Laborator 1

Cuprins

Problema 1

Să se inverseze linia 1 cu linia 3 în matricea $a = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$.

```
a = [1 2;
     3 4;
     5 6;];
aux = a(1,:);
a(1,:) = a(3,:);
a(3,:) = aux;
a
```

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

Problema 2

Să se inverseze coloana 2 cu coloana 3 în matricea $b = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$.

```
b = [1 2 3;
     4 5 6;];
aux = b(:,2);
b(:,2) = b(:,3);
b(:,3) = aux;
b
```

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

Problema 3

Se dă vectorul $v = [1 \quad 2 \quad 3 \quad 5 \quad 7 \quad 11 \quad 13]$. Să se extragă elementele $3\ 5\ 7\ 11$.

```
v= [1 2 3 5 7 11 13];
v_p(1:4)=v(3:6);
v_p
```

```
v_p = 1×4
     3     5     7    11
```

Problema 4

Se dă matricea $a = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$. Să se extragă submatricea $\begin{bmatrix} 4 & 5 \\ 7 & 8 \end{bmatrix}$.

```
a=[1 2 3;
    4 5 6;
    7 8 9;];
aux(1,1)=a(2,1);
aux(1,2)=a(2,2);
aux(2,1)=a(3,1);
aux(2,2)=a(3,2);
aux
```

```
aux = 2×2
     4     5
     7     8
```

Problema 5

Utilizarea instrucțiunilor *zeros*, *ones*, *eye*. Construirea de exemple proprii.

Instrucțiunea *zeros* creează o matrice de zerouri (O_n).

```
O_n = zeros
```

```
O_n = 0
```

```
O_n = zeros(7)
```

```
O_n = 7×7
     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     0     0     0
     0     0     0     0     0     0     0
     0     0     0     0     0     0     0
     0     0     0     0     0     0     0
```

```
O_n = zeros(5,7)
```

```
O_n = 5×7
     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     0     0     0
     0     0     0     0     0     0     0
```

Instrucțiunea *ones* creează o matrice cu toate elementele 1.

```
x = ones
```

```
x = 1
```

```
x = ones(7)
```

```
x = 7×7
     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     1     1     1
     1     1     1     1     1     1     1
     1     1     1     1     1     1     1
     1     1     1     1     1     1     1
```

```
x = ones(6,3)
```

```
x = 6×3
     1     1     1
     1     1     1
     1     1     1
     1     1     1
     1     1     1
     1     1     1
```

Instrucțiunea *eye* creează o matrice cu 1 pe diagonala principală și 0 în rest (I_n).

```
I_n = eye
```

```
I_n = 1
```

```
I_n = eye(5)
```

```
I_n = 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
```

```
I_n = eye(6,4)
```

```
I_n = 6×4
     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1
     0     0     0     0
     0     0     0     0
```

```
I_n = eye(3,7)
```

```
I_n = 3×7
     1     0     0     0     0     0     0
     0     1     0     0     0     0     0
     0     0     1     0     0     0     0
```

Problema 6

Se dau vectorii $u = [1 \quad 2 \quad 3]$ și $v = [4 \quad 5 \quad 6]$. Să se efectueze diverse operații de comparație între acești vectori (egalitate, inegalitate, $<, >, \leq, \geq$).

```
u = [1 2 3];
v = [4 5 6];
u < v
```

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

```
u > v
```

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

```
u <= v
```

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

```
u >= v
```

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

Problema 7

Pentru vectorii de la Problema 6, să se concateneze $2u$ și $-3v$.

```
u = [1 2 3];
v = [4 5 6];
u(1:3) = 2.*(u(1:3));
v(1:3) = -3.*(v(1:3));
fprintf('u(x) = %2.5f, u(y) = %2.5f, u(z) = %2.5f \n', u(1), u(2), u(3))
```

```
u(x) = 2.00000, u(y) = 4.00000, u(z) = 6.00000
```

```
fprintf('v(x) = %2.5f, v(y) = %2.5f, v(z) = %2.5f \n', v(1), v(2), u(3))
```

```
v(x) = -12.00000, v(y) = -15.00000, v(z) = 6.00000
```

Problema 8

Să se rezolve un sistem de ecuații liniare la alegere.

$$\begin{cases} 3x + 2y - z = 1 \\ 6x - 4y + 2z = 3 \\ 5x + y - 2z = -6 \end{cases}$$

Metoda 1

```
A = [3 2 -1;
     6 -4 2;
     5 1 -2;];
B = [1;
     3;
    -6;];
linsolve(A,B)
```

```
ans = 3×1
    0.4167
    2.5278
    5.3056
```

Metoda 2

Verificăm compatibilitatea sistemului:

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

if rank(A)== 3 && rank(A)==rank(A_bar)
    fprintf('A și A barat au rangul maxim egal.')
end
```

```
A și A barat au rangul maxim egal.
```

$\mathrm{Rang}(A) = \mathrm{Rang}(\overline{A}) = 3 \Rightarrow$ sistem compatibil determinat. Aplicăm Regula lui Cramer:

```
A_x = [1 2 -1;
       3 -4 2;
      -6 1 -2;];
A_y = [3 1 -1;
       6 3 2;
       5 -6 -2;];
A_z = [3 2 1;
       6 -4 3;
       5 1 -6;];
x=det(A_x)/det(A);
y=det(A_y)/det(A);
z=det(A_z)/det(A);
fprintf('x = %2.5f \n', x)
```

```
x = 0.41667
```

```
fprintf('y = %2.5f \n', y)
```

```
y = 2.52778
```

```
fprintf('z = %2.5f \n', z)
```

```
z = 5.30556
```

Problema 9

Să se scrie o funcție MATLAB care să realizeze următoarele operații între matrici: A+B, A-B, A*B, A*A*A.

```
A = [1 4 7;
     5 3 -5;
     -5 7 11;];
B = [5 4 -5;
     7 -9 10;
     2 12 4;];
matrixAddition(A,B)
```

```
ans = 3×3
     6     8     2
    12    -6     5
    -3    19    15
```

```
matrixSubtraction(A,B)
```

```
ans = 3×3
    -4     0    12
    -2    12   -15
    -7    -5     7
```

```
matrixCrossProduct(A,B)
```

```
ans = 3×3
    47    52    63
    36   -67   -15
    46    49   139
```

```
matrixCube(A)
```

```
ans = 3×3
    -9   587   281
   190   -83   -40
   110   491    -4
```

```
matrixCube(B)
```

```
ans = 3×3
        -327         796        -995
        1393       -3113        1990
         398        2388        -526
```

Problema 10

Să se scrie o funcție MATLAB care să realizeze pe componente următoarele operații A.*B, A./B, A.^2.

```
A = [1 -1 0;
     2 3 -5;
     6 7 10;];
B = [1 4 5;
     6 9 10;
     2 -1 4;];
matrixDotProduct(A,B)
```

```
ans = 3×3
     1    -4     0
    12    27   -50
    12    -7    40
```

```
matrixDivide(A,B)
```

```
ans = 3×3
    1.0000   -0.2500         0
    0.3333    0.3333   -0.5000
    3.0000   -7.0000    2.5000
```

```
matrixSquareComp(A)
```

```
ans = 3×3
     1     1     0
     4     9    25
    36    49   100
```

```
matrixSquareComp(B)
```

```
ans = 3×3
     1    16    25
    36    81   100
     4     1    16
```

Problema 11

Să se execute comenzile:

```
format short
a = 4/3
```

```
a = 1.3333
```

```

format rat
a = 4/3
```

```
a = 
       4/3
```

```

format long
a = 4/3
```

```
a = 
   1.333333333333333
```

```
format hex
a = 4/3
```

```
a =
   3ff5555555555555
```

Problema 12

Să se execute comenzile:

```
format short
a = 1; b = 2; c = 3;
fprintf('a = %d, b = %d, c = %d \n', a, b, c)
```

```
a = 1, b = 2, c = 3
```

```
fprintf('a = %d b = %d ', a, b)
```

```
a = 1 b = 2
```

```
fprintf('a + b = %d \n', a + b)
```

```
a + b = 3
```

```
a = 3.7; b = 4;
fprintf('a = %2.3f, b = %8d \n', a, b)
```

```
a = 3.700, b =        4
```

```
fprintf('Suma este = %2.5f \n', a+b)
```

```
Suma este = 7.70000
```

```
x=0:0.2:1
```

```
x = 1×6
         0    0.2000    0.4000    0.6000    0.8000    1.0000
```

```
disp(x)
```

```
         0    0.2000    0.4000    0.6000    0.8000    1.0000
```

```
fprintf('%2.3f ', x);fprintf('\n')
```

```
0.000 0.200 0.400 0.600 0.800 1.000
```

```
fprintf('%2.3f \n',x)
```

```
0.000
0.200
0.400
0.600
```

```
0.800
1.000
```

```
a=[x; 5*x];
disp(a)
```

```
         0    0.2000    0.4000    0.6000    0.8000    1.0000
         0    1.0000    2.0000    3.0000    4.0000    5.0000
```

```
fprintf('%4.2f %10.6f\n',a)
```

```
0.00   0.000000
0.20   1.000000
0.40   2.000000
0.60   3.000000
0.80   4.000000
1.00   5.000000
```

Problema 13

```
help if
```

```
 if Conditionally execute statements.
    The general form of the if statement is

       if expression
         statements
       ELSEIF expression
         statements
       ELSE
         statements
       END

    The statements are executed if the real part of the expression
    has all non-zero elements. The ELSE and ELSEIF parts are optional.
    Zero or more ELSEIF parts can be used as well as nested if's.
    The expression is usually of the form expr rop expr where
    rop is ==, <, >, <=, >=, or ~=.

    Example
       if I == J
         A(I,J) = 2;
       elseif abs(I-J) == 1
         A(I,J) = -1;
       else
         A(I,J) = 0;
       end

    See also RELOP, else, elseif, end, for, while, switch.

    Documentation for if
```

```
help for
```

```
 for    Repeat statements a specific number of times.
    The general form of a for statement is:

       for variable = expr, statement, ..., statement END
```

```
    The columns of the expression are stored one at a time in
    the variable and then the following statements, up to the
    END, are executed. The expression is often of the form X:Y,
    in which case its columns are simply scalars. Some examples
    (assume N has already been assigned a value).

         for R = 1:N
             for C = 1:N
                 A(R,C) = 1/(R+C-1);
             end
         end

    Step S with increments of -0.1
         for S = 1.0: -0.1: 0.0, do_some_task(S), end

    Set E to the unit N-vectors
         for E = eye(N), do_some_task(E), end

    Long loops are more memory efficient when the colon expression appears
    in the for statement since the index vector is never created.

    The BREAK statement can be used to terminate the loop prematurely.

    See also parfor, if, while, switch, break, continue, end, colon.

    Documentation for for
```

```
help while
```

```
 while  Repeat statements an indefinite number of times.
    The general form of a while statement is:

       while expression
         statements
       END

    The statements are executed while the real part of the expression
    has all non-zero elements. The expression is usually the result of
    expr rop expr where rop is ==, <, >, <=, >=, or ~=.

    The BREAK statement can be used to terminate the loop prematurely.

    For example (assuming A already defined):

            E = 0*A; F = E + eye(size(E)); N = 1;
            while norm(E+F-E,1) > 0
               E = E + F;
               F = A*F/N;
               N = N + 1;
            end

    See also for, if, switch, break, continue, end.

    Documentation for while
 ztest  One-sample Z-test.
    H = ztest(X,M,SIGMA) performs a Z-test of the hypothesis that the data
    in the vector X come from a distribution with mean M, and returns the
    result of the test in H.  H=0 indicates that the null hypothesis
    ("mean is M") cannot be rejected at the 5% significance level.  H=1
    indicates that the null hypothesis can be rejected at the 5% level.  The
    data are assumed to come from a normal distribution with standard
    deviation SIGMA.
```

```
X may also be a matrix or an N-D array.  For matrices, ztest performs
separate Z-tests along each column of X, and returns a vector of
results.  For N-D arrays, ztest works along the first non-singleton
dimension of X.  M and SIGMA must be scalars.

ztest treats NaNs as missing values, and ignores them.

[H,P] = ztest(...) returns the p-value, i.e., the probability of
observing the given result, or one more extreme, by chance if the null
hypothesis is true.  Small values of P cast doubt on the validity of
the null hypothesis.

[H,P,CI] = ztest(...) returns a 100*(1-ALPHA)% confidence interval for
the true mean.

[H,P,CI,ZVAL] = ztest(...) returns the value of the test statistic.

[...] = ztest(X,M,SIGMA,'PARAM1',val1,'PARAM2',val2,...) specifies one
or more of the following name/value pairs:

    Parameter       Value
    'alpha'         A value ALPHA between 0 and 1 specifying the
                    significance level as (100*ALPHA)%. Default is
                    0.05 for 5% significance.
    'dim'           Dimension DIM to work along. For example, specifying
                    'dim' as 1 tests the column means. Default is the
                    first non-singleton dimension.
    'tail'          A string specifying the alternative hypothesis:
        'both'  -- "mean is not M" (two-tailed test)
        'right' -- "mean is greater than M" (right-tailed test)
        'left'  -- "mean is less than M" (left-tailed test)

See also ttest, signtest, signrank, vartest.

Documentation for ztest
Other functions named ztest
```

Funcții utilizate

matrixAddition

```
function MA=matrixAddition(A,B)
    MA=A+B;
end
```

matrixSubtraction

```
function MS=matrixSubtraction(A,B)
    MS=A-B;
end
```

matrixCrossProduct

```
function MCP=matrixCrossProduct(A,B)
    MCP=A*B;
end
```

matrixCube

```
function MC=matrixCube(A)
    MC=A*A*A;
end
```

matrixDotProduct

```
function MDP=matrixDotProduct(A,B)
    MDP=A.*B;
end
```

matrixDivide

```
function MD=matrixDivide(A,B)
    MD=A./B;
end
```

matrixSquareComp

```
function MSC=matrixSquareComp(A)
    MSC=A.*A;
end
```