Chapitre 4
Applications

M. L-Lahlou

UNIVERSITE CADI AYYAD **Département de Mathématiques**
Faculté des Sciences Semlalia Marrakech
Master SMA/S1/Automne 2023
Langage Python
Applications au Calcul Scientifique

Application 1 (Meilleure approximation aus sens des moindres carrées(m.a.m.c))

Théorie : Soient $x_1 < x_2 < \cdots < x_m$ m abscisses distinctes, $y = (y_1, \cdots, y_m)$ un vecteur de $\mathbb{R}^m$, $\{\phi_1, \phi_2, \cdots, \phi_n\}$ des fonctions linéairement indépendantes définies sur $\mathbb{R}$ avec $n \leq m$.

On cherche une fonction $\phi = \sum_{j=1}^{n} \alpha_j \phi_j$ approchant au mieux les données y_i : $y_i \approx \phi(x_i)$, au sens de trouver un ou des vecteurs $\alpha = (\alpha_1, \cdots, \alpha_n)^T \in \mathbb{R}^n$ qui minimise

$$J(\alpha) = \left(\sum_{i=1}^{m} (\phi(x_i) - y_i)^2 \right)^{1/2} \tag{1}$$

Soit A la matrice rectangulaire $\in \mathbb{R}^{m \times n}$ de coefficients $a_{i,j} = \phi_j(x_i)$, $1 \leq i \leq m\,; 1 \leq j \leq n$. On peut montrer à titre d'exercice les résultats suivants :

- $(\phi(x_1), \cdots, \phi(x_m))^T = A\alpha$.
- $J(\alpha) = \|A\alpha - y\|_2$
- le minimum de $J(\alpha)$ est atteint en $\alpha^\star$ vérifiant le système

$$A^T A \alpha^\star = A^T y \tag{2}$$

qui admet une solution unique si $rang(A) = n$.

Application : moindre carrées polynômiale
Lorsqu'on choisit les fonctions monômes $\phi_j(x) = x^j$, on parle de la m.a.m.c. polynômiale. Par exemple pour $n = 1$, on cherche la meilleure fonction affine $\alpha_1 x + \alpha_0$ qui minimise $\left(\sum_{i=1}^{m} (\alpha_1 x_i + \alpha_0 - y_i)^2\right)^{1/2}$. On dit aussi ajustement ou lissage linéaire. Si $n = 2$, on cherche l'ajustement quadratique, ... etc.
Ainsi la matrice A pour la m;a.m.c. de degré n est une matrice $(m, n+1)$ de la forme

$$A = \begin{pmatrix} 1 & x_1 & \cdots & x_1^n \\ 1 & x_2 & \cdots & x_2^n \\ \vdots & \vdots & \cdots & \vdots \\ 1 & x_m & \cdots & x_m^n \end{pmatrix} \tag{3}$$

Mise en œuvre on développe deux modules `mcp.py` et `linamod.py` qui contienent les définitions et les tests suivant les questions dans scripts `test1.py`, `test2.py` ou `test3.py`.

Il est exigé d'implémenter le code en mode préfixage

Partie 1

Dans un premier temps on va utiliser la fonction `polyfit` de numpy pour calculer les coefficients du polynôme d'ajustement. Les deux fonctions suivantes sont à écrire dans `mcp.py.`

1. Définir une fonction `fit(`x, y, n`)` qui pour deux vecteurs x, y de même taille en arguments renvoie les coefficients du polynôme de moindre carrées de degré $n \geq 1$. Tester dans `test1.py` pour $n = 1$ et $x = (-2, -1, 0, 1, 2), y = (3/2, -1, 1, 5, 17/2)$.

2. Définir une fonction `traceFit(`x, y, n`)` qui trace la courbe en rouge du polynôme d'ajustement de degré n, ainsi que les points (x_i, y_i) en cercles noirs. Cette figure aura un titre de la forme `"Ajustement de degré` n `"` où n est remplacé par la valeur de n en mode maths.
   Tester pour les mêmes données x, y ci-dessus et $n = 1, n = 2$, puis $n = 3$.

Partie 2 On se propose de développer notre propre fonction d'ajustement et comparer les résultats avec ceux de la partie 1. Les tests sont fais dans `test2.py.` Les fonctions suivantes sont à écrire dans `mcp.py.`

1. Ecrire une fonction `matMC(`x, n`)` qui pour un array `x` et un entier $n \geq 1$ en arguments, renvoie la matrice A. Tester sur $x = (1, 2, 5, -1, 6), n = 2$.

2. Ecrire une fonction `ajustMC(`x, y, n`)` qui pour deux vecteurs x, y de même taille et un entier $n \geq 1$ en arguments, renvoie la solution du système (2).

3. Ecrire une fonction `traceMC(`x, y, n`)` qui pour des données x, y trace la courbe en rouge du polynôme d'ajustement de degré n, ainsi que les points (x_i, y_i) en cercles noirs. Cette figure aura un titre de la forme `"Ajustement de degré` n `"` où n est remplacé par la vraie valeur de n
   Tester pour $x = (-2, -1, 0, 1, 2), y = (3/2, -1, 1, 5, 17/2), n = 1, n = 2$, puis $n = 3$. Comparer avec la partie 1.

4. Ecrire dans le script `test2.py` les instructions qui mettent dans une même figure trace les ajustements en sous-figures pour x, y donnés plus haut et $n = 1, 2, 3, 4$.

Partie 3 On fera les test dans un script `test3.py`

On considère le modèle linéaire simple est défini par :

$$Y = \alpha + \beta X + \epsilon$$

avec X, Y sont des vecteurs d'observations de données liées par une relation quasi-linéaire, où ϵ représente un bruit que l'on suppose de distribution aléatoire normale d'espérance nulle.

1. Former un vecteur X de 50 données aléatoires, puis le vecteur :

   $$Y = 2 - 4X + 0.5\epsilon \qquad \text{(ml)}$$

   où ϵ est un vecteur aléatoire selon la loi normale d'espérance nulle. Pour cela on utilisera les fonctions `rand` pour x et `randn` pour ϵ de la bibliothèque `scipy`.

2. Calculer dans le script de test les coefficients d'ajustement linéaire du nuage de points X, Y du modèle (ml), trace le nuage de points (X, Y) et sa droite d'ajustement en rouge.

3. On considère le modèle (ml) avec une taille n quelconque d'obsevations X, Y. Dessiner un seul cadre le nuage de points X, Y et sa droite d'ajustement pour resp. $n = 10, 100, 1000, 10000$.

4. Ecrire dans le module `linamod.py` une fonction `nuage(X,a,b,c)` qui calcule et affiche les coefficients d'ajustement du modèle

   $$Y = a + bX + c\epsilon \qquad \text{(ml1)}$$

   puis trace le nuage de points X, Y et sa droite d'ajustement selon le modèle (ml1).

5. tester dans test3.py pour a, b, c du modèle (ml) et de taille $n = 100$ pour les observations.

Application 2 Nous considérons une équation différentielle d'ordre 2 avec conditions initiales :

$$\begin{cases} x''(t) = f(t, x(t), x'(t)) \quad t \in [t_0, t_0 + T] \\ \text{avec } x(t_0) = x0 \ ; \ x'(t_0) = xp0 \end{cases} \tag{4}$$

où $f : [t_0, t_0 + T] \times \mathbb{R}^2 \to \mathbb{R}$, $x0, xp0 \in \mathbb{R}$

1. Mettre (4) sous la forme d'un système différentiel d'ordre 1 :

$$\begin{cases} Y'(t) = F(t, Y(t)), \ t \in [t_0, t_0 + T] \\ Y(t_0) = Y_0 \end{cases} \tag{5}$$

où F est une fonction à valeur vectorielle dans $\mathbb{R}^2$ et Y_0 un vecteur initial sont à déterminer.

On se propose d'approcher (5) par la méthode d'Euler :

$$\begin{cases} Y_{i+1} = Y_i + hF(t_i, Y_i) \quad 0 \leq i \leq n-1 \\ Y_0 \text{ donné} \end{cases} \tag{6}$$

en discrétisant l'intervalle des instants $[t_0, t_0+T]$ en n parties égales avec $n \in \mathbb{N}^\star$, $h = T/n$, $t_i = t_0+ih, 0 \leq i \leq n$. On a $Y_0 = Y(t_0)$ et $Y_i \approx Y(t_i)$, $1 \leq i \leq n$.

2. Montrer que l'algorithme d'Euler pour approcher la solution de (4) peut s'écrire :

$$\forall i = 0, \cdots, n-1, \ \begin{cases} t_{i+1} = t_i + h \\ x_{i+1} = x_i + hx'_i \\ x'_{i+1} = x'_i + hf(t_i, x_i, x'_i) \end{cases} \tag{7}$$

où $x_i \approx x(t_i)$ et $x'_i \approx x'(t_i)$.

3. En se basant sur le schéma (7), écrire une fonction `Euler(f,t0,T,x0,xp0,n)` dans un module `eqdiff.py` qui
   - calcule les 2 vecteurs array de composantes $(x_i)_{i=0,\cdots,n}$, $(x'_i)_{i=0,\cdots,n}$.
   - Trace dans une seule fenêtre deux graphes : l'un est la courbe approchée de la solution x de (4). L'autre en dessous contient le diagramme de phase, c.a.d. la trajectoire qui approche la courbe paramétrée $(x(t), x'(t))$.

4. Tester dans un script `testEqdiff.py` sur les équations données suivantes et interpréter les résultats pour $n = 100, n = 1000, n = 10000$:
   (a) L'oscillateur harmonique : $\begin{cases} x'' + x = 0 & \text{pour } t \in [0, 40] \\ x(0) = 1, \ x'(0) = 0 \end{cases}$.
   (b) L'oscillateur de Van der Pol : $\begin{cases} x'' + (x^2 - 1)x' + x = 0 \quad \text{pour } t \in [0, 30] \\ x(0) = 1, \ x'(0) = 0 \end{cases}$
   (c) Pendule pesant amorti : $\begin{cases} x'' + \frac{1}{5}x' + \sin x = 0 \quad \text{pour } t \in [0, 30] \\ x(0) = 1, \ x'(0) = 0 \end{cases}$

5. Nous souhaitons maintenant travailler sur une famille de conditions initiales $((x_{0,i}, x'_{0,i}))_{1 \leq i \leq p}$ qui sera représentée par la liste $Lc = [[x_{0,1}, x'_{0,1}], \cdots, [x_{0,p}, x'_{0,p}]]$.
   Ecrire dans `eqdiff.py` une fonction `EulerPhase(f,t0,T,Lc,n)` qui applique la méthode d'Euler pour chacune des conditions initiales dans `Lc` et renvoie le tracé des trajectoires approchées des phases superposées dans le même cadre.

6. Tester dans `testEqdiff.py` pour les équations 4.b et 4.c avec $(t_0, T, n) = (0, 30, 500)$ et les conditions initiales $Lc = [[0, -1], [-3, 4], [3, 3], [3, -2]]$ pour 4.b et $Lc = [[0, 1.5], [0, 2.5], [0, 4], [0, 6], [0, 7.5]]$ pour 4.c. Ajouter une légende indiquant les conditions initiales. Commenter.

Application 3 Idée : on considère le pavé unité $\Omega = [0,1] \times [0,1]$ et le quart du disque unité $D \subset \Omega$. On simule n observations de points selon une loi uniforme dans Ω. Le rapport $\frac{aire(D)}{aire(\Omega)} = \frac{\pi}{4}$ représente la probabilité qu'un point soit dans D. Si on choisit de manière aléatoire n points dans Ω, alors à peu près $n\frac{\pi}{4}$ points tombent dans D.

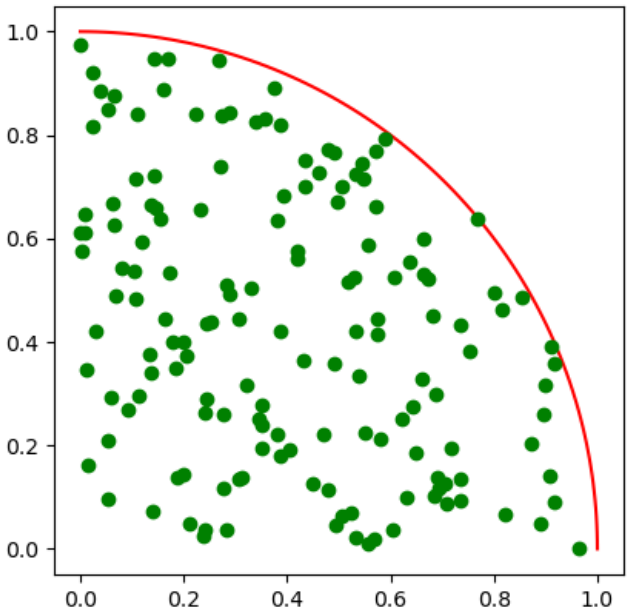
Cette méthode s'appelle approximation de monte-carlo.

1. En utilisant la fonction **`rand`** de **`numpy.`**random, former une matrice **`XY`** à 2 colonnes, où chaque colonne est remplie de $n = 200$ nombres aléatoires selon la loi uniforme sur $[0,1]$. Cette matrice représente les coordonnées des points observés.

2. Calculer exactement la proportion **`prop`** des nombres qui se trouvent à l'intérieur de D. En déduire **`app_pi`** une approximation de π. Présenter le résultat sous forme du message : **`"approximation de pi par 200 observations = ... "`**.
   Recommencer pour $n = 400, 800, \cdots$

3. On reprend la simulation pour $n = 200$. Représenter graphiquement les points générés aléatoirement dans Ω par des points noirs. avec les étiquettes **`X`** en abscisses et **`Y`** en ordonnées.

4. Réinitialiser le graphique en traçant le quart de cercle unité et ne ne gardant que les points se trouvant dans D, comme la figure



5. Ecrire les instructions qui illustrent la simulation en 4 sous-graphiques dans une seule fenêtre pour $n = 200, 500, 1000, 2000$. Mettre un titre pour chaque sous figure sous la forme $n = 200, n = 500, ...etc$, qui indique la valeur de n associée à la simulation.