Exercice 1. [Michaelis-Menten] D'après Wikipedia, l'équation de Michaelis-Menten (ou de Michaelis-Menten-Henri) permet de décrire la cinétique d'une réaction catalysée par une enzyme agissant sur un substrat unique pour donner un produit. Elle relie la vitesse de la réaction à la concentration de substrat et à des paramètres constants, caractéristiques de l'enzyme. Voici l'équation

$$v = \frac{v_m[S]}{k_s + [S]}, \tag{1}$$

où v est la vitesse initiale de réaction, v_m la vitesse initiale maximale mesurée pour une concentration saturante de substrat, k_s la constante de mi-saturation et $[S]$ la concentration en substrat. Le but est, par une méthode de moindres carrés, d'estimer les paramètres v_m et k_s d'après la série suivante de mesures de v et $[S]$:

$[S]$	1.3	1.8	3	4.5	6	8	9
v	0.07	0.13	0.22	0.275	0.335	0.35	0.36

Par sa non linéarité l'équation (1) ne semble pas rentrée dans le cadre de la régression linéaire. Une première possibilité est de faire de la régression non linéaire. Une seconde possibilité est de transformer (1) pour en faire un problème classique de moindres carrés. En effet, (1) équivaut à

$$\frac{1}{v} = \frac{k_s}{v_m}\frac{1}{[S]} + \frac{1}{v_m}. \tag{2}$$

Ainsi $\frac{1}{v}$ dépend de façon affine de $\frac{1}{[S]}$: c'est le cadre classique !

En Python (à l'aide de Numpy/Scipy/Matplotlib) écrire un programme qui
- fait la régression linéaire de $1/v$ par rapport à $1/[S]$;
- en déduit les coefficients v_m et k_s ;
- affiche sur un même graphique le nuage de points des données et la courbe obtenue.

Comme cette méthode est relativement sensible aux erreurs pour des données v petites (le passage à $1/v$ provoque des erreurs d'arrondis importants) une autre solution est d'utiliser la fonction `curve_fit`, intégrée à Scipy (version 0.8 ou supérieure). Cette fonction porte bien son nom et applique une technique de régression non linéaire. Sur le modèle

```
>>> import numpy as np
>>> from scipy.optimize import curve_fit
>>> def func(x, a, b, c):
...     return a*np.exp(-b*x) + c
>>> x = np.linspace(0,4,50)
>>> y = func(x, 2.5, 1.3, 0.5)
>>> yn = y + 0.2*np.random.normal(size=len(x))
>>> popt, pcov = curve_fit(func, x, yn)
```

où `popt` est le vecteur des données a, b et c, faire un programme Python qui
- calcule les coefficients k_m et v_m à l'aide de `curve_fit` ;
- affiche sur un même graphique le nuage de points des données et la courbe obtenue.

Comparer les deux méthodes.

Remarque. La routine `curve_fit` peut échouer dans le recherche des paramètres et retourner `[1., 1. ...]` ; c'est un problème compliqué, il y a des « minimums locaux », les bons paramètres sont très grands, etc. En cas d'échec flagrant il est possible d'aider `curve_fit` en ajoutant une entrée qui correspond au point de départ dans la recherche des paramètres. Dans l'exemple donné même si ce n'est pas utile voici une proposition

```
>>> p0=array([2.5, 1., 0.5])
>>> popt, pcov = curve_fit(func, x, yn,p0)
```

L'énergie à minimiser est différente selon que l'équation a été linéarisée (pour les moindres carrés « classiques ») ou non (moindres carrés non linéaires). Il est donc tout à fait normal d'obtenir des résultats différents.

Exercice 2. [Méthode de Jacobi] Pour résoudre certains systèmes linéaires il vaut mieux utiliser une méthode itérative, c'est-à-dire une suite de vecteurs qui converge vers la solution du système $Ax = b$.

Soit A une matrice carrée inversible de taille n et b un vecteur de $\mathbb{R}^n$. Soient L, D et U la décomposition $A = D - U - L$ où D est diagonale, U est triangulaire supérieure à diagonale nulle et L triangulaire inférieure à diagonale nulle.

Posons $M = D$ et $N = U + L$ (on a alors $A = M - N$). La méthode de Jacobi (qui n'a qu'un intérêt pédagogique) consiste à construire la suite $x^{(k)}$ de vecteur de $\mathbb{R}^n$ par

$$\begin{cases} x^{(0)} \in \mathbb{R}^n \\ Mx^{(k+1)} = Nx^{(k)} + b. \end{cases} \tag{3}$$

Comme M est diagonale, étant donné $x^{(k)}$, le vecteur $x^{(k+1)}$ est facilement et rapidement déterminé.

Faire un programme Python (Numpy, Scipy) appelé `jacobi(A,b,x0,kmax)` qui renvoie le vecteur $x^{(kmax)}$. On utilisera pour résoudre le système linéaire (3) la fonction `linsolve`, pour construire U, L et D les fonctions de Scipy `triu`, `tril` et `diag`.

Pour vérifier que le programme fonctionne il suffit de comparer la sortie de `jacobi` pour `kmax` grand (au moins 100) avec la solution du système linéaire obtenue via `linsolve`.

Dans les tests suvants, vérifier expérimentalement la convergence ou non de la méthode de Jacobi (en effet la méthode ne converge pas toujours). Un choix de $x^{(0}$ est proposé mais dans les tests essayer d'autre initialisation.

Test 1

$$A = \begin{pmatrix} 1 & 2 & -2 \\ 1 & 1 & 1 \\ 2 & 2 & 1 \end{pmatrix}, \quad b = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \quad x^{(0)} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}.$$

Test 2

$$A = \begin{pmatrix} 2 & -1 & 1 \\ 2 & 2 & 2 \\ -1 & -1 & 2 \end{pmatrix}, \quad b = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \quad x^{(0)} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}.$$

Test 3 Faire un programme qui retourne la matrice A triagonale de taille n arbitraire et vecteur b

$$A = \begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{pmatrix}, \quad b = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \\ 1 \end{pmatrix}, \quad x^{(0)} = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \\ 1 \end{pmatrix}.$$

Test pour $n = 10, 50, 100$. La convergence est-elle rapide ou lente ?

Exercice 1. [(Michaelis-Menten)^2 ou Hill] Dans le cadre de réactions enzymatiques, on propose le modèle

$$v = \frac{v_m [S]^2}{k_s^2 + [S]^2}, \tag{1}$$

où v est la vitesse initiale de réaction, v_m la vitesse initiale maximale mesurée pour une concentration saturante de substrat, k_s la constante de mi-saturation et $[S]$ la concentration en substrat. Le but est, par une méthode de moindres carrés, d'estimer les paramètres v_m et k_s d'après la série suivante de mesures de v et $[S]$:

$[S]$	1.3	1.8	3	4.5	6	8	9
v	0.07	0.13	0.22	0.275	0.335	0.35	0.36

Par sa non linéarité l'équation (1) ne semble pas rentrée dans le cadre de la régression linéaire. Une première possibilité est de faire de la régression non linéaire. Une seconde possibilité est de transformer (1) pour en faire un problème classique de moindres carrés. En effet, (1) équivaut à

$$\frac{1}{v} = \frac{k_s^2}{v_m} \frac{1}{[S]^2} + \frac{1}{v_m}. \tag{2}$$

Ainsi $\frac{1}{v}$ dépend de façon affine de $\frac{1}{[S]^2}$: c'est le cadre classique !

En Python (à l'aide de Numpy/Scipy/Matplotlib) écrire un programme qui
- fait la régression linéaire de $1/v$ par rapport à $1/[S]^2$;
- en déduit les coefficients v_m et k_s ;
- affiche sur un même graphique le nuage de points des données et la courbe obtenue.

Comme cette méthode est relativement sensible aux erreurs pour des données v petites (le passage à $1/v$ provoque des erreurs d'arrondis importants) une autre solution est d'utiliser la fonction `curve_fit`, intégrée à Scipy (version 0.8 ou supérieure). Cette fonction porte bien son nom et applique une technique de régression non linéaire. Sur le modèle

```
>>> import numpy as np
>>> from scipy.optimize import curve_fit
>>> def func(x, a, b, c):
...     return a*np.exp(-b*x) + c
>>> x = np.linspace(0,4,50)
>>> y = func(x, 2.5, 1.3, 0.5)
>>> yn = y + 0.2*np.random.normal(size=len(x))
>>> popt, pcov = curve_fit(func, x, yn)
```

où `popt` est le vecteur des données a, b et c, faire un programme Python qui
- calcule les coefficients k_m et v_m à l'aide de `curve_fit` ;
- affiche sur un même graphique le nuage de points des données et la courbe obtenue.

Comparer les deux méthodes.

Exercice 2. [Méthode de Jacobi] Pour résoudre certains systèmes linéaires il vaut mieux utiliser une méthode itérative, c'est-à-dire une suite de vecteurs qui converge vers la solution du système $Ax = b$.

Soit A une matrice carrée inversible de taille n et b un vecteur de $\mathbb{R}^n$. Soient L, D et U la décomposition $A = D - U - L$ où D est diagonale, U est triangulaire supérieure à diagonale nulle et L triangulaire inférieure à diagonale nulle.

Posons $M = D$ et $N = U + L$ (on a alors $A = M - N$). La méthode de Jacobi (qui n'a qu'un intérêt pédagogique) consiste à construire la suite $x^{(k)}$ de vecteur de $\mathbb{R}^n$ par

$$\begin{cases} x^{(0)} \in \mathbb{R}^n \\ Mx^{(k+1)} = Nx^{(k)} + b. \end{cases} \tag{3}$$

Comme M est diagonale, étant donné $x^{(k)}$, le vecteur $x^{(k+1)}$ est facilement et rapidement déterminé.

Faire un programme Python (Numpy, Scipy) appelé `jacobi(A,b,x0,kmax)` qui renvoie le vecteur $x^{(kmax)}$. On utilisera pour résoudre le système linéaire (3) la fonction `linsolve`, pour construire U, L et D les fonctions de Scipy `triu`, `tril` et `diag`.

Pour vérifier que le programme fonctionne il suffit de comparer la sortie de `jacobi` pour `kmax` grand (au moins 100) avec la solution du système linéaire obtenue via `linsolve`.

Dans les tests suvants, vérifier expérimentalement la convergence ou non de la méthode de Jacobi (en effet la méthode ne converge pas toujours). Un choix de $x^{(0}$ est proposé mais dans les tests essayer d'autre initialisation.

Test 1

$$A = \begin{pmatrix} 1 & 2 & -2 \\ 1 & 1 & 1 \\ 2 & 2 & 1 \end{pmatrix}, \quad b = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \quad x^{(0)} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}.$$

Test 2

$$A = \begin{pmatrix} 2 & -1 & 1 \\ 2 & 2 & 2 \\ -1 & -1 & 2 \end{pmatrix}, \quad b = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \quad x^{(0)} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}.$$

Test 3 Faire un programme qui retourne la matrice A triagonale de taille n arbitraire et vecteur b

$$A = \begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{pmatrix}, \quad b = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \\ 1 \end{pmatrix}, \quad x^{(0)} = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \\ 1 \end{pmatrix}.$$

Test pour $n = 10, 50, 100$. La convergence est-elle rapide ou lente ?

Exercice 1. [(Michaelis-Meten)^3 ou Hill] On modélise certaines réactions enzymatiques par l'équation

$$v = \frac{k_m [S]^3}{K + [S]^3}, \tag{1}$$

où v est la vitesse initiale de réaction, $[S]$ la concentration en substrat et k_m et K des paramètres à déterminer. Le but est, par une méthode de moindres carrés, d'estimer les paramètres K et k_s d'après la série suivante de mesures de v et $[S]$:

$[S]$	6.078×10^{-11}	7.595×10^{-9}	6.063×10^{-8}	5.788×10^{-6}	
v	0.01	0.05	0.1	0.5	
$[S]$	1.737×10^{-5}	2.423×10^{-5}	2.430×10^{-5}	2.431×10^{-5}	2.431×10^{-5}
v	1	5	10	50	100

Par sa non linéarité l'équation (1) ne semble pas rentrée dans le cadre de la régression linéaire. Une première possibilité est de faire de la régression non linéaire. Une seconde possibilité est de transformer (1) pour en faire un problème classique de moindres carrés. En effet, (1) équivaut à

$$\frac{1}{v} = \frac{K}{k_m} \frac{1}{[S]^3} + \frac{1}{k_m}. \tag{2}$$

Ainsi $\frac{1}{v}$ dépend de façon affine de $\frac{1}{[S]^3}$: c'est le cadre classique !

En Python (à l'aide de Numpy/Scipy/Matplotlib) écrire un programme qui
- fait la régression linéaire de $1/v$ par rapport à $1/[S]^3$;
- en déduit les coefficients K et k_s ;
- affiche sur un même graphique le nuage de points des données et la courbe obtenue.

Comme cette méthode est relativement sensible aux erreurs pour des données v petites (le passage à $1/v$ provoque des erreurs d'arrondis importants) une autre solution est d'utiliser la fonction `curve_fit`, intégrée à Scipy (version 0.8 ou supérieure). Cette fonction porte bien son nom et applique une technique de régression non linéaire. Sur le modèle

```
>>> import numpy as np
>>> from scipy.optimize import curve_fit
>>> def func(x, a, b, c):
...     return a*np.exp(-b*x) + c
>>> x = np.linspace(0,4,50)
>>> y = func(x, 2.5, 1.3, 0.5)
>>> yn = y + 0.2*np.random.normal(size=len(x))
>>> popt, pcov = curve_fit(func, x, yn)
```

où `popt` est le vecteur des données a, b et c, faire un programme Python qui
- calcule les coefficients k_m et K à l'aide de `curve_fit` ;
- affiche sur un même graphique le nuage de points des données et la courbe obtenue.

Comparer les deux méthodes.

Exercice 2. [Méthode de Gauss-Seidel] Pour résoudre certains systèmes linéaires il vaut mieux utiliser une méthode itérative, c'est-à-dire une suite de vecteurs qui converge vers la solution du système $Ax = b$.

Soit A une matrice carrée inversible de taille n et b un vecteur de $\mathbb{R}^n$. Soient L, D et U la décomposition $A = D - U - L$ où D est diagonale, U est triangulaire supérieure à diagonale nulle et L triangulaire inférieure à diagonale nulle.

Posons $M = D - L$ et $N = U$ (on a alors $A = M - N$). La méthode de Gauss-Seidel consiste à construire la suite $x^{(k)}$ de vecteur de $\mathbb{R}^n$ par

$$\begin{cases} x^{(0)} \in \mathbb{R}^n \\ Mx^{(k+1)} = Nx^{(k)} + b. \end{cases} \tag{3}$$

Comme M est triangulaire, étant donné $x^{(k)}$, le vecteur $x^{(k+1)}$ est facilement et rapidement déterminé (algorithme de remontée, de l'order de n^2 opérations)

Faire un programme Python (Numpy, Scipy) appelé `gaussseidel(A,b,x0,kmax)` qui renvoie le vecteur $x^{(kmax)}$. On utilisera pour résoudre le système linéaire (3) la fonction `linsolve`, pour construire U, L et D les fonctions de Scipy `triu`, `tril` et `diag`.

Pour vérifier que le programme fonctionne il suffit de comparer la sortie de `gausseidel` pour `kmax` grand (au moins 100) avec la solution du système linéaire obtenue via `linsolve`.

Dans les tests suvants, vérifier expérimentalement la convergence ou non de la méthode de Gauss-Seidel (en effet la méthode ne converge pas toujours). Un choix de $x^{(0}$ est proposé mais dans les tests essayer d'autre initialisation.

Test 1

$$A = \begin{pmatrix} 1 & 2 & -2 \\ 1 & 1 & 1 \\ 2 & 2 & 1 \end{pmatrix}, \quad b = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \quad x^{(0)} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}.$$

Test 2

$$A = \begin{pmatrix} 2 & -1 & 1 \\ 2 & 2 & 2 \\ -1 & -1 & 2 \end{pmatrix}, \quad b = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \quad x^{(0)} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}.$$

Test 3 Faire un programme qui retourne la matrice A triagonale de taille n arbitraire et vecteur b

$$A = \begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{pmatrix}, \quad b = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \\ 1 \end{pmatrix}, \quad x^{(0)} = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \\ 1 \end{pmatrix}.$$

Test pour $n = 10, 50, 100$. La convergence est-elle rapide ou lente ?

Exercice 1. Dans un cylindre on mesure la vitesse de l'écoulement dans plusieurs situations où varient le diamètre et l'inclinaison du cylindre.

Expérience	Diamètre	Pente	Flux
1	0.3	0.001	0.04
2	0.6	0.001	0.24
3	0.9	0.001	0.69
4	0.3	0.01	0.13
5	0.6	0.01	0.82
6	0.9	0.01	2.38
7	0.3	0.05	0.31
8	0.6	0.05	1.95
9	0.9	0.05	5.66

On suppose que le modèle suit l'équation

$$Q = \alpha_0 D^{\alpha_1} S^{\alpha_2}, \tag{1}$$

où Q est la vitesse de l'écoulement, D le diamètre du tube circulaire et S la pente.

On souhaite déterminer par une méthode de moindres carrés les coefficients α_0, α_1 et α_2. Comme la loi est non linéaire il faut la transformer. La fonction logarithme nous donne la relation

$$\ln(Q) = \ln(\alpha_0) + \alpha_1 \ln(D) + \alpha_2 \ln(S). \tag{2}$$

À l'aide de l'équation (2), faire un programme Python qui
- construit la matrice A de taille 3×9 à partir des données du tableau ;
- par la méthode des moindres carrés donne α_0, α_1 et α_2 ;

Exercice 2. [Méthode de Gauss-Seidel] Pour résoudre certains systèmes linéaires il vaut mieux utiliser une méthode itérative, c'est-à-dire une suite de vecteurs qui converge vers la solution du système $Ax = b$.

Soit A une matrice carrée inversible de taille n et b un vecteur de $\mathbb{R}^n$. Soient L, D et U la décomposition $A = D - U - L$ où D est diagonale, U est triangulaire supérieure à diagonale nulle et L triangulaire inférieure à diagonale nulle.

Posons $M = D - L$ et $N = U$ (on a alors $A = M - N$). La méthode de Gauss-Seidel consiste à construire la suite $x^{(k)}$ de vecteur de $\mathbb{R}^n$ par

$$\begin{cases} x^{(0)} \in \mathbb{R}^n \\ Mx^{(k+1)} = Nx^{(k)} + b. \end{cases} \tag{3}$$

Comme M est triangulaire, étant donné $x^{(k)}$, le vecteur $x^{(k+1)}$ est facilement et rapidement déterminé (algorithme de remontée, de l'order de n^2 opérations)

Faire un programme Python (Numpy, Scipy) appelé `gaussseidel(A,b,x0,kmax)` qui renvoie le vecteur $x^{(kmax)}$. On utilisera pour résoudre le système linéaire (3) la fonction `linsolve`, pour construire U, L et D les fonctions de Scipy `triu`, `tril` et `diag`.

Pour vérifier que le programme fonctionne il suffit de comparer la sortie de `gausseidel` pour `kmax` grand (au moins 100) avec la solution du système linéaire obtenue via `linsolve`.

Dans les tests suivants, vérifier expérimentalement la convergence ou non de la méthode de Gauss-Seidel (en effet la méthode ne converge pas toujours). Un choix de $x^{(0}$ est proposé mais dans les tests essayer d'autre initialisation.

Test 1

$$A = \begin{pmatrix} 1 & 2 & -2 \\ 1 & 1 & 1 \\ 2 & 2 & 1 \end{pmatrix}, \quad b = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \quad x^{(0)} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}.$$

Test 2

$$A = \begin{pmatrix} 2 & -1 & 1 \\ 2 & 2 & 2 \\ -1 & -1 & 2 \end{pmatrix}, \quad b = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \quad x^{(0)} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}.$$

Test 3 Faire un programme qui retourne la matrice A triagonale de taille n arbitraire et vecteur b

$$A = \begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{pmatrix}, \quad b = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \\ 1 \end{pmatrix}, \quad x^{(0)} = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \\ 1 \end{pmatrix}.$$

Test pour $n = 10, 50, 100$. La convergence est-elle rapide ou lente ?

Université de Rouen
M1 IBIOM
Année 2010-2011

Mathématiques Appliquées
Exercices Python, sujet 5

Exercice 1. On suppose que les données x et y sont reliées par la loi

$$y = \alpha x \exp(\beta x). \tag{1}$$

À partir des données du tableau on souhaite estimer α et β.

x	0.1	0.2	0.4	0.6	0.9	1.3	1.5	1.7	1.8
y	0.75	1.25	1.45	1.25	0.85	0.55	0.35	0.28	0.18

Une première méthode consiste à « linéariser » l'équation (1) de façon à se placer dans le cadre classique de la régression linéaire. En effet grâce à la fonction logarithme, l'équation (1) est équivalente à

$$\ln(y) - \ln(x) = \ln(\alpha) + \beta x \tag{2}$$

En Python (à l'aide de Numpy/Scipy/Matplotlib) écrire un programme qui
- à l'aide de la régression linéaire et l'équation (2) détermine les coefficients α et β ;
- affiche sur un même graphique le nuage de points des données et la courbe obtenue.

Comme la loi est non linéaire une autre solution est d'utiliser une technique de régression non linéaire avec la fonction `curve_fit`, intégrée à Scipy (version 0.8 ou supérieure). Cette fonction porte bien son nom et applique une technique de régression non linéaire. Sur le modèle

```
>>> import numpy as np
>>> from scipy.optimize import curve_fit
>>> def func(x, a, b, c):
...     return a*np.exp(-b*x) + c
>>> x = np.linspace(0,4,50)
>>> y = func(x, 2.5, 1.3, 0.5)
>>> yn = y + 0.2*np.random.normal(size=len(x))
>>> popt, pcov = curve_fit(func, x, yn)
```

où `popt` est le vecteur des données a, b et c, faire un programme Python qui
- calcule les coefficients α et β à l'aide de `curve_fit` et selon le modèle (1)
- affiche sur un même graphique le nuage de points des données et la courbe obtenue.

Comparer les deux méthodes.

Exercice 2. [Relaxation ou presque.] Pour résoudre certains systèmes linéaires il vaut mieux utiliser une méthode itérative, c'est-à-dire une suite de vecteurs qui converge vers la solution du système $Ax = b$.

Soit A une matrice carrée inversible de taille n et b un vecteur de $\mathbb{R}^n$. Soient L, D et U la décomposition $A = D - U - L$ où D est diagonale, U est triangulaire supérieure à diagonale nulle et L triangulaire inférieure à diagonale nulle.

Posons $M = \frac{2}{3}D - L$ et $N = -\frac{1}{3}D + U$ (on a alors $A = M - N$). La méthode de relaxation[1] consiste à construire la suite $x^{(k)}$ de vecteur de $\mathbb{R}^n$ par

$$\begin{cases} x^{(0)} \in \mathbb{R}^n \\ Mx^{(k+1)} = Nx^{(k)} + b. \end{cases} \tag{3}$$

Comme M est triangulaire, étant donné $x^{(k)}$, le vecteur $x^{(k+1)}$ est facilement et rapidement déterminé (algorithme de remontée, de l'order de n^2 opérations)

Faire un programme Python (Numpy, Scipy) appelé `pseudorelax(A,b,x0,kmax)` qui renvoie le vecteur $x^{(kmax)}$. On utilisera pour résoudre le système linéaire (3) la fonction `linsolve`, pour construire U, L et D les fonctions de Scipy `triu`, `tril` et `diag`.

Pour vérifier que le programme fonctionne il suffit de comparer la sortie de `pseudorelax` pour `kmax` grand (au moins 100) avec la solution du système linéaire obtenue via `linsolve`.

1. la véritable relaxation consiste à poser $M = \frac{1}{\omega}D - L$ et $N = \frac{1-\omega}{\omega}$ avec $0 < \omega < 2$ généralement

Dans les tests suvants, vérifier expérimentalement la convergence ou non de la méthode de (pseudo)relaxation (en effet la méthode ne converge pas toujours). Un choix de $x^{(0}$ est proposé mais dans les tests essayer d'autre initialisation.

Test 1

$$A = \begin{pmatrix} 1 & 2 & -2 \\ 1 & 1 & 1 \\ 2 & 2 & 1 \end{pmatrix}, \quad b = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \quad x^{(0)} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}.$$

Test 2

$$A = \begin{pmatrix} 2 & -1 & 1 \\ 2 & 2 & 2 \\ -1 & -1 & 2 \end{pmatrix}, \quad b = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \quad x^{(0)} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}.$$

Test 3 Faire un programme qui retourne la matrice A triagonale de taille n arbitraire et vecteur b

$$A = \begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{pmatrix}, \quad b = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \\ 1 \end{pmatrix}, \quad x^{(0)} = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \\ 1 \end{pmatrix}.$$

Test pour $n = 10, 50, 100$. La convergence est-elle rapide ou lente ?

Université de Rouen
M1 IBIOM
Année 2010-2011

Mathématiques Appliquées
Exercices Python, sujet 6

Exercice 1. [Sinus] Voici le relevé des températures maximales à certains jours de l'année (un jour par mois excepté le mois de juillet)

Jour de l'année	14	45	75	105	135	165	225	255	285	315	345
T, °C	18.9	21.1	23.3	27.8	32.2	37.2	36.1	34.4	29.4	23.3	18.9

On émet l'hypothèse que la température suit le modèle

$$T(t) = A_0 + C_1 \cos(\omega t + \theta), \tag{1}$$

et on désire par une méthode de moindres carrés estimer les paramètres A_0, C_1 et θ. Comme notre évolution de température est de période une année de 360 jours (12 mois de 30 jours !) on posera $\omega = 2\pi/360$. On linéarise l'équation (1) :

$$T(t) = A_0 + A_1 \cos(\omega t) + B_1 \sin(\omega t) \tag{2}$$

avec

$$A_1 = C_1 \cos(\theta), \quad B_1 = -C_1 \sin(\theta). \tag{3}$$

On donc les équations

$$\theta = \arctan\Big(-\frac{B_1}{A_1}\Big) \quad (\text{si } A_1 < 0 \text{ ajouter } \pi) \tag{4}$$

et

$$C_1 = \sqrt{A_1^2 + B_1^2}. \tag{5}$$

Avec l'équation (2) nous sommes dans le cadre standard des problèmes de moindres carrés. En Python (à l'aide de Numpy/Scipy/Matplotlib) écrire un programme qui

- à l'aide de la régression linéaire et l'équation (2) détermine les coefficients A_0, A_1 et B_1 ;
- calcul C_1 et θ
- affiche sur un même graphique le nuage de points des données et la courbe obtenue.
- donner une estimation de la température mi-juillet ($t = 195$).

Exercice 2. [Relaxation ou presque.] Pour résoudre certains systèmes linéaires il vaut mieux utiliser une méthode itérative, c'est-à-dire une suite de vecteurs qui converge vers la solution du système $Ax = b$.

Soit A une matrice carrée inversible de taille n et b un vecteur de $\mathbb{R}^n$. Soient L, D et U la décomposition $A = D - U - L$ où D est diagonale, U est triangulaire supérieure à diagonale nulle et L triangulaire inférieure à diagonale nulle.

Posons $M = \frac{2}{3}D - L$ et $N = -\frac{1}{3}D + U$ (on a alors $A = M - N$). La méthode de relaxation[2] consiste à construire la suite $x^{(k)}$ de vecteur de $\mathbb{R}^n$ par

$$\begin{cases} x^{(0)} \in \mathbb{R}^n \\ Mx^{(k+1)} = Nx^{(k)} + b. \end{cases} \tag{6}$$

Comme M est triangulaire, étant donné $x^{(k)}$, le vecteur $x^{(k+1)}$ est facilement et rapidement déterminé (algorithme de remontée, de l'order de n^2 opérations)

Faire un programme Python (Numpy, Scipy) appelé `pseudorelax(A,b,x0,kmax)` qui renvoie le vecteur $x^{(kmax)}$. On utilisera pour résoudre le système linéaire (6) la fonction `linsolve`, pour construire U, L et D les fonctions de Scipy `triu`, `tril` et `diag`.

Pour vérifier que le programme fonctionne il suffit de comparer la sortie de `pseudorelax` pour `kmax` grand (au moins 100) avec la solution du système linéaire obtenue via `linsolve`.

Dans les tests suvants, vérifier expérimentalement la convergence ou non de la méthode de (pseudo)relaxation (en effet la méthode ne converge pas toujours). Un choix de $x^{(0}$ est proposé mais dans les tests essayer d'autre initialisation.

2. la véritable relaxation consiste à poser $M = \frac{1}{\omega}D - L$ et $N = \frac{1-\omega}{\omega}$ avec $0 < \omega < 2$ généralement

Test 1

$$A = \begin{pmatrix} 1 & 2 & -2 \\ 1 & 1 & 1 \\ 2 & 2 & 1 \end{pmatrix}, \quad b = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \quad x^{(0)} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}.$$

Test 2

$$A = \begin{pmatrix} 2 & -1 & 1 \\ 2 & 2 & 2 \\ -1 & -1 & 2 \end{pmatrix}, \quad b = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \quad x^{(0)} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}.$$

Test 3 Faire un programme qui retourne la matrice A triagonale de taille n arbitraire et vecteur b

$$A = \begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{pmatrix}, \quad b = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \\ 1 \end{pmatrix}, \quad x^{(0)} = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \\ 1 \end{pmatrix}.$$

Test pour $n = 10, 50, 100$. La convergence est-elle rapide ou lente ?

Exercice 1. L'effet de la radiation solaire sur le taux de photosynthèse de plantes aquatiques est modélisé par l'équation

$$P = P_m \frac{I}{I_{sat}} \exp\Big(-\frac{I}{I_{sat}} + 1\Big), \tag{1}$$

où P est le taux de photosynthèse, P_m le taux maximale de photosynthèse, I la radiation solaire et I_{sat} la radiation solaire optimale. À partir des données

I	50	80	130	200	250	350	450	550	700
P	99	177	202	248	229	219	173	142	72

on désire evaluer par une méthode des moindres carrés les constantes I_{sat} et P_m.

La première méthode consiste à « linéariser » (1) avec la fonction logarithme : (1) est équivalente à

$$\ln(P) - \ln(I) = \ln(P_m) - \ln(I_{sat}) - \frac{I}{I_{sat}} + 1 \tag{2}$$

et la quantité $\ln(P) - \ln(I)$ dépend linéairement de I.

En Python (à l'aide de Numpy/Scipy/Matplotlib) écrire un programme qui
- à l'aide de la régression linéaire et l'équation (2) détermine les coefficients P_m et I_{sat} ;
- affiche sur un même graphique le nuage de points des données et la courbe obtenue.

Comme la loi est non linéaire une autre solution est d'utiliser une technique de régression non linéaire avec la fonction `curve_fit`, intégrée à Scipy (version 0.8 ou supérieure). Cette fonction porte bien son nom et applique une technique de régression non linéaire. Sur le modèle

```
>>> import numpy as np
>>> from scipy.optimize import curve_fit
>>> def func(x, a, b, c):
...     return a*np.exp(-b*x) + c
>>> x = np.linspace(0,4,50)
>>> y = func(x, 2.5, 1.3, 0.5)
>>> yn = y + 0.2*np.random.normal(size=len(x))
>>> popt, pcov = curve_fit(func, x, yn)
```

où `popt` est le vecteur des données a, b et c, faire un programme Python qui
- calcule les coefficients P_m et I_{sat} à l'aide de `curve_fit` et selon l'équation modèle (1)
- affiche sur un même graphique le nuage de points des données et la courbe obtenue.

Comparer les deux méthodes.

Exercice 2. [Relaxation ou presque.] Pour résoudre certains systèmes linéaires il vaut mieux utiliser une méthode itérative, c'est-à-dire une suite de vecteurs qui converge vers la solution du système $Ax = b$.

Soit A une matrice carrée inversible de taille n et b un vecteur de $\mathbb{R}^n$. Soient L, D et U la décomposition $A = D - U - L$ où D est diagonale, U est triangulaire supérieure à diagonale nulle et L triangulaire inférieure à diagonale nulle.

Posons $M = \frac{2}{3}D - L$ et $N = -\frac{1}{3}D + U$ (on a alors $A = M - N$). La méthode de relaxation[3] consiste à construire la suite $x^{(k)}$ de vecteur de $\mathbb{R}^n$ par

$$\begin{cases} x^{(0)} \in \mathbb{R}^n \\ Mx^{(k+1)} = Nx^{(k)} + b. \end{cases} \tag{3}$$

Comme M est triangulaire, étant donné $x^{(k)}$, le vecteur $x^{(k+1)}$ est facilement et rapidement déterminé (algorithme de remontée, de l'order de n^2 opérations)

Faire un programme Python (Numpy, Scipy) appelé `pseudorelax(A,b,x0,kmax)` qui renvoie le vecteur $x^{(kmax)}$. On utilisera pour résoudre le système linéaire (3) la fonction `linsolve`, pour construire U, L et D les fonctions de Scipy `triu`, `tril` et `diag`.

3. la véritable relaxation consiste à poser $M = \frac{1}{\omega}D - L$ et $N = \frac{1-\omega}{\omega}$ avec $0 < \omega < 2$ généralement

Pour vérifier que le programme fonctionne il suffit de comparer la sortie de `pseudorelax` pour `kmax` grand (au moins 100) avec la solution du système linéaire obtenue via `linsolve`.

Dans les tests suvants, vérifier expérimentalement la convergence ou non de la méthode de (pseudo)relaxation (en effet la méthode ne converge pas toujours). Un choix de $x^{(0}$ est proposé mais dans les tests essayer d'autre initialisation.

Test 1

$$A = \begin{pmatrix} 1 & 2 & -2 \\ 1 & 1 & 1 \\ 2 & 2 & 1 \end{pmatrix}, \quad b = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \quad x^{(0)} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}.$$

Test 2

$$A = \begin{pmatrix} 2 & -1 & 1 \\ 2 & 2 & 2 \\ -1 & -1 & 2 \end{pmatrix}, \quad b = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \quad x^{(0)} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}.$$

Test 3 Faire un programme qui retourne la matrice A triagonale de taille n arbitraire et vecteur b

$$A = \begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{pmatrix}, \quad b = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \\ 1 \end{pmatrix}, \quad x^{(0)} = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \\ 1 \end{pmatrix}.$$

Test pour $n = 10, 50, 100$. La convergence est-elle rapide ou lente ?