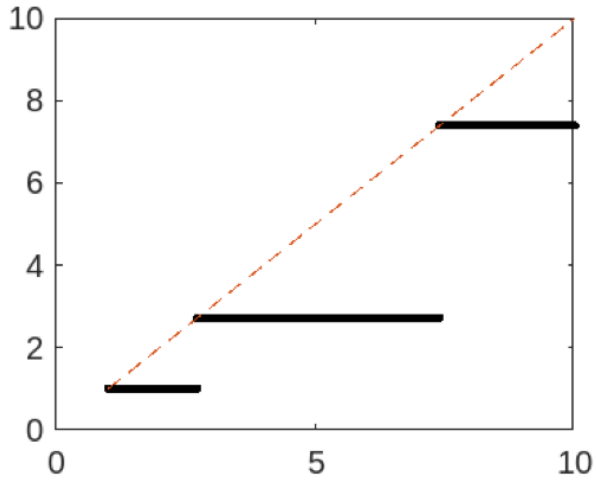
Practical 1 - Introduction to MATLAB and floating-point arithmetic

Exercise 1

1- Normally, the program named Higham should display 4 because it runs 52 times $\sqrt{x}$ and then it runs 52 times x^2.

2- The result is 2.718281808182473e+00, and this can be explained by the fact that MATLAB uses only rounded values due to numerical limitations in the representation of floating-point numbers. The lack of precision in the result is directly associated with rounding errors. We can also notice that the result of the program is the approximation of exp(1).

3- Below is the resulting graph. It can be observed that x and y do not follow the dashed line representing the line x=y. However, at certain points, x and y coincide, notably at 1, 2.7189, and 7.44337. It is noticeable that the result remains the same for a certain period before transitioning to the result of a specific x. We may speculate that iterations could lead to a temporary stabilization around certain constants, resulting in this staircase-like graph.



Exercise 2

1-

```
%ex2
function result = calculateIntegral(n)
    I0 = 1 - exp(-1);


    if n == 0
        result = I0;
    else
        result = -exp(-1) + n * calculateIntegral(n - 1);
    end
end
```

2-

Given $0 \leq x \leq 1$, it follows that $0 \leq x^n \leq 1$, and $\frac{1}{\exp(1)} \leq \exp(-x) \leq 1$. Consequently, we have $0 \leq x^n \exp(-x) \leq \frac{1}{\exp(1)}$. Using the recurrence relation, we can establish $0 \leq x^n \exp(-x) \leq 1 - \exp(-1)$. This implies that $\int_0 \leq \int x^n \exp(-x) \leq \int 1 - \exp(-1)$. Therefore, $0 \leq I_n \leq 1 - \exp(-1)$.

By varying the value of n, it becomes apparent that as n increases, the integral becomes smaller. For instance, with $n = 2$, we have $I_n = 0.1606$, while for $n = 52$, $I_n = 0.007072$, and for $n = 152$, $I_n = 0.0024202$. As we approach an extremely large n, such as 2^{52}, I_n approaches 0.

3-

$I_n = -e^{-1} + n \cdot I_{n-1}$, so $I_n + e^{-1} = n \cdot I_{n-1}$. And so $\frac{I_{n+1}+e^{-1}}{n+1} = I_n$.

4-

```
n = 10;
m = 10;
result = calculate_In(n, m);

disp(['In from n + m = ', num2str(n + m), ' is: ', num2str(result)]);

function result = calculate_In(n, m)
    % Initialize In+m arbitrarily
    I = 12;

    for i = n + m : -1 : n + 1
        I = (I + exp(-1)) / i;
    end

    result = I;
end
```

5-

For $n = 5$ we have:
For $m = 10$: $0.071302sec$
For $m = 20$: $0.071302sec$
For $m = 50$: $0.071302sec$
For $m = 100$: $0.071302sec$
We can notice that the value of m does not influence the values of n.

Exercise 3

1/ MATLAB column-oriented program:

```
m = 3;
n = 4;
A = randn(m, n);

% Column-oriented program
s_column = zeros(m, 1);

tic;
for i = 1:m
    s_column(i) = sum(abs(A(i, :)));
end
time_column = toc;

disp('Column-oriented result:');
disp(s_column);
disp(['Time for column-oriented program: ' num2str(time_column) ' seconds']);
```

Program calculating with BLAS

```
% BLAS
s_blas = zeros(m, 1);

tic;
for i = 1:m
    s_blas(i) = norm(A(i, :), 1);
end
time_blas = toc;

disp('BLAS result:');
disp(s_blas);
disp(['Time for BLAS: ' num2str(time_blas) ' seconds']);
```

By running the programs we can see that the program with BLAS is faster than the Column-oriented program even if the result is the same. For example, for one of the executions we have:

```
Column-oriented result:
     3.081443970331370e+00
     3.168345790014684e+00
     2.657821519083696e+00

Time for column-oriented program: 0.000478 seconds
BLAS result:
     3.081443970331370e+00
     3.168345790014685e+00
     2.657821519083696e+00

Time for BLAS: 0.00037 seconds
>>
```

2/ program which calculates AB :

```
%exo 3 q°2
n = 100;

A = randn(n, n);
B = randn(n, n);

C_manual = zeros(n, n);

% Manual matrix multiplication
tic;
for i = 1:n
    for j = 1:n
        for k = 1:n
            C_manual(i, j) = C_manual(i, j) + A(i, k) * B(k, j);
        end
    end
end
time_manual = toc;
```

When we execute the 2 programs for a 3*3 matrix we have:

```
Time for manual multiplication: 0.000285 seconds
Time for built-in multiplication: 2.5e-05 seconds
```

We notice that manual multiplication takes almost 10 times longer than built-in multiplication. And we notice this time difference between the 2 modes even more when we take matrices of size 100 with:

```
Time for manual multiplication: 0.026168 seconds
Time for built-in multiplication: 0.000214 seconds
>>
```

Exercise 4

1-

```
%ex4
function [L, U, P] = lup_decomposition(A)
    [m, n] = size(A);

    if m ~= n
        error('Input matrix must be square for LUP decomposition.');
    end

    L = eye(n);  % Initialize L as an identity matrix
    P = eye(n);  % Initialize P as an identity matrix
    U = A;       % Initialize U as a copy of the input matrix

    for k = 1:n
        % Partial pivoting: find the pivot element and swap rows if necessary
        [~, pivot_row] = max(abs(U(k:n, k)));
        pivot_row = pivot_row + k - 1;

        if pivot_row ~= k
            % Swap rows in U
            temp_row = U(k, :);
            U(k, :) = U(pivot_row, :);
            U(pivot_row, :) = temp_row;

            % Swap rows in P
            temp_row = P(k, :);
            P(k, :) = P(pivot_row, :);
            P(pivot_row, :) = temp_row;

            % Swap rows in L (only the part below the diagonal)
            if k > 1
                temp_row = L(k, 1:k-1);
                L(k, 1:k-1) = L(pivot_row, 1:k-1);
                L(pivot_row, 1:k-1) = temp_row;
            end
        end

        % Gaussian elimination to update U and L
        for i = k+1:n
            factor = U(i, k) / U(k, k);
            L(i, k) = factor;
            U(i, k:n) = U(i, k:n) - factor * U(k, k:n);
        end
    end
end
```

2/ Example 1:

```
A = [2, 1, -1; -3, -1, 2; -2, 1, 2];
tic;
[L, U, P] = lup_decomposition(A);
lup_time = toc;

disp(' LUP Decomposition Time:');
disp(lup_time);

disp('LUP Decomposition:');
disp('L:');
disp(L);
disp('U:');
disp(U);
disp('P:');
disp(P);
```

```
LUP Decomposition:
L:
    1.0000         0         0
    0.6667    1.0000         0
   -0.6667    0.2000    1.0000

U:
   -3.0000   -1.0000    2.0000
         0    1.6667    0.6667
         0         0    0.2000

P:
     0     1     0
     0     0     1
     1     0     0
```

Example 2 :

```
79
80      %ex4
81
82      A = [4, 2, 1; 7, 10, 2; 2, 5, 6];
83      tic;
84      [L, U, P] = lup_decomposition(A);
85      lup_time = toc;
86
87      disp(' LUP Decomposition Time:');
88      disp(lup_time);
89
90      disp('LUP Decomposition:');
91      disp('L:');
92      disp(L);
93      disp('U:');
94      disp(U);
95      disp('P:');
96      disp(P);
97
```

Command Window

```
LUP Decomposition:
L:
     1.000000000000000e+00                         0                         0
     5.714285714285714e-01     1.000000000000000e+00                         0
     2.857142857142857e-01    -5.769230769230771e-01     1.000000000000000e+00

U:
     7.000000000000000e+00     1.000000000000000e+01     2.000000000000000e+00
                         0    -3.714285714285714e+00    -1.428571428571428e-01
                         0                         0     5.346153846153847e+00

P:
     0     1     0
     1     0     0
     0     0     1
```

3/ When we execute the matlab lu decomposition we can see that the result is the same :

```
MATLAB lu Decomposition:
L (MATLAB):
     1.000000000000000e+00                         0                         0
     5.714285714285714e-01     1.000000000000000e+00                         0
     2.857142857142857e-01    -5.769230769230771e-01     1.000000000000000e+00

U (MATLAB):
     7.000000000000000e+00     1.000000000000000e+01     2.000000000000000e+00
                         0    -3.714285714285714e+00    -1.428571428571428e-01
                         0                         0     5.346153846153847e+00

P (MATLAB):
     0     1     0
     1     0     0
     0     0     1
```

But when we compare the execution times of each program we see that the MATLAB program is much faster with an execution time approximately 10 times faster. For example, for the matrix seen previously (example 2) :

LUP Decomposition Time = 1.817000000000000e-03 MATLAB lu Decomposition Time = 1.300000000000000e-04