Modélisation et résolutions numérique et symbolique

via les logiciels Maple et Matlab

Jeremy Berthomieu Mohab Safey El Din Stef Graillat

Mohab.Safey@lip6.fr

Outline

- Previous course: partial review of what you should know;
- Linear Algebra: basic algorithms;
- Linear Algebra: Solving and CRT
- Linear Algebra: how to survive in Maple

Linear Algebra basic concepts (I)

- Vector spaces and basis
  Finite dimensional vector spaces
- Linear applications – representation through matrices
  Kernels, images and properties
- rank and determinants $\rightarrow$ basic measures
- Linear solving and Cramer's formulas

Linear Algebra basic concepts (II)

Solving $AX = b$

- One can use Cramer's formula

$$x_i = \frac{\det B_i}{\det A}$$

avec $B_i = \begin{pmatrix} a_{0,0} & \cdot & a_{0,i-1} & b_0 & a_{0,i+1} & \cdot & a_{0,n-1} \\ a_{1,0} & \cdot & a_{1,i-1} & b_1 & a_{1,i+1} & \cdot & a_{1,n-1} \\ \vdots & \vdots & & \vdots & & & \\ a_{n-1,0} & \cdot & a_{n-1,i-1} & b_{n-1} & a_{n-1,i+1} & \cdot & a_{n-1,n-1} \end{pmatrix}$

- Without any "smart" algorithm for computing determinants, one uses the formula based on cofactors $\rightsquigarrow O(n!)$
  Such a complexity is prohibitive

Linear Algebra basic concepts (III)

Let $A = (a_{i,j})_{1 \leq i,j \leq n}$ be a $n \times n$ matrix with entries in $\mathbb{Z}$.
Then

$$|\det(A)| \leq \prod_{j=1}^{n} \sqrt{\sum_{1 \leq i \leq n} a_{i,j}^2}$$

- Good upper bound on the size of $\det(A)$ (proved previously).

Linear Algebra basic concepts (IV)

Definitions

We say that λ is an eigenvalue of A if and only if there exists $v \neq \mathbf{0}$ such that $Av = \lambda v$; in this case v is an eigenvector. The eigenspace E_λ associated to λ is the set of vectors v such that $Av = \lambda v$.

- E_λ is a vector subspace.
- λ is an eigenvalue of A if and only if it is a root of $\det(A - L\mathbf{Id})$.
- The ground field plays a particular role: if A is with entries in $\mathbb{K}$, eigenvalues may not lie in $\mathbb{K}$ but in a "larger" field (an algebraic closure).

Eigenvalues play a crucial role in many areas of computer science: data-mining (PageRank) – decision theory, imagery, scient. computing, etc.
BUT first we need good algorithms for solving linear systems (with polynomial bit complexity) $\rightarrow$ Determinants

Solving linear systems: the easy case

Assume now that our system has the following shape

$$\begin{pmatrix} a_{0,0} & a_{0,1} & \dots & \dots & a_{0,n-1} \\ 0 & a_{1,1} & \dots & \dots & a_{1,n-1} \\ \vdots & \vdots & \dots & \dots & \vdots \\ 0 & 0 & \dots & a_{n-2,n-2} & a_{n-2,n-1} \\ 0 & 0 & \dots & 0 & a_{n-1,n-1} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-2} \\ x_{n-1} \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_{n-2} \\ b_{n-1} \end{pmatrix}$$

Note that if $a_{n-1,n-1} \neq 0$ then $x_{n-1} = \frac{b_{n-1}}{a_{n-1,n-1}}$

An example

Solve the system:

$$\begin{cases} 2x - 3y + z - 5t &= -1 \\ 2x - y - 3z &= 1 \\ x + 4y &= 2 \\ 2x &= 3 \end{cases}$$

Solving triangular linear systems

RecSolveTriangle(A, B, n)

Si $n = 1$ alors $X = B/A$

Sinon

$x \leftarrow \frac{b_{n-1}}{a_{n-1,n-1}}$

$A' \leftarrow$ RedMat$(A, n-1)$

$B' \leftarrow B - x.C_{n-1}$

$X \leftarrow$

(RecSolveTriangle$(A', B', n-1), x)$

ResTriangleIt(A, B, n)

Pour i = n-1 Ãă 0 faire

$b_i \leftarrow \frac{b_i}{a_{i,i}}$

Si $(i > 0)$ alors

Pour$j = i - 1$Ãă 0 faire

$b_j \leftarrow b_j - b_i * a_{j,i}$

$a_{j,i} \leftarrow 0$

$a_{i,i} \leftarrow 1$

Complexity analysis and consequences

- The number of multiplications for n-variate system is denoted by $T(n)$
- Recurrence formula : $T(n) = T(n-1) + n$ et $T(1) = 1$
- Total number of products $\frac{n^2+n}{2}$

Core idea: reduce linear solving to triangular solving in polynomial time $\rightsquigarrow$ bit complexity

Elementary transformations

Elementary transformations on lines (by transposition, we get those on columns)

- multiply a line by a non-zero element of $\mathbb{K}$;
- swap two lines
- add to a line the product of a line with an element of $\mathbb{K}$.

Definition

The matrix obtained by performing the elementary transformation on the identity matrix is the elementary matrix to the considered transformation.

Property

Every elementary transformation on lines (resp. columns) is obtained by left- (resp. right-) multiplication of A with the corresponding elementary matrix.

Gauss algorithm and LU decomposition (I)

- Reduction to triangular solving through Gaussian elimination using exclusively elementary transformations of third type
  $\rightsquigarrow$ in the case of square matrices, the determinant is unchanged.
- Once an equivalent triangular system is obtained, apply procedures solving triangular linear systems.
- Gaussian elimination simulates on matrices the elimination of variables one after another in the equations (via symbolic manipulation such as multiplication by a scalar and addition of equations).

Gauss algorithm and LU decomposition (II)

PremGauss($A, , B, n$)

For i **from** 0 **to** $n-1$ **do**

Si $A_{i,i} = 0$ **et** $i < n$ **alors** $j = i+1$,

Tant que $A_{j,i} = 0$ **et** $j < n-1$ **faire** $j = j+1$

Si $j = n-1$ **et** $A_{j,i} = 0$ **alors** Pas de solution

Sinon $T_{i..n-1} := A_{i,i..n-1}; A_{i,i..n-1} := A_{j,i..n-1}; A_{j,i..n-1} := T_{i..n-1}$

$X := B_i; B_i := B_j; B_j := X$

for j **from** $i+1$ **to** $n-1$ **do**

$B_j := B_j - (A_{j,i}/A_{i,i}) * Bi;$

$A_{j,i..n-1} := A_{j,i..n-1} - (A_{j,i}/A_{i,i}) * A_{i,i..n-1};$

return A, B

Complexity: $\simeq \frac{n(n-1)(2n-1)}{6} + \frac{n(n-1)}{2}$

Gauss algorithm and LU decomposition (III)

$$\left[\begin{array}{cccc} 10 & 7 & 8 & 7 \\ 7 & 5 & 6 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 10 \end{array}\right], \left[\begin{array}{c} 32 \\ 23 \\ 33 \\ 31 \end{array}\right]$$

$$\left[\begin{array}{cccc} 10 & 7 & 8 & 7 \\ 0 & 1/10 & 2/5 & 1/10 \\ 0 & 2/5 & \frac{18}{5} & \frac{17}{5} \\ 0 & 1/10 & \frac{17}{5} & \frac{51}{10} \end{array}\right], \left[\begin{array}{c} 32 \\ 3/5 \\ \frac{37}{5} \\ \frac{43}{5} \end{array}\right] \rightarrow \left[\begin{array}{cccc} 10 & 7 & 8 & 7 \\ 0 & 1/10 & 2/5 & 1/10 \\ 0 & 0 & 2 & 3 \\ 0 & 0 & 3 & 5 \end{array}\right], \left[\begin{array}{c} 32 \\ 3/5 \\ 5 \\ 8 \end{array}\right]$$

$$\rightarrow \left[\begin{array}{cccc} 10 & 7 & 8 & 7 \\ 0 & 1/10 & 2/5 & 1/10 \\ 0 & 0 & 2 & 3 \\ 0 & 0 & 0 & 1/2 \end{array}\right], \left[\begin{array}{c} 32 \\ 3/5 \\ 5 \\ 1/2 \end{array}\right] \rightarrow \left[\begin{array}{cccc} 10 & 7 & 8 & 7 \\ 0 & 1/10 & 2/5 & 1/10 \\ 0 & 0 & 2 & 3 \\ 0 & 0 & 0 & 1/2 \end{array}\right], \left[\begin{array}{c} 32 \\ 3/5 \\ 5 \\ 1/2 \end{array}\right]$$

$$\left[\begin{array}{cccc} 10 & 7 & 8 & 0 \\ 0 & 1/10 & 2/5 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{array}\right], \left[\begin{array}{c} 25 \\ 1/2 \\ 2 \\ 1 \end{array}\right] \rightarrow \left[\begin{array}{cccc} 10 & 7 & 0 & 0 \\ 0 & 1/10 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array}\right], \left[\begin{array}{c} 17 \\ 1/10 \\ 1 \\ 1 \end{array}\right]$$

$$\rightarrow \left[\begin{array}{cccc} 10 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array}\right], \left[\begin{array}{c} 10 \\ 1 \\ 1 \\ 1 \end{array}\right] \rightarrow \left[\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array}\right], \left[\begin{array}{c} 1 \\ 1 \\ 1 \\ 1 \end{array}\right]$$

$$\rightarrow \left[\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array}\right], \left[\begin{array}{c} 1 \\ 1 \\ 1 \\ 1 \end{array}\right]$$

Examples

Solve the systems

$$\begin{cases} x+y+z+t &= 5 \\ 2x-y-z+t &= 4 \\ 3x+y+z-t &= 1 \\ x+y-t+z &= -1 \end{cases}$$

and

$$\begin{cases} 2x-y-t-z &= -1 \\ 3x-2y-z+t &= 1 \\ x-2y-3-2t &= 2 \\ x-y-z-t &= 3 \end{cases}$$

Question: What happens when you don't find a pivot $\neq 0$?

Solving several linear systems

Assume we want to solve

$$\begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} \\ a_{1,0} & a_{1,1} & a_{1,2} \\ a_{2,0} & a_{2,1} & a_{2,2} \end{pmatrix} \times \begin{pmatrix} x_{0,0} & x_{0,1} & x_{0,2} \\ x_{1,0} & x_{1,1} & x_{1,2} \\ x_{2,0} & x_{2,1} & x_{2,2} \end{pmatrix} = \begin{pmatrix} b_{0,0} & b_{0,1} & b_{0,2} \\ b_{1,0} & b_{1,1} & b_{1,2} \\ b_{2,0} & b_{2,1} & b_{2,2} \end{pmatrix}$$

Using Gaussian elimination, this leads us to solve:

$$\begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} \\ a_{1,0} & a_{1,1} & a_{1,2} \\ a_{2,0} & a_{2,1} & a_{2,2} \end{pmatrix} \times \begin{pmatrix} x_{0,i} \\ x_{1,i} \\ x_{2,i} \end{pmatrix} = \begin{pmatrix} b_{0,i} \\ b_{1,i} \\ b_{2,i} \end{pmatrix} \text{ pour } i = 0, 1, 2$$

Strategy and application

Given a square matrix A of size n:

- Encode elementary transformations with elementary matrices during Gauss algorithm.
- This will result in 3 matrices P, L and U where
  - P is a permutation matrix;
  - L is a low triangular matrix (all elements of the diagonal being 1);
  - U is an upper triangular matrix;

  such that

$$A = PLU$$

- Then solving $AX = B_1, \ldots, AX = B_k$ can be done by computing once for all the PLU-decomposition ($O(n^3)$) and next linear triangular solving ($O(n^2)$).

Bit complexity of Gauss algorithm – First attempt

Let A be a matrix with entries in $\mathbb{Z}$. Gauss algorithm manipulates rationals.
$\rightarrow$ **Size of a rational number:** $\text{size}(a/b) = \text{size}(a) + \text{size}(b)$
Denote by τ the maximum size of the entries of A.
Now, let n be the size of $A \rightarrow n$ eliminations through n pivotings.

Elim. 1: size of new entries $3 \times \tau$ (due to pivoting)

Elim. 2: size of new entries $3 \times 3 \times \tau$ (pivoting on entries after 1 Elim)

Elim. 3: size of new entries $3^3\tau$ (pivoting on entries after 2 Elim)

Elim. 4: size of new entries $3^4\tau$ (pivoting on entries after 3 Elim)

$\ldots$

- Bit complexity $\rightsquigarrow O(\tau 3^n n^3)$?

Bit complexity of Gauss algorithm – First attempt

Let A be a matrix with entries in $\mathbb{Z}$. Gauss algorithm manipulates rationals.
$\rightarrow$ **Size of a rational number:** $\text{size}(a/b) = \text{size}(a) + \text{size}(b)$
Denote by τ the maximum size of the entries of A.
Now, let n be the size of $A \rightarrow n$ eliminations through n pivotings.

Elim. 1: size of new entries $3 \times \tau$ (due to pivoting)

Elim. 2: size of new entries $3 \times 3 \times \tau$ (pivoting on entries after 1 Elim)

Elim. 3: size of new entries $3^3\tau$ (pivoting on entries after 2 Elim)

Elim. 4: size of new entries $3^4\tau$ (pivoting on entries after 3 Elim)

$\ldots$

- Bit complexity $\rightsquigarrow O(\tau 3^n n^3)$?

No, this complexity analysis is not sharp enough and a bit naive
Bit complexity of Gauss is polynomial in n and τ.
We can prove it by designing an algorithm with such a complexity.

CRT and Solving (in general)

Consider an algorithm ALGO that outputs an integer.

Requirement:

- A bound on the size of the output (with respect to a bound on the size of the inputs)
- ALGO perform ring or field arithmetic operations.

Strategy:

- Given inputs, compute the bound B on the output.
- Choose "small" prime numbers (e.g. of size less than the word machine) $p_1, \ldots, p_\ell$ such that

$$B \leq p_1 \cdots p_\ell$$

- Run the algorithm ALGO in $\frac{\mathbb{Z}}{p_1\mathbb{Z}}, \ldots, \frac{\mathbb{Z}}{p_\ell\mathbb{Z}}$ after taking modular images of the input.
- Run CRT algorithm to reconstruct the correct output knowing its image modulo $p_1, \ldots, p_\ell$.

CRT and Solving (in general)

Intrinsic difficulty:

- Divisions by 0 may occur when running ALGO modulo a prime number.

Complexity:

- Let Θ be the function taking as input sizes of inputs of ALGO and returning the arithmetic complexity.
- Let $p_1, \ldots, p_\ell$ be the selected prime numbers and τ be a bound on their size.
- Complexity is $O(\ell\tau\Theta + (\ell\tau)^2)$.
  $O(\ell\tau\Theta)$ is running time of ALGO modulo ℓ prime numbers
  $O((\ell\tau)^2)$ is the cost of reconstruction with CRT

To go further/discussion:

- log-linear complexity for the output ?

  This is advanced algorithm – not in this course

CRT and Linear Solving (I)

It is not immediate to apply CRT for linear solving:

- The output is **not** a vector of integers but a vector of rationals
- These rationals may be **negative**

BUT:

- the coordinates of the solution are fractions of determinants. Cramer's formula
- Each determinant can be obtained computed separately

  parallel computation?
- There is only one remaining problem: the sign of the determinant.

CRT and Linear Solving (II)

Now our problem is to construct integers (maybe nonnegative).
Let B be the size of the output.

- Take $p_1, \ldots, p_\ell$ such that $2B \leq p_1 \cdots p_\ell$
- Use CRT; let $\mathbf{x}$ be the output.
- If $\mathbf{x} \leq B$ then return $\mathbf{x}$
- else return $\mathbf{x} - p_1 \cdots p_\ell$.

Implementations:

- Can take advantage of multi-threading/parallelism/vectorization
- Take care about redundant computations in the computations of determinants.

Very fast implementations require good programming skills

Another application: solving linear systems with parameters?

Consider a system of equations

$$\begin{array}{rcl} a_{1,1}(\mathbf{Y})X_1 + \cdots + a_{1,n}(\mathbf{Y}) & = & b_1 \\ \vdots & \vdots & \vdots \\ a_{n,1}(\mathbf{Y})X_1 + \cdots + a_{n,n}(\mathbf{Y}) & = & b_n \end{array}$$

where $a_{i,j}$ are univariate polynomials in $\mathbb{K}[\mathbf{Y}]$

- Propose formulas to solve it;
- Propose a technique/idea that yields an (efficient?) algorithm solving it.

In this case, we are actually working over the field of rational fractions $\mathbb{K}(\mathbf{Y})$

$\rightsquigarrow$ Use Cramer's formulae to obtain solutions (rational coefficients)

$\rightsquigarrow$ Use modular techniques to interpolate polynomials

Interpolation and Linear Algebra

Theorem

Let $a_0, \ldots, a_d$ and $v_0, \ldots, v_d$ be sequence of elements in $\mathbb{K}$ such that $a_i \neq a_j$ for $i \neq j$. There exists a unique polynomial f of degree d such that $f(a_i) = v_i$.

- **Mathematical proof is easy.**
- **Algorithm.**
  - Consider coefficients of f as unknowns;
  - Write equations generated by $f(a_0) = v_0, \ldots, f(a_d) = v_d \rightsquigarrow$ these are linear equations
  - Use Gaussian elimination to solve these linear equations.

Complexity: $O(d^3)$.

Lagrange formulas

Another way of writing equations $f(a_0) = v_0, \ldots, f(a_d) = v_d$:

Lagrange formulas

Another way of writing equations $f(a_0) = v_0, \ldots, f(a_d) = v_d$:

$$f = v_0 \mod (X - a_0), \ldots, f = v_0 \mod (X - a_d)$$

- where $\mod$ denotes the remainder (generalized to univariate polynomials by considering degrees instead of absolute values).
- besides it yields a generalization of Euclid's algorithm to univariate polynomials
- remark that the GCD of $(X - a_i)$ and $(X - a_j)$ is 1 iff $a_i = a_j$.

$\rightarrow$ Generalization of CRT reconstruction to interpolation.

1. $P \leftarrow \prod_{i=0}^{d}(X - a_i)$
2. $f = \sum_{i=0}^{d} \frac{P}{(X_i - a_i)} \times \frac{1}{\prod_j (a_j - a_i)} \times v_i$

Complexity: $O(d^2)$

Examples

- $f(1) = 1, f(2) = -1, f(3) = 0;$
- $f(0) = -1, f(1) = -2, f(-1) = 1.$

Linear Algebra in Maple

- Up to release 10, the main package was the `linalg` package
  New package `LinearAlgebra` has appeared recently but we will use `linalg`.
- All basic linear algebra algorithms are implemented.
  Feature: focus on symbolic algorithms providing exact results.
- Efficiency issues are critical. New releases of Maple include functions based on BLAS, FLAS, etc.

Linear Algebra in Maple

- Definition of matrix matrice

```
> A := matrix(2,2,[1,2,3,4]); B := matrix(2,2,[4,2,1,4]):
```

$$A := \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

- Evaluation of matrix

```
> evalm(A+B);
```

$$\begin{bmatrix} 5 & 4 \\ 4 & 8 \end{bmatrix}$$

- Element of a matrix

```
> A[2,1];
```

$$3$$

Linear Algebra in Maple

Les premières commandes font partie du package `linalg`. Les secondes demandent l'utilisation de la fonction `evalm`

`matrix`	définition d'une matrice
`add(A,B)`	somme des matrices A et B
`add(A,B,lambda,mu)`	combinaison linéaire $A + B$
`multiply(A,B)`	produit des matrices A et B
`rank(A)`	rang de la matrice A
`det(A)`	déterminant de A
`array(1..n,1..n,identity)`	matrice identité d'ordre n
`inverse(A)`	inverse de A
`transpose(A)`	matrice transposée de A
`nullspace(A)`	noyau de A
`charpoly(A,x)`	polynôme caractéristique de A
`eigenvals(A)`	valeurs propres de A
`eigenvects(A)`	vecteurs propres de A
`A+B`	somme des matrices A et B
`alpha*A`	produit de la matrice A par le scalaire
`A &* B`	produit des matrices A et B
`A^k`	A^k
`&*()`	matrice identité
`A^(-1)`	A^{-1}

Bibliothèques

En plus des fonctions par défaut, Maple dispose de fonctions supplémentaires qui appartiennent à des bibliothèques, appelées packages. Pour pouvoir utiliser ces fonctions, il faut charger en mémoire ces bibliothèques. Ce chargement se fait à l'aide de la commande `with` :

```
> with(package) ;
```

Par exemple, le chargement du package `linalg` se fera par :

```
> with(linalg) ;
[BlockDiagonal; GramSchmidt; JordanBlock; LUdecomp; ...]
```

Cette méthode a un léger inconvénient : toutes les fonctions chargées sont affichées à l'écran, d'où l'intérêt d'utiliser les :