MINF0402 TP2 Scilab
Les premiers exercices sont à but pédagogiques
Les exercices **** seront à rendre pour le rapport.

Extras+ 1.20 - RS

Note 1 (Utilité - pratique pour les tests et/ou la mise au point dans le TP1)
Comment générer une matrice inversible à coefficients entiers dont l'inverse soit aussi inversible à coefficients entiers Avec les instructions rand et round générer une matrice triangulaire supérieure U et une matrice triangulaire inférieure L dont les coefficients soient entiers compris entre -Nmax et +Nmax. (Nmax devant être préalablement choisi) Remplacer les coefficients diagonaux de L et U par des 1 ou -1. Former alors la matrice $A = L * U$. La matrice ainsi formée sera une matrice de test "agréable"... qu'on pourra utiliser pour les exercices d'inversion.
En fait, mieux ! si tous les coefficients diagonaux de la matrice L sont remplacés par des "1" alors la matrice triangulaire supérieure obtenue dans la réduction de Gauss est la matrice U ! (Ceci est encore vrai si on ne remplace pas les coefficients diagonaux de U par des 1 et -1, mais dans ce cas l'inverse de A=L*U n'est plus à coefficients entiers)

Exercice 1 (****)
Reprendre le dernier exercice du TP1 sur les matrices tridiagonales- On devra faire des test avec $n = 10$ par exemple Cet exercice est utilisé dans l'exercice suivant

Exercice 2 (** Un problème classique : l'équation de Laplace)**

Lire très soigneusement cet énoncé et les notes qui suivent, malgré sa longueur, il s'avère qu'il y a relativement peu de choses à faire, le travail central ayant été fait dans le dernier exercice du TP1, c'est à dire l'exercice ci dessus.
Toute l'information nécessaire est dans cet énoncé, Googliser "équation de Laplace" sera inutile et source de complication.

N.B. *Dans ce qui suit on entend souvent par matrice un tableau scilab bidimensionnel et par vecteur un tableau scilab mono-dimensionnel (en colonne, pour éviter tout souci) quand il ne s'agit pas des objets mathématiques associés.*

Soit le problème suivant :

$f(x)$ désignant une fonction donnée définie sur $[0, 1]$ trouver une fonction $u(x)$ définie sur $[0, 1]$ telle que :

$$\begin{cases} -u''(x) = f(x) \text{ sur } [0,1] \\ u(0) = 0 \text{ et } u(1) = 0. \quad \text{(les deux conditions aux bords)} \end{cases} \tag{1}$$

Ce problème peut correspondre à la flexion d'une poutre, aux répartitions de temperatures sur une barre chauffée...
Soit $N \in \mathbb{N}^*$ et soit $h = \frac{1}{N+1}$, on pose alors $x_i = ih$ pour $0 \leq i \leq N + 1$ (les noeuds). On va alors rechercher une solution approchée sous la forme d'une approximation des $u(x_i)$ qu'on notera u_i en approximant $u''(x_i)$ par :

$$\frac{u_{i-1} - 2u_i + u_{i+1}}{h^2}$$

ce qui donne alors en reportant dans le problème (1) *un système d'équations linéaires* à résoudre avec les u_i en écrivant l'équation qui doit être satisfaite en tous les points x_i (les noeuds) intérieurs à $[0, 1]$:

$$-u_{i-1} + 2u_i - u_{i+1} = h^2 f_i \quad 1 \leq i \leq N \quad \text{avec } f_i = f(x_i) \tag{2}$$

Pour les noeuds extrêmes, on a alors directement selon les conditions aux bords $u_0 = u(x_0) = u(0) = 0$ et $u_{N+1} = u(x_{N+1}) = u(1) = 0$, on a donc en fait un système de N équations avec N inconnues, les u_i, $1 \leq i \leq N$, car u_0 et u_{N+1} disparaissent des équations.

Ecrire dans un premier temps - à la main ! sur du papier (sisi ça existe) les 3 ou 4 premières équations et écrire les deux dernières. (Bien aligner les inconnues : u_1 sur la première colonne, u_2 sur la seconde etc...) Quelle est la matrice du système linéaire obtenue et quel est son second membre ? La matrice est tridiagonale comme dans le dernier exercice du TP1.

Ecrire le programme Scilab, effectuant cette résolution sous formes de fonctions et script.
Il faut - ayant choisi N - générer :

- h et les x_i dans un vecteur x (utiliser linspace) - attention il pourra être nécessaire d'évacuer x_0 et x_{N+1}
- Générer la matrice du système - dans un premier temps on on pourra générer une matrice pleine. - dans un second temps on généra seulement les trois diagonales pour utiliser l'algorithme établi au précédent TP.
- Calculer les $f(x_i)$ et générer le second membre.

- Résoudre avec votre programme du dernier exerice TP1 pour les matrices tridiagonales !
- Réintégrer u_0 et u_{N+1} (nuls) à la solution trouvée u et x_0 et x_{N+1} aux x_i. (On rajoute simplement un élément au début et à la fin de chacun des vecteurs u et x)
- Tracer la solution obtenue u en fonction de x - comparer avec la solution exacte.

On utilisera le programme de résolution pour les matrices tri-diagonales effectué dans le TP précédent.
Il faudra donc tester sur plusieurs exemples, et en particulier en fonction de N, le programme élaboré dans des cas où l'on connait déjà la solution.

On se donnera donc une fonction u(x) vérifiant les conditions aux bords, on calculera à la main $f(x) = -u''(x)$, puis en injectant le f ainsi calculé dans le programme de résolution on déterminera une solution approchée. On utilisera les exemples du tableau (1) ci dessous. ([1])

Il doit être clair que le nom de la fonction f doit être passée comme argument dans l'appel du sous-programme de résolution, et que les noms des fonctions u et f doivent être passées en arguments du programme de test (Voir la note 3) pour pouvoir effectuer de plus les comparaisons.

On devra donc avoir des fonctions scilab f_a, f_b, f_c, f_d et f_e correspondants aux $-u''_a(x)$, $-u''_b(x)$, $-u''_c(x)$ et $-u''_d(x)$ si on utilise les fonctions suggérées. (soit donc en termes d'identificateurs disons fa, fb fc et fd d'une part et ua, ub, uc et ud d'autre part les fonctions de références qu'on comparera aux solutions trouvées).

> On suggère cependant dans un premier temps pour mettre au point vos programmes de tout écrire en script (sans fonction donc, hors les fonctions f et u et hors la fonction de résolution de système linéaire tridiagonal Ax=b et encore on pourra même utiliser toujours dans ce premier temps, une vraie matrice A de type (N,N) et la résolution par Scilab par x=A\b)
> Cela permet d'avoir un accès direct au différentes variables depuis la console et de ne pas appeler sa mère (ou moi) en pleurant par ce que cela ne marche pas.

On pourra représenter la solution trouvée, ou mieux : les résidus, c'est à dire la différence entre la solution trouvée et la solution exacte. Il faudra voir comment évoluent les résidus quand N augmente. On pourra prendre successivement pour valeur de N par exemple 10, 20, 30, 40 ... On prendra garde en particulier au tracé pour les points aux bords ($u_0 = 0$ et $u_{N+1} = 0$), qu'il faut rajouter aux extrémités du vecteur u trouvé, augmentant donc la taille de ce vecteur (attention sous Scilab les indices commencent à 1 pas à 0, ce qui induit un décalage) et il faut bien entendu de même, rajouter les abscisses $x_0 = 0$ et $x_{N+1} = 0$ au vecteur x.

TABLE 1 – Quelques fonctions tests u et f.

	$u(x)$	$f(x) = -u"(x)$
a	$4x(1-x)$	8
b	$\sin(\pi x)$	$\pi^2 \sin(\pi x)$
c	$x^2(1-x)$	$6x-2$
d	$\sin(5\pi x)$	$25\pi^2 \sin(5\pi x)$
e	$x\sin(7\pi x)$	$49\pi^2 x \sin(7\pi x) - 14\pi\cos(7\pi x)$
...	...	...

Notons que pour le cas (a) la fonction f est constante ! Les fonctions (a) et (b) sont là essentiellement là pour faire des tests simples (Avec $N = 10$ par exemple pour mettre au point vos scripts et programmes)

On suggère de tracer les différentes fonctions solutions $u(x)$ ($x \in [0, 1]$), juste pour savoir à quoi s'attendre.

Attention - pour résoudre le problème on se donne N et la fonction recherchée est approximée sur en fait $N + 2$ points d'abcisses : $0 = x_0, x_1, ..., x_N, x_{N+1} = 1$. Par contre lorsque l'on trace la solution exacte (avec la formule de la colonne de gauche du tableau), on peut tracer sur un bien plus grand nombre de points $NN = 200$, par exemple. On pourra ainsi se rendre compte que $N = 10$ est bien trop petit pour obtenir une approximation satisfaisante de la solution dans les cas (d) et (e).

Pour un script avec plusieurs illustrations, faites des pauses à l'affichage en utilisant par exemple l'instruction >sleep(4000) pour des pauses de 4 secondes.

Note 2 (Rajouter un élément à un tableau mono-dimensionnel ligne ou colonne à la fin ou au début)
Tester les instructions suivantes :

```
// Ligne
Q=1:3
R=[Q 100]
R=[100 Q]
```

```
// Colonne
P=Q'
S=[P;100]
S=[100; P]
```

Note 3 (Passage de noms de fonctions dans une liste d'appel sous Scilab :)

Fragments de code :

1. On pourra aussi construire son propre exemple si on le désire -attention à bien prendre une fonction u vérifiant les conditions aux bords

```
...
// on se place dans le cas d'une solution connue (la fonction du cas (a)
// ua: solution exacte du problème - ceci n'est connu que
       lorsque que l'on fait des tests spécifiques
// où on a déjà la solution
function res=ma_fonction_ua(x)
   res=4*x.*(1-x)
endfunction
// fa: la fonction permettant de définir le second membre
       et qui devrait nous permettre de retrouver une approximation
       de ua après le calcul.
function res=ma_fonction_fa(x)
   res=8*ones(x)   // question importante pourquoi n'a t-on
                      pas écrit: "res=8" ??
endfunction
...
// idem tests avec la fonction du cas (b)
function res=ma_fonction_fb(x)
....
endfunction
function res=ma_fonction_ub(x)
....
endfunction
....
// le programme d'illustration de votre algorithme
// effectuant la résolution et comparant par les tracés
    (ou le tracé de la différence)
    avec la solution exacte
function [...]=ma_resolution_illustree(...., FONCTION_F, FONCTION_U)
....
endfunction
....
// pour faire les tests avec f_fa et u_ua:
[...]=ma_resolution_illustree(...., ma_fonction_fa, ma_fonction_ua)
.....
// pour faire les tests avec f_fb et u_ub :
[...]=ma_resolution_illustree(...., ma_fonction_fb, ma_fonction_ub)
....
```

Ce sont $ma\_fonction\_fa$ et $ma\_fonction\_fb$ qui apparaissent dans la liste d'appel! Ce ne sont pas - $ma\_fonction\_fa(x)$ et $ma\_fonction\_fb(x)$ qui sont des scalaires (ou des vecteurs) A noter à ce propos que si vos fonctions "f" (et "u") sont correctement écrites c'est à dire vectorisées (voir la note 4), elles devraient donc si x est un vecteur de taille N renvoyer un résultat vectoriel correspondant à $(f(x_1), f(x_2)...f(x_N))$ (respectivement à $(u(x_1), u(x_2)...u(x_N))$ (ce que font ici les fonctions $ma\_fonction\_fa$ et $ma\_fonction\_ua$ complètement écrites)

Note 4 (Fonction Vectorisée)

Si f est une fonction de $\mathbb{R}$ dans $\mathbb{R}$ qu'on implémente sous Scilab, On dira que la fonction est vectorisée si >y=f(x) renvoie un vecteur y de même taille que x avec y(i)=f(x(i)) . Les fonctions mathématiques internes de Scilab sont vectorisées, par contre prenez garde à vectoriser celles que vous écrivez, par exemple pour vectoriser la fonction f(x)=(x−2)(x−3) il faudra dans le corps de votre fonction écrire y=(x−2).*(x−3) et pas y=(x−2)*(x−3). (voir aussi l'écriture de la fonction fa dans la note (3).

Note 5 (Représentation d'une fonction - Rappels et compléments)

Pour représenter une fonction f sur un intervalle $[a, b]$, il faut discrétiser l'intervalle $[a, b]$ par exemple :

```
>x=linspace(a,b,1000)
```

et pour chaque x_i calculer $y_i = f(x_i)$. (**Si** la fonction f est **vectorisée** (voir la note 4) cela peut se faire avec $y = f(x)$). On peut ensuite tracer y en fonction de x avec l'instruction plotx(x,y) par exemple

```
>clf;
>NN=200
>x=linspace(-10,10,NN);
>y=sin(x);
>plot(x,y)
```

A noter qu'une instruction de la forme ">plot(y)" n'a intrinsèquement pas de sens!! Pour tracer il faut abscisses et ordonnées!. **Ce type d'erreur sera sévèrement sanctionné.** ([2]) Un tracé un peu plus compliqué : (suivez strictement les instructions, ne fermez pas la fenêtre graphique, ne l'effacez pas, si ce n'est pas indiqué)

2. l'instruction ">plot(y)" ne génère pas d'erreur, elle est interprétée par Scilab en prenant pour abscisse les numéros des composantes de y, ce n'est cependant pas ce que l'on cherche usuellement à tracer et les abscisses sont donc alors fausses

```
>clf;
>NN=200;
>x=linspace(-10,10,N);
>y=x;
>plot(x,y)
>y=-x;
>plot(x,y)
>y=x.*sin(2*%pi*x);
>plot(x,y,'r')
>x=1/4+linspace(-10,10,41);
>y=x.*sin(2*%pi*x);
>plot(x,y,'g*')
```

On notera les superpositions de courbes, les couleurs et les symboles

Remarque Si un dessin étant affiché sous scilab, on effectue $> plot(a, b)$ où a et b sont deux scalaires, on ne voit rien s'afficher ! en effet : on affiche un point qui est de taille nulle !, donc si on veut voir quelque chose à l'affichage il faut utiliser un symbole ou marqueur par exemple $- > plot(a, b, "r + ")$ pour afficher le symbole + en rouge au point de coordonnées (a, b).

——— • • ———

Un œuf de Pâques

Distraction hors sujet pour s'amuser pendant les vacances - aller voir et télécharger le fichier "cryptic.c" du répertoire TP, comprendre ce que fait ce code C de 23 petites lignes...

```
#include <stdio.h>
#include <math.h>
#include <unistd.h>
#include <sys/ioctl.h>

                main() {
            short a[4];ioctl
         (0,TIOCGWINSZ,&a);int
      b,c,d=*a,e=a[1];float f,g,
   h,i=d/2+d%2+1,j=d/5-1,k=0,l=e/
  2,m=d/4,n=.01*e,o=0,p=.1;while (
printf("\x1b[H\x1B[?25l"),!usleep(
30000)){for (b=c=0;h=2*(m-c)/i,f=-
.3*(g=(l-b)/i)+.954*h,c<d;c+=(b=++
b%e)==0)printf("\x1B[%dm ",g*g>1-h
*h?c>d-j?b<d-c||d-c>e-b?40:100:b<j
||b>e-j?40:g*(g+.6)+.09+h*h<1?100:
 47:((int)(9-k+(.954*g+.3*h)/sqrt
  (1-f*f))+(int)(2+f*2))%2==0?107
    :101);k+=p,m+=o,o=m>d-2*j?
      -.04*d:o+.002*d;n=(l+=
         n)<i||l>e-i?p=-p
              ,-n:n;}}
```

Si on renonce (ce qui est comprehensible) le compiler et le lancer dans un terminal :

```
$> gcc -w -o cryptic cryptic.c -lm
$> ./cryptic
```

Eviter la migraine ...

——— • **FIN** • ———