OENG1206 - Digital Fundamentals

Practical Exercise 2

Programming in MATLAB

Practical Exercise 2: Programming in MATLAB

Learning Outcomes

Upon successful completion of this practical exercise, you'll be able to:

- Use MATLAB programming structures to write a program that decodes the colour codes on 4- or 5- band resistors.
- Apply programming structures including loops and conditional statements to a MATLAB program.
- Become familiar with writing and implementing user-defined functions in MATLAB.

Introduction to the practical

Resistors are one of the fundamental building blocks of electric circuits and can be found in just about every electrical/electronic circuit in existence. Their purpose is to impede the current flow in an electrical circuit. The relationship between voltage, current and resistance in a circuit is defined by **'Ohm's Law:'**

$$V = IR$$

where V is the voltage (in Volts), I is the current (in Amperes) and R is resistance (in Ohms).

For *through-hole* type resistors, where the legs of a resistor are inserted through a printed circuit board (PCB) and soldered onto the under-side of the board, a **colour-code** is utilised to allow people to determine the value and tolerances of a resistor.

There are two main types of resistor colour codes; **4-band** codes and **5-band** codes. For the four band codes the colours show the following:

- The first and second colours refer to the first two numbers of the resistor value.
- The third band is the multiplier and
- The last band is the tolerance.

In the 5-band case the colours are similar:

- The first, second and third colour bands now represent the first three numbers in the resistor value.
- The fourth band is now the multiplier and
- The fifth band is the tolerance.

The chart in Figure 1 shows how resistor values are determined by using the colour codes.

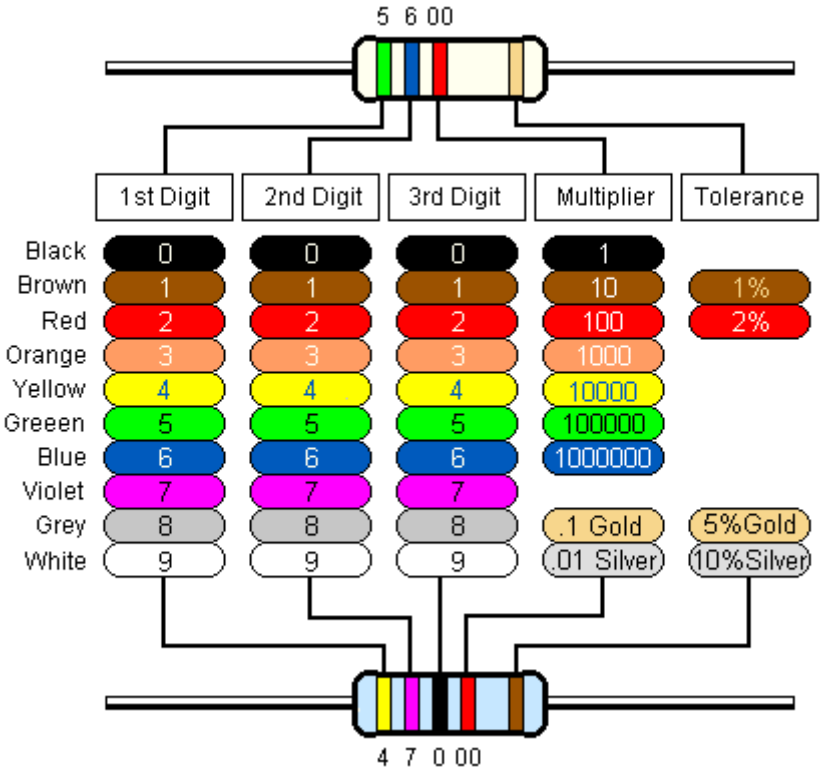

Figure 1: Resistor colour codes [1]

Therefore if a 4-band resistor has the colour code: green, blue, red and gold (as shown in the top diagram) it would have the value: 56 × 100 = 5600Ω (or 5.6 kΩ) with a tolerance of ±5%. In the 5-band example the colour code yellow, violet, black, red and brown would give a resistance of 470 × 100 = 47,000Ω (or 47 kΩ) with a tolerance of ±1%.

Part 1 (weeks 5 & 6): Resistor Colour Code Decoder

A company wants you to write a MATLAB program that:

- **Calculates and displays a resistor value** along with its **tolerance** for a colour code entered by a user.
- Consists of an initial **menu system** displayed to the user with **instructions** on how to enter their colour bands.
- Allows a user to enter resistor band colours for **either 4-band or 5-band resistor types**.
- Has **error checking/input validation** including notifying and re-prompting the user for input if they enter an invalid menu selection or colour.
- Displays the decoded resistor value and tolerance to the user in a **user-friendly** way on the **Command Window**.

This program must work for both 4- and 5- band resistor types.
Your program must also be a text-based user interface (**not** a Graphical User Interface (GUI)).

Refer to the sample program we tested in class during the week 1 lectorials for suggestions on how your program could work.

Task 1 – State the Problem

Exercise:
Start by writing down the problem statement so the requirements of this task are clear to you, check with your tutor if you are unsure of what you need to be doing.

Points to be addressed:

- The problem statement should be **clear and concise**, double-check with your tutor if you are not sure of anything written in the problem statement.

Task 2 – Specify your input and output data

Exercise:

Using the information given on the previous page determine the input you have/need to solve the problem and what you need to output to accomplish the objective.

Points to be addressed:

- What input **data types** do you need/have (numeric, words, images, graphical, files?). Describe these in detail including format of any files needing to be imported.
- Also think about **what you'll need to output** to fulfil this task, (numbers, words, images, graphical output?). Describe these in detail.
- What **assumptions** (if any) are you making to help you solve/simplify the problem

Task 3 – Algorithm Design

Exercise:

Using the information given on the previous pages, determine:

- What would be the **best way to obtain colour selections** from your user? What would reduce the possibility of invalid inputs?
- How could these selections be used to **mathematically calculate** the resistor's value and tolerance? Clearly show any mathematics your report.
- The **order** your program will be required to perform operations in (use a **flow-chart** to show this).

Points to be addressed:

- Make sure you have clearly shown the flow of your program as a **flow-chart** giving clear indications of the order you need to perform operations in.
- Show evidence your selected equations/formulae will work by **testing your algorithm using a small set of data/resistor colours**.
- Make sure all this section is neatly typed (not handwritten) in your write-up.

Task 4 – Use MATLAB to realise your algorithm

Exercise:

In a new MATLAB script file implement your algorithm.

You'll need to use the `disp()` and `input()` functions to display a menu and get your user's input.

You'll also need to use loops and conditional statements to check for the validity of your user's input and make a decision on whether to continue or re-prompt your user for input.

Points to be addressed:

- Make sure to test your program and perform a reality check by comparing your MATLAB output to either your handworked answer from task 3 and/or the sample resistor values from Lectorial 1. The output should be the same as these. If not, troubleshoot your program to find where the mistake is.
- Make sure to use appropriate names for all your variables and have commented your code.
- Make sure all Command Window output is easy to understand and follow. **Your program must be user-friendly**.

Part 2 (week 7): User-Defined Functions

For this part of the practical, you're required to enhance your program from part 1 by including a **user-defined function** that handles the mathematics required to decode the resistor colour codes.
Your program must be structured as follows:

- Your **main MATLAB script** must handle the **user-interface elements** of your program only. I.e. it will contain the **menu system**, **user input** and the code that **outputs the final answer** to the Command Window.
- Your **main MATLAB script** must also **call your user-defined function** from within its code.
- Your **user-defined function** must only handle the mathematical elements of the colour code decoding. The resistor's value and tolerance will then be **returned** from your function back to your main script.
- **Error checking/input validation** may be included in both your **main script** and **user-defined function** where relevant.
- Please ensure you've watched the lecture videos on **user-defined functions** (week 6 lecture) and ask teaching staff questions about this if you're stuck.
- Please also refer to the list of useful functions at the end of this document (Table I) for some of the functions available in MATLAB that will help with writing user-defined functions.

This program must still work for both 4- and 5- band resistor types.

Your program must still be a text-based user interface (**not** a Graphical User Interface (GUI)).

Refer to the sample program we tested in class during the week 1 lectorials for suggestions on how your program could work.

Task 5 – State the Problem and Determine the Input/Output

Exercise:

Using the information given on the previous page state the extended problem concisely and determine the input you have/need to solve the problem and what you need to output to answer the question.

Points to be addressed:

- The problem statement should again be **clear and concise**, double-check with your tutor if you are not sure of anything written in the problem statement (if you're unsure of what a user-defined function is, please make sure to check this before continuing the task).
- What input/output **data types** do you need (numeric, words, images, files, graphical output?). Again, be detailed in your explanation here.
- What **assumptions** are you making to help you solve/simplify the problem.

Task 6 – Algorithm Design

Exercise:

Again, use the information given on the previous pages to design your extended program:

- Use **flow-charts** to show the order of operations for your program, including the user interface and your user-defined function.

Points to be addressed:

- Ensure this section is neatly typed (not handwritten) in your write-up.

Task 7 – Use MATLAB to Write and Test Program

Exercise:

Now extend your existing resistor colour-code decoder program to include the MATLAB user-defined function that decodes the resistor colours.

Your solution MUST contain a user-defined function to ensure you've fulfilled the requirements of this practical task. Make sure you've done this before submitting your work to Canvas!

Points to be addressed:

- Make sure to again test your program works by comparing your MATLAB output to your handworked answers and/or the sample resistor values from Lectorial 1.
- Make sure to present thorough evidence of **testing your program with a wide range of inputs (resistor colour-codes)** in your report.
- Also, make sure to clearly **present your program's output with a discussion** on how well it fulfils the requirements of the task in your report (strengths and weaknesses).
- Lastly, check the **report guidelines on Canvas** (under the Week 1 Module -> Assessment Task Instructions (IMPORTANT) -> Practical Instructions) to make sure you've covered all aspects of the problem-solving methodology adequately.

Table I: Some useful MATLAB functions

Function	Description
Functions for Functions	
`nargin`	Returns the number of arguments that have been input into a function. Used in functions to make sure the correct number of arguments has been passed to a function. Can be used with if-statements inside a function.
`nargout`	Returns the number of expected arguments being output from a function. Again this can be used to check an appropriate number of output arguments have been requested from the function. Again can be used with if-statements inside a function.
`varargin`	Allows for a variable number of optional arguments to be input into a function. **Syntax example:** `function x = MyFunc(a, b, varargin)`
`varargout`	Allows for a variable number of optional arguments to be output from a function. **Syntax example:** `function [out1, varargout] = MyFunc(x, y)`
Logical Operators	

`strcmpi(x, 'str')`	Case-insensitive string compare. Takes a string variable (`x`) and compares it to the string `'str'`.
	`strcmpi()` will return true (1) if the two strings match and false (0) if they are different.
	Syntax example:
	`if strcmp(x, 'hello') %does x contain the string 'hello?'`
	`disp('You said hello!')`
	`else`
	`disp('You didn't say hello.')`
	`end`

==	Equivalence. Often used in conditional statements to evaluate whether two variables are equal to each other. It returns true (1) if the two values are the same, false (0) if they are different.
	`x == 2 % is x equal to 2?`
	`y == x % is y equal to x?`
	This is not to be confused with the single = sign which means assignment.

~=	Not equal. Same as above but checks if two variables are not the same. Returns true (1) if values are not equal and false (0) if they are equal.
>	Greater than. Checks to see if one variable is greater than the other.
<	Less than. Checks to see if one variable is less than the other.
>=	Greater than or equal to. Checks to see if one variable is greater than or the same as the other.
<=	Less than or equal to. Checks to see if one variable is less than or the same as the other.

||	Logical OR. Often used in conditional statements to check if one or more conditions are true.
	Syntax example:
	`if (x==2) || (y==4)`
	`.....`
	`end`
	In this example if either `x` is equal to 2 or `y` is equal to 4 the if-statement will execute

&&	Logical AND. Checks if multiple conditions are true AT THE SAME TIME.
	Syntax example:
	`if (x==2) && (y==4)`
	`.....`
	`end`
	In this example if `x` is equal to 2 and `y` is equal to 4 the if statement will execute.

General useful functions	

`clear`	`clear` by itself will clear all variables from the workspace.
	If `clear` is followed by a variable's name(s), only that variable(s) will be cleared from the workspace.
	Syntax example:
	`clear % clears every variable`
	`clear var % clears var from workspace`
	`clear var1 var2 var3 % clears var1, var2 & var3 from workspace`

`clc`	Clears the command window

`disp(x)`	Displays `x` on the command window. `x` can either be a string or a variable.
	Syntax example:
	`disp('Hello World!') %Displays Hello World! on the command window`
	`disp(x) %Displays the contents of the variable x on the command window`

`X=input('prompt')`
`x=input('prompt','s')`
	Gets user input from the command window and stores that input in the variable x.
	Syntax example:
	`x=input('Enter a number: ') %Prompts user to enter a number onto the command line`
	`x=input('Enter a string: ','s') %Prompts user to enter a string onto the command line`

`length(x)`	Returns the number of elements in the array `x`.

[1] Stack Exchange. (2012), *Resistor Colour Codes* [online]. Available: http://electronics.stackexchange.com/tags/colour-coding/info