Sorbonne Université

Numerical Algorithms

Practical 1

Méline Trochon
Anatole Vercelloni

Teacher : Stef Graillat

Janvier 2023

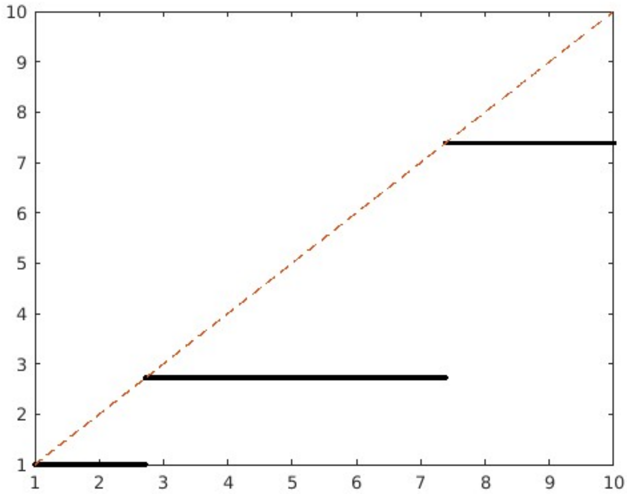
Table des matières

1 Exercise 1

1. The program should display x.

2. We run the program with 10 and we got 7.3891. The result is different from what we expected because there is round up with floating point arithmetic.

3. We run the code and obtain :



The abscissa represent x and the ordinate is what we compute. If the computation were exact we should have a line x = y. So all the point on the diagonal are "exact" computations. We can remark that these points correspond to e^i, $i \in \mathbb{N}$. We can explain that by the fact that, $x \geq 1$, so $\sqrt{x} \geq 1$, so the decimal part will be absorbed, the e^i are accurate because of the construction of the floating point numbers.

2 Exercise 2

Let $f_n(x) := x^n e^{-x}$ and $I_n = \int_0^1 f_n(x)\,dx$

1. We wrote a program named 'integraln'.

2. We computed the derivative of f_n and it is vanished in 0 and n. So, for $n \geq 1$, f is growing between 0 and 1. Therefore f([0,1]) = [0, e^{-1}].

For I_n, we can do successive integration by part and get :
$\int_0^1 x_n e^{-x}\,dx$
$= [-x^n e^{-x}]_0^1 + \int_0^1 n x^{n-1} e^{-x}\,dx$
...
$= -e^{-1} - ne^{-1} - n(n-1)e^{-1} - ... - n!e^{-1} + n! \int_0^1 x_n e^{-x}\,dx$
$= n! - e^{-1} \sum_{k=0}^{n} \frac{n!}{k!}$
$\to n! - e^{-1} e^1 n!$ (exponential series)
$\to 0$

So a cloture for In is $[0, 1 - e^{-1}]$ (we can show easily that I_n is a decreasing sequence, indeed $\frac{I_{n+1}}{I_n} < 1$)
3. We have :

$$\begin{aligned} I_n &= -e^{-1} + n.I_{n-1} \\ \iff I_{n-1} &= (I_n + e^{-1})/n \\ \iff I_n &= (I_{n+1} + e^{-1})/(n+1) \end{aligned}$$

4. We wrote a program 'integralnm' that takes three arguments (n, m and I_{n+m}) and return the computation of I_n.
5. As we can see, we are near the results in a very short amount of iterations.

```
>> integralnm(5, 10, 12)

ans =

   0.071302178178496

>> integralnm(5, 20, 12)

ans =

   0.071302178109803

>> integralnm(5, 50, 12)

ans =

   0.071302178109803

>> integralnm(5, 100, 12)

ans =

   0.071302178109803
```

```
>> integralnm(10, 10, 12)

ans =

   0.036461334624958

>> integralnm(10, 20, 12)

ans =

   0.036461334624107

>> integralnm(10, 50, 12)

ans =

   0.036461334624107

>> integralnm(10, 100, 12)

ans =

   0.036461334624107
```

Figure 1 – results for n=5 and n=10

3 Exercise 3

1. We created two functions col_sum, which is that takes A then return S, vector $of s_i$, then we implemented blas_sum using the norm function.

2.

```
>> A=rand(1000);
>> tic; col_sum(A); toc
Elapsed time is 0.025971 seconds.
>> tic; blas_sum(A); toc
Elapsed time is 0.013012 seconds.
>> A=rand(2000);
>> tic; col_sum(A); toc
Elapsed time is 0.015579 seconds.
>> tic; blas_sum(A); toc
Elapsed time is 0.054584 seconds.
>> A=rand(5000);
>> tic; col_sum(A); toc
Elapsed time is 0.072414 seconds.
>> tic; blas_sum(A); toc
Elapsed time is 0.172087 seconds.
>> A=rand(10000);
>> tic; col_sum(A); toc
Elapsed time is 0.269613 seconds.
>> tic; blas_sum(A); toc
Elapsed time is 0.592548 seconds.
```

```
>> A = magic(100); B = magic(100);
>> tic; A*B; toc
Elapsed time is 0.000157 seconds.
>> tic; matrix_prod(A, B); toc
Elapsed time is 0.030192 seconds.
>> A = magic(1000); B = magic(1000);
>> tic; A*B; toc
Elapsed time is 0.032489 seconds.
>> tic; matrix_prod(A, B); toc
Elapsed time is 4.805673 seconds.
>> A = magic(1500); B = magic(1500);
>> tic; A*B; toc
Elapsed time is 0.076746 seconds.
>> tic; matrix_prod(A, B); toc
Elapsed time is 30.591490 seconds.
```

Figure 2 – Efficiency comparison for the sums then the matrix product

We can see that the col_sum is worse than what we blas_sum for small matrices, but it get better for matrix of size greater than 2000, and the difference is wider the greater the matrix's size is.

But for the matrix multiplication it is the contrary. The implemented function is better for small matrix. Then, for large enough matrix, the difference between the two implementation is huge.

4 Exercise 4

1. We wrote a programme called 'lup'.

2. We test the LU-decomposition on A = $\begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{pmatrix}$

And we get :

```
>> [L_,U_,P_] = lup(A)

L_ =

    1.0000         0         0
    0.6667    1.0000         0
    0.3333    0.5000    1.0000


U_ =

     3     6     9
     0     2     4
     0     0     0


P_ =

     3
     1
     2
```

We can compare that with the computation of the lu fonction implemented in matlab which is :

```
>> [L,U,P] = lu(A)

L =

    1.0000         0         0
    0.3333    1.0000         0
    0.6667    0.5000    1.0000


U =

     3     6     9
     0     2     4
     0     0     0


P =

     0     0     1
     1     0     0
     0     1     0
```

The difference is that we made the choice to represented the permutation matrix by a vector.