Corrigé Exercices OCTAVE - Analyse Numérique
Dr. S. Deparis
Série 2 - Systèmes linéaires : méthodes directes

Exercice 1

a) On définit la matrice A et le vecteur **b** par les commandes suggérées dans l'énoncé :

```
>> f='x.*(1-x)';
>> N=20; h=1/N;
>> x=[h:h:1-h]' ;% on transpose pour avoir un vecteur colonne
>> b=eval(f);
>> A=(N^2)*(diag(2*ones(N-1,1),0)-diag(ones(N-2,1),1)-diag(ones(N-2,1),-1));
```

Ensuite, on calcule la factorisation LU de A. En général, Octave peut décider d'effectuer des permutations de lignes pendant le processus de factorisation, ce qui mène à une factorisation $PA = LU$. Ainsi, la syntaxe de la commande `lu` à utiliser est la suivante :

```
>> [L,U,P] = lu(A);
```

De cette façon, on a stocké dans la variable `P` la matrice de permutation P. Dans notre cas, on peut vérifier que `P` est la matrice identité : par exemple on peut calculer l'écart maximale de la valeur absolue des éléments de P et I, et on obtient :

```
>> max(max(abs(P-eye(N-1))))
ans = 0
```

Les facteurs `L` et `U` diffèrent du facteur H de la factorisation de Cholesky $A = HH^T$, qu'on calcule par la commande

```
>> H = chol(A)';
```

car

```
>> max(max(abs(L - H)))
ans = 27.284
```

b) D'après le cours, on exploite la factorisation LU de la façon suivante :

```
>> y = L \ b;
>> u = U \ y;
```

Il faut savoir que la commande \ fait des tests préliminaires sur la matrice : si la matrice est triangulaire, la commande appliquera les substitutions rétrogrades/progressives pour trouver la solution et le coût de calcul sera modéré.
Pour afficher le temps de calcul avec `tic` et `toc` :

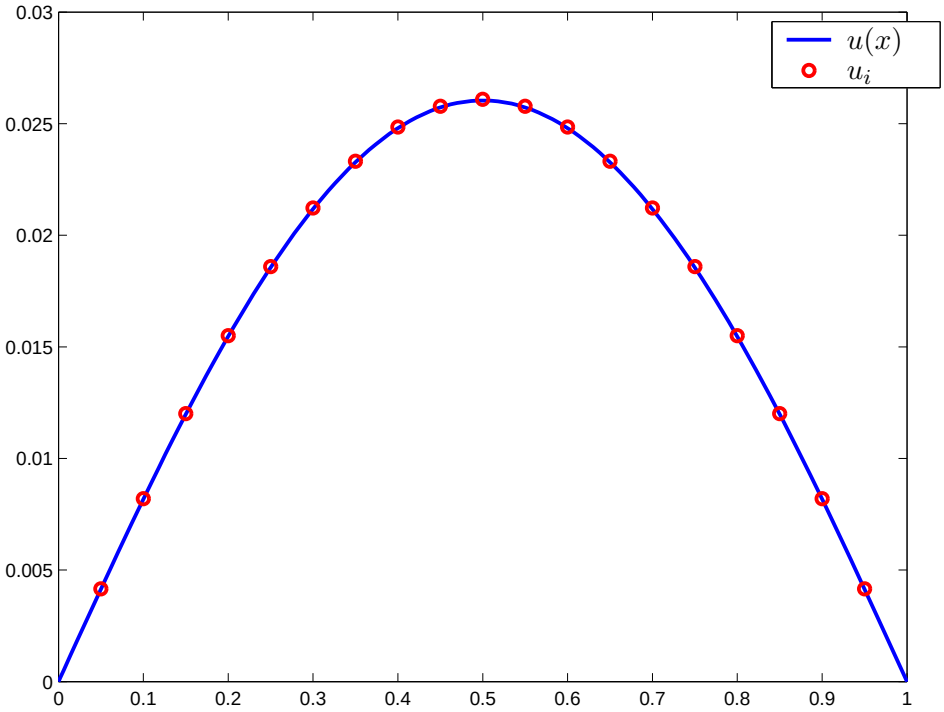

FIG. 1 – Solution exacte $u(x)$ et déplacements approchés u_i

```
>> tic; u = A \ b; toc
ans = 0.034896
```

donc la solution directe du système prend 34 millièmes de seconde, tandis que, si après avoir effacé toutes les variables et recalculé les facterurs L et U, on tape

```
>> tic; y = L \ b; u = U \ y; toc
ans = 0.016613
```

on a que la solution à partir des facteurs triangulaires prend seulement 16 millièmes de seconde. Attention, ces valeurs dépendent de votre processeur ; de plus, si on n'execute pas un `clear all` avant de calculer `y` et `u`, on va obtenir un "faux" temps, très petit (0.8 millièmes), car Octave reutilisera les informations en mémoire.

c) On peut faire un plot du déplacement du câble avec :

```
>> plot(x, u, '-or');
```

La figure 1 montre les déplacements u_i aux noeuds x_i (cercles rouges). Pour la partie facultative de l'exercice : si $f(x)$ est un polynôme, il est facile de trouver la solution exacte du problème différentiel. Dans notre cas[1] on a $u(x) = x^4/12 - x^3/6 + x/12$. On a ajouté le graphe de $u(x)$ afin de montrer comme avec $N = 20$ sous-intervalles on obtient une solution

[1]Comme $u''(x) = -x + x^2$, on intègre deux fois et on trouve $u(x) = x^4/12 - x^3/6 + ax + b$, a, b étant deux

approchée déjà assez précise. La figure 1 a été obtenue par les commandes :

```
>> u_exact = 'x.^4 / 12 - x.^3 / 6 + x / 12';
>> fplot(u_exact, [0,1]);
>> hold on;
>> plot(x, u, 'bo');
```

Il faut remarquer que la taille du vecteur **u** est $N-1$; en effet seulement les valeurs des déplacements aux noeuds x_i pour $i = 1, \ldots, N-1$ sont inconnus, car $u_0 = u_N = 0$ (conditions au bord).

d) Vu que la matrice A dépend seulement de N, elle est la même que celle des points précédents; par contre le vecteur **b** sera donné par les valeurs aux noeuds de la nouvelle force appliquée $\tilde{f}$. Donc, les facteurs L et U de A étant déjà disponibles, la stratégie la meilleure en termes de coût de calcul consistera à calculer le nouveau vecteur **b** et à résoudre le système par substitutions progressives/rétrogrades en exploitant la factorisation LU :

```
>> f_tilde = 'x.*sin(2*pi*x).^2';
>> b=eval(f_tilde); % on calcule le nouveau b
>> y = L \ b;       % substitutions en avant
>> u = U \ y;       % substitutions en arri\'ere
>> hold off;
>> plot(x, u);
```

Le résultat des calculs est affiché en figure 2 : la force $\tilde{f}$ n'étant pas symétrique, les déplacements ne le sont pas non plus.

e) Pour chaque valeur de N on a une matrice A différente. Donc il faut coder une boucle qui, pour $N = 10, 20, 30, \ldots, 120$, construit cette matrice et calcule son conditionnement. Cette valeur sera ensuite mémorisée dans un vecteur `K`. La boucle peut se coder comme il suit :

```
>> for N = 10:10:120;
> A=(N^2)*(diag(2*ones(N-1,1),0)-diag(ones(N-2,1),1)-diag(ones(N-2,1),-1));
> K(N/10) = cond(A);
> disp(sprintf('N = %i: K(A) = %e',N, K(N/10))); % ceci affiche
                                                 % les valeurs calculees
> end

N = 10: K(A) = 3.986346e+01
N = 20: K(A) = 1.614476e+02
N = 30: K(A) = 3.640898e+02
N = 40: K(A) = 6.477890e+02
N = 50: K(A) = 1.012545e+03
N = 60: K(A) = 1.458358e+03
```

constantes, que l'on trouve en imposant les conditions aux bord :

$$u(0) = 0 \implies b = 0, \qquad u(1) = 0 \implies a = 1/12.$$

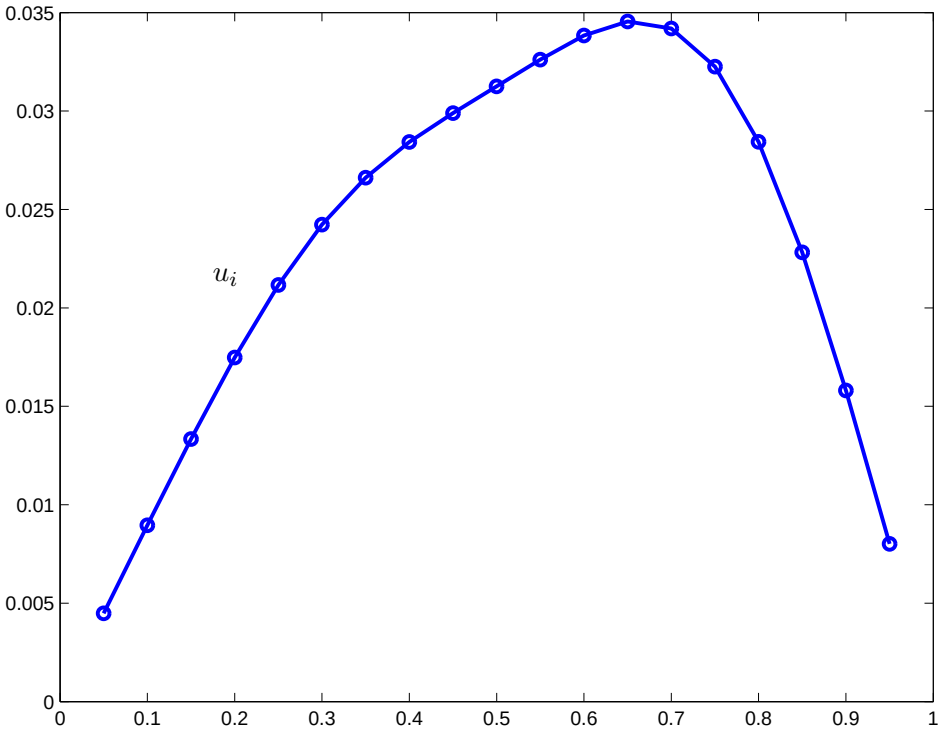

FIG. 2 – Déplacements approchés u_i lorsque la force $\tilde{f}(x) = x\sin(2\pi x)^2$ est appliquée au câble

```
N = 70: K(A) = 1.985229e+03
N = 80: K(A) = 2.593156e+03
N = 90: K(A) = 3.282140e+03
N = 100: K(A) = 4.052181e+03
N = 110: K(A) = 4.903279e+03
N = 120: K(A) = 5.835434e+03
```

Le graphe (commande `plot([10:10:120],K)`) et le graphe bilogarithmique (commande `loglog([10:10:120],K); grid on`) de $K(A)$ en fonction de N sont affichés en figure 3. On voit que le graphe bilog est bien celui d'une droite, donc du type

$$\log_{10} K = m \log_{10} N + c.$$

On calcule m et c directement sur le graphe, en mesurant la pente entre les abscisses $\log_{10} N = 1$ (si $N = 10$) et $\log_{10} N = 2$ (pour $N = 100$), ou bien en utilisant Octave :

```
>> m = ( log10(K(10)) - log10(K(1)) ) / (2 - 1)
m = 2.0071

>> c = log10(K(1)) - m*1
c = -0.40654

>> C = 10^c
C = 0.39216
```

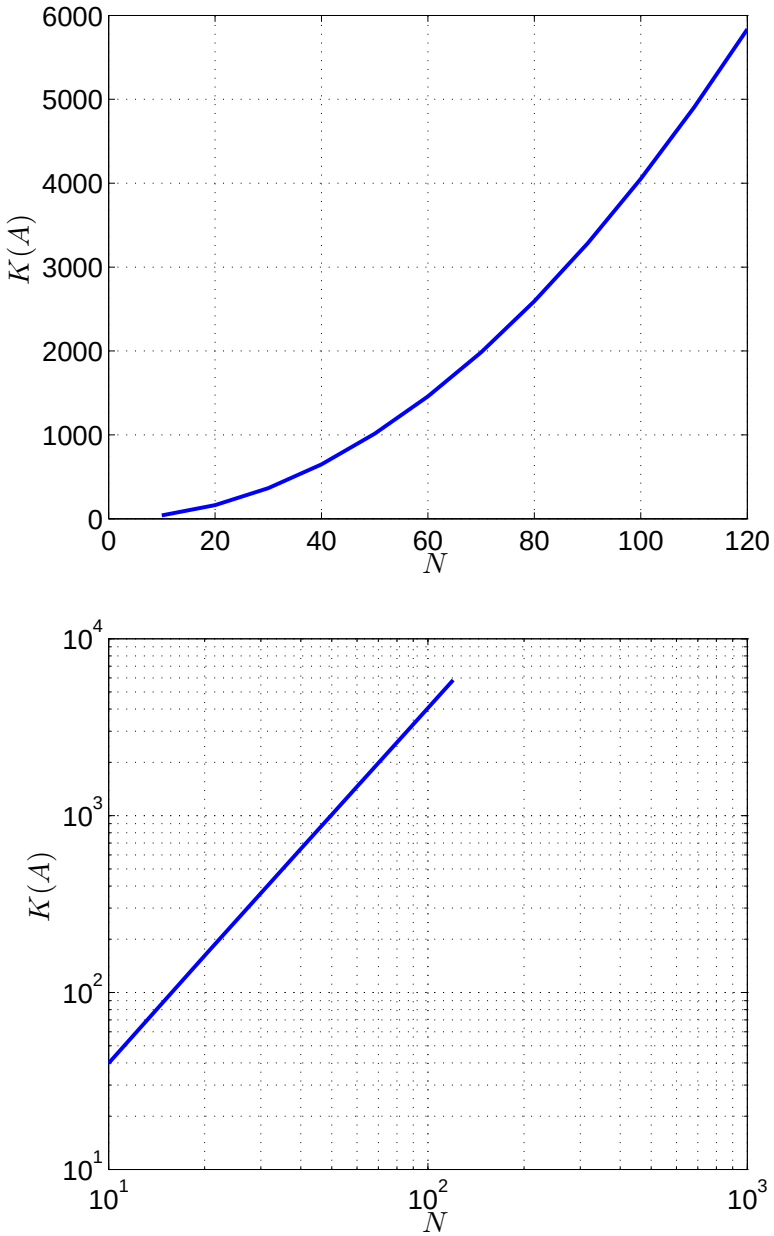

FIG. 3 – Nombre de conditionnement $K(A)$ en fonction de N, graphes linéaire (à gauche) et bilogarithmique (à droite)

La pente est quasiment égale à 2, donc la formule à proposer sera bien

$$K(A) = CN^2,$$

avec $C \simeq 0.4$. Donc $K(A)$ croît quadratiquement avec N (si l'on double N, $K(N)$ devient quatre fois plus grand), ce qui signifie que la solution du système linéaire par la méthode de factorisation LU devient de plus en plus sensible aux perturbations sur les données ou aux erreurs d'arrondissement.