Project 0

Sophie Clark

Exercise 1: Working with Matrices in MATLAB

Part 1. Creating matrices

```
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
```

Yes; x and y are of the same size. x has 1 row and 4 columns, while y has 1 column and 4 rows, resulting in each of them having the same number of entries (4).

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

```
ans = 3
```

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

```
ans = 4
```

The first command gives the number of rows in matrix A, while the second command gives 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
```

This line displays the matrix A, then the value contained in the 1st row and 3rd column (7), then all the values in the third column, then all the values in the second row, and lastly the matrix excluding the fourth column.

```
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
```

This line displays the matrix A, and then the values contained in the first two rows and second and fourth columns.

```
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
```

In this line a new matrix, F, is created to have four columns and four rows, the last column of which is designated by the command F(:,4). Next, the values in the first and third rows and second and third columns are designated so that the original values (0) are overwritten.

```
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
```

This line displays the matrices A and F, and then modifies the second and third rows of F to be equivalent to the first and third rows of A.

```
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
```

This line displays F, then modifies F so that the first and second columns are interchanged.

```
F, F(2:4,:), F(end,:)
```

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

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

This line displays F, then displays only the last three rows of F, and finally displays only the final row of 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=F(:,1:3)
```

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

```
b=F(:,end)
```

```
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
```

In the first two lines, matrices are created by adding all the columns of the second matrix in brackets to the columns of the first matrix in the brackets. In the third line, a matrix is created by adding a fourth row to the matrix A and placing the values of x to this row. The fourth line creates a matrix by diagonally adding the matrices; in other words, the values of B are placed in the first two columns and frist three rows while the values of A are placed in the last four columns and last three rows. All other undesignated values become 0.

Part 4: Some special functions for creating matrices

```
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 first entry creates a reduched echelon matrix with 5 columns and 5 rows. The second entry creates a matrix of zeros with 3 rows and 4 columns, while the third entry creates a matrix of zeros with 2 rows and 2 columns. The fourth entry creates a matrix of ones with 3 rows and 2 columns, while the fifth entry creates a matrix of ones with 5 rows and 5 columns.

```
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 first entry creates an empty matrix of 5 rows and 5 columns, and then fills the first entry with the first specified value, and places the following values in each right downward diagonal space. The second and third entries do the same, but the second entry adds an additional empty column to the left and an empty row on the bottom, and the third entry creates two additional empty columns to the right and two empty rows on the top.

```
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 second entry prints the two top diagonal entries in the matrix B, and the third entry prints the entries of the previous entry as a diagonal matrix.

```
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 second entry changes A so that each leading entry in each row has only zeros in the rows below it, while no other entries are altered. The third entry does the reverse, where there is only one nonzero entry in the first row (the leftmost value), two nonzero entries in the second row (the two leftmost values), and three nonzero entries in the third row (the three leftmost values).

```
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 second entry causes all the values in the first column to become zero, and then the leading entry excluding this column has only zeros in the row below it. The third entry does the opposite, shifting the function left and causing there to only be one zero in the leftmost bottom space.

```
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
```

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
```

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 matrix magic is generated by applying values throughout the matrix that cause column, row, and diagonal sums to all be equal. The matrix hilb is created by dividing 1 by the sum of the column and row numbers with 1 subtracted. The matrix pascal is generated so each entry is equivalent to the sum of the value above it and the value to the left of it (with the first row and column being composed of ones).

```
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 space 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, 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
```

MATLAB uses the short format by default, which provides 5 significant figures here. The sym command expresses it using a symbol, which is here the square root symbol. The long format provides 16 significant figures. The rat format produces a fraction/ratio.

Part 7: Operations on Matrices

```
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 matrix x is the first row of x+y (since the first value of matrix y is 0), and each of the following rows is the matrix x where each subsequent value of y has been added to every value in x.

```
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
```

Both A' and the command transpose switch the rows and columns of the matrix.

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
```

x*y is created by matrix multiplication, wherein the values of the first row in x are multiplied by the values in the first column in y and then summed respectively, causing it to be empty besides the value 40 in the first entry. y*x is created in the same manner, but each row of y is multiplied by the values of the first column of x and then summed, causing most entries besides the first row (composed of zeros) to be filled.

```
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
```

P*Q was calculated using standard matrix multiplication (described previously). P.*P and P.^2 are calculated by replacing each value in P with the same value squared.

```
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 matrices with random entries

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

```
ans = 4×4
    0.3755    0.8601    0.5595    0.6390
    0.9737    0.4019    0.9336    0.8876
    0.9723    0.6319    0.7203    0.1987
    0.6437    0.9852    0.4840    0.3954
ans = 3×4
    0.9922    0.9013    0.1084    0.5671
    0.4024    0.9954    0.0361    0.9620
    0.6589    0.6532    0.6181    0.7461
ans = 2×2
    67    26
    53    97
ans = 2×4
     6     7     1     4
     1     6     9     3
ans = 2×4
    13    17    12    18
    19    30    18    37
```

The first matrix is 4x4 with a range of decimals between 0 and 1; the second is 3x4 also with values between 0 and 1; the third is 2x2 with values between 0 and 100; the third is 2x4 with values between 0 and 10; the fourth is 2x4 with values between 10 and 40.

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

```
ans = 3×3
    2.2217    3.9163    4.2430
    3.7796    0.5697    0.2532
    3.0165    4.8928    2.3310
ans = 3×3
   -1.3717   -0.1006   -0.7579
    0.1510    0.0158   -2.8229
   -1.8485   -0.0006   -0.4309
```

```
-1+6*rand(2,4), randi([0 40],2,4)-20
```

```
ans = 2×4
```

```
    1.4464    1.7593    2.3068    3.2051
   -0.3517    1.7053    3.8324    4.2334
ans = 2×4
   -18    -2    12    -7
   -11    19    -2   -18
```

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
```

The second command (M==L) returns an error message, because these arrays have different sizes.

```
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
```

No error messages are received here, because the values in N are only zeros and threes, and these commands act on individual values, not the entire matrix.

```
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
```

The first command determines whether each individual value in the two matrices are equal. The second determines whether the two matrices are equal in all values (which they are not). The last determines whether or not they are not equal (which they are).

```
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
```

The first command determines which individual values are equal, while the second determines which individual values are unequal.

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 of 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')
end
```

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

The first command determines whether or not all entries of N are nonzero; the second determines if all entries of N are zero; the third determines if all of the columns and all of the rows in N are nonzero; the fourth determines if all the columns are nonzero and there are rows of zero; the fifth determines if all the rows are nonzero and there are columns of zero.

```
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 are incorrect for determining if a matrix has non-zero entries. b is incorrect because it accounts for columns, and the last column of Q is comprised of only zeros. c is incorrect because there are zero entries, so not every value of Q is nonzero. f is incorrect because not every entry of Q is nonzero, so the all command is not correctly applied here.

If Q were a vector, c is incorrect, as it will posit that all entries of Q are zero if there is one nonzero and one zero entry.

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
```

A was first created as a matrix of zeros with 3 rows and 4 columns. Then, each value is overwritten to be the sum of its column number and its row number with one subtracted.

```
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
```

In part a, ZF is first created as a 4x5 matrix of zeros. The for loop fills each column with numbers 1-4 to different powers. in the first column, 1-4 is raised to the power of 4; in the second, it's raised to the power of 3; in the third, the power of 2; in the fourth, the power of 1; in the fifth, the power of 0. In part b, ZV is, again, first created as a 4x5 matrix of zeros. n is defined as the size of the row (5), and then i is defined as a single row of numbers 4-0. Then, i is applied as the power of z in each column, producing the same matrix as ZF.

```
format rat
%(a)
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
```

```
%(b)
HF=zeros(5);
for i=1:5
    for j=1:5
        HF(i,j)=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
```

```
%(c)
HV=zeros(5);
h=ones(5,1);
i=(1:1:5)';
j=(1:1:5);
HV(i,j)=h./(i+j-1);
HV
```

```
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(i,j)=(P(i-1,j)+P(i,j-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(1,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);
if S1==ones(4)
    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 worked on G by replacing any values of G whose absolute value is less than 10^-1 with a zero, then producing matrix X.

```
type span
```

```
function x=span(A,b)
format
[m,n]=size(A);
fprintf('matrix A has %i rows\n',m)
x=[];
if length(b)~=m
    sprintf('the dimensions mismatch: vector b is not in R^%i\n',m)
    return
end
fprintf('the dimensions match: vector b is in R^%i\n',m)
%outputting variables
rankA=rank(A);
aug=[A b];
%testing if b is in the span of the columns of A:
if rankA==m
    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
end
% outputting a solution using different approaches depending on the rankA
if rankA==m
    x=A\b;
else
    R=rref(aug);
    R=R(1:rankA,:);
    x=R(:,1:n)\R(:,end);
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=[]
    return
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 displaying it:
if rankA==n
    disp('the unique vector of weights on columns of A to produce b is')
    disp(x)
else
    disp('a vector of weights on columns of A to produce b is')
    disp(x)
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
    0.7409
    0.5068
    0.1999
```

```
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
    0.0486
   -0.0129
    0.0608
```

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

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

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

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

The first command A*x==b, in general, does not confirm that x is a solution of Ax=b. This is because the function closetozeroroundoff must be used to compare matrices that may have extremely miniscule differences (accounting for the use of 10^-7 in the function above).

```
%(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
    0.4272
    0.1687
    0.7517
    0.3684
```

```
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)==1
    disp('each equation of the system represents a line')
else
    disp('the input A is geometrically invalid')
```

```
    return
end
aug=[A c];
if rankA==rank(aug)
    disp('the system 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('the 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('the 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
    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
```

```
%(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
the 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;1\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 system 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 system is consistent
the lines have only one point in common
solution =
'point'
the 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
the 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 system 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
     5
     1
```

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

```
each equation of the system represents a line
the system is consistent
the lines have only one point in common
solution =
'point'
the angle between the intersecting lines is 7.125016e+00°
point x is common to the two lines
x = 2×1
    14
```

-18