Transformée de Fourier discrète (DFT) et transformée de Fourier rapide (FFT)

Théorie et programmation

La transformée de Fourier discrète s'inscrit dans les méthodes d'évaluation et d'interpolation de polynômes. Nous commençons par nous placer dans l'ensemble des nombres complexes, celui-ci possède une structure de corps, puis nous ferons de même dans l'ensemble des nombres modulo *p* avec p premier, cet ensemble ayant aussi une structure de corps.

Considérons un polynôme de degré inférieur à *n*, de la forme :

$P(X) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \ldots + a_{n-1} X^{n-1}$

où les coefficients a_i sont des nombres complexes, ou réels. Ce polynôme est caractérisé par le tableau à *n* entrées de ses coefficients a_i. Il s'agit d'un vecteur à *n* composantes (complexes ou réelles) si l'on préfère. Les valeurs attribuées à la variable *X* seront aussi des nombres complexes.

1) Evaluation et interpolation d'un polynôme

Evaluer le polynôme en X_0, nombre complexe donné, signifie calculer $P(X_0)$. Par exemple $P(0) = a_0$, ou $P(1) = a_0 + a_1 + a_2 + a_3 + \ldots + a_{n-1}$. Maintenant évaluons le polynôme en *n* valeurs X_0, X_1, …, X_{n-1}, toutes différentes les unes des autres. On obtient ainsi un vecteur à *n* composantes :

$P(X_0), P(X_1), P(X_2) \ldots P(X_{n-1})$

Cette évaluation peut s'écrire sous forme de matrice. Par exemple pour $n = 4$:

$$\begin{pmatrix} P(X_0) \\ P(X_1) \\ P(X_2) \\ P(X_3) \end{pmatrix} = M \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} \text{ avec } M = \begin{pmatrix} 1 & X_0 & X_0^2 & X_0^3 \\ 1 & X_1 & X_1^2 & X_1^3 \\ 1 & X_2 & X_2^2 & X_2^3 \\ 1 & X_3 & X_3^2 & X_3^3 \end{pmatrix}$$

On démontre[1] que cette matrice *M* , appelée matrice de Vandermonde, est inversible, ce qui permet d'écrire :

[1] Montrons-le pour $n = 4$, la généralisation étant simple. Remplaçons provisoirement X_3 par une variable *X*, avec la matrice $\begin{pmatrix} 1 & X_0 & X_0^2 & X_0^3 \\ 1 & X_1 & X_1^2 & X_1^3 \\ 1 & X_2 & X_2^2 & X_2^3 \\ 1 & X & X^2 & X^3 \end{pmatrix}$, et calculons le déterminant de cette matrice. Ce déterminant est un polynôme en *X* de degré trois. On constate qu'en faisant $X = X_0$ ou X_1 ou X_2, on a deux lignes égales, le polynôme s'annule, il a pour racines X_0, X_1, X_2. On peut mettre en facteur $(X - X_0)$ $(X - X_1)$ $(X - X_2)$. Puis on fait $X = X_3$. En appelant V_4 le déterminant de la matrice *M* pour $n = 4$, on en déduit que $V_4 = V_3 (X_3 - X_0) (X_3 - X_1) (X_3 - X_2)$ car V_3 est le coefficient de X_3^3. Cela se généralise à *n* quelconque. Puis on démontre, par un raisonnement par récurrence aisé, que $V_n = \prod_{0 \le i < j < n} (X_j - X_i)$. Ce déterminant étant non nul, la matrice *M* est inversible.

$$\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = M^{-1} \begin{pmatrix} P(X_0) \\ P(X_1) \\ P(X_2) \\ P(X_3) \end{pmatrix}$$

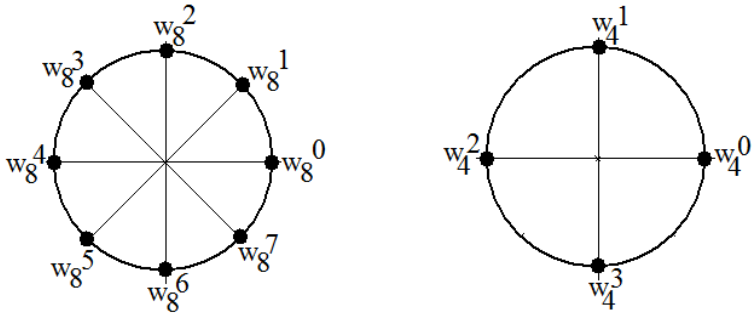
Autrement dit, la connaissance des n valeurs du polynôme en n points tous distincts, soit $y_i = P(X_i)$, permet de retrouver le polynôme P. Ce passage, inverse de celui de l'évaluation, s'appelle interpolation.[2] Il s'agit maintenant de préciser sur quels nombres distincts X_i vont être pratiquées l'évaluation ou l'interpolation. Le choix va se porter sur les racines $n^{èmes}$ de l'unité.

2) Racines $n^{èmes}$ de l'unité

L'équation $z^n = 1$ admet n solutions complexes. Cela signifie que le nombre 1 admet n racines $n^{èmes}$. Celles-ci ont pour images dans le plan complexe les sommets d'un polygone régulier inscrit dans le cercle de centre O et de rayon 1, avec un sommet de coordonnées (1, 0).[3] Les coordonnées de ces points, ou encore les parties réelle et imaginaire des racines $n^{èmes}$ sont $(\cos(2\pi\, k / n), \sin(2\pi\, k / n))$ pour k allant de 0 à $n - 1$.

Appelons w_n la racine principale $n^{ème}$ de 1, soit $w_n = \cos(2\pi / n) + i \sin(2\pi / n)$. Les n racines $n^{èmes}$ peuvent toutes s'écrire comme des puissances de celle-ci, soit $w_n^0 = 1, w_n^1, w_n^2, \ldots, w_n^{n-1}$.

Exemples :



Les 8 racines $8^{èmes}$ de 1, comme puissances de w_8. A droite, les 4 racines $4^{èmes}$ de 1.

On constate qu'en élevant au carré les huit racines $8^{èmes}$ de 1, on trouve les 4 racines $4^{èmes}$ de 1, celles-ci étant répétées deux fois :

$(w_8^0)^2 = w_4^0 = 1$,
$(w_8^1)^2 = w_8^2 = w_4^1$
$(w_8^2)^2 = w_8^4 = w_4^2$
$(w_8^3)^2 = w_8^6 = w_4^3$
$(w_8^4)^2 = w_8^8 = w_8^0 = w_4^0$
$(w_8^5)^2 = w_8^2 = w_4^1$
$(w_8^6)^2 = w_8^4 = w_4^2$
$(w_8^7)^2 = w_8^6 = w_4^2$

Remarquons aussi que les quatre dernières racines 8èmes sont les opposées des quatre premières, comme par exemple $w_8^5 = - w_8^1$, avec la formule (pour n pair) : $w_n^{k+n/2} = - w_n^k$.

Ces propriétés, qui se généralisent aisément, nous seront très utiles par la suite.

---

[2] Pour plus de précision sur l'évaluation et l'interpolation des polynômes, voir chapitre *calculs* dans le cours *informatique et mathématiques*, rubrique *enseignements*.

[3] Pour un rappel sur les nombres complexes, voir chapitre *nombres complexes* dans la rubrique *enseignements / informatique et mathématiques*.

3) Transformée de Fourier discrète

(ou *DFT, Discrete Fourier Transform*)

Par définition, la transformée de Fourier discrète du polynôme $P(X)$ de degré inférieur à n, ou encore du vecteur à n composantes complexes a_0, a_1, a_2, a_3, …, a_{n-1}, est le vecteur obtenu en évaluant le polynôme sur les n racines $n^{èmes}$ de l'unité, soit

$P(w_n^{\,0}), P(w_n^{\,1}), P(w_n^{\,2}), \ldots, P(w_n^{\,n-1})$

La matrice M associée a pour coefficient sur la ligne i et la colonne j le nombre complexe $w_n^{\,i\,j}$, elle s'écrit par exemple pour $n = 4$:

$$M = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & w_n & w_n^2 & w_n^3 \\ 1 & w_n^2 & w_n^4 & w_n^6 \\ 1 & w_n^3 & w_n^6 & w_n^9 \end{pmatrix}$$

Pourquoi avoir choisi les racines $n^{èmes}$ de 1 plutôt que d'autres nombres, notamment des nombres réels puisque les coefficients a_i sont souvent eux-mêmes réels ? Parce que le fait de faire intervenir des cosinus et des sinus, qui sont les coordonnées des racines de l'unité, rapproche la transformée de Fourier discrète de la transformée de Fourier classique, telle qu'on la définit dans le domaine du continu, et par la même occasion cela rend la transformation inverse très simple, à savoir l'interpolation.

4) Interpolation et inverse de la transformée de Fourier discrète

L'interpolation, qui fait passer des valeurs $y_i = P(w_n^{\,i})$ aux coefficients a_i du polynôme, fait intervenir la matrice M^{-1}, par exemple pour $n = 4$:

$$\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = M^{-1} \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix}$$

On montre[4] que $M^{-1} = \frac{1}{4} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & w_n^{-1} & w_n^{-2} & w_n^{-3} \\ 1 & w_n^{-2} & w_n^{-4} & w_n^{-6} \\ 1 & w_n^{-3} & w_n^{-6} & w_n^{-9} \end{pmatrix}$ ici pour $n = 4$.

Autrement dit, la matrice M^{-1} est proche de la matrice M, seuls les exposants des racines de l'unité sont changés de signe, avec une division supplémentaire par n.

Ainsi, pour $n = 4$ et i entre 0 et 3 :
$a_i = (1/4)\,(y_0 + y_1\, w_n^{\,-i} + y^2\, w_n^{\,-2i} + y^3\, w_n^{\,-3i}$

et dans le cas général :

[4] Vérifions qu'avec ce choix de M^{-1} on a bien $M\,M^{-1} = Id$. Le terme général du produit, sur la ligne i et la colonne j, vaut $\frac{1}{n}\sum_{k=0}^{n-1} w_n^{ik} w_n^{-kj} = \frac{1}{n}\sum_{k=0}^{n-1} w_n^{k(i-j)}$. Pour $i = j$, cela fait $n/n = 1$, et pour $j \neq i$, la somme donne 0, comme somme de termes d'une suite géométrique. C'est bien l'identité Id.

$$a_i = \frac{1}{n}\sum_{j=0}^{n-1} y_j w_n^{-ij}$$

Un autre intérêt de la transformée de Fourier, qui pratique l'évaluation du polynôme sur les racines de l'unité, est de rendre possible un algorithme particulièrement performant, selon la technique de la division en deux. C'est ce qui s'appelle la transformée de Fourier rapide.

5) Transformée de Fourier rapide

(ou *FFT, Fast Fourier Transform*)

Supposons dorénavant que n est une puissance de 2, ce qui va permettre de faire des divisions successives par 2 sans erreur. On peut réécrire le polynôme $P(X)$ en séparant ses coefficients d'indice pair et ceux d'indice impair, soit :

$P(X) = (a_0 + a_2 X^2 + a_4 X^4 + a_6 X^6) + X (a_1 + a_3 X^2 + a_5 X^4 + a_7 X^7)$ ici pour $n = 8$,

$= P_{\text{pair}}(X^2) + X P_{\text{impair}}(X^2)$

en posant $P_{\text{pair}}(x) = a_0 + a_2 x + a_4 x^2 + a_6 x^3$ et $P_{\text{impair}}(x) = a_1 + a_3 x + a_5 x^2 + a_7 x^3$

Maintenant remplaçons X par une racine $n^{\text{ème}}$ de l'unité. On a vu que le fait d'élever au carré transformait une racine $n^{\text{ème}}$ en racine $(n/2)^{\text{ème}}$, et cela à deux reprises lorsque l'on prend toutes les racines $n^{\text{èmes}}$. On sait aussi que les racines $n^{\text{èmes}}$, avec n pair, sont deux à deux opposées. Prenons par exemple $n = 8$:

$P(w_8^0) = P_{\text{pair}}(w_4^0) + w_8^0 P_{\text{impair}}(w_4^0)$
$P(w_8^1) = P_{\text{pair}}(w_4^1) + w_8^1 P_{\text{impair}}(w_4^1)$
$P(w_8^2) = P_{\text{pair}}(w_4^2) + w_8^2 P_{\text{impair}}(w_4^2)$
$P(w_8^3) = P_{\text{pair}}(w_4^3) + w_8^3 P_{\text{impair}}(w_4^3)$
$P(w_8^4) = P_{\text{pair}}(w_4^0) - w_8^0 P_{\text{impair}}(w_4^0)$
$P(w_8^5) = P_{\text{pair}}(w_4^1) - w_8^1 P_{\text{impair}}(w_4^1)$
$P(w_8^6) = P_{\text{pair}}(w_4^2) - w_8^2 P_{\text{impair}}(w_4^2)$
$P(w_8^7) = P_{\text{pair}}(w_4^3) - w_8^3 P_{\text{impair}}(w_4^3)$

Evaluer le polynôme sur n valeurs revient à évaluer deux polynômes de longueur moitié sur $n/2$ valeurs, avec en plus $n/2$ multiplications par des racines $n^{\text{èmes}}$ (et n additions aussi).[5] Cela explique la performance de cet algorithme, et son nom de transformée de Fourier rapide.

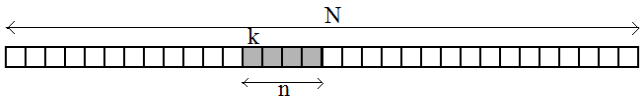
6) Programmation récursive de la FFT

Le fait d'avoir une fonction d'évaluation qui se rappelle deux fois sur deux moitiés invite à faire un programme récursif. Après s'être donne N sous forme d'une puissance de 2, on place les N coefficients a_i du polynôme initial dans un tableau $a[N]$ déclaré en global, de façon qu'il soit sensible à chaque appel de fonction. A la fin de l'exécution du programme, ce même tableau $a[]$ contiendra les coefficients de la transformée de Fourier.[6] Comme la fonction d'évaluation se rappelle sur des moitiés, puis des moitiés de moitiés, etc., il s'agit de bien connaître le morceau du tableau sur lequel on opère. D'où la fonction *evaluer* (k, n) qui agit sur le morceau commençant par la case numéro k et de longueur n. Le fait de calculer de nouvelles valeurs dans ce morceau à partir des anciennes exige de

[5] Le temps $T(n)$ mis pour évaluer n cas est tel que $T(n) = 2\ T(n/2) + n$, d'où $T(n)$ de l'ordre de $n \log n$.

[6] On notera l'analogie de ce programme avec celui du tri par fusion, un classique des algorithmes récursifs agissant sur un tableau. Comme celui-ci est plus simple que celui de la transformée de Fourier, on aura intérêt à bien le comprendre avant de se lancer dans le programme actuel (voir chapitre *récursif et itératif* dans le cours d'*algorithmique*, rubrique *enseignements*). Pour les mêmes raisons, cet algorithme est en $n \log n$.

faire les calculs dans un tableau auxiliaire *t*[] pour éviter les interférences, avant de renvoyer les résultats obtenus dans le morceau du tableau *a*[] concerné, où les nouvelles valeurs écrasent les anciennes.



Le test d'arrêt de la fonction se produit pour $n = 2$. L'évaluation se fait sur le polynôme de degré inférieur à 2, de la forme $P(x) = a_k + a_{k+1}\, x$, par exemple $P(x) = a_0 + a_1\, x$ si $k = 0$. Il s'agit de l'évaluer sur les deux racines carrées de 1, soit 1 et – 1. On trouve :

$P(1) = a_k + a_{k+1}$
$P(-1) = a_0 - a_{k+1}$

Lorsque l'on fait *evaluer*(*k*, 2), ces deux résultats sont d'abord calculés dans un tableau auxiliaire *t*[], puis ils sont placés dans les cases *k* et *k* +1 du tableau *a*[] où ils écrasent les anciennes valeurs qui étaient a_k et a_{k+1}.

Pour comprendre le cas général, prenons l'exemple simple où $N = 4$, et appelons la fonction *evaluer*(0, 4). Le polynôme $P(x) = a_0 + a_1\, x + a_2\, x^2 + a_3\, x^3$ dont les 4 coefficients sont placés dans le tableau *a*[], commence par être séparé suivant ses coefficients d'indice pair et ceux d'indice impair, par le biais d'un tableau auxiliaire *t*[]. Au terme de ce transfert, le tableau contient les coefficients $a_0\ a_2\ a_1\ a_3$ dans cet ordre. Tout se passe comme si l'on avait deux polynômes de degré moitié en succession : $P_{pair}(x) = a_0 + a_2\, x^2$ et $P_{impair}(x) = a_1 x + a_3\, x^3$. Puis la fonction d'évaluation est rappelée sur ces deux moitiés : $a_0\ a_2$ (pour $k = 0$) et $a_1\ a_3$ (pour $k = 2$). Grâce au test d'arrêt, le tableau *a*[] contient maintenant les quatre termes $a_0 + a_2$, $a_0 - a_2$, $a_1 + a_3$, $a_1 - a_3$. Il s'agit, dans cet ordre, de

$P_{pair}(1)$, $P_{pair}(-1)$, $P_{impair}(1)$, $P_{impair}(-1)$.

Ces quatre nombres permettent d'évaluer le polynôme *P* sur les 4 racines quatrièmes de l'unité, en faisant :
$P(1) = P_{pair}(1) + P_{impair}(1) = a_0 + a_1 + a_2 + a_3$
$P(i) = P_{pair}(-1) + i\, P_{impair}(-1) = a_0 - a_2 + i\,(a_1 - a_3)$
$P(-1) = P_{pair}(1) - P_{impair}(1) = a_0 + a_1 - (a_2 + a_3)$
$P(-i) = P_{pair}(-1) - i\, P_{impair}(-1) = a_0 - a_2 - i\,(a_1 - a_3)$
et ces résultats sont placés dans le tableau *a*[] après avoir transité par un tableau auxiliaire *t*[].

Dans le cas général, où *N* peut être bien supérieur à 4, il arrivera que la fonction se rappelle sur *evaluer*(*k*, 4) et non plus sur *evaluer*(0, 4). Il conviendra de tenir compte de ce décalage lorsque l'on effectue les calculs. Il n'y a plus qu'à écrire le programme. Dans ce qui suit, le programme principal se contente de calculer au préalable les racines $N^{èmes}$ de 1, puis il est chargé de remplir le tableau initial *a*[*N*]. En fait, les coefficients a_i étant des nombres complexes, on fabrique deux tableaux, l'un avec la partie réelle de a_i, l'autre avec sa partie imaginaire. Puis on appelle la fonction *evaluer*(0, *N*) qui ramène la transformée de Fourier dans le même tableau *a*[].

Cela étant fait, on peut alors faire agir sur le tableau *a*[] obtenu l'inverse de la transformée de Fourier afin de retrouver à la fin le tableau *a*[] du polynôme initial. Il suffit pour cela de prendre les racines $-N^{èmes}$ à la place des racines $N^{èmes}$ dans le tableau *w*[] et de les injecter dans la même fonction d'évaluation, en divisant finalement les résultats par *N*. Et la boucle est bouclée.

```
#define N 16
#define deuxpi (2.*M_PI)
double wreel[N],wimag[N],areel[N],aimag[N], aareel[N],aaimag[N],treel[N],timag[N];

int main()
{  int i;
   racinesnemesde1(N);
```

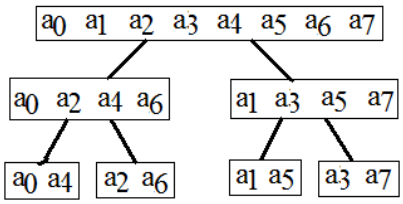
```
   polynome(N);
   evaluer(0,N);
   printf("\n\n Evluation par FFT\n");
   for(i=0;i<N;i++) printf( " (%3.1lf  %3.1lf) ",areel[i],aimag[i]);

/* interpolation par FFT inverse à partir du résultat obtenu par la FFT */
   racinesmoins(N);
   evaluer(0,N);
   printf("\n\n Interpolation par FFT\n");
   for(i=0;i<N;i++) printf( " (%3.1lf  %3.1lf) ",areel[i]/(float)N,aimag[i]/(float)N);
  getchar();return 0;
}

void racinesnemesde1(int n)
{ int i;
  for(i=0;i<n;i++)  { wreel[i]=cos(deuxpi*i/(float)n);  wimag[i]=sin(deuxpi*i/(float)n);  }
}
void polynome(int n)
 { int i;
   for(i=0;i<n;i++) {areel[i]=i+1; aimag[i]=0;}  /* un exemple de polynôme */
   printf("\n\n Polynome avec ses %d coefficients :\n",N);
   for(i=0;i<N;i++) printf(" (%3.2lf  %3.2lf) ",areel[i],aimag[i]);
 }
void evaluer(int k, int n)
{  double wPimpreel,wPimpimag;    int nsur2,i,j,jj,expo;
   if (n==2)  { treel[0]=areel[k]; treel[1]=areel[k+1];   /* test d'arrêt */
                areel[k]=treel[0]+treel[1]; areel[k+1]=treel[0] - treel[1];
                timag[0]=aimag[k]; timag[1]=aimag[k+1];
                aimag[k]=timag[0]+timag[1]; aimag[k+1]=timag[0] - timag[1];
              }
   else
      {  nsur2=n/2;
         for(i=0;i<nsur2;i++)
           { j= k+ 2*i;
             treel[i]=areel[j]; treel[i+nsur2]=areel[j+1];  /* P_pair et P_impair  */
             timag[i]=aimag[j]; timag[i+nsur2]=aimag[j+1];
           }
         for(i=0;i<n;i++)  {  areel[k+i]=treel[i];  aimag[k+i]=timag[i];}
         evaluer(k,nsur2);   /* rappels de la fonction d'évaluation */
         evaluer(k+nsur2,nsur2);
         jj=N/n;
         for(i=0;i<nsur2;i++)
           { expo=i*jj;
             /* calculs du produit de P_impair avec les racines n^èmes */
              wPimpreel= areel[k+nsur2+i]*wreel[expo] - aimag[k+nsur2+i]*wimag[expo];
              wPimpimag= areel[k+nsur2+i]*wimag[expo] +aimag[k+nsur2+i]*wreel[expo];
              treel[i]=areel[k+i]+wPimpreel;    timag[i]=aimag[k+i]+wPimpimag;
              treel[i+nsur2]=areel[k+i]-wPimpreel;   timag[i+nsur2]=aimag[k+i]-wPimpimag;
           }
         for(i=0;i<n;i++)  { areel[k+i]=treel[i];   aimag[k+i]=timag[i];   }
      }
}
void racinesmoins(int n)
{ int i;
  for(i=0;i<n;i++)  { wreel[i]=cos(-deuxpi*i/(float)n);  wimag[i]=sin(-deuxpi*i/(float)n);  }
}
```

7) Programmation itérative de la FFT

La méthode de la FFT consiste à partir du polynôme P avec ses coefficients a_i (i de 0 à $N-1$), puis à le couper en deux de façon répétée, les nouveaux polynômes correspondants étant placés en succession l'un après l'autre. Il se produit alors un changement dans l'ordre des coefficients. Par exemple pour $N = 8$, on observe l'évolution suivante :



L'ordre final des coefficients correspond à l'écriture des indices en binaire dans le sens croissant (des poids faibles au poids forts), puis à convertir ces nombres en décimal en les lisant dans le sens décroissant. Il s'agit d'une lecture à l'envers des nombres en binaire (en anglais *bit-reverse*)

```
0    1    2    3    4    5    6    7      nombres successifs
000  001  010  011  100  101  110  111    écriture binaire des nombres dans le sens décroissant
000  100  010  110  001  101  011  111    écriture binaire des nombres dans le sens croissant
0    4    2    6    1    5    3    7      conversion des nombres ci-dessus en décimal
```

Lorsque le programme récursif part d'en haut pour descendre jusqu'aux polynômes de degré 1, le programme itératif fait le parcours en sens inverse : il part d'en bas pour aller vers le haut. La première chose à faire est de prendre le tableau a[] des coefficients du polynôme P et de les replacer dans un tableau t[] en suivant maintenant l'ordre des bits inversés. Cela se fait grâce aux fonctions suivantes :

```
int rev(int i)  /* passage du nombre i (en base 10) à son transformé nombre=rev(i) avec les bits à l'envers */
{ int q,nombre,puissdeux,k,bit;;
  nombre=0; puissdeux=2; q=i;
  for(k=0;k<S;k++)
    { bit=q%2;   nombre +=bit*N/puissdeux;
      q=q/2;  /* division par deux avec quotient q et reste bit */
      puissdeux*=2;
    }
  return nombre;
}
void bitreversedeaverst(void)  /* les coefficients a[] sont mis dans t[] dans l'ordre des bits inversés */
{  int i,j;
   for(i=0;i<N;i++)
    { j=rev(i); treel[j]=areel[i]; timag[j]=aimag[i];}
}
```

Mais comment faire la remontée ? Reprenons l'exemple de $N = 8$, où la racine 8ème principale de l'unité est $w = e^{i\,2\pi/8}$. Le polynôme $P(X)$ qui est coupé en deux :

$P(X) = P_P(X^2) + X\,P_I(X^2)$ où P_P a pour coefficients : $a_0\ a_2\ a_4\ a_6$ et P_I : $a_1\ a_3\ a_5\ a_7$

Puis le polynôme P_P est à son tour coupé en deux :

$P_P(X) = P_{PP}(X^2) + X\,P_{PI}(X^2)$ où P_{PP} a pour coefficients : $a_0\ a_4$ et P_{PI} : $a_2\ a_6$
ainsi que le polynôme P_I : $P_I(X) = P_{IP}(X^2) + X\,P_{II}(X^2)$ où P_{IP} a pour coefficients : $a_1\ a_5$ et P_{PI} : $a_3\ a_7$

La succession des polynômes P_{PP}, P_{PI}, P_{IP}, P_{II} donne bien l'ordre des coefficients dans le sens des bits inversés. Commençons par évaluer ces polynômes en $w^0 = 1$ et $w^4 = -1$, en appelant leurs évaluations respectives $b_0\ b_1\ b_2\ b_3\ b_4\ b_5\ b_6\ b_7$:

$b_0 = a_0 + a_4$ $\quad b_1 = a_0 - a_4$ (avec $P_{PP}(1) = b_0$ et $P_{PP}(-1) = b_1$)
$b_2 = a_2 + a_6$ $\quad b_3 = a_2 - a_6$ ($P_{PI}(1)$ et $P_{PI}(-1)$)
$b_4 = a_1 + a_5$ $\quad b_5 = a_1 - a_5$ ($P_{IP}(1)$ et $P_{IP}(-1)$)
$b_6 = a_3 + a_7$ $\quad b_7 = a_3 - a_7$ ($P_{II}(1)$ et $P_{II}(-1)$)

ou encore en utilisant le tableau $t[]$:
$b_0 = t_0 + t_1$ $\quad b_1 = t_0 - t_1$
$b_2 = t_2 + t_3$ $\quad b_3 = t_2 - t_3$
$b_4 = t_4 + t_5$ $\quad b_5 = t_4 - t_5$
$b_6 = t_6 + t_7$ $\quad b_7 = t_6 - t_7$

A partir de là, évaluons les polynômes P_P et P_I sur les racines w^0, w^2, $w^4 = -w^0$, $w^6 = -w^2$ en notant leurs évaluations respectives $c_0\ c_1\ c_2\ c_3\ c_4\ c_5\ c_6\ c_7$, comme notamment
$c_0 = P_P(1) = P_{PP}(1) + P_{PI}(1)$, $\quad c_1 = P_P(w^2) = P_{PP}(-1) + w^2\ P_{PI}(-1)$, etc. :

$c_0 = b_0 + b_2$ $\quad c_1 = b_1 + w^2\ b_3$ ($P_P(1)$ et $P_P(w^2)$)
$c_2 = b_0 - b_2$ $\quad c_3 = b_1 - w^2\ b^3$ ($P_P(-1)$ et $P_P(-w^2)$)
$c_4 = b_4 + b_6$ $\quad c_5 = b_5 + w^2\ b_7$ ($P_I(1)$ et $P_I(w^2)$)
$c_6 = b_4 - b_6$ $\quad c_7 = b_5 - w^2\ b^7$ ($P_I(-1)$ et $P_I(-w^2)$)

Enfin, évaluons P sur les 8 racines $8^{èmes}$ de l'unité, en notant d_0, d_1, …, d_7 ces évaluations, comme par exemple $d_0 = P(1) = P_P(1) + P_I(1)$, $d_1 = P(w) = P_P(w^2) + w\ P_I(w^2)$, $d_2 = P(w^2) = P_P(-1) + w^2\ P_I(-1)$, etc. :

$d_0 = c_0 + c_4$, $d_1 = c_1 + w\ c_5$, $d_2 = c_2 + w^2\ c_6$, $d_3 = c_3 + w^3\ c_7$
$d_4 = c_0 - c_4$, $d_5 = c_1 - w\ c_5$, $d_6 = c_2 - w^2\ c_6$, $d_7 = c_3 - w^3\ c_7$

On s'aperçoit alors d'une similarité dans les passages successifs des t vers b, des b vers c et des c vers d. C'est cela qui permet la programmation itérative.

L'étape $s = 1$ fait passer des éléments de $t[]$ aux évaluations b, avec les formules :
$b_k = t[k] + 1\ t[k+1]$ avec k pair,
$b_{k+1} = t[k] - 1\ t[k+1]$ où 1 et – 1 sont les racines $8^{èmes}$ prises de 4 en 4.
Ces évaluations sont faites à l'aide des variables auxiliaires u et v, avec ici $u = t[k]$ et $v = r\ t[k+1]$, où k prend les valeurs paires 0, 2, 4, 6, et r est la racine $8^{ème}$ concernée. Une fois ces évaluations effectuées, celles-ci sont remises dans le tableau $t[]$.

L'étape $s = 2$ fait passer de $t[]$ aux évaluations c, avec les formules :
$c_k = t[k] + w^{2k}\ t[k+2]$
$c_{k+2} = t[k] - w^{2k}\ t[k+2]$
Notons que les racines $8^{èmes}$ sont prises de 2 et 2, et que k prend les valeurs 0, 1, 4 et 5.
Une fois ces évaluations faites, on les remet dans le tableau $t[]$.

A l'étape finale $s = 3$, on passe aux évaluations d, avec les formules :
$d_k = t[k] + w^k\ t[k+4]$
$d_{k+4} = t[k] - w^k\ t[k+4]$, avec k prenant les valeurs 0, 1, 2, 3 et les racines $8^{èmes}$ prises de 1 en 1.

On constate que l'on utilise à chaque étape le même type de formule. La principale difficulté consiste à calculer les indices k, ce qui nous amène à faire une double boucle de la forme :

for(j=0; j<n/2; j++) for(k=j; k<N; k+=n) où $n = 2^s$.

Programme

```
#define N 128
#define deuxpi (2.*M_PI)
void racinesnemesde1(int n);
void polynome(int n);
int rev(int i);
void bitreversedeaverst(void);
double wreel[N],wimag[N],areel[N],aimag[N],treel[N],timag[N],ureel,uimag,vreel,vimag;
int S;

int main()
{  int i,j,k,expo,s,n;
   racinesnemesde1(N);  /* fonctions déjà faites, voir le programme récursif */
   polynome(N);
  S=(log(N+0.0001)/log(2.));   /* pour N = 8 = 2^3, on a  S = 3 */
  bitreversedeaverst();
 for(s=1;s<=S;s++)  /* S étapes de calcul */
   { n=(int)pow(2.,s);  /* pour N = 8, n prend les valeurs 2, 4, 8 */
     for(j=0;j<n/2;j++)   for(k=j;k<N;k+=n)
          { ureel=treel[k]; uimag=timag[k];  /* u avec parties réelle et imaginaire */
            vreel=wreel[N/n*j]*treel[k+n/2]-wimag[N/n*j]*timag[k+n/2];  /* v = r t[k+n/2] */
            vimag=wreel[N/n*j]*timag[k+n/2]+wimag[N/n*j]*treel[k+n/2];
            treel[k]=ureel+vreel; timag[k]=uimag+vimag;
            treel[k+n/2]=ureel-vreel; timag[k+n/2]=uimag-vimag;
          }
    }
  printf("\n\n Transformee de Fourier\n");
  for(i=0;i<N;i++)  printf(" (%3.1lf  %3.1lf) ", treel[i],timag[i]);
  getchar();return 0;
}
```

8) Transformée de Fourier rapide dans un corps fini

Nous avons vu la transformée de Fourier d'un polynôme P de degré inférieur à N dans le corps **C** des nombres complexes, qui consistait à évaluer le polynôme sur les N racines $N^{\text{èmes}}$ de l'unité dans C, de la forme w^i avec w racine primitive $N^{\text{ème}}$. Le choix de ces racines $N^{\text{èmes}}$ s'expliquait par la facilité de pratiquer la transformée de Fourier inverse dans ce cas, ce qui permet d'interpoler le polynôme, c'est-à-dire de retrouver le polynôme à partir de ses valeurs sur les N racines de l'unité. Rappelons que le passage du polynôme P (vecteur à N composantes) à sa transformée de Fourier (qui est aussi un vecteur à N composantes) s'effectuait par le biais d'une matrice de Vandermonde M_w, dont le déterminant $|M_w|$ est $\prod_{0 \le i < j < N} (w^i - w^j)$

Pour appliquer la transformée de Fourier rapide (FFT) on devait en plus choisir N comme une puissance de 2, de façon à découper le polynôme en deux à plusieurs reprises. Nous allons maintenant généraliser cela dans un corps qui n'est plus celui des nombres complexes, mais un corps fini, pour nous ce sera $\mathbf{Z}_p$, c'est-à-dire l'ensemble des nombres modulo p avec p premier, ce qui peut s'écrire $\mathbf{Z}_p = \{0, 1, 2, \ldots, p-1\}$ lorsque l'on ramène les nombres modulo p, entre 0 et $p-1$. Mais commençons par quelques précisions théoriques.

a) Racine primitive $N^{ème}$ de l'unité dans un anneau [7] ou un corps

Si dans un corps tous les éléments sauf 0 sont inversibles, dans un anneau tous les éléments ne sont pas inversibles, et il peut exister des diviseurs de 0. Par exemple $Z_6 = \{0, 1, 2, 3, 4, 5\}$ le groupe des inversibles est $\mathbf{U}(6) = \{1, 5\}$ à savoir les nombres premiers avec 6, et il existe des diviseurs de 0, ici 2, 3, 4 puisque $2\times3 = 0$ ou $3\times4 = 0$. Précisons qu'un élément qui n'est pas un diviseur de 0 est forcément inversible.

Définition

Dans un anneau, un élément w est une racine primitive $N^{ème}$ de l'unité si :
- $w^N = 1$
- $1 - w^i$ est inversible pour tout i entre 1 et $N - 1$

Cette définition implique d'abord que tous les w^i sont inversibles, puisque 1 est inversible, et que pour i entre 1 et $N - 1$, le fait d'avoir $w^i\, w^{N-i} = 1$ prouve que w^i est inversible. Elle implique aussi que w est d'ordre N, puisque $w^N = 1$ et $w^i \neq 1$ pour les i de 1 à $N - 1$ (en effet, si l'on avait $w^i = 1$, on aurait $1 - w_i = 0$, et $1 - w^i$ ne serait pas inversible). Enfin, sachant que le déterminant $|M_w|$ de la matrice de Vandermonde est formé de facteurs de la forme $(w^i - w^j) = w^i(1 - w^{j-i})$, chacun de ces facteurs est inversible, et par suite le déterminant $|M_w|$ est inversible, ce qui signifie que la matrice M_w est aussi inversible dans l'anneau : $(M_w)^{-1}$ existe.

Si l'anneau est un corps, la définition se simplifie : Une racine primitive $N^{ème}$ de l'unité est un élément w d'ordre N : $w^N = 1$ et $w^i \neq 1$ pour i de 1 à $N - 1$.

Racine primitive $N^{ème}$ de l'unité dans le corps $\mathbf{Z}_p$ et transformée de Fourier rapide

Dans le corps $\mathbf{Z}_p$ à p éléments, avec son groupe d'inversibles $\mathbf{U}(p)$ à $p - 1$ éléments (de 1 à $p - 1$), on sait que l'ordre d'un élément inversible est un diviseur de $p - 1$. On a finalement deux contraintes sur N :

N divise $p - 1$,

N est une puissance de 2, de façon à pouvoir pratiquer la transformée de Fourier rapide.

Dans ces conditions, comment avoir une racine primitive $N^{ème}$ de l'unité ? Sachant que $\mathbf{U}(p)$ est un groupe cyclique, il admet au moins un générateur g dont les puissances successives décrivent $\mathbf{U}(p)$, c'est-à-dire que g est d'ordre $p - 1$. Prenons $g^{(p-1)/N}$, cet élément est d'ordre N, il s'agit d'une racine primitive w.

Exemples

- $p = 17$ et $N = 8$, puissance de 2 qui divise $p - 1$. Un générateur de $\mathbf{U}(17)$ est 3. Un élément d'ordre $N = 8$ est $3^{16/8} = 9$. Son inverse 2 est aussi d'ordre 8, comme on peut le vérifier en prenant ses puissances successives : 2, 4, 8, 16, 15, 13, 9, 1.
- $p = 17$ et $N = 16$, puissance de 2 qui divise $p - 1$. Grâce à ce qui précède, une racine primitive est $w = 3$ qui est un générateur, et son inverse 6 est aussi d'ordre 16.
- $p = 97$ et $N = 32$. Un générateur de $\mathbf{U}(97)$ est 5. On prend comme racine primitive $32^{ème}$ de l'unité $w = 5^{96/32} = 5^3 = 28$.

[7] Ce paragraphe permet de définir une transformée de Fourier dans un anneau [NAU1992], de façon plus générale que dans un corps. Mais nous nous contentons ici de donner la définition d'une racine primitive, sans aborder le problème plus avant. Notre propos concerne en effet le cas plus simple du corps $\mathbf{Z}_p$.

b) Programmation

On se donne le nombre premier P ainsi que le nombre N qui est une puissance de 2 et qui divise $N - 1$, comme on l'a fait dans les exemples précédents. On détermine aussi un générateur g du groupe cyclique $\mathbf{U}(p)$, ce qui permet de trouver une racine primitive $N^{\text{ème}}$ de l'unité dans $\mathbf{U}(p)$, soit W. Les racines $N^{\text{èmes}}$ de l'unité sont les puissances successives de W, ramenées modulo P. Elles sont obtenues par la fonction suivante, qui les enregistre dans le tableau $w[N]$:

```
void racinesnemesde1(int ww)  /* dans le programme principal, on prendre ww = W */
{ int i;
  w[0]=1;   /* le tableau w[] doit être déclaré en global */
  for(i=1;i<N;i++)   { w[i]=ww*w[i-1]; while (w[i]>=P) w[i]-=P; }
}
```

Les coefficients $a[]$ du polynôme P donné sont mis dans l'ordre des bits inversés et placés dans le tableau $t[]$, comme on l'a déjà fait précédemment, ce tableau $t[]$ étant déclaré en global :

```
void bitreversedeaverst(int aa[])  /* cette fonction s'appliquera au polynôme P initial */
{  int i,j;
   for(i=0;i<N;i++)  { j=rev(i); t[j]=aa[i];} /* reprendre la fonction rev() traitée précédemment */
}
```

Ces deux fonctions vont être intégrées dans la fonction *fft*() qui prend comme variables le tableau des coefficients $a[]$ du polynôme dont on veut avoir la transformée de Fourier rapide, ainsi que la racine primitive W utilisée pour cela. Il s'agit d'un programme analogue à celui de la transformée de Fourier avec des nombres complexes, sauf que dans le cas présent, il s'agit de nombres entiers beaucoup plus simples, que l'on se contente de ramener modulo P lors des calculs. La transformée de Fourier du polynôme se trouve placée à la fin dans le tableau $t[]$ dont le contenu s'est transformé au cours de l'exécution de la fonction *fft*().

```
void fft(int aa[],int ww)
{ int j,k,s,n,u,v;
  racinesnemesde1(ww);
  S=(log(N+0.0001)/log(2.));    /* S est déclaré en global */
  bitreversedeaverst(aa);
  n=2;
  for(s=1;s<=S;s++)
    { for(j=0;j<n/2;j++)   for(k=j;k<N; k+=n)
            { u=t[k];
              v=w[N/n*j]*t[k+n/2]; while(v>=P) v-=P;
              t[k]=u+v; if (t[k]>=P) t[k]-=P;
              t[k+n/2]=u-v; if (t[k+n/2]>=P) t[k+n/2]-=P;   if (t[k+n/2]<0) t[k+n/2]+=P;
            }
       n=2*n;
    }
}
```

Le programme principal consiste à entrer le polynôme P avec ses coefficients $a[]$. Puis la fonction *fft*() est appelée, et les résultats affichés dans le tableau $t[]$ ou si l'on préfère dans un tableau *ffta*[].

```
   for(i=0;i<N;i++) a[i]=rand()%(N-1)+1;  /* un exemple de polynôme */
   printf("\n Polynome a avec ses %d coefficients :\n",N);   for(i=0;i<N;i++) printf(" %d ",a[i]);

   fft(a,W);
   printf("\n Transformee de Fourier de a\n");    for(i=0;i<N;i++) { ffta[i]=t[i]; printf(" %d ", ffta[i]);}
```

c) Produit de deux polynômes

Un intérêt majeur de la transformée de Fourier est de rendre plus rapide la multiplication de deux polynômes. Dans le cas présent, on se place dans Z_p et l'on a défini N comme une puissance de 2 divisant $p - 1$, avec W racine primitive $N^{ème}$ de l'unité.

Les polynômes utilisés ont leurs coefficients dans $\mathbf{Z}_p$ et leurs évaluations sont aussi faites dans $\mathbf{Z}_p$. On sait que le produit de deux polynômes de degré inférieur à $N/2$ est un polynôme de degré inférieur à N. Par les méthodes de calcul classiques, la performance de l'algorithme du produit est de l'ordre de N^2.[8] Une autre méthode consiste à utiliser la transformée de Fourier, avec une performance améliorée, de l'ordre de $N \log N$, comme on l'a vu. Comment s'y prend-on ?

On se donne deux polynômes A et B de degré inférieur à $N/2$ définis par leurs coefficients $a[N]$ et $b[N]$ modulo p, par exemple pour $N = 8$:

$A(X) = 7 + 2X + 7X^2 + 6X^3$ avec le tableau $a[8]$: 7, 2, 7, 6, 0, 0, 0, 0
$B(X) = 4 + 3X + 6X^2 + X^3$

On commence par chercher leurs transformées de Fourier, ce qui donne les évaluations des polynômes sur les N puissances de W, soit :

$FFT(A, W)$: 5, 2, 1, 7, 6, 0, 16, 2
$FFT(B, W)$: 14, 8, 6, 6, 6, 14, 7, 5

L'évaluation du produit $C = AB$ sur les N puissances de W se fait tout simplement en multipliant les évaluations de A et de B terme à terme, par exemple $5 \times 14 = 2$ [modulo17], puis $2 \times 8 = 16$, etc. On obtient ainsi très facilement ce qui est la transformée de Fourier C' de C. Dans le cas présent :

$C' = FFT(C, W)$: 2, 16, 6, 8, 2, 0,10, 10

Pour obtenir C, il suffit de prendre la transformée de Fourier inverse. On sait que celle-ci consiste à faire la transformée de Fourier avec les puissances de W^{-1}, et à multiplier le résultat par N^{-1}. Finalement :

$C = N^{-1} FFT(C', W^{-1})$

Dans notre exemple, on trouve $C(X) = 11 + 12X + 8X^2 + 13X^3 + 11X^4 + 9X^5 + 6X^6$ [modulo 17]

Le programme correspondant reprend les fonctions utilisées précédemment. Le programme principal s'articule ainsi :

```
#define P 97  /* un exemple */
#define N 32  /* puissance de 2 qui divise 96 */
#define W 28  /* racine primitive Nème de l'unité */
int w[N],t[N],y[N],S,invN,invW;

int main()
```

[8] Le produit de deux polynômes définis par leurs tableaux de coefficients $a[]$ et $b[]$ suivant les puissances croissantes donne un polynôme dont les coefficients $c[]$ peuvent être calculés ainsi :

```
printf("\n\n Polynome c = ab  avec ses %d coefficients :\n",N);
for(i=0;i<N;i++) c[i]=0;
for(i=0;i<N/2;i++) for(j=0;j<N/2;j++) c[i+j]+=a[i]*b[j];
for(i=0;i<N;i++) { while (c[i]>=P) c[i]-=P; printf(" %d ",c[i]);}
```

Pour plus de détails, voir le chapitre *calculs numériques* dans le cours *informatique et mathématiques*, rubrique *enseignements*.

```
{ int i,j,a[N],ffta[N];   int b[N],fftb[N]; int c[N],p[N];
   for(i=0;i<N/2;i++) a[i]=i+1;    for(i=N/2;i<N;i++) a[i]=0; /* polynome a */
   printf("\n Polynome a avec ses %d coefficients :\n",N);   for(i=0;i<N;i++) printf(" %d ",a[i]);

   fft(a,W);
   printf("\n Transformee de Fourier de a\n"); for(i=0;i<N;i++) { ffta[i]=t[i]; printf(" %d ", ffta[i]);}

   printf("\n\n");  /* polynome b */
  for(i=0;i<N/2;i++) b[i]=rand()%(N-1)+1;
   for(i=N/2;i<N;i++) {b[i]=0;}
   printf(" Polynome b avec ses %d coefficients :\n",N);    for(i=0;i<N;i++) printf(" %d ",b[i]);

   fft(b,W);
   printf("\n Transformee de Fourier de b\n"); for(i=0;i<N;i++) { fftb[i]=t[i]; printf(" %d ", fftb[i]);}

   printf("\n\n Produit  p des transformees de Fourier de a et de b terme a terme\n" );
  for(i=0;i<N;i++) { p[i]=ffta[i]*fftb[i]; while(p[i]>=P) p[i]-=P;}
  for(i=0;i<N;i++) printf(" %d ", p[i]);

  /* transformée inverse de p */
  invW=inverse(W);   invN=inverse(N);
  fft(p,invW);
  for(i=0;i<N;i++) { t[i]=t[i]*invN; while (t[i]>=P) t[i]-=P;}
  printf("\n\n **** Transformee inverse de p donnant c = ab \n"); for(i=0;i<N;i++) { printf(" %d ",t[i]); }
 getchar();return 0;
}
```

Avec l'appoint de la fonction déterminant l'inverse d'un nombre modulo *p*, écrite ici de façon rustique :

```
int inverse(int x)
{ int i,produit ;
   for(i=1;i<=P-1;i++)
     { produit=x*i; while (produit>=P) produit-=P;
        if (produit==1) return i;
      }
}
```