Sorbonne-Université, Faculté des Sciences et Ingénierie
Licence de Mécanique **Année universitaire 2019–2020**

LU3ME005 : Méthodes numériques pour la mécanique
Travaux Pratiques

TP1 — Racines d'équations

On souhaite résoudre dans $\mathbb{R}$ l'équation $e^x = 4x$. Pour ce faire, on introduit la fonction $f(x) = e^x - 4x$ et on cherche numériquement les racines de l'équation $f(x) = 0$ par des méthodes de type point fixe (simple, Newton, ...)

1. **Visualisation de la fonction à intégrer (Gnuplot)**
   Visualiser la fonction $f(x)$ sous gnuplot. Dans un terminal, taper `gnuplot &`
   Dans la fenêtre qui s'ouvre, taper les instructions suivantes pour obtenir le graphe :

```
plot exp(x)-4*x
set grid
replot
set xrange [0:3]
replot
...
```

   Remarque : pour sauvegarder un graphe au format image (png par exemple), on tape la séquence suivante :

```
set term png
set output "graphe.png"
replot
quit
```

   Donner visuellement un encadrement de chacune des racines de l'équation $f(x) = 0$.

2. **Programmation de la méthode de point fixe**
   On se propose d'écrire un programme de recherche de racine par méthode de point fixe. Cette méthode est basée sur le calcul des termes d'une suite définie par récurrence par $x_{k+1} = \phi(x_k)$ à partir d'une valeur initiale x_0 entrée au clavier par l'utilisateur.
   (a) Programmer la procédure fonction $\phi_1(x) = e^x/4$ ainsi que le programme principal de résolution par méthode de point fixe à $\epsilon = 10^{-12}$ près. Afficher le nombre d'itérations nécessaires. Compiler en double précision (`g95 -r8`) et exécuter.
   (b) ☺Programmer la procédure fonction $\psi_1(x, \theta) = (1 - \theta)x + \theta\phi_1(x)$ et utiliser cette fonction d'itération pour différentes valeurs du paramètre θ. Cette relaxation accélère-t-elle la convergence ?
   (c) Donner les deux racines à ϵ près.

3. **Programmation de la méthode de Newton**
   (a) Programmer les fonctions $f(x)$ et $f'(x)$, et implémenter la méthode de Newton $x_{k+1} = \phi_N(x_k)$.
   (b) Implémenter la méthode suivante dite de Jacobi–Chebyshev $x_{k+1} = \phi_J(x_k) = x_k - \dfrac{f(x_k)}{f'(x_k)} - \dfrac{f''(x_k)[f(x_k)]^2}{2[f'(x_k)]^3}$.

Comparer le nombre d'itérations nécessaire pour satisfaire le critère de convergence par ces différentes méthodes.

TP2 — Méthodes directes pour les systèmes linéaires

On souhaite résoudre par une méthode directe le système linéaire $Ax = b$ de vecteur inconnu $x \in \mathbb{R}^n$, où la matrice $A \in \mathbb{R}^{n,n}$ et le vecteur $b \in \mathbb{R}^n$ sont donnés.

1. Programmer une procédure de résolution du système dans le cas où A est triangulaire inférieure.
2. Programmer une procédure de résolution du système dans le cas où A est triangulaire supérieure.
3. Tester rapidement les procédures précédentes sur des systèmes 3×3.
4. Programmer une procédure de factorisation LU d'une matrice A. On choisira la version pour laquelle L est à diagonale unité ($\ell_{jj} = 1$ à ne pas oublier !) :

$$A = LU\ , \quad 1 \le j \le n\ : \quad \left| \begin{array}{ll} (1 \le i \le j) & u_{ij} = a_{ij} - \displaystyle\sum_{k=1}^{i-1} \ell_{ik} u_{kj} \\ (j+1 \le i \le n) & \ell_{ij} = \dfrac{1}{u_{jj}} \left(a_{ij} - \displaystyle\sum_{k=1}^{j-1} \ell_{ik} u_{kj} \right) . \end{array} \right.$$

5. Programmer la résolution du système $Ax = b$ utilisant les procédures écrites précédemment.
6. Application : prendre la matrice et le second membre du TD 2.1.
7. ☺Programmer une procédure de factorisation de Cholesky LL^t d'une matrice symétrique définie positive :

$$A = LL^t\ ,\quad 1 \le j \le n\ :\ \ell_{jj} = \sqrt{a_{jj} - \sum_{k=1}^{j-1} \ell_{jk}^2}\ ,\qquad (i > j)\ \ell_{ij} = \frac{1}{\ell_{jj}} \left(a_{ij} - \sum_{k=1}^{j-1} \ell_{ik}\ell_{jk} \right)\ .$$

8. ☺Programmer la résolution d'un système linéaire de matrice symétrique définie positive.
9. ☺Application : prendre la matrice et les seconds membres du TD 2.2.

TP3 — Méthodes itératives pour les systèmes linéaires

On souhaite résoudre par une méthode itérative le système linéaire $Ax = b$ de vecteur inconnu $x \in \mathbb{R}^n$, où la matrice $A \in \mathbb{R}^{n,n}$ et le vecteur $b \in \mathbb{R}^n$ sont donnés. On se restreint ici aux méthodes dites de "splitting" basées sur la décomposition de A en une matrice diagonale D, une matrice triangulaire inférieure E et une matrice triangulaire supérieure F :

$$A = D - E - F\ .$$

Au splitting $A = M - N$ correspond la méthode itérative

$$Mu^{(k+1)} = Nu^{(k)} + b$$

pour laquelle il est nécessaire de partir d'une solution initiale $u^{(0)}$. Il est préférable de mettre cette relation sous la forme dite de Richardson généralisée en écrivant $Mu^{(k+1)} = (M - A)u^{(k)} + b$ ou encore $M(u^{(k+1)} - u^{(k)}) = b - Au^{(k)}$. On est donc amené à résoudre à chaque itération un système d'inconnue $e^{(k+1)}$:

$$Me^{(k+1)} = r^{(k)}\ ,$$

où le second membre $r^{(k)} \equiv b - Au^{(k)}$ (le résidu) est calculé préalablement, et dont on déduit après résolution $u^{(k+1)} = u^{(k)} + e^{(k+1)}$. Il est à noter que, d'une façon générale, on ne cherche pas à inverser la matrice M, mais on résout le système linéaire à chaque itération. On contrôle la convergence sur la norme du résidu. Les principales méthodes sont :

— Jacobi : $M = D$
— Gauss–Seidel : $M = D - E$
— Relaxation (SOR) : $M = D/\omega - E$ où $\omega \in]0; 2[$ est le paramètre de relaxation ; il est alors plus commode de résoudre le système $(D - \omega E)e^{(k+1)} = \omega r^{(k)}$.

1. Programmer les méthodes itératives ci-dessus, en prenant soin de les tester systématiquement sur les exemples du TD 3.1. Afficher la norme du résidu à chaque itération. On pourra créer pour chaque méthode une procédure effectuant une itération.
2. Déterminer pour une matrice A la valeur optimale de ω pour SOR.
3. ☺Essayer de retrouver le rayon spectral des matrices d'itération en calculant la réduction de l'erreur à chaque itération.

TP4 — Valeurs propres

On considère la matrice de Vandermonde $A \in \mathbb{R}^{n,n}$ définie par $A_{ij} = j^i$. Pour $n = 4$, par exemple, on a :

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 4 & 9 & 16 \\ 1 & 8 & 27 & 64 \\ 1 & 16 & 81 & 256 \end{bmatrix}$$

1. Calculer par la méthode de la puissance itérée la valeur propre λ_1 de plus grand module et un vecteur propre x_1 associé. Prendre $n = 4$ pour tester l'algorithme (compiler en double précision !).
2. Utiliser la méthode de déflation pour calculer $(\lambda_2,\, x_2), \cdots, (\lambda_n,\, x_n)$.
3. ☺Recalculer $(\lambda_n,\, x_n)$ par la méthode de la puissance itérée inverse.