Topics: One-dimensional array—vector, easy plots in MATLAB
Reading: CFile Chapter 5 Sec 5.1

1-Dimensional Array: Vector

An array is a *named* collection of data values organized into rows and/or columns. A 1-d array is a row or a column, also known as a *vector*. An *index* is a positive integer that identifies the position of a value in the vector. MATLAB **array index starts at 1**, not zero. To access a value in an array, use parentheses to enclose the index value. For example, `x(2)` is the value in the 2nd cell of vector `x`. MATLAB distinguishes between *row* and *column* vectors. Use *square brackets* to delimit arrays.

Creating a vector

MATLAB function **zeros**: **vecA= zeros(1,5)**

MATLAB function **ones**: **vecB= ones(5,1)**

MATLAB short-cut expression for consecutive numbers: **1:6** or **1:1:6**
Note that the syntax is ⟨*left bound*⟩**:**⟨*increment*⟩**:**⟨*right bound*⟩ , so the expression **7:-2:0** gives [7 5 3 1].

Assignment: **vecC(5) = 9** gives [0 0 0 0 9]

Build using square brackets: **vecD = [2 3.5 6]**

Example 1

Write a program fragment that calculates the *cumulative sums* of a given vector **v**. The cumulative sums should be stored in a vector of the same length as **v**. E.g., the cumulative sums for the sequence 1,3,5,0 is 1,4,9,9. Do not use MATLAB predefined functions other than **length**.

Example 2

Write a function **evalPoly** to evaluate an n^{th} order polynomial of x:

$$a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

The input parameters are **coef** and **x** where **coef** has length $n+1$ and contains the coefficients of the polynomial and **x** is the value at which to evaluate the polynomial. Return the evaluated value. Note that **coef(1)** is the coefficient for the term x^0. Do not use MATLAB predefined functions other than **length**.

```
function val= evalPoly(coef,x)
% val is the value of a polynomial with coefficients coef evaluated at x.
% coef is a vector and coef(1) is the coefficient for the term x^0.
```

Example 3: A random walk with graphics

Write a function **randomWalk** that performs n steps of a "random walk" starting from position (x_0, y_0) and draws the path. In a random walk, possible moves are left, right, up, or down (in a Cartesian plane).

```
function randomWalk(n,x0,y0)
% Perform n steps of random walk starting from position (x0,y0).  Display the path.

% possible movements:  ( deltaX(i), deltaY(i) )

deltaX=

deltaY=


x= [x0 zeros(1,n)];  % trajectory in x direction
y= [y0 zeros(1,n)];  % trajectory in y direction

% Perform walk, each step is based on a random integer
for k = 2:n+1
  % get a random integer in (1..4)
    r=

  % take the step

    x(k)=

    y(k)=
end

% Show the walk
plot(x,y,x(1),y(1),'r*',x(end),y(end),'ro')
axis('equal')
title([num2str(n) ' steps of random walk from * to o'])
```

Plotting

It is very easy to make plots using MATLAB. An x-y plot can be generated using the built-in function **plot**. The command

plot(a,b,'-', c,d,'*')

will generate a plot with two graphs, one showing the data contained in vectors **a** (in x-direction) and **b** (in y-direction) as a *line* and the other showing the data in vectors **c** and **d** as *asterisks*. Use the *help* facility in MATLAB to learn more about **plot** and the many formatting options. If you omit the formatting option (**'-'** and **'*'** above), the default on most system is to show the data as a line.