TME n° 2

Ryan Lahfa

22 décembre 2019

Table des matières

Table des figures

Exercice 11

Q1. On implémente la méthode en se basant sur le cours:

```
function res = golden_search(a, b, f, tol)
    tau = (sqrt(5) - 1)/2;
    x1 = a + (1 - tau)*(b - a);
    f1 = f(x1);
    x2 = a + tau*(b - a);
    f2 = f(x2);
    i = 0;

    while (b - a) > tol && i < 100
        if f1 > f2
```

```
            a = x1;
            x1 = x2;
            f1 = f2;
            x2 = a + tau*(b - a);
            f2 = f(x2);
        else
            b = x2;
            x2 = x1;
            f2 = f1;
            x1 = a + (1 - tau)*(b - a);
            f1 = f(x1);
        end
        i = i + 1;
    end
    res = x1;
end
```

Q2. On n'emploiera pas la Symbolic Toolbox pour les calculs de dérivée première et seconde, mais on optera pour des différences finies du premier ordre et du second ordre, en effet, compte tenu de comment on utilisera la méthode de Newton, cela semble plus judicieux.

Au passage, on démontre un petit résultat sur la différence finie du second ordre appliquée à f par Taylor:

Lorsque $h \to 0^+$, on a:

$$f(x+h) = f(x) + f'(x)h + \frac{f''(x)}{2}h^2 + o(h^2)$$
$$f(x-h) = f(x) - f'(x)h + \frac{f''(x)}{2}h^2 + o(h^2)$$

D'où:

$$f''(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}$$

Ce qui fournit le code suivant:

```
function res = num_deriv(f, x, h)
    res = (f(x + h) - f(x))/h;
end

function res = num_deriv2(f, x, h)
    res = (f(x + h) - 2*f(x) + f(x - h))/h^2;
end
```

```
function res = newton(x0, f, nb_iterations, step_size)
    x_k = x0;

    df = num_deriv(f, x_k, step_size);
    df2 = num_deriv2(f, x_k, step_size);

    for i=1:nb_iterations
        x_k = x_k - df/df2;
        df = num_deriv(f, x_k, step_size);
        df2 = num_deriv2(f, x_k, step_size);
    end

    res = x_k;
end
```

Q3. On obtient donc les tracés de la figure 1, 2, 3, 4, 5.

Tout d'abord, pour la figure 1, on constate que toutes les méthodes sont d'accord, cela assez normal, d'autant plus que la fonction à optimiser est de classe $\mathcal{C}^\infty$ sur $\mathbb{R}$, donc se prête très bien à la dérivée seconde (pour Newton en tout cas).

Ensuite, pour la figure 2, cette fois-ci, Newton se trompe complètement, tandis que la section dorée et MATLAB sont en accord. Cela peut s'expliquer par le caractère périodique de sin que Newton ne prend pas vraiment en compte, les intervalles étant restreints, cela rend la tâche encore plus difficile, puisque seul $\pi/2$ est point critique de sin sur cet intervalle.

Maintenant, pour la figure 3, on constate que la section dorée et MATLAB sont à nouveau en accord tandis que Newton se trompe, cette fois-ci, on est sur un intervalle qui est envoyée par le logarithme sur deux intervalles. Le logarithme et la valeur absolue modifie le caractère dérivable en 1, et donc compromette la faisabilité de la méthode de Newton, surtout lorsque le minimum global est justement en 1.

À présent, pour la figure 4, on constate que la section dorée et Newton sont plutôt en accord, tandis que MATLAB a une précision nettement plus élevée. Or, lorsqu'on regarde le graphe de la fonction, on constate qu'il y a un panel de valeurs autour de 0 qui sont arbitrairement proche du minimum global. Donc ici, on a une problématique de précision qui peut sûrement se résoudre en augmentant la tolérance, mais sauf en utilisant des flottants en précision arbitraire, cette solution a une limite qui sont les limites eux même de représentation en norme IEE754. Il est aussi possible, que les dérivées successives alternent de signe ce qui rend encore plus difficile la convergence en raison des changements de monotonies locales qui pourraient survenir.

Finalement, pour la figure 5, on constate le même problème, cependant celui-ci se résoudra plus facilement par la solution présenté au dessus en raison du caractère affine du graphe, tandis que précédemment, la fonction précédente est

en $x^2 - \frac{2x^4}{3}$ localement en 0. Pire, la dérivée seconde est nulle et on ne dispose pas de dérivée première en 0.

Exercice 12

Q1.

```
syms f(x, y);
f(x, y) = 100*(y - x^2)^2 + (1 - x)^2;

% Q1.
G = gradient(f);
H = hessian(f);
```

Q2. On calcule le gradient et le déterminant en ce point:

```
grad f(x^*) = (0, 0), det H_f(x^*) = 400
```

D'où, ceci est un point critique, et de plus, un minimum local par caractérisation de la hessienne.

Q3/4. On traite la question 4 dans la foulée.

```
% Q3.
figure;
subplot(2,1,1);
iters = newton(f, [-1 -2], 5)
hold on
for i=1:5
    plot(iters(i, 1), iters(i, 2), 'ro');
end
for i=1:4
    p1 = iters(i, :);
    p2 = iters(i + 1,:);
    dp = p2 - p1;
    quiver(p1(1), p1(2), dp(1), dp(2), 0);
end
fcontour(f, [-1.5 2 -3 3]); % ezcontour est obsolète, il est désormais remplacé par fcontou

% Q4.
error = zeros(5, 1);

for i=1:6
    error(i) = norm(iters(i,:) - [1 1]);
end

subplot(2,1,2);
plot(0:5, error);
```

```
% Newton impl
function iters = newton(f, x0, itmax)
    H = hessian(f);
    G = gradient(f);
    x = x0;
    iters = zeros(itmax + 1, 2);
    iters(1,:) = x0;
    for i=1:itmax
        s = linsolve(H(x(1), x(2)), -G(x(1), x(2)))';
        x = x + s;
        iters(i + 1,:) = x;
    end
end
```

On en tire les tracés de la figure 6.

On observe donc une convergence assez rapide vers le minimum local attendu, en 4 itérées dans l'ordre fléché.

On voit donc que l'algorithme de Newton effectue une recherche en zig-zag à la façon d'une méthode du gradient.

Exercice 13

Q1/2. On traite les deux questions simultanément, tout d'abord, on se donne les implémentations de référence:

```
function y = gradient_method(f, alpha, x0, itmax)
    y = x0;
    G = gradient(f);
    for i=1:itmax
        y = y - alpha*G(y(1), y(2))';
    end
end
```

Puis on trace les erreurs par rapport aux α en faisant une recherche dans la figure 7.

Remarque que l'axe des ordonnées est logarithmique, puisque les erreurs sont de l'ordre de 10^M avec $5 \leq M \leq 30$.

On constate donc que la méthode du gradient est très sensible par rapport à son α et que de petites variations peuvent entraîner des erreurs catastrophiques, comme pour f.

Mais pour la fonction de Rosenbrock, les variations en erreurs sont faibles, mais les variations d'α sont très très petits.

Par ailleurs, on constate que l'efficacité de la méthode du gradient est très mauvaise, elle est extrêmement lente et peu d'itérations peuvent être effectués. [1]

Q3. Voici le code, d'abord:

```
function t = wolfe_linear_search(f, G, y, d)
    t_g = 0;
    t_d = +inf;
    t = 1;
    m_1 = 0.1;
    m_2 = 0.9;
    d_g0 = dot(-d, d);


    go = true;
    while go
        % dérivées directionnelles
        % h : R → R^p
        % h(t) = x + th
        % g = f rond h
        % g'(t) = (grad f(x + th) | h)
        p = y + t*d;
        d_gt = dot(G(p(1), p(2))', d);

        c1 = m_1 * t * d_g0;
        c2 = m_2 * d_g0;

        % g(t) <= g(0) + m_1 t g'(0) && g'(t) >= m_2 g'(0)
        if f(p(1), p(2)) <= f(y(1), y(2)) + c1 && d_gt >= c2
            go = false;
        % g(t) >= g(0) + m_1 t g'(0)
        elseif f(p(1), p(2)) > f(y(1), y(2)) + c1
            t_d = t;

            if isinf(t_d)
                t = 10*t_g;
            else
                t = (t_d + t_g) / 2;
            end
        % g(t) <= g(t) + m_1 t g'(0) && g'(t) < m_2 g'(0)
        elseif f(p(1), p(2)) <= f(y(1), y(2)) + c1 && d_gt < c2
            t_g = t;
            if isinf(t_d)
                t = 10*t_g;
```

1. L'auteur a découvert que MATLAB perdait le contrôle du processus sur certaines entrées et qu'il fallait `killall -SIGKILL MATLAB` afin de pouvoir interrompre l'exécution de force.

```
            else
                t = (t_d + t_g)/2;
            end
        end
    end
end

function y = gradient_with_wolfe(f, x0, itmax)
    y = x0;
    G = gradient(f);

    for i=1:itmax
        fprintf("[Wolfe] %d/%d\n", i, itmax);
        u = G(y(1), y(2))';
        a = wolfe_linear_search(f, G, y, -u);
        y = y - a*u;
    end
end
```

On emploie la Symbolic Toolbox car les différences finies semblaient inappropriés dans ce contexte, en pratique, on peut faire de la différentiation automatique, donc c'est acceptable.

On obtient le résultat suivant:

```
x = [-5.28345e-01 2.92796e-01] (erreur en norme 2: 1.68404e+00)
```

En 12 itérations, mais la méthode est très lente, on voit donc que sur Rosenbrock, la méthode du Gradient à pas optimal n'améliore pas significativement les performances de la méthode du gradient à α fixé sauf si α choisi n'est pas très bon.

Mais comme on peut le voir, chercher le bon α prend à peu près le même temps de façon empirique que de façon automatique.

On conclut donc que les deux méthodes se valent et doivent être utilisés selon les cas précis.

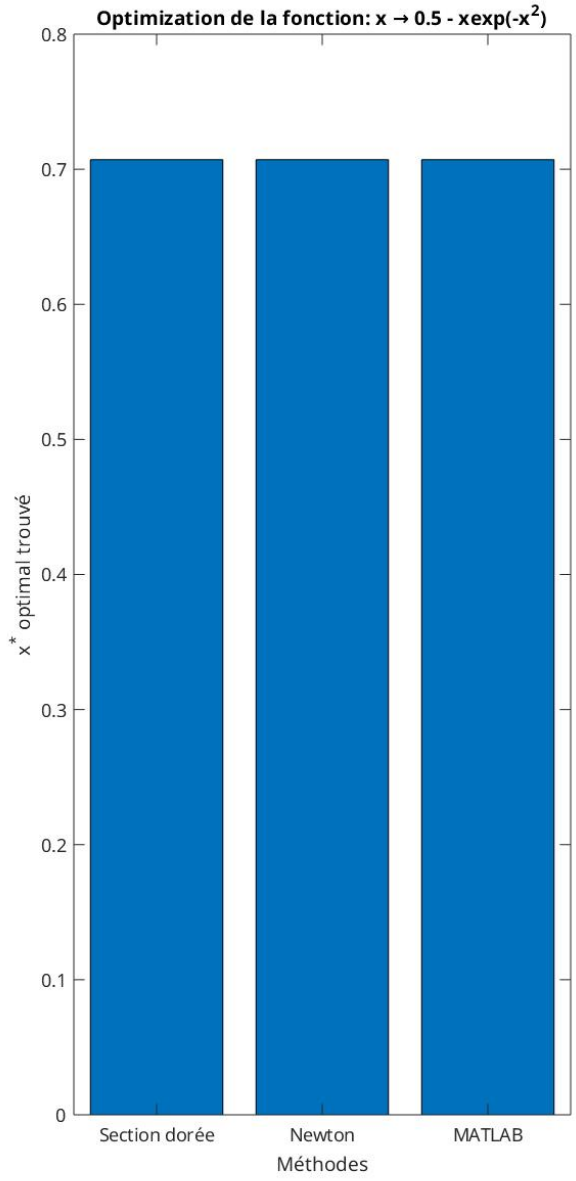


FIGURE 1 – Résultats d'optimisation sur $x \mapsto \frac{1}{2} - x\exp(-x^2)$

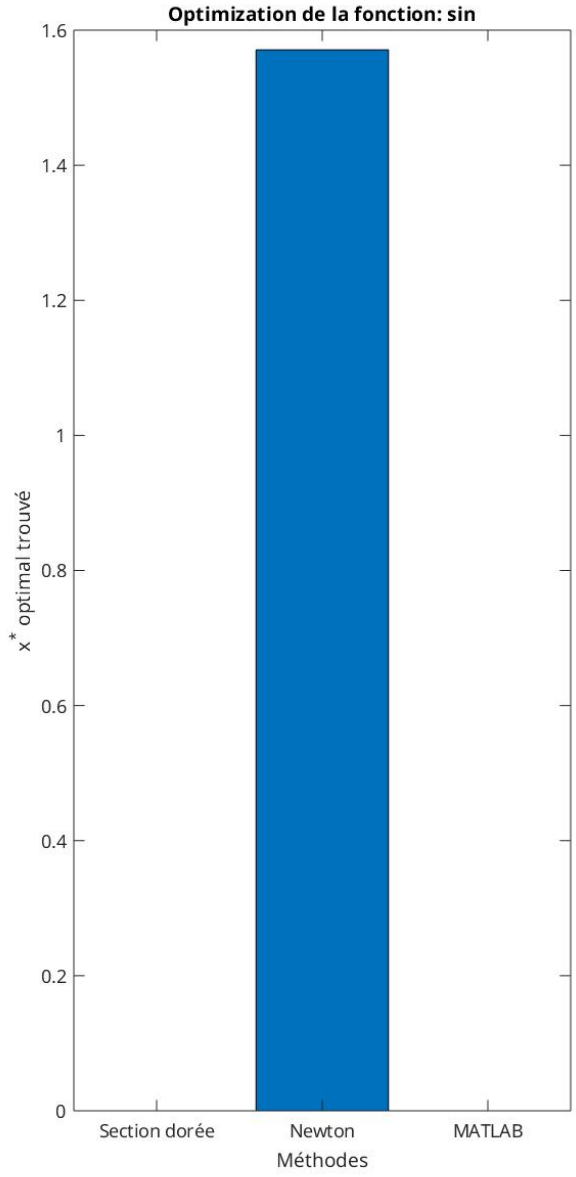


FIGURE 2 – Résultats d'optimisation sur sin

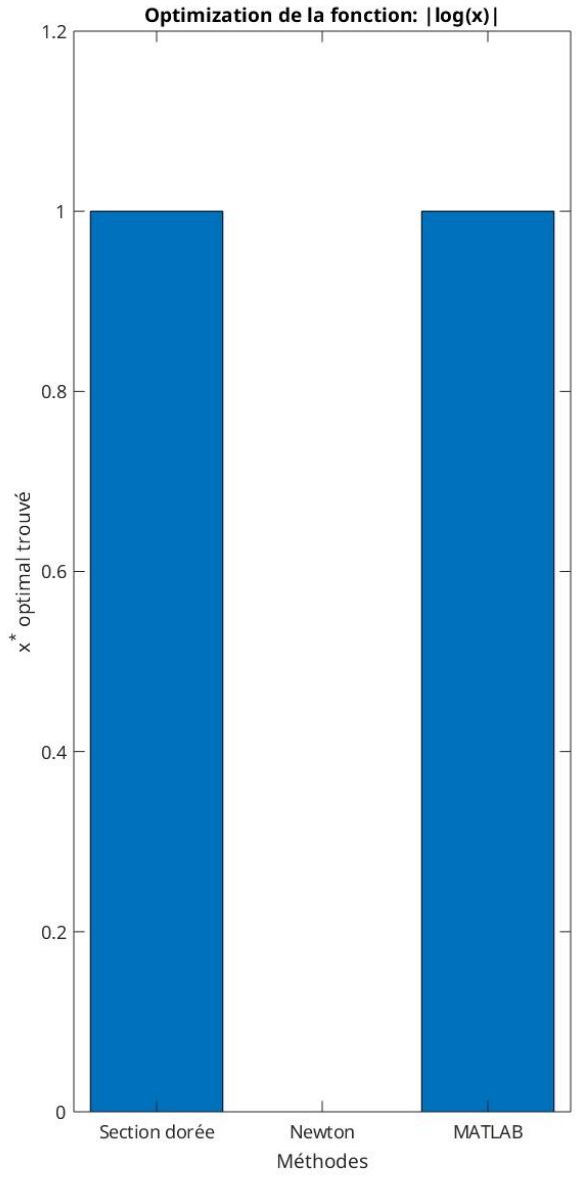


FIGURE 3 – Résultats d'optimisation sur $|\cdot| \circ \log$

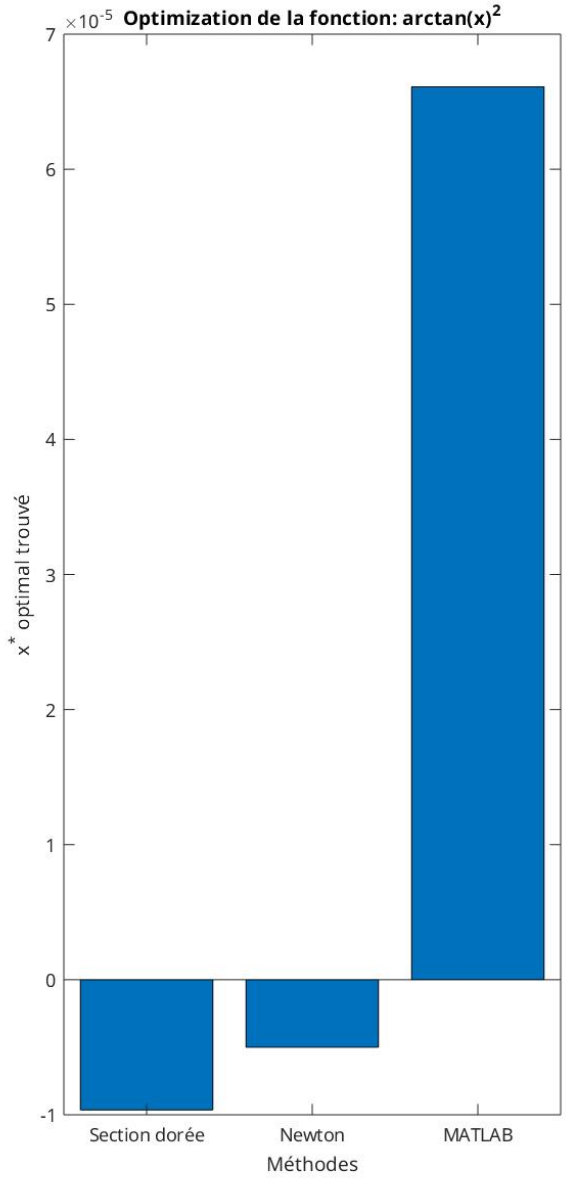


FIGURE 4 – Résultats d'optimisation sur $x \mapsto \arctan(x)^2$

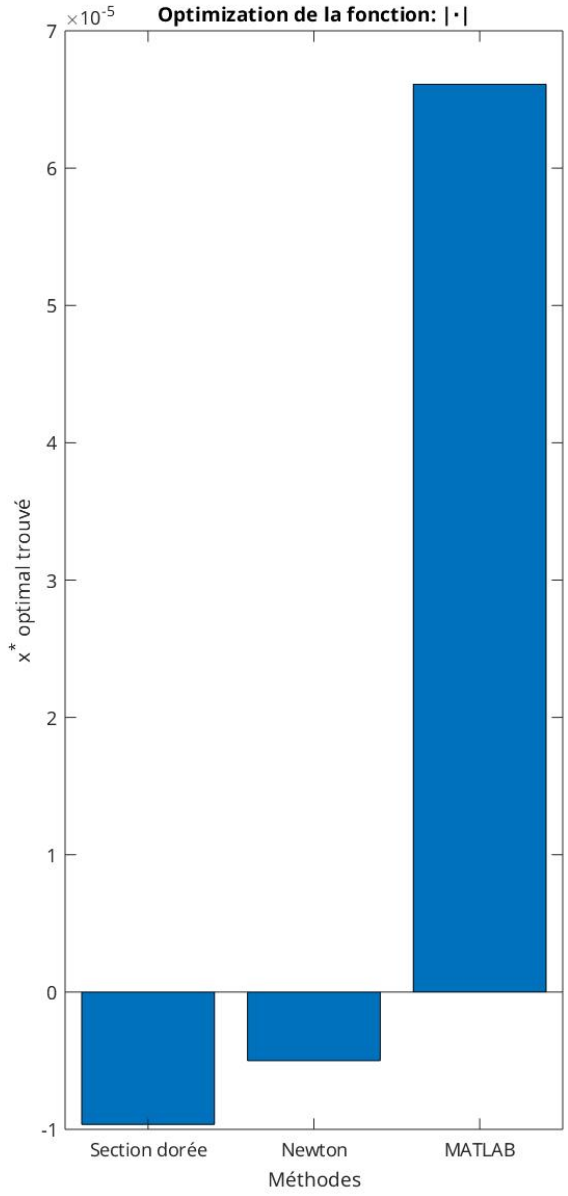


FIGURE 5 – Résultats d'optimisation sur $|\cdot|$

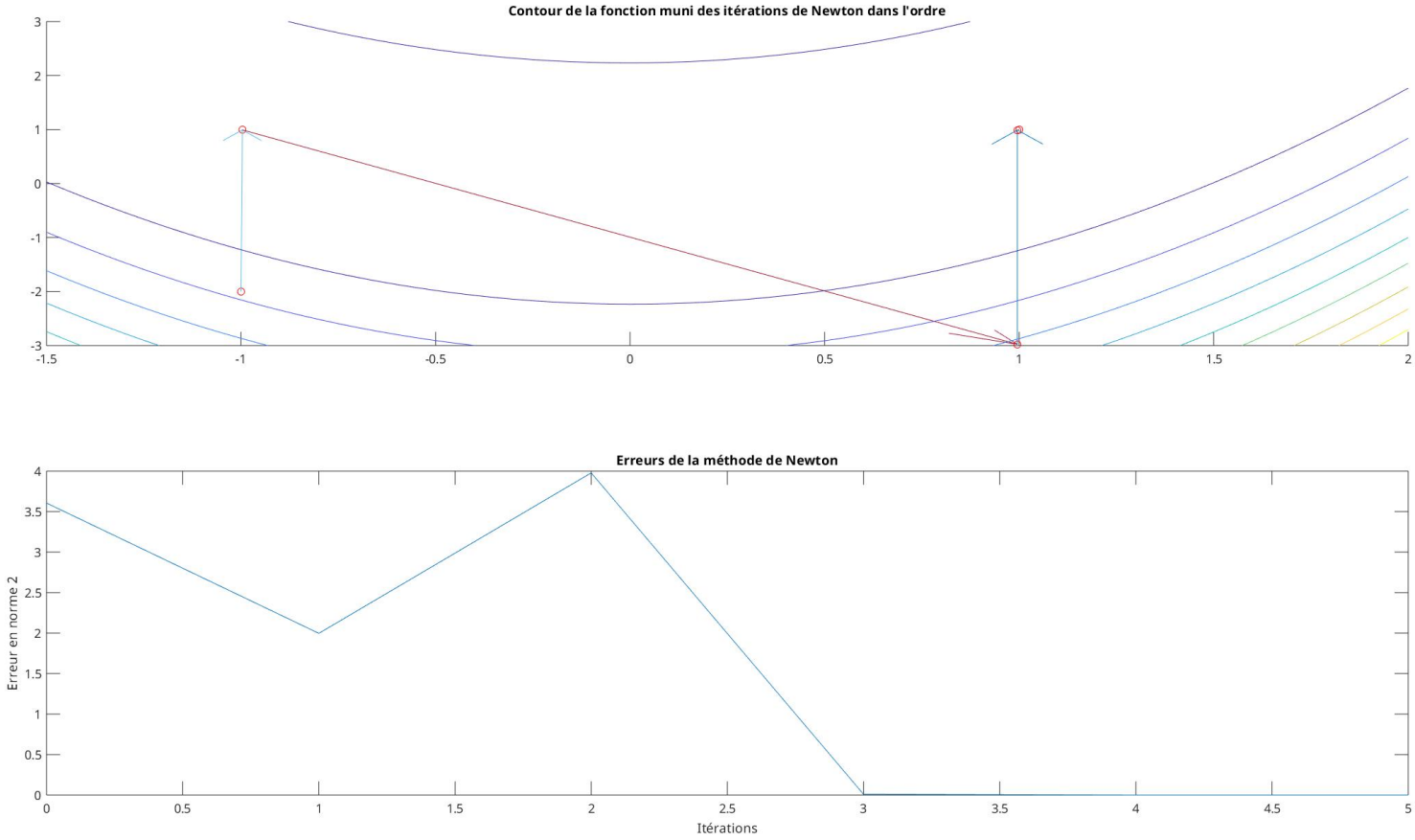
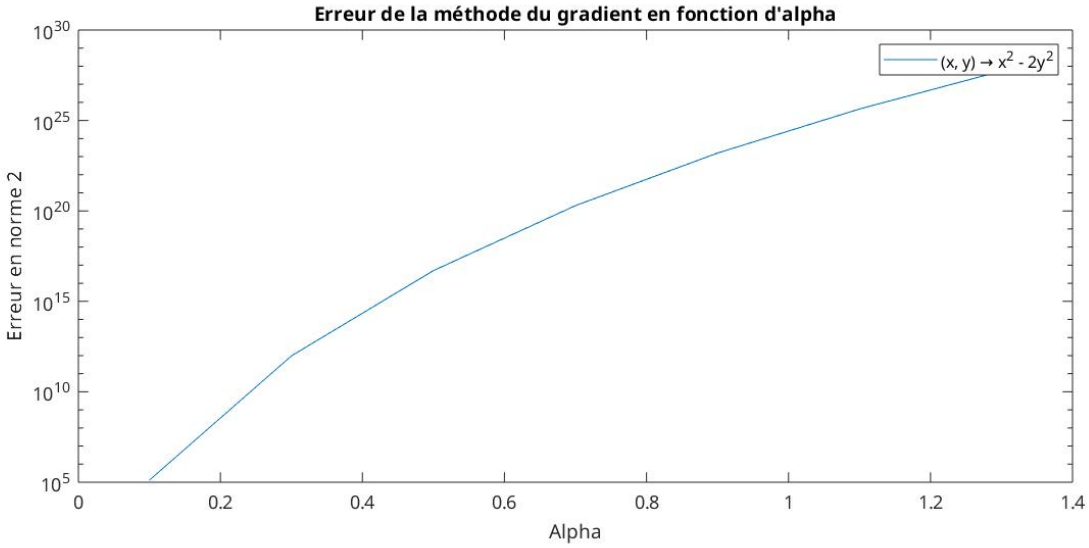


FIGURE 6 – Contours et erreur en norme de la méthode du gradient sur la fonction de Rosenbrock



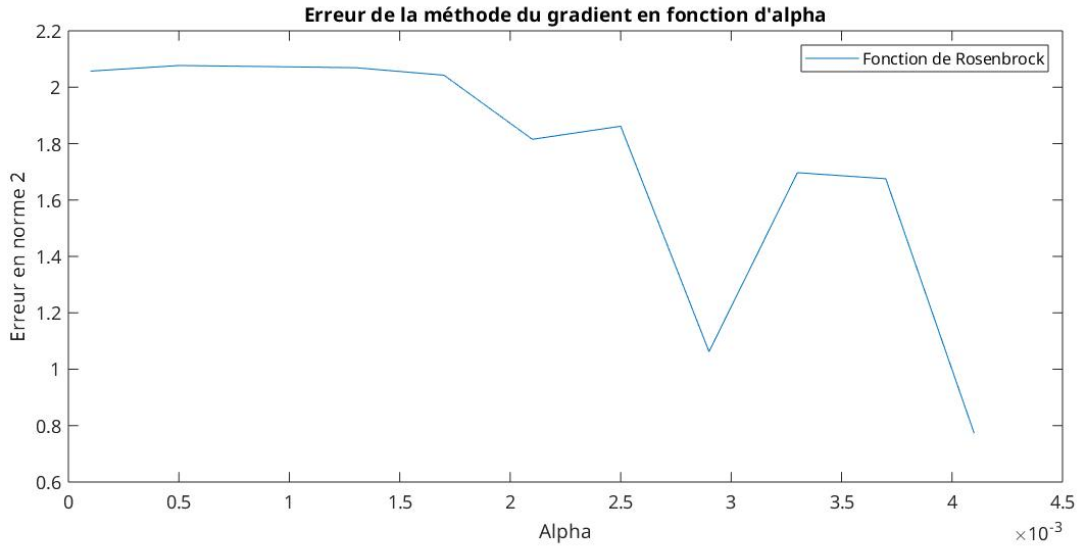


FIGURE 7 – Méthode du gradient pour Rosenbrock et la fonction f avec des itérations maximum entre 20 et 35.