**Introduction à l'analyse numérique**
Année : 2018-2019
Formation : L2 Mathématiques

# TP noté 2 : Résolution de système linéaire, circuit RLC et exponentiation rapide

Ce sujet est composé en trois parties indépendantes. Pour chaque partie il faudra créer deux fichiers dans lesquels vous enregistrerez votre code : un fichier en .sci qui contiendra toutes les fonctions de la partie et un fichier en .sce qui contiendra les autres commandes. A la fin du TP vous devez envoyer vos fichiers par mail à l'adresse suivante : florian.le-manach@math.u-bordeaux.fr. N'oubliez pas d'indiquer votre nom et prénom (et celui de votre binôme si vous faites le TP à deux) dans le mail.

## 1 Résolution de système linéaire

Nous nous intéressons à plusieurs méthodes itératives de résolution d'un système d'équation linéaire $Ax = b$ où $A \in \mathcal{M}_n(\mathbb{R})$ et $b \in \mathbb{R}^n$. Pour ces méthodes, on construit une suite de vecteurs

$$\left\{ \begin{array}{rcl} x^0 & = & 0 \\ \forall n \in \mathbb{N}, \quad x^{n+1} & = & \phi(x^n) \end{array} \right.$$

pour une certaine application $\phi$ bien choisie et dont le calcul est facile, rapide et numériquement stable. Lorsque la suite $(x^n)_{n\in\mathbb{N}}$ converge (ce qui n'arrive pas toujours), sa limite $x$ vérifie alors $Ax = b$ et ses termes, pour un rang $n$ assez grand, fournissent une approximation convenable de la solution du système.

Plus précisément, soit $A$ une matrice carrée. On définit $D$ comme la partie diagonale de la matrice $A$, $-E$ (attention au signe !) comme sa partie triangulaire inférieure, et $-F$ comme sa partie triangulaire supérieure.

Nous étudions ici les algorithmes de Jacobi et de Gauss-Seidel. Pour ces deux méthodes de résolution, on construit une suite $(x^k)_{k\geq 0}$ que l'on souhaite faire converger vers la solution $x \in \mathbb{R}^n$ du problème. Pour tout $k \geq 0$, le $(k+1)$-ème terme de la suite est défini de la manière suivante :

$$\begin{aligned} x^{k+1} &= D^{-1}(b + (E+F)x^k) && \textit{(Jacobi)} \\ x^{k+1} &= (D-E)^{-1}(b + Fx^k) && \textit{(Gauss-Seidel)} \end{aligned}$$

De manière équivalente, le $i$-ème élément du vecteur $x^{k+1}$ s'écrit :

$$x_i^{k+1} = \frac{1}{a_{ii}}\Big(b_i - \sum_{j=1, j\neq i}^{n} a_{ij} x_j^{\mathbf{k}}\Big) \qquad \text{(Jacobi)} \qquad (1a)$$

$$x_i^{k+1} = \frac{1}{a_{ii}}\Big(b_i - \sum_{j=1, j<i}^{n} a_{ij} x_j^{\mathbf{k+1}} - \sum_{j=1, j>i}^{n} a_{ij} x_j^{\mathbf{k}}\Big) \qquad \text{(Gauss-Seidel)} \qquad (1b)$$

Si l'algorithme de Jacobi n'utilise que les valeurs de $x^k$, l'algorithme de Gauss-Seidel utilise le fait que les termes $x_j^{k+1}$ ont déjà été calculé précédemment si $j < i$. Ceci améliore considérablement la vitesse de convergence de l'algorithme.

1. Définissez une fonction function D = diagonale(A) qui prend en paramètre une matrice carrée $A$ et qui renvoie la liste $D$ des éléments diagonaux de la matrice $A$.
   C'est-à-dire si $A = (a_{i,j})_{1\leq i,j\leq n}$ alors D = $[a_{1,1}\ a_{2,2}\ \cdots\ a_{n,n}]$.
   On rappelle que la commande size permet d'obtenir la taille d'une matrice.

**2.** Définissez une fonction `function M = EplusF(A)` qui prend en paramètre une matrice carrée $A$ et qui renvoie la matrice $E + F$ définie ci-dessus dans la méthode de Jacobi.

**3.** Définissez la fonction `function s=normInf(x)` qui prend en paramètre un vecteur $x$ et qui renvoie $\|x\|_\infty$ (c'est-à-dire la maximum des valeurs absolues des coefficients de $x$). Dans cette question la seule fonction Scilab autorisée est la fonction `abs`.

**4.** A l'aide des fonctions précédentes, écrivez la fonction : `function x = Jacobi(A,b,tol)`, où $A \in \mathcal{M}_n(\mathbb{R})$, $b \in \mathbb{R}^n$ et qui renvoie $x \in \mathbb{R}^n$ la solution du système linéaire $Ax = b$. La tolérance `tol` est utilisée pour le critère d'arrêt : $\|Ax - b\|_\infty \leq tol$.
Remarque : Attention le résultat `D` de la fonction `diagonale` est une liste et non une matrice carrée. Il n'est donc pas possible de calculer l'inverse (matricielle) `D ^-1` mais il est possible de calculer la liste des inverse de la liste `D` en faisant `D . ^-1`

**5.** Testez l'algorithme de Jacobi avec les paramètres suivants :

$$A = \begin{pmatrix} 5 & 0 & 3 \\ 1 & -9 & 1 \\ -2 & 2 & 6 \end{pmatrix},\ b = \begin{pmatrix} 1 \\ -3 \\ 0 \end{pmatrix},\ \text{ et } tol = 10^{-8}.$$

Vous pourrez comparer vos résultats avec la commande $A\backslash b$ de Scilab qui calcule $x = A^{-1}b$ par une méthode directe d'inversion matricielle (très couteuse si n est grand).

**6.** A l'aide de la formule (1b), écrivez la fonction : `function x = GaussSeidel(A,b,tol)` qui résout l'équation $Ax = b$ par la méthode de Gauss-Seidel.

**7.** Testez l'algorithme de Gauss-Seidel pour le même exemple. Il doit évidemment donner les mêmes résultats !

On s'intéresse maintenant à la résolution numérique d'un système linéaire $Ax = b$ pour lequel la matrice $A \in \mathcal{M}_n(\mathbb{R})$ est tridiagonale (à coefficient constant par diagonale), et s'écrit sous la forme :

$$A = A(c_1, c_2, c_3) = \begin{pmatrix} \mathbf{c_1} & \mathbf{c_3} & 0 & \cdots & 0 & \mathbf{c_2} \\ \mathbf{c_2} & \ddots & \ddots & \ddots & & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & & \ddots & \ddots & \ddots & \mathbf{c_3} \\ \mathbf{c_3} & 0 & \dots & 0 & \mathbf{c_2} & \mathbf{c_1} \end{pmatrix}$$

**8.** Définissez la fonction `function A = tridiag(c1,c2,c3,n)` qui renvoie la matrice $A = A(c_1, c_2, c_3)$ de taille $n$.

**9.** A l'aide de la fonction précédente et de l'algorithme de Jacobi (ou de Gauss-Seidel) résolvez l'équation $Ax = b$ avec

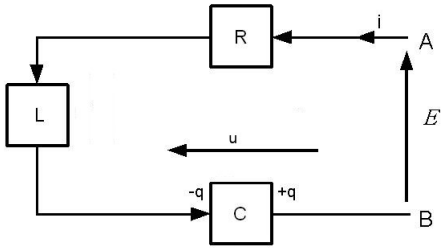
$$A = \begin{pmatrix} 10 & -3 & 0 & 0 & 4 \\ 4 & 10 & -3 & 0 & 0 \\ 0 & 4 & 10 & -3 & 0 \\ 0 & 0 & 4 & 10 & -3 \\ -3 & 0 & 0 & 4 & 10 \end{pmatrix},\ b = \begin{pmatrix} 1 \\ 2 \\ -4 \\ 0 \\ -3 \end{pmatrix},\ \text{ et } tol = 10^{-8}.$$

On pourra comparer ce résultat avec la commande $A\backslash b$ de Scilab.

# 2 Résolution d'une équation différentielle

---

Dans cette partie on cherche à déterminer numériquement l'évolution, au cours du temps, de la tension au borne d'un condensateur d'un circuit RLC.




Si on note $u$ cette tension alors $u$ vérifie l'équation différentielle suivante

$$LCu''(t) + RCu'(t) + u(t) = E$$

avec $u(0) = 0$ et $u'(0) = 0$. On prendra dans cet exercice $L = 30$, $C = 3$, $R = 1$ et $E = 12$.

En considérant

$$Y(t) = \begin{pmatrix} u(t) \\ u'(t) \end{pmatrix}$$

résolvez l'équation différentielle à l'aide de la méthode de Runge-Kutta d'ordre 2 (RK2) avec un pas de temps $dt = 0,1$ et jusqu'au temps final $t_f = 300$. Affichez sur un graphique la courbe représentative de la tension $u$. Remarque : si vous ne vous rappelez pas de la méthode RK2 utilisez la méthode d'Euler explicite (cela rapporte moins de points).

Question bonus : Donnez en commentaire la solution exacte $u$ de l'équation différentielle.

# 3 Calcul de puissance

---

L'objectif de cette partie est de construire une fonction puissance qui permet, pour $x \in \mathbb{R}$ et $n \in \mathbb{N}$, de calculer $x^n$. Évidemment dans toute cette partie il sera interdit d'utiliser la commande Scilab `x^n` pour calculer $x^n$.

Les trois première questions peuvent être traitées de manière indépendante.

Dans un premier temps on va construire une fonction puissance de manière naïve qui effectuera $n$ (ou $n-1$) multiplications pour calculer $x^n$.

**1.** Définissez la fonction `function r=puiss1(x,n)` qui prend en paramètre un réel $x$ et un entier naturel $n$ et qui renvoie $x^n$. On rappelle qu'il est interdit d'utiliser le symbole ^ dans cette fonction.

On cherche maintenant a améliorer le calcul de $x^n$ en effectuant le moins d'opération possible. Pour cela on s'intéresse à la décomposition en base 2 de $n$.

On rapelle que tout nombre entier $n$ se décompose de manière unique en

$$n = \sum_{i=0}^{k} c_i 2^i = c_0 + c_1 \times 2 + c_2 \times 2^2 + \cdots + c_k 2^k$$

avec $c_i = 0$ ou $c_i = 1$.

**2.** Définissez la fonction `function T=decompo2(n)` qui prend en paramètre un entier naturel $n$ et qui renvoie la liste `T`=[$c_0$, $c_1$,..., $c_k$] de sa décomposition en base 2.
*Indication* : On commence par déterminer $c_0$ en regardant la parité de $n$ : si $n$ est pair alors $c_0 = 0$ et si $n$ est impair alors $c_0 = 1$. Ensuite pour déterminer $c_1$ on recommence cette opération avec $n/2$ si $n$ est pair et $(n-1)/2$ si $n$ est impair. On itère ce procéder jusqu'à obtenir $n = 0$.

**3.** Définissez la fonction function L=puissListe2(x,k) qui prend en paramètre une réel $x$ et un entier naturel $k$ et qui renvoie L=[$x$, $x^2$, $x^{2^2}$,..., $x^{2^k}$]. Cette fonction ne doit pas faire plus de $k$ multiplications. On rappelle qu'il est interdit d'utiliser le symbole ^ dans cette fonction.
*Indication* : $x^{2^{i+1}} = x^{2^i} \times x^{2^i}$.

On revient maintenant au problème initial, à savoir calculer $x^n$ avec le moins d'opération possible. Si la décomposition de $n$ en base 2 est

$$n = \sum_{i=0}^{k} c_i 2^i$$

alors on observe que

$$x^n = x^{c_0} \times (x^2)^{c_1} \times (x^{2^2})^{c_2} \times \cdots \times (x^{2^k})^{c_k}.$$

**4.** A l'aide de l'observation ci-dessus et à l'aide des fonctions définies dans les questions 2 et 3, définissez la fonction function r=puiss2(x,n) qui prend en paramètre une réel $x$ et un entier naturel $n$ et qui renvoie r=$x^n$.

**5.** Combien faut-il de multiplications pour calculer $x^{12}$ à l'aide de la seconde méthode ?

**6.** Comparez, avec la commande timer(), le temps d'exécution des commandes puiss1(2,10000000) et puiss2(2,10000000). Quelle fonction est la plus rapide ?

**7.** Écrire une fonction function r=puiss3(x,n) basée sur la fonction puiss2 mais cette fois sans utiliser les fonctions des question 2 et 3 et sans utiliser de listes. Le but de cette question est de réduire l'utilisation de la mémoire.