Sqrt(x) nthroot(x, n) exp(x) abs(x) log(x) log10(x) factorial(x) sin(radians) sind(degrees)

Round(x) fix(towards 0) ceil(twd infinity) floor(twd –infinity) rem(x,y) sign(x) eps(x)

Rowvector = [1 2 3] Colvector = [1; 2; 3] 1stspacinglast = [1:3:13] V=linspace(1st, last, numelements)

Matrix=[1strow; 2ndrow; 3rdrow] zr=zeros(3,4) one=ones(3,4) diag=eye(5) trans = A’

VCT(4) = fourthelement va(:) = allelementsinvector va(m:n) = elements’m’to’n’ KNH=[RE GT]

Kt(delete) = [ ] length(A) size(A) reshape(A, m, n) diag(vector) = elementsdiagwithallzeros diag(matrix)=Vofdiag

B= ‘My name is’ C=inv(A) \-->to solve AX=B for X .\* (and ./, .^ etc)….element-by-element (before exponents)

Mean(A) C=max(A) C=min(A) sum(A) sort(Ascending) median(A) std(A)

Rand rand(1, n)🡪n-element row vector of rand #s rand(n)🡪n x n matrix of rands

rand between A-B 🡪 (b-a)\*rand+a (and with round makes integers) randn 🡪normal dist of rands

Var=input(‘textstring to be displayed’) 🡪 requests user input before proceeding

Disp(Var) 🡪 displays variable/array fprintf(‘text typed as string’) -🡪displays, and with formatting

Fprintf(‘text %-5.2f other text’, variablename)

Save file\_name load file\_name or load(‘file\_name’) function[output args] = function\_name (input args)

>>5>8 (ans = 0 bec it’s false) d = c>= b (checks which elements of c are >= to the b elements)

& (AND—if both true, result is 1. If not, 0)

| (OR---if either or both true, result is 1. If not, 0)

~ (NOT---gives opposite of operand)

Also, nonzero numbers are true while zero is false (3&7 🡪 ans = 1)

Xor(a, b) (exclusive or. Is true if one operand is true and the other false)

All(A) 🡪 is true if all elements in vector are true (nonzero)

Any(A) 🡪 is true if any element in vector is true (nonzero)

Find(A) 🡪if A is vector, gives location of true elements

If-end: if cond expression is true, program executes following commands to ‘end’ statement

If-else-end: if cond expression is true, prog executes following commands to ‘end’. If false, skips to false and executes other group of commands, then skips to end.

If-elseif-else-end: if cond expression is false, gives another cond expression with two sets of commands again

Switch-case: matches expression to one of the following cases and if match, executes that case

For-end---🡪 for k = f:s:t (f is value of k in 1st pass, s is increment, t is last pass value)

While-end 🡪 while ‘conditional expression’

Plot(x, y, ‘line specifiers’, ‘PropertyName’, PropertyValue) fplot(‘function’, limits, ‘line specifiers’)

Limits: vector which specifies domain of x … [xmin, xmax]

To have multiple plots:

(1) plot(x, y, x, y2, x, y3) OR

(2) plot(x, y)

hold on

plot(x, y2)

hold off

(3) plot(x, y)

Line(x, y2)

Line(x, y3)

To format plots, input these after the *plot* or *fplot* commands:

xlabel (‘text as string’)

ylabel (‘text as string’)

title(‘text as string’)

text(x,y,’text as string’) -🡪 for a text label with coordinates (x, y)

legend(‘string1’, ‘string2’, …, pos)

grid on 🡪 adds grid lines to plot grid off 🡪 removes grid lines from the plot

stairs(x, y) stem(x, y) pie(x) hist(vector) OR hist(y, nbins) w/nbins = scalar

polar(theta, radius, ‘line specifiers’)

subplot(m, n, p) 🡪 *m* x *n* rectangular subplots; p makes that plot current

figure --- cmd executed means MATLAB opens new Figure Window

global variable\_name

name = @ (arglist) expr e.g. cube = @ (x) x^3

inline fxn --- instead of M-script file name = inline (‘math expression typed as a string’) cube = inline (‘x^3’)

a function function: accepts another function in order to work

function handle: a name which can be used to pass as an argument into another function (created by typing **@** in front of function name (e.g. @cos is handle of fxn cos)

fxn fxn: funplot(Fun,a,b) Input args 🡪 (Fun, a, b) … Fun is dummy name for imported fxn, a,b are domain endpoints

passing a user-defined fxn into fxn fxn:

function y = Fdemo(x) 🡪 ydemo = funplot(@Fdemo, 0.5, 4)

y = exp(-0.17\*x).\*x.^3-2\*x.^2+0.8\*x-3

‘feval’ cmd evaluates value of fxn for a given value of argument: variable = feval(‘function name’, argument value)

X = feval (‘sqrt’, 64) 🡪 ans = 8

Variables declared in functions are local unless specifically declared to be global

Sort cmd: A = [ 3 7 5

6 8 3

0 4 2 ];

>> [B, IB] = sort(A,1,’descend’) … the 1 is sort by column, a 2 is for sort by row

B = 6 8 5

3 7 3

0 4 2

IB = 2 2 1

1 1 2

3 3 3

Reverse a vector:

function [ outVector ] = reverseMe( inVector )

outVector = inVector(end:-1:1);

end

-------

function [ vsumprod ] = SumOfAdjacentProducts( v )

vsumprod = 0;

for iCount = 1:2:length(v)

vsumprod = vsumprod + v(iCount)\*v(iCount+1);

end

end

*Alternate answer, without using a loop:*

function [ vsumprod ] = SumOfAdjacentProducts1( v )

vsumprod = v(1:2:end-1)\*v(2:2:end)’;

end

cell arrays use { }, not [ ] x = {1 2 3 4 5} 🡪 x = [1] [2] [3] [4] [5]

so, for ‘content indexing’ – to select the contents of cells to work with--- us { } like janet {1, 4} = bob (and bob is matrix)

plot3(x, y, z, ‘line specifiers’, ‘PropertyName’, property value) xyz are vectors with equal # of elements

mesh/surface plots-🡪 w/in given domain value of z can be calculated for any combo of x and y

1. Create x-y grid defined by matrices….. use X and Y as matrices [X, Y] = meshgrid (x, y)
2. Calculate value for z (use z = fxn with element-by-element ops)
3. Use mesh (X, Y, Z) or surf (X, Y, Z) OR mesh (Z)

Grid can be turned off using ‘grid off’ command

Contour3 (X, Y, Z) sphere (n) [X, Y, Z] = cylinder (r) bar3 (Y) scatter3 (X, Y, Z) pie3 (X, explode)

At end of code, put xlabel (‘x’) ylabel (‘y’) zlabel (‘z’)

view (az, el)

function daMeans = makeMeans3D (daData, levelsX, levelsY)

numXs = length (levelsX)

numYs = length (levelsY)

daMeans = Nan (numYs, numXs)

for x = 1:numYs

xyData = daData (daData(:,2) == levelsX(x) & daData(:,3) == levelsY (y), :)

myMean = mean (xyData)

daMeans (y, x) = myMean (1)

end

end