Project 0

Exercise 1

Part 1. Creating matricies

```
clear
A=[-1 4 7 10; -2 5 8 11; -3 6 9 12]
```

```
A = 3×4
    -1     4     7    10
    -2     5     8    11
    -3     6     9    12
```

```
B=[-1 2; -3 4; -5 6]
```

```
B = 3×2
    -1     2
    -3     4
    -5     6
```

```
X=[1; 3; 5]
```

```
X = 3×1
     1
     3
     5
```

```
x=[2 4 6 8]
```

```
x = 1×4
     2     4     6     8
```

```
y=[0; 1; 2; 3]
```

```
y = 4×1
     0
     1
     2
     3
```

```
size(A)
```

```
ans = 1×2
     3     4
```

```
size(B)
```

```
ans = 1×2
     3     2
```

```
size(X)
```

```
ans = 1×2
     3     1
```

```
size(x)
```

```
ans = 1×2
     1     4
```

```
size(y)
```

```
ans = 1×2
     4     1
```

Matricies x and y are not the same size because they do not have the same number of columns and rows as each other.

```
size(A,1)
```

```
ans = 3
```

```
size(A,2)
```

```
ans = 4
```

The outpur of size(A,1) produces the number of rows of matrix A and size(A,2) produces the number of columns in matrix A.

Part 2. Accessing particular matrix entries and changing entries

```
A, A(1,3), A(:,3), A(2,:), A(:, 1:3)
```

```
A = 3×4
    -1     4     7    10
    -2     5     8    11
    -3     6     9    12
ans = 7
ans = 3×1
     7
     8
     9
ans = 1×4
    -2     5     8    11
ans = 3×3
    -1     4     7
    -2     5     8
    -3     6     9
```

The output for A constructs a matrix that was previously established. The output of A(1,3) gives the entry value at row 1 and column 3. The output of A(:,3) gives you all entries of column 3 The command A(2,:) outputs all entries of column in row 2. The command A(:, 1:3) outputs a matrix of all rows for the columns 1 through 3.

```
A, A([1 2], [2 4])
```

```
A = 3×4
    -1     4     7    10
    -2     5     8    11
    -3     6     9    12
ans = 2×2
     4    10
     5    11
```

The command A outpurts the matrix A. The command A([1 2], [2 4]) outputs a matrix with row 1 having the values of A of row 1 and column 2, and row 1 and column 4. The second row in the matrix has outputs of row 2 and column 2, and row 2 and column 4.

```
F(:,4)=[-1 1 -4 3], F([1 3], [2 3]) = [1 -3; 2 -5]
```

```
F = 4×4
     0     0     0    -1
     0     0     0     1
     0     0     0    -4
     0     0     0     3
F = 4×4
     0     1    -3    -1
     0     0     0     1
     0     2    -5    -4
     0     0     0     3
```

The command F(:,4) = [-1 1 -4 3] creates a matrix of number of input of rows and 4 columns. In column 4, the values on the right will be assigned to each of the following rows. The command F([1 3], [2 3]) = [1 -3; 2 -5] puts the values on the right to the rows and columns it wanted on. Value 1 will be on row 1 and column 2, -3 will be on row 1 and column 3, 2 will be on row 3 and column 2, and -5 will be on row 3 and column 3.

```
A, F, F([2 3], :) = A([1 3], :)
```

```
A = 3×4
    -1     4     7    10
    -2     5     8    11
    -3     6     9    12
F = 4×4
     0     1    -3    -1
     0     0     0     1
     0     2    -5    -4
     0     0     0     3
F = 4×4
     0     1    -3    -1
    -1     4     7    10
    -3     6     9    12
     0     0     0     3
```

The command A outputs matrix A. The command F outputs matrix F. The command F([2 3], :) = A([1 3], :) uses rows 1 and 3 of matrix A and copies all the values for each column and places those values in rows 2 and 3 in matrix F.

```
F, F(:, [1 2])=F(:, [2 1])
```

```
F = 4×4
     0     1    -3    -1
    -1     4     7    10
    -3     6     9    12
     0     0     0     3
F = 4×4
     1     0    -3    -1
     4    -1     7    10
     6    -3     9    12
     0     0     0     3
```

The commnad F outputs matrix F. The commnad F(:, [1 2]) = F(:, [2 1]) swaps the values of each row for column 1 to column 2 and column 2 to column 1.

```
F, F(2:3,:), F(end,:)
```

```
F = 4×4
     1     0    -3    -1
     4    -1     7    10
     6    -3     9    12
     0     0     0     3
ans = 2×4
     4    -1     7    10
     6    -3     9    12
ans = 1×4
     0     0     0     3
```

The command F outputs matrix F. The command F(2:3,:) gives the matrix for rows 2 to 3 of F and all the columns values for both rows. The command F(end, :) outputs the values of all columns for the last row in matrix F.

```
y, F, F(2,:) = y
```

```
y = 4×1
     0
     1
     2
     3
F = 4×4
     1     0    -3    -1
     4    -1     7    10
     6    -3     9    12
     0     0     0     3
F = 4×4
     1     0    -3    -1
     0     1     2     3
     6    -3     9    12
     0     0     0     3
```

```
F([1 3],:) = F([3 1],:)
```

```
F = 4×4
     6    -3     9    12
     0     1     2     3
     1     0    -3    -1
     0     0     0     3
```

```
F1(:,1:3) = F(:,1:3)
```

```
F1 = 4×3
     6    -3     9
     0     1     2
     1     0    -3
     0     0     0
```

```
F(:, 4), b(:,1) = F(:, 4)
```

```
ans = 4×1
    12
     3
    -1
```

```
     3
b = 4×1
    12
     3
    -1
     3
```

Part 3. Pasting blocks together (concatenation)

```
A,B,[A B],[B A]
```

```
A = 3×4
    -1     4     7    10
    -2     5     8    11
    -3     6     9    12
B = 3×2
    -1     2
    -3     4
    -5     6
ans = 3×6
    -1     4     7    10    -1     2
    -2     5     8    11    -3     4
    -3     6     9    12    -5     6
ans = 3×6
    -1     2    -1     4     7    10
    -3     4    -2     5     8    11
    -5     6    -3     6     9    12
```

```
A,X,[A X]
```

```
A = 3×4
    -1     4     7    10
    -2     5     8    11
    -3     6     9    12
X = 3×1
     1
     3
     5
ans = 3×5
    -1     4     7    10     1
    -2     5     8    11     3
    -3     6     9    12     5
```

```
A,x,[A ; x]
```

```
A = 3×4
    -1     4     7    10
    -2     5     8    11
    -3     6     9    12
x = 1×4
     2     4     6     8
ans = 4×4
    -1     4     7    10
    -2     5     8    11
    -3     6     9    12
     2     4     6     8
```

```
blkdiag(B,A)
```

```
ans = 6×6
    -1     2     0     0     0     0
    -3     4     0     0     0     0
```

```
    -5     6     0     0     0     0
     0     0    -1     4     7    10
     0     0    -2     5     8    11
     0     0    -3     6     9    12
```

The matricies were created by first constructing the first matrix (such as A) for first rows and columns and would create additional columns or rows to add the second matrix depending if the : commnad used between the two matrix command. The command blkdiag(B,A) constructs matrix B based on its size. After B is constructed, matrix A is constructed on its on rows and columns, usually diagonally of the last row and column of matrix B. The constructs a large matrix of 6 rows and 6 columns.

Part 4. Some special functions for creating matricies

```
eye(5), zeros(3,4), zeros(2), ones(3,2), ones(5)
```

```
ans = 5×5
     1     0     0     0     0
     0     1     0     0     0
     0     0     1     0     0
     0     0     0     1     0
     0     0     0     0     1
ans = 3×4
     0     0     0     0
     0     0     0     0
     0     0     0     0
ans = 2×2
     0     0
     0     0
ans = 3×2
     1     1
     1     1
     1     1
ans = 5×5
     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1
```

The command eye(5) creates a matrix of 5 rows by 5 columns with a 1 following a straight diagonal of each of its own row and column. The command zeros(3,4) creates a matrix of three rows and four columns with all the values equaling to 0. The command zeros(2) creates a matrix of 2 rows by 2 columns with all values equaling to 0.The command ones(3,2) creates a matrix of three rows and two columns with all the values equaling to one. The command ones(5) creates a matrix of five rows by five columns with all values being one.

```
diag([1 2 5 6 7]), diag([1 2 5 6 7],1), diag([1 2 5 6 7],-2)
```

```
ans = 5×5
     1     0     0     0     0
     0     2     0     0     0
     0     0     5     0     0
     0     0     0     6     0
     0     0     0     0     7
ans = 6×6
     0     1     0     0     0     0
     0     0     2     0     0     0
```

```
     0     0     0     5     0     0
     0     0     0     0     6     0
     0     0     0     0     0     7
     0     0     0     0     0     0
ans = 7×7
     0     0     0     0     0     0     0
     0     0     0     0     0     0     0
     1     0     0     0     0     0     0
     0     2     0     0     0     0     0
     0     0     5     0     0     0     0
     0     0     0     6     0     0     0
     0     0     0     0     7     0     0
```

The command dia([1 2 5 6 7]) constructs a matrix of five rows by five rows with all values equaling to zero except from starting from the top right with the first value to each value following down diagonally. The command diag([1 2 5 6 7],1) is simialar to the command previously but it creates a column of all 0s before adding in it values til it ends. The command diag([1 2 5 6 7], -2) is similar to the previous command but the negative affects the rows rather than columns. It first creats two rows of zeroes befores creating a diagonal representation of the values listed.

```
B, diag(B), diag(diag(B))
```

```
B = 3×2
    -1     2
    -3     4
    -5     6
ans = 2×1
    -1
     4
ans = 2×2
    -1     0
     0     4
```

The command B creates the matrix B. The command diag(B) creates a matrix with one column with all the values going diagonally be on each row. The command diag(diag(B)) creates a matrix of number of columns and rows based on how many diagonal values that exist and creates a matrix with each value going diagonally use the values from diag(B) to place to each row and column.

```
A, triu(A), tril(A)
```

```
A = 3×4
    -1     4     7    10
    -2     5     8    11
    -3     6     9    12
ans = 3×4
    -1     4     7    10
     0     5     8    11
     0     0     9    12
ans = 3×4
    -1     0     0     0
    -2     5     0     0
    -3     6     9     0
```

The command creates the matrix A. The command triu(A) makes all values below the diagonal line all become 0. The command tril(A) makes all values above the diagonal line become 0.

```
A, triu(A,1), triu(A,-1)
```

```
A = 3×4
    -1     4     7    10
    -2     5     8    11
    -3     6     9    12
ans = 3×4
     0     4     7    10
     0     0     8    11
     0     0     0    12
ans = 3×4
    -1     4     7    10
    -2     5     8    11
     0     6     9    12
```

The command A creates the matrix A. The command triu(A,1) makes all values below the diagonal line including one additional diagonal line become zero which causes the main diagonal line become all zeroes. The command triu(A, -1) makes all values below the diagonal line except the one diagonal line close to the main diagonal line, to all become zeroes.

```
A, tril(A,-1), tril(A,2)
```

```
A = 3×4
    -1     4     7    10
    -2     5     8    11
    -3     6     9    12
ans = 3×4
     0     0     0     0
    -2     0     0     0
    -3     6     0     0
ans = 3×4
    -1     4     7     0
    -2     5     8    11
    -3     6     9    12
```

The command A creates the matrix A. The command tril(A,-1) makes all values above the diagonal line including one additional diagonal line become zero which causes the main diagonal line become all zeroes. The command tril(A, 2) makes all values above the diagonal line except the two diagonal line close to the main diagonal line, to all become zeroes.

```
magic(5), help magic
```

```
ans = 5×5
    17    24     1     8    15
    23     5     7    14    16
     4     6    13    20    22
    10    12    19    21     3
    11    18    25     2     9
 magic  Magic square.
    magic(N) is an N-by-N matrix constructed from the integers
    1 through N^2 with equal row, column, and diagonal sums.
    Produces valid magic squares for all N > 0 except N = 2.

    Documentation for magic
```

The magic command createed a 5 by 5 matrix with each column, row, and diagonal sums all being equal. Each entry has to be a number 1 to 25.

```
hilb(5), help hilb
```

```
ans = 5×5
    1.0000    0.5000    0.3333    0.2500    0.2000
    0.5000    0.3333    0.2500    0.2000    0.1667
    0.3333    0.2500    0.2000    0.1667    0.1429
    0.2500    0.2000    0.1667    0.1429    0.1250
    0.2000    0.1667    0.1429    0.1250    0.1111
 hilb   Hilbert matrix.
    H = hilb(N) is the N-by-N matrix with elements 1/(i+j-1), which is a
    famous example of a badly conditioned matrix. The INVHILB function
    calculates the exact inverse.

    H = hilb(N,CLASSNAME) returns a matrix of class CLASSNAME, which can be
    either 'single' or 'double' (the default).

    hilb is also a good example of efficient MATLAB programming
    style, where conventional FOR or DO loops are replaced by
    vectorized statements.

    Example:

    hilb(3) is

           1.0000    0.5000    0.3333
           0.5000    0.3333    0.2500
           0.3333    0.2500    0.2000

    See also invhilb.

    Documentation for hilb
```

The hilb creates a function using the input as the size of the matrix. The matrix itself starts at 1 at the first row and column but would start to decrease for each additional column or row it is away from the top left corner.

```
pascal(4), help pascal
```

```
ans = 4×4
     1     1     1     1
     1     2     3     4
     1     3     6    10
     1     4    10    20
 pascal Pascal matrix.
    P = pascal(N) is the Pascal matrix of order N. P a symmetric positive
    definite matrix with integer entries, made up from Pascal's triangle.
    Its inverse has integer entries. pascal(N).^r is symmetric positive
    semidefinite for all nonnegative r.

    P = pascal(N,1) is the lower triangular Cholesky factor (up to signs of
    columns) of the Pascal matrix. P is involutory, that is, it is its own
    inverse.

    P = pascal(N,2) is a rotated version of pascal(N,1), with sign flipped
    if N is even. P is a cube root of the identity matrix.

    P = pascal(...,CLASSNAME) returns a matrix of class CLASSNAME, which
    can be either 'single' or 'double' (the default).

    Example:

    pascal(4) is

           1     1     1     1
           1     2     3     4
           1     3     6    10
```

```
     1     4    10    20
```

```
Documentation for pascal
```

The pascal command creates a 4 by 4 column and row matrix which it creates a upside down pascal triangle.

```
C = eye(3)
```

```
C = 3×3
     1     0     0
     0     1     0
     0     0     1
```

```
D = diag([1 3 5], -1)
```

```
D = 4×4
     0     0     0     0
     1     0     0     0
     0     3     0     0
     0     0     5     0
```

```
E = ones(2,3)
```

```
E = 2×3
     1     1     1
     1     1     1
```

Part 5. Using the colon (vectorized statement) to create a vector with evenly spaced entries

```
V1=1:7, V2=2:0.5:6.5, V3=3:-1:-5
```

```
V1 = 1×7
     1     2     3     4     5     6     7
V2 = 1×10
    2.0000    2.5000    3.0000    3.5000    4.0000    4.5000    5.0000    5.5000 ···
V3 = 1×9
     3     2     1     0    -1    -2    -3    -4    -5
```

```
V4=-5:1:1, V5=10:-3:-2, V6=5:-0.5:2, V7=0:-0.5:-3
```

```
V4 = 1×7
    -5    -4    -3    -2    -1     0     1
V5 = 1×5
    10     7     4     1    -2
V6 = 1×7
    5.0000    4.5000    4.0000    3.5000    3.0000    2.5000    2.0000
V7 = 1×7
         0   -0.5000   -1.0000   -1.5000   -2.0000   -2.5000   -3.0000
```

Part 6. Format commands

```
%(a)
r=sqrt(2)
```

```
r = 1.4142
```

```
sym(r)
```

```
ans = √2
```

```
format long, r
```

```
r =
   1.414213562373095
```

```
format rat, r
```

```
r =
    1393/985
```

```
format short, r
```

```
r = 1.4142
```

```
format
r
```

```
r = 1.4142
```

```
disp(r)
```

```
    1.4142
```

```
%(b)
P=0.0000632178
```

```
P = 6.3218e-05
```

```
format long, P
```

```
P =
     6.321780000000000e-05
```

```
format short, P
```

```
P = 6.3218e-05
```

The format command chagnes how the values of r and P are presented and save. Having format ratio would give you a ratio of a the value you inputed. Format short would produce a shorty version of the value than long would in decimal form.

Part 7. Operations on Matricies

```
A, A+A
```

```
A = 3×4
    -1     4     7    10
    -2     5     8    11
    -3     6     9    12
ans = 3×4
    -2     8    14    20
    -4    10    16    22
    -6    12    18    24
```

```
A, 3*A
```

```
A = 3×4
    -1     4     7    10
    -2     5     8    11
    -3     6     9    12
ans = 3×4
    -3    12    21    30
    -6    15    24    33
    -9    18    27    36
```

```
T=eye(4), U=magic(4), T+U
```

```
T = 4×4
     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1
U = 4×4
    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
ans = 4×4
    17     2     3    13
     5    12    10     8
     9     7     7    12
     4    14    15     2
```

```
x, y, x+y
```

```
x = 1×4
     2     4     6     8
y = 4×1
     0
     1
     2
     3
ans = 4×4
     2     4     6     8
     3     5     7     9
     4     6     8    10
     5     7     9    11
```

The x+y command creates a matrix where it would at location row n and column m where it uses the value of x at m and and value of y at n whcih produces the sum of both number at location n,m of the new matrix.

```
A, A', transpose(A)
```

```
A = 3×4
    -1     4     7    10
    -2     5     8    11
    -3     6     9    12
ans = 4×3
    -1    -2    -3
     4     5     6
     7     8     9
    10    11    12
ans = 4×3
    -1    -2    -3
     4     5     6
     7     8     9
    10    11    12
```

The command A' and transpose(A) both do the same function but are just written differently to be executed. The matrix to rearranged with the values of columns are now rows and the value of rows are now columns.

Part 8. Matrix multiplication

```
x, y, x*y, y*x
```

```
x = 1×4
     2     4     6     8
y = 4×1
     0
     1
     2
     3
ans = 40
ans = 4×4
     0     0     0     0
     2     4     6     8
     4     8    12    16
     6    12    18    24
```

The result of x*y and y*x are completly different. The output of x*y gives the dot product of x and y which gives a single value because it took the size of rows of x and size of columns for y. On the other hand, y*x outputs a matrix with all rows being multiplied by each column which produces a matrix.

```
P=[1 2 3 4; 1 1 1 1], Q= transpose(P)
```

```
P = 2×4
     1     2     3     4
     1     1     1     1
Q = 4×2
     1     1
     2     1
     3     1
     4     1
```

```
P*Q, P.*P, P.^2
```

```
ans = 2×2
    30    10
    10     4
ans = 2×4
     1     4     9    16
     1     1     1     1
ans = 2×4
     1     4     9    16
     1     1     1     1
```

The command P*Q outputs a matrix multiplication which creates a matrix being size of 2 by 2. The command P.*P and P.^2 both produce the same output. Each value at each location are multiplied with each other which outputs a new matrix.

```
G = [1 2 3; 4 5 6; 7 8 9]
```

```
G = 3×3
     1     2     3
     4     5     6
```

```
     7     8     9
```

```
G^2, G*G
```

```
ans = 3×3
    30    36    42
    66    81    96
   102   126   150
ans = 3×3
    30    36    42
    66    81    96
   102   126   150
```

```
G*G==G^2, isequal(G*G,G^2)
```

```
ans = 3×3 logical array
   1   1   1
   1   1   1
   1   1   1
ans = logical
   1
```

Part 9. Creating matricies with random entries

```
rand(4), rand(3,4), randi(100,2), randi(10,2,4), randi([10 40],2,4)
```

```
ans = 4×4
    0.8759    0.2077    0.8443    0.2277
    0.5502    0.3012    0.1948    0.4357
    0.6225    0.4709    0.2259    0.3111
    0.5870    0.2305    0.1707    0.9234
ans = 3×4
    0.4302    0.9797    0.2581    0.2622
    0.1848    0.4389    0.4087    0.6028
    0.9049    0.1111    0.5949    0.7112
ans = 2×2
    23    30
    12    32
ans = 2×4
     5     1     9    10
     6     3     1     8
ans = 2×4
    25    17    39    26
    27    24    26    17
```

The rand(4) command creates a 4 by 4 matrix with values ranging from 0 to 1. The rand(3,4) command creates a 3 by 4 matrix with values ranging from 0 to 1. The randi(100,2) command creates a 2 by 2 matrix wiht the numbers ranging from 0 to 100 as integers. The randi(10,2,4) command creates a 2 by 4 matrix with numbers ranging from 0 to 10. The randi([10 40],2,4) creates a 2 by 4 matrix with numbers ranging from 10 to 40.

```
5*rand(3), -3+5*rand(3)
```

```
ans = 3×3
    2.4445    1.9776    0.1887
    3.1203    1.8372    4.4258
    3.3957    4.9399    4.5664
ans = 3×3
    0.9809   -1.3232    0.6061
   -2.5064    0.3986   -2.4662
   -1.6906   -2.3172    0.2688
```

```
-1+6*rand(2,4), randi([-20 20],2,4)
```

```
ans = 2×4
    1.9650    3.2902    4.3455    3.1925
    3.6743    4.4223    1.0050    0.1869
ans = 2×4
   -19     0    17     5
    10    -1     5    15
```

Exercise 2: Programming Techniques

```
M=3*ones(3), N=[0 0 3; 0 3 0; 0 0 3], L=3*ones(3,2)
```

```
M = 3×3
     3     3     3
     3     3     3
     3     3     3
N = 3×3
     0     0     3
     0     3     0
     0     0     3
L = 3×2
     3     3
     3     3
     3     3
```

Part 1. Logical Operations

```
isequal(M,L)
```

```
ans = logical
   0
```

M==L outputs an error because it is not compatible with matricies.

```
N==0, N==3
```

```
ans = 3×3 logical array
   1   1   0
   1   0   1
   1   1   0
ans = 3×3 logical array
   0   0   1
   0   1   0
   0   0   1
```

There was no error because it is checking a matrix with a marix.

```
M==N, isequal(M,N), ~isequal(M,N)
```

```
ans = 3×3 logical array
   0   0   1
   0   1   0
   0   0   1
ans = logical
   0
ans = logical
   1
```

M==N checks each individual value to see if they are equal and this is represented with a matrix. isequal checks if matrix M and matrix N are equal which they are so it produces 0 (false). ~isequal checks if matrix M and matrix N are not equal which they are so it produces 1(Parttrue).

```
M==N, M~=N
```

```
ans = 3×3 logical array
   0   0   1
   0   1   0
   0   0   1
ans = 3×3 logical array
   1   1   0
   1   0   1
   1   1   0
```

M==N checks each individual value to see if they are equal and this is represented with a matrix with 1s and 0s. M~=N checks each individual value to see if they are not equal and this is represented with a matrix with 1s and 0s.

Part 2. Conditional Statements & Logical Commands

```
N=[0 0 3; 0 3 0; 0 0 3]
```

```
N = 3×3
     0     0     3
     0     3     0
     0     0     3
```

```
if all(N,'all')
    disp('all entries of N are nonzero')
elseif ~any(N,'all')
    disp('all entries N are zero')
elseif any(N) & any(N,2)
    disp('all columns and all rows of N are nonzero')
elseif any(N)
    disp('All columns are nonzero but there are zero rows')
elseif any(N,2)
    disp('all rows are nonzero but there are zero columns')
else
    disp('N has both zero and non-zero columns and rows')
end
```

```
all rows are nonzero but there are zero columns
```

all(N, 'all') checks if all values of N are not zeroes. ~any(N, 'all') checks if all values of N are zeroes. any(N) & any(N,2) checks if each column and row are not zeroes. any(N) checks each each column if there are any with zero rows. any(N,2) checks each row if there are any with zero columns. If none of the if statement worked, then matrix N must have both zeros and non zeros for columns and rows.

```
Q=[1 0 0; 2 3 0; 0 0 0]
```

```
Q = 3×3
     1     0     0
```

```
     2     3     0
     0     0     0
```

```
%(a)
if any(Q,"all")
disp('Q has nonzero entries')
else
disp('All entries of Q are zero')
end
```

```
Q has nonzero entries
```

```
%(b)
if any(Q)
disp('Q has nonzero entries')
else
disp('All entries of Q are zero')
end
```

```
All entries of Q are zero
```

```
%(c)
if Q~=0
disp('Q has nonzero entries')
else
disp('All entries of Q are zero')
end
```

```
All entries of Q are zero
```

```
%(d)
if any(any(Q))
disp('Q has nonzero entries')
else
disp('All entries of Q are zero')
end
```

```
Q has nonzero entries
```

```
%(e)
if ~all(Q==0,"all")
disp('Q has nonzero entries')
else
disp('All entries of Q are zero')
end
```

```
Q has nonzero entries
```

```
%(f)
if ~all(Q==0)
disp('Q has nonzero entries')
else
disp('All entries of Q are zero')
end
```

```
All entries of Q are zero
```

B,C, and F will not prperly check if the matrix has non-zero entries because they chekcing if all the entries are zeroes which would make it false and have it output all entries are zero.

Part 3. "For" Loops and Vectorized Statements

```
format
A=zeros(3,4);
for i=1:3
    for j=1:4
        A(i,j)=i+j-1;
    end
end
A
```

```
A = 3×4
     1     2     3     4
     2     3     4     5
     3     4     5     6
```

For 3 rows and 4 columns, add the number of columns and rows away from base to give it its value.

```
z=(1:4)';
m=size(z,1);
n=5;
%(a)
ZF=zeros(m,n);
for i=1:n
ZF(:,i)=z.^(n-i);
end
ZF
```

```
ZF = 4×5
     1     1     1     1     1
    16     8     4     2     1
    81    27     9     3     1
   256    64    16     4     1
```

```
%(b)
ZV=zeros(m,n);
n=size(ZV,2);
i=n-1:-1:0;
ZV=z.^i
```

```
ZV = 4×5
     1     1     1     1     1
    16     8     4     2     1
    81    27     9     3     1
   256    64    16     4     1
```

For code a, z is created using a column of size 4. M uses z to create a martix size 4 by 1. ZF is created with the zeros function and was created with a size of 4 by 5. ZF constructed with for loops with each row going right to left mutliplying each other by what row number they are on. ZV uses i which already has an array to go through each z to create ZV.

```
format rat
H = hilb(5)
```

```
H =
       1            1/2          1/3          1/4          1/5
       1/2          1/3          1/4          1/5          1/6
       1/3          1/4          1/5          1/6          1/7
       1/4          1/5          1/6          1/7          1/8
       1/5          1/6          1/7          1/8          1/9
```

```
HF=zeros(5)
```

```
HF =
       0            0            0            0            0
       0            0            0            0            0
       0            0            0            0            0
       0            0            0            0            0
       0            0            0            0            0
```

```
for i=1:5
    for j=1:5
        HF(j,i)= (1/(i+j-1));
    end
end
HF
```

```
HF =
       1            1/2          1/3          1/4          1/5
       1/2          1/3          1/4          1/5          1/6
       1/3          1/4          1/5          1/6          1/7
       1/4          1/5          1/6          1/7          1/8
       1/5          1/6          1/7          1/8          1/9
```

```
HV = zeros(5)
```

```
HV =
       0            0            0            0            0
       0            0            0            0            0
       0            0            0            0            0
       0            0            0            0            0
       0            0            0            0            0
```

```
i = 1:5
```

```
i =
       1            2            3            4            5
```

```
j = transpose(1:5)
```

```
j =
       1
       2
       3
       4
       5
```

```
HV=1./(i+j-1)
```

```
HV =
       1            1/2          1/3          1/4          1/5
       1/2          1/3          1/4          1/5          1/6
```

```
    1/3         1/4         1/5         1/6         1/7
    1/4         1/5         1/6         1/7         1/8
    1/5         1/6         1/7         1/8         1/9
```

```
if isequal(H, HF, HV)
    disp('Everything is fine!')
else
    disp('Oh no! There is a bug in my code!')
end
```

```
Everything is fine!
```

```
format
P=ones(5);
for i=2:5
    for j=2:5
        P(j,i)=P(j-1,i)+P(j,i-1);
    end
end
P
```

```
P = 5×5
     1     1     1     1     1
     1     2     3     4     5
     1     3     6    10    15
     1     4    10    20    35
     1     5    15    35    70
```

```
if P==pascal(5)
    disp('the pascal matrix is constructed correctly')
end
```

```
the pascal matrix is constructed correctly
```

```
format
B=magic(4)
```

```
B = 4×4
    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
```

```
S=sum(B);
C=ones(4);
for i=1:4
    C(:,i) = B(:,i) / S(i);
end
C
```

```
C = 4×4
    0.4706    0.0588    0.0882    0.3824
    0.1471    0.3235    0.2941    0.2353
    0.2647    0.2059    0.1765    0.3529
    0.1176    0.4118    0.4412    0.0294
```

```
S1=sum(C)
```

```
S1 = 1×4
     1     1     1     1
```

```
if all(S1 == 1)
    disp('the columns of C are probability vectors')
end
```

```
the columns of C are probability vectors
```

Exercise 3: Create and Call a Function in MATLAB

```
type closetozeroroundoff.m
```

```
function B=closetozeroroundoff(A,p)
A(abs(A)<10^-p)=0;
B=A;
end
```

```
format
p=1
```

```
p = 1
```

```
H=hilb(8);
i=1:8;
G=(-1).^(i+i').*H
```

```
G = 8×8
    1.0000   -0.5000    0.3333   -0.2500    0.2000   -0.1667    0.1429   -0.1250
   -0.5000    0.3333   -0.2500    0.2000   -0.1667    0.1429   -0.1250    0.1111
    0.3333   -0.2500    0.2000   -0.1667    0.1429   -0.1250    0.1111   -0.1000
   -0.2500    0.2000   -0.1667    0.1429   -0.1250    0.1111   -0.1000    0.0909
    0.2000   -0.1667    0.1429   -0.1250    0.1111   -0.1000    0.0909   -0.0833
   -0.1667    0.1429   -0.1250    0.1111   -0.1000    0.0909   -0.0833    0.0769
    0.1429   -0.1250    0.1111   -0.1000    0.0909   -0.0833    0.0769   -0.0714
   -0.1250    0.1111   -0.1000    0.0909   -0.0833    0.0769   -0.0714    0.0667
```

```
X=closetozeroroundoff(G,p)
```

```
X = 8×8
    1.0000   -0.5000    0.3333   -0.2500    0.2000   -0.1667    0.1429   -0.1250
   -0.5000    0.3333   -0.2500    0.2000   -0.1667    0.1429   -0.1250    0.1111
    0.3333   -0.2500    0.2000   -0.1667    0.1429   -0.1250    0.1111   -0.1000
   -0.2500    0.2000   -0.1667    0.1429   -0.1250    0.1111   -0.1000         0
    0.2000   -0.1667    0.1429   -0.1250    0.1111   -0.1000         0         0
   -0.1667    0.1429   -0.1250    0.1111   -0.1000         0         0         0
    0.1429   -0.1250    0.1111   -0.1000         0         0         0         0
   -0.1250    0.1111   -0.1000         0         0         0         0         0
```

The function of closetozeroround takes the absolute value of the each value and if the value is less than 0.1, then it is changed to 0. The function uses G as the matrix to change and uses p as the range of which the numbers close to zero will be rounded to zero.

```
type span
```

```
%{
For an m x n matrix A, the function determines if a vector b is in the
span of the columns of A, that is, if b can be represented as a linear
combination of the columns of A. This is possible if and only if b is
in R^m and the system Ax=b has a solution; moreover, the entries of
a solution x are the weights on the columns of A to produce vector b.
%}
function x=span(A,b)
format
[m,n]=size(A); %outputting the size of A
fprintf('matrix A has %i rows\n',m)
x=[]; %pre-allocating x
if length(b)~=m %checking if the vector b is in R^m
sprintf('the dimensions mismatch: vector b is not in R^%i',m)
return %terminating the code (the dimensions mismatch)
end
fprintf('the dimensions match: vector b is in R^%i\n',m)
%outputing variables:
rankA=rank(A);
aug=[A b];
%testing if b is in the span of the columns of A:
if rankA==m %there is a pivot position in every row
fprintf('the columns of A span the whole R^%i\n',m )
disp('thus the vector b is in the span of the columns of A')
elseif rank(aug)==rankA
fprintf('the columns of A do not span the whole R^%i\n',m)
disp('but the vector b is in the span of the columns of A')
else
disp('the vector b is not in the span of the columns of A')
return %terminating the code (the system Ax=b is inconsistent)
end
%outputting a solution using different approaches depending on the rankA
if rankA==m %there is pivot position in every row
x=A\b; %outputting a solution
else %rank of A is less than the number of rows
R=rref(aug); %outputting the reduced echelon form of matrix aug
R=R(1:rankA,:); %deleting the zero rows on the bottom
x=R(:,1:n)\R(:,end); %outputting a solution
end
%checking if x is not a solution of Ax=b (within the given margin):
if any(closetozeroroundoff(A*x-b,7))
disp('check the code!')
x=[] %re-assigning to x an empty output and displaying it
return %terminating the code (x is not calculated correctly)
end
%making the entries of x that are in the range of 10^-7 from 0 show as 0:
x=closetozeroroundoff(x,12);
%checking if the solution is unique and diplaying it:
if rankA==n %there is a pivot position in every column (unique solution)
disp('the unique vector of weights on columns of A to produce b is')
disp(x) %displaying the solution
else %there are infinitely many solutions
disp('a vector of weights on columns of A to produce b is')
disp(x) %displaying a solution
end
end
```

```
%(a)
A=magic(3), b=rand(3,1)
```

```
A = 3×3
     8     1     6
     3     5     7
```

```
     4     9     2
b = 3×1
    0.8055
    0.5767
    0.1829
```

```
x=span(A,b);
```

```
matrix A has 3 rows
the dimensions match: vector b is in R^3
the columns of A span the whole R^3
thus the vector b is in the span of the columns of A
the unique vector of weights on columns of A to produce b is
    0.0470
   -0.0171
    0.0745
```

```
A*x==b
```

```
ans = 3×1 logical array
   1
   0
   0
```

```
closetozeroroundoff(A*x-b,7)==0
```

```
ans = 3×1 logical array
   1
   1
   1
```

The function A*x==b does not confirm that x is a solution of Ax=b because it checks if each value are one to one exactly and it relies on creating another matrix to make the comparison. The closetozeroroundoff function runs on the matrix directly rather than creating another matrix to check whether each value are 0.

```
%(b)
A=magic(4),b=ones(3,1)
```

```
A = 4×4
    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
b = 3×1
     1
     1
     1
```

```
x=span(A,b);
```

```
matrix A has 4 rows
ans =
'the dimensions mismatch: vector b is not in R^4'
```

```
%(c)
A=[magic(3);ones(1,3)],b=rand(4,1)
```

```
A = 4×3
     8     1     6
     3     5     7
```

```
     4     9     2
     1     1     1
b = 4×1
    0.2399
    0.8865
    0.0287
    0.4899
```

```
x=span(A,b);
```

```
matrix A has 4 rows
the dimensions match: vector b is in R^4
the vector b is not in the span of the columns of A
```

```
%(d)
A=[magic(3);ones(1,3)],b=sum(A,2)
```

```
A = 4×3
     8     1     6
     3     5     7
     4     9     2
     1     1     1
b = 4×1
    15
    15
    15
     3
```

```
x=span(A,b);
```

```
matrix A has 4 rows
the dimensions match: vector b is in R^4
the columns of A do not span the whole R^4
but the vector b is in the span of the columns of A
the unique vector of weights on columns of A to produce b is
     1
     1
     1
```

```
%(e)
A=[magic(3),ones(3,1)],b=ones(3,1)
```

```
A = 3×4
     8     1     6     1
     3     5     7     1
     4     9     2     1
b = 3×1
     1
     1
     1
```

```
x=span(A,b);
```

```
matrix A has 3 rows
the dimensions match: vector b is in R^3
the columns of A span the whole R^3
thus the vector b is in the span of the columns of A
a vector of weights on columns of A to produce b is
    0.0667
    0.0667
    0.0667
         0
```

```
%(f)
A=magic(4),b=ones(4,1)
```

```
A = 4×4
    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
b = 4×1
     1
     1
     1
     1
```

```
x=span(A,b);
```

```
matrix A has 4 rows
the dimensions match: vector b is in R^4
the columns of A do not span the whole R^4
but the vector b is in the span of the columns of A
a vector of weights on columns of A to produce b is
    0.0196
         0
    0.0588
    0.0392
```

Exercise 4: Working With A Function In Live Editor

```
type lines
```

```
function [solution,x] = lines(A,c)
format
solution=[];
x=[];
rankA=rank(A);
if any(A,2)
    disp('each equation of the system represents a line')
else
    disp('the input A is geometrically invalid')
end
aug=[A c];

if rank(aug) == rankA
    disp('the syatem is consistent')

    if rankA == size(A,2)
        disp('the lines have only one point in common')
        solution='point'
        n1=A(1,:);
        n2=A(2,:);
        cos_theta=abs(dot(n1,n2))/(norm(n1)*norm(n2));
        if cos_theta >= 1
            theta=0;
        else
            theta=acosd(cos_theta);
        end
        fprintf('angle between the intersecting lines is %i°\n',theta)
    else
        disp('the lines are coincident')
        solution='line'
        aug=aug(1,:);
    end
```

```
    x=aug(:,1:2)\aug(:,end);
    if closetozeroroundoff(A*x-c,7)== 0
        disp('point x is common to the two lines')
        x
    else
        disp('x is not common to the lines? That cannot be true!')
        x=[]
    end
else
    disp('system is inconsistent and the lines have no points in common')
    N=ones(2,1);
    for i=1:2
        N(i)=norm(A(i,:));
    end
    aug=aug./N;
    n1=aug(1,1:2);
    n2=aug(2,1:2);
    s=sign(dot(n1,n2));
    d=abs(aug(1,end)-s*aug(2,end));
    fprintf('the distance between the parallel lines is %i\n',d)
end
```

```
type closetozeroroundoff
```

```
function B=closetozeroroundoff(A,p)
A(abs(A)<10^-p)=0;
B=A;
end
```

```
%(a)
A=[1 2;0 0],c=[5;-2]
```

```
A = 2×2
     1     2
     0     0
c = 2×1
     5
    -2
```

```
[solution,x] = lines(A,c);
```

```
the input A is geometrically invalid
system is inconsistent and the lines have no points in common
the distance between the parallel lines is NaN
```

```
%(b)
A=[0 2;0 5],c=[5;1]
```

```
A = 2×2
     0     2
     0     5
c = 2×1
     5
     1
```

```
[solution,x] = lines(A,c);
```

```
each equation of the system represents a line
system is inconsistent and the lines have no points in common
the distance between the parallel lines is 2.300000e+00
```

```
%(c)
```

```
A=[0 2;0 5],c=[2;5]
```

```
A = 2×2
     0     2
     0     5
c = 2×1
     2
     5
```

```
[solution,x] = lines(A,c);
```

```
each equation of the system represents a line
the syatem is consistent
the lines are coincident
solution =
'line'
point x is common to the two lines
x = 2×1
     0
     1
```

```
%(d)
A=pascal(2),c=[-1;3]
```

```
A = 2×2
     1     1
     1     2
c = 2×1
    -1
     3
```

```
[solution,x] = lines(A,c);
```

```
each equation of the system represents a line
the syatem is consistent
the lines have only one point in common
solution =
'point'
angle between the intersecting lines is 1.843495e+01°
point x is common to the two lines
x = 2×1
    -5
     4
```

```
%(e)
A=[1 2; 2 4],c=ones(2,1)
```

```
A = 2×2
     1     2
     2     4
c = 2×1
     1
     1
```

```
[solution,x] = lines(A,c);
```

```
each equation of the system represents a line
system is inconsistent and the lines have no points in common
the distance between the parallel lines is 2.236068e-01
```

```
%(f)
A=[1 2; 2 4],c=sum(A,2)
```

```
A = 2×2
     1     2
     2     4
c = 2×1
     3
     6
```

```
[solution,x] = lines(A,c);
```

```
each equation of the system represents a line
the syatem is consistent
the lines are coincident
solution =
'line'
point x is common to the two lines
x = 2×1
         0
    1.5000
```

```
%(g)
A=hilb(2),c=randi(5,2,1)
```

```
A = 2×2
    1.0000    0.5000
    0.5000    0.3333
c = 2×1
     1
     5
```

```
[solution,x] = lines(A,c);
```

```
each equation of the system represents a line
the syatem is consistent
the lines have only one point in common
solution =
'point'
angle between the intersecting lines is 7.125016e+00°
point x is common to the two lines
x = 2×1
  -26.0000
   54.0000
```