MINF0402 TP2 - Version Mise à Jour- 2021 - Scilab
Les trois exercices 0, 1 & 2 seront à rendre pour le rapport. 3.01 - TU

Exercice 0 (Matrices tridiagonales)

Faire **Exercice 6 du TP1** avec ses consignes et produire le programme RESOUTRI (que vous allez devoir utiliser dans l'exercice 2). et valider avec :

1. Faire une paire tests aléatoires avec $n \geq 10$ de votre choix avec les vérifications.
2. On va étudier le comportement de la méthode pour des grandes valeurs de n.
   Pas de fonction à écrire ici simplement un script avec une boucle, il suffira de réfléchir un peu et de suivre ce qui est demandé.

   Avec les notations du EX6 TP1, on considère la matrice M de type $(n,\ n)$ tridiagonale avec des 2 sur la diagonale, des -1 juste au dessus et juste en dessous. C'est à dire que A et C sont formés uniquement de -1 et B uniquement de 2. On considère par ailleurs également un D du même taille n uniquement formé de 1. On a donc à nouveau l'équation $MX = D$ qu'on résoudra avec RESOUTRI dans le contexte suivant :

   La commande >timer() renvoie le temps écoulé depuis son dernier appel, ainsi appelée avant et après par exemple l'appel d'une fonction, on aura lors du second appel à >timer() le temps passé dans cette fonction.

   Pour les valeurs de n : n_1, n_2 ... : 10, 10^2, 10^3, 10^4, 10^5, 10^6 qu'on stocke dans un tableau NN.

   - Résoudre l'équation $MX = D$,
   - determiner les durées d'exécution Ex_1, Ex_2 ,... qu'on stocke dans un tableau EX.
   - Calculer et afficher la norme de l'"erreur" : en calculant avec la solution X trouvée la norme $||MX - D||$ (voir la page 4 du TP1 et la fonction >norm)
   - Tracer alors le logarithme décimal du temps d'exécution en fonction du logarithme décimal de n. (Sous Scilab, on obtient le logarithme décimal avec la fonction >log10.

   Dans un premier temps, pour la mise au point, aller seulement jusqu'à $n = 10^4$ ensuite vous pourrez aller jusque 10^6 (une vingtaine de seconde sur ma vieille machine) et aller jus'à 10^7 si cela ne parait pas trop long.
   On peut aussi chercher à tracer le logarithme décimal de l'erreur en fonction du logarithme décimal de n.
   (NB : ne surtout pas chercher à former la matrice M trop grosse pour tenir en mémoire - quelle taille prendrait-elle en octets ? et il ne faut bien entendu pas afficher X et les autres pour des raisons évidentes)

Exercice 1 (Illustration Magique)

Cet exercice n'a qu'un but pédagogique, on n'a de toute façon que très rarement à inverser explicitement une matrice, on a par contre souvent à résoudre Ax=b pour A matrice inversible.

Ecrire une fonction d'inversion de matrice MAGIQUE(A) utilisant la méthode magique (c'est à dire la méthode des tableaux) pour calculer l'inverse de A. On suppose que les matrices fournies, sont effectivement inversibles et qu'il n'est pas nécessaire de permuter des lignes. L'algo n'a pas été fait en TD, il va donc falloir l'établir pour pouvoir faire cet exercice.

Tester votre programme avec des matrices aléatoires (au moins 5x5) - Vérifier le résultat (pas à la main ! !) - On affichera les "grands" tableaux intermédiaires (parties gauche et droite) obtenues à l'issue de chaque itération (utilisez disp) On devra pouvoir afficher les lignes en entier sans retour à la ligne - la fenêtre de console devra être suffisamment grande - cela permet de bien visualiser le focntionnement.

Voir le document en ligne TP2-EX1-Complément

```
// remarque sur les affichages de matrices en parallèle :
// regarder les effets des instructions suivantes
--> A=rand(4,4)
 A  =
```

```
    0.2693125    0.0437334    0.2806498    0.1121355
    0.6325745    0.4818509    0.1280058    0.6856896
    0.4051954    0.2639556    0.7783129    0.1531217
    0.9184708    0.4148104    0.211903     0.6970851

--> B=eye(4,4)
 B  =
    1.    0.    0.    0.
    0.    1.    0.    0.
    0.    0.    1.    0.
    0.    0.    0.    1.

--> C=[A,B]
 C  =
    0.2693125    0.0437334    0.2806498    0.1121355    1.    0.    0.    0.
    0.6325745    0.4818509    0.1280058    0.6856896    0.    1.    0.    0.
    0.4051954    0.2639556    0.7783129    0.1531217    0.    0.    1.    0.
    0.9184708    0.4148104    0.211903     0.6970851    0.    0.    0.    1.
```

Exercice 2 (Illustration graphique d'une approximation)

Pour les tracés de courbes revoir le TP0 page 5 ! !
Soient les fonctions suivantes définies sur [0,1] :

TABLE 1 – Les deux exemples de fonctions u et f.

	$u(t)$	$f(t)$
(a)	$t\sin(7\pi t)$	$49\pi^2 t\sin(7\pi t) - 14\pi\cos(7\pi t)$
(b)	$(t-t^2)(2+\sin(9\pi t))$	$4 + 2\sin 9\pi t + (2t-1)18\pi\cos 9\pi t + (t-t^2)(9\pi)^2\sin 9\pi t$

Il y a donc deux fonctions u que nous notons : u_a et u_b et deux fonctions f que nous notons f_a et f_b. Ci dessous on parlera uniquement de u et f, il faudra donc faire le travail successivement avec u_a et f_a d'une part et u_b et f_b d'autre part.

Attention à ne pas vous tromper lorsque vous programmerez ces fonctions, sinon tout sera faux !

1. Représenter en trait plein, la fonction $u(t)$ pour $t \in [0,1]$ avec disons 2000 points.
2. — On se donne N un entier (on pourra faire les tests avec N=10, 20, 30 inutile de dépasser 100 - voir plus bas)
   — On pose $h = \frac{1}{N+1}$
   — On considère la matrice M de type $(N,\ N)$ tridiagonale avec des 2 sur la diagonale, des -1 juste au dessus et juste en dessous.
   — On définit pour i entier avec $1 \leq i \leq N$:

   $$T_i = ih \quad \text{et} \quad D_i = h^2 f(T_i)$$

   ce qui définit deux tableaux monodimensionnels de longueur N : T et D.
   — On considère l'équation linéaire :

   $$MY = D \tag{1}$$

   Cette équation devra être résolue par votre programme RESOUTRI - sans former explicitement donc la matrice M. Soit donc Y sa solution, tracer Y en fonction de T, sur le même dessin que précédemment : on tracera un "+" (rouge) en chaque point $(T_i,\ Y_i)$
3. Faire les tracés pour différentes valeurs de N : 10, 20, 30, 40, 50 ... (en effaçant à chaque fois) les "+" doivent de plus en plus se placer près de la courbe. Faire des pauses avec l'instruction "sleep" entre chaque dessin

* Commentaire : pour les curieux, on a en fait ici $f(t) = -u''(t)$, le programme ici développé permet ici de retrouver sur $[0, 1]$, f à partir de u avec les conditions supplémentaires - dites conditions aux bords - $u(0) = u(1) = 0$, on est en fait en train de résoudre numériquement l'équation différentielle $-u''(t) = f(t)$ sur $[0, 1]$ avec les conditions aux bords indiquées - ici dans cette situation de test - on connait donc déjà la solution exacte (u) et on peut voir comment la solution approchée converge vers elle quand N augmente.

On devra donc rédiger un script pour le couple u_a – f_a d'une part et un autre pour le couple u_b – f_b. U_b se laisse moins facilement approcher que u_a.

———— • • ————

Distraction : Un œuf de Pâques

Cryptic

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=(1-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 compréhensible) le compiler et le lancer dans un terminal :

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

Eviter la migraine ...

———— • **FIN** • ————