TME n° 1

Ryan Lahfa

22 décembre 2019

Table des matières

Table des figures

Exercice 1

Q1. Nous nous attendons à avoir 4 en sortie, puisqu'on calcule $\left(\sqrt{4}^{(\circ 52)}\right)^{2^{52}}$ de façon itérative.

Q2. On obtient: `2.718281808182473e+00`.

Au préalable, remarquons que $f_{k+1} : x \mapsto \sqrt{f_k(x)}$ converge simplement vers la fonction constante égale à 1 lorsque $k \to +\infty$ pour $k \geq 0$ avec $f_0 : x \mapsto x$.

Tout d'abord, écrivons $4 = \exp(\log(4))$.

Si on pose: $x_{k+1} = \exp\left[\frac{\log(x_k)}{2}\right]$.

Or, on l'effectue 52 fois d'une part, donc le résultat obtenu est: $\exp\left[\frac{\log(4)}{2^{52}}\right]$.

Or: $\log(4) = 2\log(2) \sim 1.3$, autrement dit: on calcule $2^{-52} \sim 10^{-16}$.

On comprend donc que $\log(4) \times 2^{-52} \sim 2^{-52}$, d'où, en repassant à l'exponentielle, on obtient: ~ 1.

Pour être précis, quelque chose de l'ordre de $1 + 2^{-52}$.

D'où, dans la seconde phase, on met ceci 52 fois au carré.

Cependant, si on fait ça 52 fois ou ∞ fois, ça ne change pas grand chose numériquement, puisqu'on a des éléments en $10^{-16}, 10^{-32}, \ldots$. D'où lorsque $n \to +\infty$, on a:

$$\left(1 + 1 \times 2^{-n}\right)^{2^n} \to e$$

Ce qui fournit le résultat qu'on attendait.

Q3. On obtient le tracé de la figure 1.

On constate empiriquement que ces points sont des puissances de e (constante de Neper [1]).

On intuite donc des intervalles en $I_k = [\exp(k), \exp(k+1)[$ pour $k \in \mathbb{N}$.

La raison heuristique qu'on peut trouver en se basant sur la question précédente, c'est de déterminer lorsque le résultat des racines carrées consécutives n'est pas 1 mais $1 + k \times \varepsilon$ pour un $\varepsilon \sim 2^{-52}$.

Rigoureusement, cela revient à demander donc lorsque le premier chiffre de $\log(x)$ est k, donc, par monotonie: $k \leq \log(x) < k + 1 \iff \exp(k) \leq x < \exp(k+1)$, puisque les chiffres qui suivent après sont écrasés par 2^{-52}.

Ainsi, on conclut avec l'analyse précédente, qu'on obtient $\exp(k)$ en remplaçant 1 par k.

Ceci est tout à fait conforme avec le graphique de la figure 1 où on constate $\exp(0) = 1, \exp(1), \exp(2)$.

Exercice 2

Q1. Tout d'abord, la précision machine est définie comme le plus petit flottant ε tel que $1 + \varepsilon \neq 1$.

D'où :

```
function res = machine_prec()
    e = 1;
    while e + 1 > 1
        res = e;
```

1. Ou nombre d'Euler, au choix.

```
        e = e/2;
    end
end
```

On obtient le résultat exact:

```
Calculée manuellement:  2.220446049250313e-16
Calculée théoriquement: 2.220446049250313e-16 = 2^(-52)
Constante MATLAB:   2.220446049250313e-16
```

Q2. Un flottant normalisé est défini comme ayant un exposant supérieur à 1, donc on cherche une puissance de 2.

D'où :

```
function res = normalized_min()
    e = eps;
    var = 1;
    while var + e ~= var
        res = var;
        e = e / 2;
        var = var / 2;
    end
end
```

On obtient le résultat exact:

```
Calculée manuellement:  2.225073858507201e-308
Calculée par sa valeur: 2.225073858507201e-308 = 2^(-1022)
Constante MATLAB:   2.225073858507201e-308
```

Q3. Un flottant dénormalisé est toujours une puissance de 2, mais cette fois ci, l'exposant est nul.

```
function res = denormalized_min()
    var = 1;
    while var > 0
        res = var;
        var = var / 2;
    end
end
```

On obtient le résultat exact:

```
Calculée manuellement:  4.940656458412465e-324
Calculée théoriquement: 4.940656458412465e-324 = 2^(-51) × 2^(-1021) = 2^(-1074)
Constante MATLAB:   4.940656458412465e-324
```

Q4. Tout d'abord, on part d'une mantisse adéquate $01\ldots11\ldots1$ en représentation IEE754.

Puis, une multiplication par 2 ne change pas la mantisse.

Lorsque on aura atteint l'infini, le flottant juste avant sera le plus grand flottant représentable.

On se donne:

```
function res = max_float()
    x = 2 - eps;
    while x < inf
        res = x;
        x = x*2;
    end
end
```

On obtient le résultat exact:

```
Calculée manuellement:  1.797693134862316e+308
Calculée théoriquement: 1.797693134862316e+308 = (2 - 2^(-52))2^(1023)
Constante MATLAB:   1.797693134862316e+308
```

Exercice 3

Q1. On remarque, lorsque $x \to 0$, on a: $\exp(x) = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + o(x^3)$.

D'où: $f(x) = \frac{1}{2} + \frac{x}{6} + o(x)$.

Donc, en pratique, on va avoir des erreurs de calcul en raison du quotient d'éléments très petits.

Q2. On peut reprendre l'équation qu'on a obtenu tout à l'heure, et la prolonger jusqu'à un ordre arbitraire, disons 15.

Exercice 4

Q1. On écrit:

```
function res = sinus_taylor1(x)
    acc = 0;
    for i=1:15
        acc = acc + (-1)^i * (x.^(2*i + 1)) / factorial(2*i + 1);
    end
    res = acc;
end
```

Q2. On utilise un mécanisme de comparaison avant l'itérée précédent pour déterminer le critère d'arrêt dans le calcul.

```
function res = sinus_taylor2(x)
    r = mod(x, 2*pi);
```

```
    acc = 0;
    i = 0;
    limit_reached = false;
    while ~limit_reached
        new_acc = acc + (-1)^i * (r.^(2*i + 1)) / factorial(2*i + 1);
        if new_acc == acc
            limit_reached = true;
        end
        acc = new_acc;
        i = i + 1;
    end
    res = acc;
end
```

Q3. On obtient la figure 2, les erreurs relatives étant mentionnés par des barres sur les points.

On constate de très bonnes erreurs relatives.

Exercice 5

Q1/2. On traite les deux questions simultanément, tout d'abord, voici l'implémentation de référence pour la comparaison:

```
function s = manual_norm(M)
    [m, n] = size(M);
    s = zeros(1, m);
    for i=1:m
        for j=1:n
            s(i) = s(i) + abs(M(i, j));
        end
    end
end

function R = mat_product(A, B)
    [m, n] = size(A);
    [p, q] = size(B);

    if p ~= n
        error('format invalid');
    end

    R = zeros([m, q]);

    for i=1:m
        for j=1:q
```

```
            for k=1:n
                R(i, j) = R(i, j) + A(i, k)*B(k, j);
            end
        end
    end
end
```

Puis, on trace à la figure 3 les graphes de temps.

Comme prévu, on voit que les BLAS sont nettement plus rapides que les fonctions MATLAB et les BLAS sont aussi beaucoup plus stables en temps que MATLAB.

On peut expliquer cela simplement par le fait que les BLAS étant écrites en C, sont compilées et optimisées sur les architectures de processeurs cibles, exploitent tous les jeux d'instructions disponibles tandis que le code MATLAB est interprété, a priori, MATLAB semble avoir un compilateur juste à temps, ce qui peut expliquer les variations du graphe à certain moments.

Reste toujours que les BLAS sont vraiment bien plus rapides.

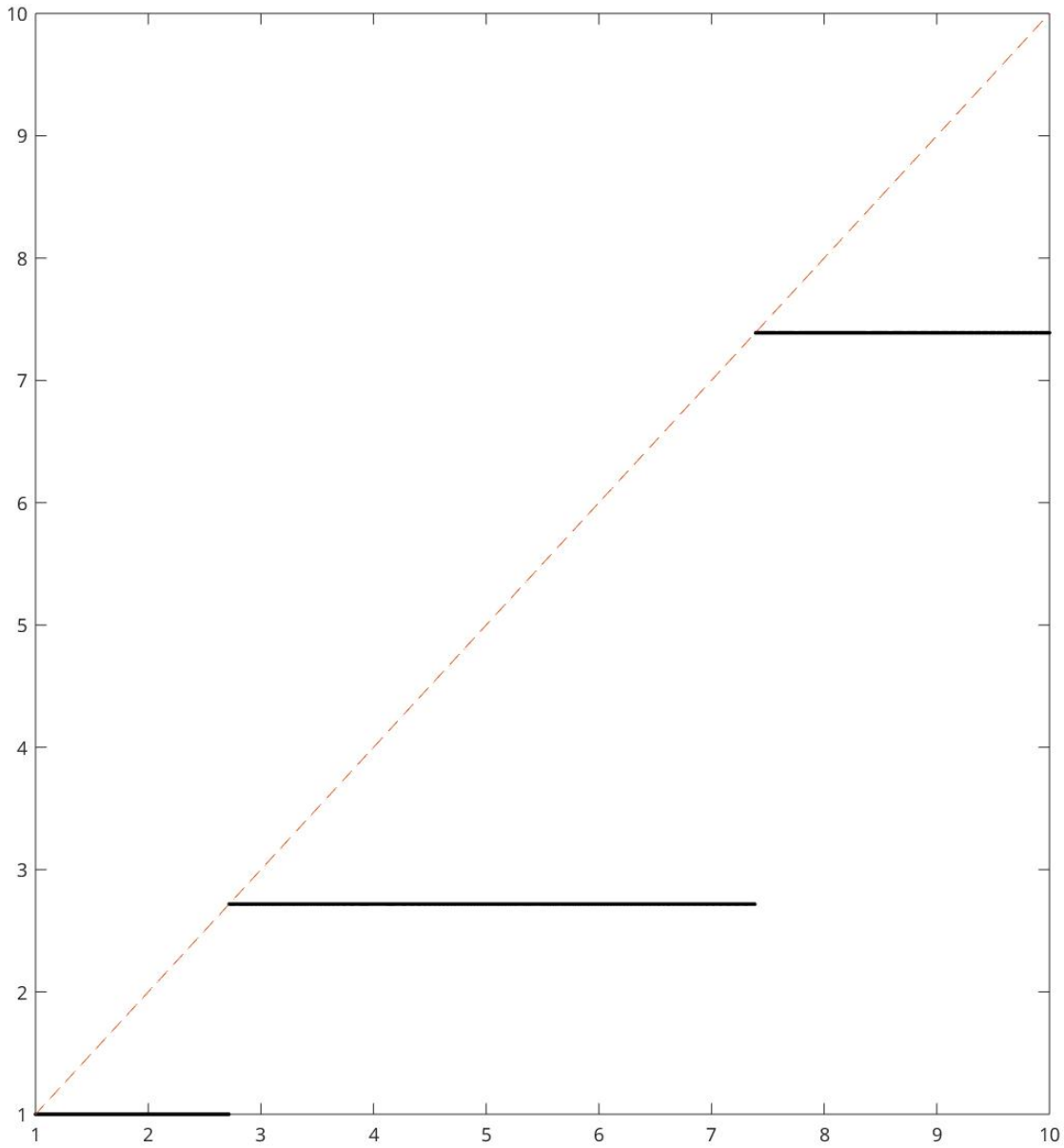

FIGURE 1 – Tracé de $y = x$ et $y = \text{Higham}(x)$

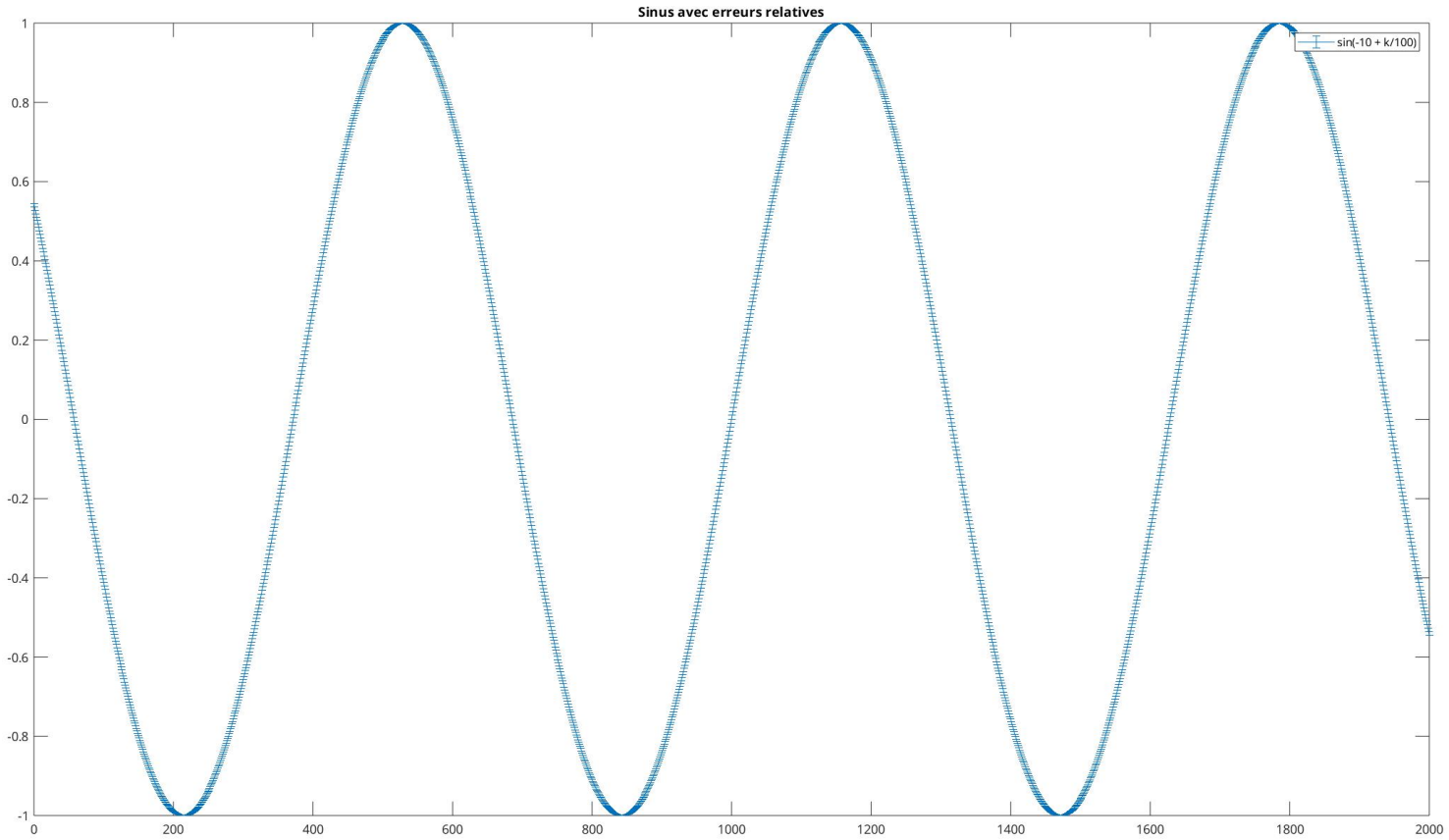

FIGURE 2 – Sinus amélioré

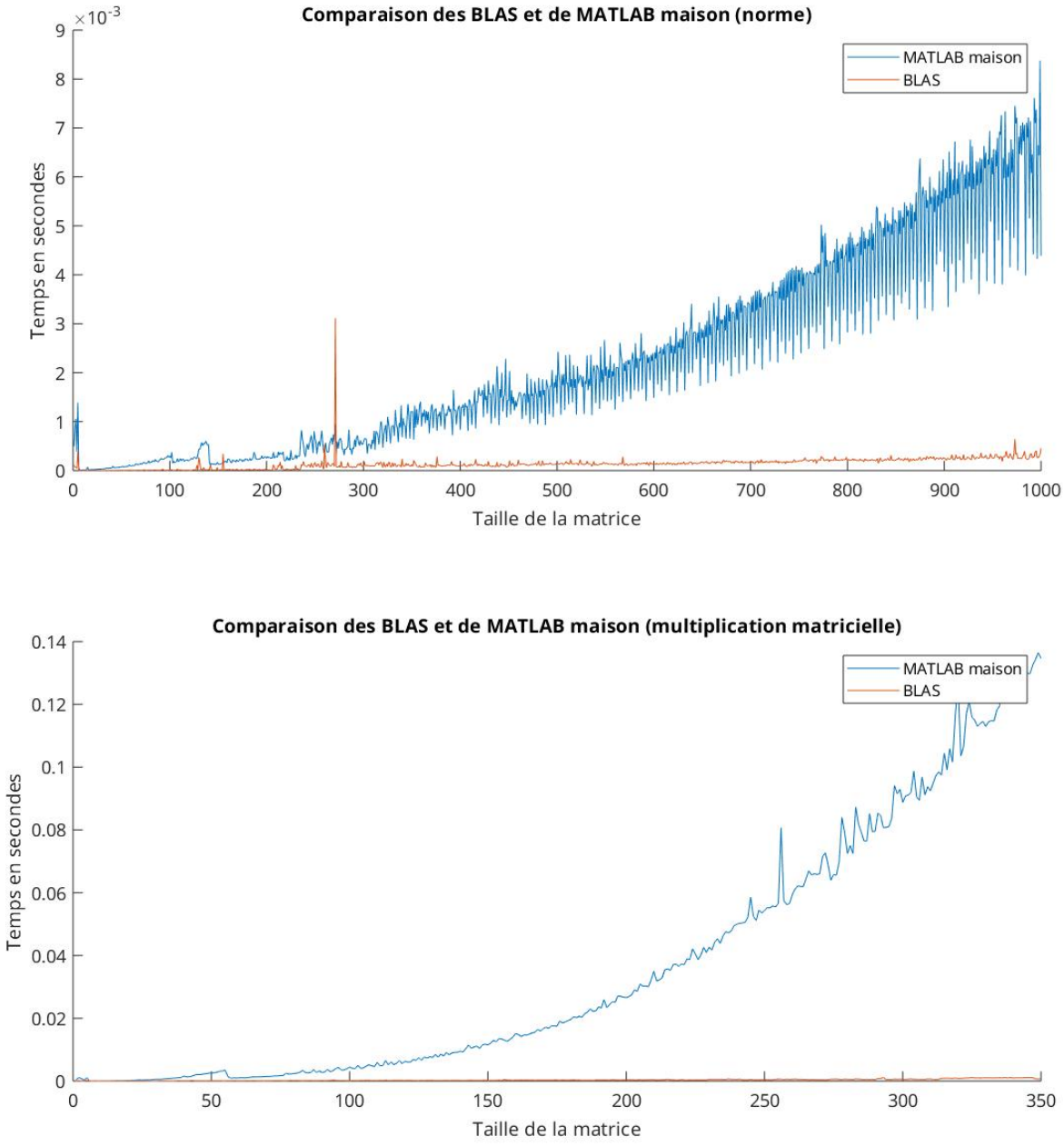

FIGURE 3 – BLAS vs MATLAB