Exemple d'usage de FFTW

Anne Cadiou

Laboratoire de Mécanique des Fluides et d'Acoustique

Informatique scientifique pour le calcul
Formations transverses des Écoles Doctorales
LyonCalcul, 2019

Transformée de Fourier

Soit $x(t)$ un signal périodique, continu en temps. Sa transformée de Fourier, $\hat{x}(k)$ où k est la fréquence, est définie par

$$\hat{x}(k) = \int_{-\infty}^{+\infty} x(t)\, \exp^{-2\pi Ikt}\, dt$$

En pratique, le signal est discret en temps et compris dans l'intervalle $t \in [0; T[$ de sorte que l'intégrale est approchée par

$$\hat{x}(k) = \sum_{n=0}^{N-1} x(t_n) \exp^{-2\pi Ikt_n}$$

où avec une discrétisation régulière, $t_n = nT/N$, $n \in [0; N-1]$. N est le nombre d'échantillons.

Implémentation naïve

L'algorithme naïf consiste à calculer chaque échantillon fréquentiel à partir de tous les échantillons temporels.

Complexité : $O(N^2)$

De façon matricielle, il s'écrit (matrice de Vandermonde) :

$$\begin{pmatrix} \hat{x}(0) \\ \hat{x}(1) \\ \vdots \\ \hat{x}(k) \\ \vdots \\ \hat{x}(N-1) \end{pmatrix} = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ 1 & \omega & & \omega^{(N-1)} \\ \vdots & & & \vdots \\ 1 & & \omega^{k\,n} & \omega^{(N-1)\,k} \\ \vdots & & & \vdots \\ 1 & \omega^{(N-1)} & \cdots & \omega^{(N-1)\,(N-1)} \end{pmatrix} \begin{pmatrix} x(0) \\ x(1) \\ \vdots \\ x(n) \\ \vdots \\ x(N-1) \end{pmatrix}$$

avec $\omega = \exp^{-2\pi I/N}$

Pseudo-code naïf

```
pour n =0 a N-1
  x(n) = 0
  pour k=0 a N-1
    X(k) = X(k) + x(k) * exp ( - 2*PI*n*k/N )
  fin pour
fin pour
```

Nombre d'opérations : $N \times N \times 8$

Transformée de Fourier rapide

Décomposition en deux sommes (suivant les échantillons pairs et impairs)

$$\hat{x}(k) = \sum_{n=0}^{N/2-1} x_{2n} \exp^{\frac{-2\pi I}{N} k(2n)} + \sum_{n=0}^{N/2-1} x_{2n+1} \exp^{\frac{-2\pi I}{N} k(2n+1)}$$

qui s'écrit aussi

$$\hat{x}(k) = \sum_{n=0}^{N/2-1} x_{2n} \exp^{\frac{-2\pi I}{N/2} kn} + \exp^{\frac{-2\pi I}{N} k} \sum_{n=0}^{N/2-1} x_{2n+1} \exp^{\frac{-2\pi I}{N/2} kn}$$

donc si $k \leq N/2 - 1$, par périodicité :

$$\hat{x}(k) = \hat{x}^p(k) + \exp^{-2\pi I \frac{k}{N}} \; \hat{x}^i(k)$$

$$\hat{x}(k + N/2) = \hat{x}^p(k) - \exp^{-2\pi I \frac{k}{N}} \; \hat{x}^i(k)$$

Complexité en $N \log_2 N$ si N est une puissance de 2 (Cooley-Tuckey, 1965).

Pseudo-code de la FFT rapide

```
pour k de 0 a N/2-1
    x_pair(k) = x(2*k)
    x_impair(k) = x(2*k+1)
fin pour

si N/2 = 1
    X_pair(0) = x_pair(0)
    X_impair(0) = x_impair(0)
sinon
    X_pair = fft(x_pair)
    X_impair = fft(x_impair)
fin si

pour k = 0 a N/2-1
    tmp = X_impair(k) * exp(-2*pi*k/N)
    X(k) = X_pair(k) + tmp
    X(N/2+k) = X_pair(k) - tmp
fin pour
```

Transformée de Fourier inverse

Transformée directe :

$$\hat{x}(k) = \sum_{n=0}^{N-1} x_n \exp^{-2\pi I \frac{kn}{N}}$$

Transformée inverse :

$$x_n = \frac{1}{N} \sum_{n=0}^{N-1} \hat{x}(k) \exp^{2\pi I \frac{kn}{N}}$$

La transformée inverse a la même complexité arithmétique que la transformée directe.

Bibliothèque FFTW

`http://www.fftw.org`
(Frigo et Johnson, 1998)

Caractéristiques

- Calcul optimal pour des puissances de 2 (basé sur des variantes de l'algorithme de Cooley-Tukey)
- Calcul efficace pour des puissances de nombres premiers (basé sur les algorithmes de Rader ou Bluestein)
- Bibliothèque portable (implémentée sur la plupart des supercalculateurs)
- Bibliothèque opensource (licence GNU)
- Écrite en `C`. Existent des interfaces pour de nombreux langages (`Fortran`, `Python`, etc.)

Usage

```c
#include<fftw3.h>
int main(void)
{
  int N;
  fftw_complex *in, *out;
  fftw_plan my_plan;

  in = (fftw_complex*) fftw_malloc(sizeof(fftw_complex)*N);
  out = (fftw_complex*) fftw_malloc(sizeof(fftw_complex)*N);
  my_plan = fftw_plan_dft_1d(N, in, out, FFTW_FORWARD, FFTW_ESTIMATE
      );

  fftw_execute(my_plan); /* repeat as needed */

  fftw_destroy_plan(my_plan);
  fftw_free(in);
  fftw_free(out);

  return 0;
}
```

(*tiré de la documentation de FFTW3.3*)

Exemple

Transformée de la fonction :

$$f(x) = A\cos(\omega x) + B\sin(\alpha\,\omega x)$$

pour $x \in [0; 2\pi\, L]$

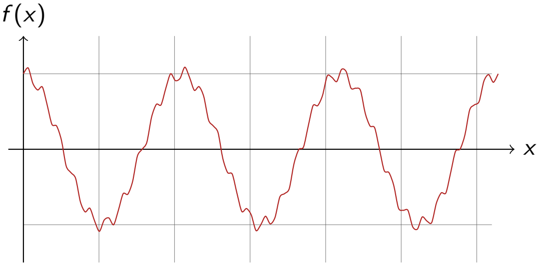
Fréquences :

- Mode pair de fréquence ω et d'amplitude $A/2$
- Mode impair de fréquence $\alpha\,\omega$ et d'amplitude $B/2$

Application :

$$A = 1, B = 0.1, \alpha = 10., \omega = 3.0, L = 1.0$$

Discrétisation sur $N = 16384 = 2^{14}$ modes.



Faire un aller/retour entre l'espace physique et l'espace spectral et retrouver les fréquences et amplitudes des modes pairs et impairs.

Fonction discrète

```fortran
subroutine function_periodic_cos_sin(N,x,kx,ain)

  ! cos(omega x)+0.1*sin(10*omega*x)
  integer, intent(in) :: N
  real(dp), intent(out) :: x(0:N-1),kx(0:N-1)
  complex(dp), intent(out) :: ain(0:N-1)

  ! local
  real(dp) :: Lx,omega
  real(dp), parameter :: PI = 4.0_dp*ATAN(1.0_dp)
  real(dp) :: dx,dkx
  integer :: i

  ! box size
  Lx = 2.0_dp*PI*1.0_dp
  ! spatial discretization
  dx = Lx/real(N)
  ! discrete function
  omega = 3.0_dp
  ! exclude periodic point
  do i=0,N-1
    x(i) = real(i)*dx
    ain(i) = cos(omega*x(i))+0.10*sin(10.*omega*x(i))
  end do
```

```fortran
    ! discrete modes
    dkx = 2.0_dp*PI/Lx
    ! hermitian symmetry
    do i=0,N/2
      kx(i) = i*dkx
    end do
    do i=N/2+1,N-1
      kx(i) = (N-i)*dkx
    end do

  end subroutine function_periodic_cos_sin
```

FFT naïve

```fortran
subroutine naive_fft(N,ain,aout,key)

  ! naive fft
  integer, intent(in) :: N
  real(dp), intent(in) :: key
  complex(dp), intent(in) :: ain(0:N-1)
  complex(dp), intent(out) :: aout(0:N-1)

  ! local
  real(dp), parameter :: PI = 4.0_dp*ATAN(1.0_dp)
  complex(dp) :: theta
  integer :: i,k

  ! check direction
  if (abs(key) /=1 ) then
    write(*,*) 'naive_fft :: wrong parameter. key should be -1 :
         forward and 1 : backward'
    stop 1
  end if
```

```fortran
    ! discrete function
    theta=CMPLX(0.0_dp,key*2.0_dp*PI/N,kind=dp)

    ! naive loop
    do k=0,N-1
      aout(k) = 0.0_dp
      do i=0,N-1
        aout(k)=aout(k)+ain(i)*EXP(theta*k*i)
      end do
    end do

  end subroutine naive_fft
```

FFTW3 (appel)

```fortran
module FFTW3
  ! provides Fortran type names constants which corresponds to C
      type
  use, intrinsic :: iso_c_binding
  ! provides constants for FFTW3 usage - include file located in /
      usr/include
  include 'fftw3.f03'
end module FFTW3
```

FFTW3 (déclaration des tableaux)

```
program fftw_1d

  ! FFTW
  use FFTW3
  ! double precision
  use mTypes, only: dp

  implicit none
  ! define Pi
  real(dp), parameter :: PI = 4.0_dp*ATAN(1.0_dp)
  integer :: i,N
  complex(C_DOUBLE_COMPLEX), dimension(:), allocatable :: ain
  complex(C_DOUBLE_COMPLEX), dimension(:), allocatable :: aout,aini
  real(dp), dimension(:), allocatable :: x,kx
  type(C_PTR) :: plan,plani
  real(C_DOUBLE) :: fact

  ! input data
  N = 16384

  ! allocate
  allocate(x(0:N-1))
  allocate(kx(0:N-1))
  allocate(ain(0:N-1))
  allocate(aini(0:N-1))
  allocate(aout(0:N-1))
```

FFTW3 (initialisation et normalisation)

```
! make plan forward and backward
ain = 1
aout = 1
aini = 1
plan = fftw_plan_dft_1d(N, ain, aout, FFTW_FORWARD, FFTW_ESTIMATE)
plani = fftw_plan_dft_1d(N, aout, aini, FFTW_BACKWARD,
     FFTW_ESTIMATE)
! normalization
fact = 1.0_dp/N

! case
call function_periodic_cos_sin(N,x,kx,ain)

write(*,*) "Function in physical space and frequencies"
do i = 0,N-1
  write(10,'(I5, 4(1X,E25.16))') i,x(i),ain(i),kx(i)
end do
```

FFTW3 (directe et inverse)

```
! execute forward dft
call fftw_execute_dft(plan, ain, aout)

write(*,*) "Fourier coefficient after forward FFT"
do i = 0,N-1
  write(11,'(I5, 3(1X,E25.16))') i,kx(i),aout(i)*fact
end do

! inverse transform
call fftw_execute_dft(plani, aout, aini)

write(*,*) "Function after forward-backward FFT"
do i = 0,N-1
  write(12,'(I5, 5(1X,E25.16))') i,x(i),ain(i),aini(i)*fact
end do
```

FFTW3 (libération de la mémoire)

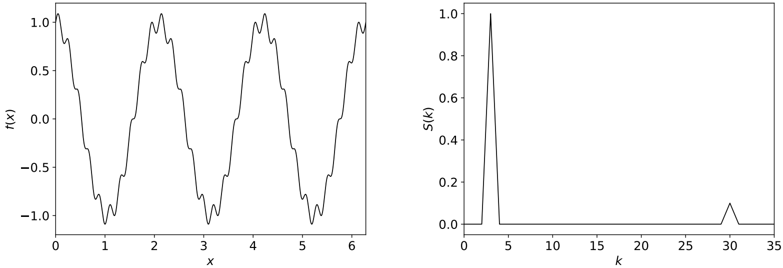
```
! destroy plans
call fftw_destroy_plan(plan)
call fftw_destroy_plan(plani)

! deallocate
deallocate(x)
deallocate(ain)
deallocate(aini)
deallocate(aout)
```

...

```
end program fftw_1d
```

Résultat

Normalisation en $1/N$



On retrouve bien les fréquences $k_A = 3$ et $k_B = 30$ avec leurs amplitudes respectives, $A = 1$ et $B = 0.1$.

Temps de calcul

Algorithme naïf :

```
real  0m27,094s
user  0m27,080s
sys 0m0,012s
```

FFTW3 :

```
real  0m0,267s
user  0m0,236s
sys 0m0,020s
```

Références

- `http://www.fftw.org`
- `La transformée de Fourier discrète`