Librairies pour le calcul numérique

Contexte

- Matlab (/Octave, Scilab)
  - Très bonnes fonctions calcul numérique

Mais ...

- Programmation non numérique laborieuse
  Pas de structures de données (hash, map, graph, etc)
- Lent
- Non interactif

=> Uniquement restreint à réaliser des prototypes

Contexte

- C++
  - Vrai language de programmation
    Structures données avancées
  - Rapide
  - Interactif (Qt, OpenGL, ...)
  - Standard en entreprise

Mais ...

- Pas de librairie numérique dans le standard (Matrices, vecteurs, ...)

=> Beaucoup de temps passé pour refaire
ce que fait un script Matlab
+ bugs

Librairies numériques

Il est possible de réécrire vos propres structures

Inversion matricielle, valeurs propres, matrices creuses, ...

Mais à éviter pour des vrais applications (sauf cas spécifique)

Préférer des librairies existantes

Testées, plus robustes et rapide qu'un code personnel

Une bonne librairie de calcul numérique, plusieurs mois/années
de developpement

Calcul numérique

A l'origine
un language est développé spécifiquement pour le calcul numérique

Fortran

(FORmula TRANslating system)

Fortan

Fortran : un des plus ancien language (1954)
(C 1970)

A beaucoup évolué
- Fortan 66
- Fortan 77
- Fortan 90
- Fortran 2008
- Fortran 2015...

Autrefois très limité

Aujourd'hui language objet avancé
- Surchage d'opérateur
- Gestion mémoire dynamique
- Héritage
- polymorphisme
- structure données abstraites

Fortan

Language très aggréable pour calcul numérique

- Calcul vectoriel en standard
- Array slicing
- Matrice en standard
- Parallélisme aisé

```
INTEGER N = ...

REAL, DIMENSION(N,N)  :: A
INTEGER, DIMENSION(N) :: v, w
DO i = 1, N
   w(i) = SUM ( A(i, : ) * v )
END DO
```

Ressemble à Matlab, + simple que C, C++, Java, ...

Avantage/limitation Fortran

+ Aggréable pour le calcul numérique
+ Portable
+ Standard le plus robuste et répendu
+ Très rapide (souvent + rapide que C ou C++)
+ Mise en place sur les supercalculateurs

- Uniquement centré sur le calcul numérique
- Gestion texte difficile
- Pas de *logiciel* en Fortran avec GUI, interaction
- Pas de graphique

Fortan et Lapack

Librairie numérique de référence

- Structure de donnée matrice

BLAS

Basic **L**inear **A**lgebra **S**ubprograms

(opérations somme, multiplication matricielles)

- Algorithmes d'algèbre linéaire

Lapack

Linear **A**lgebra **PACK**age

(inversion matrices, factorisation)

Lapack

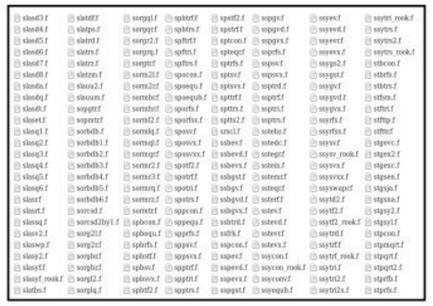
Lapack à été pendant longtemps le standard "de facto" du calcul numérique

Matlab était une *interface* au dessus de Lapack

Mais ...

Blas et Lapack sont bas niveau. Peu aggréable à utiliser

~1800 fichiers



```
DO I = 1, M-Q

   IF( I .EQ. 1 ) THEN
      DO J = 1, M
         PHANTOM(J) = ZERO
      END DO
      CALL ZUNBDB5( P, M+P, Q, PHANTOM(1), 1, PHANTOM(P+1), 1,
$                   X11, LDX11, X21, LDX21, WORK(IORBDB5),
$                   LORBDB5, CHILDINFO )
      CALL ZSCAL( P, NEGONE, PHANTOM(1), 1 )
      CALL ZLARFGP( P, PHANTOM(1), PHANTOM(2), 1, TAUP1(1) )
      CALL ZLARFGP( M-P, PHANTOM(P+1), PHANTOM(P+2), 1, TAUP2(1) )
      THETA(I) = ATAN2( DBLE( PHANTOM(1) ), DBLE( PHANTOM(P+1) ) )
      C = COS( THETA(I) )
      S = SIN( THETA(I) )
      PHANTOM(1) = ONE
      PHANTOM(P+1) = ONE
      CALL ZLARF( 'L', P, Q, PHANTOM(1), 1, DCONJG(TAUP1(1)), X11,
$                 LDX11, WORK(ILARF) )
      CALL ZLARF( 'L', M-P, Q, PHANTOM(P+1), 1, DCONJG(TAUP2(1)),
$                 X21, LDX21, WORK(ILARF) )
   ELSE
      CALL ZUNBDB5( P-I+1, M-P-I+1, Q-I+1, X11(I,I-1), 1,
$                   X21(I,I-1), 1, X11(I,I), LDX11, X21(I,I),
$                   LDX21, WORK(IORBDB5), LORBDB5, CHILDINFO )
      CALL ZSCAL( P-I+1, NEGONE, X11(I,I-1), 1 )
      CALL ZLARFGP( P-I+1, X11(I,I-1), X11(I+1,I-1), 1, TAUP1(I) )
      CALL ZLARFGP( M-P-I+1, X21(I,I-1), X21(I+1,I-1), 1,
$                   TAUP2(I) )
      THETA(I) = ATAN2( DBLE( X11(I,I-1) ), DBLE( X21(I,I-1) ) )
      C = COS( THETA(I) )
      S = SIN( THETA(I) )
      X11(I,I-1) = ONE
      X21(I,I-1) = ONE
```


Autres librairies

- Souvent une surcouche au dessus de Lapack
- Souvent spécialisé sur un domaine
  - Conteneur matriciel
  - Décomposition matrice pleine
  - Décomposition matrice creuse
  - Fitting non linéaire
  - ODE / PDE

Les grands acteurs: Vue d'ensemble

GPU — *Pour la puissance de calcul*

+ Le plus performant

- Codage long et difficile
- Portabilité

Python + Numpy — *Remplace de plus en plus Fortran + Lapack*

+ Interfacage aisé avec graphique (Qt) et GPU
- Par défaut plus lent que Fortran

Librairies C Librairies C++

+ Interfaçage direct sur un logiciel complet (Qt, drivers, OS, interaction, etc)

- Apprentissage plus long
- Pas de standard
- Portabilité

Pour développer des logiciels

Matlab / Octave / Scilab — *Pour les protos*

+ Très simple d'utilisation

- Limitées aux prototypes
- Lenteur du code

Fortran + Lapack — *Pour le standard*

+ Standard
+ Robuste et vérifié

- Limité au calcul

Exemple de librairies

GSL: GNU Scientific Library

+ Librairie open source complète

- En C (utilisation fastidieuse)
- Moins robuste, vérifiée que Lapack
  (lib issu de l'open source écrite par des informaticiens)

- Complex Numbers:
- Polynomials:
- Special Functions:
- Vectors and Matrices:
- Permutations:
- Linear Algebra:
- Eigensystems:
- Fast Fourier Transforms:
- Numerical Integration:
- Statistics:
- Monte Carlo Integration:
- Ordinary Differential Equations:
- Interpolation:
- Numerical Differentiation:
- Series Acceleration:
- Wavelet Transforms:
- Multidimensional Root-Finding:
- Least-Squares Fitting:
- Basis Splines:

```
int
gsl_schur_solve_equation_z(double ca, const gsl_matrix *A, gsl_complex *z,
                           double d1, double d2,
                           const gsl_vector_complex *b,
                           gsl_vector_complex *x, double *s, double *xnorm,
                           double smin)
{
  size_t N = A->size1;
  double scale = 1.0;
  double bnorm;

  if (N == 1)
    {
      double cr,    /* denominator */
             ci,
             cnorm; /* |c| */
      gsl_complex bval, c, xval, tmp;

      /* we have a 1-by-1 (complex) scalar system to solve */

      /* c = ca*a - z*d1 */
      cr = ca * gsl_matrix_get(A, 0, 0) - GSL_REAL(*z) * d1;
      ci = -GSL_IMAG(*z) * d1;
      cnorm = fabs(cr) + fabs(ci);

      if (cnorm < smin)
        {
          /* set c = smin*I */
          cr = smin;
          ci = 0.0;
          cnorm = smin;
        }
```


Exemple de librairies

- Blitz++ — Vecteur/Matrice dense. Librairie très rapide (utilisation métaprogrammation/template)
- Boost — Conteneurs pour matrice
- Dlib — Optimisation non linéaire, fitting. Simple d'utilisation
- IML++, SparseLib++ — Inversion matrices creuses

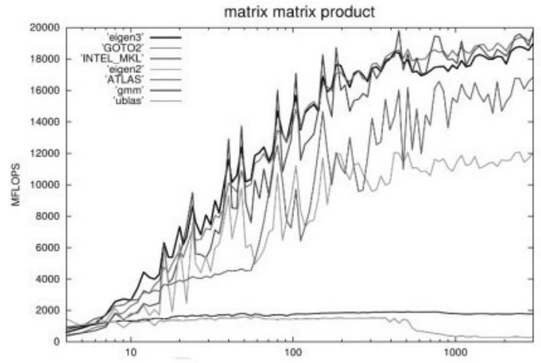
...

Eigen

- **Eigen**

  librairie C++ d'algèbre linéaire récente

  - Conteneur matrice, vecteur, matrice creuses
    Algorithme de résolution de systèmes et décomposition matricielles
  - Repose sur les templates / métaprogrammation
    Peut être plus rapide qu'un code C/Fortran équivalent
  - Utilisation instructions vectorielles, SSE



Eigen: Utilisation

Support vecteur nD

```
Eigen::Array2d p0(1.0,1.1);
Eigen::Array2d p1(0.4,-1.2);

std::cout<<p0+p1<<std::endl;
```

```
Eigen::ArrayXd p0(4);
p0[0]=1.0;p0[1]=2.2;p0[2]=4.7;p0[3]=-1.2;
p0=8.1*p0;

std::cout<<p0<<std::endl;
```

Eigen: Utilisation

Support matrices

```
Eigen::Matrix2f M;
M<<1.2,4.5,
  -0.1,2.2;

M(1,0)=-4.2;
M=2.0*M;

std::cout<<M<<std::endl;
```

2.4 9
-8.4 4.4

Eigen: Utilisation

Support inversion de matrices

```
Eigen::Matrix3f A;
A << 1,2,3, 4,5,6, 7,8,10;

std::cout<<A.inverse()<<std::endl;
```


Eigen: Utilisation

Support résolution de système linéaire Ax=b

```
Eigen::Matrix3f A;
Eigen::Vector3f b;
A << 1,2,3, 4,5,6, 7,8,10;
b << 3, 3, 4;
std::cout << "Here is the matrix A:\n" << A << std::endl;
std::cout << "Here is the vector b:\n" << b << std::endl;
Eigen::Vector3f x = A.colPivHouseholderQr().solve(b);
std::cout << "The solution is:\n" << x << std::endl;
```

Choix des approches

Decomposition	Method	Requirements on the matrix	Speed	Accuracy
PartialPivLU	partialPivLu()	Invertible	++	+
FullPivLU	fullPivLu()	None	-	+++
HouseholderQR	householderQr()	None	++	+
ColPivHouseholderQR	colPivHouseholderQr()	None	+	++
FullPivHouseholderQR	fullPivHouseholderQr()	None	-	+++
LLT	llt()	Positive definite	+++	+
LDLT	ldlt()	Positive or negative semidefinite	+++	++

Eigen: Utilisation

http://eigen.tuxfamily.org/

Support pour:

- Décompositions: LU, QR, valeurs singulières, Cholesky, Shur
- Matrices creuses
- Transformation géométriques