Bibliothèques

Thierry Dumont
–Institut Camille Jordan–

16 février 2017

Typologie

Méthodes numériques

- Algèbre linéaire, typologie.
  - BLAS
  - Matrices pleines : Lapack
  - Matrices creuses
- Transformée de Fourier Rapide
- Systèmes d'équations différentielles

Bibliothèques d'objets et méthodes associées

Parallélisme

Graphique

Et d'autres besoins ?

Communications entres langages

C(++) -> Fortran
Python -> C(++), Python -> Fortran

A l'aide !

Différentes types de bibliothèques : point de vue fonctionnel.

- algorithmes, méthodes.
  - calculs
  - parallélisme, communications,
  - graphiques.
- objets et méthodes associées.

Différentes types de bibliothèques : point de vue fonctionnel.

- algorithmes, méthodes.
  - calculs
  - parallélisme, communications,
  - graphiques.
- objets et méthodes associées.

Généricité.

Différentes types de bibliothèques : langages

1. classiques : C, Fortran,
2. à objets : C++, Python.

Différentes types de bibliothèques : langages

1. classiques : C, Fortran,
2. à objets : C++, Python.

En pratique :

1. bibliothèque binaire (archive .a ou dynamique .o)

Différentes types de bibliothèques : langages

1. classiques : C, Fortran,
2. à objets : C++, Python.

En pratique :

1. bibliothèque binaire (archive .a ou dynamique .o)+ include files (C),
2. include file.

Différentes types de bibliothèques : interface

Contrat entre l'utilisateur et la bibliothèque : si un objet correct est fournit, on aura un résultat : calcul effectué ou signal d'erreur.

Différentes types de bibliothèques : interface

Contrat entre l'utilisateur et la bibliothèque : si un objet correct est fournit, on aura un résultat : calcul effectué ou signal d'erreur.

1. compilation complètement séparée (vieux fortran 77). Pas de contrôle.

Différentes types de bibliothèques : interface

Contrat entre l'utilisateur et la bibliothèque : si un objet correct est fournit, on aura un résultat : calcul effectué ou signal d'erreur.

1. compilation complètement séparée (vieux fortran 77). Pas de contrôle.
2. C : include file : décrit l'interface du module. Contrôle du compilateur.

Différentes types de bibliothèques : interface

Contrat entre l'utilisateur et la bibliothèque : si un objet correct est fournit, on aura un résultat : calcul effectué ou signal d'erreur.

1. compilation complètement séparée (vieux fortran 77). Pas de contrôle.
2. C : include file : décrit l'interface du module. Contrôle du compilateur.
3. C++ et langages à objet. Plus rigoureux.

En pratique : Linux (Ubuntu, Debian)

Packages .deb :
libsuperlu3 et libsuperlu3-dev

En pratique : Linux (Ubuntu, Debian)

Packages .deb :
libsuperlu3 et libsuperlu3-dev
Installent :
1-

```
/usr/lib/libsuperlu.a
/usr/lib/libsuperlu.so
/usr/lib/libsuperlu.so.3
/usr/lib/libsuperlu.so.3.0.0
```

2-
`/usr/include/superlu/...` et
`/usr/share/doc/libsuperlu-dev/`.

Licences

Logiciels libres (GPL, Cecill....)

Licences

Logiciels libres (GPL, Cecill....)
Attention à quelques vieilles bibliothèques (exemple : Fishpack).

Algèbre linéaire : typologie

1. matrices pleines,
2. matrices creuses.

Algèbre linéaire : typologie

1. matrices pleines,
2. matrices creuses.

1. systèmes linéaires, moindres carrés etc...
2. valeurs et vecteurs propres.

Algèbre linéaire : typologie

1. matrices pleines,
2. matrices creuses.

1. systèmes linéaires, moindres carrés etc...
2. valeurs et vecteurs propres.

Coefficients : flottants simples ou doubles, complexes, rationnels.

Basic Linear Algebra Subroutine.

Années 80 : Linpack. Subroutines Fortran reposent sur les BLAS.
Repris par Lapack puis par de nombreuses autres bibliothèques.

1. BLAS 1 : opérations sur les vecteurs,
2. BLAS 2 : opérations (matrices, vecteurs),
3. BLAS 3 : opérations matrices x vecteurs.

Atlas :

Calculs par blocs pour optimiser les défauts de cache. S'optimisent à l'installation .

Atlas :
Calculs par blocs pour optimiser les défauts de cache. S'optimisent à l'installation .

OpenBLAS : Contient des noyaux pré-optimisés pour différentes architectures.
Autres BLAS : Intel MKL.

Atlas :
Calculs par blocs pour optimiser les défauts de cache. S'optimisent à l'installation .

OpenBLAS : Contient des noyaux pré-optimisés pour différentes architectures.
Autres BLAS : Intel MKL.
OpenBLAS et MKL : versions séquentielles et parallèles.

Atlas :
Calculs par blocs pour optimiser les défauts de cache. S'optimisent à l'installation .

OpenBLAS : Contient des noyaux pré-optimisés pour différentes architectures.
Autres BLAS : Intel MKL.
OpenBLAS et MKL : versions séquentielles et parallèles.
Exemple : 3Ghz, Intel I7, 16 cœurs. Produit de 2 matrices 1000x1000 : **300 Gigaflops** .
Il faut utiliser des BLAS optimisés ! .

Lapack

- reposent sur les blas.
- fortran 77.
- (S) simple, (D) double, (C) complexe, (Z) complexe double.
- man pages.

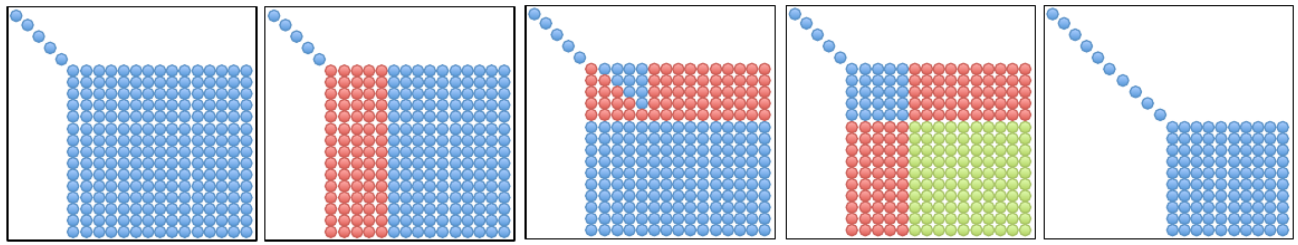
Lapack

- reposent sur les blas.
- fortran 77.
- (S) simple, (D) double, (C) complexe, (Z) complexe double.
- man pages.
- factorisations, systèmes linéaires, valeurs et vecteurs propres.
- matrices pleines, bandes, symétriques, symétriques bande.
- Toujours en développement.

Édition des liens : `-llapack -latlas`.

Exposé de Jack Dongarra, Lyon, 2016.

The Standard LU Factorization LAPACK
1980's HPC of the Day: Cache Based SMP



Factor panel with Level 1,2 BLAS

Triangular update

Schur complement update

Next Step

Main points

- Panel factorization mostly sequential due to memory bottleneck
- Triangular solve has little parallelism
- Schur complement update is the only easy parallelize task
- Partial pivoting complicates things even further
- Bulk synchronous parallelism (fork-join)
  - Load imbalance
  - Non-trivial Amdahl fraction in the panel
  - Potential workaround (look-ahead) has complicated implementation

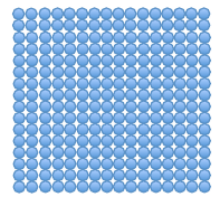
Exposé de Jack Dongarra, Lyon, 2016.

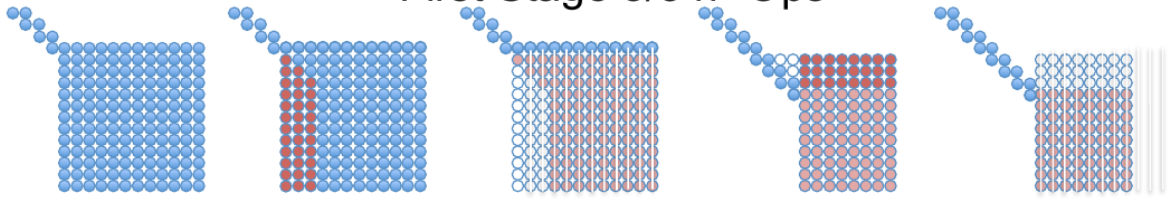
Singular Value Decomposition

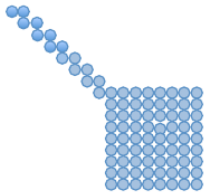
LAPACK Version 1991

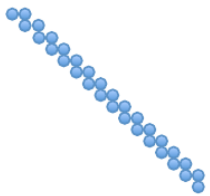
Level 1, 2, & 3 BLAS

First Stage $8/3\ n^3$ Ops

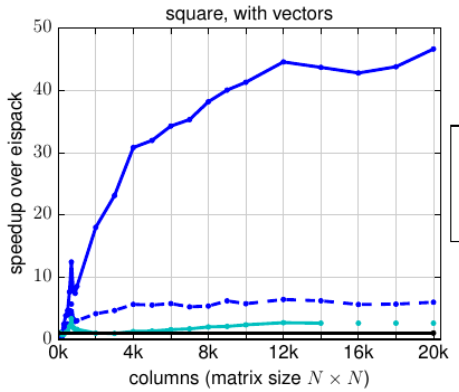








3 Generations of software compared



- LAPACK QR (BLAS in ||, 16 cores)
- LAPACK QR (restricted to 1 core)
- LINPACK QR
- EISPACK QR

QR refers to the QR algorithm for computing the eigenvalues

Dual socket – 8 core
Intel Sandy Bridge 2.6 GHz
(8 Flops per core per cycle)

Matrices creuses : une structure de données “universelle”

$$\begin{vmatrix} 1 & 2 & 0 & 0 \\ 0 & 3 & 9 & 0 \\ 0 & 1 & 4 & 0 \end{vmatrix}$$

Matrices creuses : une structure de données “universelle”

$$\begin{vmatrix} 1 & 2 & 0 & 0 \\ 0 & 3 & 9 & 0 \\ 0 & 1 & 4 & 0 \end{vmatrix}$$

Format CSL :

$$\begin{aligned} A &= \begin{vmatrix} 1 & 2 & 3 & 9 & 1 & 4 \end{vmatrix} \\ JA &= \begin{vmatrix} 0 & 1 & 1 & 2 & 1 & 2 \end{vmatrix} \\ IA &= \begin{vmatrix} 0 & 2 & 4 & 6 \end{vmatrix} \end{aligned}$$

Matrices creuses : une structure de données “universelle”

$$\begin{vmatrix} 1 & 2 & 0 & 0 \\ 0 & 3 & 9 & 0 \\ 0 & 1 & 4 & 0 \end{vmatrix}$$

Format CSL :

$$\begin{aligned} \text{A} &= \begin{vmatrix} 1 & 2 & 3 & 9 & 1 & 4 \end{vmatrix} \\ \text{JA} &= \begin{vmatrix} 0 & 1 & 1 & 2 & 1 & 2 \end{vmatrix} \\ \text{IA} &= \begin{vmatrix} 0 & 2 & 4 & 6 \end{vmatrix} \end{aligned}$$

Le format CSR s’obtient en classant les coefficients par colonnes.

$$L_{n-1} \dots L_2 L_1 A = U.$$

$$L_{n-1} \dots L_2 L_1 A = U.$$

$$L = (L_{n-1} \dots L_2 L_1)^{-1}$$

$$L_{n-1} \dots L_2 L_1 A = U.$$

$$L = (L_{n-1} \dots L_2 L_1)^{-1}$$

$$A = LU$$

Années 70 80 : abandon au profit des méthodes itératives.

Décomposition LU : le retour !

Toute la difficulté est dans la factorisation !

Décomposition LU : le retour !

Toute la difficulté est dans la factorisation !

1. algorithmes de renumérotation plus efficace (mais problème np-complet),
2. fabrication de sous blocs pleins et appel des `BLAS`.
3. technique multifrontale.
4. etc.

Décomposition LU : SuperLU, outil à tout faire.

Première implantation moderne, améliorée constamment.

Décomposition LU : SuperLU, outil à tout faire.

Première implantation moderne, améliorée constamment.

+

- en C, interfaces Fortran.

Décomposition LU : SuperLU, outil à tout faire.

Première implantation moderne, améliorée constamment.

+

- en C, interfaces Fortran.
- disponible dans les distributions Linux,
- utilisée par ne nombreux logiciels (Matlab, Scipy...)
- très fiable.
- mode “matrice symétrique”.

Décomposition LU : SuperLU, outil à tout faire.

Première implantation moderne, améliorée constamment.

\+

- en C, interfaces Fortran.
- disponible dans les distributions Linux,
- utilisée par ne nombreux logiciels (Matlab, Scipy...)
- très fiable.
- mode “matrice symétrique”.

\-

- interface désagréable (très “C”),
- variations de l'interface d'une version à l'autre, ainsi que des “include files”.

Décomposition LU : SuperLU, retour d'expérience

- performances remarquables, en tout cas en séquentiel.
- parfait pour résoudre une suite de systèmes identiques.
- Exemple de $\Delta U = F$:
  - 2d : ok jusqu'à 150 000 inconnues ou plus.
  - 3d, ou 10^6 inconnues : trop lent, trop de mémoire.
- la dernière version calcule aussi des factorisations incomplètes.

Décomposition LU : SuperLU, retour d'expérience

- performances remarquables, en tout cas en séquentiel.
- parfait pour résoudre une suite de systèmes identiques.
- Exemple de $\Delta U = F$:
  - 2d : ok jusqu'à 150 000 inconnues ou plus.
  - 3d, ou 10^6 inconnues : trop lent, trop de mémoire.
- la dernière version calcule aussi des factorisations incomplètes.

Un outil de base presque parfait

Décomposition LU : SuperLU, retour d'expérience

- performances remarquables, en tout cas en séquentiel.
- parfait pour résoudre une suite de systèmes identiques.
- Exemple de $\Delta U = F$:
  - 2d : ok jusqu'à 150 000 inconnues ou plus.
  - 3d, ou 10^6 inconnues : trop lent, trop de mémoire.
- la dernière version calcule aussi des factorisations incomplètes.

Un outil de base presque parfait

A utiliser :
-en mode mise au point (plutôt que des méthodes itératives),
-pour des problèmes de taille raisonnable.

Décomposition LU : autres implantations

Solveurs parallèles :
-MUMPS : `http://graal.ens-lyon.fr/MUMPS/`
-PASTIX :
`http://dept-info.labri.u-bordeaux.fr/~ramet/pastix/`

Méthodes itératives

- système symétrique : Gradient Conjugué.
- système non symétrique : GMRES et autres méthodes.

Méthodes itératives

- système symétrique : Gradient Conjugué.
- système non symétrique : GMRES et autres méthodes.

Préconditionnement : $AX = B => KAX = KB$.

Propriété importante : les seules expressions ou A intervient sont des produits matrices $y = Ax$.

Conséquences :

- Parallélisation MPI relativement facile,
- On n'a pas forcément besoin de connaître A, mais seulement l'action de A sur un vecteur.

Différents préconditionneurs :

- factorisations incomplètes,
- solveurs approchés,
- préconditionnement ad'hoc.

(tous les codes sont libres (GPL, CCIL)).

- Codes de Y. Saad (Mr GMRES) :
  `http://www-users.cs.umn.edu/` : sparskit, parms , itsol.
- PETSC `http://www.mcs.anl.gov/petsc/petsc-as/`.
- HIPS `http://hips.gforge.inria.fr/`.
- HYPRE `http://acts.nersc.gov/hypre/`.

PETSC, HIPS, HYPRE : PETSC quasi standard.

En bleu les codes parallèles.

(tous les codes sont libres (GPL, CCIL)).

- Codes de Y. Saad (Mr GMRES) :
  `http://www-users.cs.umn.edu/` : sparskit, parms , itsol.
- PETSC `http://www.mcs.anl.gov/petsc/petsc-as/`.
- HIPS `http://hips.gforge.inria.fr/`.
- HYPRE `http://acts.nersc.gov/hypre/`.

PETSC, HIPS, HYPRE : PETSC quasi standard.

En bleu les codes parallèles.

Valeurs et vecteurs propres : ARPACK.

FFT.

Une seule bibliothèque FFTW.
`http://www.fftw.org/`
Principe voisin de ATLAS.

- lsode. Source seulement. Méthode Gear (première méthode pour systèmes raides).
- routines fortran de H. Hairer et ses collègues (Univ. Genève) `http://www.unige.ch/~hairer/software.html`. Hautement recommandables !

GSL : (Gnu Scientific library). En C. Contient plein de bonnes choses.

Bibliothèques d'objets et méthodes associées

C++ : exemple (ancien) blitz++.

Bibliothèque de templates de tableaux.

C++ : exemple (ancien) blitz++.

Bibliothèque de templates de tableaux.

```
Array<int,2> X(100,20)
Array<double,2> X(100,20)
Array<MaClasse,2> X(100,20)
```

C++ : exemple (ancien) blitz++.

Bibliothèque de templates de tableaux.

```
Array<int,2> X(100,20)
Array<double,2> X(100,20)
Array<MaClasse,2> X(100,20)
```

Utilise les *expression templates* pour optimiser les expressions du genre :
`A= X+Y+Z;` (entre tableaux).

Pas d'algèbre linéaire.

Standard Template Library.

Objets courants : `Vector<int>`, `Set<MyClass>`, ... et iterateurs .

Standard Template Library.

Objets courants : `Vector<int>, Set<MyClass>, ....` et iterateurs .

```
Set<int> S ;
S.insert(20) ;
....
int total=0 ;
for(set<int> : :iterator I=S.begin() ; I !=S.end() ;I++)
total+=*I ;
```

Standard Template Library.

Objets courants : `Vector<int>, Set<MyClass>, ....` et iterateurs .

```
Set<int> S ;
S.insert(20) ;
....
int total=0 ;
for(set<int> : :iterator I=S.begin() ; I !=S.end() ;I++)
total+=*I ;
```

Conteneurs, Adaptateurs, Itérateurs, Algorithmes (Exemple : sort, find...).

Construire une matrice CSL (CSR). Une astuce C++

Standard template library : `map` et `pair`.

Construire une matrice CSL (CSR). Une astuce C++

Standard template library : `map` et `pair`.

- `map` : modélise une application d'un ensemble ordonné (E) dans un ensemble (X),

Construire une matrice CSL (CSR). Une astuce C++

Standard template library : `map` et `pair`.

- `map` : modélise une application d'un ensemble ordonné (E) dans un ensemble (X),
- `pair` : les paires d'objets ordonnés sont munies de l'ordre lexicographique.

Construire une matrice CSL (CSR). Une astuce C++

Standard template library : `map` et `pair`.

- `map` : modélise une application d'un ensemble ordonné (E) dans un ensemble (X),
- `pair` : les paires d'objets ordonnés sont munies de l'ordre lexicographique.

```
map<pair<int,int>,double> M;
M[make_pair(i,j)]=1.0;
```

Construire une matrice CSL (CSR). Une astuce C++

Standard template library : `map` et `pair`.

- `map` : modélise une application d'un ensemble ordonné (E) dans un ensemble (X),
- `pair` : les paires d'objets ordonnés sont munies de l'ordre lexicographique.

```
map<pair<int,int>,double> M;
M[make_pair(i,j)]=1.0;
```

Ensuite, l'itérateur associé permet de parcourir la map `M` dans l'ordre ad'hoc $=>$ construction facile de la matrice CSR (ou CSL).

Construire une matrice CSL (CSR). Une astuce C++

Standard template library : `map` et `pair`.

- `map` : modélise une application d'un ensemble ordonné (E) dans un ensemble (X),
- `pair` : les paires d'objets ordonnés sont munies de l'ordre lexicographique.

```
map<pair<int,int>,double> M;
M[make_pair(i,j)]=1.0;
```

Ensuite, l'itérateur associé permet de parcourir la map `M` dans l'ordre ad'hoc => construction facile de la matrice CSR (ou CSL).

Derrière : arbres B (B-trees, arbres équilibrés)=> performant.
Note : on peut changer l'ordre sur `pair`.

Au delà de la STL

- BOOST. `www.boost.org` (disponible dans les distributions Linux),
- mouvement d'idées.

Au delà de la STL

- BOOST. `www.boost.org` (disponible dans les distributions Linux),
- mouvement d'idées.

On vise la généricité .
Implantation : avant tout des include files.

Calcul réparti : MPI

Standard de fait du calcul réparti.
Plusieurs implémentations (Mpich, Lam, OpenMpi).

OpenMP n'est pas une bibliothèque !

Gestion de threads :

- gestion directe des threads posix.

OpenMP n'est pas une bibliothèque !

Gestion de threads :

- gestion directe des threads posix.
- BoostThread (C++).

OpenMP n'est pas une bibliothèque !

Gestion de threads :

- gestion directe des threads posix.
- BoostThread (C++).
- TBB . Threads Building Blocks. C++.
  Origine Intel, GPL.

OpenMP n'est pas une bibliothèque !

Gestion de threads :

- gestion directe des threads posix.
- BoostThread (C++).
- TBB . Threads Building Blocks. C++.
  Origine Intel, GPL.
  Découpe en taches modélisées par des classes. On dispose alors de classes comme "parallel-for".
  Simple à utiliser ! (et efficace).

Graphique

"Toolkit" VTK : `http://www.vtk.org/`
Surtout utilisé depuis Python.

Autres besoins ?

Exemples :

1. mesure du temps
2. clickodromes : QT ?
3. xml : libxml2
4. etc...

Deux choses à savoir :

1- En fortran : passage des paramètres par adresse

adresse= pointeurs C.

Deux choses à savoir :

1- En fortran : passage des paramètres par adresse

adresse= pointeurs C.

Exemple : man dgesv

```
NAME
  DGESV - computes the solution to a real system of
  linear equations   A * X = B,
SYNOPSIS
    SUBROUTINE DGESV( N, NRHS, A, LDA, IPIV, B, LDB, INFO )
      INTEGER        INFO, LDA, LDB, N, NRHS
      INTEGER        IPIV( * )
      DOUBLE         PRECISION A( LDA, * ), B( LDB, * )
```

il faut fabriquer un *header* `dgesv.h`

```
void dgesv_(int* n,int *nrhs, double* a, int* lda,
          int* ipiv, double*b,int* ldb, int* info);
```

La routine C inclura ce fichier : `#include` “`dgesv.h`”.

il faut fabriquer un *header* `dgesv.h`

```
void dgesv_(int* n,int *nrhs, double* a, int* lda,
          int* ipiv, double*b,int* ldb, int* info);
```

La routine C inclura ce fichier : `#include "dgesv.h"`.

```
int n=50,nrhs=1,lda=50;
double a[2500];
....
dgesv_(&n,&nrhs,a,&lda,.....);
```

2- Rangement des tableaux : Fortran parcourt les tableaux colonnes après colonnes et les indices commencent à 1 !

2- Rangement des tableaux : Fortran parcourt les tableaux colonnes après colonnes et les indices commencent à 1 !
Formule de passage :

```
double precision a(50,60)
a(12,22)=1.0
```

Où est `a(12,22)` ?

2- Rangement des tableaux : Fortran parcourt les tableaux colonnes après colonnes et les indices commencent à 1 !
Formule de passage :

```
double precision a(50,60)
a(12,22)=1.0
```

Où est `a(12,22)` ?

Vu du C : `a` : pointeur sur le début du tableau. Correspond à `a(1,1)`.

2- Rangement des tableaux : Fortran parcourt les tableaux colonnes après colonnes et les indices commencent à 1 !
Formule de passage :

```
double precision a(50,60)
a(12,22)=1.0
```

Où est `a(12,22)` ?

Vu du C : `a` : pointeur sur le début du tableau. Correspond à `a(1,1)`.
21 colonnes pleines avant `a(12,22)`.
Donc, vu du C, `a(12,22)` est à l'adresse : `a + 21*50+ 11`.

2- Rangement des tableaux : Fortran parcourt les tableaux colonnes après colonnes et les indices commencent à 1 !
Formule de passage :

```
double precision a(50,60)
a(12,22)=1.0
```

Où est `a(12,22)` ?

Vu du C : `a` : pointeur sur le début du tableau. Correspond à `a(1,1)`.
21 colonnes pleines avant `a(12,22)`.
Donc, vu du C, `a(12,22)` est à l'adresse : `a + 21*50+ 11`.
Formule générale : `a(i,j)` $->$ `a+ (j-1)*50+i-1`.

Python -> C

- swig `http://www.swig.org/`
- BoostPython.
- cython (Python with C extensions)

Benchmark Sage : cython le plus rapide.
Pb. des callbacks.

Python -> Fortran

- f2py. `http://cens.ioc.ee/projects/f2py2e/`
- on doit pouvoir utiliser cython.

Ces interfaces sont utilisés par Scipy.

A l'aide !

Liste de diffusion du Groupe Calcul :
`http://calcul.math.cnrs.fr/spip.php?rubrique3`