Modélisation et résolutions numérique et symbolique

via les logiciels Maple et Matlab

Jeremy Berthomieu   Mohab Safey El Din   Stef Graillat
Mohab.Safey@lip6.fr

Format du cours

Équipe pédagogique : Stef Graillat (responsable), Jeremy Berthomieu, Mohab Safey El Din

Évaluation des connaissances

- un examen réparti 1 (25%), un examen réparti 2 (60%) et une note de TD/TME (15%)
- 10 séances de TME (dont certains seront à rendre)

Horaire

- Cours : le jeudi de 13h30 à 15h30 (salle 23-24/102)
- TD/TME : le vendredi de 13h30 à 15h30 (salle 23-24/102) et de 15h45 à 17h45 (salle 31/313)

Site web :
`http://www-pequan.lip6.fr/~graillat/teach/model/index.html`

Objectives of the course

- Mathematical concepts in Computer Science : basic definitions
- Algorithms: on mathematical structure $\rightsquigarrow$ how to compute efficiently and reliably ?
- Problem solving : several models of resolution

Application fields

- cryptology
- data-mining
- signal theory and computer vision
- image rendering
- robotics
- computational geometry
- biology
- etc.

Symbolic computation/Numerical computation

Paradigms of solving:

- Global Approaches: Compute **all** the solutions and ensure the quality of the returned results.
- Local (iterative) Approaches: Compute an *approximation* of a local solution (in a prescribed domain) by iterating a certain function.

Example: Solve $f = 0$ where $f \in \mathbb{Q}[X]$

- "Compute" all the real roots (or the complex ones)
- Decide the existence of a real root
- Return an approximation "close" to a real root (or a complex one)

Family of algorithms:

- Symbolic Algorithms: provide an exact representation of the result but sometimes are slow $\rightsquigarrow$ Computer Algebra
  Cryptology, Error correcting codes, Comp. Geometry, classification problems

Family of algorithms:

- Symbolic Algorithms: provide an exact representation of the result but sometimes are slow $\rightsquigarrow$ Computer Algebra
  Cryptology, Error correcting codes, Comp. Geometry, classification problems
- Numerical Algorithms : based on finite precision arithmetic, in general faster but may provide wrong results $\rightsquigarrow$ Numerical Computing
  Numerical validation, Computer arithmetics, Numerical simulation

Family of algorithms:

- Symbolic Algorithms: provide an exact representation of the result but sometimes are slow $\rightsquigarrow$ Computer Algebra
  Cryptology, Error correcting codes, Comp. Geometry, classification problems
- Numerical Algorithms : based on finite precision arithmetic, in general faster but may provide wrong results $\rightsquigarrow$ Numerical Computing
  Numerical validation, Computer arithmetics, Numerical simulation

$\rightarrow$ Hybrid symbolic-numeric computing : use both advantages of approaches

General outline of the course

1. Introduction to Maple
2. Rings, Fields, multi-precision and multi-modular arithmetic
3. Application to linear solving (Gauss paradigm) – Evaluation/Interpolation techniques
4. Euclid's paradigm and non-linear solving
5. Introduction to MATLAB
6. Singular Value Decomposition, application to image compression
7. Eigenvectors/eigenvalues and application to Google PageRank algorithm
8. Discrete Fourier Transform, application to signal theory
9. Monte-Carlo Methods, application to pricing option in finance

Bibliographie

- Maple, Règles et fonctions essentielles, N. Puech, Springer, 2009
- Maple Sugar: Une initiation progressive à Maple, G. Le Bris, Cassini, 1999
- Introduction to Maple, A. Heck, 3e édition, Springer, 2003
- Essential Maple 7: An Introduction for Scientific Programmers, R. Corless, Springer, 2002
- Maple and Mathematica: A Problem Solving Approach for Mathematics, I. Shingareva et C. Lizarraga-Celaya, Springer, 2007
- Maple - Son bon usage en mathématiques, Ph. Dumas et X. Gourdon, Springer, 1997

General overview of Maple

- Software widely used in industry, research centers and for teaching
- Acronym for "MAthematical PLEasure"
- Initially, it was developped at the Univ. of Waterloo (Ontario, Canada).
  Then it became a product of Waterloo Maple Software Corporation.
  Bought by Toyota in 2010.

General overview of Maple

Advantages :

- computer algebra software
- interactivité/interface pratique :
  - programmation rapide et simple, facilité pour tester et modifier
  - oubli du bas niveau ; on se consacre pleinement à l'aspect algorithmique
- riche en fonctionnalités
- prises en compte des notions mathématiques formelles
- retourne des résultats sous la forme d'objets mathématiques
- permet l'interfacage avec des codes C et C++ (librairie dynamique)

Disadvantages :

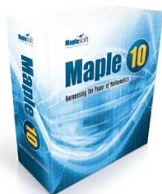
- pas de notion de compilation, peu d'entrées/sorties
- sémantique du langage pas toujours claire

Lancement de Maple

Le nom du fichier exécutable est

- `maple`, en mode texte
- `xmaple`, avec interface graphique
  plus légère avec l'option `-cw` (classic worksheet).

Sur les machines de l'ARI, il s'agit de Maple 10. Le programme est situe dans le répertoire `/usr/local/maple10/bin`



Quelques points de syntaxe

- L'instruction `quit` permet de quitter une session Maple
- L'aide sur une fonction, un type, . . . s'appelle par l'instruction `?` suivi du mot concerné. Par exemple `?string`
- Une instruction se termine par un point-virgule ; (sauf l'aide et quit) ou un double point :
  Le point-virgule ; permet d'exécuter la commande et d'afficher le résultat. Si l'on veut simplement exécuter les commandes sans voir le résultat s'afficher, il faut remplacer le point-virgule par deux points :
- Touche `Entree` $\rightarrow$ Exécution
- Commentaires : après un dièse #
- La commande `restart` remet tout dans l'état initial (annule toutes les définitions). Cela devrait être la première instruction de tout programme Maple

Quelques points de syntaxe

Le symbole > représente le prompt. Plusieurs instructions peuvent être regroupées sous un même prompt : elles forment un bloc d'instructions. Les instructions formant un bloc seront exécutées à l'aide d'une seule validation (par la touche `Entree`). Pour entrer plusieurs commandes dans le même bloc d'instructions, il suffit de taper `Maj` puis `Entree`

Il faut noter que Maple respecte la casse des lettres c'est-à-dire fait la différence entre majuscules et minuscules.

Il est aussi possible **(fortement recommandé)** d'écrire des scripts Maple (fichiers texte) dont l'exécution est lancée par `read "nom_fichier";` (utile pour écrire de gros codes).

Alternative

```
if cond1 then
   instructions1
else
   instructions2
fi
```

Si plusieurs alternatives imbriquées

```
if cond1 then
   instructions1
elif cond2 then
   instructions2
else
   instructions3
fi
```

Boucle for

```
for var from expIni to expFin do
   instructions
od;
```

Changer le pas de la variable de boucle

```
for var from expIni to expFin by pas do
   instructions
od;
```

Boucle Tant que

```
while expr do
   instructions
od;
```

Fonctions et procédures

Définition par le mot-clef `proc`

Syntaxe :
Le mot-clef `proc` permet de définir une fonction en respectant la syntaxe suivante :

```
MAFONCTION:=proc( param )
   d\'eclarations optionnelles
   corps de la fonction
end
```

Puis

```
MAFONCTION(param);
```

Visibilité des variables

Variable globale : variable présente dans l'environnement d'exécution jusqu'à la fin de la session

- Déclaration explicite : Mot clef `global`
- Déclaration explicite d'une variable globale à l'intérieur d'une fonction: dans la partie "déclarations optionnelles", par

  ```
  global s\'equence_des_variables_globales
  ```

Variable locale : variable qui n'est accessible qu'à l'intérieur d'une fonction. Elle ne vit que le temps d'exécution de cette fonction.

L'espace mémoire qui lui est alloué est libéré à la fin de l'exécution de la fonction.

- Déclaration explicite : mot clef `local`
- Déclaration explicite par

  ```
  local s\'equence_variables_locales
  ```

Exemple de fonctions

```
fibo:= proc(n)
if n<=1 then return 1;
else return fibo(n-1)+fibo(n-2);
end:
```

```
toto:=proc(n)
  local res;
  res:=1;
  for i from 1 to n do
    res:=res*i;
  od;
end;
```

```
foo:=proc(a, b)
if a=b then return a;
else
   if a>=b then return foo(a-b, b);
   else return foo(b-a, a);
   fi;
end;
```

Entiers, rationnels, flottants, complexes

- Maple peut gérer des entiers jusqu'à 500000 chiffres décimaux

```
> N := 100!;
N := 9332621544394415268169923885626670049071596826438
     16214685929638952175999932299156089414639761565182
     86253697920827223758251185210916864000000000000000
     000000000
```

- Maple simplifie systématiquement les rationnels

```
> x := 1/21 + 3/35;
                        x := 2/15
```

- La précision des nombres flottants est contrôlée par la variable `Digits`. Par défaut, `Digits` vaut 10

```
> 1.2^20;
                        38.33759992

> Digits:=20: 1.2^20;
                        38.337599924474751222
```

Entiers, rationnels, flottants, complexes (suite)

- On peut évaluer une expression en flottant par la commande `evalf`

```
> Digits:=30 : x:=exp(Pi*sqrt(163)); evalf(x);
                                        1/2
                    x := exp(Pi 163    )
                                                          18
                    0.262537412640768744000000000024 10
```

- Le nombre complexe $i = \sqrt{-1}$ est représenté par `I` en Maple

```
> z:=(1+2*I)^2;
                    z := -3 + 4 I
```

- Pour mettre un complexe sous forme cartésienne, on utilise `evalc`

```
> z^(1/2);
                                   1/2
                    (-3 + 4 I)
> evalc(%);
                    1 + 2 I
```

Entiers, rationnels, flottants, complexes (suite)

`iquo(m,n)`	quotient de la division euclidienne de m par n
`irem(m,n)`	reste de la division euclidienne de m par n
`ifactor(n)`	factorisation de l'entier n
`numer`, `denom`	numérateur et dénominateur d'une fraction
`trunc`, `round`, `frac`, `floor`, `ceil`	parties entières ou fractionnaire
`evalf`	évaluation numérique en flottant
`evalc`	évaluation d'un complexe

Some abstaction

All algorithms run over domains (sets of objects) that share arithmetic operations.

Abstracting these domains, i.e. identifying the properties satisfied/required by these domains allows to

- Introduce some genericity in programming;
- Extend computational paradigms/algorithms designed in "simple" domains to more sophisticated ones.

Some abstaction

All algorithms run over domains (sets of objects) that share arithmetic operations.

Abstracting these domains, i.e. identifying the properties satisfied/required by these domains allows to

- Introduce some genericity in programming;
- Extend computational paradigms/algorithms designed in "simple" domains to more sophisticated ones.

Our roadmap

- We will define abstract domains rings and fields and relate them to sets of numbers.
- We will review some well-known computational paradigms in this context.
- We will extend them to more sophisticated rings and fields (e.g. the polynomial world)

Rings

A ring is a set R equipped with a $+$ and a $\times$ where everything happens as expected.

Addition and substraction

- $a - a = 0$
- $a + b = b + a$
- $a + (b + c) = (a + b) + c$

Multiplication

- $a \times (b \times c) = (a \times b) \times c$

Multiplication and Addition

- $a \times (b + c) = a \times b + a \times c$

and

- there exists $\mathbf{0} \in R$ such that, for all $a \in R$, $a + \mathbf{0} = a$;
- for all $a \in R$, there exists b such that $a + b = \mathbf{0}$;
- there exists an element $\mathbf{1} \in R$such that, for all $a \in R$, $\mathbf{1} \times a = a$

Examples (and non-examples) of rings

Examples integers, rationals, reals, complex numbers

Counter-example The set of (machine) floats is not a ring !

```
void main(){
  float a, b, c;
  a = 3432.675;
  b = 0.03232;
  c = 24.535;
  printf("%f\n", ((a+b)+c) - (a+(b+c)));
}
```

-0.000244

Subsidiary question: is the set of even odd integers a ring?

A bit more sophisticated example

Bits with xor $(+)$ and and $(\times)$ form a ring

$+$	0	1
0	0	1
1	1	0

and

$\times$	0	1
0	0	0
1	0	1

Rule Do operations over the integers and reduce modulo 2

Notation $\{0,1\} = \mathbb{F}_2 = \frac{\mathbb{Z}}{2\mathbb{Z}}$

Back to integers

Representation (base 2) $n = a_0 + a_1 2 + a_2 2^2 + \cdots + a_P 2^P$ with $a_i \in \{0, 1\} = \mathbb{F}_2 = \frac{\mathbb{Z}}{2\mathbb{Z}}$ (with $a_P \neq 0$)

Size of an integer

- We expect it to be the size of the array $a_0, \ldots, a_P$
- In base 2: size$(n) = \simeq \lceil \log_2(n) \rceil$
- In base B: $\simeq \lceil \log_B(n) \rceil$ (size in base 2 multiplied by a constant)

Bit complexity counts the number of operations of single/bit operations $\rightarrow$ takes into account growth of coefficients

Complexity of basic operations

Assuming $\tau = \text{size}(n) \geq \text{size}(m)$,

Addition of n, m in $O(\text{size}(n))$

Multiplication of n, m:

$$\mathsf{M}_{\text{int}}(\tau) \in O(\tau^2)$$

(naive)

$$\mathsf{M}_{\text{int}}(\tau) \in O(\tau^{1.59})$$

(Karatsuba)

$$\mathsf{M}_{\text{int}}(\tau) \in O(\tau \log \tau \log \log \tau)$$

(Fast Fourier Transform)

Quotients and remainders

Let A and B be two integers. The quotient and remainder are defined through

$$A = BQ + R$$

with

$$0 \leq R \leq |B|.$$

Notation: $Q = A \text{ div } B$ and $R = A \text{ rem } B$

Complexity Assuming that $\text{size}(A) = \tau \geq \text{size}(B)$,

- $O(\tau^2)$ bit operations (naive)
- $O(\mathsf{M}_{\text{int}}(\tau))$ bit operations (Newton, not in this course)

GCD and Euclide's algorithm

Let A and B be two integers. The greatest common divisor (GCD) of A, B is the greatest non-negative integer G dividing both A and B.

Some immediate relations

- $\mathrm{GCD}(A, B) = \mathrm{GCD}(B, A)$
- if $A \geq B$, $\mathrm{GCD}(A, B) = \mathrm{GCD}(A - B, B)$
- $\mathrm{GCD}(CA, CB) = C\,\mathrm{GCD}(A, B)$

GCD and Euclide's algorithm

Input: A, B integers
Output: R and U, V such that $R = \text{GCD}(A, B)$ and $R = AU + BV$
Q, RS, US, VS are local variables which will store intermediate datas
Equations $R = AU + BV$ and $R' = AU' + BV'$ are loop invariants

```
#Initialize :
R := A, R' := B, U := 1, V := 0, U' := 0, V' := 1
while (R' <> 0) do
    Q := iquo(r, r');
    RS := R, US := U, VS := V;
    R := R', U := U', V := V';
    R' := RS - Q *R', U' = US - Q*U', V' = VS - Q*V';
od;
return R, U, V;
```

Complexity is in $O(\text{size}(A)^2)$ (assuming $A \geq B$) – Proof is skipped.

Bézout relation

Let A and B be two integers and $G = \mathrm{GCD}(A, B)$.
There exist integers U and V such that

$$AU + BV = G$$

Some consequences

- If $G = 1$ (one says that A and B are coprime), there exist U and V such that $AU + BV = 1$
- Then, U is the inverse of A modulo B; in other words

$$AU \,\mathrm{rem}\, B = 1$$

Fields

A field is a set F with a $+$ and a $\times$ where everything happens as expected when one wants to use division.

- $(F, +, \times)$ is a ring
- for all $a \in F$ such that $a \neq 0$ there exists b such that

$$a \times b = 1$$

We write $b = a^{-1}$ and call it the inverse of a.

Examples of fields

Intuitive examples rationals, reals, complex numbers

Counter examples integers

Def. Let N be a positive integer.

$\frac{\mathbb{Z}}{N\mathbb{Z}} = \{0, 1, \ldots, N-1\}$ equipped with $\oplus$ and $\otimes$ such that

- for a, b in $\frac{\mathbb{Z}}{N\mathbb{Z}}$,
$$a \oplus b = (a + b) \text{ rem } N$$
- for a, b in $\frac{\mathbb{Z}}{N\mathbb{Z}}$,
$$a \otimes b = (a \times b) \text{ rem } N$$

Theorem If N is prime, then $\frac{\mathbb{Z}}{N\mathbb{Z}}$ is a field.

Multi-modular arithmetic

Some observations:

- Bit complexity analysis of algorithms will take into account coefficient growth during the whole computational process
- For some algorithms, it may be possible that the size of the output is smaller than the size of intermediate coefficients.
- Idea: compute the output "modulo" several prime numbers (e.g. in a field) $\rightarrow$ control of the size of the intermediate coefficients.

Chinese Remainder Theorem

Let $P_1, \ldots, P_k$ be pairwise coprimes and $A_1, \ldots, A_k$ be a sequence of integers such that $A_i \in \frac{\mathbb{Z}}{P_i\mathbb{Z}}$.

There exists an integer X such that

$$\begin{aligned} X &= A_1 \mod P_1 \\ &\vdots \\ X &= A_k \mod P_k \end{aligned}$$

Moreover, all solutions are congruent modulo $N = P_1 \cdots P_k$.

Application: Choose $P_1, \ldots, P_k$ prime of size smaller than the word machine.

Chinese Remainder Reconstruction (2 equations)

Let us focus on the case of 2 equations:

$$X = A_1 \mod P_1 \quad \text{and} \quad X = A_2 \mod P_2$$

- Since P_1, P_2 are co-prime, $\mathrm{GCD}(P_1, P_2) = 1$.
- By the Bézout relation, this implies that there exist U, V such that

$$UP_1 + VP_2 = 1.$$

- Consequently, for any X, the following holds:

$$XUP_1 + XVP_2 = X$$

.

Now, remark that

$$X \mod P_1 = XVP_2 \mod P_1 = X \times 1 \mod P_1$$

Thus, taking

$$X = A_1VP_2 + A_2UP_1$$

provides a solution.

Chinese Remainder Reconstruction

Take $P = P_1 \times \cdots \times P_k$.
For $1 \leq i \leq k$,

- remark that $\frac{P}{P_i}$ and P_i are co-prime.
- take U_i as the inverse of $\frac{P}{P_1}$ modulo P_i

$$U_i \frac{P}{P_i} + V P_i = 1$$

Construct

$$X = A_1 \frac{P}{P_1} U_1 + \cdots + A_k \frac{P}{P_k} U_k$$

Examples

- Solve $X = 2 \mod 3$ and $X = 3 \mod 5$
- Solve $X = 1 \mod 2$ and $X = 2 \mod 3$ and $X = 3 \mod 5$