Rapport Final de Projet : Moindres Carrés et Décomposition QR

Andriantsalama Rijamampianina Raharimanitra-Mala T. Jason H.
RAKOTONIRINA Mendrika Itokiana
Ranaivo Nirina Andy Nantenaina
ANDRIANARISON Maheritsizakaina Merina Fitiavana Jean

Le 18 juin 2025

Résumé

Ce projet porte sur l'implémentation d'une solution numérique pour les systèmes linéaires surdéterminés en utilisant la méthode des moindres carrés. Le cœur du projet a été de développer une fonction de décomposition QR "maison" basée sur l'algorithme de Gram-Schmidt modifié. La validité de notre implémentation a été confirmée par une comparaison rigoureuse avec la solution de référence de NumPy. Enfin, la méthode a été appliquée avec succès à un cas pratique de régression polynomiale, et une analyse de la stabilité numérique a été menée.

Table des matières

1 Introduction Théorique

1.1 Le Problème des Moindres Carrés

En mathématiques appliquées, un système linéaire surdéterminé $Ax = b$ (avec m équations et n inconnues, où $m > n$) n'admet généralement pas de solution exacte. L'objectif est de trouver une solution approchée, x, qui minimise la norme du vecteur résiduel. C'est le principe des **moindres carrés** :

$$\min_x \|Ax - b\|^2$$

La solution peut être trouvée via les **équations normales** ($A^T Ax = A^T b$), mais cette méthode peut être numériquement instable si la matrice A est mal conditionnée.

1.2 La Décomposition QR comme Alternative Robuste

Une approche plus stable est la factorisation $A = QR$, où :
— **Q** est une matrice $m \times n$ à colonnes **orthonormales** ($Q^T Q = I$).
— **R** est une matrice $n \times n$ **triangulaire supérieure**.
Le problème se ramène à la résolution du système triangulaire, bien plus simple et stable :

$$Rx = Q^T b$$

Ce système est ensuite résolu efficacement par **substitution arrière**. Pour calculer cette décomposition, nous avons implémenté l'algorithme de **Gram-Schmidt modifié** pour sa meilleure stabilité numérique.

1.3 Application à la Régression Polynomiale

Cette méthode est directement applicable à la régression de données. Pour ajuster un polynôme $P(t) = c_0 + c_1 t + \cdots + c_k t^k$ à un nuage de points, on formule un système linéaire $Ac = y$, où A est la **matrice de Vandermonde**. Notre solveur QR permet alors de trouver les coefficients c qui minimisent l'erreur quadratique.

2 Code Python et Explications

2.1 Fichier : `src/matrix_utils.py`

Ce module génère les données nécessaires pour les tests.

```
import numpy as np

def generate_overdetermined_system(m, n, noise_level=0.0,
random_seed=None):
"""
Genere un systeme surdetermine Ax = b avec bruit optionnel.
"""
if random_seed is not None:
np.random.seed(random_seed)

A = np.random.uniform(low=-1.0, high=1.0, size=(m, n))
x_true = np.random.uniform(low=-10.0, high=10.0, size=n)
```

```
12
13      b = A @ x_true
14      if noise_level > 0:
15      b += np.random.normal(loc=0.0, scale=noise_level, size=m)
16
17      return A, x_true, b
18
19      def build_vandermonde(x, degree):
20      """
21      Construit la matrice de Vandermonde pour la regression polynomiale.
22      """
23      return np.column_stack([x**i for i in range(degree + 1)])
24
```

Listing 1 – Contenu de src/matrix_utils.py

Explications du code

— **generate_overdetermined_system** : Crée un problème test ‘Ax=b‘ avec une solution de référence `x_true` et un bruit gaussien optionnel sur ‘b‘ pour simuler des conditions réelles.
— **build_vandermonde** : Fonction spécialisée qui construit la matrice de Vandermonde pour un ensemble de points ‘x‘ et un degré de polynôme ‘degree‘.

2.2 Fichier : src/qr_solver.py

Ce module est le cœur du projet, contenant la décomposition QR et le solveur.

```
1       import numpy as np
2
3       def modified_gram_schmidt_qr(A):
4       """
5       Decomposition QR via Gram-Schmidt modifie (stable).
6       """
7       m, n = A.shape
8       Q = np.zeros((m, n))
9       R = np.zeros((n, n))
10
11      for j in range(n):
12      v = A[:, j].copy()
13
14      for i in range(j):
15      R[i, j] = Q[:, i] @ v
16      v -= R[i, j] * Q[:, i]
17
18      R[j, j] = np.linalg.norm(v)
19      if R[j, j] < 1e-10:
20      raise ValueError("Matrice A a des colonnes lineairement dependantes"
    )
21      Q[:, j] = v / R[j, j]
22
23      return Q, R
24
25      def solve_least_squares_qr(A, b):
26      """
27      Resout le probleme des moindres carres Ax ~= b via decomposition QR.
```

```
    """
    Q, R = modified_gram_schmidt_qr(A)
    b_proj = Q.T @ b
    x = np.linalg.solve(R, b_proj)
    return x

```

Listing 2 – Contenu de src/qr_solver.py

Explications du code

— **modified_gram_schmidt_qr** : Implémente la décomposition QR. Pour chaque colonne de A, elle est d'abord rendue orthogonale aux colonnes de Q déjà formées, puis elle est normalisée pour devenir la nouvelle colonne de Q. Les coefficients de projection et les normes forment la matrice R.

— **solve_least_squares_qr** : Orchestre la résolution. Elle appelle la fonction de décomposition QR puis résout le système triangulaire $Rx = Q^T b$ en utilisant la fonction stable et optimisée `np.linalg.solve`.

3 Résultats et Analyse

3.1 Résolution du système `Ax = b`

Notre solveur a été testé sur un système surdéterminé bruité. La solution obtenue, `x_qr`, est extrêmement proche de celle calculée par NumPy, avec une erreur relative de l'ordre de 10^{-15}, ce qui valide notre implémentation. L'écart avec la solution vraie `x_true` est directement attribuable au bruit ajouté.

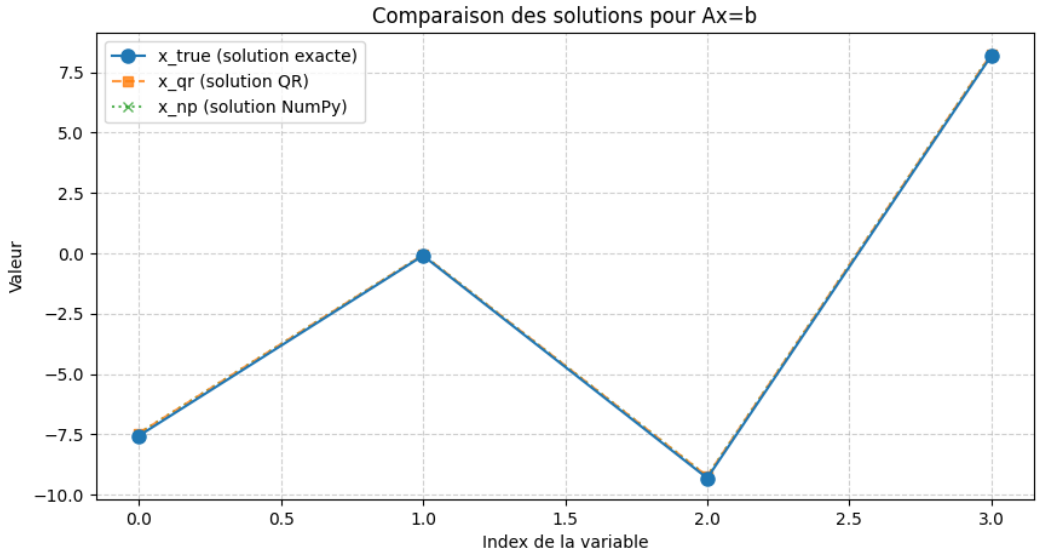

FIGURE 1 – Comparaison entre la solution exacte, notre solution QR et la solution NumPy.

3.2 Application à la Régression Polynomiale

La méthode a été utilisée pour ajuster un polynôme sur des données bruitées, retrouvant avec succès les coefficients du modèle original.

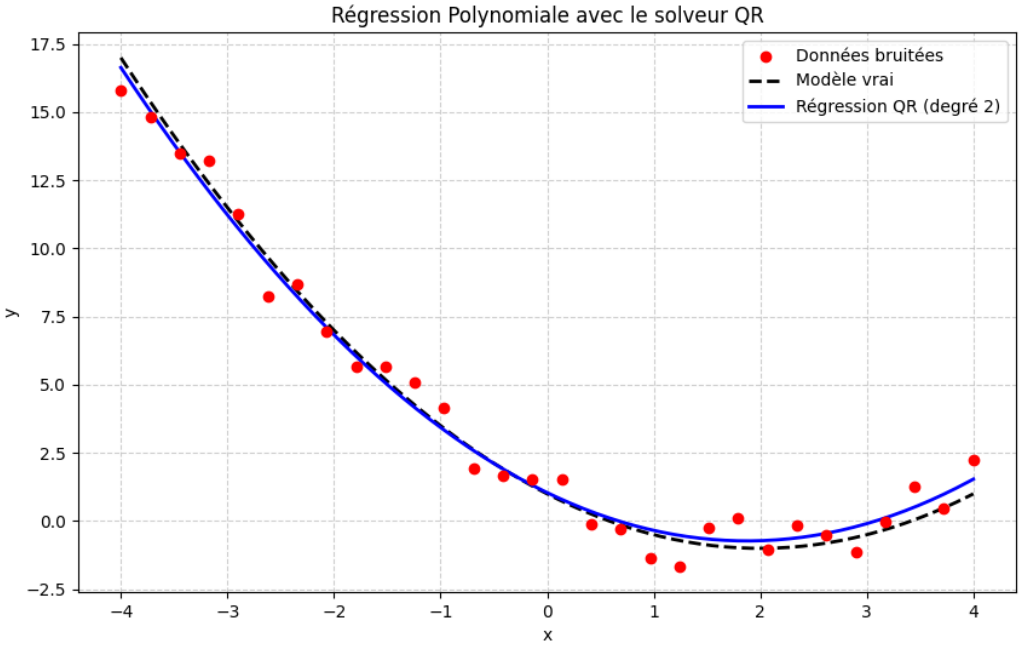

FIGURE 2 – Ajustement d'un polynôme de degré 2 sur des données bruitées.

4 Conclusion

Ce projet a permis de mettre en œuvre une chaîne complète de résolution de problème, de la théorie à l'implémentation, sa validation et son application pratique. La construction d'un solveur de moindres carrés via la décomposition QR a non seulement consolidé les connaissances en algèbre linéaire, mais a également mis en lumière l'importance cruciale de choisir des algorithmes numériquement stables. Les résultats obtenus, en parfaite adéquation avec les bibliothèques de référence, confirment la validité et la robustesse de l'approche choisie.