Project 0

Exercise 1

Part 1: Creating Matrices

```
A=[0 3 6 9; 1 4 7 10; 2 5 8 11]
```

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

```
B=[2 -2; 3 -3;4 -4]
```

```
B = 3×2
     2    -2
     3    -3
     4    -4
```

```
X=[2;4;6]
```

```
X = 3×1
     2
     4
     6
```

```
x=[2 3 4 5]
```

```
x = 1×4
     2     3     4     5
```

```
y=[1;3;5;7]
```

```
y = 4×1
     1
     3
     5
     7
```

```
size(A)
```

```
ans = 1×2
     3     4
```

```
size(B)
```

```
ans = 1×2
     3     2
```

```
size(X)
```

```
ans = 1×2
     3     1
```

```
%The matrices x and y are not the same.
%This is because the dimension of x is 1x4 while y is 4x1. This makes matrix x larger.
size(x), size(y)
```

```
ans = 1×2
     1     4
ans = 1×2
     4     1
```

```
%The command (X, DIM) returns the dimension length in a row vector.
size(A,1), size(A,2)
```

```
ans = 3
ans = 4
```

Part 2: Accessing particular matrix entries and changing entries

```
%Prints matrix A
A
```

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

```
%Prints the value of row 1 and col. 3 of matrix A
A(1,3)
```

```
ans = 6
```

```
%Prints col. 3 of matrix A
A(:,3)
```

```
ans = 3×1
     6
     7
     8
```

```
%Prints row 2 of matrix A
A(2,:)
```

```
ans = 1×4
     1     4     7    10
```

```
%Prints matrix without the last col. (the result, not the augmented matrix)
A(:, 1:3)
```

```
ans = 3×3
     0     3     6
     1     4     7
     2     5     8
```

```
%Prints the first 2 values of column 2 and 4
A, A([1 2], [2 4])
```

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

```
%Adds a fourth column in the matrix F.
```

```
%The new numbers are indicated in [].
F(:,4) = [-1 1 -4 3]
```

```
F = 4×4
     1    -1    -3    -1
     3     9     6     1
     5    11     8    -4
     7     3     0     3
```

```
%Changes row 1, col. 2 and 3 to 1 and -3 respectively,
%Changes row 3, col. 2 and 3 to 2 and -5 respectively.
F([1 3], [2 3]) = [1 -3; 2 -5]
```

```
F = 4×4
     1     1    -3    -1
     3     9     6     1
     5     2    -5    -4
     7     3     0     3
```

```
%Changes the rows 2 and 3 matrix F to row 1 and 3 of matrix A
A, F, F([2 3],:) = A([1 3],:)
```

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

```
%Interchanges the position of col. 1 and 2.
F, F(:,[1 2]) = F(:,[2 1])
```

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

```
%Removes col. 1 of matrix F.
F, F(:, 2:4)
```

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

```
      2     8    11
      7     0     3
```

(I) Run the commands that will output y, F, and a new matrix obtained from F by assigning to the first column of F the vector y – the last output will be your new matrix F.

```
y, F, F(:, 1) = (y)
```

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

(II) Run the commands that will output F from part (I) and create a new matrix obtained from F by switching rows 2 and 4 – it will be your new matrix F.

```
F, F(:,[2 4]) = F(:,[4 2])
```

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

(III) Run the commands that will output the matrix F from part (II) and create a new matrix F1 formed by the first two rows of F.

```
F, F1 = F(:, 1:2)
```

```
F = 4×4
     1    -1    -3     1
     3     9     6     0
     5    11     8     2
     7     3     0     7
F1 = 4×2
     1    -1
     3     9
     5    11
     7     3
```

Part 3: Pasting blocks together

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

```
A = 3×4
```

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

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

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

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

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

Part 4: Special functions for creating matrices

```
%Creates a 5x5 identity matrix.
eye(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
```

```
%Create a 3x4 matrix of zeros
zeros(3,4)
```

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

```
     0     0     0     0
```

```
%Creates a 2x2 matrix of zeros.
zeros(2)
```

```
ans = 2×2
     0     0
     0     0
```

```
%Creates a 3x2 array of ones.
ones(3,2)
```

```
ans = 3×2
     1     1
     1     1
     1     1
```

```
%Creates a 5x5 array of ones
ones(5)
```

```
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
```

```
%Creates a diagonal matrix with elements in the square bracket.
diag([1 2 5 6 7])
```

```
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
```

```
%Returns a diagonal square matrix. Leaves the first row empty,
% and starts to  fill the second row.
diag([1 2 5 6 7],1)
```

```
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
```

```
%Returns a diagonal square matrix. However, the last two rows are left empty.
diag([1 2 5 6 7],-2)
```

```
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
```

```
%Gets the elements onto the main diagonal.
B, diag(B)
```

```
B = 3×2
     2    -2
     3    -3
     4    -4
ans = 2×1
     2
    -3
```

```
%Completes the square diagonal with 0s.
diag(diag(B))
```

```
ans = 2×2
     2     0
     0    -3
```

```
%Returns the upper triagular portion of matrix A.
triu(A)
```

```
ans = 3×4
     0     3     6     9
     0     4     7    10
     0     0     8    11
```

```
%Returns the lower triagular portion of matrix A.
tril(A)
```

```
ans = 3×4
     0     0     0     0
     1     4     0     0
     2     5     8     0
```

```
%Extracts the upper triangular portion of matrix A
%only the elements above and below the main diagonal, respectively.
F, triu(F), triu(F,1), triu(F,-1)
```

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

```
%Extracts from the lower triangular portion of matrix A
%only the elements below and above the main diagonal, respectively.
```

```
F, tril(F), tril(F,-1),tril(F,1)
```

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

```
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
```

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

```
V1=1:7
```

```
V1 = 1×7
     1     2     3     4     5     6     7
```

```
V2=2:0.5:6.5
```

```
V2 = 1×10
    2.0000    2.5000    3.0000    3.5000    4.0000    4.5000    5.0000    5.5000 ⋯
```

```
V3=3:-1:-5
```

```
V3 = 1×9
     3     2     1     0    -1    -2    -3    -4    -5
```

```
V4=-5:1
```

```
V4 = 1×7
    -5    -4    -3    -2    -1     0     1
```

```
V5=10:-3:-2
```

```
V5 = 1×5
    10     7     4     1    -2
```

```
V6=5:-0.5:2
```

```
V6 = 1×7
    5.0000    4.5000    4.0000    3.5000    3.0000    2.5000    2.0000
```

```
V7=0:0.4:4
```

```
V7 = 1×11
         0    0.4000    0.8000    1.2000    1.6000    2.0000    2.4000    2.8000 ⋯
```

Part 6: The format commands

```
R=2.46721652
```

```
R = 2.4672
```

```
%Sets the output format to the long fixed-decimal format,
%in scientific notion, and rounds up to the 5th decimal place of R
format long, R, round(R,5)
```

```
R =
   2.467216520000000
ans =
   2.467220000000000
```

```
%Transforms the value of R into a fraction.
format rat, R
```

```
R =
    5983/2425
```

```
%Shortens the value of R by rounding it to the 3rd decimal place.
format short, R, round(R,3)
```

```
R = 2.4672
ans = 2.4670
```

Part 7: Operations on Matrices

```
A, A+A
```

```
A = 3×4
     0     3     6     9
     1     4     7    10
     2     5     8    11
ans = 3×4
     0     6    12    18
     2     8    14    20
     4    10    16    22
```

```
A, 3*A
```

```
A = 3×4
     0     3     6     9
     1     4     7    10
     2     5     8    11
ans = 3×4
     0     9    18    27
     3    12    21    30
     6    15    24    33
```

```
F, magic(4), U=F+magic(4)
```

```
F = 4×4
     1    -1    -3     1
     3     9     6     0
     5    11     8     2
     7     3     0     7
ans = 4×4
    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
U = 4×4
    17     1     0    14
     8    20    16     8
    14    18    14    14
    11    17    15     8
```

```
%The matrix 'x+y' is created by adding each col. of x to each row of y,
%e.g. first col. of the new matrix is 2 + vector y.
x, y, x+y
```

```
x = 1×4
     2     3     4     5
y = 4×1
```

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

```
%Reverses the col. to rows and rows to col. of the matrix.
A,A',transpose(A)
```

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

Part 8: Matrix multiplication

```
P=[1 2 3 4; 1 1 1 1]
```

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

```
Q=transpose(P)
```

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

```
%The dot product of P and Q
P*Q
```

```
ans = 2×2
    30    10
    10     4
```

```
%The multiplication of matrix P by itself, entry-by-entry.
P.*P
```

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

```
%The multiplication of matrix P by itself, entry-by-entry.
P.^2
```

```
ans = 2×4
```

```
     1     4     9    16
     1     1     1     1
```

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

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

```
%The previous two commands demostrates the dot product of matrix G and itself.
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
```

```
%Checks if the two statements are equal. If it is equal, it returns the value of 1.
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

```
%Generates a 4x4 matrix of uniformly distributed random number values.
rand(4)
```

```
ans = 4×4
    0.5497    0.7537    0.0540    0.1299
    0.9172    0.3804    0.5308    0.5688
    0.2858    0.5678    0.7792    0.4694
    0.7572    0.0759    0.9340    0.0119
```

```
%Generates a 3x4 matrix of uniformly distributed random number values.
rand(3,4)
```

```
ans = 3×4
    0.3371    0.3112    0.6020    0.6892
    0.1622    0.5285    0.2630    0.7482
    0.7943    0.1656    0.6541    0.4505
```

```
%Generates a 2x2 matrix of uniformly distributed random numbers values under 100.
randi(100,2)
```

```
ans = 2×2
     9    92
    23    16
```

```
%Generates a 2x4 matrix of uniformly distributed random numbers caleus under 10.
randi(10,2,4)
```

```
ans = 2×4
     9    10     5    10
     6     1     2     1
```

```
%Generates a 2x4 column vector of uniformly distributed random integers
%from the interval 10 - 40.
randi([10 40],2,4)
```

```
ans = 2×4
    34    36    22    34
    35    12    18    23
```

```
5*rand(3)
```

```
ans = 3×3
    4.5532    0.7277    2.8985
    0.9092    0.6803    2.7493
    1.3190    4.3465    0.7248
```

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

```
ans = 3×3
    1.2652   -0.4338   -1.8004
    0.1103   -0.9910   -2.3834
   -1.2452   -2.6202   -2.0805
```

```
r = 2 + (8)*rand(3,2)
```

```
r = 3×2
    3.9196    9.2217
    5.3381    9.5583
    2.3972    5.9269
```

```
randi([40,70],3,2)
```

```
ans = 3×2
    55    51
    50    43
    67    64
```

Exercise 2: Programming Techniques

Part 1: Logical Operations

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

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

```
%Tests if a relational comparison is a logical array indicating the locations
%where the relation is true, M equals to N and M does not equal to N, respectively.
%The output returns logical '1' (true) if A and B are equal;
%it returns logical '0' if (false)
M==N, M~=N
```

```
ans = 2×2 logical array
   0   1
   1   1
ans = 2×2 logical array
   1   0
   0   0
```

```
%This tests whether M is equal or has any difference to N.
%The output returns logical '1' (true) if A and B are equal;
%it returns logical '0' if (false).
isequal(M,N), ~isequal(M,N)
```

```
ans = logical

   0
ans = logical

   1
```

Part 2: Conditional Statements

```
x=[2 3 4 5]
```

```
x = 1×4
     2     3     4     5
```

```
y=[1;3;5;7]
```

```
y = 4×1
     1
     3
     5
     7
```

```
y'
```

```
ans = 1×4
     1     3     5     7
```

```
size(x)
```

```
ans = 1×2
     1     4
```

```
size(y)
```

```
ans = 1×2
     4     1
```

```
size(y')
```

```
ans = 1×2
     1     4
```

```
if isequal(size(x),size(y))
disp('matrices x and y are of the same size')
else
disp('the sizes of x and y are different')
end
```

```
the sizes of x and y are different
```

```
x, transpose(y)
```

```
x = 1×4
     2     3     4     5
ans = 1×4
     1     3     5     7
```

```
if isequal(x, transpose(y))
disp("matrices x and transpose(y) are equal")
elseif isequal(size(x),size(transpose(y)))
disp("matrices x and transpose(y) are of the same size")
else
disp("Matrices x and transpose(y) are neither equal nor they have the same size")
end
```

```
matrices x and transpose(y) are of the same size
```

```
% #1
if ~isequal(M, N)
disp('M and N are different')
else
disp('M and N are the same')
end
```

```
M and N are different
```

```
% #2
if M~=N
disp('M and N are different')
else
disp('M and N are the same')
end
```

```
M and N are the same
```

```
% #3
if M==N
disp('M and N are the same')
else
disp('M and N are different')
end
```

```
M and N are different
```

```
%The first output is wrong. In the if statment, it is testing if matrices M and N are equal,
%but the print statement saids "they are different."
%The author of the statment must of interchanged the correct print output.
```

Part 3. “For Loops” and Vectorized Statements

```
L=zeros(5);
for i=1:5
for j=1:5
L(i,j)=i+j;
```

```
end
end
L
```

```
L = 5×5
     2     3     4     5     6
     3     4     5     6     7
     4     5     6     7     8
     5     6     7     8     9
     6     7     8     9    10
```

```
%First, a 5x5 matrix of zeros is generated.
%Then, 'i' and 'j' vectors are generated, containing the values from 1 - 5.
%Adds vectors 'i' and 'j' together to create a new 'L' matrix.
```

```
z=transpose(1:5);
ZF=zeros(5,3);
n=size(ZF,2);
for i=1:n
ZF(:,i)=z.^i;
end
ZF
```

```
ZF = 5×3
     1     1     1
     2     4     8
     3     9    27
     4    16    64
     5    25   125
```

```
z=transpose(1:5);
ZV=zeros(5,3);
n=size(ZF,2);
i=1:n;
ZV(:,i)=z.^i
```

```
ZV = 5×3
     1     1     1
     2     4     8
     3     9    27
     4    16    64
     5    25   125
```

```
%The output remains the same.
%But, the matrix ZF uses a for loop statement to generate the matrix,
%while matrix ZV uses a vectorized statement.
```

```
%Creates a 5x5 Hilbert matrix
H=hilb(5)
```

```
H = 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
```

```
%Creates a 5x5 Hilbert matrix using a double for-loop statement.
s = 5;
HF = zeros(s);
for c = 1:s
for r = 1:s
HF(r,c) = 1/(r+c-1);
end
end
HF
```

```
HF = 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
```

```
%Creates a 5x5 Hilbert matrix using a vectorized statement.
HV=zeros(5);
i = 1:5;
j = transpose(1:5);
HV = 1./(i+j-1)
```

```
HV = 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
```

Exercise 3: Create and Call a Function in MATLAB

```
format compact
p=1
```

```
p = 1
```

```
H=hilb(8)
```

```
H = 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
```

```
closetozeroroundoff(H,p)
```

```
ans = 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
```

```
type closetozeroroundoff
```

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

```
type produc
```

```
%The function calculates the product of two matrices A and B, when it is
%defined. Each column of the matrix AB is calculated as a product of the
%matrix A and the corresponding column of the matrix B. The output AB is
%compared with the output of the MATLAB built-in function A*B. Then, we
%demonstrate that the property of invertible matrices holds: the inverse
%of the product of two matrices inv(A*B) is equal to the product of the
%inverses but in the reverse order: inv(B)*inv(A.
function [AB,InvAB,InvBInvA] = produc(A,B,p)
%calculating the sizes:
[m,n]=size(A);
[k,q]=size(B);
%making assignments:
AB=[];
InvAB=[];
InvBInvA=[];
%checking if the matrices inner dimensions agree for multiplication
if isequal(n,k)
disp('the matrices dimensions agree for matrix multiplication')
%calculating the product of A and B:
i=1:q;
AB(:,i)=A*B(:,i);
fprintf('the product of two matrices A and B is\n')
AB
%verifying if the code is correct by running matlab function A*B
C=zeros(m,q);
fprintf('the output for matlab function A*B is\n')
C=A*B;
disp(C)
if isequal(AB,C)
fprintf('the output AB is the same as the output for A*B\n')
else
disp('check the code!')
return
end
%The next task applies to square matrices A and B of the same size
%which are also invertible. We demonstrate that the following
%property holds: inv(A*B)=inv(B)*inv(A)
if m==n && k==q && rank(A)==m && rank(B)==k
disp('A and B are invertible matrices of the same size')
fprintf('the inverse of A*B is\n')
InvAB=inv(AB)
fprintf('product of inverses in reverse order inv(B)*inv(A) is\n')
InvBInvA=inv(B)*inv(A)
if closetozeroroundoff(InvAB-InvBInvA,p)==0
fprintf('inv(A*B)=inv(B)*inv(A) holds for the given A and B\n')
fprintf('and within the given precision 10^(-%i)\n',p)
else
fprintf('inv(A*B)=inv(B)*inv(A) does not hold for the given\n')
fprintf('A and B within the given precision 10^(-%i)\n',p)
end
end
```

```
else
fprintf('the matrices dimensions disagree for matrix multiplication\n')
end
end
```

```
p=7;
%(a)
A=magic(3), B=hilb(3)
```

```
A = 3×3
     8     1     6
     3     5     7
     4     9     2
B = 3×3
    1.0000    0.5000    0.3333
    0.5000    0.3333    0.2500
    0.3333    0.2500    0.2000
```

```
[AB,InvAB,InvBInvA]=produc(A,B,p);
```

```
the matrices dimensions agree for matrix multiplication
the product of two matrices A and B is
AB = 3×3
   10.5000    5.8333    4.1167
    7.8333    4.9167    3.6500
    9.1667    5.5000    3.9833
the output for matlab function A*B is
   10.5000    5.8333    4.1167
    7.8333    4.9167    3.6500
    9.1667    5.5000    3.9833
the output AB is the same as the output for A*B
A and B are invertible matrices of the same size
the inverse of A*B is
InvAB = 3×3
    2.9417    3.5667   -6.3083
  -13.5333  -24.5333   36.4667
   11.9167   25.6667  -35.5833
product of inverses in reverse order inv(B)*inv(A) is
InvBInvA = 3×3
    2.9417    3.5667   -6.3083
  -13.5333  -24.5333   36.4667
   11.9167   25.6667  -35.5833
inv(A*B)=inv(B)*inv(A) holds for the given A and B
and within the given precision 10^(-7)
```

```
%The decimals to the right run for really long,
%therefore this function rounds the decimal to 0.
%This helps the call functions run faster and helps reduce mistakes.
InvAB==InvBInvA
```

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

```
%(b)
A=magic(4), B=ones(4)
```

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

```
     4    14    15     1
B = 4×4
     1     1     1     1
     1     1     1     1
     1     1     1     1
     1     1     1     1
```

```
[AB,InvAB,InvBInvA]=produc(A,B,p);
```

```
the matrices dimensions agree for matrix multiplication
the product of two matrices A and B is
AB = 4×4
    34    34    34    34
    34    34    34    34
    34    34    34    34
    34    34    34    34
the output for matlab function A*B is
    34    34    34    34
    34    34    34    34
    34    34    34    34
    34    34    34    34
the output AB is the same as the output for A*B
```

```
%(c)
A=magic(5), B=ones(5,4)
```

```
A = 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
B = 5×4
     1     1     1     1
     1     1     1     1
     1     1     1     1
     1     1     1     1
     1     1     1     1
```

```
[AB,InvAB,InvBInvA]=produc(A,B,p);
```

```
the matrices dimensions agree for matrix multiplication
the product of two matrices A and B is
AB = 5×4
    65    65    65    65
    65    65    65    65
    65    65    65    65
    65    65    65    65
    65    65    65    65
the output for matlab function A*B is
    65    65    65    65
    65    65    65    65
    65    65    65    65
    65    65    65    65
    65    65    65    65
the output AB is the same as the output for A*B
```

```
%(d)
A=magic(5), B=ones(4,5)
```

```
A = 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
B = 4×5
     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1
```

```
[AB,InvAB,InvBInvA]=produc(A,B,p);
```

```
the matrices dimensions disagree for matrix multiplication
```

```
%(e)
A=magic(5), B=hilb(5)
```

```
A = 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
B = 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
```

```
[AB,InvAB,InvBInvA]=produc(A,B,p);
```

```
the matrices dimensions agree for matrix multiplication
the product of two matrices A and B is
AB = 5×5
   34.3333   20.8500   15.3429   12.2345   10.2095
   34.5333   20.3833   14.9357   11.9167    9.9611
   20.7333   14.9167   11.9095    9.9738    8.6016
   28.1833   18.4500   14.0619   11.4417    9.6726
   30.6333   19.6500   14.7857   11.9274   10.0214
the output for matlab function A*B is
   34.3333   20.8500   15.3429   12.2345   10.2095
   34.5333   20.3833   14.9357   11.9167    9.9611
   20.7333   14.9167   11.9095    9.9738    8.6016
   28.1833   18.4500   14.0619   11.4417    9.6726
   30.6333   19.6500   14.7857   11.9274   10.0214
the output AB is the same as the output for A*B
A and B are invertible matrices of the same size
the inverse of A*B is
InvAB = 5×5
   -0.0050    0.0028    0.0015   -0.0090    0.0097
    0.0871   -0.0491   -0.0304    0.1738   -0.1816
   -0.3616    0.2055    0.1383   -0.7657    0.7844
    0.5320   -0.3049   -0.2170    1.1735   -1.1853
   -0.2553    0.1474    0.1092   -0.5803    0.5799
product of inverses in reverse order inv(B)*inv(A) is
InvBInvA = 5×5
   -0.0050    0.0028    0.0015   -0.0090    0.0097
    0.0871   -0.0491   -0.0304    0.1738   -0.1816
   -0.3616    0.2055    0.1383   -0.7657    0.7844
    0.5320   -0.3049   -0.2170    1.1735   -1.1853
   -0.2553    0.1474    0.1092   -0.5803    0.5799
inv(A*B)=inv(B)*inv(A) does not hold for the given
A and B within the given precision 10^(-7)
```

```
p = 6;
A=magic(5), B=hilb(5)
```

```
A = 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
B = 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
```

```
[AB,InvAB,InvBInvA]=produc(A,B,p);
```

```
the matrices dimensions agree for matrix multiplication
the product of two matrices A and B is
AB = 5×5
   34.3333   20.8500   15.3429   12.2345   10.2095
   34.5333   20.3833   14.9357   11.9167    9.9611
   20.7333   14.9167   11.9095    9.9738    8.6016
   28.1833   18.4500   14.0619   11.4417    9.6726
   30.6333   19.6500   14.7857   11.9274   10.0214
the output for matlab function A*B is
   34.3333   20.8500   15.3429   12.2345   10.2095
   34.5333   20.3833   14.9357   11.9167    9.9611
   20.7333   14.9167   11.9095    9.9738    8.6016
   28.1833   18.4500   14.0619   11.4417    9.6726
   30.6333   19.6500   14.7857   11.9274   10.0214
the output AB is the same as the output for A*B
A and B are invertible matrices of the same size
the inverse of A*B is
InvAB = 5×5
   -0.0050    0.0028    0.0015   -0.0090    0.0097
    0.0871   -0.0491   -0.0304    0.1738   -0.1816
   -0.3616    0.2055    0.1383   -0.7657    0.7844
    0.5320   -0.3049   -0.2170    1.1735   -1.1853
   -0.2553    0.1474    0.1092   -0.5803    0.5799
product of inverses in reverse order inv(B)*inv(A) is
InvBInvA = 5×5
   -0.0050    0.0028    0.0015   -0.0090    0.0097
    0.0871   -0.0491   -0.0304    0.1738   -0.1816
   -0.3616    0.2055    0.1383   -0.7657    0.7844
    0.5320   -0.3049   -0.2170    1.1735   -1.1853
   -0.2553    0.1474    0.1092   -0.5803    0.5799
inv(A*B)=inv(B)*inv(A) holds for the given A and B
and within the given precision 10^(-6)
```

```
%The difference between 'p = 6' and part 'e' is rounding off the
%decimal number. The e script rounds the decimals at the 7th place to 0,
%while 'p = 6' rounds the 6th decimal place to 0.
% Since the product of inverses in reverse order only extends to the 4th decimal place,
%the 'p' parameter that gives the 6th/7th decimal place precision, dosen't matter.
```

Exercise 4: Working with a Function in Live Editor

```
type dotangle
```

```
function [d1,d2,d] = dotangle(u,v)
m = length(u);
n = length(v);
p = 5;
if isequal(m,n)
    fprintf('both vectors have %i entries\n',n)
else
    disp('the dot product is not defined')
    d1=[];
    d2=[];
    d=[];
    return;
end

d1=transpose(u)*v;
d2=0;
for i=1:n
        d2=d2+u(i)*v(i);
end
d=dot(u,v);
if isequal(d1,d), isequal(d2,d)
    fprintf('the code is correct\n')
else
    disp('check the code!')
    return;
end

theta = acosd(d/(norm(u)*norm(v)));

if closetozeroroundoff(theta, p) == closetozeroroundoff(90, p)
    fprintf('the angle between the vectors is 90 degrees\n')
elseif closetozeroroundoff(theta, p) == closetozeroroundoff(0, p)
    fprintf('the angle between the vectors is zero\n')
elseif closetozeroroundoff(theta, p) == closetozeroroundoff(180, p)
    fprintf('the angle between the vectors is 180 degrees\n')
elseif theta < 90
    fprintf('angle between vectors is acute and its value theta= %p', theta);
elseif theta > 90
    fprintf('angle between vectors is obtuse and its value theta= %p', theta);

end
end
```

```
%(a)
u=randi(10,5,1),v=randi(10,4,1)
```

```
u = 5×1
     4
     3
     5
     1
     2
v = 4×1
    10
    10
     6
     1
```

```
[d1,d2,d]=dotangle(u,v);
```

```
the dot product is not defined
```

```
%(b)
u=randi(15,5,1), v=randi(15,5,1)
```

```
u = 5×1
     4
     6
    13
     1
     1
v = 5×1
     3
    10
    11
    10
     7
```

```
[d1,d2,d]=dotangle(u,v);
```

```
both vectors have 5 entries
ans = logical

   1
the code is correct
angle between vectors is acute and its value theta=
```

```
%(c)
u=u, v=-v
```

```
u = 5×1
     4
     6
    13
     1
     1
v = 5×1
    -3
   -10
   -11
   -10
    -7
```

```
[d1,d2,d]=dotangle(u,v);
```

```
both vectors have 5 entries
ans = logical

   1
the code is correct
angle between vectors is obtuse and its value theta=
```

```
%(d)
u=u, v=2*u
```

```
u = 5×1
     4
     6
    13
     1
     1
v = 5×1
```

```
     8
    12
    26
     2
     2
```

```
[d1,d2,d]=dotangle(u,v);
```

```
both vectors have 5 entries
ans = logical

   1
the code is correct
the angle between the vectors is zero
```

```
%(e)
u=u, v=-3*u
```

```
u = 5×1
     4
     6
    13
     1
     1
v = 5×1
   -12
   -18
   -39
    -3
    -3
```

```
[d1,d2,d]=dotangle(u,v);
```

```
both vectors have 5 entries
ans = logical

   1
the code is correct
the angle between the vectors is 180 degrees
```

```
%(f)
u=[1;3],v=[-3;1]
```

```
u = 2×1
     1
     3
v = 2×1
    -3
     1
```

```
[d1,d2,d]=dotangle(u,v);
```

```
both vectors have 2 entries
ans = logical

   1
the code is correct
the angle between the vectors is 90 degrees
```