Efficient convolution using the Fast Fourier Transform, Application in C++

Jeremy Fix

May 30, 2011

Contents

1 Introduction

Convolution products are often encountered in image processing but also in other works such as evaluating a convolutive neural network. A naive implementation of a convolution product of signals of size N involves an order of N^2 operations. An efficient implementation of convolution product can be designed by making use of the Fast Fourier Transform and mathematical properties linking convolution products and Fourier transforms. This can boil down the complexity of computing a convolution product to an order of $N \log N$ operations. In this document, we briefly introduce the convolution product, Fourier transform, the use of Fourier transforms for performing convolution products and sample codes as well as benchmarks showing the performances of two libraries : GSL and FFTW. All the benchmarks were done on an Intel(R) Core(TM) i5 CPU with GSL 1.13, FFTW 3.2.2 and gcc 4.4.4 under Fedora 10.04.

In the following we need to divide integers by two, sometimes takes its opposite. When writing $N/2$, we always consider it to be the largest integer smaller or equal to $N/2$ (e.g $3/2 = 1$). When writing $-N/2$, it is always the opposite of $N/2$ and not $-N$ divided by 2 (e.g. $-3/2 = -1$). Given this, the following results are used in the following: $N - \frac{N-1}{2} = \frac{N}{2} + 1, N - \frac{N+1}{2} = \frac{N}{2}$.

1.1 Convolution product : linear and circular

1.1.1 Definition

Consider two signals $f[n]$ of size N and $g[m]$ of size M. Suppose f is defined on $[0, N-1]$ and g is defined on $[0, M-1]$[1]. We use indices starting from 0 to be coherent with indices of C arrays. The discrete convolution product of these two sequences is defined as :

$$\forall k, h[k] = (f * g)[k] = \sum_{i=-\infty}^{\infty} f_\infty[i].g_\infty[k-i] = (g * f)[k] \tag{1}$$

where f_∞ and g_∞ are defined by :

$$\begin{aligned} f_\infty[n] &= \begin{cases} f[n] & \text{if} \quad 0 \leq n \leq N-1 \\ 0 & \text{otheriwse} \end{cases} \\ g_\infty[m] &= \begin{cases} g[m] & \text{if} \quad 0 \leq m \leq M-1 \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

These signals correspond to the original signal f and g to which we added zeros on the left and right up to an infinite horizon. Obviously, the signals having a finite horizon (padded zeros up-to and starting from certain indices), the convolution is null up-to and starting from some indices (see fig. 1). Namely, the full convolution has only at most $N + M - 1$ non zero values :

$$h[k] = (f * g)[k] = \begin{cases} \sum_{i=\max(0,k-M+1)}^{\min(N-1,k)} f[i]g[k-i] & \text{if} \quad 0 \leq k \leq N+M-2 \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

Also, some of the products involve padded values of f while others not. This leads to define, as in Matlab, three different results (see fig. 1):

- *Full* : The result with all the non-zero elements, of size $N + M - 1$
- *Same* : The result with the central components, of the same size N of f, all the indices for which the central element of g is multiplied by a non-padded value of f
- *Valid* : The result with the components that do not involve any padded zero of f, of size $N - M + 1$

[1] Be careful, we use mathematical, also used for C arrays, indices running from 0 to $N-1$ for an array of size N, contrary to Matlab which uses indices running from 1 up-to N.

We get the following expressions for the three result types :

$$\textbf{Full}\ , \forall k \in [0, N+M-2], \quad h_f[k] = \sum_{i=\max(0,k-M+1)}^{\min(N-1,k)} f[i]g[k-i] \tag{3}$$

$$\textbf{Same}\ , \forall k \in [0, N-1], \qquad h_s[k] = h_f[k+\frac{M}{2}] = \sum_{i=\max(0,k-\frac{M-1}{2})}^{\min(N-1,k+\frac{M}{2})} f[i]g[k+M/2-i] \tag{4}$$

$$\textbf{Valid}\ , \forall k \in [0, N-M], \qquad h_s[k] = h_f[k+M-1] = \sum_{i=k}^{k+M-1} f[i]g[k+M-1-i] \tag{5}$$

This corresponds to the **linear convolution**. As we saw previously, we may need to pad the f signal with zeros when required values of the signal are undefined. One may also define the **circular convolution**, denoted $\circledast_P$, which, instead of padding the signals with zeros, wrap them arround, i.e. considering them to be periodic. This means that when an undefined value of the signal is required, e.g. $f[-1]$, we use the wrapped around value, i.e. $f[-1 \bmod N] = f[N-2]$. The circular convolution modulo P is defined formally as :

$$\begin{aligned}\forall k, h_P[k] = (f \circledast_P g)[k] &= \sum_{i=0}^{P-1} f_P[i].g_P[k-i] \\ &= (g \circledast_P f)[k]\end{aligned}$$

with :

$$\begin{aligned}\forall i, f_P[i] &= \sum_{p=-\infty}^{\infty} f_\infty[i+pP] \\ \forall i, f_\infty[i] &= \left\{\begin{array}{ccc} f[i] & \text{if} & 0 \leq i \leq N-1 \\ 0 & \text{otherwise} & \end{array}\right. \\ \forall i, g_P[i] &= \sum_{p=-\infty}^{\infty} g_\infty[i+pP] \\ \forall i, g_\infty[i] &= \left\{\begin{array}{ccc} g[i] & \text{if} & 0 \leq i \leq M-1 \\ 0 & \text{otherwise} & \end{array}\right.\end{aligned}$$

Examples of circular convolution, modulo N, the size of f are shown on figure 2 with $N=5, P=5$ and $M=3$ or $M=4$. Since the f_P and g_P signals are periodic of period P, the circular convolution modulo P is periodic of period P. Note that when $P \geq N$, f_P is nothing more than repetitions of f, padded with $P-N$ zeros. Let's rewrite also a particular case, when $P = N$ and $M \leq N$ which is usual in image processing. In this case, we have :

$$\begin{aligned}\forall i \in [0, N-1], f_N[i] &= f[i] \\ \forall i \in [-N+1, N-1], g_N[i] &= \left\{\begin{array}{ccc} g[i+N] & \text{if} & -N+1 \leq i \leq -(N-M)-1 \\ 0 & \text{if} & -(N-M) \leq i \leq -1 \\ g[i] & \text{if} & 0 \leq i \leq M-1 \\ 0 & \text{if} & M \leq i \leq N-1 \end{array}\right.\end{aligned} \tag{6}$$

And, if we compute the N terms of the circular convolution, shifted by $\frac{M}{2}$:

$$\begin{aligned}\forall k \in [0, N-1], h_N[k+\frac{M}{2}] &= (f \circledast_N g)[k+\frac{M}{2}] \\ &= (g \circledast_N f)[k+\frac{M}{2}] \\ &= \sum_{i=0}^{N-1} g_P[i] f_P[k+\frac{M}{2}-i] \\ &= \sum_{i=0}^{M-1} g[i] f_P[k+\frac{M}{2}-i]\end{aligned}$$

Since we consider the circular convolution modulo N, the size of f, then $\forall i, f_N[i] = f[i \bmod N]$ and finally :

$$\begin{aligned}\forall k \in [0, N-1], h_N[k+\frac{M}{2}] &= \sum_{i=0}^{M-1} g[i] f_P[k+\frac{M}{2}-i] \\ &= \sum_{i=0}^{M-1} g[i] f[k+\frac{M}{2}-i \bmod N] \qquad (7)\end{aligned}$$

We can observe that there is a link between the linear convolutions as defined by equations 3, 4, 5, and the circular convolution we defined previously. We will see the importance of this link latter on, when we will see that a circular convolution can be efficiently computed with the Fourier Transform. Given the link between linear and circular convolution, we will also be able to compute efficiently linear convolutions.

1.1.2 Linear convolutions as particular cases of circular convolution

Consider two finite sequences f and g of respectively size N and M, then we have the following links between circular and linear convolutions :

Full $, \forall P \geq N+M-1, \forall k \in [0, N+M-2], \quad h_f[k] = \sum_{i=\max(0,k-M+1)}^{\min(N-1,k)} f[i]g[k-i] = (f \circledast_P g)[k]$ (8)

Same $, \forall P \geq N+\frac{M}{2}, \forall k \in [0, N-1], \quad h_s[k] = \sum_{i=\max(0,k-\frac{M-1}{2})}^{\min(N-1,k+\frac{M}{2})} f[i]g[k+M/2-i] = (f \circledast_P g)[k+\frac{M}{2}]$ (9)

Valid $, \forall P \geq N, \forall k \in [0, N-M], \quad h_v[k] = \sum_{i=k}^{k+M-1} f[i]g[k+M-1-i] = (f \circledast_P g)[k+M-1]$ (10)

Full linear convolution:

Let's begin by showing equation 8. First, we observe that $P \geq N+M-1 \geq N$, therefore f_P is just a periodic signal of period P in which one period is made of f padded with $P-N$ zeros. We also have :

$$\forall i \in [0, P-1], f_P[i] = \begin{cases} f[i] & \text{if} \quad 0 \leq i \leq N-1 \\ 0 & \text{otherwise} \end{cases}$$

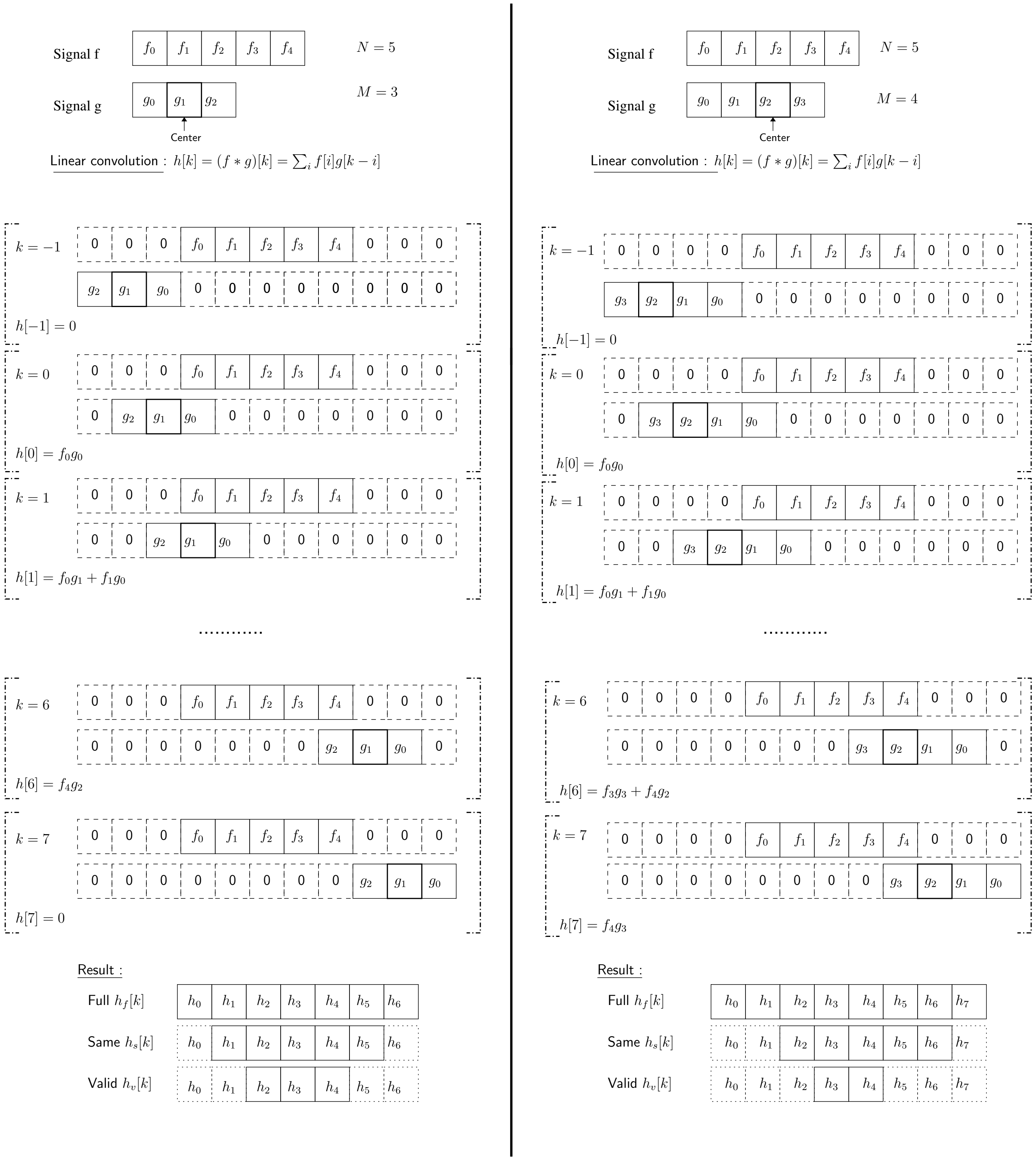


Figure 1: Schematic illustration of the linear convolution for an odd and even filter size

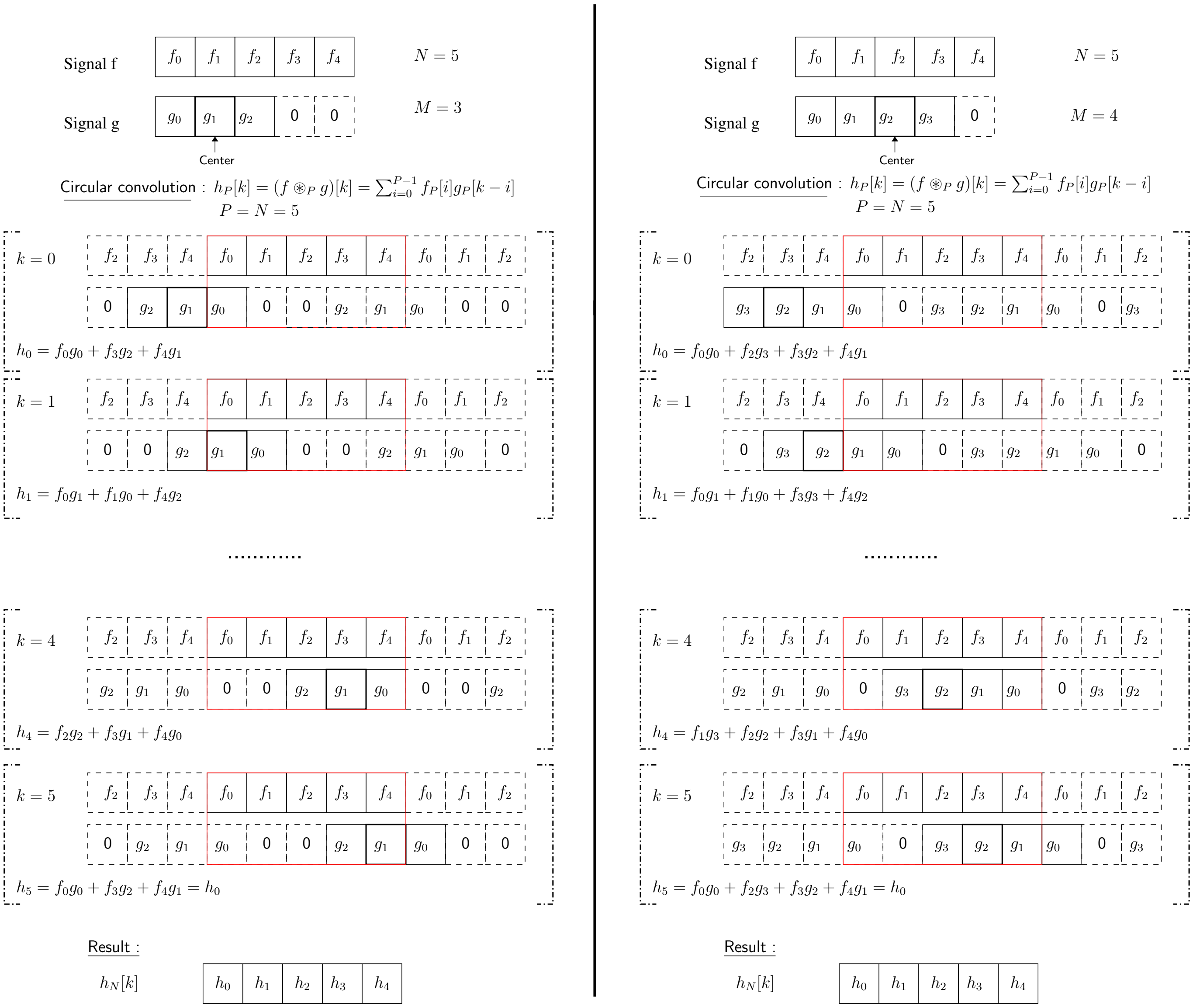

Signal f
f_0 f_1 f_2 f_3 f_4
$N = 5$
Signal g
g_0 g_1 g_2 0 0
$M = 3$
Center
Circular convolution : $h_P[k] = (f \circledast_P g)[k] = \sum_{i=0}^{P-1} f_P[i] g_P[k-i]$
$P = N = 5$
$k = 0$
$h_0 = f_0g_0 + f_3g_2 + f_4g_1$
$k = 1$
$h_1 = f_0g_1 + f_1g_0 + f_4g_2$
....
$k = 4$
$h_4 = f_2g_2 + f_3g_1 + f_4g_0$
$k = 5$
$h_5 = f_0g_0 + f_3g_2 + f_4g_1 = h_0$
Result :
$h_N[k]$
h_0 h_1 h_2 h_3 h_4
Signal f
f_0 f_1 f_2 f_3 f_4
$N = 5$
Signal g
g_0 g_1 g_2 g_3 0
$M = 4$
Center
Circular convolution : $h_P[k] = (f \circledast_P g)[k] = \sum_{i=0}^{P-1} f_P[i] g_P[k-i]$
$P = N = 5$
$k = 0$
$h_0 = f_0g_0 + f_2g_3 + f_3g_2 + f_4g_1$
$k = 1$
$h_1 = f_0g_1 + f_1g_0 + f_3g_3 + f_4g_2$
....
$k = 4$
$h_4 = f_1g_3 + f_2g_2 + f_3g_1 + f_4g_0$
$k = 5$
$h_5 = f_0g_0 + f_2g_3 + f_3g_2 + f_4g_1 = h_0$
Result :
$h_N[k]$
h_0 h_1 h_2 h_3 h_4

Figure 2: Schematic illustration of the circular convolution for an odd and even filter size

We can therefore restrict the sum of the circular convolution on the indexes $[0, N-1]$:

$$\begin{aligned}\forall k \in [0, N+M-2], (f \circledast_P g)[k] &= \sum_{i=0}^{P-1} f_P[i] g_P[k-i] \\ &= \sum_{i=0}^{N-1} f[i] g_P[k-i]\end{aligned}$$

Since $P \geq N+M-1 \geq M$, the g_P signal is also periodic of period P in which one period is the signal g padded with $P-M$ zeros. We also have :

$$\forall k \in [0, N+M-2], \forall i \in [0, N-1], -(P-M) \leq -N+1 \leq k-i \leq P-1$$

Therefore, when the index $k-i$ of g_P is negative, it is never below $-(P-M)$ and, by definition, $\forall i \in [-(P-M), 0], g_P[i] = 0$. Also, by definition, the signal g_P is null when $M \leq k-i \leq P-1$ ($\forall k \in [0, N+M-2], \forall i \in [0, N-1], k-i \leq P-1$). Therefore the sum of the circular convolution can be restricted on the indexes i where $0 \leq k-i \leq M-1$, i.e. $i \in [k-M+1, k]$:

$$\begin{aligned}\forall k \in [0, N+M-2], (f \circledast_P g)[k] &= \sum_{i=0}^{N-1} f[i] g_P[k-i] \\ &= \sum_{i=\max(0,k-M+1)}^{\min(N-1,k)} f[i] g[k-i] \\ &= h_f[k]\end{aligned}$$

Same linear convolution:

To compute the *same* linear convolution, we may compute the *full* linear convolution and restrict the result to the N points of interest. But we can do even better by only padding with $\frac{M}{2}$ zeros and extracting the relevant part from the result. Let's suppose $P \geq N + \frac{M}{2}$. As before, since $P \geq N$, the sum on $i \in [0, P-1]$ can be restricted to $i \in [0, N-1]$ and replacing f_p with f :

$$\begin{aligned}\forall k \in [0, N-1], (f \circledast_P g)[k + \frac{M}{2}] &= \sum_{i=0}^{P-1} f_P[i] g_P[k + \frac{M}{2} - i] \\ &= \sum_{i=0}^{N-1} f[i] g_P[k + \frac{M}{2} - i] \qquad (11)\end{aligned}$$

Now, one has to remind the defintion of g_P. This signal is defined by chunking g_∞ (the signal g padded infinitely with zeros on the two sides) into pieces of size P that are summed. For some values of P (namely $P \geq M$) this simply means taking g and padding it with zeros, as it was the case before. Here we have $P \geq N + \frac{M}{2}$ and we are not ensured that $P \geq M$. In fact, if $N < \frac{M}{2}$ there are some values of P for which several chunks of g_∞ will contain non-zero values. To make the derivation easier, we consider two cases.

First, if $P \geq M$, then we can rewrite g_P as :

$$\forall i, g_P[i] = \begin{cases} g[i] & \text{if} \quad i \bmod P \leq M-1 \\ 0 & \text{otherwise} \end{cases} \qquad (12)$$

Therefore we can restrict the sum, in equation (11) over the indexes where $k + \frac{M}{2} - i \bmod P \leq M-1$. Let's now consider which parts of g_P are involved in the sum :

$$\begin{aligned}\forall k \in [0, N-1], \forall i \in [0, N-1], -(N-1) + \frac{M}{2} \leq & \quad k - i + \tfrac{M}{2} \quad \leq N - 1 + \frac{M}{2} \qquad (13) \\ M - P \leq & \quad k - i + \tfrac{M}{2} \quad \leq P - 1 \qquad (14)\end{aligned}$$

For the indexes i so that $M \leq k-i+\frac{M}{2} \leq P-1$, we know that $g_P[k-i+\frac{M}{2}] = 0$. For the negative indexes, we know that if $M-P \leq k-i+\frac{M}{2} \leq -1$, then $M \leq k-i+\frac{M}{2}+P \leq P-1$ and therefore, $g_P[k-i+\frac{M}{2}] = g_P[k-i+\frac{M}{2}+P] = 0$. Overall, we can restrict the sum on the indexes i, so that $0 \leq k-i+\frac{M}{2} \leq M-1$ and rewrite equation 11 as :

$$\begin{aligned}
\forall k \in [0, N-1], (f \circledast_P g)[k+\frac{M}{2}] &= \sum_{i=0}^{N-1} f[i] g_P[k+\frac{M}{2}-i] \\
&= \sum_{i=\max(0,k-\frac{M-1}{2})}^{\min(N-1,k+\frac{M}{2})} f[i] g[k+\frac{M}{2}-i] \qquad (15)\\
&= h_s[k] \qquad (16)
\end{aligned}$$

which is what we wanted to demonstrate.

Second, if $N+\frac{M}{2} \leq P < M$, then we still have :

$$\begin{aligned}
\forall k \in [0, N-1], (f \circledast_P g)[k+\frac{M}{2}] &= \sum_{i=0}^{P-1} f_P[i] g_P[k+\frac{M}{2}-i] \\
&= \sum_{i=0}^{N-1} f[i] g_P[k+\frac{M}{2}-i] \qquad (17)
\end{aligned}$$

Also, since $N+\frac{M}{2} < M$, then $N < \frac{M+1}{2}$ and $N-1 \leq \frac{M}{2} \Rightarrow N-1 = \min(N-1, k+\frac{M}{2}) \forall k \in [0, N-1]$. Therefore, we already get the upper bound which is a bit artificial. For the lower bound, we simply observe that $\forall k \in [0, N-1], k-\frac{M-1}{2} \leq N-1-\frac{M-1}{2} \leq \frac{M+1}{2}-1-\frac{M-1}{2} \leq 0$, therefore $\forall k \in [0, N-1], 0 = \max(0, k-\frac{M-1}{2})$.

$$\begin{aligned}
\forall k \in [0, N-1], (f \circledast_P g)[k+\frac{M}{2}] &= \sum_{i=0}^{N-1} f[i] g_P[k+\frac{M}{2}-i] \\
&= \sum_{i=\max(0,k-\frac{M-1}{2})}^{\min(N-1,k+\frac{M}{2})} f[i] g_P[k+\frac{M}{2}-i]
\end{aligned}$$

Now, what does the g_P signal looks like ? Given $P \geq N+\frac{M}{2}$, when we slice the g signal in slices of size P, the second slice contains at most $M-(N+M/2) = (M-1)/2-N$ non-zero components. Therefore, only at most the first $(M-1)/2-N$ of g_P are sums of more than one non-zero element of g. The others simply equal the corresponding component of g, in other words :

$$\forall k \in [\frac{M-1}{2}-N, P-1], g_P[k] = g[k]$$

The indexes of g_P are running in this domain :

$$\begin{aligned}
\forall k \in [0, N-1], \forall i \in [0, N-1], -N+1+\frac{M}{2} \leq \quad & k+\tfrac{M}{2}-i \quad \leq N-1+\frac{M}{2} \\
\frac{M-1}{2}-N \leq \quad & k+\tfrac{M}{2}-i \leq P-1
\end{aligned}$$

Therefore, we can replace g_P by g in the sum to get our result :

$$\begin{aligned}
\forall k \in [0, N-1], (f \circledast_P g)[k+\frac{M}{2}] &= \sum_{i=\max(0,k-\frac{M-1}{2})}^{\min(N-1,k+\frac{M}{2})} f[i] g_P[k+\frac{M}{2}-i] \\
&= \sum_{i=\max(0,k-\frac{M-1}{2})}^{\min(N-1,k+\frac{M}{2})} f[i] g[k+\frac{M}{2}-i] \\
&= h_s[k]
\end{aligned}$$

Valid linear convolution We finished with the easiest situation, the **valid** linear convolution. First, this convolution results in a non-empty result if and only if $M \leq N$. Let's suppose $P \geq N$. Then, the signal f_P and g_P are simply the signals f and g to which we added respectively $P-N$ and $P-M$ zeros :

$$\begin{aligned} \forall i \in [0, P-1], f_P[i] &= \begin{cases} f[i] & \text{if} \quad 0 \leq i \leq N-1 \\ 0 & \text{otherwise} \end{cases} \\ \forall i \in [0, P-1], g_P[i] &= \begin{cases} g[i] & \text{if} \quad 0 \leq i \leq M-1 \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

Therefore, the circular convolution modulo P reads :

$$\begin{aligned} \forall k \in [0, N-M], (f \circledast_P g)[k+M-1] &= \sum_{i=0}^{P-1} f_P[i] g_P[k+M-1-i] \\ &= \sum_{i=0}^{N-1} f[i] g_P[k+M-1-i] \end{aligned}$$

Since :

$$\begin{aligned} \forall k \in [0, N-M], \forall i \in [0, N-1], M-N \leq & \quad k+M-1-i & \leq N-1 \\ M-P \leq & \quad k+M-1-i & \leq P-1 \end{aligned}$$

we can restrict the sum on the indexes i such that $k+M-1-i \in [0, M-1]$, namely $i \in [k, k+M-1]$:

$$\begin{aligned} \forall k \in [0, N-M], (f \circledast_P g)[k+M-1] &= \sum_{i=0}^{N-1} f[i] g_P[k+M-1-i] \\ &= \sum_{k}^{k+M-1} f[i] g[k+M-1-i] \\ &= h_v[k] \end{aligned}$$

The linear and circular convolutions as defined before require roughly an order of $N \times M$ computations (slightly less for the linear convolutions). In the next section we introduce the Discrete Fourier Transform and its efficient implementations known as Fast Fourier Transforms which boils the complexity down to an order of $N \log(N)$ computations. We also show two technical points; the first one is that it is possible to compute two fourier transforms of real signals with a single call to the transform and that padding the signals with more than necessary zeros improve the performances. This last point comes from the fast that the FFT algorithms are particularly efficient for specific sizes of the signals.

1.2 Discrete Fourier Transform

1.2.1 Introduction

The Discrete Fourier Transform of a discrete signal $f[n]$ of length N is defined as :

$$FT[f][k] = \hat{f}[k] = \sum_{n=0}^{N-1} f[n] e^{-\frac{2i\pi kn}{N}}$$

The discrete fourier transform of a discrete signal of length N is periodic of period N :

$$\begin{aligned}
\forall p \in \mathbb{Z}, \hat{f}[k+pN] &= \sum_{n=0}^{N-1} f[n] e^{-\frac{2i\pi(k+pN)n}{N}} \\
&= \sum_{n=0}^{N-1} f[n] e^{-\frac{2i\pi kn}{N}} e^{-\frac{2i\pi pNn}{N}} \\
&= \sum_{n=0}^{N-1} f[n] e^{-\frac{2i\pi kn}{N}} e^{-2i\pi pn} \\
&= \hat{f}(k)
\end{aligned}$$

The inverse discrete fourier transform of a discrete frequency signal $\hat{f}[k]$ is defined as :

$$IFT[\hat{f}][n] = f[n] = \frac{1}{N} \sum_{k=0}^{N-1} \hat{f}[k] e^{\frac{2i\pi kn}{N}}$$

The scaling factor $\frac{1}{N}$ is required to get the true inverse. To check this, let's compute $IFT[FT[f]][n]$ to check it equals to $f[n]$:

$$\begin{aligned}
IFT[FT[f]][n] &= \frac{1}{N} \sum_{k=0}^{N-1} FT[f][k] e^{\frac{2i\pi kn}{N}} \\
&= \frac{1}{N} \sum_{k=0}^{N-1} \sum_{m=0}^{N-1} \left(f[m] e^{-\frac{2i\pi km}{N}} \right) e^{\frac{2i\pi kn}{N}} \\
&= \frac{1}{N} \sum_{k=0}^{N-1} \sum_{m=0}^{N-1} f[m] e^{\frac{2i\pi k(n-m)}{N}} \\
&= \frac{1}{N} \sum_{m=0}^{N-1} f[m] \sum_{k=0}^{N-1} e^{\frac{2i\pi k(n-m)}{N}}
\end{aligned}$$

The last sum equals N when $n \equiv m[N]$ and 0 otherwise since it is a geometric serie of common ratio $e^{\frac{2i\pi(n-m)}{N}}$:
If $n \equiv m[N]$ (m is in [0, N-1]): $\sum_{k=0}^{N-1} (e^{\frac{2i\pi(n-m)}{N}})^k = \sum_{k=0}^{N-1} 1 = N,$
Otherwise : $\sum_{k=0}^{N-1} (e^{\frac{2i\pi(n-m)}{N}})^k = \frac{1-(e^{\frac{2i\pi(n-m)}{N}})^N}{1-e^{\frac{2i\pi(n-m)}{N}}} = \frac{1-e^{2i\pi(n-m)}}{1-e^{\frac{2i\pi(n-m)}{N}}} = 0.$
From the previous result, we get :

$$\begin{aligned}
IFT[FT[f]][n] &= \frac{1}{N} \sum_{m=0}^{N-1} f[m] \sum_{k=0}^{N-1} e^{\frac{2i\pi k(n-m)}{N}} \\
&= \frac{1}{N} (N.f[n \bmod N]) = f[n \bmod N]
\end{aligned}$$

Therefore : $IFT\ o\ FT = identity$ when restricted on the horizon [0,N-1]. In fact, we have built a periodic signal of period N which is repetitions of f.

In terms of complexity, we need an order of N^2 operations to compute the DFT of a signal of length N (N products for each of the N components). In the following sections, we first show how we can compute a n-dimensional DFT from only 1D DFT and then briefly introduces the Fast Fourier Transform which decreases the number of required operations down to an order of $N \log N$ for a performing a DFT of a signal of length N.

1.2.2 Computing a 2D Fourier transform from 1D Fourier transforms

A 2D Fourier transform can be computed from 1D fourier transforms. For sake of simplicity, let's write $w_P = e^{\frac{-2i\pi}{P}}$. The 2D Fourier Transform of a signal $s[n, m]$ of size (N, M) is defined as :

$$\begin{aligned} S[k,j] &= \sum_{n=0}^{N-1}\sum_{m=0}^{M-1} s[n,m]e^{\frac{-2i\pi jm}{M}}e^{\frac{-2i\pi kn}{N}} & (18)\\ &= \sum_{n=0}^{N-1}\sum_{m=0}^{M-1} s[n,m]w_M^{jm}w_N^{kn} & (19)\\ &= \sum_{n=0}^{N-1}(\sum_{m=0}^{M-1} s[n,m]w_M^{jm})w_N^{kn} & (20)\end{aligned}$$

It turns out that the 2D DFT can be computed by performing a 1D DFT on the rows of s (the inner sum is on n fixed) followed by a 1D DFT on the columns of the result. Let's denote $\hat{S}[n, j]$ the 2D signal for which the row n holds the 1D DFT of the line n of the original signal $s[n, m]$:

$$\hat{S}[n,j] = \sum_{m=0}^{M-1} s[n,m]w_M^{jm} \tag{21}$$

We can write :

$$S[k,j] = \sum_{n=0}^{N-1} \hat{S}[n,j]w_N^{kn} \tag{22}$$

The previous equation corresponds to a 1D DFT performed on the columns of $\hat{S}$. Therefore, to compute a 2D DFT of s you need to :

1. Perform a 1D DFT on the rows of s which leads to the 2D signal $\hat{S}$
2. Perform a 1D DFT on the columns of $\hat{S}$

The two previous operations can be done in the reverse order, no matter. This way of computing a 2D DFT from 1D DFTs is interesting because you can now compute DFT of n-dimensional signals involving only 1D-DFTs.

1.2.3 Computing 2 DFT of real sequences at once

When we will apply the DFT to compute convolution products, we will need to compute two DFT : one for the image and one for the kernel. In fact, two DFTs can be simply computed with only one DFT[2]. Given f and g are real signals, you can combine them to form a complex signal $h[n] = f[n] + ig[n]$. Given the DFT is linear : $FT[h][k] = FT[f][k] + iFT[g][k]$. The individual DFTs $FT[f]$ and $FT[g]$ are not the real and imaginary parts of $FT[h]$ given that the components of the DFTs are complex numbers. However, you know that :

$$f[n] = \frac{h[n] + (h[n])^*}{2}$$
$$g[n] = \frac{h[n] - (h[n])^*}{2i}$$

[2] Adapted from **Digital Signal Processing**, J. Proakis, D. Manolakis. Chap 8, link

where c^* is the conjugate of the complex number c. Therefore, you can write :

$$\begin{aligned} FT[f][k] &= \frac{FT[h][k] + FT[h^*][k]}{2} \\ FT[g][k] &= \frac{FT[h][k] - FT[h^*][k]}{2i} \end{aligned}$$

It is possible to express $FT[h^*][k]$ in terms of $FT[h][k]$ by simply observing that :

$$\begin{aligned} FT[h^*][k] &= \sum_{n=0}^{N-1} h^*[n]w_N^{kn} = \sum_{n=0}^{N-1} h^*[n]w_N^{kn}w_N^{-Nn} \\ &= \sum_{n=0}^{N-1} h^*[n]w_N^{-(N-k)n} = (\sum_{n=0}^{N-1} h[n]w_N^{(N-k)n})^* \\ &= (FT[h][N-k])^* \end{aligned}$$

We finally get :

$$\begin{aligned} \forall k \in [1, N-1] FT[f][k] &= \frac{FT[h][k] + (FT[h][N-k])^*}{2}, FT[f][0] = \mathrm{Re}(FT[h][0]) \\ \forall k \in [1, N-1] FT[g][k] &= -i\frac{FT[h][k] - (FT[h][N-k])^*}{2}, FT[g][0] = \mathrm{Im}(FT[h][0]) \end{aligned}$$

This technique is not required if you want to convolve several images with the same kernel since, in that case, you can precompute the DFT of the kernel one for all.

1.2.4 The symmetry of DFT of real sequences

The DFT of real sequences has a specific symmetry that the algorithms use. Namely, a DFT of a real sequence is half-complex. Let $f[n]$ be a real sequence of length N, then :

$$\begin{aligned} \forall i \in [0, N[, (FT[f][k])* &= (\sum_{n=0}^{N-1} f[n]w_N^{kn})* = \sum_{n=0}^{N-1} f[n]w_N^{-kn} \\ &= \sum_{n=0}^{N-1} f[n]w_N^{-kn}w_N^{Nn} = \sum_{n=0}^{N-1} f[n]w_N^{(N-k)n} \\ &= \sum_{n=0}^{N-1} f[N-k]w_N^{kn} = FT[f][N-k] \end{aligned}$$

Therefore, when one wants to compute the DFT of a real sequence, only half the coefficients actually need to be computed. This leads to the definition of *half-complex* or *complex conjugate* formats used by libraries such as the GSL of FFTW.

1.2.5 Fast Fourier Transform

To compute a DFT according to the previous definition, we need an order of N^2 operations. There are some efficient algorithms that have been designed, which involve only an order of $N \log N$ operations. They are called Fast Fourier Transform algorithms. We do not provide here how they work but just some pointers to some of the FFT algorithms :

- Cooley-Tukey algorithm (Wikipedia) : breaks down a DFT of size $N = N_1N_2$ into two DFT of size N_1 and N_2
- Prime-factor algorithm (Wikipedia) : breaks down a DFT of size $N = N_1N_2$ into two DFT of size N_1 and N_2 when N_1 and N_2 have no common factors

- Rader's algorithm (Wikipedia)
- Winograd's algorithm

For the divide-and-conquer like algorithms (Cooley-Tukey, Prime-Factor), computing a DFT of a signal of size N can be done by computing fourier transforms of smaller sizes, in particular of sizes which correspond to the prime factors of N. Quite a lot of details can be found in the documentation of the GSL on FFT algorithms (take the archive online and browse in the doc directory to find the file fft_algorithms.texi) (see also the GSL documentation for some pointers). The FFTW library makes use of the Cooley-Tukey algorithm, the prime factor algorithm, Rader's algorithm for prime sizes, and a split-radix algorithm and they also propose a generator of adhoc FFT algorithms for specific sizes, which, according to their documentation "produces new algorithms that we do not completely understand".

1.2.6 Computational complexity

There are several librairies providing optimized methods for computing a DFT. To cite only two of them : the GSL (Gnu Scientific Library) and `FFTW3`. Others are cited on the FFTW website. In these libraries, there are a bunch of methods for computing several types of Fourier Transforms: for real or complex data, for powers of two. For the FFTW, it seems that the algorithms specifically designed for signals with a power of 2 size perform slightly better than the mixed-radii algorithms. In the following we propose to compare the performances of the GSL and FFTW in computing the FFT of real sequences. For the FFTW, this means using plans such as *fftw_plan_dft_r2c_2d* and *fftw_plan_dft_c2r_2d*. With the GSL, there is no method for computing 2D FFT, forcing us to make use of 1D transforms. However, I was not able to write down a method computing a 2D FFT with 1D transforms, using their methods working with real sequences (*gsl_fft_real_transform*. The code used for the benchmark of the GSL involves the method working with complex sequences (*gsl_fft_complex_forward*). To be almost fair in the comparison, as computing FFT of real sequences involves computing roughly half of the coefficients, the time we plot is half the time required by the complex GSL transforms.

The scripts used to compute the Fourier Transform with the GSL and FFTW3 are `fft_gsl.cc, fft_gsl.h` and `fft_fftw.cc, fft_fftw.h`.

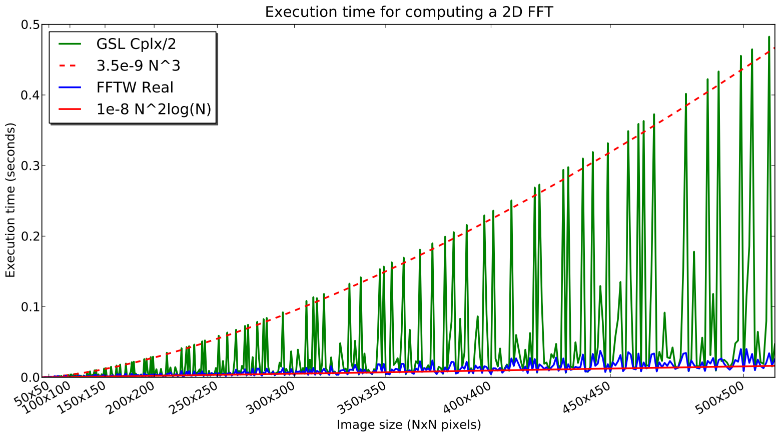


Figure 3: Execution time for computing a 2D FFT. With a signal of size $N \times N$, the performancs of the FFTW is well approximated with a curve in $N^2 \log(N)$ while the performances of the GSL are sometimes an order of magnitude worse.

The image 3 shows the time it takes to compute the 2D-FFT for different signal sizes, the signals being randomly generated. The lower bound of the execution times fits well with a $N \log(N)$ relationship. For some sizes, the GSL is performing worse than this relationship. The possible reason for this comes from how the GSL and the FFTW compute the FFT. As explained before, the algorithms rely on a factorization of the size of the signal. With the GSL, the implemented factors are $2, 3, 4, 5, 6, 7$ (see GSL website) and for FFTW the implemented factors are $2, 3, 5, 7, 11, 13$ (see FFTW website). For the GSL, and apparently not the FFTW, if the size of the signal is not a product of only these factors (e.g. a large prime number), the performances can be worse than $N \log(N)$. In particular, this is the case for large prime numbers. If we look at the sizes where the performances are bad, they correspond to sizes like : 127x127, 131x131, .., 157x157, .. which are all prime numbers. It seems that FFTW handles successfully these large prime numbers since it is using the Rader's algorithm while the GSL only seems to use the Cooley-Tukey algorithm.

Given the FFTW is faster than the GSL to compute a FFT, we don't use the GSL any further for computing convolution products.

2 Convolution product using the Fast Fourier Transform

2.1 Introduction

In the previous sections, we introduced the linear and circular convolutions as well as the Fourier transform with its efficient implementation of the Fast Fourier Transform. Here, we detail the link between the fourier transform and the convolution product.

2.1.1 Convolution theorem : circular convolution and fourier transform

Suppose we want to convolve a signal of N with a filter of size M. As we saw in the introduction, a direction implementation of the convolution product, using nested for loops requires an order of $N.M$ operations. It is possible to speed up this computation by using the Fourier Transform. This relies on the the convolution theorem : the circular convolution product of two signals equals the inverse Fourier transform of the product of the fourier transforms of the signals :

Theorem 1. *Let f and g be two discrete sequences of respectively length N and M and $P \in \mathbb{N}^*$. The inverse fourier transform of the product of the fourier transform of f_P and g_P equals the circular convolution of f and g modulo P:*

$$\forall k \in [0, P-1], (f \circledast_P g)[k] = IFT[FT[f_P].FT[g_P]][k]$$

To show this, let's compute the fourier transform of the circular convolution modulo P of f and g :

$$\begin{aligned}
\forall k \in [0, P-1], FT[f \circledast_P g][k] &= \sum_{i=0}^{P-1} \left((f \circledast_P g)[i] w_P^{ki}\right) && (23)\\
&= \sum_{i=0}^{P-1} \sum_{j=0}^{P-1} f_P[j] g_P[i-j] w_P^{ki} && (24)\\
&= \sum_{j=0}^{P-1} f_P[j] w_P^{kj} \sum_{i=0}^{P-1} g_P[i-j] w_P^{k(i-j)} && (25)\\
& && (26)
\end{aligned}$$

Since the signal g_P is periodic of period P, then :

$$\forall j \in [0, P-1], \sum_{i=0}^{P-1} g_P[i-j] w_P^{k(i-j)} = \sum_{i=0}^{P-1} g_P[i] w_P^{ki}$$

Which leads to :

$$\begin{aligned}
\forall k \in [0, P-1], FT[f \circledast_P g][k] &= \sum_{j=0}^{P-1} f_P[j] w_P^{kj} \sum_{i=0}^{P-1} g_P[i-j] w_P^{k(i-j)} && (27)\\
&= \sum_{j=0}^{P-1} f_P[j] w_P^{kj} \sum_{i=0}^{P-1} g_P[i] w_P^{ki} && (28)\\
&= FT[f_P][k] FT[g_P][k] && (29)
\end{aligned}$$

By taking the inverse fourier transform of both sides, we get the result we wanted to demonstrate.

2.1.2 Convolutions and Fourier Transform

We can now bring together the elements we saw in the previous section and in section 1.1.2 on the link between circular and linear convolutions :

Proposition 1. *Let f and g be two discrete sequences of respectively length N and M and $P \geq N + M - 1$. The* **Full** *linear convolution can be computed with the Fast Fourier Transform:*

$$\begin{aligned} \forall k \in [0, N+M-2], \quad h_f[k] &= \sum_{i=\max(0,k-M+1)}^{\min(N-1,k)} f[i]g[k-i] \\ &= (f \circledast_P g)[k] \\ &= IFT[FT[f_P].FT[g_P]][k] \end{aligned}$$

with :

$$\begin{aligned} \forall k \in [0, P-1], f_P[k] &= \begin{cases} f[k] & \textit{if} \quad 0 \leq k \leq N-1 \\ 0 & \textit{otherwise} \end{cases} \\ \forall k \in [0, P-1], g_P[k] &= \begin{cases} g[k] & \textit{if} \quad 0 \leq k \leq M-1 \\ 0 & \textit{otherwise} \end{cases} \end{aligned}$$

Proposition 2. *Let f and g be two discrete sequences of respectively length N and M and $P \geq N + \frac{M}{2}$. The* **Same** *linear convolution can be computed with the Fast Fourier Transform:*

$$\begin{aligned} \forall k \in [0, N-1], \quad h_s[k] &= \sum_{i=\max(0,k-\frac{M-1}{2})}^{\min(N-1,k+\frac{M}{2})} f[i]g[k+M/2-i] \\ &= (f \circledast_P g)[k+\frac{M}{2}] \\ &= IFT[FT[f_P].FT[g_P]][k+\frac{M}{2}] \end{aligned}$$

with :

$$\begin{aligned} \forall k \in [0, P-1], f_P[k] &= \begin{cases} f[k] & \textit{if} \quad k \leq N-1 \\ 0 & \textit{otherwise} \end{cases} \\ \forall k \in [0, P-1], g_P[k] &= \begin{cases} g[k] & \textit{if} \quad \frac{M-1}{2} - N \leq k \leq M-1 \\ 0 & \textit{otherwise} \end{cases} \end{aligned}$$

Proposition 3. *Let f and g be two discrete sequences of respectively length N and M and $P \geq N$. The* **Valid** *linear convolution can be computed with the Fast Fourier Transform:*

$$\begin{aligned} \forall k \in [0, N-1], \quad h_v[k] &= \sum_{i=k}^{k+M-1} f[i]g[k+M-1-i] \\ &= (f \circledast_P g)[k+M-1-i] \\ &= IFT[FT[f_P].FT[g_P]][k+M-1-i] \end{aligned}$$

with :

$$\begin{aligned} \forall k \in [0, P-1], f_P[k] &= \begin{cases} f[k] & \textit{if} \quad k \leq N-1 \\ 0 & \textit{otherwise} \end{cases} \\ \forall k \in [0, P-1], g_P[k] &= \begin{cases} g[k] & \textit{if} \quad 0 \leq k \leq M-1 \\ 0 & \textit{otherwise} \end{cases} \end{aligned}$$

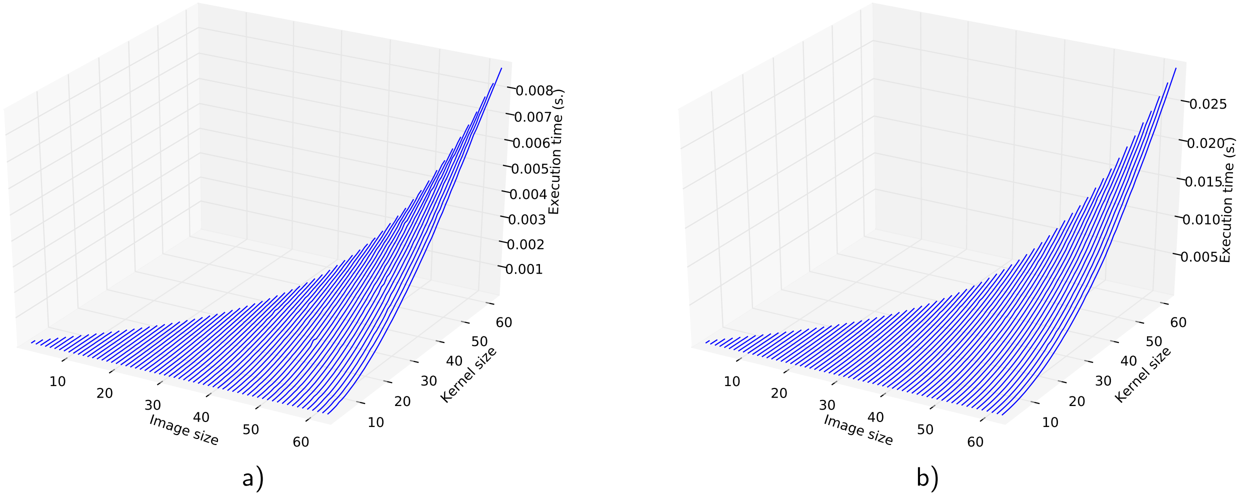
2.1.3 Speeding-up by padding

Computing the DFT is faster for some specific sizes. As we saw previously on figure 3, the performances can be really reduced if the image's size is non-optimal, especially with the GSL. In the previous section, we gave lower bounds for the sizes of the FFT we need in order to compute the different types of convolution. We can use more padding than necessary in order to reach sizes that are optimally computed with the different libraries. To determine if a size is optimal, we just factorize it and check if the factorization contains only factors for which sub-transforms are optimally computed. In GSL, this means only size that can be written $2^a * 3^b * 4^c * 5^d * 6^e * 7^f$ (see GSL website). In addition, it seems that when the size is a multiple of $4.4.4.2 = 128$, the mixed-radii algorithm does not perform a FFT efficiently. Therefore, all the sizes multiple of 128 are considered non-optimal. In FFTW3, an optimal size is of the form $2^a * 3^b * 5^c * 7^d * 11^e * 13^f$, with $e + f$ equals 0 or 1 (see FFTW website). We also discard the sizes multiple of $4.4.4.2$ which appeared to decrease the performances.

2.2 Benchmarks

2.2.1 Convolutions with a direct implementation

We first measure the time it takes to compute a linear or circular convolution for different image and kernel sizes to take it as a basis for comparing it to the FFT-based implementation. The execution times for computing a **same** linear and circular convolution are shown on figure 4. The x-axis and y-axis are labelled by the width N of the image of size $N \times N$ and width M of the kernel of size $M \times M$.



a)

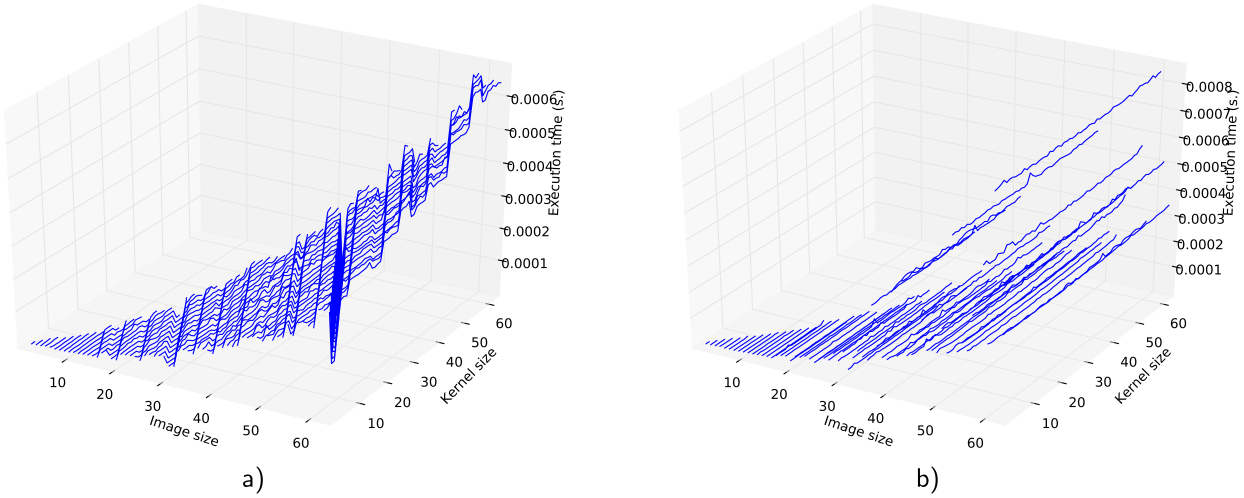
b)

Figure 4: Execution times for computing a linear or circular convolution of an image of size $N \times N$ with a kernel of size $M \times M$, with $M \leq N$. a) Execution time for performing a **same** linear convolution with nested for loops. b) Execution time for performing a circular convolution modulo N.

2.2.2 Convolutions with FFTW

The figure 5 illustrates the time it takes to perform a **Same** linear convolution or a circular convolution between an image of size $N \times N$ and a kernel of size $M \times M$. The x and y axis are plotting N and M, not the real size.

The benefits of padding can be seen on figure 6. Here we plot the time it takes to compute a convolution between an image of size 128×128 and a kernel of variable sizes, from 3×3 upto 63×63. Both the times with and without padding are displayed.



a)

b)

Figure 5: a) Execution time for performing a **same** linear convolution with FFTW. Even if the execution times seem quite noisy, it is still between 1 and 3 times faster than without padding. b) Execution time for performing a circular convolution modulo N. Here the time depends only on the size of the image. No padding was used.

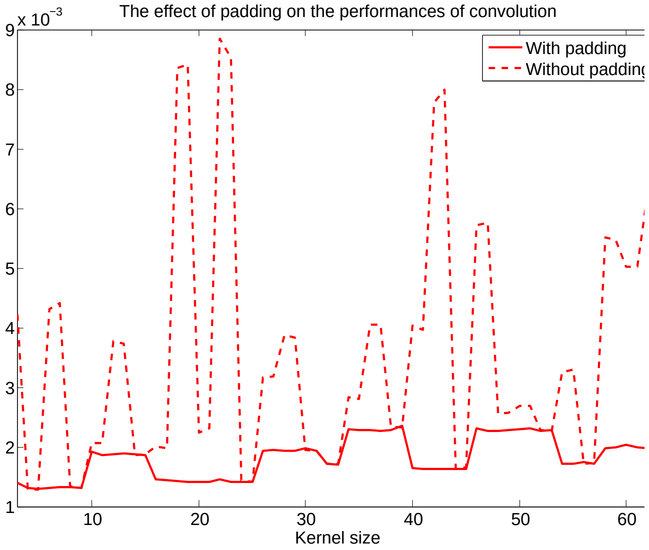
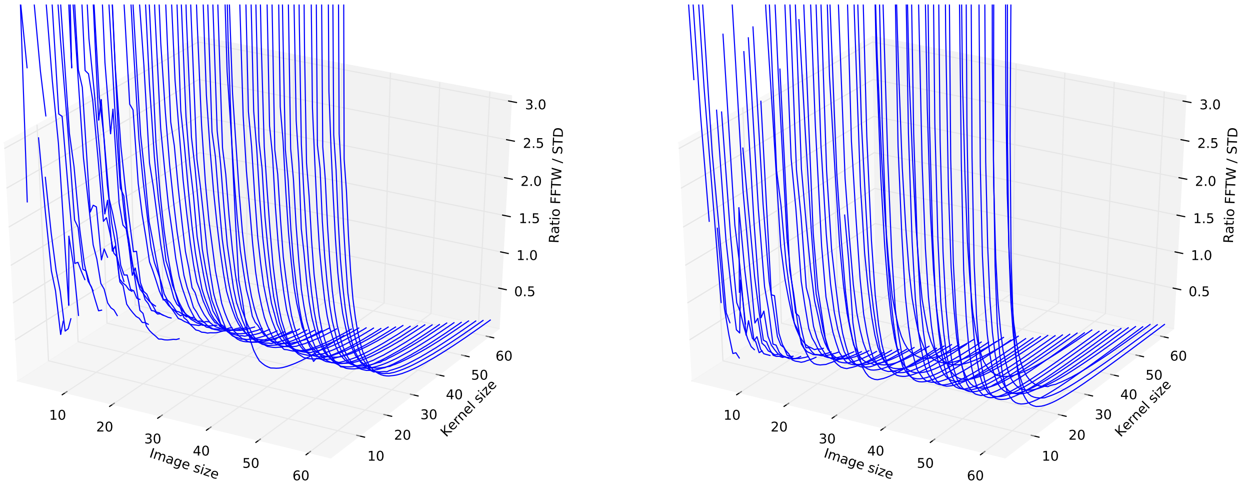


Figure 6: Both the time to compute a linear convolution with and without padding are displayed. The image's size is 128×128. The kernel's size varies between 3×3 and 63×63.

2.3 Comparison between FFT based and standard convolution

The previous benchmarks allowed to detect which FFT based algorithm was faster and if padding improves or not the performances. However, it is not always beneficial to use the FFT to compute a convolution product. For the linear convolutions, you are actually computing FFT of signals larger than the image since you need to pad it with zeros to avoid the aliasing. There are also the element wise products of the FFTs that add costs to the theoretical $N\log(N)$ complexity. In fact, for small kernels, using the FFT based convolution is actually slower than using the direct implementation with nested for loops. On figure 7, we compute the ratio of the execution times of convolutions using the FFT or direct implementation. The linear convolutions are faster with the FFT based implementation for kernel's size at least 12×12 independently of the image's size. The circular convolutions are faster with the FFT based implementation for kernel's size at least 8×8 independently of the image's size.



a) b)

Figure 7: Comparison of the direct and FFT based implementations of convolution products. a) Linear convolutions : linear convolutions are faster with the FFT based implementation if the kernel is at least 12×12, independently of the image's size. b) Circular convolutions : circular convolutions are faster with the FFT based implementation if the kernel's size is at least 8×8.

3 C++ codes

Here, we just provide a list of the C++ scripts used through this document :

- Computing a 2D Fourier Transform :
  - with GSL : fft_gsl.cc, fft_gsl.h
  - with FFTW : fft_fftw.cc,fft_fftw.h
- Computing linear and circular convolutions with the direct implementation : convolution_std.cc, convolution_std.h
- Computing linear and circular convolutions with the FFT and FFTW :convolution_fftw.cc, convolution_fftw.h
- The script for the benchmarks of the direct implementation of the convolution : bench_convolution_std.cc
- The script for the benchmarks of the FFT-based convolution with FFTW :bench_convolution_fftw.cc
- The python script for plotting the execution times with the FFTW convolution :plot_fftw_convolution.py
- The python script for plotting the execution times of the direct implementation :plot_std_convolution.py
- The python script for comparing the execution times of the two implemenations : plot_compare_convolution.py
- The python script for comparing the execution times for computing a 2D FFT with the GSL or FFTW: plot_test_fft.py