LEM2I

–

Bibliothèques Scientifiques

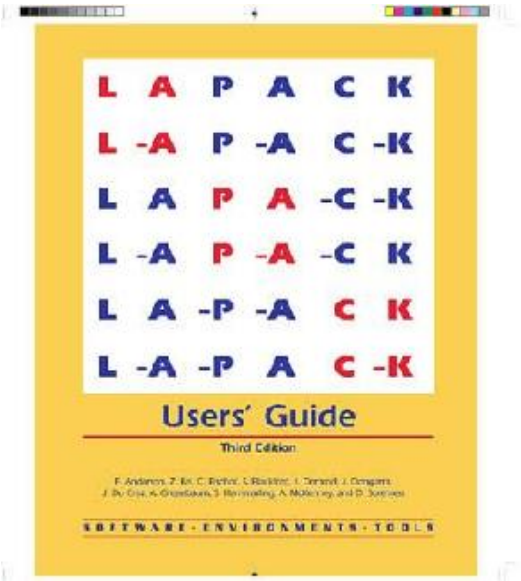
Violaine Louvet [1]

[1] ICJ - CNRS

Tipaza, janvier 2013

Objectifs de ce cours

- Comprendre ce qu'est une bibliothèque logicielle.
- Connaître les principales bibliothèques utilisées en calcul scientifique.
- Savoir utiliser quelques bibliothèques.



Sommaire

1. Motivations
2. Principales bibliothèques utilisées en calcul scientifique
   - Algèbre linéaire
   - Systèmes linéaires
   - Valeurs et vecteurs propres
   - Transformées de Fourier
   - Solveurs d'équations différentielles
   - Entrées/Sorties, visualisation
   - Parallélisation
   - Bibliothèques généralistes
3. Mécanismes d'optimisation
   - Problématique
   - Bibliothèques constructeurs
   - Optimisation des BLAS
   - FFTW
4. Interfaçage des langages
   - Appel croisé fortran et C/C++
   - Appel d'autres langages depuis Python
5. Utilisation de bibliothèques
   - Installation
   - Principes généraux
   - Problématique des licences
6. Références
7. Travaux pratiques
   - Calcul de valeurs propres
   - Résolution d'ODE non raides et raides

Sommaire

1. Motivations
2. Principales bibliothèques utilisées en calcul scientifique
   - Algèbre linéaire
   - Systèmes linéaires
   - Valeurs et vecteurs propres
   - Transformées de Fourier
   - Solveurs d'équations différentielles
   - Entrées/Sorties, visualisation
   - Parallélisation
   - Bibliothèques généralistes
3. Mécanismes d'optimisation
   - Problématique
   - Bibliothèques constructeurs
   - Optimisation des BLAS
   - FFTW
4. Interfaçage des langages
   - Appel croisé fortran et C/C++
   - Appel d'autres langages depuis Python
5. Utilisation de bibliothèques
   - Installation
   - Principes généraux
   - Problématique des licences
6. Références
7. Travaux pratiques
   - Calcul de valeurs propres
   - Résolution d'ODE non raides et raides

Intérêt des bibliothèques

- Capitaliser, réutiliser
  - Réduction du temps de développement.
  - Portabilité.
  - Performance, optimisation.
  - Fiabilité, stabilité.
  - Communauté utilisateurs, support.

Besoins récurrents

- Algèbre Linéaire
  - dense
  - creuse
- Résolution de systèmes linéaires
  - directe
  - itérative
- Résolution d'EDO
- Calcul de valeurs propres
- FFT
- Générateurs aléatoires
- Optimisation
- Outils système
  - mesure du temps
  - threads
- Communications
  - MPI
  - ...
- Entrées/Sorties
  - Fichiers normalisés (hdf5, xml)
  - Visualisation

Sommaire

1 Motivations

2 Principales bibliothèques utilisées en calcul scientifique
- Algèbre linéaire
- Systèmes linéaires
- Valeurs et vecteurs propres
- Transformées de Fourier
- Solveurs d'équations différentielles
- Entrées/Sorties, visualisation
- Parallélisation
- Bibliothèques généralistes

3 Mécanismes d'optimisation
- Problématique
- Bibliothèques constructeurs
- Optimisation des BLAS
- FFTW

4 Interfaçage des langages
- Appel croisé fortran et C/C++
- Appel d'autres langages depuis Python

5 Utilisation de bibliothèques
- Installation
- Principes généraux
- Problématique des licences

6 Références

7 Travaux pratiques
- Calcul de valeurs propres
- Résolution d'ODE non raides et raides

Algèbre linéaire, BLAS

Basic Linear Algebra Subroutines

- Opérations algébriques de bas niveau
  - level 1 : opérations sur des vecteurs

$$y = \alpha \times x + y$$

  - level 2 : opérations matrices/vecteurs

$$y = \alpha \times A \times x + y$$

  - level 3 : opérations matrices/matrices

$$C = \alpha \times A \times B + C$$

- En fortran 77.
- Diffusée en 1979.
- Base de très nombreuses bibliothèques.
- `http://www.netlib.org/blas`.
- Différentes implémentations : autres langages, parallèles ...

Algèbre linéaire

- ✓ Blitz++
  - Bibliothèque générique de tableaux et vecteurs en C++.
  - Utilise intensivement les « Expression Templates »
  - `http://www.oonumerics.org/blitz/`
- ✓ SparseKit
  - Manipulation de matrices cresues de différents formats.
  - Ecrit en Fortran 90.
  - `http://people.sc.fsu.edu/~burkardt/f_src/sparsekit/sparsekit.html`

Systèmes linéaires, LAPACK

Linear Algebra PACKage

- Algèbre linéaire de haut niveau.
- Matrices pleines ou bandes, mais pas adapté aux matrices creuses.
- Factorisations LU, Cholesky, QR et Schur, valeurs propres et vecteurs propres, décomposition en valeurs singulières ...
- Utilise intensivement les BLAS.
- Base de nombreux outils numériques (Matlab, Scilab ...).
- Ecrit en fortran 77.
- Dernière version : 3.2.1, avril 2009.
- `http://www.netlib.org/lapack`.

Systèmes linéaires

- ✓ PETSc
  - Gestion de Matrices et vecteurs parallèles (basé sur MPI).
  - Solveurs itératifs de systèmes linéaires parallèles.
  - `http://www-unix.mcs.anl.gov/petsc/petsc-2/`
- ✓ SuperLU
  - Résolution directe de très grands systèmes creux non symétriques par factorisation LU.
  - Plusieurs versions séquentielles et parallèles.
  - `http://crd.lbl.gov/~xiaoye/SuperLU/`
- ✓ MUMPS
  - Solveur direct parallèle de systèmes linéaires creux.
  - Différents types de matrices (symétriques définies positives, symétriques générales, et non-symétriques générales).
  - `http://graal.ens-lyon.fr/MUMPS/`
- ✓ PaStiX, Parallel Sparse matriX package
  - Résolution de très grand systèmes linéaires creux en utilisant une méthode directe.
  - Parallélisme de type MPI et/ou Thread.
  - `http://pastix.gforge.inria.fr/`

Systèmes linéaires

- ✓ HIPS, Hierarchical Iterative Parallel Solver
  - Méthodes de résolution hybride directe/itérative.
  - `http://hips.gforge.inria.fr/`
- ✓ Scotch
  - Partitionneur séquentiel et parallèle de graphes et renuméroteur de matrices creuses.
  - Intégré dans les solveurs MUMPS et PaStiX.
  - `http://www.labri.fr/~pelegrin/scotch/`
- ✓ Metis
  - Partitionneur de graphes, de maillages et renumérotation de matrices creuses.
  - Intégré notamment dans les solveurs MUMPS et HIPS.
  - `http://glaros.dtc.umn.edu/gkhome/views/metis`
- ✓ Hypre
  - Résolution de très grands sytèmes linéaires creux sur machine parallèle
  - Préconditionneurs performants dont multigrille sur grille structurée et non structurée
  - `http://www.llnl.gov/CASC/linear_solvers/`

Valeurs et vecteurs propres

- ✓ Lapack
  - Matrices denses
- ✓ ARPACK, ARnoldi PACKage
  - Calcul des valeurs propres et des vecteurs propres de grosses matrices creuses.
  - Basée sur les algorithmes itératifs de Lanczos/Arnoldi.
  - Ecrit en fortran 77.
  - `http://www.caam.rice.edu/software/ARPACK/`.
- ✓ SLEPc
  - Implémentation parallèle de recherche de valeurs propres. Décomposition en valeurs singulières.
  - Construit au dessus de PETSc.
  - `http://www.grycap.upv.es/slepc/`.

Transformées de Fourier, résolution d'ODE

- ✓ FFTW
  - Fast Fourier Transform in the West
  - `http://www.fftw.org/`
- ✓ FFTPACK
  - Package fortran pour la FFT.
  - `http://www.netlib.org/fftpack/`.

Résolution d'ODE

- ✓ ODEPACK
  - Solveurs écrits en fortran 77 pour la résolution de systèmes différentiels ordinaires.
  - `https://computation.llnl.gov/casc/odepack/odepack_home.html`.
- ✓ GSL (Gnu Scientific Library)
  - Large choix de routines mathématiques écrites en C/C++ dont des solveurs d'ODE
  - `http://www.gnu.org/software/gsl/gsl.html`
- ✓ SUNDIALS, SUite of Nonlinear and DIfferential/ALgebraic equation Solvers
  - Résolution de systèmes d'ODE et d'ADE.
  - Ecrit en C. `https://computation.llnl.gov/casc/sundials/main.html`.

Entrées/Sorties

- ✓ HDF5, Hierarchical Data Format
  - Format standardisé d'entrées/sorties, séquentielles et parallèles.
  - Existence d'API en C, C++ et Fortran 90.
  - `www.hdfgroup.org/HDF5/`.
- ✓ XML, Extensible Markup Language
  - Langage informatique de balisage générique.
  - Nombreuses librairies permettant la génération de fichiers en XML.
  - `http://www.w3.org/XML/`.

Visualisation

- ✓ VTK, Visualization ToolKit
  - Bibliothèque pour la visualisation de gros volumes de données 2D ou 3D.
  - Ecrite en C++. Peut être utilisée soit directement via les langages C++, Python, ..., soit indirectement via des interfaces graphiques telles que Paraview ou Mayavi.
  - `http://public.kitware.com/VTK/`.
- ✓ OpenDX, Open Data Explorer
  - Bibliothèque d'IBM pour la visualisation de données scientifiques.
  - `http://www.opendx.org/`.

Parallélisation

- ✓ Bibliothèques MPI
  - MPICH, `http://www-unix.mcs.anl.gov/mpi/mpich/`.
  - LAM/MPI, `http://www.lam-mpi.org/`.
  - openMPI, `http://www.open-mpi.org/`.
- ✓ Bibliothèques de threads
  - TBB, Threading Building Blocks, `http://www.threadingbuildingblocks.org/`.
  - Boost Threads, `http://www.boost.org/`.
  - Norme POSIX : fournit un ensemble de primitives permettant de réaliser des threads, les pthreads.
- ✓ Corba, Common Object Request Broker Architecture
  - Architecture logicielle pour le développements de composants distribués, `http://www.omg.org/`.
  - Nombreuses implémentations existantes .

Bibliothèques généralistes et autres outils

- ✓ BOOST
  - Bibliothèques C++ généralistes offrant de nombreuses fonctionnalités : outils systèmes, threads, sérialisation, interface C++/Python ...
  - `http://www.boost.org/`.
- ✓ GSL
  - Gnu Scientific Library, bibliothèque généraliste en C/C++
  - `http://www.gnu.org/software/gsl/gsl.html`
- ✓ Bibliothèques Eléments finis
  - GetFEM++, bibliothèque C++ d'éléments finis, `http://home.gna.org/getfem/`.
  - MELINA, bibliothèque de calculs éléments finis en fortran, `http://perso.univ-rennes1.fr/daniel.martin/melina/`.
  - OFELI, bibliothèque éléments finis orienté objet en C++, `http://www.ofeli.net/`.
  - ...

Bibliothèques généralistes et autres outils

- ✓ Outils généralistes
  - Trilinos, boîtes à outils d'algorithmes performants, `http://trilinos.sandia.gov`
  - OpenFOAM, outil pour la CFD, `http://www.openfoam.com`
  - ...
- ✓ Générateurs de nombres aléatoires
  - GSL, `http://www.gnu.org/software/gsl/gsl.html`
  - BOOST, `http://www.boost.org/`
  - Fonctions intrinsèques en Fortran et en C++

Sommaire

1 Motivations
2 Principales bibliothèques utilisées en calcul scientifique
- Algèbre linéaire
- Systèmes linéaires
- Valeurs et vecteurs propres
- Transformées de Fourier
- Solveurs d'équations différentielles
- Entrées/Sorties, visualisation
- Parallélisation
- Bibliothèques généralistes

3 Mécanismes d'optimisation
- Problématique
- Bibliothèques constructeurs
- Optimisation des BLAS
- FFTW

4 Interfaçage des langages
- Appel croisé fortran et C/C++
- Appel d'autres langages depuis Python

5 Utilisation de bibliothèques
- Installation
- Principes généraux
- Problématique des licences

6 Références

7 Travaux pratiques
- Calcul de valeurs propres
- Résolution d'ODE non raides et raides

Problématique

- ✓ Obtenir de bonnes performances.
- ✓ Sur une grande variété de matériels.
- ✓ Prenant en compte les caractéristiques de la machine.
- ✓ Tout en restant portable !

Bibliothèques constructeurs

- Certaines bibliothèques très utilisées ont été optimisées sur les différentes architectures par les constructeurs : par exemple blas et lapack. Il faut les utiliser lorsqu'elles existent !
  Avantages : Performances optimales pour les machines concernées, rien à réécrire dans le code.

- D'autres bibliothèques conçues par les constructeurs ne sont pas portables : leur interface change d'un constructeur à l'autre. C'est le cas des FFT.
  Inconvénients : non portable, prévoir des interfaces différentes selon les machines.

Optimisation des BLAS

- ATLAS, Automatically Tuned Linear Algebra Software
  - ✓ API BLAS en C et fortran 77, implémentant quelques fonctionnalités de LAPACK.
  - ✓ Automated Empirical Optimization of Software (AEOS) : **compilation adaptive** qui ajuste les paramètres en fonction des caractéristiques de la machine.
  - ✓ `http://math-atlas.sourceforge.net/`
- GOTO BLAS
  - ✓ **Réimplémentation** des BLAS.
  - ✓ Prise en compte des caractéristiques des architectures actuelles.
  - ✓ `http://www.tacc.utexas.edu/tacc-projects/`

FFTW, Fastest Fourier Transform Of the West

- Bibliothèque de fonctions écrites en C permettant d'effectuer des calculs de transformées de Fourier discrètes en une ou plusieurs dimensions.
- Adapte le choix de l'algorithme aux détails du matériel sous-jacent de manière à réaliser les meilleures performances.
- La transformée se fait en 2 phases :
  - **Planification FFTW** : « apprend » la meilleure stratégie pour calculer la transformée sur la machine cible.
  - **Exécution** de la/les transformées en fonction de ce plan.

Sommaire

1 Motivations
2 Principales bibliothèques utilisées en calcul scientifique
- Algèbre linéaire
- Systèmes linéaires
- Valeurs et vecteurs propres
- Transformées de Fourier
- Solveurs d'équations différentielles
- Entrées/Sorties, visualisation
- Parallélisation
- Bibliothèques généralistes

3 Mécanismes d'optimisation
- Problématique
- Bibliothèques constructeurs
- Optimisation des BLAS
- FFTW

4 Interfaçage des langages
- Appel croisé fortran et C/C++
- Appel d'autres langages depuis Python

5 Utilisation de bibliothèques
- Installation
- Principes généraux
- Problématique des licences

6 Références

7 Travaux pratiques
- Calcul de valeurs propres
- Résolution d'ODE non raides et raides

Motivations

- ✓ Existence de nombreuses bibliothèques fortran très optimisées.
- ✓ Utiliser le meilleur des différents langages.

ATTENTION

- Différences entre les **types**.
- Différences dans la **gestion des I/O**.
- Différences dans la **gestion de la mémoire** ...

Appel croisé fortran et C/C++

- Echanges de données : Type de données communs :
  - entier,
  - flottants,
  - double précision.
- Attention au stockage des tableaux :
  - row order pour fortran,
  - column order pour C/C++.
- Attention à l' indexation :
  - débute à 1 en fortran,
  - à 0 en C/C++.

En pratique :

- ✓ Définir un **prototype** pour la fonction appelée.
- ✓ Comme un appel de fonction C depuis le C++ : précéder le prototype de la fonction par le **mot clé extern suivi de "C"**.
- ✓ Attention au nom de la fonction fortran dans le code C++, en **minuscule et terminé par « _ »**.
- ✓ Passage des paramètres par **adresse**.

Appel d'autres langages depuis Python

- Appels de code C++ depuis Python :
  - ✓ SWIG, Simplified Wrapper and Interface Generator, appels de programmes en C et C++ depuis des langages de programmation haut niveau (Python, PHP, Perl ...), `http://www.swig.org/`
  - ✓ BoostPython, interopérabilité C++/Python, `http://www.boost.org/doc/libs/1_42_0/libs/python/doc/index.html`
  - ✓ Pyrex, permet de mixer du code Python et C, `http://www.cosc.canterbury.ac.nz/greg.ewing/python/Pyrex/`
- Appels de code fortran depuis Python :
  - ✓ f2py, Fortran to Python interface generator, `http://cens.ioc.ee/projects/f2py2e/`
  - ✓ PyFort, création d'extensions python à partir de routines fortran, `http://pyfortran.sourceforge.net/`

Sommaire

1 Motivations
2 Principales bibliothèques utilisées en calcul scientifique
- Algèbre linéaire
- Systèmes linéaires
- Valeurs et vecteurs propres
- Transformées de Fourier
- Solveurs d'équations différentielles
- Entrées/Sorties, visualisation
- Parallélisation
- Bibliothèques généralistes

3 Mécanismes d'optimisation
- Problématique
- Bibliothèques constructeurs
- Optimisation des BLAS
- FFTW

4 Interfaçage des langages
- Appel croisé fortran et C/C++
- Appel d'autres langages depuis Python

5 Utilisation de bibliothèques
- Installation
- Principes généraux
- Problématique des licences

6 Références
7 Travaux pratiques
- Calcul de valeurs propres
- Résolution d'ODE non raides et raides

Installation

Quand on a identifié la bibliothèque qui nous intéresse, plusieurs possibilités :

- Elle est packagée dans les distributions Linux
  - Installation via le gestionnaire de paquet de la distribution
    apt-get install libfftw3-3
  - Attention, dans ce cas, l'installation se fait sous forme binaire : pas de compilation donc potentiellement moins bonnes performances
- Elle est fournie sous forme d'archive de ses sources
  - fichier tar.gz
  - configure, make, make install
  - à la main (il y a quand même en général un Makefile)
- Exemple : installation de FFTW via ses sources

Principes généraux

- programme = A
- bibliothèque statique = B (extension .a)
- bibliothèque dynamique = C (extension .so)

Compilation en dynamique : l'exécutable ne contient que A mais il faut que C soit toujours installé sinon on ne peut pas utiliser le programme.

Compilation en statique : l'exécutable contient A+B (plus précisément : A+(copie de B)). Ca va toujours marcher même sur un autre système où il n'y a ni B ni C.

En pratique

Edition des liens

- Compilation du code avec référence aux bibliothèques utilisées par les options :
  - *-L/chemin/vers/libnom.so* : pour préciser le chemin
  - *-lnom* : quand la bibliothèque a été installée de façon classique
- Attention à l'ordre d'appel des bibliothèques

Exécution du programme

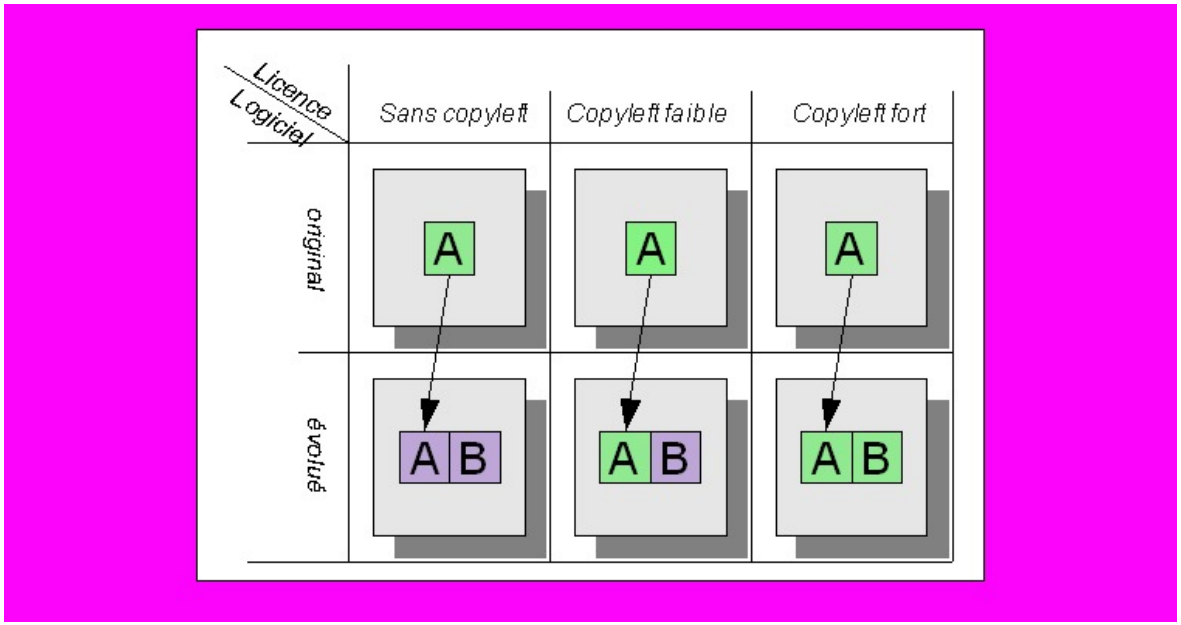
- On peut voir les dépendances d'un code avec l'outil ldd
- La variable d'environnement *LD_LIBRARY_PATH* permet d'indiquer l'emplacement de bibliothèques
- En général, le système va chercher dans */usr/lib*, puis *lib* (défini dans */etc/ld.so.conf*).

Problématique des licences

- NE PAS utiliser des briques logicielles sans connaître leur origine, leur licence.
- NE PAS diffuser du code sans licence.

On n'a pas le droit de diffuser comme on veut : du point de vue juridique, seul le titulaire des droits patrimoniaux (celui qui paye) peut décider d'une diffusion du logiciel en libre ou non.

Certaines licences sont contaminantes :



Sommaire

1 Motivations
2 Principales bibliothèques utilisées en calcul scientifique
- Algèbre linéaire
- Systèmes linéaires
- Valeurs et vecteurs propres
- Transformées de Fourier
- Solveurs d'équations différentielles
- Entrées/Sorties, visualisation
- Parallélisation
- Bibliothèques généralistes

3 Mécanismes d'optimisation
- Problématique
- Bibliothèques constructeurs
- Optimisation des BLAS
- FFTW

4 Interfaçage des langages
- Appel croisé fortran et C/C++
- Appel d'autres langages depuis Python

5 Utilisation de bibliothèques
- Installation
- Principes généraux
- Problématique des licences

6 Références

7 Travaux pratiques
- Calcul de valeurs propres
- Résolution d'ODE non raides et raides

Références

- Listes de logiciels :
  - http://www.projet-plume.org/maths
  - http://www.ann.jussieu.fr/~lehyaric/freesoft/
  - http://www.twiki.org/cgi-bin/view/Main/FlorianDeVuyst
  - http://www.netlib.org/
- Problématique des licences :
  - http://www.projet-plume.org/fr/ressource/faq-licence-copyright

Sommaire

1 Motivations
2 Principales bibliothèques utilisées en calcul scientifique
- Algèbre linéaire
- Systèmes linéaires
- Valeurs et vecteurs propres
- Transformées de Fourier
- Solveurs d'équations différentielles
- Entrées/Sorties, visualisation
- Parallélisation
- Bibliothèques généralistes

3 Mécanismes d'optimisation
- Problématique
- Bibliothèques constructeurs
- Optimisation des BLAS
- FFTW

4 Interfaçage des langages
- Appel croisé fortran et C/C++
- Appel d'autres langages depuis Python

5 Utilisation de bibliothèques
- Installation
- Principes généraux
- Problématique des licences

6 Références

7 Travaux pratiques
- Calcul de valeurs propres
- Résolution d'ODE non raides et raides

TP 1 : Calcul de valeurs propres

Objectifs

Calculer les valeurs propres d'une matrice dense générée aléatoirement.

Contraintes

- Utiliser le langage Fortran comme langage principal
- Utiliser le générateur de nombres aléatoires de la GSL
- Utiliser Lapack pour calculer les valeurs et vecteurs propres

Etapes du TP

- Ecrire un wrapper pour l'appel des routines de la GSL. Pour cela, se baser sur l'exemple fourni dans le manuel : `file:///usr/share/doc/gsl-ref-html/index.html`
- Ecrire le code fortran appelant ce wrapper pour la génération aléatoire de la matrice
- Déterminer la routine Lapack à utiliser (voir `file:///usr/share/doc/liblapack-doc/lug/index.html`) et appeler la dans le programme fortran

Compilation

- On utilise *CMake*
- Ecrire un *CMakeLists.txt* avec les commandes suivantes :
  - enable_language
  - add_executable
  - target_link_libraries

TP 2 : Résolution d'ODE

- On s'intéresse aux équations des réactions chimiques :
  $A + B \longrightarrow C + D$
- Depuis les travaux de B. Belousov et quelques années plus tard A. M. Zhabotinskii, on connaît plusieurs mélanges d'espèces chimiques qui réagissent de manière oscillante.
- On va étudier 2 cas :
  - Le Brusselator
  - L'Oregonator

Système d'équations du Brusselator

- Le Brusselator est un système hypothétique modèle de réactions chimiques à plusieurs composantes, auto-catalytiques et oscillantes.
- Après quelques simplifications, on se ramène ainsi à un système à 2 équations :

$$\begin{cases} \dot{X} &= A + X^2 Y - (B+1)X \\ \dot{Y} &= BX - X^2 Y \end{cases}$$

- Point fixe : $(A, \frac{B}{A})$, stable pour $B < A^2 + 1$ (donc attracteur), instable sinon, le système évoluant vers un cycle limite.

Etapes du TP

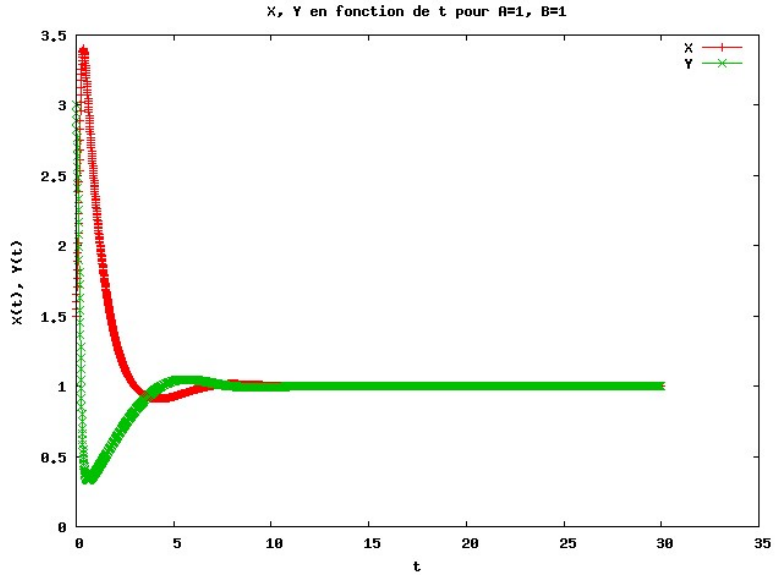
- ✓ On utilise un schéma de Runge-Kutta d'ordre 4 pour résoudre le système.
- ✓ Compilation et exécution du code fourni.
  - Compilation

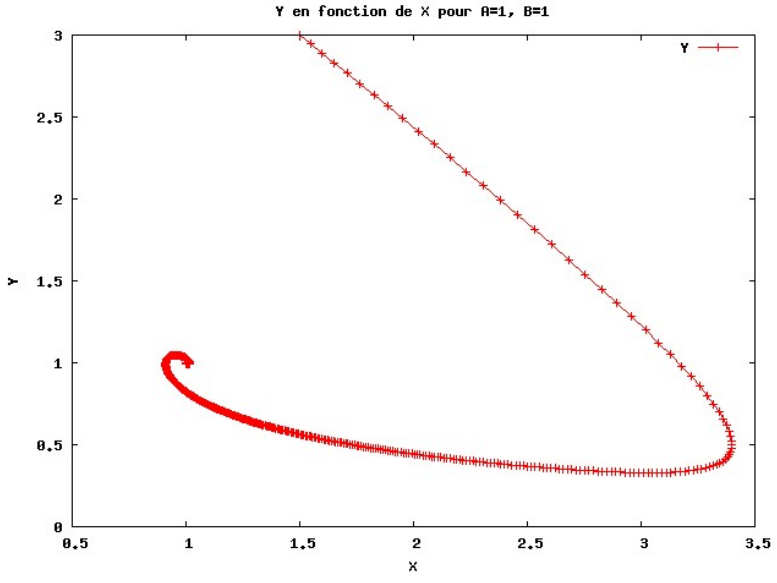
    ```
    mkdir build ; cd build ; cmake ..
    ```
  - Exécution

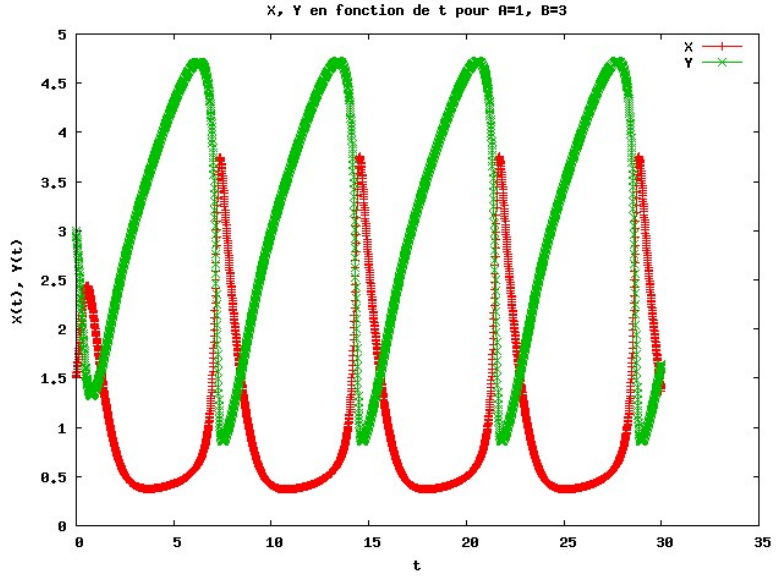
    ```
    build/bruss > ../res
    ```
  - Visualisation

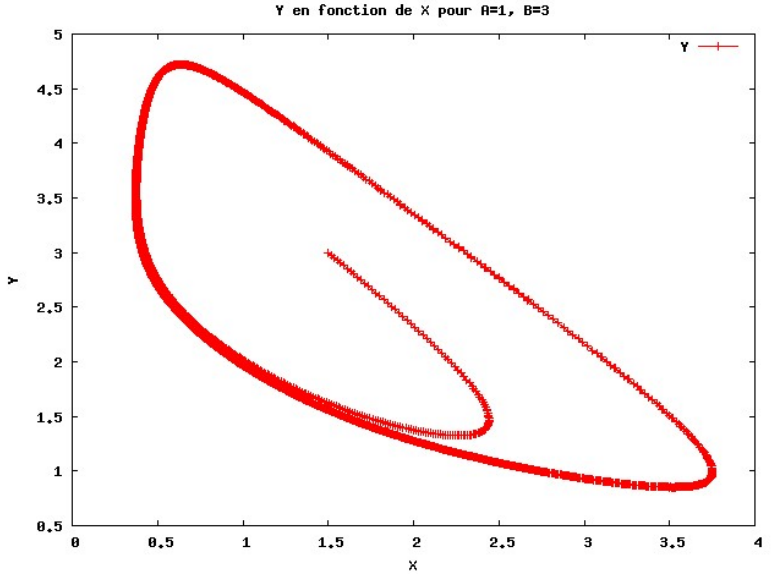
    ```
    cd .. ; ./visu
    ```
- ✓ On choisit $A = 1$ et on fait varier B autour de la valeur limite $A^2 + 1 = 2$ (par exemple, $B = 1$, $B = 3$).

Résultats sur le Brusselator









Système d'équations de l'Oregonator

- Système sur les espèces intermédiaires X, Y et Z.

$$\begin{cases} \dfrac{dX}{dt} &= p(Y - XY + X - rX^2) \\ \dfrac{dY}{dt} &= \dfrac{1}{p}(-Y - XY + fZ) \\ \dfrac{dZ}{dt} &= q(X - Z) \end{cases}$$

avec $f = 1$, $p = 77.27$, $q = 0.161$ et $r = 8.375.10^{-6}$.

- Système raide !

Etapes du TP

- ✓ On remplace le système d'équations du Brusselator par celui de l'Oregonator dans le code avec les conditions intiales :

$$\begin{cases} X(0) &= 1. \\ Y(0) &= 2. \\ Z(0) &= 3. \end{cases}$$

  - Compilation

    `mkdir build ; cd build ; cmake ..`
  - Exécution

    `./orego > ../res`
  - Visualisation

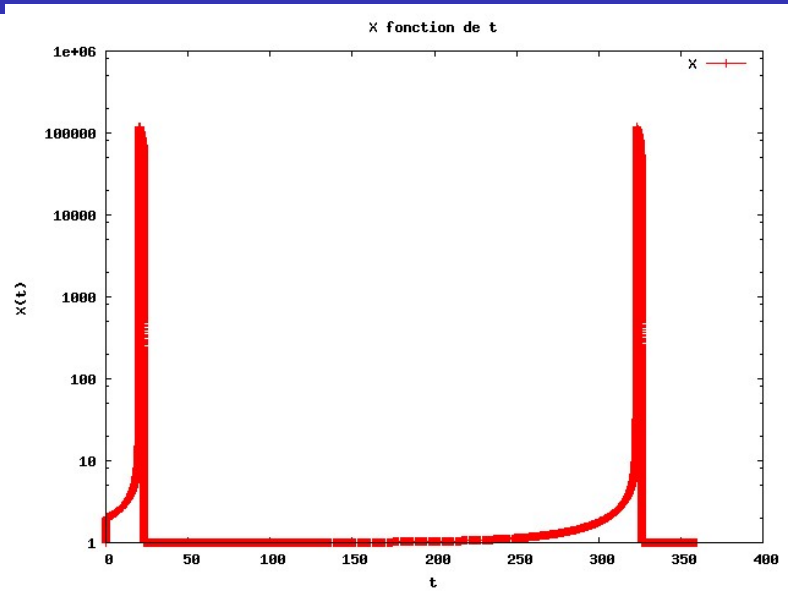
    `cd .. ; ./visu`

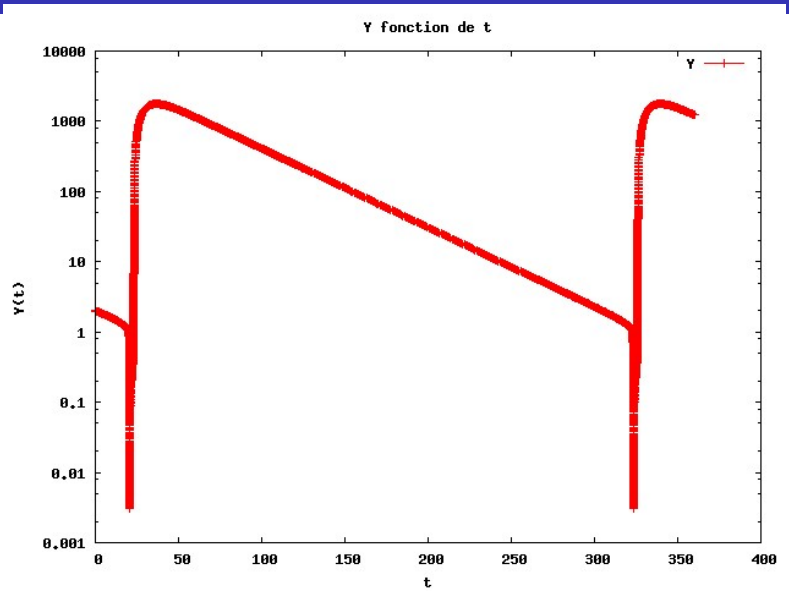
- ✓ Que se passe-t-il ?

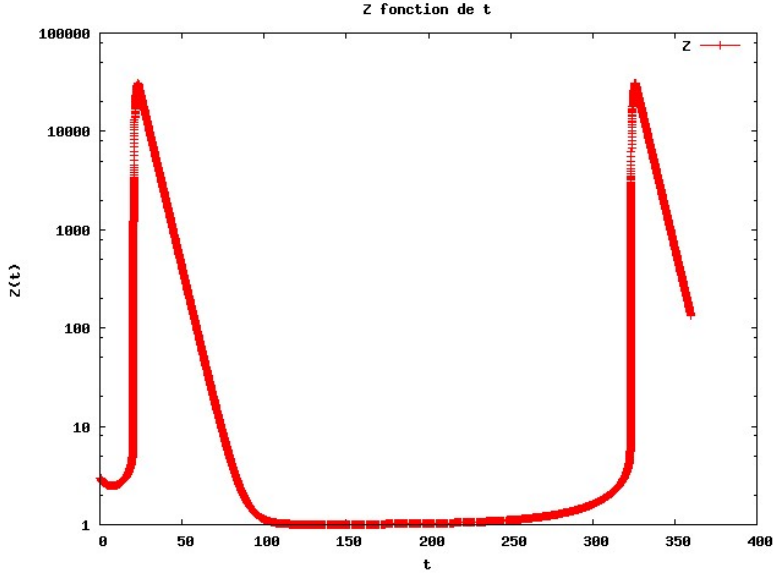
Utilisation d'un solveur dédié

- On utilise un schéma Backward Differentiation Formula adapté aux systèmes raides : bibliothèque ODEPACK
- Remplacer l'appel à *rk4* par un appel à *lsode* :
  - Etudier comment se présente cette bibliothèque et l'appel à dlsode, la routine qui nous intéresse :
    `cd odepack ; less opkdmain.f`
  - Ecrire le fichier *CMakeLists.txt* pour la compilation d'odepack sous forme de bibliothèque (utilisation de la commande *add_library*)
  - Modifier le fichier fortran pour appeler *dlsode* à la place de *rk4* :
    - Ajout des variables et tableaux de travail de dlsode
    - Initialisation des ces variables et tableaux
    - Modification de la fonction *derive* : réfléchir à la façon de passer les paramètres du modèle
    - Ajout d'une routine pour le calcul du jacobien
    - Modification du fichier *CMakeLists.txt* pour le linkage avec la bibliothèque odepack (utilisation de la commande *target_link_libraries*)
- Compiler, exécuter et visualiser les résultats comme précédemment

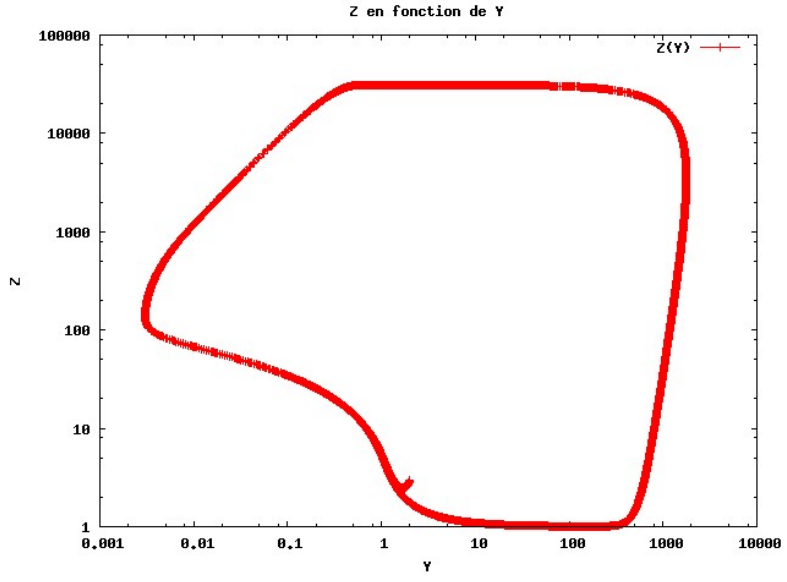
Résultats

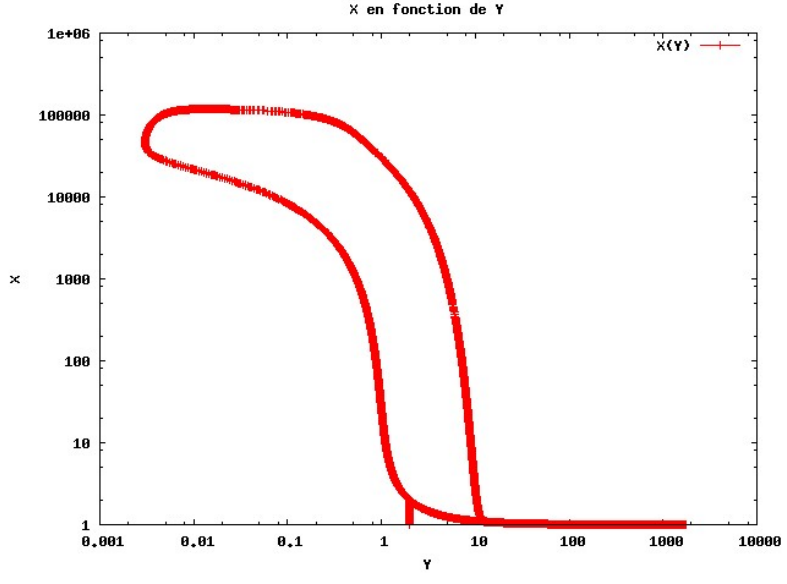

X fonction de t
X
X(t)
t
1e+06
100000
10000
1000
100
10
1
0
50
100
150
200
250
300
350
400


Y fonction de t
Y
Y(t)
t
10000
1000
100
10
1
0.1
0.01
0.001
0
50
100
150
200
250
300
350
400


Z fonction de t
Z
Z(t)
t
100000
10000
1000
100
10
1
0
50
100
150
200
250
300
350
400

Résultats


Z en fonction de Y
Z(Y)
100000
10000
1000
100
10
1
Z
0.001
0.01
0.1
1
10
100
1000
10000
Y


X en fonction de Y
X(Y)
1e+06
100000
10000
1000
100
10
1
X
0.001
0.01
0.1
1
10
100
1000
10000
Y