TP optimisation

June 15, 2024

TP Optimisation continue.

TP 1

$$f_1(x) = \frac{1}{2}\langle Ax, x\rangle - \langle b, x\rangle$$

où $X \in \mathbb{R}^n$, A est une matrice SDP et $b \in \mathbb{R}^n$.

Dans la suite, on prend la matrice A et le vecteur b suivants pour vérifier nos algorithmes :

$$A = \begin{pmatrix} 10 & -4 \\ -4 & 4 \end{pmatrix} \quad \text{et} \quad b = \begin{pmatrix} 1 \\ 5 \end{pmatrix}$$

[22]:
```
A=[10 -4; -4 4];
eig(A)
```

```
ans =

    2
   12
```

Les valeurs propres de A sont positives donc A est SDP.

1 La méthode de gradient à pas fixe pour f_1

[1]:
```
function [xmin, fmin, indica] = gradientDescentFixedStep(A, b, x0, rho, epsilon,␣
↪Nmax)
    % Initialisation des variables
    x = x0;
    k = 0;
    grad = A * x - b;
    path = x; % Pour stocker le chemin suivi par l'algorithme

    % Boucle de gradient à pas fixe
    while (norm(grad) > epsilon) && (k <= Nmax)
        x = x - rho * grad;
        grad = A * x - b;
        k = k + 1;
        path = [path, x]; % Ajouter la nouvelle position à la trajectoire
    end

    % Vérification de la convergence
    if norm(grad) < epsilon
        xmin = x;
        fmin = 0.5 * (x' * A * x) - (b' * x);
        indica = 1;
        disp('L''algorithme converge.');
        disp('Le point de minumun :' );
        disp(xmin);
        disp('la valeur de minmale de la fonction :' );
        disp(fmin);
    else
        xmin = x;
        fmin = 0.5 * (x' * A * x) - (b' * x);
        indica = 0;
        disp('L''algorithme diverge.');
    end

    % Représentation graphique de la fonction et du chemin
    plotQuadraticFunctionAndPath(A, b, path);
end
```

```
warning: using the gnuplot graphics toolkit is discouraged

The gnuplot graphics toolkit is not actively maintained and has a number
of limitations that are ulikely to be fixed.  Communication with gnuplot
uses a one-directional pipe and limited information is passed back to the
Octave interpreter so most changes made interactively in the plot window
will not be reflected in the graphics properties managed by Octave.  For
example, if the plot window is closed with a mouse click, Octave will not
be notified and will not update it's internal list of open figure windows.
We recommend using the qt toolkit instead.
```

[2]:
```
function plotQuadraticFunctionAndPath(A, b, path)
    % Génération d'une grille de points pour la représentation de la fonction
    [X, Y] = meshgrid(-3:0.1:3, -3:0.1:3);
    Z = 0.5 * (A(1,1)*X.^2 + (A(1,2) + A(2,1))*X.*Y + A(2,2)*Y.^2) - b(1)*X -
 ↪b(2)*Y;

    % Affichage de la surface de la fonction quadratique
    figure;
    contour(X, Y, Z, 100); % Contour plot for better visualization
    hold on;
    % Affichage du chemin suivi par l'algorithme
    plot(path(1, :), path(2, :), 'r-o', 'LineWidth', 2, 'MarkerSize', 5);
    title('Descente de Gradient sur la Fonction Quadratique');
    xlabel('x_1');
    ylabel('x_2');
    legend('Courbes de niveau', 'Chemin de descente');
    hold off;
end
```

Les valeurs propres de A sont positives alos A est SDP.

Calcul du point de minumun à la main.

$$\begin{aligned}\nabla f_1(x) &= Ax - b \\ &= \begin{pmatrix} 10 & -4 \\ -4 & 4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} - \begin{pmatrix} 1 \\ 5 \end{pmatrix} \\ &= \begin{pmatrix} 10x_1 - 4x_2 - 1 \\ -4x_1 + 4x_2 - 5 \end{pmatrix}\end{aligned}$$

Le point de minumun vérifie l'equation d'Euler.

$$\nabla f_1(x) = 0$$

Résolvons le système d'équations suivant :

$$\begin{cases} 10x_1 - 4x_2 - 1 = 0 \\ -4x_1 + 4x_2 - 5 = 0 \end{cases}$$

Ce qui équivaut à :

$$\begin{cases} 10x_1 - 4x_2 = 1 \\ -4x_1 + 4x_2 = 5 \end{cases}$$

En sommant les deux équations, on obtient :

$$6x_1 = 6$$
$$x_1 = 1$$

Ensuite, en substituant $x_1 = 1$ dans la deuxième équation, on obtient :

$$4x_2 = 9$$
$$x_2 = \frac{9}{4} = 2,25$$

$$\begin{cases} x_1 = 1 \\ x_2 = 2,25 \end{cases}$$

Par conséquent, le point de minimum est $X_{\min} = \begin{pmatrix} 1 \\ 2.25 \end{pmatrix}$.

Determinons Le point de minimun à partir de l'algorithme.

On prend $X_0 = \begin{pmatrix} -2 \\ -3 \end{pmatrix}$.

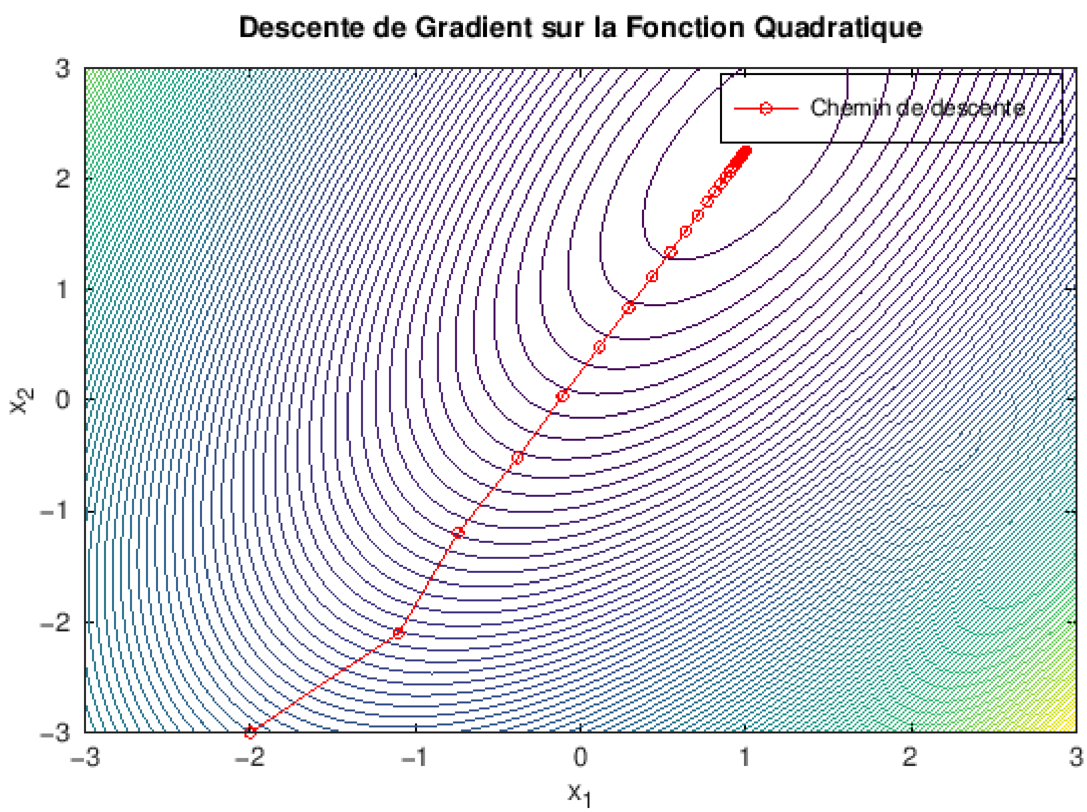
$\rho = 0.1 \quad \text{et} \quad \epsilon = 1\text{e}{-6}$

[6]:
```
b=[1 ; 5]
```

```
b =

   1
   5
```

[4]:
```
x0 = [-2; -3];
rho = 0.1;
epsilon = 1e-6;
Nmax = 1000;
[xmin, fmin, indica] = gradientDescentFixedStep(A, b, x0, rho, epsilon, Nmax);
```

```
b =

   1
   5

L'algorithme converge.
Le point de minumun :
   1.0000
   2.2500
la valeur de minmale de la fonction :
-6.1250
```



la ligne rouge correspond au chemin de descende de l'agorithme pour approcher le point de minimum

Donc l'algorithme converge bien vers la solution minimal théorique $X_{\min} = \begin{pmatrix} 1 \\ 2.25 \end{pmatrix}$.

2

La méthode de gradient à pas optimal pour f_1

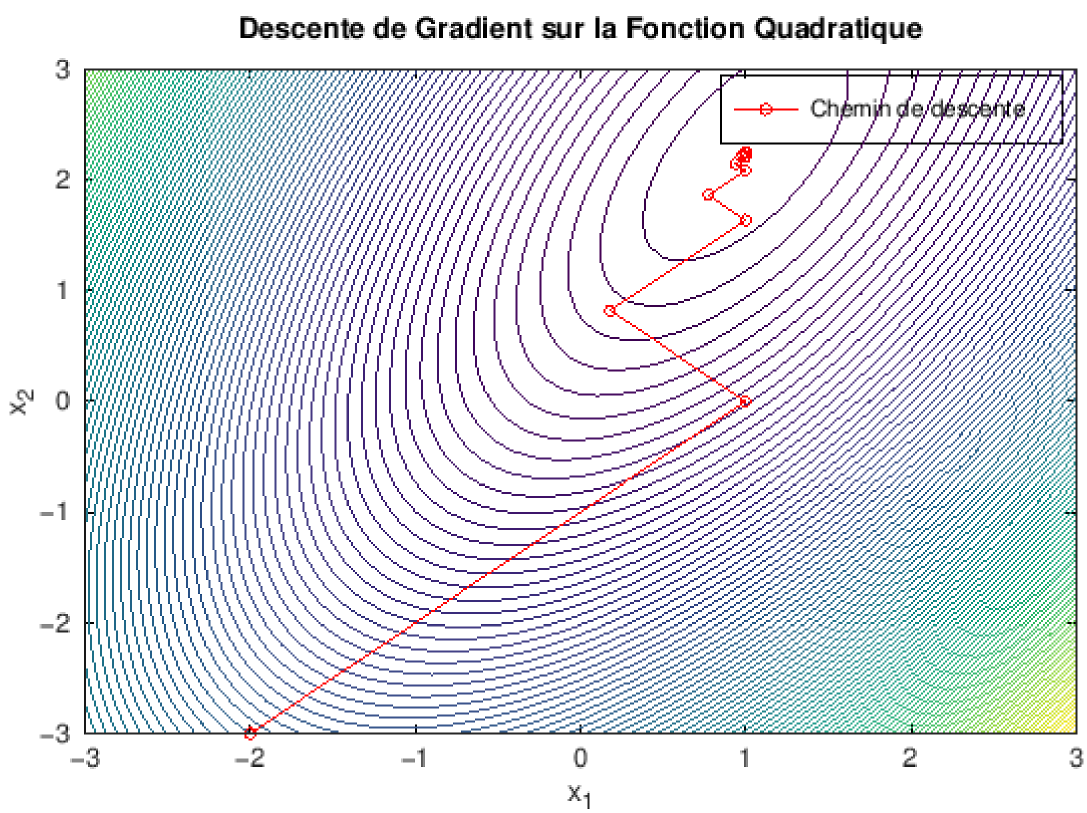
[5]:
```
function [xmin, fmin, indica] = gradientDescentOptimalStep(A, b, x0, epsilon,␣
↪Nmax)
% Initialisation des variables
x = x0;
k = 0;
grad = A * x - b;
path = x; % Pour stocker le chemin suivi par l'algorithme

% Boucle de gradient à pas optimal
while (norm(grad) > epsilon) && (k <= Nmax)
% Calcul du pas optimal
rho_optimal = (grad' * grad) / (grad' * A * grad);
% Mise à jour de x
x = x - rho_optimal * grad;
% Mise à jour de grad
grad = A * x - b;
% Mise à jour du compteur d'itérations
k = k + 1;
% Ajouter la nouvelle position à la trajectoire
path = [path, x];
end

% Vérification de la convergence
if norm(grad) < epsilon
xmin = x;
fmin = 0.5 * (x' * A * x) - (b' * x);
indica = 1;
disp('L''algorithme converge.');
disp('Le point de minumun :' );
disp(xmin);
disp('la valeur de minmale de la fonction :' );
disp(fmin);
else
xmin = x;
fmin = 0.5 * (x' * A * x) - (b' * x);
indica = 0;
disp('L''algorithme diverge.');
end

% Représentation graphique de la fonction et du chemin
plotQuadraticFunctionAndPath(A, b, path);
end
```

[6]:
```
function plotQuadraticFunctionAndPath(A, b, path)
% Génération d'une grille de points pour la représentation de la fonction
[X, Y] = meshgrid(-3:0.1:3, -3:0.1:3);
Z = 0.5 * (A(1,1)*X.^2 + (A(1,2) + A(2,1))*X.*Y + A(2,2)*Y.^2) - b(1)*X - b(2)*Y;
```

```
% Affichage de la surface de la fonction quadratique
figure;
contour(X, Y, Z, 100); % Contour plot for better visualization
hold on;
% Affichage du chemin suivi par l'algorithme
plot(path(1, :), path(2, :), 'r-o', 'LineWidth', 2, 'MarkerSize', 5);
title('Descente de Gradient sur la Fonction Quadratique');
xlabel('x_1');
ylabel('x_2');
legend('Courbes de niveau', 'Chemin de descente');
hold off;
end
```

[7]:
```
% Appel de la fonction gradientDescentOptimalStep
[xmin, fmin, indica] = gradientDescentOptimalStep(A, b, x0, epsilon, Nmax);
```

```
L'algorithme converge.
Le point de minumun :
   1.0000
   2.2500
la valeur de minmale de la fonction :
-6.1250
```



La courbe rouge représente le chemin emprunté par l'algorithme de descente de gradient pour approcher le point

On constate que l'algorithme à pas optimal converge plus rapidement vers le point de minimum que l'algorithme

3

La methode de gradient utilisant la règle d'Armijo pour f_1

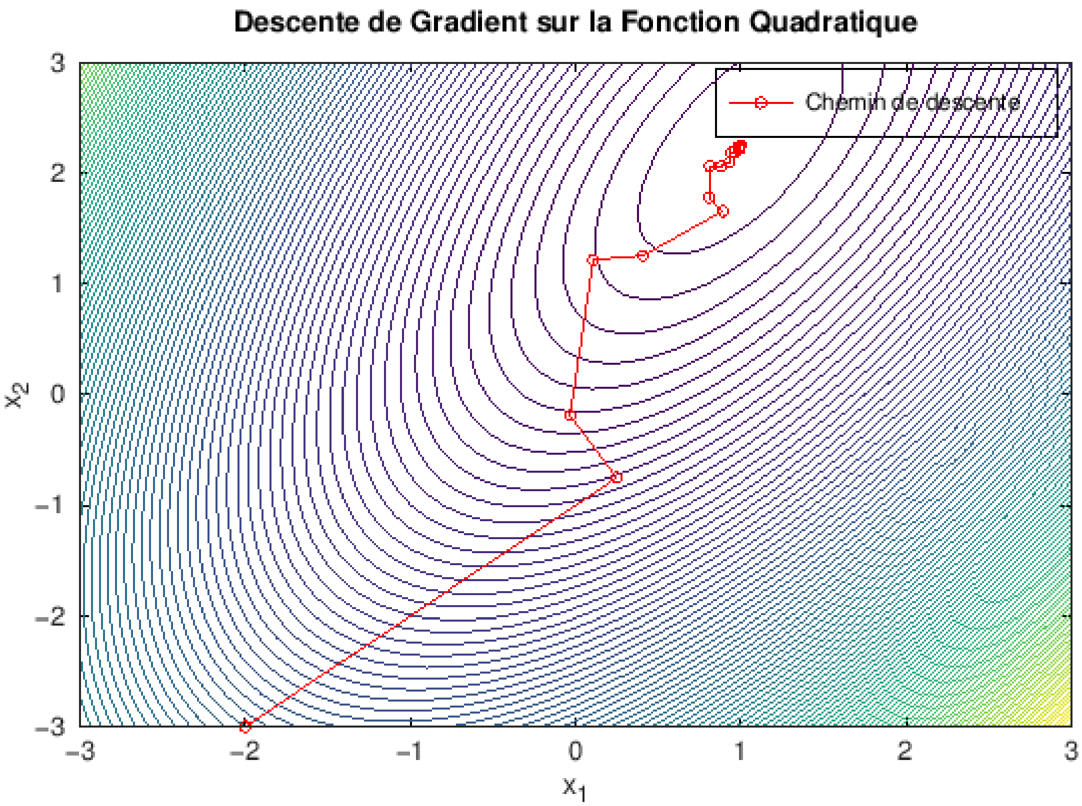
[14]:
```
function [xmin, fmin, indica] = gradientDescentArmijo(A, b, x0, epsilon, Nmax)
% Initialisation des variables
x = x0;
k = 0;
grad = A * x - b;
path = x; % Pour stocker le chemin suivi par l'algorithme
alpha = 0.5; % Paramètre de décroissance pour la règle d'Armijo
beta = 0.5; % Paramètre de réduction du pas

% Boucle de gradient à pas optimal avec règle d'Armijo
while (norm(grad) > epsilon) && (k <= Nmax)
% Calcul du pas optimal
rho = 1; % Valeur initiale du pas
while 0.5*(x - rho * grad)'*A*(x - rho * grad)- (b' * (x - rho * grad))> 0.5 *
 ↪(x' * A * x) - (b' * x)- alpha * rho * norm(grad)^2
rho = beta * rho; % Réduire le pas
end
% Mise à jour de x
x = x - rho * grad;
% Mise à jour de grad
grad = A * x - b;
% Mise à jour du compteur d'itérations
k = k + 1;
% Ajouter la nouvelle position à la trajectoire
path = [path, x];
end

% Vérification de la convergence
if norm(grad) < epsilon
xmin = x;
fmin = 0.5 * (x' * A * x) - (b' * x);
indica = 1;
disp('L''algorithme converge.');
disp('Le point de minumun :' );
```

```
disp(xmin);
disp('la valeur de minmale de la fonction :' );
disp(fmin);
else
xmin = x;
fmin = 0.5 * (x' * A * x) - (b' * x);
indica = 0;
disp('L''algorithme diverge.');
end

% Représentation graphique de la fonction et du chemin
plotQuadraticFunctionAndPath(A, b, path);
end
```

[9]:
```
function plotQuadraticFunctionAndPath(A, b, path)
% Génération d'une grille de points pour la représentation de la fonction
[X, Y] = meshgrid(-3:0.1:3, -3:0.1:3);
Z = 0.5 * (A(1,1)*X.^2 + (A(1,2) + A(2,1))*X.*Y + A(2,2)*Y.^2) - b(1)*X - b(2)*Y;

% Affichage de la surface de la fonction quadratique
figure;
contour(X, Y, Z, 100); % Contour plot for better visualization
hold on;
% Affichage du chemin suivi par l'algorithme
plot(path(1, :), path(2, :), 'r-o', 'LineWidth', 2, 'MarkerSize', 5);
title('Descente de Gradient sur la Fonction Quadratique');
xlabel('x_1');
ylabel('x_2');
legend('Courbes de niveau', 'Chemin de descente');
hold off;
end
```

[10]:
```
% Appel de la fonction gradientDescentOptimalStep
[xmin, fmin, indica] =gradientDescentArmijo(A, b, x0, epsilon, Nmax);
```

```
L'algorithme converge.
Le point de minumun :
   1.0000
   2.2500
la valeur de minmale de la fonction :
-6.1250
```



[ ]:

La fonction f_2 est définie pour tout $x \in \mathbb{R}^n$ par :

$$f_2(x) = \frac{1}{2}\langle Ax, x\rangle - \langle b, x\rangle + g(x)$$

où A est une matrice carrée d'ordre n symétrique définie positive, $b \in \mathbb{R}^n$ et $g : \mathbb{R}^n \to \mathbb{R}$ une fonction de classe C^2

On prend la fonction g définie pour tout $x \in \mathbb{R}^n$ par :

$$g(x) = e^{(\lambda_1 x_1)} + e^{(\lambda_2 x_2)} + ... + e^{(\lambda_n x_n)} = \sum_{i=1}^{n} e^{(\lambda_i x_i)}$$

où $\lambda_1, \ldots, \lambda_n \in \mathbb{R}$ sont des paramètres donnés.

La fonction f_2 est la somme d'une fonction quadratique et d'une fonction convexe g.

La partie quadratique est fortement convexe car la matrice A est SDP.

Par conséquent, f_2 est également fortement convexe.

4

La méthode de gradient à pas fixe pour f_2

Prénons l'exemple le plus simple dans $\lambda_1, \lambda_2 \in \mathbb{R}^2$ tel que $\lambda_1 = \lambda_2 = 0$

Alors $g(x) = e^{(\lambda_1 x_1)} + e^{(\lambda_2 x_2)} = e^0 + e^0 = 2$

C'est qui donne

$$f_2(x) = \frac{1}{2}\langle Ax, x\rangle - \langle b, x\rangle + g(x) = f_1(x) + 2$$

$$\begin{aligned} \nabla f_2(x) &= \nabla f_1(x) \\ &= Ax - b \end{aligned}$$

Calcul du point de minumun à la main.

$$\begin{aligned} \nabla f_2(x) &= Ax - b \\ &= \begin{pmatrix} 10 & -4 \\ -4 & 4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} - \begin{pmatrix} 1 \\ 5 \end{pmatrix} \\ &= \begin{pmatrix} 10x_1 - 4x_2 - 1 \\ -4x_1 + 4x_2 - 5 \end{pmatrix} \end{aligned}$$

Le point de minumun vérifie l'equation d'Euler.

$$\nabla f_2(x) = 0$$

[11]:
```
function [xmin, fmin, indica] = gradientDescentFixedStep(A, b, lambda, x0, rho,␣
↪epsilon, Nmax)
    % Initialisation des variables
```

```
    x = x0;
    k = 0;
    grad = A * x - b + lambda .* exp(lambda .* x);
    path = x; % Pour stocker le chemin suivi par l'algorithme

    % Boucle de gradient à pas fixe
    while (norm(grad) > epsilon) && (k <= Nmax)
        x = x - rho * grad;
        grad = A * x - b + lambda .* exp(lambda .* x);
        k = k + 1;
        path = [path, x]; % Ajouter la nouvelle position à la trajectoire
    end

    % Vérification de la convergence
    if norm(grad) < epsilon
        xmin = x;
        fmin = 0.5 * (x' * A * x) - (b' * x) + sum(exp(lambda .* x));
        indica = 1;
        disp('L''algorithme converge.');
        disp('Le point de minimum :' );
        disp(xmin);
        disp('La valeur minimale de la fonction :' );
        disp(fmin);
    else
        xmin = x;
        fmin = 0.5 * (x' * A * x) - (b' * x) + sum(exp(lambda .* x));
        indica = 0;
        disp('L''algorithme diverge.');
    end

    % Représentation graphique de la fonction et du chemin
    plotQuadraticFunctionAndPath(A, b, lambda, path);
end
```

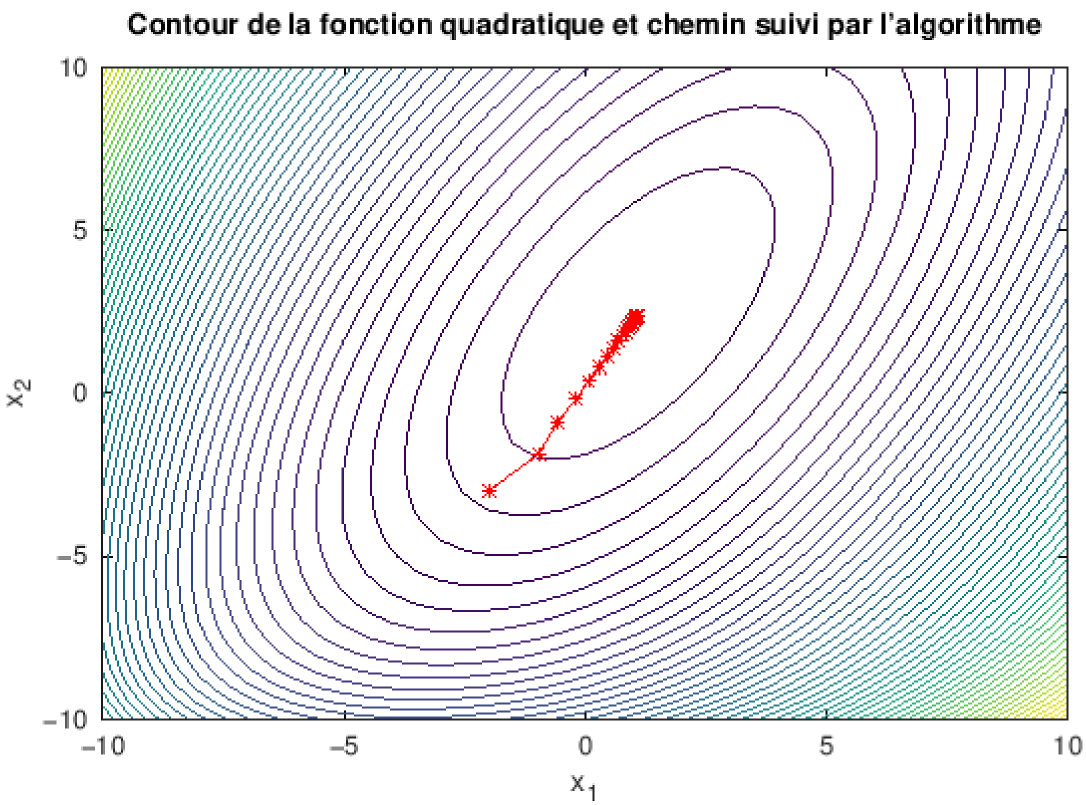
[12]:

```
function plotQuadraticFunctionAndPath(A, b, lambda, path)
  [X,Y] = meshgrid(-10:0.5:10, -10:0.5:10);
    Z = 0.5 * (A(1,1) * X.^2 + 2 * A(1,2) * X .* Y + A(2,2) * Y.^2) - (b(1) * X␣
 ↪+ b(2) * Y) + exp(lambda(1) * X) + exp(lambda(2) * Y);

    figure;
    contour(X, Y, Z, 50);
    hold on;
    plot(path(1, :), path(2, :), 'r*-');
    title('Contour de la fonction quadratique et chemin suivi par␣
 ↪l''algorithme');
    xlabel('x_1');
    ylabel('x_2');
```

```
    hold off;
end
```

[13]:
```
% Appel de la fonction
lambda=[-0.5; -0.5];
Nmax=10000;
[xmin, fmin, indica] =gradientDescentFixedStep(A, b, lambda, x0, rho, epsilon,
 ↪Nmax);
```

```
L'algorithme converge.
Le point de minimum :
   1.0743
   2.3626
La valeur minimale de la fonction :
-5.2142
```



5 La méthode de gradient à pas optimal pour f_2

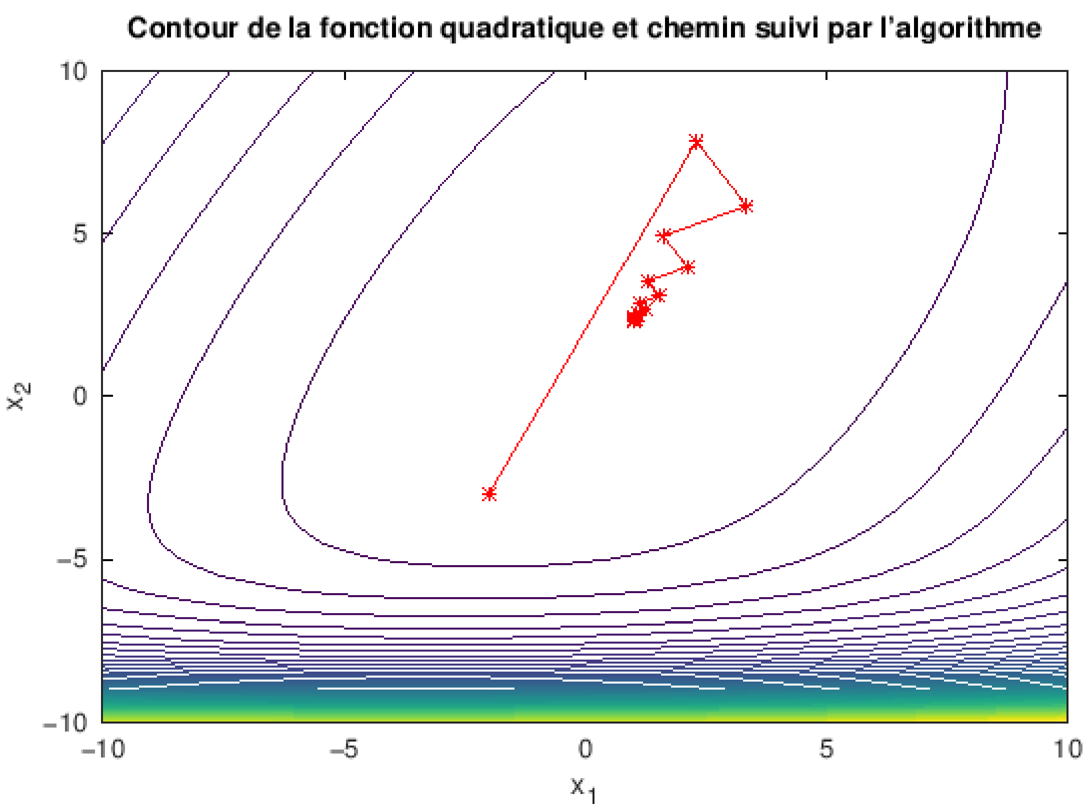
[24]:
```
function [xmin, fmin, indica] = gradientDescent_OptimalStep(A, b, lambda, x0,␣
↪epsilon, Nmax)
    % Initialisation des variables
    x = x0;
    k = 0;
    grad = A * x - b + lambda .* exp(lambda .* x);
    path = x; % Pour stocker le chemin suivi par l'algorithme

    % Boucle de gradient à pas optimal
    while (norm(grad) > epsilon) && (k <= Nmax)
        % Calcul du pas optimal
        rho_optimal = (grad' * grad) / (grad' * A * grad);
        % Mise à jour de x
        x = x - rho_optimal * grad;
        % Mise à jour de grad
        grad = A * x - b + lambda .* exp(lambda .* x);
        % Mise à jour du compteur d'itérations
        k = k + 1;
        % Ajouter la nouvelle position à la trajectoire
        path = [path, x];
    end

    % Vérification de la convergence
    if norm(grad) < epsilon
        xmin = x;
        fmin = 0.5 * (x' * A * x) - (b' * x) + sum(exp(lambda .* x));
        indica = 1;
        disp('L''algorithme converge.');
        disp('Le point de minimum :');
        disp(xmin);
        disp('La valeur minimale de la fonction :');
        disp(fmin);
    else
        xmin = x;
        fmin = 0.5 * (x' * A * x) - (b' * x) + sum(exp(lambda .* x));
        indica = 0;
        disp('L''algorithme diverge.');
    end

    % Représentation graphique de la fonction et du chemin
    plotQuadraticFunctionAndPath(A, b, lambda, path);
end
```

[95]:
```
function plotQuadraticFunctionAndPath(A, b, lambda, path)
    [X, Y] = meshgrid(-10:0.5:10, -10:0.5:10);
    Z = 0.5 * (A(1,1) * X.^2 + 2 * A(1,2) * X .* Y + A(2,2) * Y.^2) - (b(1) * X␣
↪+ b(2) * Y) + exp(lambda(1) * X) + exp(lambda(2) * Y);

    figure;
```

```
    contour(X, Y, Z, 50);
    hold on;
    plot(path(1, :), path(2, :), 'r*-');
    title('Contour de la fonction quadratique et chemin suivi par␣
 ↪l''algorithme');
    xlabel('x_1');
    ylabel('x_2');
    hold off;
end
```

[104]:
```
lambda=[0.1; -0.9];
Nmax=100000;
[xmin, fmin, indica] = gradientDescent_OptimalStep(A, b, lambda, x0, epsilon,␣
 ↪Nmax);
```

```
L'algorithme converge.
Le point de minimum :
   1.0009
   2.2798
La valeur minimale de la fonction :
-4.8896
```



6

La methode de gradient utilisant la règle d'Armijo pour f_2

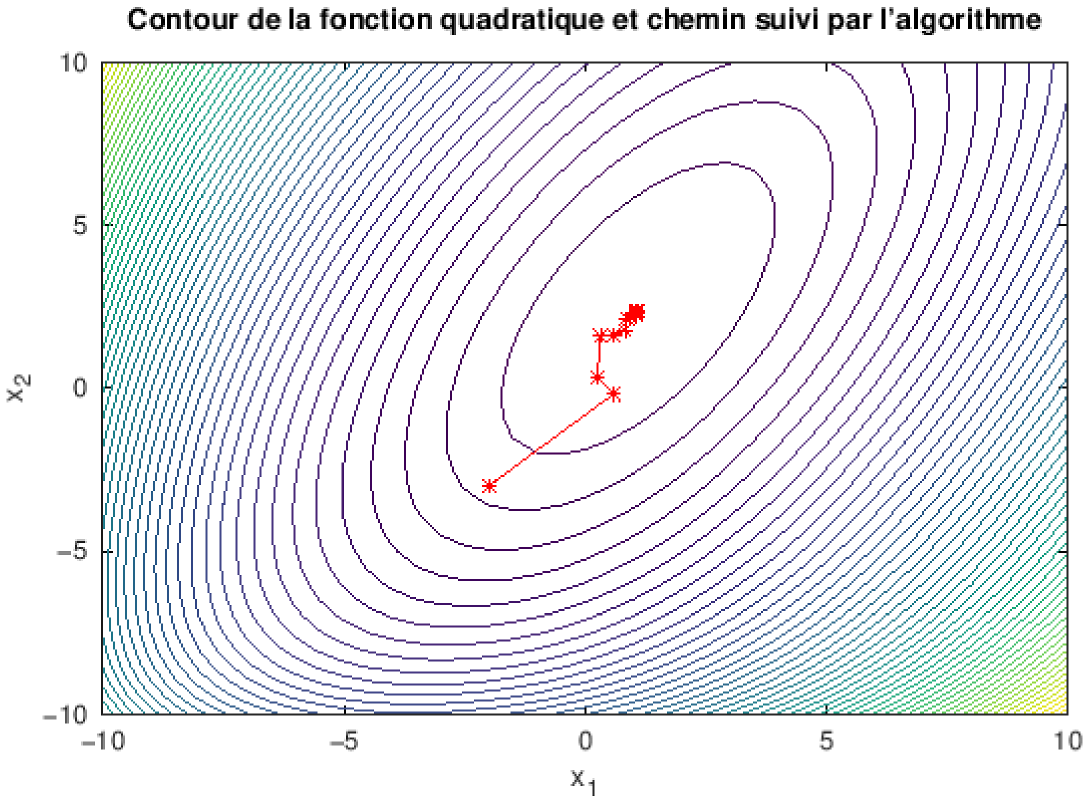
[23]:
```
function [xmin, fmin, indica] = gradientDescent_Armijo(A, b, lambda, x0,␣
↪epsilon, Nmax)
    % Initialisation des variables
    x = x0;
    k = 0;
    grad = A * x - b + lambda .* exp(lambda .* x);
    path = x; % Pour stocker le chemin suivi par l'algorithme
    alpha = 0.5; % Paramètre de décroissance pour la règle d'Armijo
    beta = 0.5; % Paramètre de réduction du pas

    % Boucle de gradient à pas variable avec règle d'Armijo
    while (norm(grad) > epsilon) && (k <= Nmax)
        % Calcul du pas optimal
        rho = 1; % Valeur initiale du pas
        while 0.5 * (x - rho * grad)' * A * (x - rho * grad) - (b' * (x - rho *␣
↪grad)) + sum(exp(lambda .* (x - rho * grad))) > ...
              0.5 * (x' * A * x) - (b' * x) + sum(exp(lambda .* x)) - alpha *␣
↪rho * norm(grad)^2
            rho = beta * rho; % Réduire le pas
        end
        % Mise à jour de x
        x = x - rho * grad;
        % Mise à jour de grad
        grad = A * x - b + lambda .* exp(lambda .* x);
        % Mise à jour du compteur d'itérations
        k = k + 1;
        % Ajouter la nouvelle position à la trajectoire
        path = [path, x];
    end

    % Vérification de la convergence
    if norm(grad) < epsilon
        xmin = x;
        fmin = 0.5 * (x' * A * x) - (b' * x) + sum(exp(lambda .* x));
        indica = 1;
        disp('L''algorithme converge.');
        disp('Le point de minimum :');
        disp(xmin);
        disp('La valeur minimale de la fonction :');
        disp(fmin);
    else
        xmin = x;
        fmin = 0.5 * (x' * A * x) - (b' * x) + sum(exp(lambda .* x));
```

```
        indica = 0;
        disp('L''algorithme diverge.');
    end

    % Représentation graphique de la fonction et du chemin
    plotQuadraticFunctionAndPath(A, b, lambda, path);
end
```

[90]:
```
function plotQuadraticFunctionAndPath(A, b, lambda, path)
    [X, Y] = meshgrid(-10:0.5:10, -10:0.5:10);
    Z = 0.5 * (A(1,1) * X.^2 + 2 * A(1,2) * X .* Y + A(2,2) * Y.^2) - (b(1) * X␣
 ↪+ b(2) * Y) + exp(lambda(1) * X) + exp(lambda(2) * Y);

    figure;
    contour(X, Y, Z, 50);
    hold on;
    plot(path(1, :), path(2, :), 'r*-');
    title('Contour de la fonction quadratique et chemin suivi par␣
 ↪l''algorithme');
    xlabel('x_1');
    ylabel('x_2');
    hold off;
end
```

[110]:
```
[xmin, fmin, indica] = gradientDescent_Armijo(A, b, lambda, x0, epsilon, Nmax)
```

```
L'algorithme converge.
Le point de minimum :
   1.0743
   2.3626
La valeur minimale de la fonction :
-5.2142
xmin =

   1.0743
   2.3626

fmin = -5.2142
indica = 1
```



[ ]:

```
```

La fonction f_3 est définie pour tout $u \in \mathbb{R}^n$ par :

$$f_3(u) = \frac{1}{2}\|u\|^2 + \frac{\alpha}{2}(x_n - d)^2$$

où $\alpha > 0$ est un paramètre donné, $d \in \mathbb{R}^n$ est le vecteur des états souhaités,

$x \in \mathbb{R}^n$ est le vecteur des états du système contrôlé n.

Le vecteur des états x est lié au vecteur de contrôle u par la loi d'évolution suivante :

$$\begin{cases} x_{k+1} = ax_k + bu_{k+1} & k \in \{0, \dots, n-1\} \\ x_0 \in \mathbb{R} & \text{donnée} \end{cases}$$

où $a, b \in \mathbb{R}$ sont des paramètres donnés, avec $b \neq 0$.

On choisit P tel que ses composantes satisfont la relation de récurrence rétrograde :

$$\begin{cases} P_k = aP_{k+1}, \quad k = n-1, n-2, ..., 1 \\ P_n = \alpha(x_n - d) \end{cases}$$

$$\nabla f_3(u) = u + bP$$

Le point de minimum U^* de f_3 satisfait le système d'optimalité suivant : il existe x^*, p^* in $\mathbb{R}^n$ tels que :

$$\begin{array}{ll} x^*_{k+1} = ax^*_k + bu^*_{k+1} & k = 0, 1, 2, \ldots, n-1 \\ x^*_0 = x_0 & \\ p^*_k = ap^*_{k+1} & k = n-1, n-2, \ldots, 1 \\ p^*_n = \alpha(x_n - d) & \\ u^*_k + bp^*_k = 0 & \end{array}$$

7

La méthode de gradient à pas fixe pour f_3

[10]:
```
function [xmin, fmin, indica] = gradientDescentFixedStep(f3, grad_f3, u0, rho,␣
↪epsilon, Nmax)
    % Initialisation des variables
    u = u0;
    k = 0;
    grad = grad_f3(u);
    path = u; % Pour stocker le chemin suivi par l'algorithme

    % Boucle de gradient à pas fixe
    while (norm(grad) > epsilon) && (k <= Nmax)
        u = u - rho * grad;
        grad = grad_f3(u);
        k = k + 1;
        path = [path, u]; % Ajouter la nouvelle position à la trajectoire
    end

    % Vérification de la convergence
    if norm(grad) < epsilon
        xmin = u;
        fmin = f3(u);
```

```
        indica = 1;
        disp('L''algorithme converge.');
        disp('Le point de minimum :');
        disp(xmin);
        disp('La valeur minimale de la fonction :');
        disp(fmin);
    else
        xmin = u;
        fmin = f3(u);
        indica = 0;
        disp('L''algorithme diverge.');
    end

    % Représentation graphique de la fonction et du chemin
    plotFunctionAndPath(f3, path);
end
```

[11]:
```
function [f_val] = f3(u, alpha, d, a, b, x0, n)
    x = zeros(n, 1);
    x(1) = x0;
    for k = 1:n-1
        x(k+1) = a*x(k) + b*u(k);
    end
    f_val = 0.5 * norm(u)^2 + 0.5 * alpha * (x(n) - d)^2;
end
```

[12]:
```
function [grad_val] = grad_f3(u, alpha, d, a, b, x0, n)
    x = zeros(n, 1);
    p = zeros(n, 1);
    x(1) = x0;
    for k = 1:n-1
        x(k+1) = a*x(k) + b*u(k);
    end
    p(n) = alpha * (x(n) - d);
    for k = n-1:-1:1
        p(k) = a * p(k+1);
    end
    grad_val = u + b * p;
end
```

[13]:
```
function plotFunctionAndPath(f3, path)
    % Fonction de représentation de la trajectoire
    figure;
    plot(path(1, :), path(2, :), 'r*-');
    title('Chemin suivi par l''algorithme');
    xlabel('u_1');
    ylabel('u_2');
```

```
    hold off;
end
```

Prenons un exemple simple dont on connaît la solution :

$$\text{Prendre } \alpha = a = b = d = 1,\ x_0 = \frac{1}{2},\ \text{et } n = 2.$$

Nous avons trouvé dans le TT que le point minimal est donné par :

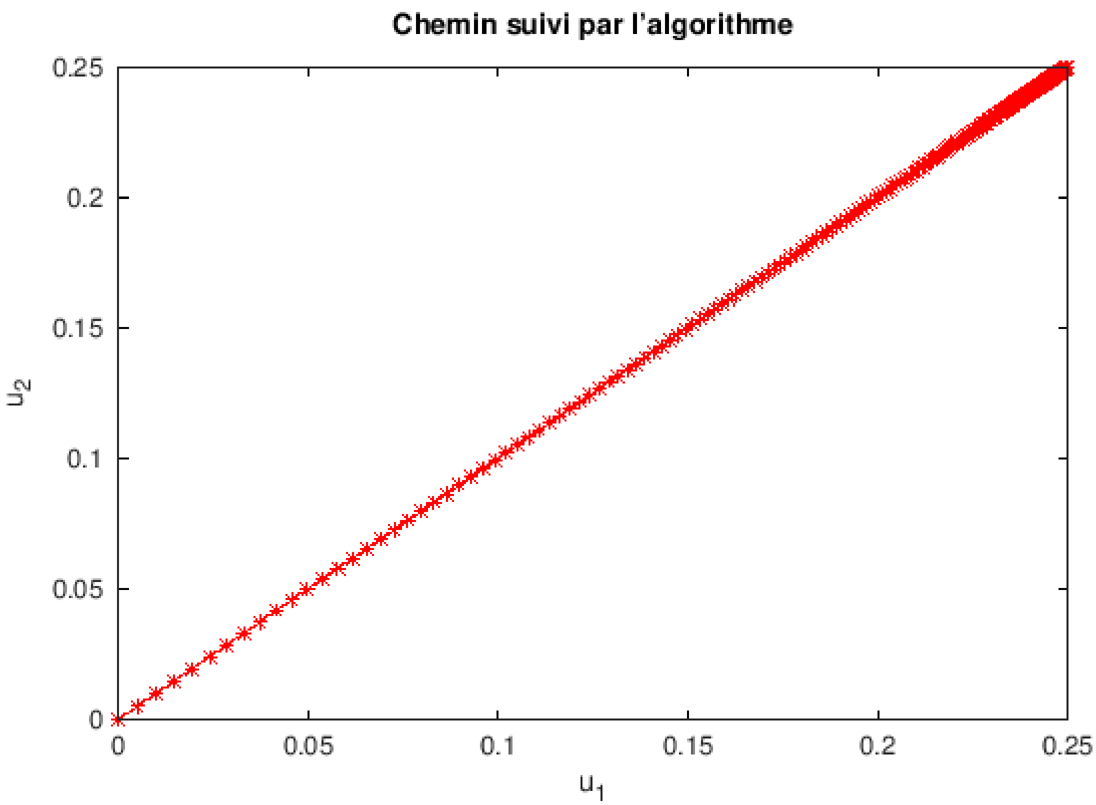
$$\begin{aligned}
U_k^* &= -b\alpha a^{n-k}\left(\frac{a^n}{1+b^2a^2\alpha}x_0 + \frac{b^2a^2\alpha\alpha}{1+b^2a^2\alpha}d - d\right)\\
U_1^* &= -1*1*1^{2-1}\left(\frac{1}{2}*\frac{1}{2}-\frac{1}{2}-1\right)\\
&= -\left(\frac{1}{4}-\frac{1}{2}\right)\\
&= \frac{1}{4}\\
U_2^* &= -1*1*1^{2-2}\left(\frac{1}{2}*\frac{1}{2}-\frac{1}{2}-1\right)\\
&= -\left(\frac{1}{4}-\frac{1}{2}\right)\\
&= \frac{1}{4}
\end{aligned}$$

$$\text{Donc, } U^* = \begin{pmatrix} U_1^* \\ U_2^* \end{pmatrix} = \begin{pmatrix} \frac{1}{4} \\ \frac{1}{4} \end{pmatrix} \text{ est le point de minimum.}$$

[20]:
```
% Exemple
alpha = 1;
d = 1;
a = 1;
b = 1;
x0 = 0.5;
n = 2;
rho = 0.01;
epsilon = 1e-6;
Nmax = 1000;

% Fonction f3 et gradient de f3
f3_func = @(u) f3(u, alpha, d, a, b, x0, n);
grad_f3_func = @(u) grad_f3(u, alpha, d, a, b, x0, n);
```

```
% Point initial
u0 = zeros(n, 1);

% Appeler l'algorithme de descente de gradient à pas fixe
[xmin, fmin, indica] = gradientDescentFixedStep(f3_func, grad_f3_func, u0, rho,
↪epsilon, Nmax);

% Afficher les résultats
disp('Indicateur de convergence (1 si converge, 0 sinon) :');
disp(indica);
```

```
L'algorithme converge.
Le point de minimum :
   0.2500
   0.2500
La valeur minimale de la fonction :
0.093750
Indicateur de convergence (1 si converge, 0 sinon) :
1
```



Donc l'algorithme converge bien vers la solution minimal théorique $U^* = \begin{pmatrix} \frac{1}{4} \\ \frac{1}{4} \end{pmatrix}$.

Prenons un exemple complexe dont on connaît pas la solution :

$$\alpha = 2$$

a=1.2

b = 0.5

d = 1

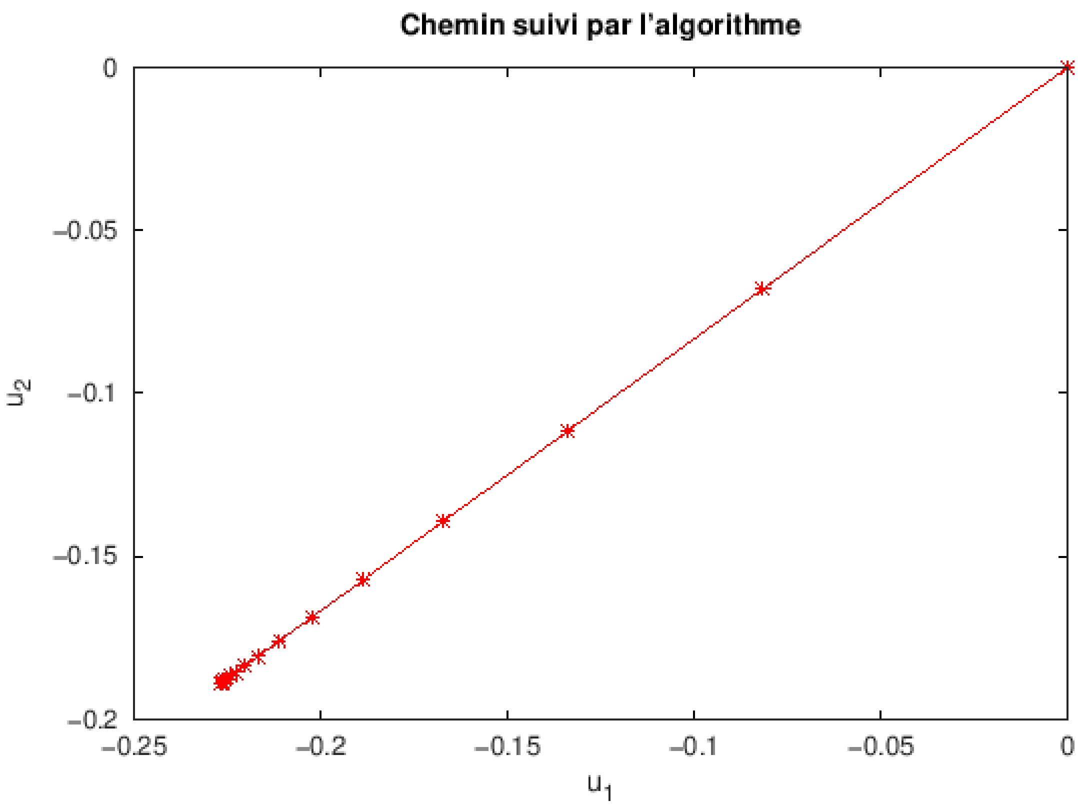
$$x_0 = \frac{1}{2}$$

$$n = 10.$$

[21]:
```
% Exemple d'utilisation
alpha = 2;
d = 1;
a = 1.2;
b = 0.5;
x0 = 0.5;
n = 10;
rho = 0.01;
epsilon = 1e-6;
Nmax = 1000;

% Fonction f3 et gradient de f3
f3_func = @(u) f3(u, alpha, d, a, b, x0, n);
grad_f3_func = @(u) grad_f3(u, alpha, d, a, b, x0, n);

% Point initial
u0 = zeros(n, 1);

% Appeler l'algorithme de descente de gradient à pas fixe
[xmin, fmin, indica] = gradientDescentFixedStep(f3_func, grad_f3_func, u0, rho,
 ↪epsilon, Nmax);

% Afficher les résultats
disp('Indicateur de convergence (1 si converge, 0 sinon) :');
disp(indica);
```

```
L'algorithme converge.
Le point de minimum :
  -0.226814
  -0.189011
  -0.157509
```

```
  -0.131258
  -0.109382
  -0.091151
  -0.075959
  -0.063300
  -0.052750
  -0.043958
La valeur minimale de la fonction :
0.083918
Indicateur de convergence (1 si converge, 0 sinon) :
1
```



L'algorithme converge bien vers la solution minimal $U^* = \begin{pmatrix} -0.226814 \\ -0.189011 \\ -0.157509 \\ -0.131258 \\ -0.109382 \\ -0.091151 \\ -0.075959 \\ -0.063300 \\ -0.052750 \\ -0.043958 \end{pmatrix}$.

8 La méthode de gradient à pas optimal pour f_3

[46]:
```
function [xmin, fmin, indica] = gradientDescentOptimalStep(f3, grad_f3, u0,␣
↪epsilon, Nmax)
    % Initialisation des variables
    u = u0;
    k = 0;
    grad = grad_f3(u);
    path = u; % Pour stocker le chemin suivi par l'algorithme

    % Boucle de gradient à pas optimal
    while (norm(grad) > epsilon) && (k <= Nmax)
        % Fonction auxiliaire pour rechercher le pas optimal
        phi = @(rho) f3(u - rho * grad);
        % Recherche de rho optimal qui minimise phi(rho)
        rho_optimal = fminbnd(phi, 0, 1); % Limites initiales pour rho

        % Mise à jour de u
        u = u - rho_optimal * grad;
        % Mise à jour du gradient
        grad = grad_f3(u);
        % Mise à jour du compteur d'itérations
        k = k + 1;
        % Ajouter la nouvelle position à la trajectoire
        path = [path, u];
    end

    % Vérification de la convergence
    if norm(grad) < epsilon
        xmin = u;
        fmin = f3(u);
        indica = 1;
        disp('L''algorithme converge.');
```

```
        disp('Le point de minimum :');
        disp(xmin);
        disp('La valeur minimale de la fonction :');
        disp(fmin);
    else
        xmin = u;
        fmin = f3(u);
        indica = 0;
        disp('L''algorithme diverge.');
    end

    % Représentation graphique de la fonction et du chemin
    plotFunctionAndPath(f3, path);
end
```

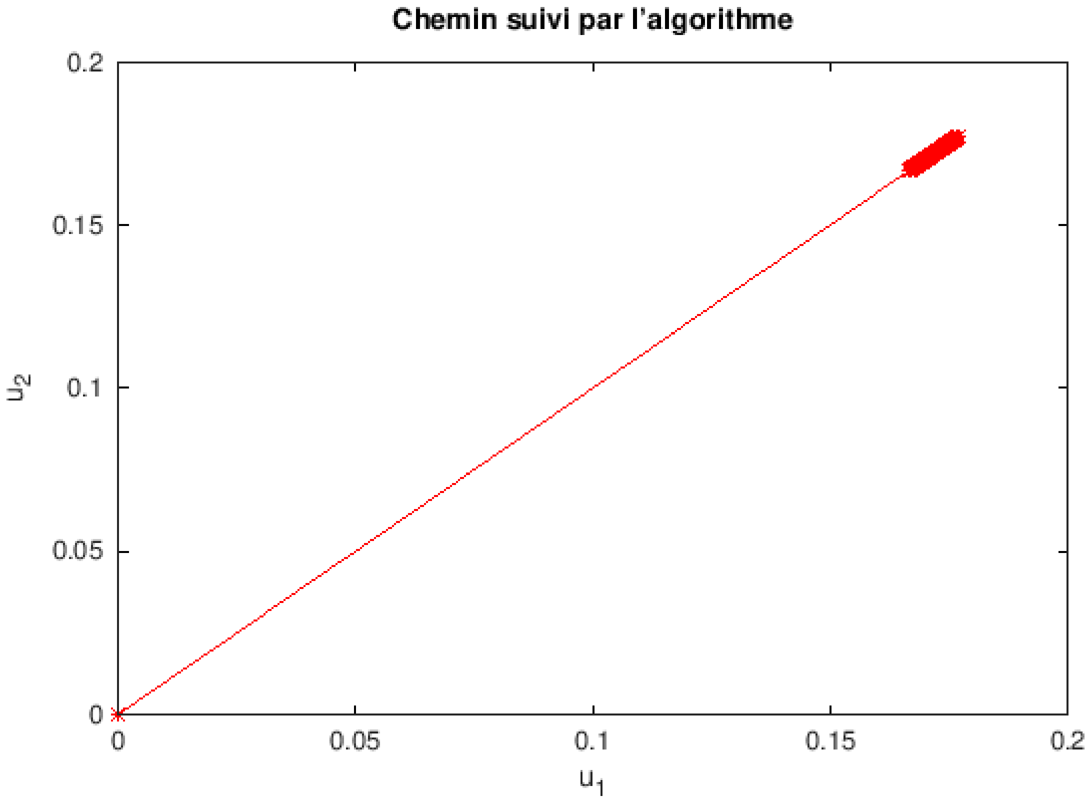
[47]:
```
function [f_val] = f3(u, alpha, d, a, b, x0, n)
    x = zeros(n, 1);
    x(1) = x0;
    for k = 1:n-1
        x(k+1) = a * x(k) + b * u(k);
    end
    f_val = 0.5 * norm(u)^2 + 0.5 * alpha * (x(n) - d)^2;
end
```

[48]:
```
function [grad_val] = grad_f3(u, alpha, d, a, b, x0, n)
    x = zeros(n, 1);
    p = zeros(n, 1);
    x(1) = x0;
    for k = 1:n-1
        x(k+1) = a * x(k) + b * u(k);
    end
    p(n) = alpha * (x(n) - d);
    for k = n-1:-1:1
        p(k) = a * p(k+1);
    end
    grad_val = u + b * p;
end
```

[49]:
```
function plotFunctionAndPath(f3, path)
    % Fonction de représentation de la trajectoire
    figure;
    plot(path(1, :), path(2, :), 'r*-');
    title('Chemin suivi par l''algorithme');
    xlabel('u_1');
    ylabel('u_2');
    hold off;
end
```

[59]:
```
% Exemple d'utilisation
alpha = 1;
d = 1;
a = 1;
b = 1;
x0 = 0.5;
n = 2;
epsilon = 1e-6;
Nmax = 1000;

% Fonction f3 et gradient de f3
f3_func = @(u) f3(u, alpha, d, a, b, x0, n);
grad_f3_func = @(u) grad_f3(u, alpha, d, a, b, x0, n);

% Point initial
u0 = zeros(n, 1);

% Appeler l'algorithme de descente de gradient à pas optimal
[xmin, fmin, indica] = gradientDescentOptimalStep(f3_func, grad_f3_func, u0,␣
↪epsilon, Nmax);

% Afficher les résultats
disp('Point de minimum :');
disp(xmin);
disp('Valeur minimale de la fonction :');
disp(fmin);
```

```
L'algorithme diverge.
Point de minimum :
   0.1770
   0.1770
Valeur minimale de la fonction :
0.083493
```



$$\text{Avec cet méthode on obtient pas le point minimun théorique } U^* = \begin{pmatrix} \frac{1}{4} \\ \frac{1}{4} \end{pmatrix}.$$

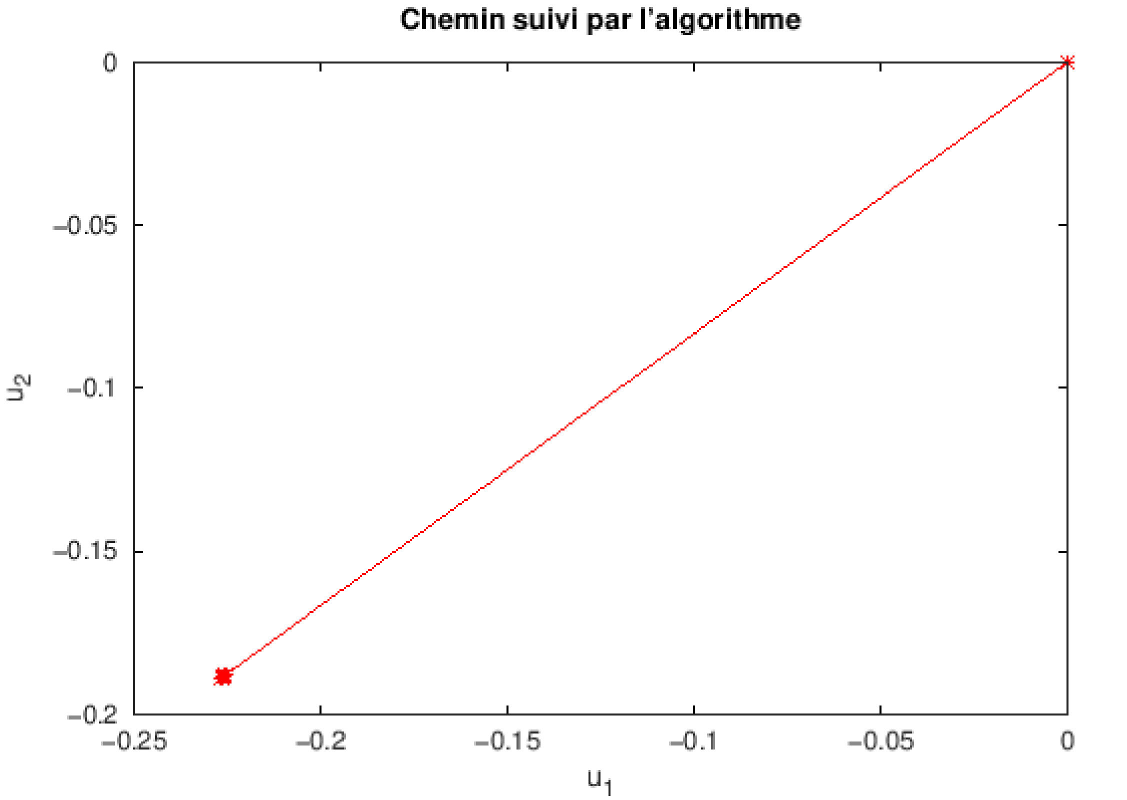
$$\text{Mais } U\text{sol} = \begin{pmatrix} 0.1770 \\ 0.1770 \end{pmatrix}.$$

EXemple complexe

[61]:
```
% Exemple d'utilisation
alpha = 2;
d = 1;
a = 1.2;
b = 0.5;
x0 = 0.5;
n = 10;
rho = 0.01;
epsilon = 1e-6;
Nmax = 1000;

% Fonction f3 et gradient de f3
```

```
f3_func = @(u) f3(u, alpha, d, a, b, x0, n);
grad_f3_func = @(u) grad_f3(u, alpha, d, a, b, x0, n);

% Point initial
u0 = zeros(n, 1);

% Appeler l'algorithme de descente de gradient à pas optimal
[xmin, fmin, indica] = gradientDescentOptimalStep(f3_func, grad_f3_func, u0, epsilon, Nmax);

% Afficher les résultats
disp('Point de minimum :');
disp(xmin);
disp('Valeur minimale de la fonction :');
disp(fmin);
```

```
L'algorithme diverge.
Point de minimum :
  -0.226689
  -0.188907
  -0.157423
  -0.131186
  -0.109321
  -0.091101
  -0.075918
  -0.063265
  -0.052721
  -0.043934
Valeur minimale de la fonction :
0.083903
```



9

La methode de gradient utilisant la règle d'Armijo pour f_3

[23]:
```
function [xmin, fmin, indica] = gradientDescentArmijo3(f3, grad_f3, u0,␣
↪alpha_armijo, beta, epsilon, Nmax)
    % Initialisation des variables
    u = u0;
    k = 0;
    grad = grad_f3(u);
    path = u; % Pour stocker le chemin suivi par l'algorithme

    % Boucle de gradient à pas optimal avec règle d'Armijo
    while (norm(grad) > epsilon) && (k <= Nmax)
        % Calcul du pas optimal
        rho = 0.5; % Valeur initiale du pas
        while f3(u - rho * grad) > f3(u) - alpha_armijo * rho * norm(grad)^2
            rho = beta * rho; % Réduire le pas
        end
```

```
        % Mise à jour de u
        u = u - rho * grad;
        % Mise à jour de grad
        grad = grad_f3(u);
        % Mise à jour du compteur d'itérations
        k = k + 1;
        % Ajouter la nouvelle position à la trajectoire
        path = [path, u];
    end

    % Vérification de la convergence
    if norm(grad) < epsilon
        xmin = u;
        fmin = f3(u);
        indica = 1;
        disp('L''algorithme converge.');
        disp('Le point de minimum :');
        disp(xmin);
        disp('La valeur minimale de la fonction :');
        disp(fmin);
    else
        xmin = u;
        fmin = f3(u);
        indica = 0;
        disp('L''algorithme diverge.');
    end

    % Représentation graphique de la fonction et du chemin
    plotFunctionAndPath(f3, path);
end
```

[18]:
```
function [f_val] = f3(u, alpha, d, a, b, x0, n)
    x = zeros(n, 1);
    x(1) = x0;
    for k = 1:n-1
        x(k+1) = a*x(k) + b*u(k);
    end
    f_val = 0.5 * norm(u)^2 + 0.5 * alpha * (x(n) - d)^2;
end
```

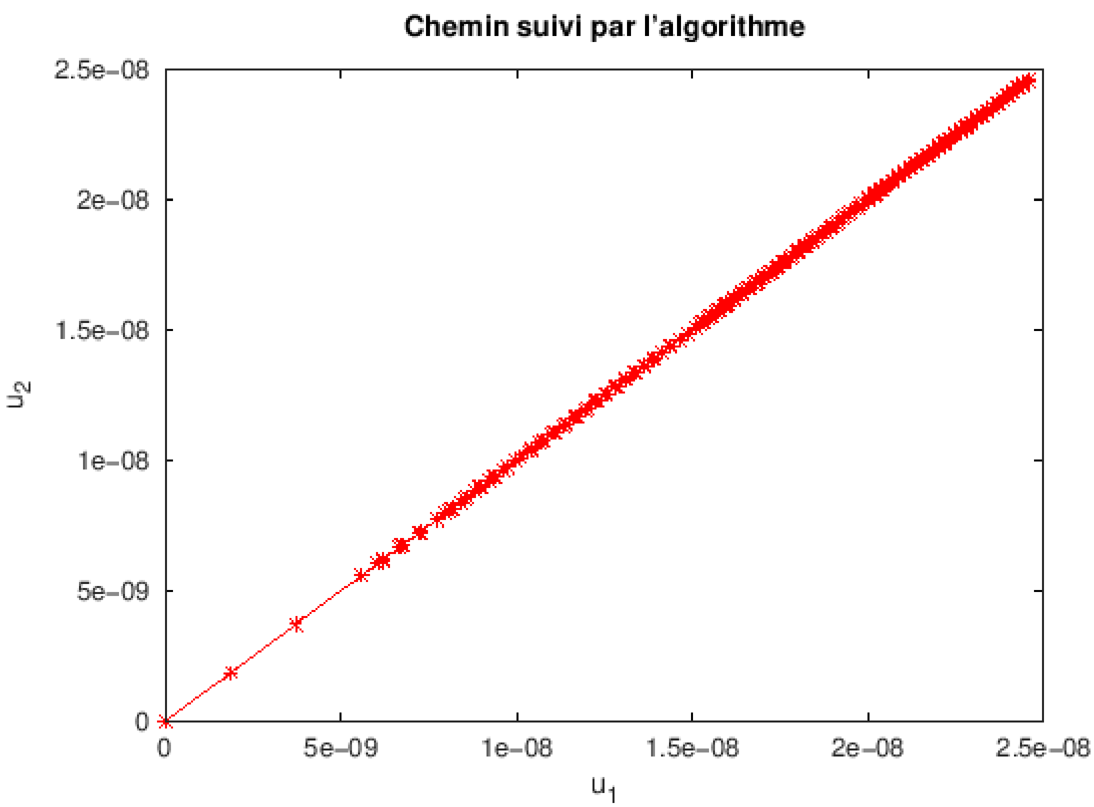
[19]:
```
function [grad_val] = grad_f3(u, alpha, d, a, b, x0, n)
    x = zeros(n, 1);
    p = zeros(n, 1);
    x(1) = x0;
    for k = 1:n-1
        x(k+1) = a * x(k) + b * u(k);
    end
```

```
    p(n) = alpha * (x(n) - d);
    for k = n-1:-1:1
        p(k) = a * p(k+1);
    end
    grad_val = u + b * p;
end
```

[20]:
```
function plotFunctionAndPath(f3, path)
    % Fonction de représentation de la trajectoire
    figure;
    plot(path(1, :), path(2, :), 'r*-');
    title('Chemin suivi par l''algorithme');
    xlabel('u_1');
    ylabel('u_2');
    hold off;
end
```

[24]:
```
% Exemple d'utilisation
alpha = 1;
d = 1;
a = 1;
b = 1;
x0 = 0.5;
n = 2;
epsilon = 1e-6;
Nmax = 1000;
alpha_armijo = 0.5; % Paramètre de décroissance pour la règle d'Armijo
beta = 0.5; % Paramètre de réduction du pas

% Fonction f3 et gradient de f3
f3_func = @(u) f3(u, alpha, d, a, b, x0, n);
grad_f3_func = @(u) grad_f3(u, alpha, d, a, b, x0, n);

% Point initial
u0 = zeros(n, 1);

% Appeler l'algorithme de descente de gradient à pas optimal avec règle d'Armijo
[xmin, fmin, indica] = gradientDescentArmijo3(f3_func, grad_f3_func, u0,␣
↪alpha_armijo, beta, epsilon, Nmax);

% Afficher les résultats
disp('Point de minimum :');
disp(xmin);
disp('Valeur minimale de la fonction :');
disp(fmin);
disp('Indicateur de convergence (1 si converge, 0 sinon) :');
disp(indica);
```

```
L'algorithme diverge.
Point de minimum :
   2.4564e-08
   2.4564e-08
Valeur minimale de la fonction :
0.1250
Indicateur de convergence (1 si converge, 0 sinon) :
0
```



exemple Complexe

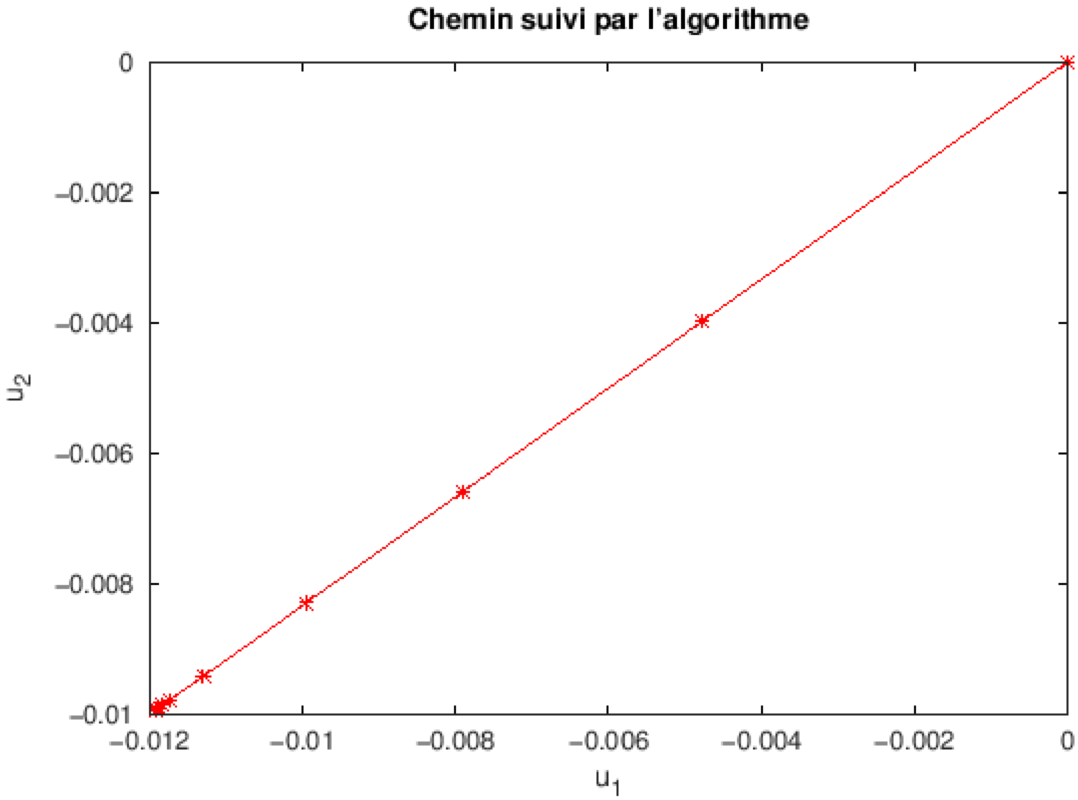
$$\text{Avec cet méthode on obtient pas le point minimun théorique } U^* = \begin{pmatrix} \frac{1}{4} \\ \frac{1}{4} \end{pmatrix}.$$

$$\text{Mais } U\text{sol} = \begin{pmatrix} 2.4564e-08 \\ 2.4564e-08 \end{pmatrix}.$$

[21]:
```
% Exemple d'utilisation
alpha = 2;
d = 1;
a = 1.2;
```

```
b = 0.5;
x0 = 0.5;
n = 5;
epsilon = 1e-6;
Nmax = 1000;
alpha_armijo = 0.5; % Paramètre de décroissance pour la règle d'Armijo
beta = 0.5; % Paramètre de réduction du pas

% Fonction f3 et gradient de f3
f3_func = @(u) f3(u, alpha, d, a, b, x0, n);
grad_f3_func = @(u) grad_f3(u, alpha, d, a, b, x0, n);

% Point initial
u0 = zeros(n, 1);

% Appeler l'algorithme de descente de gradient à pas optimal avec règle d'Armijo
[xmin, fmin, indica] = gradientDescentArmijo3(f3_func, grad_f3_func, u0,
    alpha_armijo, beta, epsilon, Nmax);

% Afficher les résultats
disp('Point de minimum :');
disp(xmin);
disp('Valeur minimale de la fonction :');
disp(fmin);
disp('Indicateur de convergence (1 si converge, 0 sinon) :');
disp(indica);
```

```
L'algorithme diverge.
Point de minimum :
  -1.1918e-02
  -9.9313e-03
  -8.2761e-03
  -6.8968e-03
  -5.7473e-03
Valeur minimale de la fonction :
3.1453e-04
Indicateur de convergence (1 si converge, 0 sinon) :
0
```



TP 1

```
[ ]:
```

On reprend dans cette partie la matrice A et le vecteur b du TP1 pour les utiliser :

$$A = \begin{pmatrix} 10 & -4 \\ -4 & 4 \end{pmatrix} \quad \text{et} \quad b = \begin{pmatrix} 1 \\ 5 \end{pmatrix}$$

10 La Méthode de gradient avec projection pour f_1

$$f_1(x) = \frac{1}{2}\langle Ax, x\rangle - \langle b, x\rangle$$

où $X \in \mathbb{R}^n$, A est une matrice SDP et $b \in \mathbb{R}^n$.

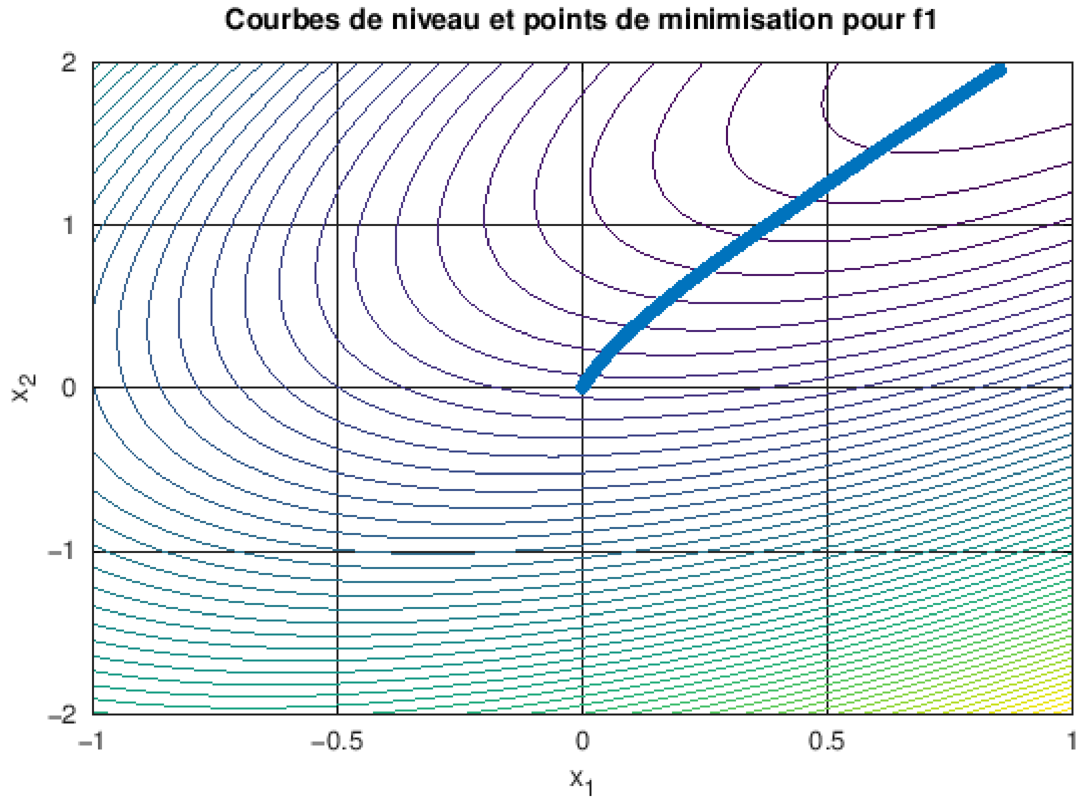
[43]:
```
function [x_opt, fval, convergence, points] = gradient_projection_f1(A, b, ai,␣
↪bi, tol, max_iter)
    % Fonction de gradient avec projection pour f1
    % A : matrice SDP
    % b : vecteur
    % ai, bi : bornes des contraintes
    % tol : tolérance pour la convergence
    % max_iter : nombre maximal d'itérations

    % Initialisation
    n = length(b);
    x = zeros(n, 1); % Point initial
    alpha = 1e-3; % Pas de gradient
    k = 0;
    convergence = []; % Enregistrement des valeurs de la fonction objective
    points = []; % Enregistrement des points

    while k < max_iter
        % Gradient de f1
        grad_f1 = A * x - b;

        % Mise à jour de x
        x_new = x - alpha * grad_f1;

        % Projection sur U
        x_new = max(min(x_new, bi), ai);

        % Enregistrer la valeur de la fonction objective et les points
        fval = 0.5 * (x_new' * A * x_new) - (b' * x_new);
        convergence = [convergence, fval];
        points = [points, x_new];

        % Vérifier la convergence
        if norm(x_new - x, 2) < tol
            break;
        end

        % Mise à jour de x
        x = x_new;
        k = k + 1;
    end

    x_opt = x;
end
```

Exemple d'utilisation avec la matrice A et b ci-dessus, et $U = [-1; 1] \times [-2; 2]$

ET nous avons deja calculé le point de minimum de f_1 dans le TP1: $X_{\min} = \begin{pmatrix} 1 \\ 2.25 \end{pmatrix}$.

[73]:
```
% Définir les paramètres
ai = [-1;-2]; % Définir les bornes inférieures
bi = [1;2 ]; % Définir les bornes supérieures
lambda = [0;0]; % Définir les paramètres pour g(x)
tol = 1e-6; % Tolérance
max_iter = 1000; % Nombre maximal d'itérations

% Pour f1
[Xmin, fmin, convergence_f1, points_f1] = gradient_projection_f1(A, b, ai, bi,␣
↪tol, max_iter);
Xmin

% Créer une grille de points pour tracer les courbes de niveau
x1 = linspace(ai(1), bi(1), 100);
x2 = linspace(ai(2), bi(2), 100);
[X1, X2] = meshgrid(x1, x2);

% Calculer les valeurs de f1 sur la grille
F1 = 0.5 * (A(1,1) * X1.^2 + 2 * A(1,2) * X1 .* X2 + A(2,2) * X2.^2) - (b(1) *␣
↪X1 + b(2) * X2);

% Tracer les courbes de niveau pour f1 et les points de minimisation
figure;
contour(X1, X2, F1, 50); % Tracer les courbes de niveau
hold on;
plot(points_f1(1, :), points_f1(2, :), '-o'); % Tracer les points de minimisation
title('Courbes de niveau et points de minimisation pour f1');
xlabel('x_1');
ylabel('x_2');
grid on;
hold off;
```

```
Xmin =

   0.8514
   1.9529
```

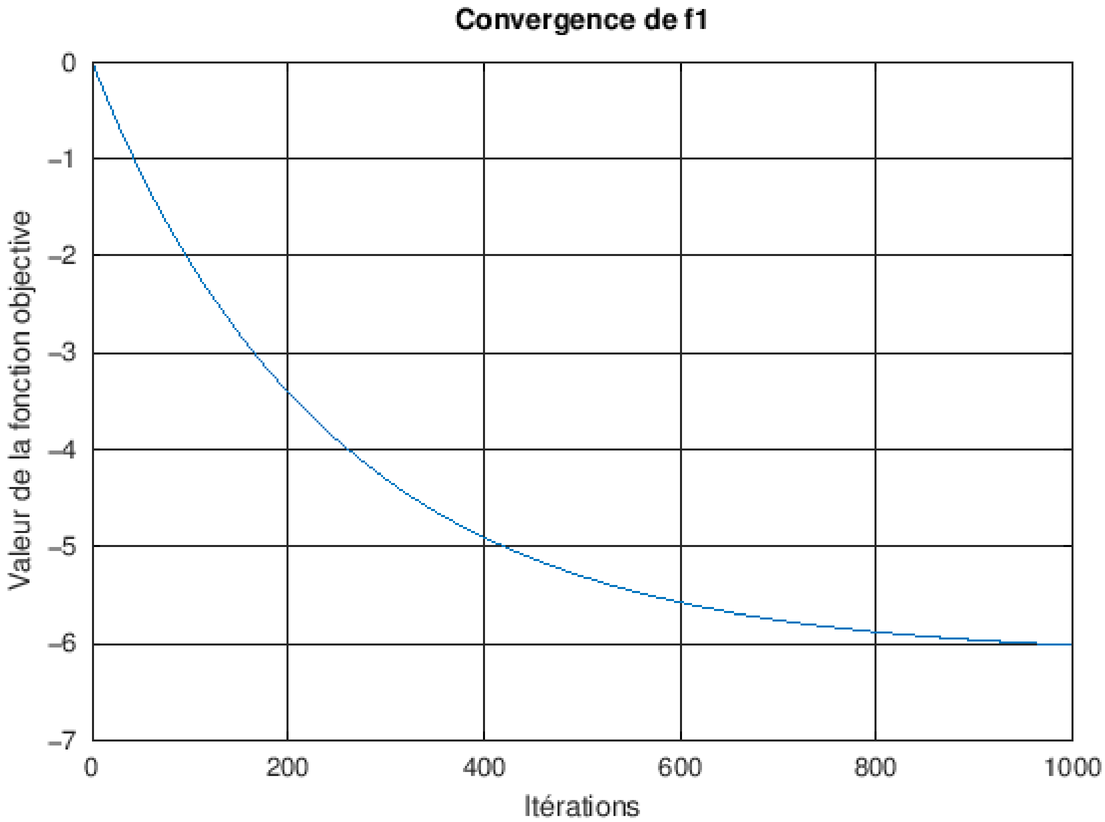


La méthode de gradient avec la projection donne $U_{\text{sol}} = \begin{pmatrix} 0.8514 \\ 1.9529 \end{pmatrix}$.

qui converge vers le point de minimum sans atteindre le point de minimum théorique $U_{\text{min}} = \begin{pmatrix} 1 \\ 2.25 \end{pmatrix}$.

[74]:
```
% Tracer la courbe de convergence pour f1
figure;
plot(convergence_f1);
title('Convergence de f1');
xlabel('Itérations');
ylabel('Valeur de la fonction objective');
grid on;

fmin
```

```
fmin = -6.0146
```



11

La Méthode de gradient avec projection pour f_2

La fonction f_2 est définie pour tout $x \in \mathbb{R}^n$ par :

$$f_2(x) = \frac{1}{2}\langle Ax, x\rangle - \langle b, x\rangle + g(x)$$

où A est une matrice carrée d'ordre n symétrique définie positive, $b \in \mathbb{R}^n$ et $g : \mathbb{R}^n \to \mathbb{R}$ une fonction de classe C^2

Prénons l'exemple le plus simple dans $\lambda_1, \lambda_2 \in \mathbb{R}^2$ tel que $\lambda_1 = \lambda_2 = 0$

Alors $g(x) = e^{(\lambda_1 x_1)} + e^{(\lambda_2 x_2)} = e^0 + e^0 = 2$

C'est qui donne

$$f_2(x) = \frac{1}{2}\langle Ax, x\rangle - \langle b, x\rangle + g(x) = f_1(x) + 2$$

$$\begin{aligned}\nabla f_2(x) &= \nabla f_1(x)\\ &= Ax - b\end{aligned}$$

[ ]:

```

```

[77]:

```
function [x_opt, fval, convergence, points] = gradient_projection_f2(A, b,␣
↪lambda, ai, bi, tol, max_iter)
    % Fonction de gradient avec projection pour f2
    % A : matrice SDP
    % b : vecteur
    % lambda : vecteur des paramètres pour g(x)
    % ai, bi : bornes des contraintes
    % tol : tolérance pour la convergence
    % max_iter : nombre maximal d'itérations

    % Initialisation
    n = length(b);
    x = zeros(n, 1); % Point initial
    alpha = 1e-3; % Pas de gradient
    k = 0;
    convergence = []; % Enregistrement des valeurs de la fonction objective
    points = []; % Enregistrement des points

    while k < max_iter
        % Gradient de f2
        grad_g = lambda .* exp(lambda .* x);
        grad_f2 = A * x - b + grad_g;

        % Mise à jour de x
        x_new = x - alpha * grad_f2;

        % Projection sur U
        x_new = max(min(x_new, bi), ai);

        % Enregistrer la valeur de la fonction objective et les points
        fval = 0.5 * (x_new' * A * x_new) - (b' * x_new) + sum(exp(lambda .*␣
↪x_new));
        convergence = [convergence, fval];
        points = [points, x_new];

        % Vérifier la convergence
        if norm(x_new - x, 2) < tol
            break;
```

```
        end

        % Mise à jour de x
        x = x_new;
        k = k + 1;
    end

    x_opt = x;
end
```

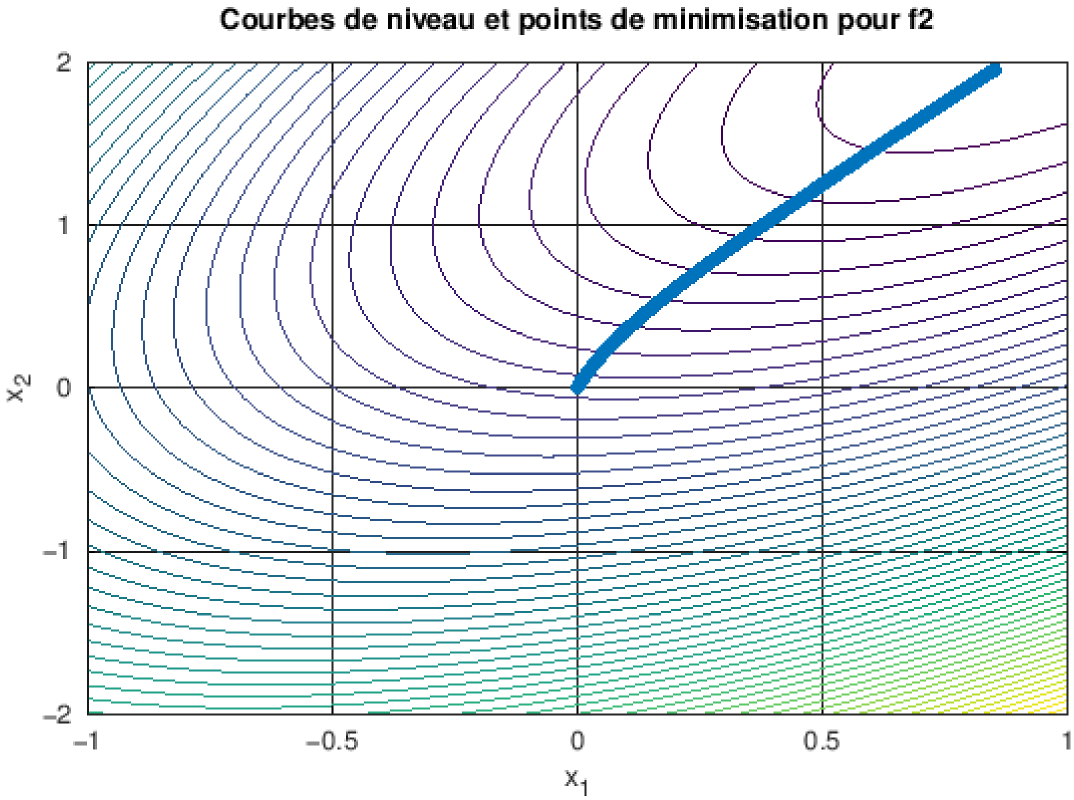
Exemple d'utilisation avec la matrice A et b ci-dessus, et $U = [-1;1] \times [-2;2]$

ET nous avons deja calculé le point de minimum de f_2 dans le TP1: $X_{\min} = \begin{pmatrix} 1 \\ 2.25 \end{pmatrix}$.

[80]:

```
% Définir les paramètres
ai = [-1;-2]; % Définir les bornes inférieures
bi = [2;3 ]; % Définir les bornes supérieures
lambda = [0;0]; % Définir les paramètres pour g(x)
tol = 1e-6; % Tolérance
max_iter = 1000; % Nombre maximal d'itérations
% Calculer les valeurs de f2 sur la grille
F2 = 0.5 * (A(1,1) * X1.^2 + 2 * A(1,2) * X1 .* X2 + A(2,2) * X2.^2) - (b(1) *␣
 ↪X1 + b(2) * X2) + exp(lambda(1) * X1) + exp(lambda(2) * X2);

disp('Le point de minimum');
Xmin

% Tracer les courbes de niveau pour f2 et les points de minimisation
figure;
contour(X1, X2, F2, 50); % Tracer les courbes de niveau
hold on;
plot(points_f2(1, :), points_f2(2, :), '-o'); % Tracer les points de minimisation
title('Courbes de niveau et points de minimisation pour f2');
xlabel('x_1');
ylabel('x_2');
grid on;
hold off;
```

```
Le point de minimum
Xmin =

   0.8514
   1.9529
```



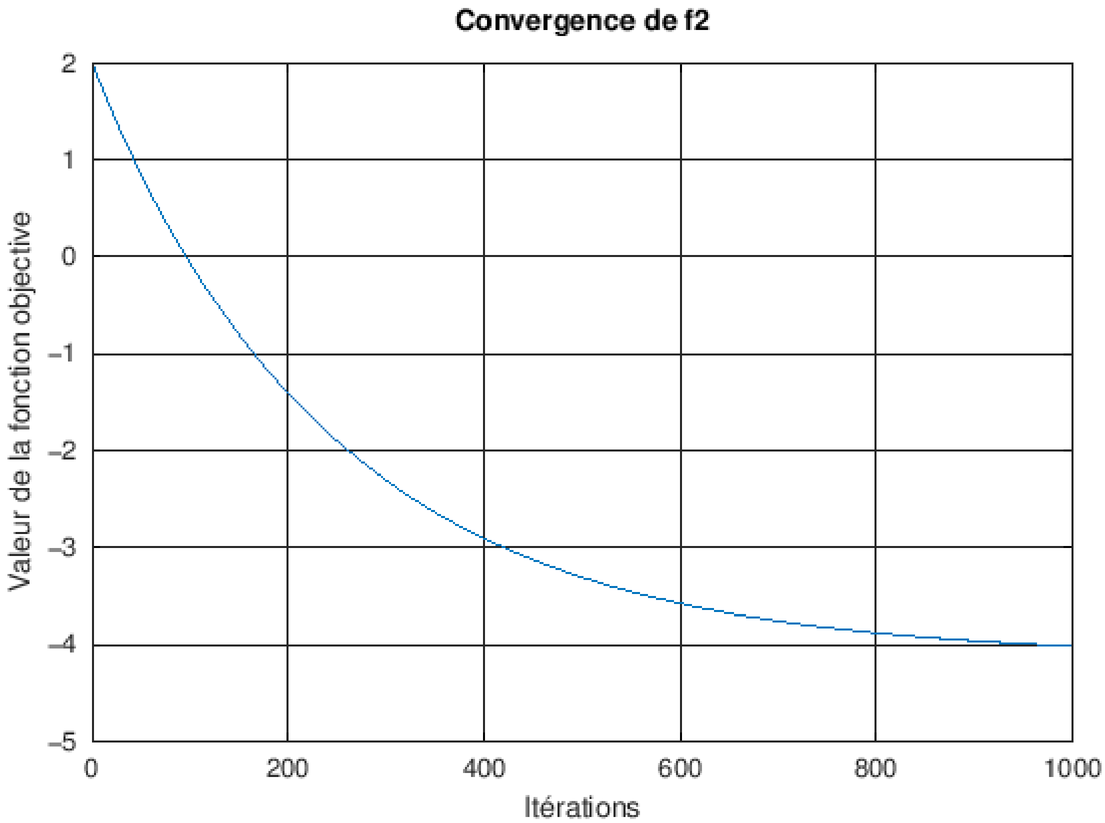
la methode de gradient avec la projection donne $U_{\text{sol}} = \begin{pmatrix} 0.8514 \\ 1.9529 \end{pmatrix}$.

qui converge vers le point de minimum sans atteindre le point de minimum théorique $U_{\text{min}} = \begin{pmatrix} 1 \\ 2.25 \end{pmatrix}$.

[79]:

```
% Pour f2
[Xmin, fval_f2, convergence_f2, points_f2] = gradient_projection_f2(A, b,␣
↪lambda, ai, bi, tol, max_iter);
disp('fmin');
fval_f2
% Tracer la courbe de convergence pour f2
figure;
plot(convergence_f2);
title('Convergence de f2');
xlabel('Itérations');
ylabel('Valeur de la fonction objective');
grid on;
```

```
fmin
fval_f2 = -4.0146
```




```
[ ]:
```