3
The discrete Fourier transform

3.1 Introduction

Let $f(t)$ be a function describing a physical process in the *time domain*. An *equivalent* representation of the process, $F(\nu)$, is possible in the *frequency domain*, and the two representations are connected by means of the direct and inverse Fourier transforms:

$$F(\nu) = \int_{-\infty}^{\infty} f(t)\, e^{2\pi i\nu t} dt \tag{3.1}$$

$$f(t) = \int_{-\infty}^{\infty} F(\nu)\, e^{-2\pi i\nu t} d\nu. \tag{3.2}$$

In the most common situations, the function $f(t)$ is sampled at a finite number of time arguments. Suppose that we have N consecutive sampled values:

$$f_j \equiv f(t_j),$$

with

$$t_j \equiv (j-1)\Delta, \quad j = 1, 2, \ldots, N, \tag{3.3}$$

where Δ denotes the *sampling interval*.

We consider the N discrete frequencies

$$\nu_k = \frac{k-1}{N-1}\frac{1}{\Delta}, \quad k = 1, 2, \ldots, N,$$

and approximate the integral (3.1) by a discrete sum:

$$F(\nu_k) \simeq \sum_{j=1}^{N} f_j \, e^{2\pi i \nu_k t_j} \Delta = \Delta \sum_{j=1}^{N} f_j \exp\left[2\pi i \frac{(j-1)(k-1)}{N-1}\right].$$

The last sum is called the *discrete Fourier transform* of the N values f_j:

$$F_k = \sum_{j=1}^{N} f_j \exp\left[2\pi i \frac{(j-1)(k-1)}{N-1}\right], \quad k = 1, 2, \ldots, N. \tag{3.4}$$

The discrete Fourier transform maps N complex numbers, f_j, into N complex numbers, F_k.

Having the N values F_k of the discrete Fourier transform, the N original values f_j can be recovered by the *discrete inverse Fourier transform*:

$$f_j = \frac{1}{N} \sum_{k=1}^{N} F_k \exp\left[-2\pi i \frac{(j-1)(k-1)}{N-1}\right]. \tag{3.5}$$

It is obvious that a routine designed to calculate the discrete Fourier transform can calculate the inverse transform, too.

There are two important particular cases of the Fourier transform. If the transformed function $f(t)$ vanishes at the origin ($f(0) \equiv f_0 = 0$), than the basis functions should behave similarly and the natural transform in this case is the *sine transform*:

$$F_k = \sum_{j=1}^{N} f_j \sin\left[\frac{(j-1)(k-1)}{N-1}\pi\right], \quad k = 1, 2, \ldots, N. \tag{3.6}$$

If the *derivative* of the transformed function $f(t)$ vanishes at the origin, than the basis functions should show the same behavior, too, and the natural transform in this case is the *cosine transform*:

$$F_k = \sum_{j=2}^{N-1} f_j \cos\left[\frac{(j-1)(k-1)}{N-1}\pi\right] + \frac{1}{2}\left[f_1 + (-1)^{k-1} f_N\right], \quad k = 1, 2, \ldots, N. \tag{3.7}$$

The cosine transform is its own inverse,

$$f_j = \sum_{k=2}^{N-1} F_k \cos\left[\frac{(j-1)(k-1)}{N-1}\pi\right] + \frac{1}{2}\left[F_1 + (-1)^{j-1} F_N\right], \quad j = 1, 2, \ldots, N, \tag{3.8}$$

and the transformed values F_k appear in the Fourier expansion of f_j as weights of the cosine functions with the frequencies

$$\nu_k = \frac{k-1}{N-1}\frac{1}{2\Delta}, \quad k = 1, 2, \ldots, N. \tag{3.9}$$

We remind that $t_j = (j-1)\Delta$.

The straightforward implementation of the discrete cosine transform (3.7)-(3.9) is:

```
!==============================================================================
subroutine FTcos0(f,ft,n)
!------------------------------------------------------------------------------
!  Straightforward implementation of the discrete cosine transform
!
!  f  - input data
!  ft - transformed data
!  n  - no. of data values

!  Frequencies corresponding to the transformed data:
!     freq(k) = (k-1) / (2.d0 * dt * (n-1)), k=1,n
!------------------------------------------------------------------------------
   implicit real(8) (a-h,o-z)

   real(8), parameter :: pi = 3.1415926535897932385d0

   real(8) f(n), ft(n)

   isgn = 1
   fact = pi / (n-1)
   do k = 1,n
      theta = (k-1) * fact

      sum = 0.d0
      do j = 2,n-1
         sum = sum + f(j) * cos((j-1)*theta)
      end do

      ft(k) = sum + 0.5d0 * (f(1) + isgn*f(n))
      isgn = -isgn
   end do
end
```

Due to the repeated evaluations of the cosine function, the above implementation is highly inefficient and the execution time becomes critical when the number of observations N exceeds 10^5.

3.2 Efficient discrete Fourier transform

We consider only the sum in expression (3.7) of the cosine transform F_k,

$$G_k = \sum_{j=2}^{N-1} f_j \cos\left[(j-1)\theta_k^0\right], \tag{3.10}$$

with

$$\theta_k^0 = \frac{k-1}{N-1}\pi, \tag{3.11}$$

and split it in even-j and odd-j terms:

$$G_k = \sum_{j=2,4} f_j \cos\left[(j-1)\theta_k^0\right] + \sum_{j=3,5} f_j \cos\left[(j-1)\theta_k^0\right].$$

The addition formula $\cos[(j-1)\theta] = \cos\theta\cos(j\theta) + \sin\theta\sin(j\theta)$ is used to expand the first sum, and the index j is changed to $j+1$ in the second one:

$$G_k = \sum_{j=2,4} f_j \left[C_k \cos(j\theta_k^0) + S_k \sin(j\theta_k^0)\right] + \sum_{j=2,4} f_{j+1} \cos(j\theta_k^0),$$

where

$$C_k = \cos\theta_k^0, \quad S_k = \sin\theta_k^0.$$

Now, we replace the index j by $2j$ in both sums and group together the sine and cosine functions with the same argument:

$$G_k = \sum_{j=1}^{M} \left[U_j \cos(j\theta_k) + V_j \sin(j\theta_k)\right]. \tag{3.12}$$

Here we use the notations:

$$U_j = C_k f_{2j} + f_{2j+1}, \quad V_j = S_k f_{2j}, \tag{3.13}$$

$$\theta_k = 2\theta_k^0, \tag{3.14}$$

and

$$M = \left[\frac{N-1}{2}\right] \tag{3.15}$$

is the integer part of $(N-1)/2$.

Next, the arguments of the trigonometric functions will be modified in the even-j moiety. To this end, G_k is split first into odd-j and even-j terms, and in the odd-j terms the argument $j\theta_k$ is replaced by $(j+1)\theta_k - \theta_k$:

$$
\begin{aligned}
G_k &= \sum_{j=1,3}^{M \text{ or } M-1} \{U_j \cos[(j+1)\theta_k - \theta_k] + V_j \sin[(j+1)\theta_k - \theta_k]\} \\
&+ \sum_{j=2,4}^{M \text{ or } M-1} [U_j \cos(j\theta_k) + V_j \sin(j\theta_k)] \,.
\end{aligned}
$$

In the first sum, addition formulas for the modified cosine and sine functions are applied:

$$
\begin{aligned}
G_k &= \sum_{j=1,3}^{M \text{ or } M-1} U_j \{\cos[(j+1)\theta_k] \cos\theta_k + \sin[(j+1)\theta_k] \sin\theta_k\} \\
&+ \sum_{j=1,3}^{M \text{ or } M-1} V_j \{\sin[(j+1)\theta_k] \cos\theta_k - \cos[(j+1)\theta_k] \sin\theta_k\} \\
&+ \sum_{j=2,4}^{M \text{ or } M-1} [U_j \cos(j\theta_k) + V_j \sin(j\theta_k)] \,.
\end{aligned}
$$

Now, all arguments imply even multiples of θ_k. In order to be able to group the trigonometric functions, $j+1$ is replace by $2j$ in the first two sums, and j by $2j$ in the last one:

$$
\begin{aligned}
G_k &= \sum_{j=1} U_{2j-1} \{\cos(2j\theta_k) \cos\theta_k + \sin(2j\theta_k) \sin\theta_k\} \\
&+ \sum_{j=1} V_{2j-1} \{\sin(2j\theta_k) \cos\theta_k - \cos(2j\theta_k) \sin\theta_k\} \\
&+ \sum_{j=1} [U_{2j} \cos(2j\theta_k) + V_{2j} \sin(2j\theta_k)] \,.
\end{aligned}
$$

Collecting the terms and making the notations:

$$
\begin{aligned}
U_j^{(1)} &= U_{2j-1} \cos\theta_k - V_{2j-1} \sin\theta_k + U_{2j} \\
V_j^{(1)} &= U_{2j-1} \sin\theta_k + V_{2j-1} \cos\theta_k + V_{2j},
\end{aligned}
\tag{3.16}
$$

leads to the following form of G_k:

$$G_k = \sum_{j=1}^{[(M+1)/2]} \left[U_j^{(1)} \cos(2j\theta_k) + V_j^{(1)} \sin(2j\theta_k) \right], \tag{3.17}$$

where $[(M+1)/2]$ is the integer part of $(M+1)/2$. For odd M, the new coefficients, $U_j^{(1)}$ and $V_j^{(1)}$, imply for $j = [(M+1)/2]$ the old coefficients, U_{M+1} and V_{M+1}, which do not actually exist. These terms should be interpreted as zero.

By comparing expression (3.17) of G_k with its original form (3.12), it can be seen that at the expense of having to evaluate a new set of coefficients ($U_j^{(1)}$ and $V_j^{(1)}$ for $j = 1, [(M+1)/2]$), the number of summed terms was condensed by a factor of almost 2 (according to the parity of M). The corresponding sine and cosine functions may be easily evaluated from the duplication formulas:

$$\begin{aligned} \sin(2j\theta_k) &= 2\sin(j\theta_k)\cos(j\theta_k), \\ \cos(2j\theta_k) &= 2\cos^2(j\theta_k) - 1. \end{aligned} \tag{3.18}$$

A careful analysis shows that if such a *condensation* procedure is repeated N_{cond} times, where

$$N_{\text{cond}} = [\log_2(2M-1)], \tag{3.19}$$

then just two terms remain and one gets the final expression:

$$G_k = U_1^{(N_{\text{cond}})} \cos(2^{N_{\text{cond}}}\theta_k) + V_1^{(N_{\text{cond}})} \sin(2^{N_{\text{cond}}}\theta_k), \tag{3.20}$$

from which G_k can be easily evaluated.

By this condensation procedure, the number of sines and cosines evaluations is reduced from M to N_{cond}, or, for instance, from 1000 to 10. Actually, one just has to evaluate before starting the condensation procedure $\cos\theta_k$ and $\sin\theta_k$ for $j = 1, M$, the rest of the sine and cosine functions forming a sequence which can be readily generated by the duplication formulas (3.18).

The routine listed below represents the exact implementation of the presented algorithm.

```
!=============================================================================
subroutine FTcos(f,ft,n)
!-----------------------------------------------------------------------------
!  Calculates the cosine transform of an array f of n real values.
!  The transformed data is returned in array ft.
!-----------------------------------------------------------------------------
   implicit real(8) (a-h,o-z)
```

```
   parameter (pi = 3.1415926535897932385d0)

   real(8) f(n), ft(n)
   real(8), allocatable :: u(:), v(:)

   n2 = (n-1)/2
   allocate(u(n2+1), v(n2+1))

   ncond = int(log(2.d0*n2-1.d0)/log(2.d0))

   isgn = 1
   fact = pi / (n-1)
   fn = f(n); f(n) = 0.d0
   do k = 1,n
      theta = (k-1) * fact
      c = cos(theta); s = sin(theta)

      do j = 1,n2
         j2 = 2 * j
         u(j) = c * f(j2) + f(j2+1)
         v(j) = s * f(j2)
      end do

      s = 2.d0 * s * c
      c = 2.d0 * c * c - 1.d0
      nterm = n2
      do icond = 1,ncond
         u(nterm+1) = 0.d0; v(nterm+1) = 0.d0
         nterm = int((nterm+1)/2)
         do j = 1,nterm
            j2 = 2*j - 1
            utmp = c * u(j2) - s * v(j2) + u(j2+1)
            v(j) = s * u(j2) + c * v(j2) + v(j2+1)
            u(j) = utmp
         end do
         s = 2.d0 * s * c
         c = 2.d0 * c * c - 1.d0
      end do
      ft(k) = c * u(1) + s * v(1) + 0.5d0 * (f(1) + isgn*fn)
      isgn = -isgn
   end do

   deallocate(u,v)
end
```

Bibliografie

[1] Aubanel, E.E. and Oldham, K.B.., "Fourier smoothing without the fast Fourier transform", *Byte*, February 1985, p. 207.

[2] W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P Flannery, *Numerical Recipes in C, Second Edition* (Cambridge University Press, Cambridge, 1992).