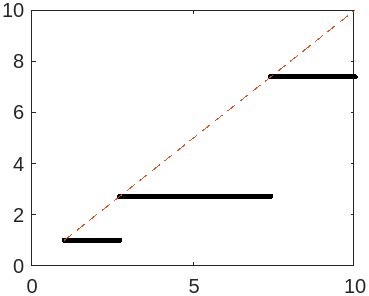
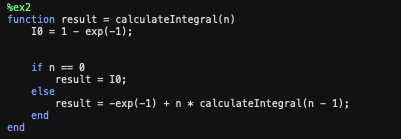
Practical 1 - Introduction to MATLAB and floating-point arithmetic

Exercise 1

1. Normally, the program named Higham should display 4 because it runs 52 times sqrt(x) end 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.



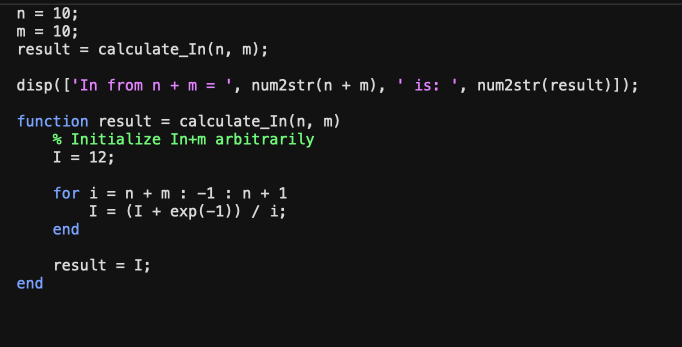
Exercice 2

1-

2- Given 0 ≤ x ≤ 1, it follows that 0 ≤ xn ≤ 1, and 1/exp(1) ≤ exp( -x ) ≤ 1. Consequently, we have 0 ≤ xn \* exp( -x ) ≤ 1/exp(1). Using the recurrence relation, we can establish 0 ≤ xn \* exp( -x ) ≤ 1 – exp(-1). This implies that ∫ 0 ≤ ∫ xn \* exp( -x ) ≤ ∫ 1 - exp(-1). Therefore, 0 ≤ In ≤ 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 In= 0.1606, while for n=52, In= 0.007072, and for n=152, In= 0.0024202. As we approach an extremely large n, such as 252, In approaches 0.

3/ In = −e−1 + n.In−1 so In +e−1 = n.In−1. and so (In+1 +e−1 )/( n+1) = In.

4/

5/ For n=5 we have :

for m=10 : 0.071302

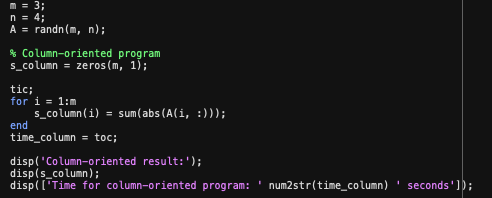
for m= 20 : 0.071302

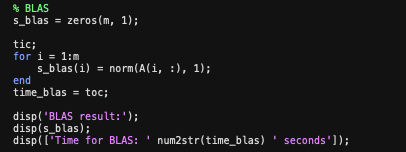
for m= 50 : 0.071302

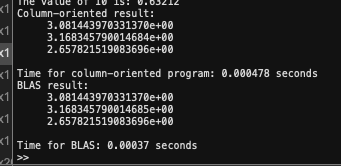
for m= 100 : 0.071302

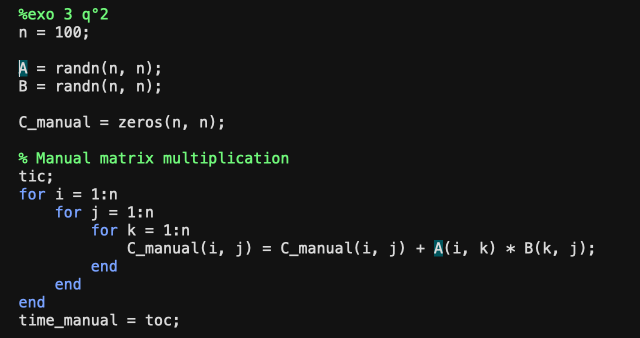
We can notice that the value of m does not influence the values of n.

Exercice 3

1/ MATLAB column-oriented program:

Program calculating with BLAS

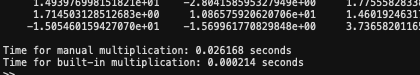
By running the programs we can see that the program with BLAS is faster than the Column-oriented program even if the result is the same. For example, for one of the executions we have:

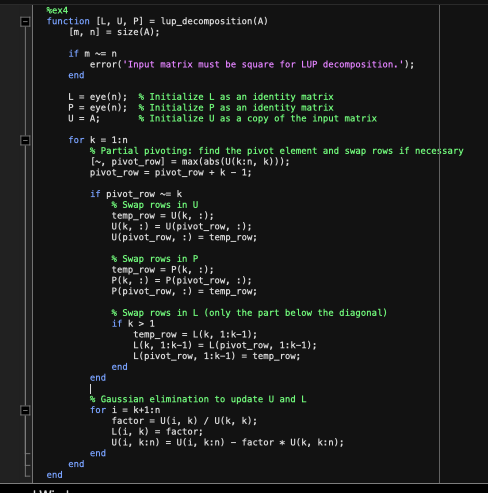
2/ program which calculates AB :

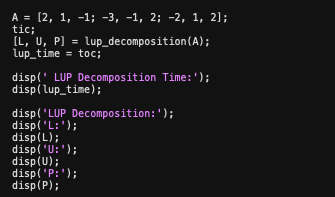
When we execute the 2 programs for a 3\*3 matrix we have:

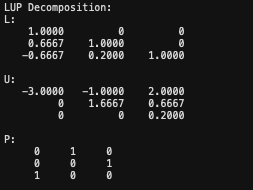
On remarque que la multiplication manuelle prend près de 10 fois plus de temps que la built-in multiplication.

We notice that manual multiplication takes almost 10 times longer than built-in multiplication.

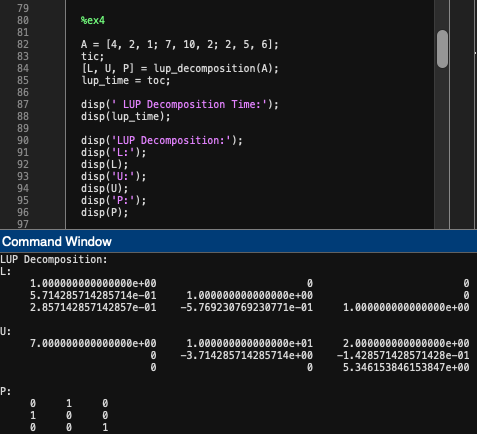
We notice this time difference between the 2 modes even more when we take matrices of size 100 with:

Exercise 4

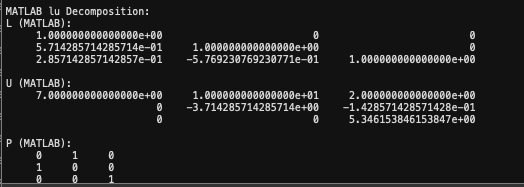
2/ Example 1:



Example 2 :



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



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

Full code :

Higham(2)

%ex 2

n=0;

result = calculateIntegral(n);

disp(['The value of I', num2str(n), ' is: ', num2str(result)]);

n = 2;

m = 50;

result = calculate\_In(n, m);

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

%exo3

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']);

% 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']);

%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;

% Built-in matrix multiplication

tic;

C\_builtin = A \* B;

time\_builtin = toc;

% Display the results and timings

disp('Manual multiplication result:');

disp(C\_manual);

disp('Built-in multiplication result:');

disp(C\_builtin);

disp(['Time for manual multiplication: ' num2str(time\_manual) ' seconds']);

disp(['Time for built-in multiplication: ' num2str(time\_builtin) ' seconds']);

%ex4

A = [4, 2, 1; 7, 10, 2; 2, 5, 6];

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);

% Compare with MATLAB lu function

tic;

[L\_matlab, U\_matlab, P\_matlab] = lu(A);

matlab\_lu\_time = toc;

disp('MATLAB lu Decomposition Time:');

disp(matlab\_lu\_time);

disp('MATLAB lu Decomposition:');

disp('L (MATLAB):');

disp(L\_matlab);

disp('U (MATLAB):');

disp(U\_matlab);

disp('P (MATLAB):');

disp(P\_matlab);

%ex 1

x = logspace(0, 1, 2013);

y = Higham(x);

plot(x, y, 'k.', x, x, '--')

format longE;

function res = Higham(x)

for i=1:52

x=sqrt(x);

end

for i=1:52

x = x.^2; end

res = x; end

%ex2

function result = calculateIntegral(n)

I0 = 1 - exp(-1);

if n == 0

result = I0;

else

result = -exp(-1) + n \* calculateIntegral(n - 1);

end

end

%exo2Q4

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

%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