Lab: Custom Functions

Question 1 (15 minutes): Solar Panel Parameters

A solar panel consists of many photodiodes connected together to convert light into electrical power.

Task 1: Write a function that returns the output voltage and current of a solar panel given the number of rows and columns of photodiodes in the solar panel. The equation that gives the output voltage and current of a solar panel is below.

The function header: *[Vout Iout] = solar_panel(Nrow, Ncol)*

$$Output\ Voltage\ (Volts) = Number\ of\ rows * 0.6\ Volts$$

$$Output\ Current\ (mA) = Number\ of\ columns * 50\ mA$$

Task 2: The maximum output power of the solar panel is 90% of the product of the output voltage and output current.

Using the function you wrote, write a script that calls this function to calculate the maximum output power for square solar panels with sides ranging from 10 photodiodes to 100 photodiodes. In a square solar panel, the number of rows is equal to the number of columns.

Task 3: Do not modify the function 'solar_panel()', write code to plot the maximum output power for square solar panels versus the *total* number of photodiodes (for Nrow = Ncol = 10 to 100) in the solar panel, for . (x-axis is the number of photodiodes)

Lab: Custom Functions

Question 2 (20 minutes): Optimal Solar Panel Parameters

A solar panel consists of many photodiodes ("pd") connected together to convert light into electrical power.

Task 1: Write a function that calculates the *optimal* output voltage and current (optimal voltage and current gives *maximum* output power) of a photodiode given an illumination condition.

The function header:

[Voptimal Ioptimal] = pd_optimal(illum)

where $Voptimal = 1.5 * 0.026 * \left(\log\left(\frac{illum_{current}}{0.6*10^{-6}}\right) + 1\right) - 0.093$ *and*

$$Ioptimal = illum\_current - 0.6 * 10^{-6}\left(e^{\left(\frac{Voptimal}{1.5*0.026}\right)} - 1\right)$$

where *illum* relates to *illum_current* as follows: (the equations above use *illum_current*, NOT *illum*. You decide how to look up *illum_current* based on *illum*)

illum	*illum_current*
500	0.21
600	0.3
700	0.4
800	0.5
900	0.6
1000	0.6
1100	0.7
1200	0.8
1300	0.9
1400	1.1

Task 2: The maximum output power of the solar panel is simply the product of the optimal output voltage and output current.

Using the function you wrote, write a script that calls this function to calculate the maximum output power for square solar panels with sides ranging from 10 photodiodes to 100 photodiodes. In a square solar panel, the number of rows is equal to the number of columns. Fix 'illum' to 500. See the formulas below on how to calculate the output voltage and current of the solar panel, Vpanel and Ipanel.

Vpanel = Nrow * Voptimal;

Ipanel = Ncol * Ioptimal;

Lab: Custom Functions

Task 3: Do not modify the function 'pd_optimal()', write code to plot the maximum output power of the square solar panels versus the *total* number of photodiodes (for Nrow = Ncol = 10 to 100) in the solar panel, for . (x-axis is the number of photodiodes)

Lab: Custom Functions

Question 3 (15 minutes): Measuring MATLAB Performance

Task: We will explore the power of MATLAB by implementing our own '.*' operator using a simple for loop. Hopefully you will truly appreciate the built-in MATLAB operators/functions after this task.

Step 1: Create two 1000x1000 matrices, each containing random numbers by using

'x = rand(1000); y = rand(1000)'

Step 2: Perform an element-wise multiply of the two matrices, **x** and **y**. Store the result to a matrix **z**.

Step 3: We will now implement our own '.*' operator. Write a nested for-loop such that the outer loop iterates through each element in each row, and the inner loop iterate through each element in each column. Inside the inner loop, multiply the element in **x** with the element in **y**. Store the result in **z**, at the same location as **x** and **y**.

Step 4: We will now use MATLAB's built-in timer to time the two approaches, and compare them. Use the command 'tic' to start the timer. Use the command 'toc' to stop the timer. Place 'tic' just before the 'z = x.*y'' statement, and place 'toc' just after it. If you run the code now, you should see something like the following:

```
Elapsed time is 0.008232 seconds.
```

Step 5: Now place 'tic' just before the outer for-loop, and place 'toc' just after the outer for-loop terminates. Run the code. You should see something similar to the output in Step 4.

Step 6: Compare the times you obtained in Step 4 and Step 5. For this task, how much faster is the MATLAB's implementation of the element-wise operator than your nested for-loop implementation?

The lesson in this task is that you should always be careful when using loops to process large amounts of data. Taking the extra time to learn built-in MATLAB operators/commands can be very useful and saves you <u>hours</u> in simulation time.

Lab: Custom Functions

Question 4 (15 minutes): Delay in Using Functions

Task: We will explore the delay incurred from using a function, as opposed to keeping all code inside the same script.

Step 1: Write code that performs element-wise addition of two vector variables. You may assume the two variables will always be the same size. Use rand() to initialize the two vectors.

Step 2: Place the code from Step 1 inside a for-loop that runs for one million times.

Step 3: Create a new function called 'myadd'. Copy the code you wrote in Step 1 and place it inside this function. Note: this function should take two input arguments, and return one output argument. Save the function/file.

Step 4: We will now use MATLAB's built-in timer, tic and toc to compare the two approaches. Use the command 'tic' to start the timer. Use the command 'toc' to stop the timer.

In the same script as Step 2, place 'tic' just before the code segment, and place toc right after the code segment. If you run the code now, you should see something like the following:

```
Elapsed time is 0.008232 seconds.
```

Step 5: Now adding to the script in Step 4, write code to call the function 'myadd()' a million times. Place tic and toc before and after the function call, respectively. Run the code. You should see something similar to the output below, where the first line is the result for performing element-wise addition by calling 'myadd()', and the second line is the result without using a function. We see that using a function incurs a large delay in code execution time.

```
Elapsed time is 4.128054 seconds.

Elapsed time is 0.320287 seconds.

>>
```

The lesson in this task is that you should always be careful when calling a function many times. Taking the extra time to think about whether the contents of a function should really be simply placed inside the main script can potentially save hours of simulation time.

Lab: Custom Functions

Question 5 (20 minutes): Comparing Fixed and Tracking Solar Panel Configurations

Fixed-tilt solar panel is a type of solar panel configuration where the panels do not track the movement of the sun. The panel simply faces the South (if you are north of the equator), and is tilted at an angle equal to the panel's geographical latitude from the normal. This configuration is simple and is relatively inexpensive. Table 1 below contains the output voltages and currents of a fixed solar panel at different load conditions.

Single-axis tracking solar panel is a type of solar panel configuration where the solar panels track the movement of the sun as it moves across the sky. The solar panel's tilt angle is fixed, but it does not just face South, its orientation can rotate from east to west. As you can see, this configuration potentially yields a higher power output but comes at a cost of more complex electronics. Table 2 below contains output voltages and currents of a single-axis tracking solar panel

A third configuration, *double-axis tracking* tracks both the sun's movement throughout the day and its altitude as the season changes. We will not be discussing this configuration in this lab.

	Fixed-Tilt, no tracking	
Load Condition	Voltage	Current
1	0.424	0.0338
2	0.628	0.0338
3	1.282	0.0338
4	1.832	0.0336
5	2.808	0.0336
6	3.948	0.033
7	4.88	0.033
8	8.718	0.032
9	11.642	0.031
10	14.45	0.0286
11	16.644	0.021
12	17.18	0.017
13	17.632	0.012
14	17.928	0.008
15	18.086	0.0052
16	18.168	0.004
17	18.242	0.0028
18	18.27	0.002

Table 1. Output voltages and currents of a solar panel under various load conditions.

Lab: Custom Functions

	Single Axis Tracking	
Load Condition	Voltage	Current
1	1.844	0.1462
2	2.738	0.146
3	5.578	0.1454
4	7.932	0.1446
5	12.018	0.1426
6	16.218	0.1356
7	17.562	0.1192
8	18.922	0.0738
9	19.304	0.0514
10	19.488	0.0384
11	19.658	0.025
12	19.722	0.019
13	19.776	0.0132
14	19.816	0.0088
15	19.84	0.006
16	19.852	0.004
17	19.862	0.003
18	19.864	0.002

Table 2. Output voltages and currents of a single-axis tracking solar panel.

In the tasks below, we will use the technique of curve fitting to find an equation to model the data above.

Task 0: Type 'load q5.mat to load the input vectors, **fix** and **single**. **fix** and **single** both contains two columns; first column is the solar panel's output voltage and second column is the current. The content of **fix** is shown in Table 1 above. The content of **single** is shown in Table 2 above.

We will now use the 'polyfit()' function to curve fit the data in Table 1 above. The x-axis will be voltage and the y-axis will be current.

Task 1: Assign the first column of **fix** to a variable name 'x' and assign the second column to variable 'y'.

Task 2: Write code to plot 'x' v.s. 'y' on a new figure. This is the current-voltage relationship (IV curve) of a solar panel. We saw in the previous lab that this curve shifts up and down

depending on the illumination. Note: this is a nonlinear relationship. It has a logarithmic relationship (rotated 90 degrees clockwise).

Task 3: In order to fit this log relationship to a polynomial, we must first linearize it. To linearize, we will apply 'exp()' to the x-axis data. Assign the results of this operation to a variable call 'x_exp'. Write code to plot 'x_exp' v.s. 'y'. We see that this is almost a straight line.

Task 4: We will now fit this line to an equation of the form $y = mx + b$. Call 'polyfit()' using 'x_exp' and 'y'. Fit it to a first degree polynomial. Store the output of 'polyfit()' to a variable call 'p'. Consult the lecture notes and Matlab "help" on how to use 'polyfit()'.

The result of the above task is stored in 'p'. The coefficients *m* and *b* are in p(1) and p(2), respectively. Follow steps below to compute and plot the linearized fitted curve.

Since we linearized the x-axis data using the exponential function earlier in Task 3, we need to compute $y = mx + b$ using a logarithmically spaced x vector.

Task 5: Generate a logarithmically-spaced vector from 10^1 to $10^{8.6}$ using 'logspace()'. Store the vector to a variable call 'x_logspace'. Assign P(1) to variable *m*, and P(2) to variable *b*. Now use the equation below to compute 'yt'.

$$yt = m * x\_logspace + b$$

Task 6: Write code that plot 'x_logspace' v.s. 'yt' (in red) on the same figure as 'x_exp' v.s. 'y' (in blue). Note: the third argument of 'plot()' is the color/format string. Example, "plot(x,y,'r')" plots the line in red while "plot(x,y)" simply plots the line in blue by default.

We will now reverse the effects of linearization.

Task 7: Obtain a linearly spaced vector from 'x_logspace' by using the 'log()': 'log(x_logspace)'. Assign the result to a variable called 'x_linspace'.

Task 8: Write code to plot 'x_linspace' v.s. 'yt' (in red) on the same figure as 'x' v.s. 'y' (in blue). This completes the curve fitting process.

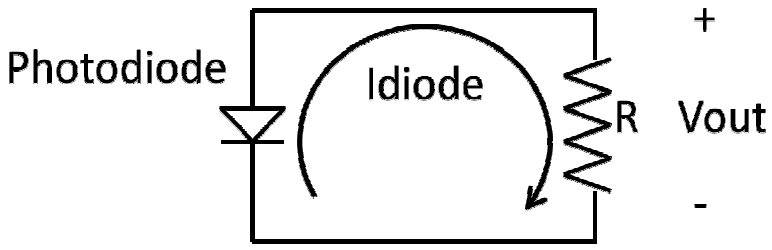
We will now compare the current generated by a fixed configuration vs. a tracking configuration.

Task 9: The variable *b* is the current generated by the solar panel. What is the value of *b* for the fixed-tilt solar panel configuration? What is the value of *b* for the single-axis tracking solar panel configuration?

Lab: Custom Functions

Question 6 (30 minutes): Solving Nonlinear Equations With Newton's Method

The figure below shows the schematic (a circuit diagram) of a photodiode connected to a resistive load, R. The equation that describes the circuits is directly beneath the schematic. The equations relates



$$Vout = Idiode * R$$
$$Idiode = Ilight - Idark * \left(e^{\left(\frac{Vout}{Vt}\right)} - 1\right)$$

$WHERE \quad Ilight = 500 * 10^{-3} Amps; \quad Idark = 0.6 * 10^{-6} Amps$

$$Vt = 26 * 10^{-3} Volts; \qquad R = 10$$

We see that Vout is a function of Idiode, but Idiode is also a function of Vout. We have two equations and two unknowns. So we should be able to solve it, right? But no, that is not the case here due to the exponential. Equations in this form ($x = e^{-x}$) are called transcendental equations. They must be solved either graphically or numerically. In this lab, we will solve this problem using three different methods.

Method 1: Finding the solution graphically

Task 1: Input equations (1) and (2) below into MATLAB as two anonymous functions. Write code that plots the two functions on the same figure. By looking at the graph on the screen, zoom-in and estimate the point of intersection graphically. Write down the current and voltage that corresponds to this point as MATLAB comments. Let x-axis be voltage and y-axis b current.

Let vout range from 0 to 0.4 at steps of 0.01.

$$Idiode(Vout) = \frac{Vout}{R} \qquad \text{equation (1)}$$

$$Idiode(Vout) = Ilight - Idark * \left(e^{\left(\frac{Vout}{Vt}\right)} - 1\right) \qquad \text{equation (2)}$$

where R, Ilight, Idark, and Vt are as before.

Lab: Custom Functions

Task 2: Let's now use MATLAB's function 'fzero()' to check our answer for Task 10. You will use the equation below (equation (3)) as the input argument to 'fzero()'. Equation (3) was obtained by first solving for Idiode in equation (1) then equating equations (1) and (2).

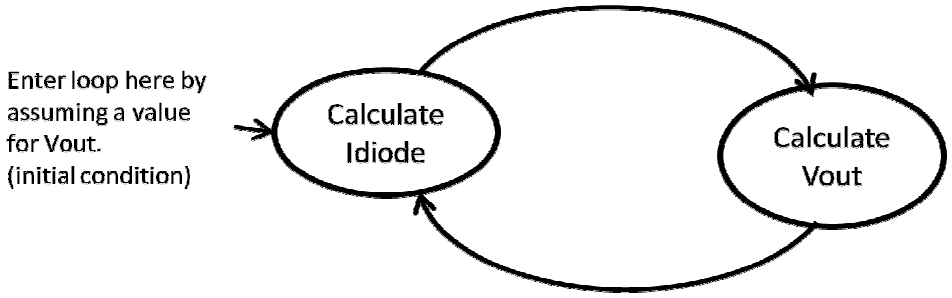
Type 'help fzero'on how to use 'fzero()'. Assign the value returned by 'fzero()' to a variable named 'actual_root'.

$$0 = Ilight - Idark * \left(e^{\left(\frac{Vout}{Vt}\right)} - 1\right) - \frac{Vout}{R} \qquad \text{equation (3)}$$

Method 2: Approximation through iteration.

Task 3: In this method, we will successively approximate 'Vout' by calculating 'Idiode' and 'Vout' in a loop. The more times the loop runs, the more accurate 'Vout' becomes. You will first make an initial guess for the value of 'Vout'. Using this guess for 'Vout', the loop first calculates Idiode, and then recalculates 'Vout'. The loop now moves onto the next iteration and the whole process repeats. (Similar to Newton's process of estimating the square root of a number in the homework problem)

A diagram depicting this process is below.



First input equations 4 and 5 below as two anonymous functions.

Now, write a for-loop that iterates through equations (4) and (5) below 10 times.
Use Vout=0.35 volts as an initial guess.

$$Idiode = \frac{Vout}{R} \qquad \text{equation (4)}$$

$$Vout = Vt * \log\left(\frac{Ilight - Idiode}{Idark} + 1\right) \qquad \text{equation (5)}$$

Compare this value to results obtained in Tasks 1 and 2. They should be the same.

Lab: Custom Functions

Method 3: Newton's Method.

We will now use Newton's method to solve for 'Vout'. In particular, we will find the roots of equation (6) below. In this course, we will simply use the results from Newton's method to help us solve numerical problems. The derivation of Newton's method and its application in optimization problems can be found in any numerical analysis and optimization textbooks.

Newton's method is simply stated as follows: Given f(x), the roots of f(x) can be approximated by iterating through the equation:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

where $x_n$is the current approximation of the root of f(x). The next approximation x_{n+1} is adjusted by subtracting the term $\frac{f(x_n)}{f'(x_n)}$ from the current approximation.

Task 4: Now write a function to calculate 'Vout' using Newton's method. Your function should take one input parameter and return one output parameter. The function header should look like the following: (see below for definitions for 'NumIteration' and 'myOutput')

myOutput = myNewton(NumIteration)

Start by defining the anonymous functions for equations (6) and (7) below.

- Equation (6) was obtained by solving for 'Idiode' in equation (1), then setting that expression equal to equation (2). This is the same as equation (3).
- Equation (7) is the derivative of equation (6)

$$f(vout) = Ilight - Idark\left(e^{\frac{Vout}{Vt}} - 1\right) - \frac{Vout}{R} \qquad \text{equation (6)}$$

$$f'(vout) = \frac{df(vout)}{dvout} = -\frac{1}{Vt} * Idark * e^{\frac{Vout}{Vt}} - \frac{1}{R} \qquad \text{equation (7)}$$

Now, use a for-loop to iterate through the following equation 10 times.

The parameter, 'NumIteration' is the number of times the for-loop runs, and 'myOutput' is the final value for 'Vout'. Initial Condition: Vout(1) = 0. 'i' in the equation below is the loop index.

$$Vout(i+1) = Vout(i) - \frac{f(Vout(i)}{f'(Vout(i))}$$

The final approximation for 'Vout' is in the eleventh entry, Vout(11). What is this value?

Lab: Custom Functions

Now iterate through it 50 times. What is the value of Vout? Note: The final approximation for 'Vout' is now at the 51st entry.

Now iterate through it 100 times. What is the value of Vout?

Now iterate through it 1000 times. What is the value of Vout?

Does it match the results obtained in Task 3?