EECS LECTURE NOTES

Day 1:

- What is MATLAB?
  - Numerical computing environment that has its own programming language
    - Interactive : user enters commands, stuff happens
    - Visualization: plotting functionality
    - Programmable: user can create programs that can be run within the MATLAB environment
- Example 1: Plot the line y = 0.5x – 1 on the domain -1 <= x <= -5
  - >>x = -1:5 (the ":" in between the numbers means from the number on the left, to the number on the right)
  - >> y = 0.5*x-1 (have to use * to indicate multiplication)
  - Plot(x, y, 'o-') (the " 'o-'" means the plot will have that symbol in it)
- Example 2: Find the intersection between the previous equation and y = -1/3 x + 2
  - >> hold on (this retains plots in current axes so that new plot do not delete existing plot)
  - >> y = -1/3*x+2
  - >> plot (x, y, 'r*-')
  - This will show a graphical intersection, but to find actual numbers rewrite as matrix form:
    - >> A = [1/2 -1;

      1/3 1]

      >> b = [1;

      2]

      >> u = A \ b

      U=

      3.6

      0.8

- Real numbers
  - Most matlab applications deal with real numbers (opposed to integer numbers)
  - If you type a plain number into matlab, then matlab will interpret that number to be a real number of type **double**
    - Short for double precision floating – point
    - Computer memory is limited, cannot store numbers with infinite precision, at some point will have to cut off

- Idea is to compose a number of two main parts
    - Significand → contains the number's digits, negative significands represent negative numbers
    - Exponent → says where the decimal/ binary point is placed relative to the beginning of the significand, negative exponents represent numbers that are very small (close to zero)
    - This format satisfies all the requirements
        - Represent numbers at wildly different magnitudes (limited by length of exponent)
        - Provides same relative accuracy at all magnitudes (limited by length of significand)
        - Allows calculations across magnitudes: multiplying large and small number preserves accuracy of both in the result
- Decimal floating numbers usually take on form of scientific notation, with an explicit point between first and second digits
- Exponent is either written explicitly including the base, or an e is used to separate it from significand

Significand	**Exponent**	**Scientific Notation**	**Fixed-PointValue**
1.5	4	1.5 · 10^4	15000
-2.001	2	-2.001 · 10^2	-200.1

- Arithmetic operations
    - + , - , * , \ , ^
- MATLAB follows bedmas

Important commands:

- Plot(x,y) will plot the function x,y
- Hold on, will allow you to plot a second function on the same axes

Day 2: Basic Calculations and Variables

- Arithmetic examples
    - Compute circumference and area of a circle with radius 2.5
        - >> 2 * pi * 2.5
        - Ans = 15.7080
        - >> pi * 2.5 ^ 2
        - Ans = 19.6350
- Matlab computes values using 15-17 sig digs, but by default will display values between-1000 and 1000 using 4 digits after decimal place
- Can use "format" command to change this (format short and format longg)
- Functions
    - Matlab provides functions, this allows matlab programmer a way to perform well defined task using well defined interface
    - Interface
        - Function name
        - Inputs to the function
        - Outputs to the function
    - Function interface documented through built in help mechanism
        - "help function-name"
        - "doc function-name"
    - Elementary mathematical functions
        - Many elementary mathematical functions for trig, exponents and logs, and rounding
        - "help elfun" or "doc elfun"
- Variables
    - Will almost want to store the result of a computation
    - Variable = name given to a stored value; the statement z = 1 + 2 causes the following to occur :
        - Compute 1 + 2
        - Store result in variable named z
        - Note : 1 + 2 = z results in error
        - Matlab automatically creates z if it doesn't exist
    - "=" is the assignment operator
        - Z = 1 + 2
            - Evaluate the expression on the right hand side of =
            - Store the result in the variable on the left hand side of =
                - Store means store in computer memory
- A simple memory model
    - Memory model useful for understanding important programming concepts
    - Table with 3 columns

        - An address (starts at 1 and counts up by 1 for each row)
        - Variable name
        - Value stored
- Variable names
    - Must start with a letter
    - Rest of name can include letters, digits, or underscores
    - Names are case sensitive
    - Matlab has reserved words called keywords, cannot be used as variable names
        - Use command iskeyword to get full list
- Advice on choosing variable names
    - Use short, meaningful names
        - Useful for others who are trying to read your code
    - Use lowerCamelCase for most variable names
        - thetaRad instead of thetarad
    - avoid long names

Day 3: Representing numbers on a computer

- Matlab and computer memory
  - Whenever you type in a matlab statement into the command window, matlab immediately interprets and executes the statement
    - Most of the time, this causes a memory access to retrieve a value/a modification to memory to change a value
- Numbers and Computer Memory
  - Rule number 3 of memory model
    - Each value occupies a fixed, finite amount of computer memory
    - Assume every value occupies same amount of memory required to hold a double precision floating point value
- Significand + exponent representation
  - 4 (exponent) 1793(significand)
  - =1793 x 10^(4 – 7) = 1.793
  - 10 = base (a constant), and the 7 represents a bias (also a constant, it can be any number so long as it ends up being in scientific form)
- Double precision floating point
  - Most computer hardware uses base 2 (binary) instead of base 10
  - Most common representation for floating point numbers is defined by a standard
    - IEEE 754
      - 64 bits for double precision (64 boxes that can hold a 0 or a 1)
      - All numbers can be written in the form → $s \times 2^e$
      - Significand s and the exponent e are integer values
- Importance
  - Principal architect of IEEE 754 was William Morton kahan
    - Awarded Turing prize

Day 4: Basics of electric circuits

- Electricity
  - What a physicist would call “electric current”
  - i.e., flow of charge carriers
- Electric current
  - Flow of charge carriers is measured in (number of charge carriers / seconds)
  - Charge carriers in the electric circuits that we are studying are electrons
  - SI unit for number of charge carriers is called the coulomb ©
    - 1 C = 6.241 x 10^18 electrons
  - SI unit for electric current is called ampere (A)
    - 1 A = 1C / s
  - Single closed loop circuit vs open loop
    - Closed has a complete path for current to flow, open doesn’t → therefore it isn’t functional
- Measuring Current
  - Ammeter is used, but a multimeter can measure many things
  - Charge cannot be destroyed, current must be the same everywhere in a single loop circuit
- Power Source
  - For current to flow there must be a source of energy that can do work to move the charge carriers
- Voltage
  - Electric potential energy per unit of charge
    - Joules per coloum
  - Si unit = V, 1v = 1J / 1C
- Resistance
  - Electronic device that opposes the flow of current
    - Si symbol = Ω, called ohm
  - Measured by applying a potential difference across the resistor and measuring the current that flows through the resistor
  - Ohms law → V = IR
- Series resistance
  - Rtotal = R1 + 2
- Parallel Resistance
  - 1/Rtotal = 1/R1 + 1 /Rn

Day 5: Text / Formatted text

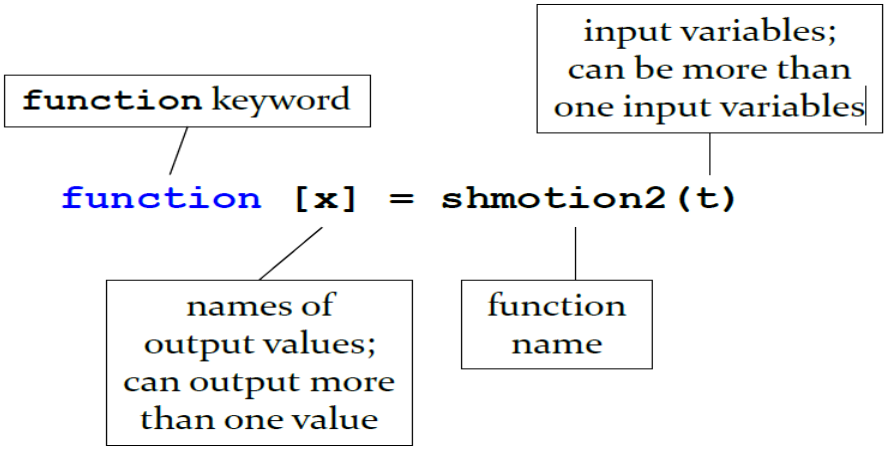
- Text output
  - A matlab string is a row vector of characters
  - Any text enclosed by single quotes is considered a matlab string
  - S = 'Any characters'
  - String is actually a vector that contains the numeric codes for the characters (codes 0 to 127 are ASCII)
  - Length of s = number of characters
  - Quotation within the string is indicated by 2 quotation marks
  - Command "disp" will display a matlab string to the screen without displaying the name of the variable
  - >> disp(s)
  - Any characters
- Formatted output
  - Often useful to generate strings programmatically
    - Display output more elaborate than just a variable value
    - Display table of values
    - Export data to a non-native matlab format
    - Generate string for matlab command
  - Function "sprint" used to generate formatted strings
    - Use doc sprint to get help for sprint
    - Syntax of sprint f is str = sprint(formatSpec, A1, …, An)
      - FormatSpec = formatting string with a large number of options
      - A1,…,An are variables or values containing the information to format
  - Formatting string describes how matlab should convert the information stored in the arrays into text
    - Can include :
      - Text
      - Escape characters
      - Zero or more percent signs each followed by optional operator characters and a conversion character
  - Usually will need to include a conversion character and an array containing data
    - Conversion character tells matlab what conversion it should use when converting the data in the variable to text
    - Most commonly used conversion are:
      - %d base 10 integer
      - %i base 10 integer
      - %f fixed point floating point
      - %e exponential notation

- %s string

- Examples
  - >> n=10;
  - >>sprintf('There are %d items', n)
  - There are 10 items
  -
  - Degc = 100;
  - Degf = (9/5) * degc +32;
  - Sprintf('%f deg C = %f deg F', degc, degf)
  - 10.000000 deg C = 50.000000 deg F
  - Sprintf('%e deg C = %e deg F', degc, degf)
  - 1.000000e+01 deg C = 5.000000e+01 deg F
- For floating point conversions, you can specify the number of digits after the decimal place
  - Called the precision
  - Default value is 6
  - Sprintf('%.2f deg C = %.2f deg F', degc, degf)
  - 10.00 deg C = 50.00 deg F
- For all conversions, you can specify the minimum numbers of character to put
  - N = 10;
  - Sprintf('There are %5d items',n)
  - There are 10 items

Day 6: Scripts and Functions

- MATLAB Scripts
  - Text file containing sequence of MATLAB commands
    - Each usually occurs on separate line of file
  - MATLAB can run commands in script by reading the file and interpreting the text as MATLAB commands
    - Run in order that they appear in script file
  - File name of script always has form → yourScriptName.m
  - yourScriptName must be valid MATLAB variable name
    - must begin with a letter and only contain letters and spaces and underscores
  - useful for keeping permanent record of a sequence of commands
  - typically you will want to output the result of the script
    - as a plot/message of some kind
  - also want to save results of computations that the script has performed
  - MATLAB will run script if you type the name of script in the command window
    - Must be saved in a folder that is on the current MATLAB path
      - Includes current working folder shown in the address bar
  - Useful to organize all scripts and functions in a common folder
    - See path command (and related functions)
  - Can create new variable or it can reuse existing variable in workspace
    - Can overwrite an existing variable in workspace
- User-defined functions
  - Scripts useful but
    - Variables in script appear in current workspace
    - Values of the variables used in script are fixed
  - In a UDF
    - Variables used in function do not appear in the current workspace
    - User of the function can specify the input values to the function
  - UDF is usually plaintext file having filename with form → yourFunctionName.m
    - yourFunctionName must be valid MATLAB name
  - first line of matlab function = function header
  -



- Block of comments immediately following the header is help documentation
  - Not required but very useful
  - Notice that in matlab functions, function name, names of output values, and names of input variables are written in uppercase in help
    - So that these names are more apparent in command window
    - Don't have to do this
- Function [x] = shmotion2 (t)
- %SHMOTION2
- X = SCHMOTION2(T) returns the position X of a simple harmonic oscillator %evaluated at each time in the vector T
- End
- Body of function appears after help comments
- Contains MATLAB statements that are interpreted in a separate workspace
  - Variables that appear in body don't appear in user workspace, and vice versa
- **function [x] = shmotion2(t) ← user supplies t, and function computes x**
- **% help comments not shown**
- **% amplitude**
- **A = 2;**
- **% spring constant**
- **k = 10;**
- **% mass**
- **m = 1;**
- **% angular frequency**
- **omega = sqrt(k / m);**
- **% position**
- **x = A * sin(omega * t - pi/2); ← must assign value for every output variable**
- to use function, it must be in a folder on the current MATLAB path, can call invoke function in usual way
- >> t = 0:0.01:5;
- >>x= shmotion2(t);
- >>comet (t,x)
- So far this function is only a function of time, would be useful to make it a function for the other variables as well
- **function [x] = shmotion2(t, A, k, m) ← now use supplies t, a, K and m)**
- **% help comments not shown**
- **% angular frequency**
- **omega = sqrt(k / m);**
- **% position**
- **x = A * sin(omega * t - pi/2);**
- **end**

- What happens when a function is called?
  - All functions run in their own workspace (not main)

        - Variables in the function workspace are completely separate from variables in the main workspace even if they have the same name
    - How are argument values transferred from main to function workspace and vice versa? → Wikipedia evaluation strategies
- Call by value
    - Matlab provides the illusion of call by value
        - Values of the arguments are copied into the function workspace by assigning the values to the input variables
        - Example → shmotion2.m
            - The values get copied in to the shmotion 2 workspace from the main workspace, then the value for x is copied from shmotion2 workspace to main
    - Consequence
        - Function cannot change value of any variable not in functions workspace
            - Includes arrays ; if input argument is an array then any changes that the function makes to the array do not affect the caller's array

Day 7: Vectors and Matrices 1

- Arrays
  - Multidimensional table
  - Size of array of dimension k = d_1 x d_2 x … x d_k
  - D1 = number of rows, and d2 = number of columns
  - All matlab variables are multidimensional arrays
  - Size of array in matlab:
    - >>help size
  - Notion of an empty array exists
    - >> size ([])
- Scalars
  - Is an array of size 1 x 1
- Vectors
  - 2 dimensional array where one of the size of one of the dimensions is 1 (row vector = 1 x 3 , column vector = 5 x 1)
- Creating row vectors
  - Can be created directly by entering the values of the vector inside a pair of square brackets with the values separated by spaces or commas
    - >> v = [1 2 3 4]
    - V = 1 2 3 4
    - >> v = [1,2,3,4]
    - V = 1 2 3 4
  - Colon operator can be used to create row vectors having values that are equally spaced
    - >> v = 1:4
    - V = 1 2 3 4
  - can specify the spacing of values using the colon operator
    - >> v = 1:2:9
    - V= 1 3 5 7 9
    - >> v = 1:2:8
    - V= 1 3 5 7
    - >> v = 8:1
    - V = empty matrix 1 by 0
    - >> start = 5;
    - >> step = 5;
    - >>stop = 25;
    - >> v = start:step:stop
    - V = 5 10 15 20 25
  - Step size can be negative if start > stop
    - >>start = 25;
    - >>step = -5;
    - >>stop = 5;
    - V = 25 20 15 10 5
  - Observe stop value is not guaranteed to be at the end of the vector
    - >>start = 25;

        - >>step = -5;
        - >> stop = 6;
        - V = 25 20 15 10
    - Function **linspace** will generate a linearly spaced vector that includes the start and end values by calculating the step size for you
    - >> help linspace linspace Linearly spaced vector. linspace(X1, X2) generates a row vector of 100 linearly equally spaced points between X1 and X2. linspace(X1, X2, N) generates N points between X1 and X2. For N = 1, linspace returns X2.
- Creating column vectors
    - Can be created by entering values of vector inside a pair of square brackets with values separated by semi colons
    - >> v = [1;2;3;4]
    - V = 1
      2
      3
      4
    - Can be created from a row vector by transposing the row vector
    - >> v = [start : step : stop]' ← the single quote after a vector / matrix will compute the transpose of vector or matrix, the single quote is conjugate transpose operator, .' is the transpose operator
    - V = 25
      20
      15
      10
    - Can also be created from row vector by using colon notation like so
        - >> v = 1:4; ← the colon has two different uses in this example
        - >>w = v(:)
          w = 1
          2
          3
          4
- Number of elements in a vector
    - Function length will return the number of elements in the vector
        - >> v = [1 2 3 4];
        - >>length(v) ← does not compute the Euclidean length of a vector
          ans = 4
- Magnitude of a vector
    - Is what mathematicians call the norm of the vector
    - Many different norms
        - Euclidean norm (Euclidean length, L^2 norm, L^2 distance)
        - Taxicab norm (Manhattan norm, Manhattan distance, L^1 norm)
        - And more…
    - Use "norm" function to compute the vector norm, by default it computes the Euclidean norm

- >>v = [1 1]
- >>norm (v)
  ans = 1.4142

- Indexing Elements of a vector
  - Elements of the vector can be accessed by using an integer value called an index
  - MATLAB uses a 1-based index
    - First element of vector has index 1
    - Second has index 2, etc.
  - Use an index inside of ( ) after the vector name to access an element of the vector
  - >> v = -5:3
    v = -5 -4 -3 -2 -1 0 1 2 3
  - >> v(1) ← get the value of the first element in v
  - Ans = -5
  - V(3) = 100 ← set the value of the third element in v to 100
    - V = -5 -4 100 -2 -1 0 1 2 3
  - "end" used to access the last element of the vector
    - V(end)
      ans = 3
  - Can use arithmetic with end
    - V(end -1)
      ans = 2
  - The index does not need to be a scalar, can also be a vector of indices
    - >> v = -5:3
    - V([1 3 5])
      ans = -5 -3 -1
    - >>v([1 3 5]) = [7 8 9] ← set first, third, and fifth elements of v

- Arithmetic operations with arrays
  - Can perform element by element arithmetic with an array and a scalar

operator	name
+	addition
-	subtraction
*	multiplication
/	right division
\	left division
.^	array power

  - Can also perform element by element arithmetic with two arrays of the same size

operator	name
+	addition
–	subtraction
.*	multiplication
./	right array division
.\	left array division
.^	array power

- >> u = [1 2 3]
- >>v = [7 8 9]
- >>w = u .*v
  - W= 7 16 27
- W = u ./ v ← right array division (elements in u divided by v)
  - W = 0.1429 0.2500 0.333
- W = u .\ v ← left array division (elements in v divided by elements in u)
  - W = 7 4 3
- W = u .^ v (elements in u raised to power in v)
  - W = 1 32 729
- Examples:
  - Compute unit vector u of a non-unit vector v
    - U = v / norm (v)
  - Compute midpoint m of two points p and q
    - M = (p+q) / 2
  - Compute the dot product of vectors u and v
    - Dprod = sum(u .* v) or
    - Dprod = dot(u,v)

Day 8: Logicals

- Logicals
  - A logical expression is an expression that evaluates to either true or false
  - Logical variable is a variable whose value is either true or false
  - Logical variables called Boolean variables in computer science
- Logicals in Matlab
  - MATLAB uses 1 and 0 to represent true and false
    - Every logical expression will evaluate to either exactly 1 (true) or 0 (false), and the value of every logical variable will be either exactly 1 or 0
  - Matlab will convert any non-zero, non-NaN numeric value to logical 1
    - Only values = 0 are converted to logical 0
- Creating a logical variable
  - Literals for logical true and false are the non-keywords **true** and **false**
    - Rarely used, most people use 1 and 0
  - X = true
    x = 1
  - Y = false
    y = 0
- Relational operators
  - Logical values arise from logical expressions usually involving relational operator
  - Produce logical values by comparing two numbers

operator	name
>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to
==	equal to
~=	not equal to

  - Will operate in an element by element fashion for arrays
  - Can also compare a scalar versus an array
  - X = 1;
    y = 2;
    x > y
    ans = 0
  - X = 1:8 ;
    x >5
    ans = 0 0 0 0 0 1 1 1
    sum (x > 5) ← how many values greater than five
    ans = 3
- Example

- If you roll two 6 sided dice, what are odds that the sum of two dice is: Exactly 2? Exactly 7? Exactly 12? Less than 7? Greater than 7? Not 7?
  - Simulate many rolls and count number of times a condition is met, divide by number of rolls to get an estimate of the odds
  - nRolls = 1000000; %1
    die 1 = randi(6, 1, nRolls); %2
    die 2 = randi (6, 1, nRolls) %3
    total = die1 + die 2
  - %exactly 2?
    sum(total ==2)
    ans = 27943
    sum(total==2)/nRolls
    ans = 0.0279

- Logical operators
  - Operate on logical values
  - 5 operators, and 1 function

Operator	name
~	**not**
&	elementwise **and**
&&	short-circuit scalar **and**
|	elementwise **or**
||	short-circuit scalar **or**
xor	elementwise **exclusive or**

- Not ~
  - ~ = Boolean negation (called NOT)
  - NOT true is equal to false, NOT false is equal to true
- & elementwise and
  - % is Boolean conjunction (often called AND) applied elementwise

expression	result
true AND true	true
true AND false	false
false AND true	false
false AND false	false

  - X = [1 1 0 0]
    y = [1 0 1 0]
    x & y
    ans = 1 0 0 0

- | elementwise or
    - Boolean disjunction (called OR) applied elementwise

expression	result
`true OR true`	`true`
`true OR false`	`true`
`false OR true`	`true`
`false OR false`	`false`

    - X = [1 1 0 0]
      y = [1 0 1 0]
      x | y
      ans = 1 1 1 0

Day 9: Logicals continued

- Logical indexing
  - Can use logical array to perform indexing on another array
    - MATLAB extracts array elements corresponding to the nonzero values in the logical array
    - Output is always in the form of a column vector unless the array is a vector
    - X = 1:5
      x = 1 2 3 4 5
      I = x >3
      I = 0 0 0 1 1
      x(1) = 4 5
- Selection statements
  - So far, all scripts and functions have been made up of statements that are executed in sequence
  - Selection statement uses a logical value or values to select which statements are executed
- If statement
  - Allows you to conditionally execute a single block of MATLAB statements
  - If logical condition ← if logical condition is true then
    sequence of matlab statements do the sequence of matlab statements and then
    end
    more matlab statements do more matlab statements
  - If the logical condition is false, then skip the sequence of matlab statements, and skip to the more matlab statements
    - Find absolute value of a number x (don't do this use abs instead)
      if x < 0
      x = - x
      end
    - Print out an error message
      if degKelvin < 0
      error ('impossible temperature!');
      end
    - Change column vector to a row vector
      if size (x, 1) > 1
      x = x';
      end
    - Change row vector to a column vector
      if size (x, 2) >1
      x = x';
      end
    - Create a function that returns true if a resistor value is high enough to prevent an LED from failing
      function [tf] = rled(rSeries, vSupply, vLed, iLed)
      %RLED Validated LED resistance

%TF = RLED (RSERIES, VSUPPLY, VLED, ILED) returns TRUE if the resister RSERIES is large enough to prevent the LED from failing given a supply voltage VSUPPLY, forward voltage drop VLED, and operating current ILED
tf = false; ← assume false at first to prevent led from failing
if rSeries > (vSupply – vLed) /iLed
tf = true;
end
end

- If-else statement
  - Allows you to conditionally execute one of two blocks of matlab statements
  - If logical condition — if logical condition is true, do the first sequence of MATLAB statements — sequence of matlab statements, and then
    else
    sequence of matlab statements
    end
    more matlab statements — do more matlab statements
  - If logical condition is false, do the second sequence of matlab statements and then do more matlab statements
  - Write a function that returns true if an integer number is even and false if it is odd
    function [tf] = iseven (x)
    %ISEVEN true for an even number
    % TF = ISEVEN (X) returns 1 if x is an even integer number and 0 otherwise
    if rem(x,2) == 0 ← remainder of x divided by 2
    tf = true;
    else
    tf = false;
    end
  - Write a function that returns a sorted vector of two numbers
    function [sorted] = sort2(x, y)
    %SORT2 sort two numbers in ascending order
    % SORTED = SORT2(x, y) returns a row vector containing the numbers X and Y in ascending order
    if x <= y
    sorted [x y];
    else
    sorted = [y x] ;
    end — don't do this, there is a function "sort" instead
- If-elseif statement allows you to conditionally execute one of multiple blocks of MATLAB statements
  - If logical condition 1
    sequence of matlab statemets
    elseif logical condition 2
    sequence of MATLAB statements

```
else
sequence of matlab statements
more matlab statements
```

- Write a function that returns a sorted vector of three numbers

```
function [s] = smallest (x, y, z)
%SMALLEST Find the smallest of 3 numbers
% S = Smallest (X, Y, Z) returns the minimum of X, Y, and Z
if x <= y & x <= z
s = x;
elseif y <=x & y <= z
s = y;
else
s = z;
end                    don't do this, use the "min" function instead
```

- Write a function that categorizes blood pressure

```
function [category] = catpressure(systolic, diastolic)
% CATPRESSURE Categorize blood pressure measurement
%C = CATPRESSURE (SYSTOLIC, DIASTOLIC) returns the blood pressure category
defined by the American Heart Association for the given SYSTOLIC and
DIASTOLIC pressure measurements in mmHg. The turn value c is
0 for NORMAL
1 for PREHYPERTENSION
2 for HIGH BLOOD PRESSURE
if systolic >=140 | diastolic >= 90
category = 2:
elseif systolic >= 120 | diastolic >= 80
category = 1;
else
category 0
end
end
```

Day 10: Loops

- Loops
  - Allow you to repeatedly execute blocks of code
    - Each repetition is called an iteration
  - Two kinds of loops in matlab
    - For loop
      - Repeats a block of code a specific number of times, keeping track of each iteration using an incrementing loop variable
    - While loop
      - Repeats a block of code as long as a logical condition remains true
- For loop
  - Repeats a block of code once for each element of a control vector
    - Vector can be a multidimensional array, but ignore for now
  - Value of the element in the control vector is available inside the loop
  - For index = vector_of_values
    - Index = loop variable; you can use whatever name you want
    - Each iteration of the loop, the value of index is taken sequentially from vector_of_values
    - Loop body: a sequence of MATLAB statements repeated once for each element in vector_of_values

    End

  - %display the value of index for each iteration of a loop
    for index = 1:5
    index
    end

Iteration	Value of index
1	1
2	2
3	3
4	4
5	5

  - %display the integers 1 through 5 on separate lines
    for index = 1:%
    disp(sprintf('%d', index));
    end
    %or using fprintf
    for index = 1:5
    fprintf('%d\n', index);
    end
  - %display the integers 5 through 1 on separate lines
    for index = 5:-1:1

```
fprintf('%d\n', index);
end
```

Iteration	Value of index
1	5
2	4
3	3
4	2
5	1

- %display the integers start through stop on separate lines

```
if start <=stop
x = start:stop;
else
x = start:-1:stop;
end
for index = x
fprintf('%d\n', index);
end
```

- %compute the sum of the integers 1, 2, 3, …, n

```
%we need a variable to accumulate the sum
total = 0;
for x = 1:n
total = total +x;
end
```

- For loop: radioactive decay
  - In radioactive decay, an energetically unstable atom spontaneously emits energy in the form of ionizing radiation
  - For a single atom, decay occurs at random
    - For many atoms, decay occurs at an average constant rate α
  - Suppose you start with N radioactive atoms:
    U(1) = N
  - After 1 unit of time there will be:
    U(2) = (1- α)U(1) ← for some constant value α
  - After 2 units of time there will be:
    U(3)=(1- α)U(2)
  - And so on
  - %compute the number of atoms at each time

```
%t = 1, 2, 3, …, 10
N = 100000;
alpha = 0.05;
%we ned a vector to store the results
U=zeros(1,10);
U(1)=N;
```

for index = 2:length(U)
U(index) = (1-alpha) * U(index-1);
end

- For loop: cumulative sum
  - Cumulative sum of the elements in a vector **values** is a vector of the same length as **values** where the element at index I is the sum of **values (1)** through **values (i)**
  - %compute the cumulative sum of the elements in a vector names values
    csum = zeros(1, length(values));
    csum(1) = values(1);
    for index = 2:length(values)
    csum (index) = csum(index-1)+values(index);
    end
    % you should use cumsum instead, though
    csum = cumsum(values);
    values = [22.59 485.33 11.54 14.80 30.03]

Iteration	Value of index	Value of csum
1	2	22.59
2	3	507.92
3	4	519.46
4	5	534.26
		564.29

- While loop
  - Represents block of code as long as logical condition is true
    - Unlike a for loop
      - There is no loop variable
      - Number of times that loop runs is not necessarily determined ahead of time
  - While logical_condition ← if logical_condition is true then the loop body is run once
    loop body: a sequence of MATLAB statements
    end after the loop body is run, the loop restarts by checking the logical_condition
  - %repeat a loop until the user inputs 'y'
    repeat = 1;
    while (repeat)
    %
    % some code here you want to repeat
    %
    % ask the user if they want to repeat again
    answer = input ('Continue? (y/n)', 's');

repeat = strcmp (answer, 'y'); ←strcmp compares strings, if string is y, it will repeat
    - end
- while loop: infinite loops
    - observe that it is very easy to create an infinite loop using a while loop
        - must ensure that whatever happens in the loop body eventually causes the logical condition to become false
    - if you encounter an infinite loop in your program you can press Ctrl + c to stop your program
        - unfortunately this stops your entire program and not just your loop
    - %infinite loop example
      repeat = 1;
      %
      % some code here you want to repeat
      %
      %ask the user if they want to repeat again
      answer = input ('Continue? (y/n)', 's');
      end
- While loop: infinite loops?
    - Collatz conjecture
        - Starting with a positive integer number n repeat the following until you reach a value of n = 1
            - If n is even: n = n/2
            - Otherwise (n is odd): n = 3*n+1
    - Examples:
        - N = 4: 4, 2, 1
        - N = 6: 6, 3, 10, 5, 16, 8, 4, 2, 1
        - N = 17: 52, 26, 13, 40, 20, 5, 16, 8, 4, 2, 1
    - Conjecture state that you will eventually reach 1 for any value of n
    - Function iter = collatz(n)
      %COLLATZ Collatz sequence
      % ITER = COLLATZ (N) returns the number of iterations ITER for the Collatz conjecture starting from the positive integer number n.
      iter = 0;
      while n ~= 1
      %if n is even divide it by 2
      %otherwise, triple it and add 1
      if rem(n, 2)==0
      n = n/2
      else
      n = 3 * n + 1
      end
      iter = iter +1;

end
end

- Collatz (63728127) takes 949 iterations to stop
- Collatz(9780657631) takes 1132 iterations to stop
- It is unknown if the Collatz conjecture is true
    - There might exist some starting value n that never reaches 1
- Halting problem
    - Given a computer program and an input decide whether the program will eventually halt when run with that input, or will it run forever.