Validité du Logiciel Numérique

Nicolas LIMARE

1er février 2005

Résumé

L'objectif du travail était une comparaison de la précision de deux méthodes de résolution de système linéaire ; l'une, factorisation LU, est évaluée à l'aide des majorants d'erreur qu'elle fournit, et l'autre, Gauss, est contrôlée par arithmétique stochastique. C'est également l'occasion d'utiliser les routines LAPACK et CADNA.

Table des matières

1 Présentation

L'objectif de ce travail consiste en la comparaison de deux méthodes de résolution de système linéaire du point de vue de la précision de leurs résultats : d´une part, par factorisation LU (en utilisant la routine LAPACK `dgesvx`), et d´autre part par une méthode de Gauss avec recherche du pivot. Pour l'estimation de l'erreur, on utilisera les paramètres fournis par la routine pour la résolution LU, et l'estimation par la méthode CESTAC pour le pivot de Gauss.

Le système de n équations linéaires envisagé est de la forme $AX = B$, où X est l´inconnue à déterminer. A est définie par

$$A_{i,j} = \frac{1}{i+j-1} \text{ si } i \leq n/2 \text{ ou } j \leq n/2$$

$$A_{i,j} = \frac{1}{i+j-1} + \frac{1}{i+j-n-1} \text{ sinon}$$

et B est trivialement $B_i = i$.

Une première estimation des erreurs est possible en calculant l'erreur relative $ER_i = \text{abs}(B_i - B'_i)/B_i$, où B' est le produit de A par la valeur proposée pour X par nos algorithmes de résolution. On observe alors l'évolution en fonction de la dimension n de $\max(ER)$ et $\min(ER)$, qui nous renseignent sur la qualité de la réponse des algorithmes à ce qui leur est demandé : fournir X tel que $AX = B$[1]. De plus, chacune des méthodes fournit deux autres informations (spécifiques à la méthode) sur l'erreur.

1.1 Méthode LU|LAPACK

La routine `dgesvx` de la bibliothèque LAPACK est destinée à la résolution d'un système quelconque d'équations linéaires, par une factorisation LU.

L'algorithme détermine les coefficients d'équilibrage de la matrice A puis la rééquilibre, en la multipliant, ainsi que B, avec des matrices diagonales de telle sorte que ses lignes et/ou ses colonnes aient la même norme, ce qui réduit la sensibilité de l'algorithme aux perturbations[2]. Ensuite, la matrice A équilibrée est factorisée.

Une première approche de X est alors simplement calculée en utilisant les matrices triangulaires L et U(par `dgetrs`), puis cette approche est amélorée par raffinements successifs (par `dgerfs`). CEs raffinements successifs s'appuient sur l'approximation courante de X et l'erreur arrière (*backward error*) associée, *ie* l'erreur faite en assimilant A à LU, car X est approché à partir de L et U or les calculs de L et U sont entachés d'une erreur d'arrondi et on n'a plus vraiment $A = LU$[3]. L'idée est alors de corriger la valeur de X calculée, car on dispose d'estimateurs de cette erreur arrière, et ce jusqu'à ce que cette erreur soit incompressible.

Enfin, le vecteur X ainsi obtenu est modifié pour correspondre à la solution du système d'origine, avant rééquilibrage.

La routine fournit également deux bornes d'erreur, l'erreur avant (*forward error*, différence normalisée entre la valeur calculéé de X et sa valeur exacte) et l'erreur arrière (*bqckward error* explicitée précédemment), toutes deux basées sur des majorations effecuées au cours de la résolution du système.

[1] ... et non fournir $X = A^{-1}B$, qui serait mathématiquement équivalent, mais a un autre sens en terme de qualité de résultat dès qu'on considère que la valeur fournie est une valeur approchée.

[2] J. H. WILKINSON, Error analysis of direct methods of matrix inversion, J. ACM 8 (1961),281-330

[3] J. H. WILKINSON, *idem*

On dispose également de diverses options permettant d'obtenir entre autres le conditionnement de A ou ses coefficients de rééquilibrage, de traiter le système à partir d'une matrice déjà factorisée ou de résoudre des variantes dy système $AX = B$.

1.2 Méthode Gauss|CADNA

L'algorithme utilisé est une méthode classique de Gauss, sélectionnant à chaque itération le pivot de valeur absolue maximale, afin de limiter les erreurs d'approximation.

CADNA est intégré à cette routine (`gauss_st`), afin de controler la précision réelle des résultats et d'éventuelles instabilités numériques de l'algorithme. Cela signifie que les matrices A est initialisée en calculant $\frac{1}{i+j-1}$ à partir de valeurs stochastiques de i et j ; une première implémentation de Gauss avec CADNA se contentait de convertir en variables stochastiques les valeurs de $A_{i,j}$ déjà calculées, donc déjà entachées d'erreur, et la conséquence était que les résultats n'étaient plus significatifs pour $n \geq 26$.

Le calcul de l'erreur relative est effectué en convertissant ls variables stochastiques en variables flottantes, afin de pouvoir comparer les deux méthodes. Enfin, CADNA permet de disposer de bornes maximum et minimum sur le nombre de chiffres significatifs de l'approximation de X fournie.

2 Résultats

L'essentiel des résultats est dans les fichiers `result.txt` et `data.txt`, obtenus en effectuant la comparaison des méthodes pour n allant de 4 à 30[4]. `result.txt` est reproduit en annexe page 13, en omettant les valeurs fournies pour X, mais en conservant l'intégralité des messages CADNA. Sa lecture, et surtout les représentations graphiques qu'on produit de ses données, nous permettent de conclure sur notre comparaison

Lors de la première implémentation, les données initiales de A étaient entachées d'erreur, et celle-ci s'amplifiant au cours de l'algorithme de Gauss, on n'avait plus aucune décimale significative à partir de $n = 26$, et de nombreux test et divisions instables apparaissaient ensuite par les divisions du pivot de Gauss.

Ici, au contraire, avec des données initiales correctes, on remarque sur la figure 1 que les resultats de la méthode de Gauss restent significatifs jusqu'à $n = 30$. Quelques pertes de précision apparaissent, trois au plus, mais elles semblent très dépendantes des données initiales et peuvent ne plus se manifester pour la valeur immédiatement supérieure de n.

Le nombre de décimales fiables ne dépend pas linéairement de la dimension des données, et un essai pour $n = 100$ montre qu'il reste 6 chiffres significatifs, avec *seulement* une cinquantaine de pertes de précision mais aucune instabilité.

Cette méthode semble donc plutôt robuste. Elle mériterait quelques raffinements au niveau du choix du piveau pour éviter les tests non significatifs, des améliorations comme le rééquilibrage de A, puis une vérification élargie à une plus vaste classe de données initiales.

L'évaluation de l'erreur fournie par LAPACK est intéressante. Il apparait sur la figure 2 que l'erreur *forward* sur l'estimation de X est quasi-stationnaire, alors que l'erreur *backward* suit une progression géométrique. Or cette dernière erreur est due à la factorisation LU, au centre de l'algorithme, et tout a été fait pour la réduire, par rafffinements successifs. Cette erreur risque peut-être de réduire progressivement le nombre de chiffre significatifs du résultat. On note également

[4]... par les commandes `$ get_data 4 30 2 > data.txt` et `$ get_result 4 30 2 > result.txt`.

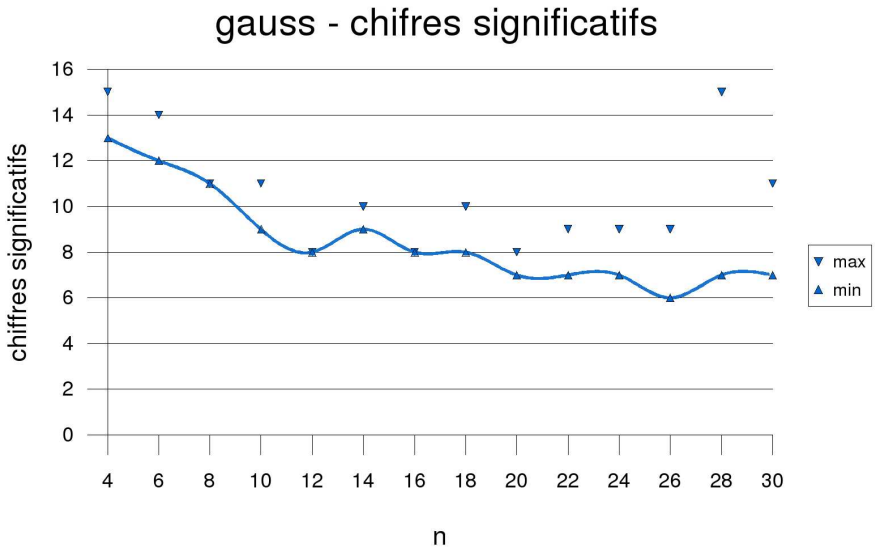

FIG. 1 – Le nombre minimum de chiffres bsignificatifs que fournit l'algorithme de Gauss est décroissant, mais sa décroissance semble fortement ralentir et ce nombre minimum de chiffres significatifs semble devoir être autour de 6 pour les systèmes de dimension au-delà de 30.

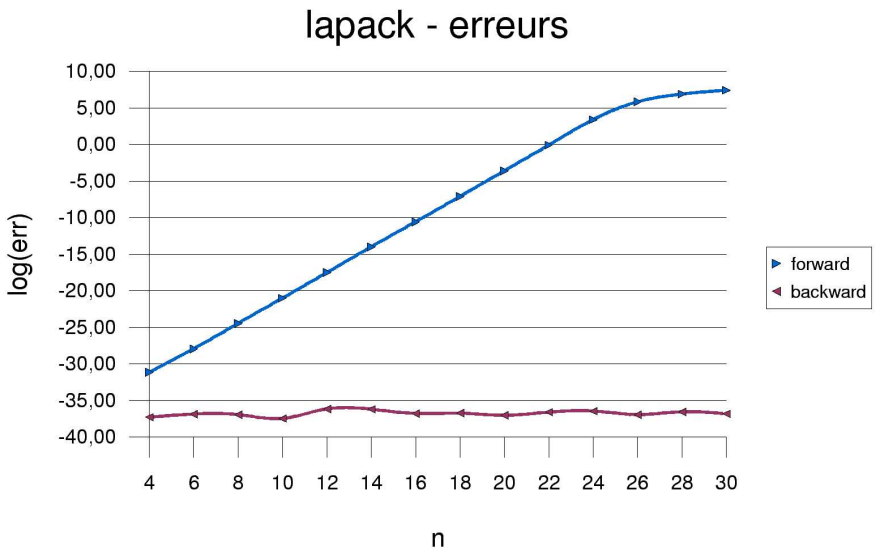

FIG. 2 – L'erreur *forward* reste stable, alors que l'erreur *backward* de la factorisation LU, elle, croît sans cesse.

que quelle que soit cette erreur *backward*, l'erreur d'approximation de X est quasi constante. La factorisation LU semble donc, être de plus en plus intéressante pour n croissant, tant que l'erreur reste raisonnable.

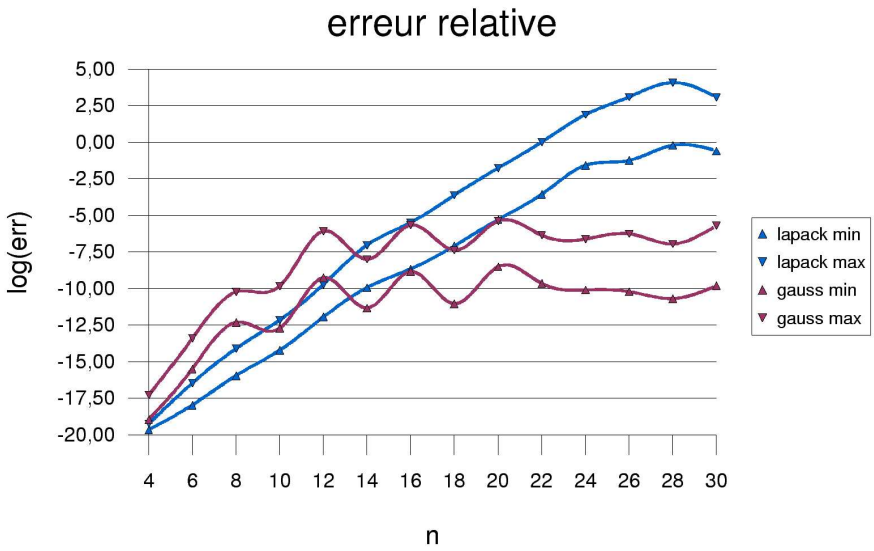

FIG. 3 – On voit clairement que les bornes de l'erreur de la méthode de Gauss passent rapidement en-dessous de celles de la factorisation LU ; si le temps de calcul n'etait pas pris en compte, la classique méthode de Gauss serait la plus satisfaisante.

La figure 3 enfin met en évidence l'adéquation des méthodes à notre objectif, qui est de trouver X tel que $AX = B$. Or il est clair ici que la méthode de Gauss sont rapidement meilleurs que la décomposition LU ; son erreur relative sur AX est nettement inférieure, et quelques essais pour de grandes valeurs de n confirment cette tendance pour la suite. Cela ne contredit pas l'appréciation précédente de LU, car nous observiions la qualité de X, non de AX. Et même si l'erreur restait quasi-constante sur X, l'influence de A, avec sa forme particulière utilisée ici et son mauvais conditionnement, prime dans l'évaluation de AX.

Dernière nuance : nous travaillons là en dimensions très réduites sur des données partiulières, alors que LAPACK est concu pour résoudre de grands systèmes[5], et qu'on peut attendre de celui qui l'utilise de savoir correctement conditionner la matrice étudiée.

[5]Un test pour $n = 200$ montre d'ailleurs une différence flagrante de rapidité, tellle que Gauss ne peut plus être envisagé.

3 Implémentation

Cette implémentation n'étant qu'un test, on suppose l'utilisateur coopératif, et peu de contrôles des entrées/sorties ou des allocations ont été effectuées. On s'est cependant appliqué à contrôler strictement les passages de paramètres, et c'est pouquoi les routines `dgesvx` et `gauss_st`, pour lesquelles `A` et `B` sont des paramètres d'entrée/sortie, sont appelées via les routines `lapack_solve` et `st_solve` qui effectuent une copie de `A` et `B` et l'utilisent dans le calcul de l'erreur relative. La conséquence est un encombrement mémoire un peu superflu, mais cela ne semble pas essentiel dans notre contexte.

De plus, des routines mineures sont appelées afin de récupérer des paramètres de ligne de commande qui sont stockés dans un fichier texte via un script shell.

3.1 Méthode LU|LAPACK

3.1.1 dgesvx

Naturellement, la routine `dgesvx` a été compilée et exécutée sans modifications. Une interface permet de l'appeler sans soucis.

3.1.2 lapack_solve

```
! lapack_solve resout le systeme ax=b, en dimension n
! - in
! * a et b : matrices du systeme
! * n : dimension du systeme
! - out
! * x : approximation de la solution du systeme via la routine gesvx
! * err_min et err_max : le min et le max de l'erreur relative er,
!   avec er=abs((b-ax)/b)
! * f_err et b_err : les erreurs 'forward' et 'backward' de la methode
!
subroutine lapack_solve(a,x,b,n,err_min,err_max,f_err,b_err)
  implicit none
  integer,intent(in) :: n
  real(kind(0.d0)),dimension(n,n),intent(in) :: a
  real(kind(0.d0)),dimension(n),intent(out) :: x
  real(kind(0.d0)),dimension(n),intent(in) :: b
  real(kind(0.d0)),intent(out) :: err_min,err_max,f_err,b_err

  real(kind(0.d0)),dimension(n,n) :: lu
  real(kind(0.d0)),dimension(4*n) :: work
  real(kind(0.d0)),dimension(n) :: r,c
  real(kind(0.d0)),dimension(1) :: ferr,berr
  integer,dimension(n) :: ipiv,iwork
  real(kind(0.d0)) :: rcond
  integer :: nrhs,info
  character :: equed,fact,trans

  real(kind(0.d0)),dimension(n,n) :: a_copy
  real(kind(0.d0)),dimension(n) :: b_copy
  real(kind(0.d0)),dimension(n) :: ax,x_err
```

```
  integer :: i

  interface
     subroutine dgesvx(fact,trans,n,nrhs,a,lda,af,ldaf,ipiv,equed,
                       r,c,b,ldb,x,ldx,rcond,ferr,berr,work,iwork,info)
       character,intent(in) :: fact,trans
       integer,intent(in) :: n,nrhs,lda,ldaf,ldb,ldx
       real(kind(0.d0)),dimension(lda,n),intent(inout) :: a
       real(kind(0.d0)),dimension(ldaf,n),intent(inout) :: af
       integer,dimension(n),intent(inout) :: ipiv
       character,intent(inout) :: equed
       real(kind(0.d0)),dimension(n),intent(inout) :: r,c
       real(kind(0.d0)),dimension(ldb,nrhs),intent(inout) :: b
       real(kind(0.d0)),dimension(ldx,nrhs),intent(out) :: x
       real(kind(0.d0)),intent(out) :: rcond
       real(kind(0.d0)),dimension(nrhs),intent(out) :: ferr,berr
       real(kind(0.d0)),dimension(4*n),intent(out) :: work
       integer,dimension(n),intent(out) :: iwork
       integer,intent(out) :: info
     endsubroutine dgesvx
  endinterface

  ! dgesvx modifie les valeurs de a et b passees en parametres
  ! on fait donc des copies afin de calculer ensuite l'erreur relative
  a_copy=a
  b_copy=b

  fact='N'
  trans='N'
  nrhs=1

  ! appel de la resolution par factorisation LU
  call dgesvx(fact=fact,trans=trans,n=n,nrhs=nrhs,a=a_copy,
              lda=n,af=lu,ldaf=n,ipiv=ipiv,equed=equed,
              r=r,c=c,b=b_copy,ldb=n,x=x,ldx=n,rcond=rcond,
              ferr=ferr,berr=berr,work=work,iwork=iwork,info=info)

  ! calcul de l'erreur relative
  ax=matmul(a,x)
  do i=1,n
     x_err(i)=abs((b(i)-ax(i))/b(i))
  enddo

  err_min=minval(x_err)
  err_max=maxval(x_err)
  f_err=ferr(1)
  b_err=berr(1)

  return
endsubroutine lapack_solve
```

3.2 Méthode Gauss|CADNA

3.2.1 gauss_st

```
! gauss_st résout le système ax=b par Gauss avec choix du pivot
! en controlant les donnees avec CADNA
! - in
! * n : dimension du systeme
! - inout
! * a et b : matrices du systeme
! - out
! * det : déterminant de a
!
subroutine gauss_st(a,b,n,det)
  use cadna
  implicit none
  integer,intent(in) :: n
  type(double_st),dimension(n,n),intent(inout) :: a
  type(double_st),dimension(n),intent(inout) :: b
  type(double_st),intent(out) :: det

  type(double_st) :: aux
  integer :: i,j,ll,k

  det=1.
  do i=1,n-1
     ll=i
     aux=0.
     do k=i,n
        ! possibilite de test instable
        if (abs(a(k,i)).ge.aux) then
           aux=abs(a(k,i))
           ll=k
        endif
     enddo
     if (ll.ne.i) then
        do j=i,n
           aux=a(i,j)
           a(i,j)=a(ll,j)
           a(ll,j)=aux
        enddo
        aux=b(i)
        b(i)=b(ll)
        b(ll)=aux
     endif
     aux=1./a(i,i)
     det=det/aux
     do j=i+1,n
        a(i,j)=a(i,j)*aux
     enddo
     b(i)=b(i)*aux
```

```
      do k=i+1,n
         aux=a(k,i)
         do j=i+1,n
            ! possibilite de perte de precision
            a(k,j)=a(k,j)-aux*a(i,j)
         enddo
         ! possibilite de perte de precision
         b(k)=b(k)-aux*b(i)
      enddo
   enddo
   i=n
   b(n)=b(n)/a(n,n)
   do i=n-1,1,-1
      do j=i+1,n
         ! possibilite de perte de precision
         b(i)=b(i)-a(i,j)*b(j)
      enddo
   enddo

return
endsubroutine gauss_st
```

3.2.2 st_solve

```
! st_solve resout le systeme ax=b, en dimension n
! - in
! * a et b : matrices du systeme
! * n : dimension du systeme
! - out
! * x : approximation de la solution du systeme via la routine dgesvx
! * err_min et err_max : le min et le max de l'erreur relative er,
!   avec er=abs((b-ax)/b)
! * deci_min et deci_max : le min et max du nombre de chiffres exacts
!
subroutine st_solve(a,x,b,n,err_min,err_max,deci_min,deci_max)
  use cadna
  implicit none
  integer,intent(in) :: n
  type(double_st),dimension(n,n),intent(in) :: a
  type(double_st),dimension(n),intent(out) :: x
  type(double_st),dimension(n),intent(in) :: b
  real(kind(0.d0)),intent(out) :: err_min,err_max
  integer,intent(out) :: deci_min,deci_max

  type(double_st) :: det

  real(kind(0.d0)),dimension(n,n) :: a2
  real(kind(0.d0)),dimension(n) :: ax,x_err,b2,x2
  integer,dimension(n) :: deci
  type(double_st),dimension(n,n) :: a_copy
```

```
  type(double_st),dimension(n) :: b_copy
  integer :: i,j

  interface
     subroutine gauss_st(a,b,n,det)
       use cadna
       integer,intent(in) :: n
       type(double_st),dimension(n,n),intent(inout) :: a
       type(double_st),dimension(n),intent(inout) :: b
       type(double_st),intent(out) :: det
     endsubroutine gauss_st
  endinterface

  ! gauss_st modifie les valeurs de a et b passees en parametres
  ! on fait donc des copies afin de calculer ensuite l'erreur relative
  a_copy=a
  b_copy=b

  ! appel de la resolution par Gauss
  call gauss_st(a_copy,b_copy,n,det)
  x=b_copy

  ! conversion en flottants et recuperation des chiffres significatifs
  do i=1,n
     do j=1,n
        a2(i,j)=a(i,j)
     enddo
     x2(i)=x(i)
     b2(i)=b(i)
     deci(i)=cestac(x(i))
  enddo

  deci_max=maxval(deci)

  return
endsubroutine st_solve
```

3.3 Programme principal

Selon l'usage souhaité, le programme principal envoie sur la sortie standard des resultats complets, avec les valeurs calculées pour X, ou des résultats abrégés n'incluant que les erreurs.

3.3.1 compare

```
}
! compare renvoie les resultats de la resolution de AX=B par deux methodes
! les parametres d'appel, recuperes dans un fichier texte, sont :
! * la dimension du systeme
! * '-data' si l'on ne souhaite que les donnees pertinentes,
!   ie les bornes d'erreur
```

```
!
program compare
  use misc_interface
  use lapack_interface
  use st_interface
  use cadna
  implicit none
  real(kind(0.d0)),allocatable,dimension(:,:) :: a
  real(kind(0.d0)),allocatable,dimension(:) :: b,x
  type(double_st),allocatable,dimension(:,:) :: a_st
  type(double_st),allocatable,dimension(:) :: b_st,x_st
  integer :: i,j,n,m,as,deci_max,deci_min
  real(kind(0.d0)) :: err_min_l,err_max_l,f_err,b_err
  real(kind(0.d0)) :: err_min_g,err_max_g
  logical :: data_flag

  call cadna_init(-1)

  ! recuperation des parametres de ligne de commande
  n=i_get_param('compare.params',1)
  i=get_params_nb('compare.params')
  if(i<2) then
     data_flag=.false.
  else
     data_flag=(c_get_param('compare.params',2).eq.'-data')
  endif

  if(n.eq.0) then
     ! en-tete des fichiers de donnees
     if(data_flag) write(unit=*,fmt='(a)') '#N ERR_MIN_L     ...'

  elseif(n.ge.1) then

     ! l'allocation n'est pas controlée, pour alléger le code
     allocate(a(n,n))
     allocate(a_st(n,n))
     allocate(b(n))
     allocate(b_st(n))
     allocate(x(n))
     allocate(x_st(n))

     ! affectation de A et B
     do i=1,n
        do j=1,n
           a(i,j)=1./(i+j-1)
           a_st(i,j)=1./(i+j-1)
        enddo
     enddo
     m=floor(real(n/2))
     do i=m+1,n
```

```
      do j=m+1,n
         a(i,j)=a(i,j)+1./(i-m+j-m-1)
         a_st(i,j)=a_st(i,j)+1./(i+j-1)
      enddo
   enddo
   do i=1,n
      b(i)=i
      b_st(i)=i
   enddo

   ! resolution du systeme
   call lapack_solve(a,x,b,n,err_min_l,err_max_l,f_err,b_err)
   call st_solve(a_st,x_st,b_st,n,err_min_g,err_max_g,
                 deci_min,deci_max)

   if(data_flag) then
      ! renvoi des erreurs
      write(unit=*,fmt='(i2,6e23.16,2i5)') n,err_min_l,err_max_l,
                                     f_err,b_err,
   err_min_g,err_max_g,
   deci_min,deci_max
   else
      ! renvoi du resultat formate
      write(unit=*,fmt='(i2,a)') n,'======================...'
      write(unit=*,fmt='(a,e25.16,e25.16)')
           '  LAPACK ERR |',err_min_l,err_max_l
      write(unit=*,fmt='(a,e25.16,e25.16)')
                 'LPCK FWD/BWD |',f_err,b_err
      write(unit=*,fmt='(a)') '--------------------------...'
      write(unit=*,fmt='(a,e25.16,e25.16)')
                 'GAUSS_ST ERR |',err_min_g,err_max_g
      write(unit=*,fmt='(a,i25,i25)')
                 'DECI MIN/MAX |',deci_min,deci_max
      write(unit=*,fmt='(a)')
                 '---------------DGESVX-----------------...'
      do i=1,n
         write(unit=*,fmt='(i12,e25.16,a,a)') i,x(i),'   ',str(x_st(i))
      enddo
   endif
endif

endprogram compare
```

3.3.2 Scripts

compare

```
#!/bin/sh
# le script 'compare'
# stocke les parametres de ligne de commande dans 'compare.params'
# et lance le binaire param.bin
```

```
#
BINARY_NAME=$0;
BINARY_FILE=$BINARY_NAME.bin;
PARAMS_FILE=$BINARY_NAME.params;
if (test -f $PARAMS_FILE); then
  /bin/rm $PARAMS_FILE;
fi
touch $PARAMS_FILE;
i=1;
while (test ${!i}); do
{ echo ${!i} >> $PARAMS_FILE;
  i=$(($i+1)); }
done
$BINARY_FILE;
```

get_result

```
#!/bin/sh
# le script 'get_result n1 n2 n3'
# appelle successivement 'compare'
# avec les parametres de dimension
# n1, n1+n3, n1+2*n3...n2
# et lit le listing cadna correspondant
#
RUN=compare;
CADNA_REPORT_FILE=cadna_stability_f90.lst;

for i in `seq ${1} ${3} ${2}`;
do
  $RUN $i
  cat $CADNA_REPORT_FILE
done
```

get_data

```
#!/bin/sh
# le script 'get_result n1 n2 n3'
# appelle successivement 'compare'
# avec les parametres de dimension
# n1, n1+n3, n1+2*n3...n2
# et l'option -data
#
RUN=compare;

$RUN 0 -data
for i in `seq ${1} ${3} ${2}`;
do
  $RUN $i -data
done
```

3.3.3 résultat.txt

Fichier `result.txt` abrégé :

```
 4============================================================
  LAPACK ERR |   0.8881784197001252E-15   0.1332267629550188E-14
LPCK FWD/BWD |   0.2920264313913801E-13   0.6341119624117019E-16
------------------------------------------------------------
GAUSS_ST ERR |   0.1776356839400250E-14   0.9473903143468002E-14
DECI MIN/MAX |                       13                       15
 6============================================================
  LAPACK ERR |   0.4736951571734001E-14   0.2131628207280301E-13
LPCK FWD/BWD |   0.7346031499100961E-12   0.9727917740892559E-16
------------------------------------------------------------
GAUSS_ST ERR |   0.5684341886080801E-13   0.4547473508864641E-12
DECI MIN/MAX |                       12                       14
 8============================================================
  LAPACK ERR |   0.3552713678800501E-13   0.2273736754432321E-12
LPCK FWD/BWD |   0.2346425738309453E-10   0.9018461976597422E-16
------------------------------------------------------------
GAUSS_ST ERR |   0.1364242052659392E-11   0.1091393642127514E-10
DECI MIN/MAX |                       11                       11
       1 PERTE BRUTALE DE PRECISION.
10============================================================
  LAPACK ERR |   0.2021099337273174E-12   0.1591615728102624E-11
LPCK FWD/BWD |   0.7461371244198400E-09   0.5358579576070156E-16
------------------------------------------------------------
GAUSS_ST ERR |   0.9094947017729282E-12   0.1637090463191271E-10
DECI MIN/MAX |                        9                       11
12============================================================
  LAPACK ERR |   0.1970571853841344E-11   0.1818989403545856E-10
LPCK FWD/BWD |   0.2449569902058693E-07   0.1959438818419593E-15
------------------------------------------------------------
GAUSS_ST ERR |   0.2910383045673370E-10   0.6984919309616089E-09
DECI MIN/MAX |                        8                        8
       1 PERTE BRUTALE DE PRECISION.
14============================================================
  LAPACK ERR |   0.1455191522836685E-10   0.2655724529176950E-09
LPCK FWD/BWD |   0.8220730770581944E-06   0.1852353954230486E-15
------------------------------------------------------------
GAUSS_ST ERR |   0.3637978807091713E-11   0.1018634065985680E-09
DECI MIN/MAX |                        9                       10
16============================================================
  LAPACK ERR |   0.5275069270282984E-10   0.1251464709639549E-08
LPCK FWD/BWD |   0.2525748240474345E-04   0.1063829750947511E-15
------------------------------------------------------------
GAUSS_ST ERR |   0.4365574568510056E-10   0.1047737896442413E-08
DECI MIN/MAX |                        8                        8
18============================================================
  LAPACK ERR |   0.2522331972916921E-09   0.8149072527885437E-08
LPCK FWD/BWD |   0.8086570739814042E-03   0.1089417466396492E-15
------------------------------------------------------------
GAUSS_ST ERR |   0.4850638409455617E-11   0.1891748979687691E-09
DECI MIN/MAX |                        8                       10
```

```
20============================================================
  LAPACK ERR |   0.1536682248115539E-08   0.5168840289115906E-07
LPCK FWD/BWD |   0.2685983444062943E-01   0.8304728553606081E-16
------------------------------------------------------------
GAUSS_ST ERR |   0.6127122201417621E-10   0.1396983861923218E-08
DECI MIN/MAX |                         7                        8
       1 PERTE BRUTALE DE PRECISION.
22============================================================
  LAPACK ERR |   0.8635900237343527E-08   0.3073364496231079E-06
LPCK FWD/BWD |   0.9039620641683197E+00   0.1272255388493614E-15
------------------------------------------------------------
GAUSS_ST ERR |   0.1940255363782247E-10   0.5238689482212067E-09
DECI MIN/MAX |                         7                        9
24============================================================
  LAPACK ERR |   0.6270905335744222E-07   0.1996755599975586E-05
LPCK FWD/BWD |   0.2916880824976895E+02   0.1465751176013723E-15
------------------------------------------------------------
GAUSS_ST ERR |   0.1265383932901465E-10   0.4074536263942719E-09
DECI MIN/MAX |                         7                        9
       1 PERTE BRUTALE DE PRECISION.
26============================================================
  LAPACK ERR |   0.8800998330116272E-07   0.6683170795440674E-05
LPCK FWD/BWD |   0.3310414228118120E+03   0.9170477286664715E-16
------------------------------------------------------------
GAUSS_ST ERR |   0.1119378094489758E-10   0.5820766091346741E-09
DECI MIN/MAX |                         6                        9
       1 PERTE BRUTALE DE PRECISION.
       2 PERTE BRUTALE DE PRECISION.
       3 PERTE BRUTALE DE PRECISION.
28============================================================
  LAPACK ERR |   0.2469335283551897E-06   0.1806020736694336E-04
LPCK FWD/BWD |   0.9665487902724597E+03   0.1345276316188853E-15
------------------------------------------------------------
GAUSS_ST ERR |   0.6984919309616089E-11   0.2910383045673370E-09
DECI MIN/MAX |                         7                       15
       1 PERTE BRUTALE DE PRECISION.
30============================================================
  LAPACK ERR |   0.1685372714338631E-06   0.6496906280517578E-05
LPCK FWD/BWD |   0.1645645344511348E+04   0.1006807697882132E-15
------------------------------------------------------------
GAUSS_ST ERR |   0.1663076026099069E-10   0.1018634065985680E-08
DECI MIN/MAX |                         7                       11
```