UNIVERSITE SULTAN MOULAY SLIMANE

ECOLE NATIONALE DES SCIENCES APPLIQUEES KHOURIBGA

Département : Mathématiques & Informatique

Filière : 1ère année Master – Big Data & Aide à la Décision

Sujet :

Optimisation sans contrainte

(Programmation Quadratique)

Réalisé par :

EL BASRI Akram
EL ALLAMI Ayoub
EL FADILI Salwa

Supervisé par :

Mr. EL HADFI Youssef

Année universitaire : 2023-2024

TABLE DES MATIERES

LISTE DE FIGURES

INTRODUCTION

L’optimisation sans contrainte (ou non contrainte) se réfère à la résolution de problèmes d'optimisation. Elle s'agit de trouver les valeurs des variables qui minimisent ou maximisent une certaine fonction, sans avoir à se conformer à des limitations spécifiques.

Les fonctions du second degré peuvent être appelées de différentes manières : trinômes, fonctions quadratiques ou encore fonctions polynomiales du second degré, avec un nombre quelconque de variables. Elles sont les fonctions les plus simples après les fonctions affines.

La programmation quadratique est considérée comme un cas particulier de la programmation non linéaire. L’optimisation quadratique est l’une des théories de la programmation mathématique la plus utilisée pour modéliser des problèmes pratiques. De nombreux domaines d’application s’appuient sur cette branche tels que : la recherche opérationnelle, la science de l’ingénieur, la physique (l’étude théorique d’une chute libre), les sciences économiques …

La représentation graphique d’une fonction du second degré est une parabole qui possède un axe de symétrie parallèle à l’axe des ordonnées. Le signe du coefficient associé au plus haut degré (ou coefficient du second degré) indique le sens de la variation de la fonction.

Ce rapport commence par une introduction, ensuite une partie concernant l’application quadratique, ensuite une partie sur la programmation quadratique en faisant déclarer quelques théorèmes, puis un exemple pratique expliquant plus cette théorie, et enfin un exercice en python.

Applications quadratiques

Définition

On dit que f : $R^n \rightarrow R$ est une application quadratique si elle s'agit d'un polynôme de degré **2**.
Elle s'écrit sous la forme :

$$\mathbf{f(x_1, x_2 \dots x_n)} = \frac{1}{2}\sum_{i=1}^{n} a_{ii}x_i^2 + \sum_{i<j} a_{ij}\, x_i x_j \; - \sum_{i=1}^{n} b_i x_i + c$$

- La forme quadratique d'une variable :

$$f(x) = ax^2$$

- La forme quadratique de deux variables :

$$f(x, y) = ax^2 + by^2 + 2cxy$$

- La forme quadratique de trois variables :

$$f(x, y, z) = ax^2 + by^2 + cz^2 + 2dxy + 2exz + 2fyz$$

Où a, b, c, d, e et f sont des nombres réels qui ne dépendent pas des variables x, y et z, avec $a \neq 0$.

La notation matricielle de l'application quadratique est sous la forme :

$$f(x) = \frac{1}{2}X^T AX - b^T X + c$$

Avec :

- **A** une matrice réelle d'ordre n
- $\mathbf{X} = \begin{pmatrix} \mathbf{x_1} \\ . \\ . \\ . \\ \mathbf{x_n} \end{pmatrix} \in \mathbf{R^n}$; $\mathbf{b} = \begin{pmatrix} \mathbf{b_1} \\ . \\ . \\ . \\ \mathbf{b_n} \end{pmatrix} \in \mathbf{R^n}$
- $\mathbf{c} \in R$ (c est une constante)

Théorème (Gradient et matrice Hessienne d'une application quadratique**) :**

Soit $\mathbf{f(x)} = \frac{\mathbf{1}}{\mathbf{2}} < \mathbf{AX}, \mathbf{X} > - < \mathbf{b}, \mathbf{X} > + \mathbf{c}$ une application quadratique. Donc, elle est infiniment différentiable.

$$\boldsymbol{\nabla f(x) = AX - b}$$

Matrice Hessienne : $\boldsymbol{\nabla^2 f(x) = A}$

Démonstration :

f polynomiale $\Rightarrow$ f est infiniment différentiable.

$$\forall \mathbf{i = 1, 2, \dots, n} \qquad \boldsymbol{\frac{\partial f}{\partial x_i}(x) = \sum_{j=1}^{n} a_{ij} x_j - b_i \Rightarrow \nabla f(x) = AX - b}$$

$$\forall \mathbf{i, j = 1, 2, \dots, n} \qquad \boldsymbol{\frac{\partial^2 f}{\partial x_i \partial x_j}(x) = a_{ij} \Rightarrow \nabla^2 f(x) = A}$$

Programmation quadratique

Théorème (Matrice semi-définie positive, définie positive, semi-définie négative, définie négative**) :**

Si f est une forme quadratique représentée par la matrice symétrique A :

- f est semi-définie positive si toutes les valeurs propres de A sont positives et si au moins l'une d'entre elles est nulle.
- f est définie positive si et seulement si toutes les valeurs propres de A sont positives.
- f est semi-définie négative si toutes les valeurs propres de A sont négatives et si au moins l'une d'entre elles est nulle.
- f est définie négative si et seulement si toutes les valeurs propres de A sont négatives.

Théorème (Caractérisation de la convexité à l'ordre 2**) :**

Si f est 2 fois différentiable sur un ouvert convexe U, alors :

- $\forall \mathbf{x} \in \mathbf{U}, \boldsymbol{\nabla^2 f(x)}$ est semi-définie positive ⇔ f est convexe sur U
- $\forall \mathbf{x} \in \mathbf{U}, \boldsymbol{\nabla^2 f(x)}$ est définie positive ⇒ f est strictement convexe

Théorème (Convexité d'une application quadratique**) :**

La convexité d'une application quadratique est totalement caractérisée par sa matrice Hessienne. En se basant sur le théorème précédent de la caractérisation de la convexité à l'ordre 2.

Soit $\mathbf{f(x)} = \frac{\mathbf{1}}{\mathbf{2}} < \mathbf{AX}, \mathbf{X} > - < \mathbf{b}, \mathbf{X} > + \mathbf{c}$ une application quadratique. Alors :

- **A est semi-définie positive ⇔** f **convexe.**
- **A est définie positive ⇔ f est fortement convexe**

  ⇔ f est strictement convexe.

Démonstration :

On a **:** $\boldsymbol{\nabla^2 f(x)}$ = **A**.

Par définition, **A** est définie positive si et seulement si **f** est fortement convexe (autrement dit, la plus petite valeur propre de **A** > **0**).

f fortement convexe ⇒ **f** strictement convexe.

Supposons que **f** est strictement convexe ; **f** convexe ⇒ **A** semi-définie positive.

Par absurde :

Supposons que **A** n'est pas définie positive : ainsi **A** admet **0** pour valeur propre et soit le sous-espace propre associé $\mathbf{E_0}$ = **ker A** ; il est au moins d'une dimension et $\forall\ \mathbf{x} \in \boldsymbol{E_0}, < \boldsymbol{AX}, \boldsymbol{X} > = \mathbf{0}.$ Le

développement de **Taylor-Young** à l'ordre 2 s'écrit comme suit, puisque f est quadratique :

$\forall x \in R^n, f(u+X) - f(u) = < \nabla f(u), X > + \frac{1}{2} < AX, X >$

Ainsi, $\forall x \in E_0, f(u+X) = f(u) + < \nabla f(u), X >$

Donc, la restriction de **f** à E_0 est une application **affine** ⇒ **f** convexe mais non strictement convexe. Contradiction : de **f** est strictement convexe. Ainsi, **A** est définie positive.

A partir de ce qu'on a vu précédemment, nous pouvons maintenant caractériser les extrema d'une application quadratique.

Théorème (Programmation quadratique**) :**

Soient $f(x) = \frac{1}{2} < AX, X > - < b, X > + c$ une application quadratique sur $\mathbf{R^n}$ et $\mathbf{m} \in \mathbf{R^n}$.

- Si **A** est semi-définie positive (resp. Semi-définie négative), alors :
  - **m est un minimum** (resp. Maximum) **local de f ;**
  - **m est un minimum** (resp. Maximum) **global de f ;**
  - **Am = b ; m** est une solution du système d'équations linéaires **AX = b**

- Si **A** est définie positive (resp. Négative), alors f admet un unique minimum (resp. Maximum) global.
- Si **A** n'est pas semi-définie positive (resp. Semi-définie négative), alors f n'admet aucun minimum (resp. maximum) local ou global.

Exemple pratique

Le but de cet exemple est de déterminer si la fonction admet un extrema global. Puis, préciser ses coordonnées.

Soit $\boldsymbol{f(x) = \frac{1}{2}X^T AX - b^T X}$, avec $\boldsymbol{A = \begin{pmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{pmatrix}}$ et $\boldsymbol{b = \begin{pmatrix} -3 \\ 1 \\ -2 \end{pmatrix}}$.

Le polynôme caractéristique de A est :

$\boldsymbol{P_A(\lambda) = \det(A - \lambda I)}$**,** où I est la matrice identité du même ordre que A. Dans ce cas, I = I_3.

$$\boldsymbol{P_A(\lambda) = 8 - 12\lambda + 6\lambda^2 - \lambda^3}$$

Puisque **A** est symétrique réelle, alors elle est diagonalisable.

Les valeurs propres de **A** sont toutes > **0**.

Donc, **A** est définie positive ⇒ **f** admet un **unique minimum global**. (D'après le théorème ci-dessus).

Ce minimum global est la solution unique du système des équations linéaires suivant : **AX = b**

$$\mathbf{AX = b} \iff \begin{cases} 2x - y = -3 \\ -x + 2y - z = 1 \\ -y + 2z = -2 \end{cases}$$

$$\iff X = \begin{pmatrix} -9/4 \\ -3/2 \\ -7/4 \end{pmatrix} \text{ Minimum de f}$$

Exercice en Python

Notre application Streamlit, intitulée "Programmation Quadratique Application", offre une interface utilisateur interactive pour explorer les propriétés d'une fonction quadratique définie par sa matrice hessienne. Cette application, entièrement implémentée en Python, fournit une approche visuelle et conviviale permettant à l'utilisateur de saisir les éléments de la matrice hessienne et du vecteur b, de visualiser les résultats et d'interpréter les implications mathématiques associées.

Fonctionnement de l'Application :

- **Page de Saisie des Paramètres :**

  Sur cette page, l'utilisateur est invité à spécifier les dimensions de la matrice hessienne, ainsi que les éléments de celle-ci et du vecteur b. Grâce à des widgets interactifs tels que des champs de saisie numérique et des boutons, l'utilisateur peut entrer les données nécessaires pour caractériser la fonction quadratique.
  Après avoir validé les données, l'application utilise des fonctions Python sous-jacentes pour calculer le polynôme caractéristique, les valeurs propres de la matrice hessienne, et présenter un aperçu des résultats.

- **Affichage des Résultats :**

  Cette page présente les résultats calculés à partir des données saisies par l'utilisateur. Elle affiche la matrice hessienne, le vecteur b, le polynôme caractéristique, ainsi que les valeurs propres associées. En fonction des propriétés des valeurs propres, l'application offre une analyse qualitative de la nature de la fonction quadratique, indiquant si elle est convexe, concave, ou si les propriétés ne sont pas déterminées uniquement par les valeurs propres.
  De plus, si la fonction est caractérisée comme strictement convexe, l'application résout également le système linéaire associé HX = b pour déterminer le minimum global. Réciproquement s'elle est concave elle déterminer le maximum global.
  (Toutes les fonctions implémentées dans cette application se trouvent en annexe ci-dessous.)

Annexes

Annexe 01 : La fonction principale :

```python
def main():
    # Déclarer la session_state pour stocker les parametres
    if "hessian_matrix" not in st.session_state:
        st.session_state.hessian_matrix = None
    if "b" not in st.session_state:
        st.session_state.b = None
    if "polynome" not in st.session_state:
        st.session_state.polynome = None
    if "valeurs_propres" not in st.session_state:
        st.session_state.valeurs_propres = None
    # Utiliser les pages pour gérer la navigation
    pages = {"Enter Parameters": input_page, "Display Results": display_result_page}
    current_page = st.radio("Options : ", list(pages.keys()))
    # Afficher la page actuelle
    pages[current_page]()

main()
```

Annexe 01 : main function

Annexe 02 : La page d'entrer:

```python
def input_page():
    st.title("Page de Saisie des Paramètres :")
    st.sidebar.title("Programmation quadratique Application")
    st.sidebar.markdown("---")
    st.write("La Matrice Hessienne : ")
    n = st.number_input("Nombre de lignes:", min_value=2, step=1)
    m = st.number_input("Nombre de colonnes:", min_value=2, step=1)
    hessian_matrix, b = create_hessian_matrix(int(n), int(m))
    if st.button("Valider"):
        st.session_state.hessian_matrix = hessian_matrix
        st.session_state.b = b
        try:
            polynome, valeurs_propres = polynome_caracteristique(hessian_matrix.tolist())
            st.session_state.polynome = polynome
            st.session_state.valeurs_propres = valeurs_propres
            st.success("Les données saisies sont correctes.")
            st.info("Cliquez sur 'Display Results' ci-dessus pour voir les résultats.")
        except Exception as e:
            st.error(f"Erreur : {e}")
```

Annexe 02 : input page

Annexe 03 : La page de résultats:

```python
def display_result_page():
    st.sidebar.title("Programmation quadratique Application")
    st.sidebar.markdown("---")
    st.title("Display Results Page: ")
    if st.session_state.hessian_matrix is None or st.session_state.b is None:
        st.info("Merci de saisir les données avant de tester.")
        return
    # Afficher la matrice depuis la session
    st.write("Matrice Hessienne saisie H:")
    st.write(np.array(st.session_state.hessian_matrix))
    st.write(f"Le vecteur b = {tuple([i for i in st.session_state.b])}")
    # Affichage du résultat
    st.write("Polynôme caractéristique :", st.session_state.polynome)
    st.write("Valeurs propres associées :", st.session_state.valeurs_propres)
```

Annexe 04 : Results page part -1-

```python
n, m = st.session_state.hessian_matrix.shape
    # Tester les valeurs propres
    if all(val > 0 for val in st.session_state.valeurs_propres):
        st.success("Toutes les valeurs propres sont strictement positives. La fonction est strictement
convexe, "
                   "admet un unique minimum global, et la solution unique du système HX = b est le minimum
global.")
        # Calculer la solution du système HX = b
        solution = solve_system(st.session_state.hessian_matrix, st.session_state.b)
        if solution is not None:
            st.write(f"La solution du système HX = b est : {solution}")
    elif all(val < 0 for val in st.session_state.valeurs_propres):
        st.success("Toutes les valeurs propres sont strictement négatives. La fonction est strictement
concave, "
                   "admet un unique maximum global.")
        solution = solve_system(st.session_state.hessian_matrix, st.session_state.b)
        if solution is not None:
            st.write(f"La solution du système HX = b est : {solution}")
    else:
        st.warning("Les valeurs propres ne sont pas toutes strictement positives ni toutes strictement
négatives. "
                   "La nature de la fonction n'est pas déterminée uniquement par le test sur les valeurs
propres.")
```

Annexe 03 : Results page partie 2

Annexe 04 : Des fonctions supplémentaires:

```python
import streamlit as st
import numpy as np
from sympy import Matrix, symbols

def create_hessian_matrix(n, m):
    matrix = np.zeros((n, m))
    b = np.zeros(n)
    for i in range(n):
        st.write(f"Entrez les éléments de la ligne {i + 1} de la matrice hessienne:")
        for j in range(m):
            matrix[i][j] = st.number_input(f"Element {j + 1} :", key=f"{i}-{j}", step=0.5)
    st.write("Entrer les valeurs du vecteur b: ")
    b = [st.number_input(f"b[{i + 1}] = ") for i in range(n)]

    return matrix, b
```

Annexe 06 : Read hessien matrix

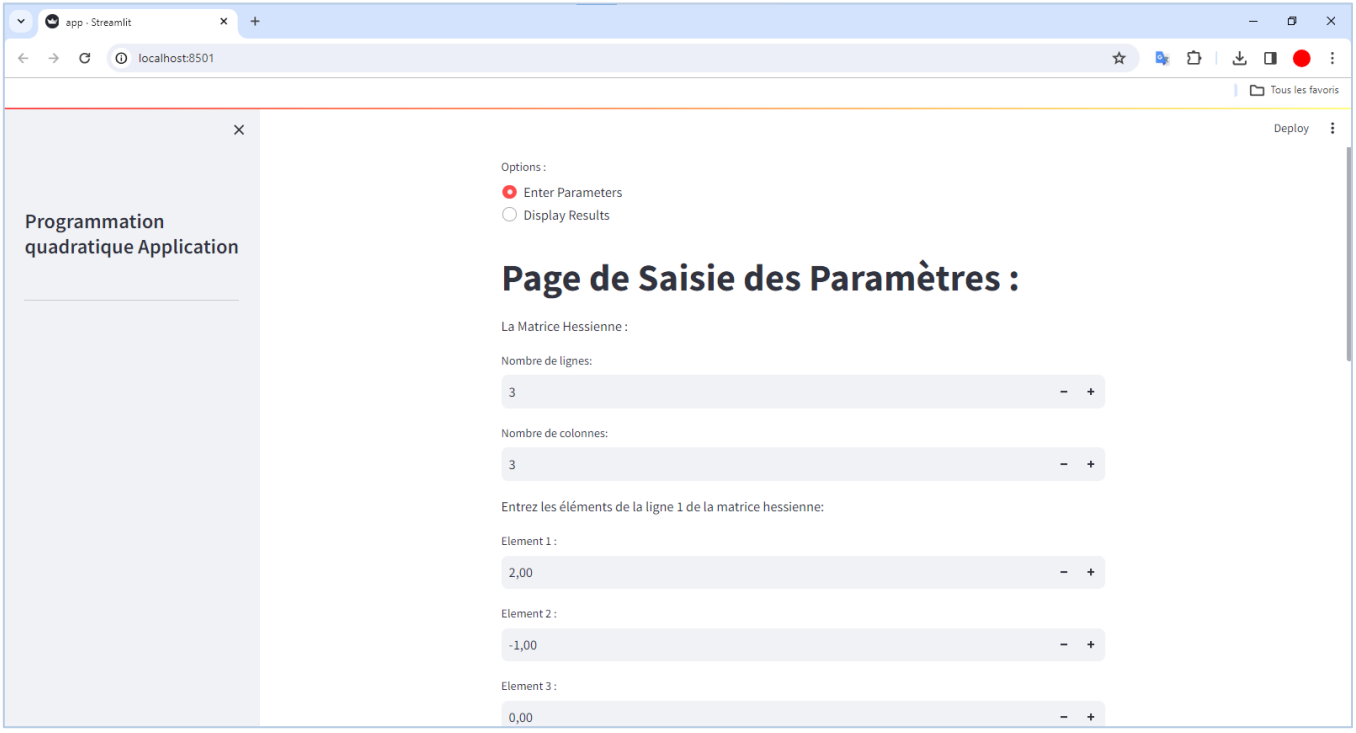
```python
def solve_system(hessian_matrix, b):
    try:
        solution = np.linalg.solve(hessian_matrix, b)
        return tuple(solution)
    except Exception as e:
        st.error(f"Erreur lors du calcul de la solution : {e}")
        return None
```

Annexe 5 : Solve system function

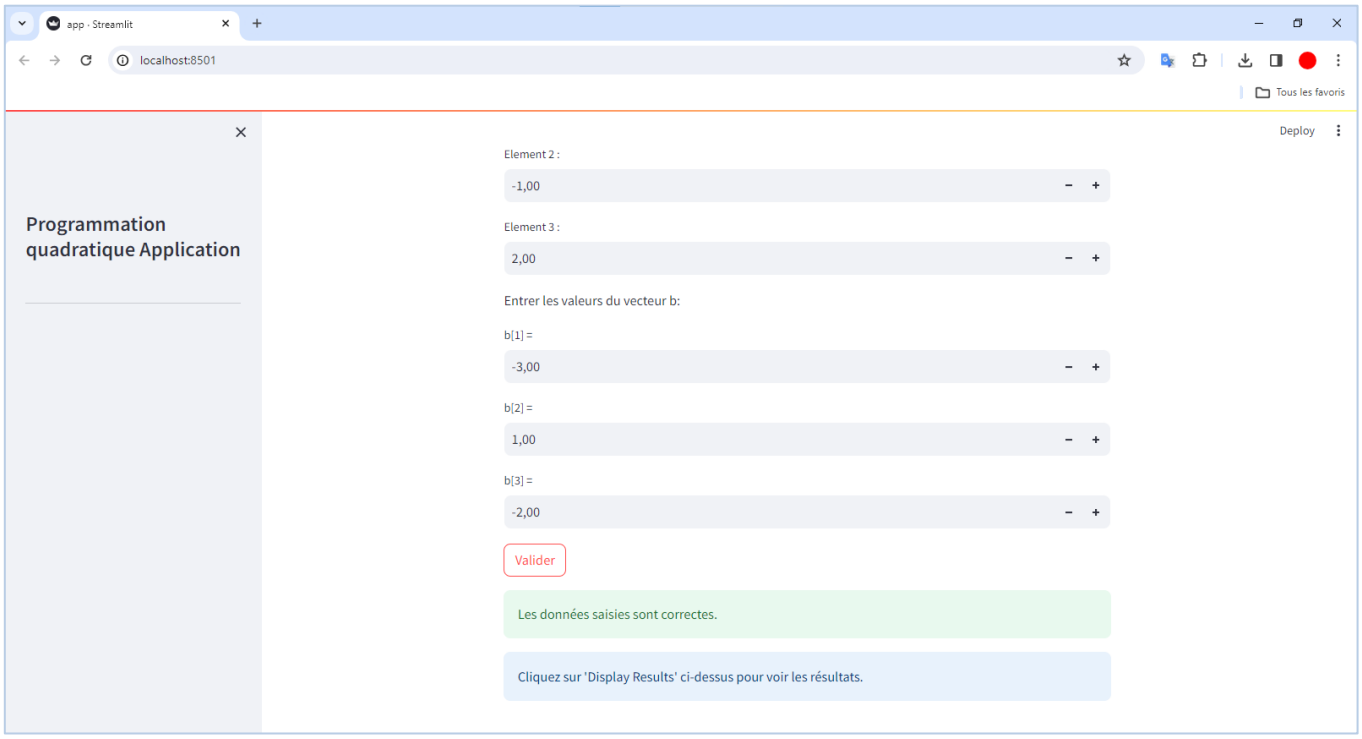
```python
def polynome_caracteristique(matrice):
    matrice_sympy = Matrix(matrice)
    lambda_sym = symbols('lambda')
    # Calculer le polynôme caractéristique
    polynome = matrice_sympy.charpoly(lambda_sym).as_expr()
    # Calculer les valeurs propres
    valeurs_propres = np.linalg.eigvals(matrice)

    return polynome, valeurs_propres
```

Annexe 07 : Compute characteristic polynomial function

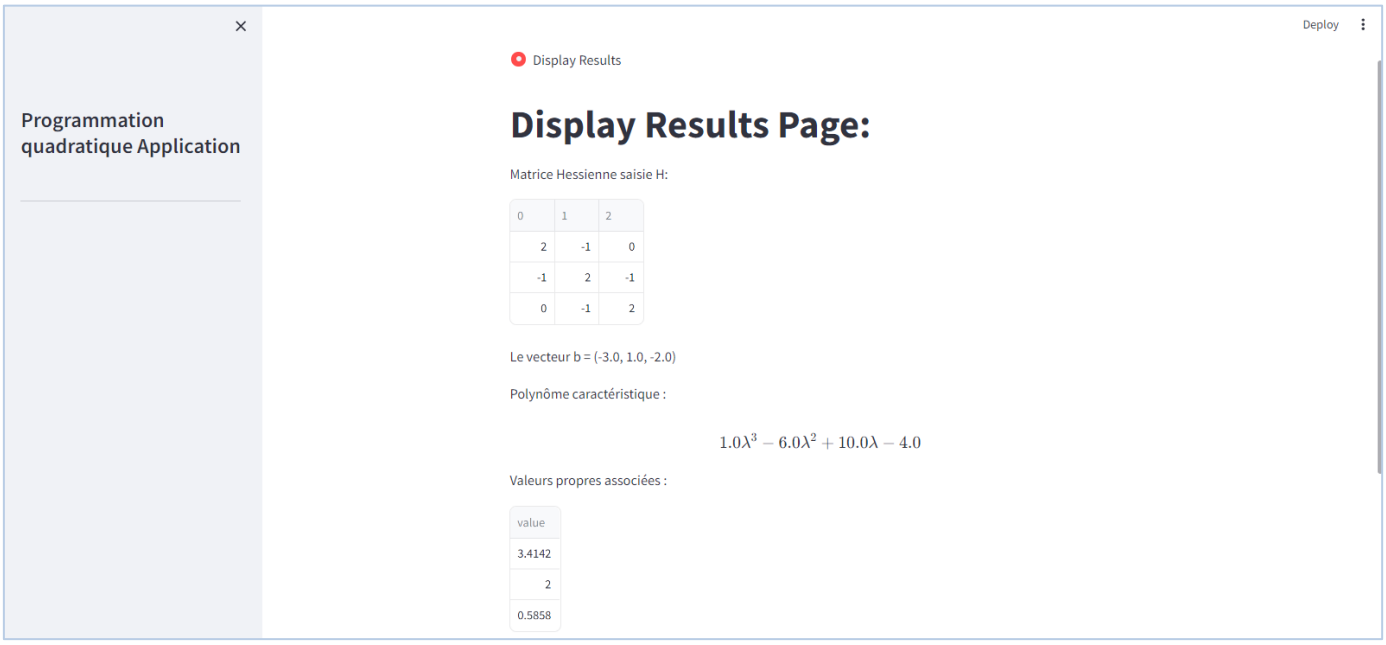
Annexe 06 : Les interfaces de notre application python :



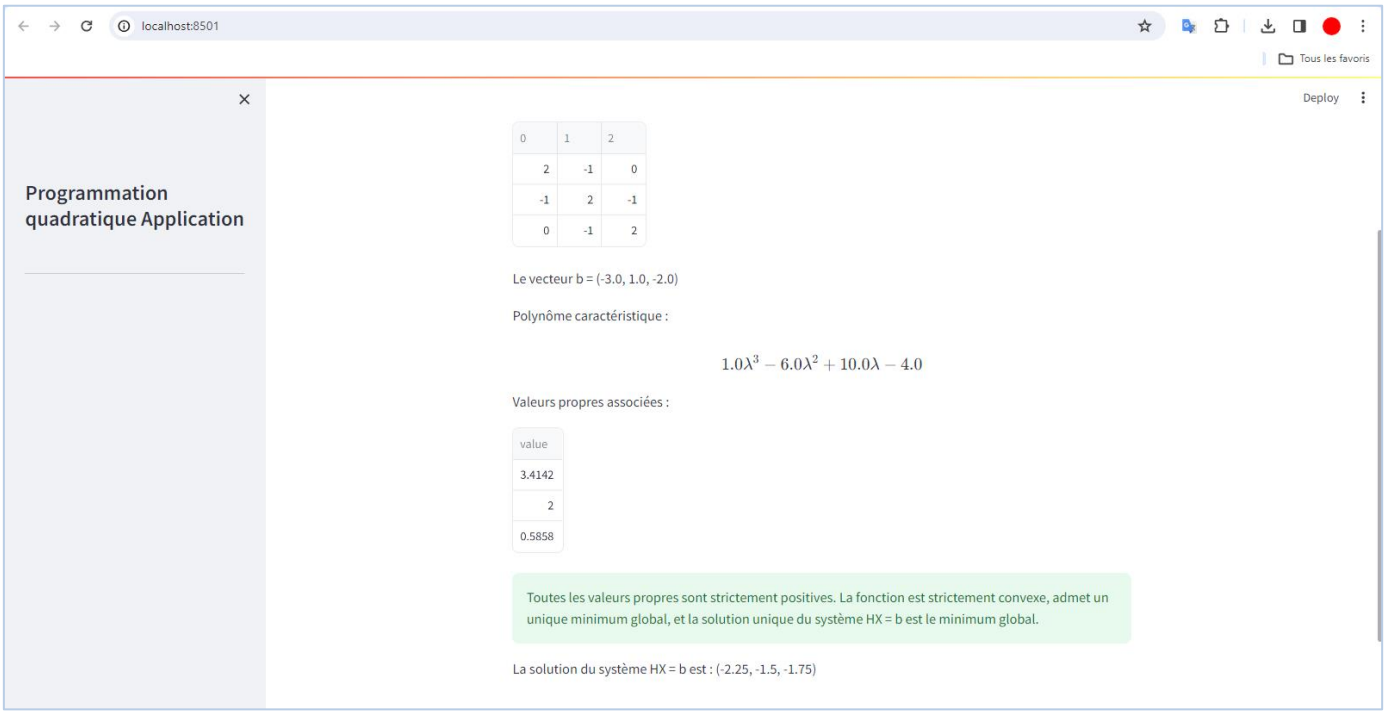
Annexe 08 : La page de saisir les paramètres (Matreice hessien & le vecteur b) part -1-



Annexe 010 : La page de saisir les paramètres - part 2- validation d'entrées



Annexe 09 : La page des résultats - part 1- calcule les valeurs propres.



Annexe 11: Affichage des extremums s'elles existes.