ECE 1050: Lab M1
MATLAB Basics

Learning Objectives:

- Get acquainted with laboratory protocols.
- Learn the basics of MATALB via the Onramp tutorial.
- Use MATLAB to calculate series and parallel resistances.

Name: ____________________ Date: ____________________

Laboratory Protocol

Welcome to your first hands-on MATLAB exercise! Before we begin with our tutorial, it is important to introduce the basic procedures and expectations of these sessions. For starters, you may have noticed that every page of this document has a space for you to sign your name and date it. The reason for this is to instill a habit of signing/dating every page in official laboratory notebooks. Although this class does not require students to maintain a proper lab notebook, it will eventually become a regular requirement in later classes.

To receive full credit, simply print out this document and fill in the blanks wherever indicated. You will also be asked to scan this document, along with any other supplemental documents (e.g., code, derivations, etc.), and then combine everything into a single PDF file. Please submit your PDF to the appropriate CANVAS page online so that the TA may grade it accordingly.

Note that many smart phones have free scanner-to-PDF apps available. Alternatively, Adobe products are freely available to University of Utah students through the Office of Software Licensing (OSL.utah.edu). If you would rather not print out a hard copy, Adobe Acrobat Pro will allow you to type directly into this PDF and then add your code to the end of the document. For Mac users, you should be able to accomplish the same thing with Preview.

Students are strongly encouraged to work together and help each other solve problems. However, each student must also perform their own work for each lab submission. That includes simulations, building circuits, taking data, writing code, submitting results, etc. Simply copying/pasting other people's work is plagiarism, and it may result in course failure if discovered.

Remember that your TA is here to help you. If you get stuck, need help, or have any other concerns, always feel free to ask questions!

Name: Hayden Walpole

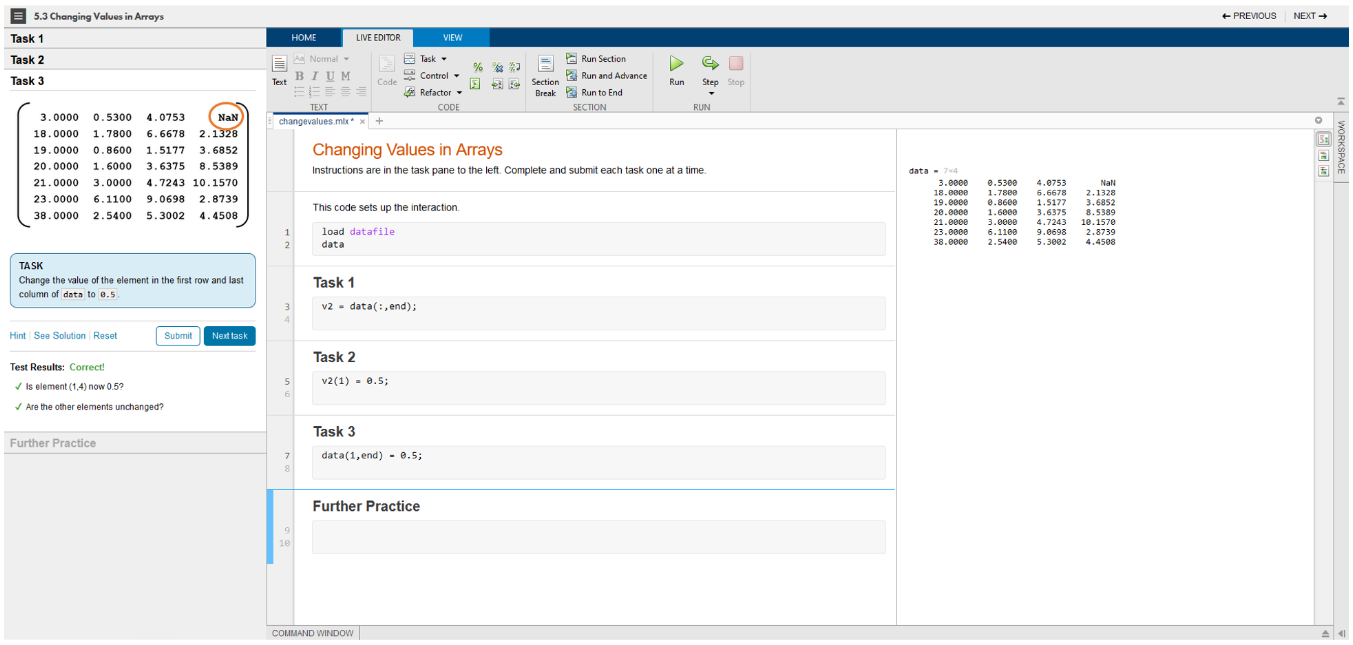
Date: Sept 26,2023

Part 1: MATLAB Onramp (20 Points)

Follow the link on the CANVAS page to the MATLAB Onramp course. Note that you may need to create a Mathworks user account to do this. Onramp is a special tutorial program designed to get you acquainted with the basic functionality of MATLAB. You do **not** require access to a MATLAB installation in order to access Onramp. It is all web-based.

Once you have Onramp up and running, follow the tutorials until you have completed Project 5 ("Indexing into and Modifying Arrays"). This should take you approximately one hour. Don't worry about Project 6 and Project 7. We will finish those next time.

Upon completion, take a screen shot of the webpage. It should look similar to the image below. Include this screen shot in your submission to verify completion of the course.



```
 3.0000   0.5300   4.0753     NaN
18.0000   1.7800   6.6678   2.1328
19.0000   0.8600   1.5177   3.6852
20.0000   1.6000   3.6375   8.5389
21.0000   3.0000   4.7243  10.1570
23.0000   6.1100   9.0698   2.8739
38.0000   2.5400   5.3002   4.4508
```

TASK
Change the value of the element in the first row and last column of `data` to `0.5`.

Test Results: Correct!
✓ Is element (1,4) now 0.5?
✓ Are the other elements unchanged?

Further Practice

Changing Values in Arrays
Instructions are in the task pane to the left. Complete and submit each task one at a time.

This code sets up the interaction.

```
load datafile
data
```

Task 1

```
v2 = data(:,end);
```

Task 2

```
v2(1) = 0.5;
```

Task 3

```
data(1,end) = 0.5;
```

Further Practice

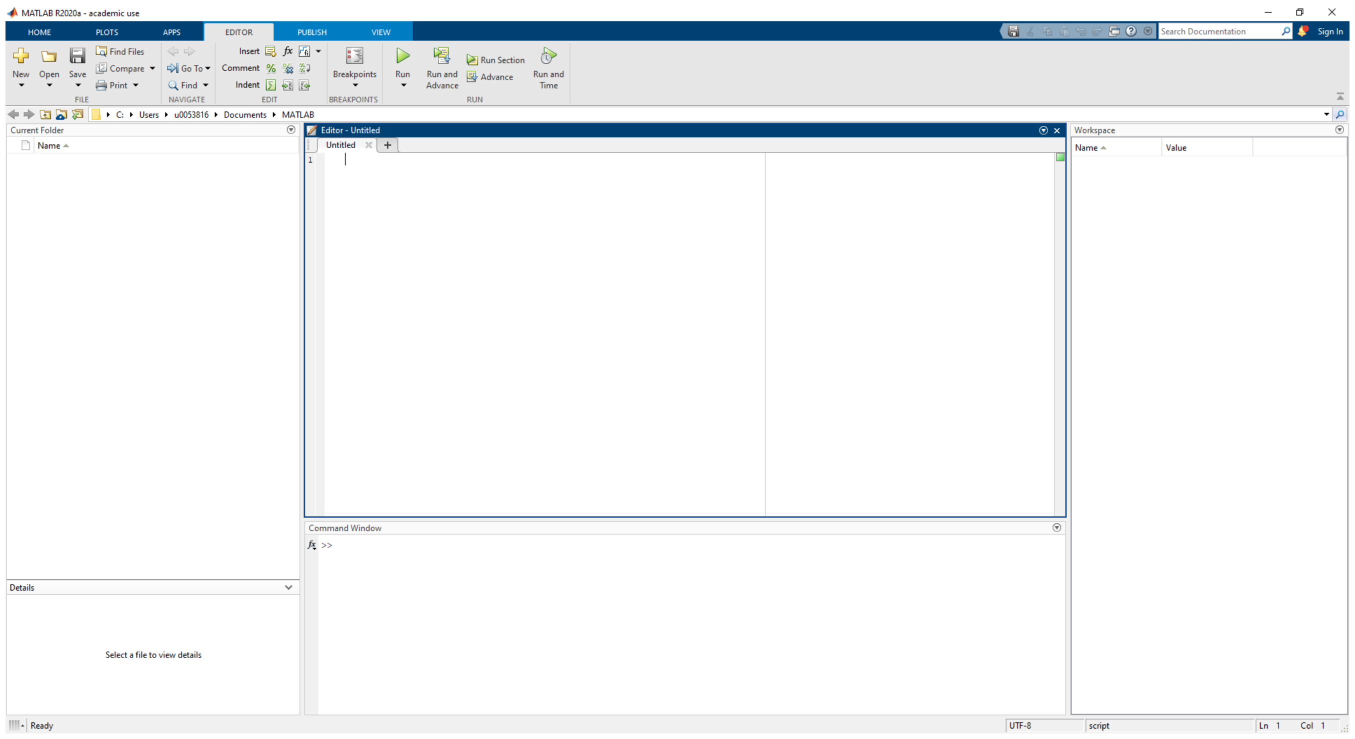
```
data = 7×4
     3.0000    0.5300    4.0753       NaN
    18.0000    1.7800    6.6678    2.1328
    19.0000    0.8600    1.5177    3.6852
    20.0000    1.6000    3.6375    8.5389
    21.0000    3.0000    4.7243   10.1570
    23.0000    6.1100    9.0698    2.8739
    38.0000    2.5400    5.3002    4.4508
```

Name: Hayden Walpole   Date: Sept 26,2023

Part 2: Scripts (5 points)

As the name implies, a *script* is simply a generic text file with a bunch of instructions in it. Those instructions are then executed line-by-line as if you were typing directly into MATLAB's command window. This way, you do not have to keep re-typing your code every time you wish to run it again or make small changes. In principle, you can write a script file with any text-editing program you like, but MATLAB provides its own built-in editor for you. The only requirement is that the file must end with a ".m" extension in order for MATLAB to recognize it.

To begin this next lab section, open up a new MATLAB window on your computer and select *New Script* from the HOME tab. The editor should immediately pop open with a new, untitled script file like in the image below:

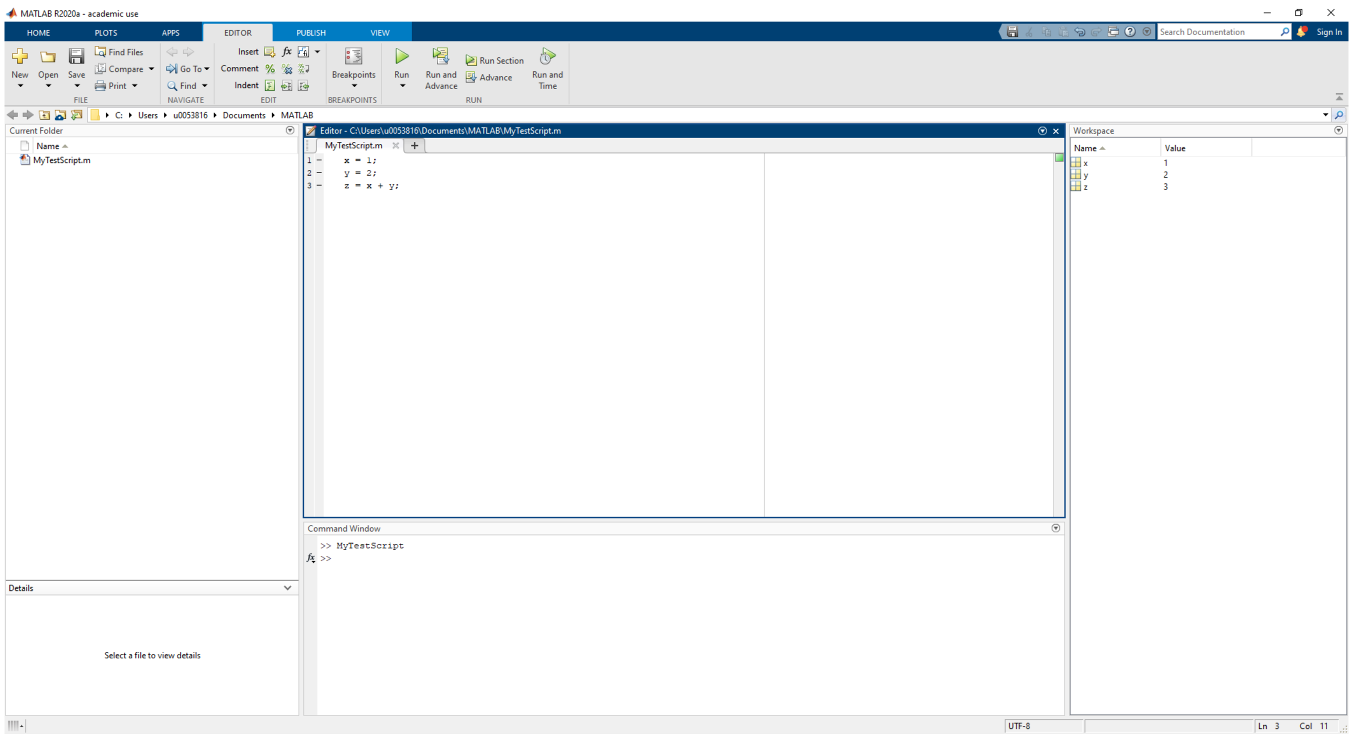


Name: Hayden Walpole Date: Sept 26,2023

Next, type the following commands into the editor:

```
x = 1;
y = 2;
z = x + y;
```

When finished, click the “Run” icon under the EDITOR tab. You will be asked to give your script file a name, so pick something simple like *MyTestScript.m*.



MyTestScript.m

Editor - C:\Users\u0053816\Documents\MATLAB\MyTestScript.m

```
x = 1;
y = 2;
z = x + y;
```

```
>> MyTestScript
>>
```

Name	Value
x	1
y	2
z	3

Notice how your script file appears in the “Current Folder” window. This is important, because script files cannot just randomly hide under obscure directories on your computer. MATLAB needs to be told where the file is before it can run it.

If you look closely at the command window, you will see that your script file has been typed into the command line. This is technically how MATLAB is commanded to run scripts, provided that it knows where to find them. Try modifying your script and then typing its name into the command line yourself. Did MATLAB respond to the new code as you expected?

Name: Hayden Walpole Date: Sept 26,2023

Part 3: Code Documentation (5 points)

Before we continue on with the main part of this lab, it is important to instill some basic documentation habits. Starting with the file name itself, create a new script file and call it *ResistorCalculator.m*. This will be the main script file used for the rest of the lab.

One of the first things to notice is that the file name is descriptive of its function—namely, it will do calculations on resistors. In contrast, a common mistake is to give our files obscure names like *Rcalc.m*, or perhaps *xyz.m*. This is generally considered bad practice, as it needless hides the purpose of the file from other human beings.

Notice also the use of capital letters mixed with lower case. Since spaces are not allowed in file names, we need a way to separate words. A common convention for this is known as *Camel Case*, whereby the first letter of each word is capitalized. This makes the file name much easier for a human to read and understand. Another common convention is to use underscores as a delimiter between words, which might look something like *resistor_calculator.m*. One could also combine the two conventions together to read *Resistor_Calculator.m*.

Next, we need to document our script file with human-readable text. Programming languages typically facilitate this practice in the form of *comments*, which are simply lines of code that get ignored during execution. MATLAB, for example, will ignore all text on a single line that comes after a percent sign (%). This allows the programmer to insert little bits of ordinary English sentences throughout the code so that other human programmers can make sense out of what is happening. Often times, that “other programmer” is actually yourself, but months after the fact. Code that once seemed intuitive to you in the moment can easily turn into gobbledygook when a year has passed by. So do everyone (and yourself) a favor by always documenting your programs. You will thank yourself for it later.

Although there are many schools of thought on the subject, a good rule of thumb is that well-written computer code should consist more of human documentation than actual code. That is to say, it is generally better to err on the side of *over*-documentation rather than *under*-documentation. As you grow proficient in your coding skills over time, you are likely to develop your own personal style of documentation. In this class, however, we shall be requiring a few basic features.

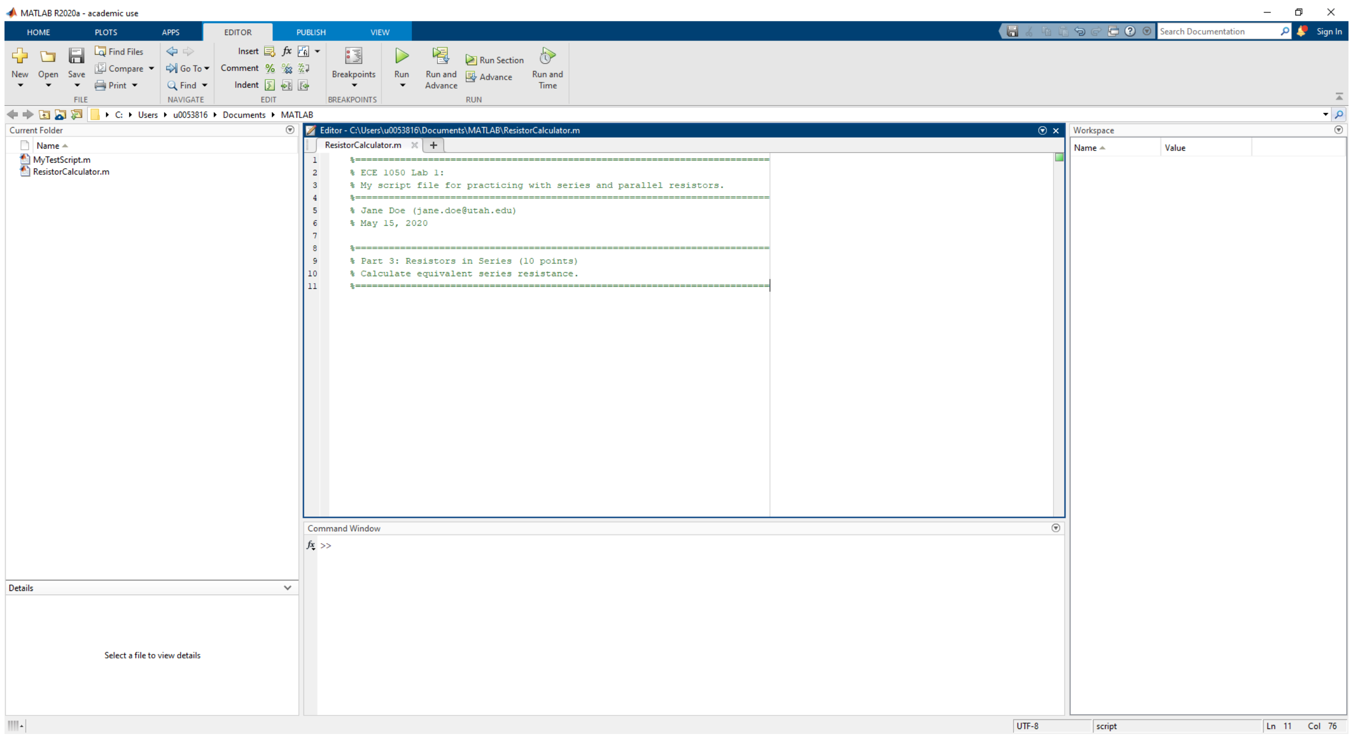
The first piece of documentation is the header, which is a lump of commentary that goes at the very top of your script file. The purpose of the header is to provide a brief summary of what your code does while also providing basic identifying information (e.g., name, date, course, etc.). Although there is technically no “correct” way to do this, a good example might look like the following:

```
%=========================================================================
% ECE 1050 Lab M1:
% My script file for practicing with series and parallel resistor circuits.
%=========================================================================
% Jane Doe (jane.doe@utah.edu)
% May 11, 2020
```

Notice the long bars of equal signs to delineate the header from other sections of code. While not essential, it can often be a nice way to visually parse your code into distinct blocks. For example, each lab session is going to be divided up into different parts as well, so it often helps to imagine large chunks of

Name: Hayden Walpole Date: Sept 26,2023

code placed between the comment blocks. Again, there is not necessarily a “right” or “wrong” way to do this, but a good example might look like this:



```
%==========================================================================
% ECE 1050 Lab 1:
% My script file for practicing with series and parallel resistors.
%==========================================================================
% Jane Doe (jane.doe@utah.edu)
% May 15, 2020

%==========================================================================
% Part 3: Resistors in Series (10 points)
% Calculate equivalent series resistance.
%==========================================================================
```

```
>>
```

As you gain confidence with your coding skills, feel free to experiment with various documentation styles. Just remember that the most important thing is for your code to be neat and legible. There is nothing more tedious than trying to make sense out of poorly written code!

Name: Hayden Walpole Date: Sept 26,2023

Part 3: Resistors in Series (10 points)

One of the most important skills in circuit theory is the ability to imagine complex arrangements of components in terms of simplified equivalent values. A good example of this is the array of resistors depicted in Fig. 1. Resistors arranged in this way are said to be in *series* with each other, and we can treat the array as if it were all squished into a single resistor. According to circuit theory, the equivalent series resistance R_s is just the sum of the individual resistor values. In other words,

$$R_s = R_1 + R_2 + R_3 \ .$$

Let us now use our *ResistorCalculator* script to calculate R_s. Underneath the header for Part 3, create three variables called "R1," "R2," and "R3" by assigning them their respective values. When finished, your code should look something like this:

```
% Define resistor values (ohm).
R1 = 1e3;
R2 = 3.3e3;
R3 = 1.5e3;
```

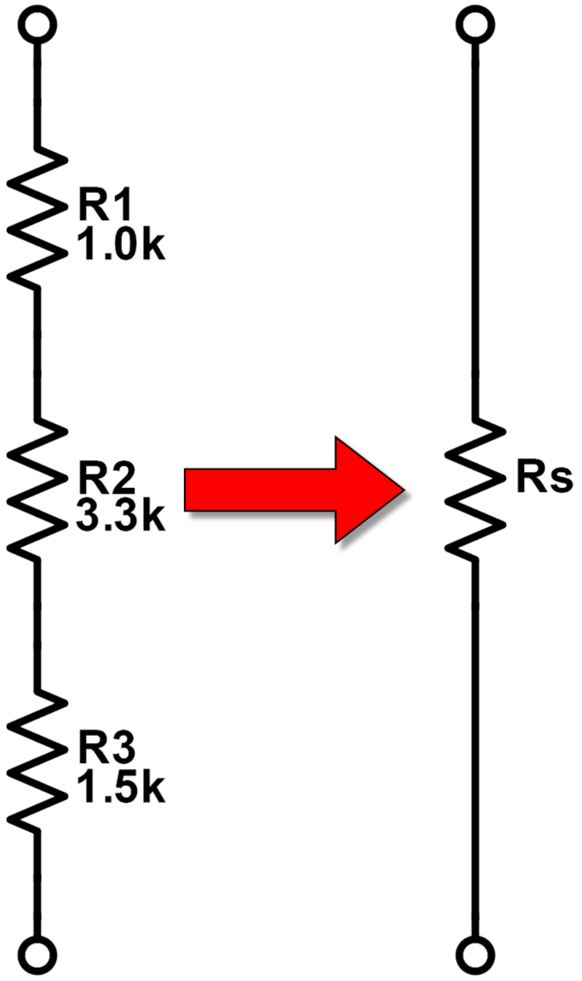


Figure 1: *Three series resistors imagined as a single equivalent resistance.*

Notice how variables in MATLAB are technically just raw numbers and thus have no implicit units. It is therefore important for you, the coder, to keep in mind what units accompany your variables. In this case, all the resistors have units of ohms (Ω), and so it has been indicated by the comment above. Another useful feature is the "e" operator, which is simply a shorthand for "$\times 10^x$". Thus, a value of "1e3" is interpreted as "1×10^3."

Finally, notice the generous use of spacing and alignment. Many beginners often make the mistake of cramming their code together with something like the following:

```
%Define resistor values (ohm).
R1=1e3;
R2=3.3e3;
R3=1.5e3;
```

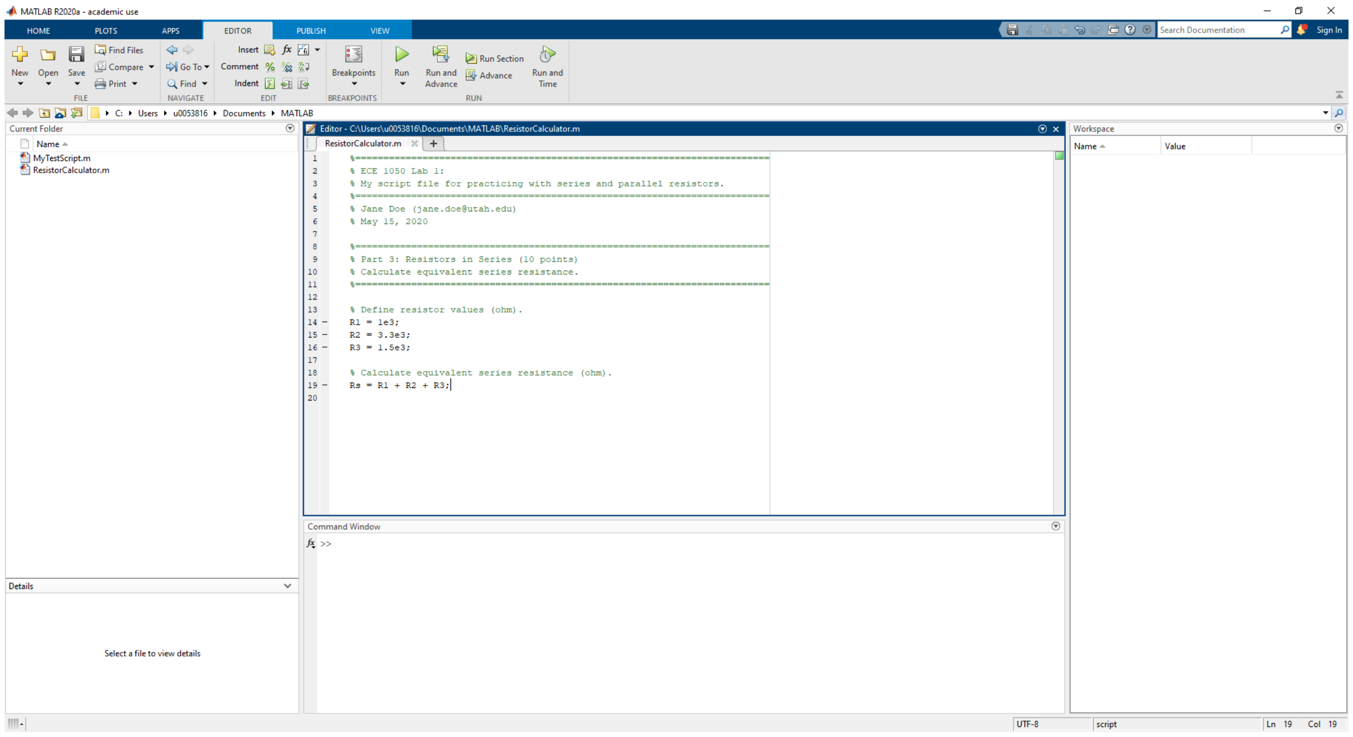
While technically valid code, it is also significantly more difficult for human eyes to read. For simple bits of code like above, this may not be a problem. However, it is only a matter of time before you start writing down long, complex instructions, and it is almost guaranteed that some of those instructions will have bugs in them. While it may be tempting to cut corners with the initial code, you also risk spending 30 minutes after the fact to track down some subtle typo. So always take the time to make your code clear and legible, lest you end up paying for it later during the debugging process.

Name: Hayden Walpole Date: Sept 26,2023

Let us now create a variable called "Rs" to indicate the total series resistance. Following the series resistance formula above, the calculation should look like the following:

```
% Calculate total series resistance (ohm).
Rs = R1 + R2 + R3;
```

For comparison, your completed script file should resemble the image below:



MATLAB R2020a - academic use

Editor - C:\Users\u0053816\Documents\MATLAB\ResistorCalculator.m

Current Folder: MyTestScript.m, ResistorCalculator.m

```
%==========================================================================
% ECE 1050 Lab 1:
% My script file for practicing with series and parallel resistors.
%==========================================================================
% Jane Doe (jane.doe@utah.edu)
% May 15, 2020

%==========================================================================
% Part 3: Resistors in Series (10 points)
% Calculate equivalent series resistance.
%==========================================================================

% Define resistor values (ohm).
R1 = 1e3;
R2 = 3.3e3;
R3 = 1.5e3;

% Calculate equivalent series resistance (ohm).
Rs = R1 + R2 + R3;
```

Run your script now and fill in the blank with the calculated value for R_s:

Rs = 5800 Ω .

Name: Hayden Walpole Date: Sept 26,2023

Part 4: Resistors in Parallel (10 points)

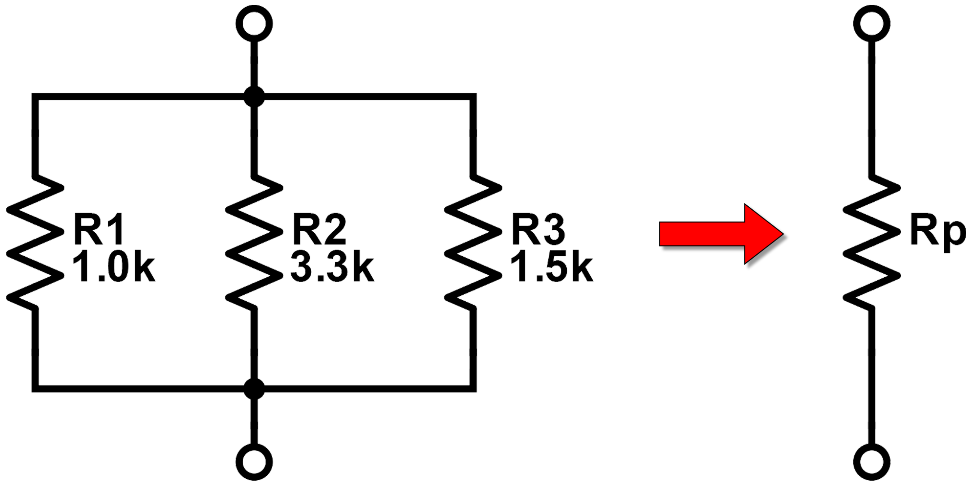


Figure 2: *Three parallel resistors imagined as a single equivalent resistance.*

Let us now consider the resistors depicted in Fig. 2 above. Resistors like this are said to be connected in *parallel*, and we can again lump them together into a single, equivalent value. However, instead of adding together linearly, the equivalent parallel resistance R_p satisfies

$$R_p = \left(\frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_3}\right)^{-1} .$$

Obviously, this is a little more complicated than the series case. Thus, to manage the new computation, we shall break it up into multiple little sub-tasks.

Begin by calculating the *conductance* G of each resistor, where conductance is defined as $G = 1/R$. Use your script to create three new variables called G1, G2, and G3 to correspond to each resistor. For example, your code may look something like this:

```
% Calculate conductances (S).
G1 = 1/R1;
G2 = 1/R2;
G3 = 1/R3;
```

Next, create a new variable called "Gp" that represents the total parallel conductance G_p of the circuit. Calculate G_p using the formula

$$G_p = G_1 + G_2 + G_3 .$$

Finally, create another variable called "Rp" to represent the equivalent parallel resistance. Calculate R_p using

$$R_p = \frac{1}{G_p} .$$

Notice the juxtaposition between series and parallel circuits. When resistors are arranged in series, the equivalent resistance is just the sum of the individual values. When resistors are arranged in parallel, the equivalent conductance is now the sum of the individual conductances.

Name: Hayden Walpole Date: Sept 26,2023

Fill in the blanks below with the calculations of your script.

$G_1 =$ 1 S .

$G_2 =$ 3.0303 S .

$G_3 =$ 6.66666666666666 S .

$G_p =$ 0.002 S .

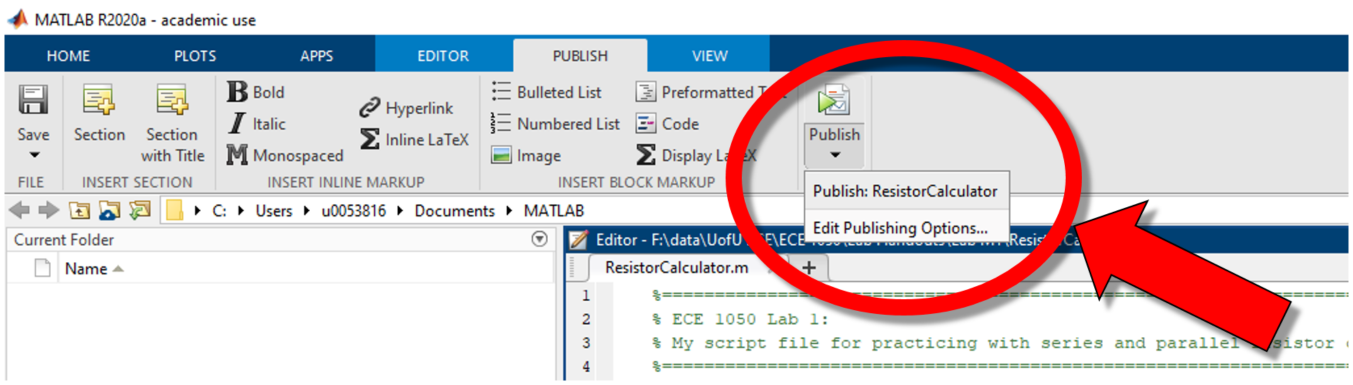
$R_p =$ 507.69 Ω .

Name: Hayden Walpole Date: Sept 26,2023

Appendix: Exporting MATLAB Code

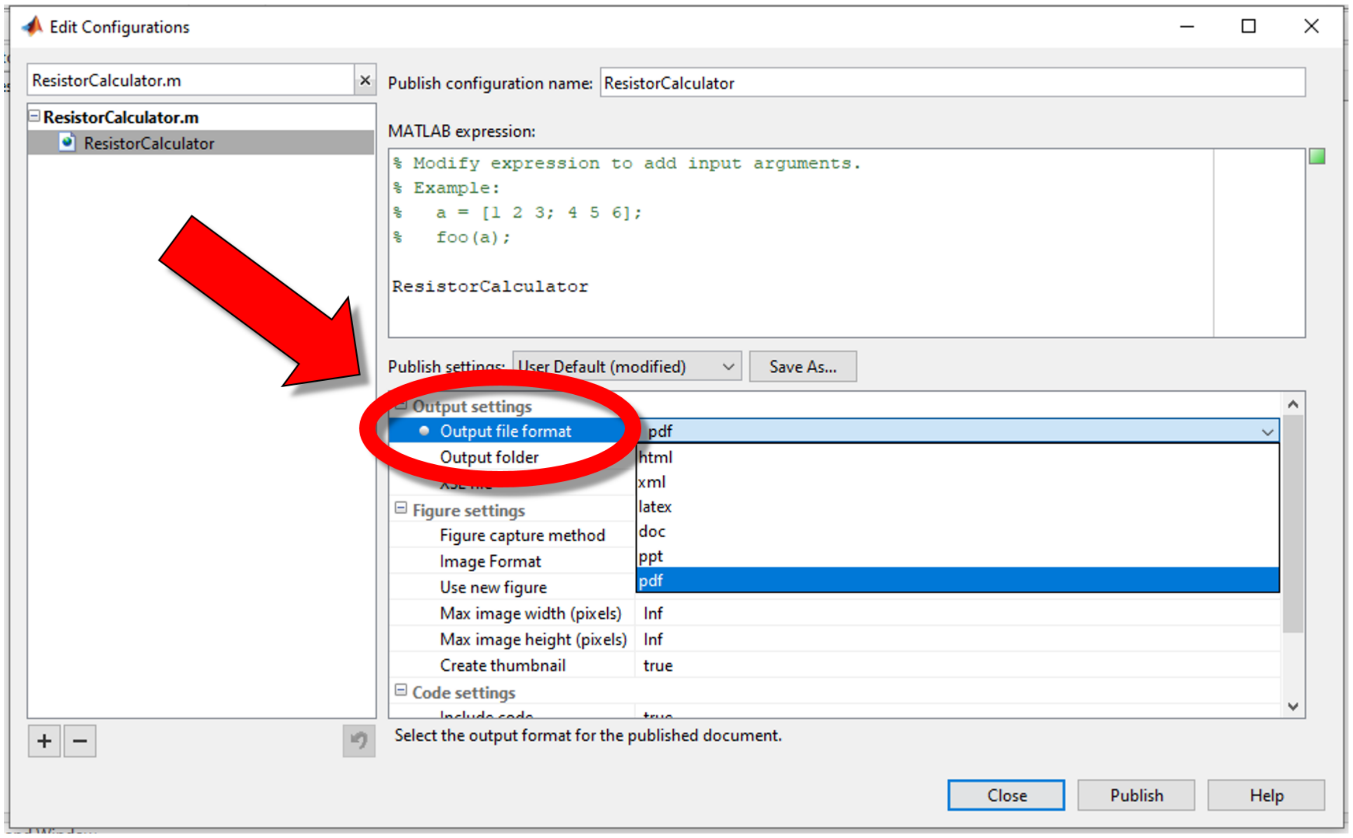
The following is a tutorial on how to convert MATLAB code into a PDF file and does not need to be signed or submitted.

Begin by selecting the PUBLISH tab from your MATLAB window and then clicking on the "Publish" dropdown menu. Select "Edit Publishing Options."



```
%==================================================================
% ECE 1050 Lab 1:
% My script file for practicing with series and parallel resistor
%==================================================================
```

A new window called "Edit Configurations" should not pop up. Continue by selecting the "Output file format" and choosing "pdf" from the drop-down list.



Publish configuration name: ResistorCalculator

MATLAB expression:

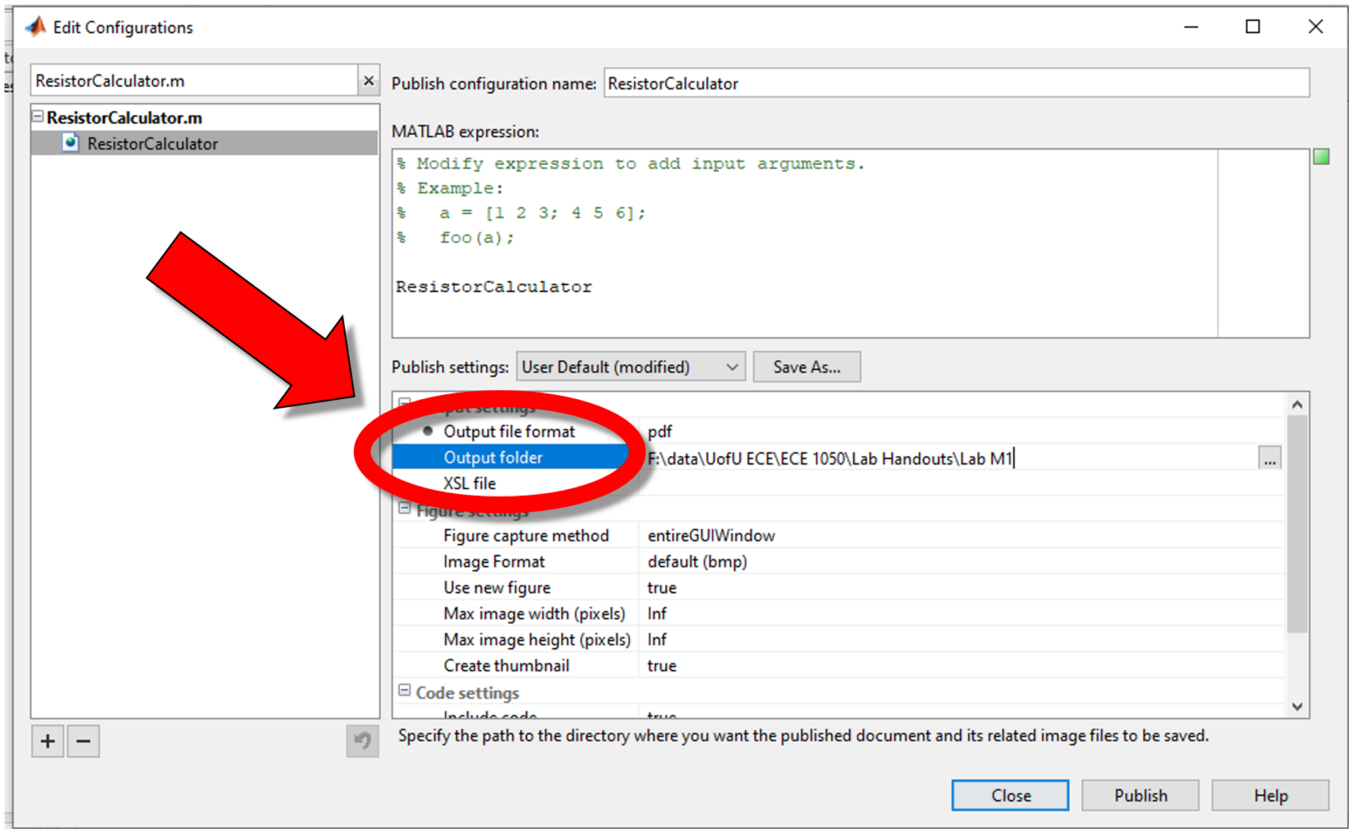
```
% Modify expression to add input arguments.
% Example:
%   a = [1 2 3; 4 5 6];
%   foo(a);

ResistorCalculator
```

Publish settings: User Default (modified)

Output settings	
Output file format	pdf
Output folder	html
XSL file	xml
	latex
Figure settings	doc
Figure capture method	ppt
Image Format	pdf
Use new figure	
Max image width (pixels)	Inf
Max image height (pixels)	Inf
Create thumbnail	true
Code settings	
Include code	true

Select the output format for the published document.

Finally, select "Output folder" and choose an appropriate destination for your PDF file.



Publish configuration name: ResistorCalculator

MATLAB expression:

```
% Modify expression to add input arguments.
% Example:
%   a = [1 2 3; 4 5 6];
%   foo(a);

ResistorCalculator
```

Publish settings: User Default (modified)

Output file format	pdf
Output folder	F:\data\UofU ECE\ECE 1050\Lab Handouts\Lab M1
XSL file	
Figure settings	
Figure capture method	entireGUIWindow
Image Format	default (bmp)
Use new figure	true
Max image width (pixels)	Inf
Max image height (pixels)	Inf
Create thumbnail	true
Code settings	
Include code	true

Specify the path to the directory where you want the published document and its related image files to be saved.