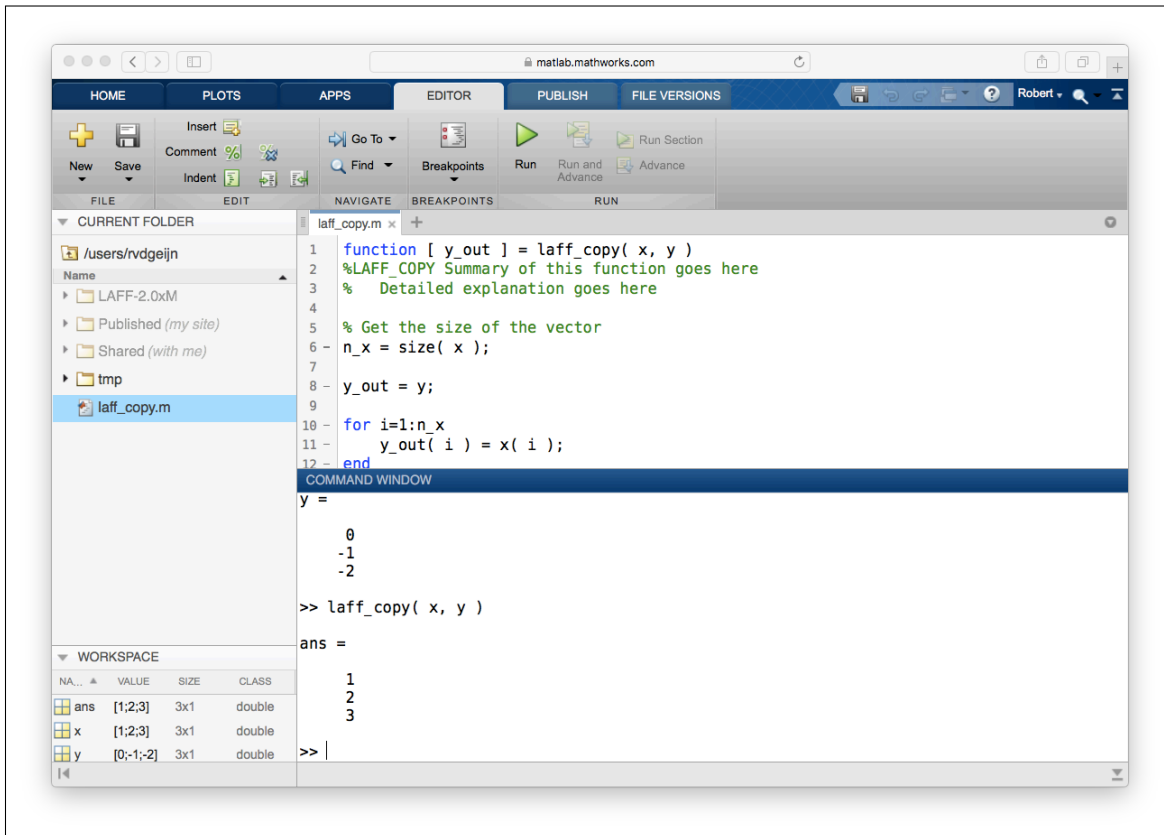


We now create a function that copies one vector to another:



which is now saved in `laff_copy.m`.

Execute `y = laff_copy(x, y)` in the Command Window:



We are going to use the `laff_copy` routine to copy not just column vectors to column vectors, but also row vectors to column vectors, row vectors to row vectors, and column vector to row vectors. Adding this functionality to the function we just wrote makes the function a bit more complex, as illustrated in Figure 1. Here we notice that in MATLAB® vectors are special cases of matrices, which have row and column sizes. Thus, treating `x` and `y` as arrays that store matrices (that happen to be column or row vectors), we can extract the row and column size (e.g., `m_x` and `n_x` for `x`) with the MATLAB® function `size`. This can then be used to determine whether `x` and/or `y` are row and/or column vectors.

Testing

Change the directory in which you are working to `LAFF-2.0xM -> Programming -> Week01`. You do so by *double clicking* on that directory. This means that the Command Window views that directory as the current directory meaning that any `".m"` file there (and associated function) is used before any such function that is elsewhere in the path. You can check what the Command Window considers to be the current directory by typing `pwd` (print current directory) in the command window.

We now test this routine. In the Command Window type

```
>> x = [
1
2
3
];

>> y_old = [
0
-1
```



```

1 function [ y_out ] = laff_copy( x, y )
2
3 % y = copy( x, y ) copies vector x into vector y
4 %   Vectors x and y can be a mixture of column and/or row vector.  In other
5 %   words, x and y can be n x 1 or 1 x n arrays.  However, one size must
6 %   equal 1 and the other size equal n.
7
8 % Extract the row and column sizes of x and y
9 [ m_x, n_x ] = size( x );
10 [ m_y, n_y ] = size( y );
11
12 % Make sure x and y are (row or column) vectors of equal length
13 if ( m_x ~= 1 & n_x ~= 1 ) | ( m_y ~= 1 & n_y ~= 1 )
14     y_out = 'FAILED';
15     return
16 end
17 if ( m_x * n_x ~= m_y * n_y )
18     y_out = 'FAILED';
19     return
20 end
21
22 if ( n_x == 1 )      % x is a column vector
23     if ( n_y == 1 )  % y is a column vector
24         % Copy the elements of x into the elements of y
25         for i=1:m_x
26             y( i,1 ) = x( i,1 );
27         end
28     else              % y is a row vector
29         % Copy the elements of x into the elements of y
30         for i=1:m_x
31             y( 1,i ) = x( i,1 );
32         end
33     end
34 else                  % x is a row vector
35     if ( n_y == 1 )  % y is a column vector
36         % Copy the elements of x into the elements of y
37         for i=1:n_x
38             y( i,1 ) = x( 1,i );
39         end
40     else              % y is a row vector
41         % Copy the elements of x into the elements of y
42         for i=1:n_x
43             y( 1,i ) = x( 1,i );
44         end
45     end
46 end
47
48 % Return the updated y in y_out
49 y_out = y;
50
51 return
52 end

```

Figure 1: Implementation of `laff_copy` that can copy a row or column vectors x to row or column vectors y . For the complete routine, see `LAFFSpring2015 -> Code -> laff -> vecvec -> laff_copy.m`.

```

10 -2
11 ];
12
13 >> z_old = [
14 4
15 3
16 2
17 1
18 ]

```

This gives us a couple of vector with which to check the behavior of our function. Next, try the following:

```

1 >> y = y_old;
2 >> y = laff_copy( x, y )

```

Does it give the expected output? Go on and also try

```

1 >> y = y_old';
2 >> y = laff_copy( x, y )

```

Here `y_old'` (notice the `'`) creates a row vector from the column vector `y_old`. This is known as *transposing* the (column) vector, which yields a row vector. Did you get the expected output? Finally, try

```

1 >> x_row = x';
2 >> y = y_old;
3 >> y = laff_copy( x_row, y )
4 >> y = y_old';
5 >> y = laff_copy( x_row, y )
6 >> z = z_old;
7 >> z = laff_copy( x, z )

```

With this last command, you should get a warning about the sizes of the vectors not matching by `z` equaling `'FAILED'`.

Scripts

What if you had made a mistake in the implementation of `laff_copy`? You would have had to test an updated version all over again, executing all the commands to do so. Wouldn't it be much more convenient to have a file of commands that you can execute? This is called a *script*. To create one, you either use your favorite editor, or you can click on **New Script**. You may want to enter the commands that we give in Figure 2. Alternatively, copy the file `... -> laff -> vecvec -> test_copy.m`. Save this script in `test_copy.m` (don't forget the `.m`) in directory `... ->` and then in the Command Window execute

```

1 >> test_copy

```

While we use `isequal` to check whether two vector are equal in `test_copy`, one must be careful using this function: later in the course we will work with vectors that are real valued. In a computer, these are stored as floating point numbers. This means that is often the case that two vectors would have been equal in exact arithmetic, but due to round off error that is incurred when computing with floating point numbers, they may not be exactly equal.

```

1 % Create some vectors
2 x = [ 1; 2; 3; ]
3 y = [ 0; -1; -2 ]
4 z = [ 4; 3; 2; 1 ]
5
6 % test column -> column copy
7 disp( 'column -> column copy' )
8 if ( isequal( laff_copy( x, y ), x ) )
9     disp( 'PASSED' )
10 else
11     disp( 'FAILED' )
12 end
13
14 % test column -> row copy
15 disp( 'column -> row copy' )
16 if ( isequal( laff_copy( x, y' ), x' ) )
17     disp( 'PASSED' )
18 else
19     disp( 'FAILED' )
20 end
21
22 % test row -> column copy
23 disp( 'row -> column copy' )
24 if ( isequal( laff_copy( x', y ), x ) )
25     disp( 'PASSED' )
26 else
27     disp( 'FAILED' )
28 end
29
30 % test row -> row copy
31 disp( 'row -> row copy' )
32 if ( isequal( laff_copy( x', y' ), x' ) )
33     disp( 'PASSED' )
34 else
35     disp( 'FAILED' )
36 end
37
38 % test column -> column copy (wrong size)
39 disp( 'column -> column copy (wrong size)' )
40 if ( isequal( laff_copy( x, z ), 'FAILED' ) )
41     disp( 'PASSED' )
42 else
43     disp( 'FAILED' )
44 end
45
46 % test column -> row copy (wrong size)
47 disp( 'column -> row copy (wrong size)' )
48 if ( isequal( laff_copy( x, z' ), 'FAILED' ) )
49     disp( 'PASSED' )
50 else
51     disp( 'FAILED' )
52 end
53 %%% MORE TESTS HERE %%%

```

Figure 2: Test script for `laff_copy`. (The complete script can be found in LAFFSpring2015 -> Code -> `laff -> vecvec -> test_copy.m`.)

Summary

We have now created a routine `laff_copy` that copies one vector to another as well as a script to test it. In the process, you have learned a few things about MATLAB®.

If anything in MATLAB® that we did puzzles you, try using `help` in the MATLAB® Command Window:

```
>> help for
>> help if
>> help ==
>> help isequal
```

function is

```
y_out = laff_copy( x, y ).
```

The reasons are

- The input parameter `y` indicates whether the output variable `y_out` is a row or a column vector.
- Almost always, we will use this routine to overwrite an existing (row or column) vector with the (row or column) vector `x`. Thus, the typical use is

```
y = laff_copy( x, y ).
```

- If we were to use a language like the C programming language, as would be common practice in scientific computing, then the routine would have been called as

```
laff_copy( x, y )
```

where the vector `y` would be overwritten with vector `x`.