

Modélisation et résolutions numérique et symbolique de problèmes via les logiciels Maple et MATLAB (MODEL)

Cours n°4 : Transformée de Fourier discrète

Stef Graillat

Université Pierre et Marie Curie (Paris 6)



Résumé du cours précédent

Calcul matriciel

- Manipulations des matrices :
 - 1 Stockage des matrices : ligne ou colonne
 - 2 Les BLAS
- Les décompositions utiles et leurs applications :
 - 1 LU
 - 2 QR
 - 3 Réduction (diagonalisation, etc.)
 - 4 SVD

Présenter l'algorithme de « Transformée de Fourier Rapide » (Fast Fourier Transform ou FFT)

Un des algorithmes les plus utilisés dans le monde avec des applications en

- traitement du signal
- traitement d'image
- calcul formel (multiplication de polynômes, de grands entiers, etc)

Plan du cours

- 1 Multiplication de polynômes et choix de représentation
- 2 Évaluation et interpolation
- 3 Racine n -ième de l'unité
- 4 Version matricielle de la FFT

- Algorithms, S. Dasgupta, C.H. Papadimitriou et U.V. Vazirani, McGraw Hill, 2006
- Introduction à l'algorithmique, Thomas Cormen, Charles Leiserson, Ronald Rivest et Clifford Stein, 2nd édition, Dunod, 2002
- The Art of Computer Programming, Volume 2 : Seminumerical Algorithms, Donald E. Knuth, 3e édition, Addison-Wesley, 1997
- Modern Computer Algebra, Joachim von zur Gathen et Jürgen Gerhard, 2nd edition, Cambridge University Press, 2003
- Numerical Recipes. The Art of Scientific Computing, William Press, Saul Teukolsky, William Vetterling et Brian Flannery, 3e édition, Cambridge University Press, 2007
- Computational Frameworks for the Fast Fourier Transform, Charles Van Loan, SIAM, 1987

Multiplication de polynômes

Le produit de 2 polynômes de degré n est un polynôme de degré au plus $2n$

$$(1 + 2x + 3x^2) \cdot (2 + x + 4x^2) = 2 + 5x + 12x^2 + 11x^3 + 12x^4$$

Plus généralement, si

$$P(x) = a_0 + a_1x + \cdots + a_nx^n \quad \text{et} \quad Q(x) = b_0 + b_1x + \cdots + b_nx^n$$

alors $R(x) = P(x)Q(x) = c_0 + c_1x + \cdots + c_{2n}x^{2n}$ avec

$$c_k = a_0b_k + a_1b_{k-1} + \cdots + a_kb_0 = \sum_{i=0}^k a_i b_{k-i}$$

Multiplication de polynômes (suite)

Algorithme 1 (Multiplication naïve)

```
function R = mult(P,Q)
  n = length(P);
  R = zeros(1,2*n-1);
  for i=1:n
    for j=1:n
      R(i+j-1) = R(i+j-1) + P(i)*Q(j);
    end
  end
end
```

Coût : $\mathcal{O}(n^2)$

Peut-on faire mieux que $\mathcal{O}(n^2)$?

→ algorithme de Karatsuba : $\mathcal{O}(n^{\log_2 3}) \approx \mathcal{O}(n^{1.58})$

Peut-on encore faire mieux ?

Changement de représentation (suite)

On représente généralement le polynôme

$$P(x) = a_0 + a_1x + \dots + a_nx^n$$

par le vecteur de ses coefficients $a = (a_0, a_1, \dots, a_{n-1})$

Question : y-a-t'il d'autres représentations des polynômes ?

Définition 1

Un nombre complexe z est une racine de $P(x)$ si $P(z) = 0$

Théorème 1 (Théorème de d'Alembert-Gauss)

Tout polynôme $P(x)$ de degré n à coefficients dans \mathbb{C} admet n racines z_1, \dots, z_n (comptées avec leurs multiplicités). On a alors

$$P(x) = a_n(x - z_1) \cdots (x - z_n)$$

Changement de représentation (suite)

On peut donc représenter le polynôme $P(x)$ par a_n et ses racines z_1, \dots, z_n .

Évaluation : P étant donné par a_n et z_1, \dots, z_n , on peut évaluer P en x en $\mathcal{O}(n)$

Multiplication : P étant donné par a_n et z_1, \dots, z_n et Q étant donné par b_n et z'_1, \dots, z'_n , PQ est donné par $a_n b_n, z_1, \dots, z_n, z'_1, \dots, z'_n$

Addition : difficile !

Changement de représentation (suite)

Théorème 2

Un polynôme de degré n est uniquement déterminé par ses valeurs sur $n + 1$ points distincts.

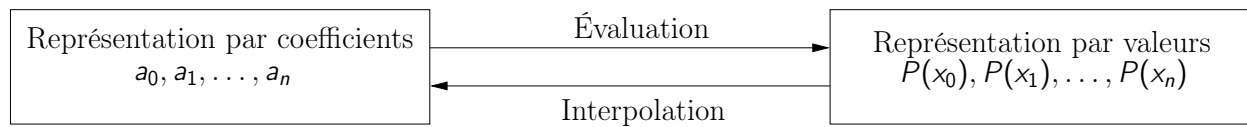
Fixons $n + 1$ points distincts x_0, \dots, x_n . On peut spécifier un polynôme $P(x) = a_0 + a_1x + \dots + a_nx^n$ de degré n par l'une des façons suivantes :

- 1 ses coefficients a_0, a_1, \dots, a_n
- 2 les valeurs $P(x_0), P(x_1), \dots, P(x_n)$

Il est facile de multiplier deux polynômes dont on connaît les valeurs. Le produit $R(z)$ en un point z est le produit de $P(z)$ par $Q(z)$!

Facile aussi pour l'addition !

L'algorithme de multiplication



Algorithme 2 (Multiplication de polynômes)

Entrée : 2 polynômes P et Q de degré n donnés via leurs coefficients

Sortie : le polynôme $R = PQ$ donné via ses coefficients

Sélection

Choisir des points x_0, x_1, \dots, x_{m-1} avec $m \geq 2n + 1$

Évaluation

Calculer $P(x_0), P(x_1), \dots, P(x_{m-1})$ et $Q(x_0), Q(x_1), \dots, Q(x_{m-1})$

Multiplication

Calculer $R(x_k) = P(x_k)Q(x_k)$ pour tout $k = 0, \dots, m - 1$

Interpolation

Retrouver $R(x) = c_0 + c_1x + \dots + c_{2n}x^{2n}$

Coût de l'algorithme

- Coût de la sélection : $\mathcal{O}(n)$
- Coût de la multiplication : $\mathcal{O}(n)$
- Qu'en est-il du coût de l'évaluation et de l'interpolation ?

$$P(x) = a_0 + a_1x + \dots + a_nx^n = (((((a_nx + a_{n-1})x + a_{n-2})x + a_{n-3})x + \dots)x + a_0$$

Algorithme 3 (Algorithme de Horner)

```
function res = Horner(P, x)
```

```
     $s_n = a_n$ 
```

```
    for i = n - 1 : -1 : 0
```

```
         $s_i = s_{i+1} * x + a_i$ 
```

```
    end
```

```
    res =  $s_0$ 
```

Coût : $\mathcal{O}(n)$

Si n évaluations à faire $\rightarrow \mathcal{O}(n^2)$

Évaluation en mode « diviser pour régner »

On suppose à partir de maintenant que n est une puissance de 2 ($n = 2^k$ avec $k \in \mathbb{N}$)

Idée : évaluer un polynôme $P(x)$ de degré $< n$ en n points par paire

$$\pm x_0, \pm x_1, \dots, \pm x_{n/2-1}$$

car on peut recouper les calculs de $P(x_i)$ et $P(-x_i)$.

En effet, on peut séparer les puissances paires et impaires,

$$P(x) = P_p(x^2) + xP_i(x^2)$$

Exemple :

$$3 + 4x + 6x^2 + x^4 + 10x^5 = (3 + 6x^2 + x^4) + x(4 + 2x^2 + 10x^4)$$

Évaluation en mode « diviser pour régner » (suite)

On remarque alors que :

$$\begin{aligned} P(x_i) &= P_p(x_i^2) + x_i P_i(x_i^2) \\ P(-x_i) &= P_p(x_i^2) - x_i P_i(x_i^2) \end{aligned}$$

Par conséquent, pour évaluer $P(x)$ en n points $\pm x_0, \pm x_1, \dots, \pm x_{n/2-1}$, il suffit d'évaluer $P_p(x)$ et $P_i(x)$ en $n/2$ points $x_0^2, \dots, x_{n/2-1}^2$

Si on continue le procédé de manière **récursive**, le nombre $T(n)$ d'opérations arithmétiques vérifie

$$T(n) = 2T(n/2) + \mathcal{O}(n)$$

On montre alors que

$$T(n) = \mathcal{O}(n \log n)$$

Évaluation en mode « diviser pour régner » : vers la récursion

- on part de n points $\pm x_0, \pm x_1, \dots, \pm x_{n/2-1}$
- après une étape, on a $n/2$ points $x_0^2, \dots, x_{n/2-1}^2$
- pour continuer la récursion, on a besoin de

$$\{x_0^2, \dots, x_{n/2-1}^2\} = \{\pm z_0, \pm z_1, \dots, \pm z_{n/4-1}\}$$

- si $z_0 = x_0^2$ et $-z_0 = x_j^2$ alors $x_0 = \pm i x_j$ avec $i^2 = -1$
- pour continuer la récursion, on a besoin des nombres complexes !

Racine n -ième de l'unité

Il s'agit des solutions dans \mathbb{C} de l'équation $z^n = 1$!

Théorème 3

Soit $\omega = e^{2i\pi/n}$. Les solutions de l'équation $z^n = 1$ sont $\omega^j = e^{2ij\pi/n}$ pour $j = 0, 1, \dots, n-1$.

Propriété 1

Si n est pair alors

- les racines n -ième de l'unité sont par paire : $\omega^{n/2+j} = -\omega^j$
- les élever au carré produit les racines $n/2$ -ième de l'unité

Notons $\omega_n = e^{2i\pi/n}$

- **Problème** : évaluer P en les racines n -ième de l'unité
- On rappelle que

$$P(x) = P_p(x^2) + xP_i(x^2)$$

- On observe que $(\omega_n^j)^2 = \omega_{n/2}^j$
- Par conséquent

$$P(\omega_n^j) = P_p((\omega_n^j)^2) + \omega_n^j P_i((\omega_n^j)^2)$$

- Ainsi évaluer P en les racines n -ième de l'unité peut s'effectuer en évaluant P_p et P_i en les racines $n/2$ -ième de l'unité

Évaluation en les racines n -ième (suite)

Algorithme 4 (Calcul de $FFT(P, \omega)$)

Entrée : le polynôme P connu par ses coefficients de degré n avec n une puissance de 2 et ω une racine primitive n -ième de l'unité

Sortie : les valeurs de $P(\omega^0), P(\omega^1), \dots, P(\omega^{n-1})$

- si $\omega = 1$ alors retourner $P(1)$
- exprimer $P(x)$ sous la forme $P(x) = P_p(x^2) + xP_i(x^2)$
- appeler $FFT(P_p, \omega^2)$ pour évaluer P_p en les puissances paires de ω
- appeler $FFT(P_i, \omega^2)$ pour évaluer P_i en les puissances paires de ω
- pour $j = 0 : n - 1$
 - calculer $P(\omega^j) = P_p(\omega^{2j}) + \omega^j P_i(\omega^{2j})$
- retourner $P(\omega^0), P(\omega^1), \dots, P(\omega^{n-1})$

Formulation matricielle

L'évaluation de $P(x) = a_0 + a_1x + \dots + a_{n-1}x^{n-1}$ en n points x_0, x_1, \dots, x_{n-1} peut s'écrire matriciellement

$$\begin{pmatrix} P(x_0) \\ P(x_1) \\ \vdots \\ P(x_{n-1}) \end{pmatrix} = \underbrace{\begin{pmatrix} 1 & x_0 & x_0^2 & \dots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \dots & x_1^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \dots & x_{n-1}^{n-1} \end{pmatrix}}_M \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{pmatrix}$$

La matrice M est une matrice de **Vandermonde**.

Propriété 2

Si x_0, x_1, \dots, x_{n-1} sont des nombres distincts alors M est inversible.

L'évaluation est la multiplication par M et l'interpolation est la multiplication par M^{-1}

Formulation matricielle : évaluation en les racines n -ième

Évaluer en les racines n -ième revient à multiplier par

$$M_n(\omega) = \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega & \omega^2 & \dots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \dots & \omega^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^j & \omega^{2j} & \dots & \omega^{(n-1)j} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \dots & \omega^{(n-1)(n-1)} \end{pmatrix}$$

Théorème 4 (Formule d'inversion)

$$M_n(\omega)^{-1} = \frac{1}{n} M_n(\omega^{-1})$$

Algorithme « diviser pour régner » matriciel

L'algorithme « diviser pour régner » se voit de manière matricielle

$$\begin{array}{c}
 \begin{array}{|c|} \hline k \\ \hline \end{array} \\
 \begin{array}{|c|} \hline \omega^{jk} \\ \hline \end{array} \\
 \begin{array}{|c|} \hline M_n(\omega) \\ \hline \end{array} \\
 \begin{array}{|c|} \hline a \\ \hline \end{array} \\
 \begin{array}{|c|} \hline a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ \vdots \\ a_{n-1} \\ \hline \end{array} \\
 = \\
 \begin{array}{|c|} \hline \text{Column} \\ \hline \end{array} \\
 \begin{array}{|c|} \hline 2k \\ \hline \end{array} \\
 \begin{array}{|c|} \hline 2k+1 \\ \hline \end{array} \\
 \begin{array}{|c|} \hline \omega^{2jk} \\ \hline \end{array} \\
 \begin{array}{|c|} \hline \omega^j \cdot \omega^{2jk} \\ \hline \end{array} \\
 \begin{array}{|c|} \hline \text{Even} \\ \text{columns} \\ \hline \end{array} \\
 \begin{array}{|c|} \hline \text{Odd} \\ \text{columns} \\ \hline \end{array} \\
 \begin{array}{|c|} \hline a \\ \hline \end{array} \\
 \begin{array}{|c|} \hline a_0 \\ a_2 \\ \vdots \\ a_{n-2} \\ a_1 \\ a_3 \\ \vdots \\ a_{n-1} \\ \hline \end{array} \\
 = \\
 \begin{array}{|c|} \hline \text{Row } j \\ \hline \end{array} \\
 \begin{array}{|c|} \hline \omega^{2jk} \\ \hline \end{array} \\
 \begin{array}{|c|} \hline \omega^j \cdot \omega^{2jk} \\ \hline \end{array} \\
 \begin{array}{|c|} \hline \text{Column} \\ \hline \end{array} \\
 \begin{array}{|c|} \hline 2k \\ \hline \end{array} \\
 \begin{array}{|c|} \hline 2k+1 \\ \hline \end{array} \\
 \begin{array}{|c|} \hline \omega^{2jk} \\ \hline \end{array} \\
 \begin{array}{|c|} \hline -\omega^j \cdot \omega^{2jk} \\ \hline \end{array} \\
 \begin{array}{|c|} \hline j+n/2 \\ \hline \end{array} \\
 \begin{array}{|c|} \hline a \\ \hline \end{array} \\
 \begin{array}{|c|} \hline a_0 \\ a_2 \\ \vdots \\ a_{n-2} \\ a_1 \\ a_3 \\ \vdots \\ a_{n-1} \\ \hline \end{array}
 \end{array}$$

Que l'on peut aussi voir comme

$$\begin{array}{|c|} \hline \text{Row } j \\ \hline \end{array} \\
 \begin{array}{|c|} \hline M_{n/2} \\ \hline \end{array} \\
 \begin{array}{|c|} \hline a_0 \\ a_2 \\ \vdots \\ a_{n-2} \\ \hline \end{array} \\
 + \omega^j \\
 \begin{array}{|c|} \hline M_{n/2} \\ \hline \end{array} \\
 \begin{array}{|c|} \hline a_1 \\ a_3 \\ \vdots \\ a_{n-1} \\ \hline \end{array} \\
 \\
 \begin{array}{|c|} \hline j+n/2 \\ \hline \end{array} \\
 \begin{array}{|c|} \hline M_{n/2} \\ \hline \end{array} \\
 \begin{array}{|c|} \hline a_0 \\ a_2 \\ \vdots \\ a_{n-2} \\ \hline \end{array} \\
 - \omega^j \\
 \begin{array}{|c|} \hline M_{n/2} \\ \hline \end{array} \\
 \begin{array}{|c|} \hline a_1 \\ a_3 \\ \vdots \\ a_{n-1} \\ \hline \end{array}
 \end{array}$$

Algorithme de FFT

Algorithme 5 (Calcul de $FFT(a, \omega)$)

Entrée : un tableau $a = (a_0, a_1, \dots, a_{n-1})$ avec n une puissance de 2 et ω une racine primitive n -ième de l'unité

Sortie : $M_n(\omega)a$

si $\omega = 1$ alors retourner a

$$(s_0, s_1, \dots, s_{n/2-1}) = FFT((a_0, a_2, \dots, a_{n-2}), \omega^2)$$

$$(s'_0, s'_1, \dots, s'_{n/2-1}) = FFT((a_1, a_3, \dots, a_{n-1}), \omega^2)$$

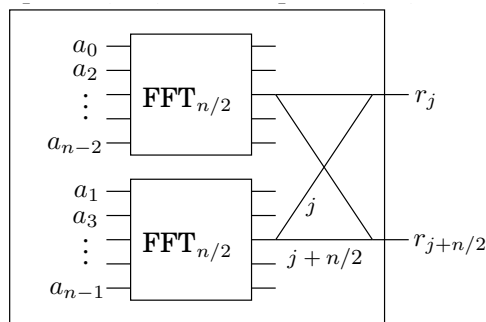
pour $j = 0 : n/2 - 1$

$$r_j = s_j + \omega^j s'_j$$

$$r_{j+n/2} = s_j - \omega^j s'_j$$

retourner $(r_0, r_1, \dots, r_{n-1})$

L'étape de récursion de l'algorithme de FFT peut se représenter par le circuit



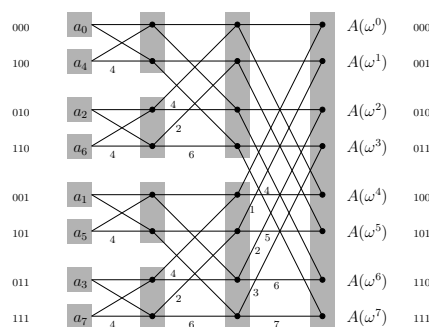
signifiant que

$$r_j = s_j + \omega^j s'_j$$

$$r_{j+n/2} = s_j - \omega^j s'_j$$

FFT version itérative (suite)

Si on déroule la récursion pour le circuit avec $n = 8$, on obtient



- il a $\log_2(n)$ niveaux chacun avec n noeuds pour un total de $n \log n$ opérations
- les entrées sont rangées dans un ordre particulier : 0,4,6,2,1,5,7

→ on remarque que les entrées sont rangées par ordre croissant des derniers bits de la représentation binaire de leurs indices !

L'ordre résultant en binaire est 000,100,010,110,001,101,011,111
 il s'agit de l'ordre classique croissant 000,001,010,011,100,101,110,111
 mais les bits sont inversés

Les inventeurs de la FFT

Gauss, Carl Friedrich, « Nachlass : Theoria interpolationis methodo nova tractata », Werke, Band 3, 265-327 (Königliche Gesellschaft der Wissenschaften, Göttingen, 1866)



Johann Carl Friedrich Gauss (1777-1855)

Les inventeurs de la FFT (suite)

James W. Cooley, and John W. Tukey, « An algorithm for the machine calculation of complex Fourier series », Math. Comput. 19, 297-301 (1965)



James Cooley (1926-)



John Tukey (1915-2000)

La FFT est implantée dans bon nombre d'outils pour le calcul scientifique.

- en MATLAB, il faut utiliser la commande `fft` pour la FFT et la commande `ifft` pour la transformée de Fourier inverse
- il existe un code très efficace en C pour calculer des FFT : FFTW (<http://www.fftw.org/>)

Conclusion

- un algorithme très efficace (en $\mathcal{O}(n \log n)$)
- des applications dans de nombreux domaines (imagerie, traitement du signal, calcul formel, théorie des nombres, etc)
- il existe d'autres algorithmes que celui de Cooley-Tukey
- si les données sont réelles, on utilise plutôt une transformée en cosinus discret