Università degli Studi di Verona

FACOLTÀ DI SCIENZE MATEMATICHE FISICHE E NATURALI
Corso di Laurea Triennale in Matematica Applicata

Tesi di Laurea

Nonequispaced Fast Fourier Transform and Applications

Candidato:
Simone Parisotto
Matricola vr069215

Relatore:
Dott. Marco Caliari

Anno Accademico 2009–2010

"Nobody said it was easy
No one ever said it would be this hard."

The Scientist - Coldplay

This thesis is dedicated to my parents, Daniele and Nadia, who supported me in every moment of joy or sadness. I cannot forget how much love they reversed over me. Doing always my best it is the only way I know to refund them for their love.

A special thanksgiving to my advisor, Dott. Marco Caliari, for his patience and assistance during the works of this thesis and also for his amazing lectures. Thanks for giving me the opportunity to going deeper into the world of Discrete Fourier Transform: it has been a great pleasure and I enjoyed it very much.

A necessary mention to my friends for their careful support over these years: Andrea Corrà, Diego Carbognin, Domenica Romaniello, Elena Zogno, Enrico Costantini, Enrico Scapin, Federica Martelli, Francesca Begali, Francesca Girella, Francesco Tosi, Gianmarco Lodola, Giulia Caldana, Ilaria Zampieri, Lucia Caldana, Manuel Bergamasco, Marco De Donatis, Marco Ferrari, Matteo Montaperto and Vera Di Stefano.

I am also very grateful to Elena Veronesi and Sara Buscarini, who introduced me to the fascinating world of mathematics. Also, thanks to Benoit Mandelbrot, whose books inspired me to try to follow the math's way.

This work is dedicated also to my dear second family of www.coldplayzone.it, the website I set up in 2006. I am proud to work with Gabriele Bigatti, Patrizio Negro, Federico Drago, Elena Curti and Denise Tonolo and share with them endless queues along uncountable and unforgettable gigs. Without them it is a waste of time.

And, in the end, thanks to the special friends of Daylight Band, the best Coldplay Tribute Band in the world, from Padua.

Sorry if I am forgetting someone but the rows remaining are few: I am keeping a lot of other people in my heart.

Viva la Vida (in Technicolor, obviously).

Simone Parisotto

Introduction

This thesis concerns the *Discrete Fourier Transform* (DFT) and its implementation and approximation in MATLAB. Given $f : [a, b) \to \mathbb{C}$ a l-periodic function with $l = b - a$, the *Discrete Fourier Transform* of f, in its exponential formulation, is

$$\sum_{k=-\infty}^{+\infty} c_k e^{i2\pi k \frac{x-a}{b-a}} \tag{1}$$

with the *Fourier coefficients*

$$c_k = \frac{1}{b-a} \int_a^b f(x) e^{-i2\pi k \frac{x-a}{b-a}} \mathrm{d}x, \ \forall k \in \mathbb{Z}. \tag{2}$$

In order to compute the coefficients c_k, the integral (2) has to be approximated: recalling the composite trapezoidal quadrature formula for periodic functions

$$\int_a^b g(x)\mathrm{d}x \approx \frac{b-a}{N} \sum_{k=1}^{N} g(x_k), \ x_k = a + (k-1)\frac{b-a}{N}, \tag{3}$$

then the approximation of c_k, through (3), is called $\hat{f}_k$. Now we are able to introduce the *truncated* and *approximated* version of formula (1):

$$F_N(x) = \sum_{k=1}^{N} \hat{f}_k e^{i2\pi k \frac{x-a}{b-a}}.$$

In order to evaluate the expression above at a set of N nodes we can use:

- the *Fast Fourier Transform* (FFT), cost $\mathcal{O}(N \log N)$, if the nodes are equispaced;
- the *Matrix Fourier Transform* (MFT), cost $\mathcal{O}(N^2)$, or the recent *Nonequispaced Fast Fourier Transform* (NFFT), cost $\mathcal{O}(N \log N + mN)$, $m \ll N$, if the nodes are not equispaced.

We will call the algorithms by their acronyms, referring to the transformation from the space domain to the frequency domain $\{f(x_k)\}_k \to \{\hat{f}_k\}_k$, or adding an I (Inverse) to their acronyms, referring to the transformation from the frequency domain to the space domain. The aims of this thesis are:

- showing that FFT is more accurate and less expensive than MFT if the number of coefficients is larger than a certain number;
- showing that NFFT is less expensive than MFT, only slightly less accurate than MFT and FFT and it can be used for a set of nonequispaced nodes;
- showing how NFFT can be used in solving hyperbolic *Partial Differential Equation* with a periodic transport coefficient.

All scripts used in this thesis are available in the enclosed CD.

Contents

Chapter 1

Theory of Fourier Series

A lot of physical phenomena, such as acoustical or electromagnetical ones, can be represented with a wave floating in the space. These waves obey to the *superposition principle*: they interact adding their effects in every point of the space. From this idea, every signal (and every function), under suitable conditions, can be represented with a superposition of elementary waves, each one with a fixed *frequency*. The *frequency* is the number of occurrences of a repeating event per unit time and *hertz* is its SI unit. Fourier series were introduced by *Joseph Fourier* (1768–1830) for the purpose of solving the heat equation in a metal plate. The heat equation is a partial differential equation. Prior to Fourier's work, there was no known solution to the heat equation in a general situation, although particular solutions were known if the heat source behaved in a simple way, in particular, if the heat source was a sine or cosine wave. These simple solutions are now sometimes called eigensolutions. Fourier's idea was to model a complicated heat source as a *superposition* (or *linear combination*) of simple sine and cosine waves, and to write the solution as a superposition of the corresponding eigensolutions. This superposition or linear combination is called the Fourier series.

1.1 Trigonometric Polynomials

Definition 1.1. A trigonometric polynomial $P_m(x)$ of order m is a 2π-periodic function, in other words $f(x) = f(x + 2k\pi)$, $\forall x \in \mathbb{R}\ \forall k \in \mathbb{Z}$, so that

$$P_m(x) = a_0 + \sum_{k=1}^{m} (a_k \cos kx + b_k \sin kx)$$

with $a_k, b_k \in \mathbb{C}$ called coefficients of P_m.

The 2π-periodic function space can be equipped with

- an inner product: $$\langle u, v\rangle = \int_0^{2\pi} u(x)\overline{v(x)}\mathrm{d}x$$

- a norm: $$||u|| = \sqrt{\langle u, u\rangle} = \Big(\int_0^{2\pi} u(x)\overline{u(x)}\mathrm{d}x\Big)^{\frac{1}{2}}.$$

The set $\mathcal{B} = \Big\{\sin nx, \cos nx,\ n \in \mathbb{N}\Big\}$ is an othogonal basis for the space of 2π-periodic funcions. The following results, called *orthogonality relations*, are true:

$$\begin{aligned}
\int_0^{2\pi} \cos mx \cos nx \mathrm{d}x &= \begin{cases} \pi & \text{if } m = n \neq 0 \ (2\pi \text{ when } m = n = 0) \\ 0 & \text{if } m \neq 0 \end{cases} \\
\int_0^{2\pi} \sin mx \sin nx \mathrm{d}x &= \begin{cases} \pi & \text{if } m = n \neq 0 \\ 0 & \text{otherwise} \end{cases} \\
\int_0^{2\pi} \cos mx \sin nx \mathrm{d}x &= 0.
\end{aligned}$$

1.2 Fourier Series for 2π-periodic functions

Our aim now is to represent every periodic function $f : [0, 2\pi) \to \mathbb{C}$ as a linear combination of sine and cosine:

$$f_\infty(x) = \frac{a_0}{2} + \sum_{k=1}^{\infty} (a_k \cos kx + b_k \sin kx) \tag{1.1}$$

Formula (1.1) is called *Trigonometric Fourier Series* of f. The coefficients are:

$$a_0 = \frac{1}{\pi} \int_0^{2\pi} f(x) \mathrm{d}x, \quad a_k = \frac{1}{\pi} \int_0^{2\pi} f(x) \cos kx \mathrm{d}x, \quad b_k = \frac{1}{\pi} \int_0^{2\pi} f(x) \sin kx \mathrm{d}x.$$

There is another notation for a *Fourier Series*: from Euler's formula $e^{i\theta} = \cos\theta + i \sin\theta$, we can write the *Exponential Fourier Series* as:

$$\sum_{k=-\infty}^{+\infty} c_k e^{ikx}, \quad \text{with coefficients} \quad c_k = \frac{1}{2\pi} \int_0^{2\pi} f(x) e^{-ikx} \mathrm{d}x.$$

The identities:

$$\cos x \equiv \frac{e^{ix} + e^{-ix}}{2} \quad \text{and} \quad \sin x \equiv \frac{e^{ix} - e^{-ix}}{2i} \tag{1.2}$$

show that *Trigonometric Fourier Series* (1.1) and *Exponential Fourier Series* (1.2) are equivalent because, for $k \geq 1$, we have:

$$a_k = c_k + c_{-k}, \quad b_k = i(c_k - c_{-k}), \qquad \text{and} \qquad c_0 = \frac{a_0}{2}, \quad c_k = \frac{a_k - ib_k}{2}, \quad c_{-k} = \frac{a_k + ib_k}{2}.$$

If we truncate the series at a fixed index N, then $f_N(x)$ is called *truncated Fourier Series of order N*.

1.3 Convergence Results

In this section we briefly report (without proofs) some convergence results to show how and under which conditions the *Fourier Series* of f can approximate f.

Theorem 1.1 (of **uniform convergence**). *Let $f(x)$ a 2π-periodic and C^1-piecewise function on the interval $[0, 2\pi)$. Then, the Fourier Series of $f(x)$ converges uniformly to $f(x)$ on every compact set which does not contain any discontnuity point.*

Remark 1.1. Under the same hypothesis of theorem (1.1), if x^* is a discontinuity point for $f(x)$, then the Fourier Series of $f(x^*)$ converges to the average of right and left limits of f to x^*.

Theorem 1.2 (of **convergence in 2nd order mean**). *Let* $f(x)$ *a* 2π*-periodic function on the interval* $[0, 2\pi)$*, with* $\int_0^{2\pi} |f(x)|^2 \mathrm{d}x < +\infty$*, then*

$$\int_0^{2\pi} |f(x) - f_N(x)|^2 \mathrm{d}x \leq \int_0^{2\pi} |f(x) - P_N(x)|^2 \mathrm{d}x$$

for any trigonometric polynomial P_N *of order* N *and*

$$\lim_{N \to +\infty} \int_0^{2\pi} |f(x) - f_N(x)|^2 \mathrm{d}x = 0.$$

Moreover, the following result is true and it's called Bessel–Parseval Identity*:*

$$\frac{1}{\pi} \int_0^{2\pi} |f(x)|^2 \mathrm{d}x = \frac{|a_0|^2}{2} + \sum_{n=1}^{\infty} \Big(|a_k|^2 + |b_k|^2 \Big) = 2 \sum_{k \in \mathbb{Z}} |c_k|^2. \tag{1.3}$$

1.4 Fourier Series for l-periodic functions

It's possible to use the Fourier Series of l-periodic functions (not only 2π-periodic ones). If $f : [a, b) \to \mathbb{R}$ is l-periodic in $[a, b)$, with $l = b - a$,

- its Trigonometric Fourier Series is:

$$\frac{a_0}{2} + \sum_{k=1}^{\infty} a_k \cos 2\pi k \Big(\frac{x-a}{b-a} \Big) + b_k \sin 2\pi k \Big(\frac{x-a}{b-a} \Big) \tag{1.4}$$

  with

$$a_k = \frac{2}{b-a} \int_a^b f(x) \cos 2\pi k \Big(\frac{x-a}{b-a} \Big) \mathrm{d}x, \quad b_k = \frac{2}{b-a} \int_a^b f(x) \sin 2\pi k \Big(\frac{x-a}{b-a} \Big) \mathrm{d}x.$$

- its Exponential Fourier Series is:

$$\sum_{k=-\infty}^{+\infty} c_k e^{i 2\pi k \frac{x-a}{b-a}} \tag{1.5}$$

  with

$$c_k = \frac{1}{b-a} \int_a^b f(x) e^{-i 2\pi k \frac{x-a}{b-a}} \mathrm{d}x, \ \forall k \in \mathbb{Z}.$$

Obviously, every convergence result is true in the new scaled interval $[a, b)$.

Chapter 2

An approximation for the Fourier Series

Let $f : [a, b) \to \mathbb{C}$ a periodic function. We suppose that $f(x)$ can be written as

$$f(x) = \sum_{k=-\infty}^{+\infty} c_k e^{i2\pi k(\frac{x-a}{b-a})}, \tag{2.1}$$

in the sense of Theorem (1.1). We want to implement (2.1) in MATLAB. Since MATLAB can't manage negative or null indexes, we have to translate the basis functions to new ones.

2.1 A good choice of basis functions

Let $[a, b) \in \mathbb{R}$ an interval, N a fixed even natural number and $l = b - a$. We can consider the following basis functions:

$$\phi_k(x) = \frac{e^{i2\pi(k-1-N/2)(x-a)/(b-a)}}{\sqrt{b-a}}, \quad \forall k \in \mathbb{Z}.$$

Now we are ready to show the orthonormal relation of the set of $\{\phi_k(x)\}_k$:

$$\int_a^b \phi_j(x)\overline{\phi_k(x)}\mathrm{d}x = \delta_{jk} \tag{2.2}$$

with δ_{jk} Kronecher's delta. This is true because

$$\phi_j(x)\overline{\phi_k(x)} = \begin{cases} \dfrac{1}{b-a} & \text{if } j = k \\ \\ \dfrac{e^{i2\pi(j-k)(x-a)/(b-a)}}{b-a} & \text{otherwise} \end{cases} \tag{2.3}$$

and

$$\int_a^b \phi_j(x)\overline{\phi_k(x)}\mathrm{d}x = \begin{cases} \displaystyle\int_a^b \frac{1}{b-a}\mathrm{d}x = \frac{1}{b-a}\int_a^b \mathrm{d}x = \frac{1}{b-a}(b-a) = 1 & \text{if } j = k \\ \\ \displaystyle\int_a^b \frac{e^{i2\pi(j-k)(x-a)/(b-a)}}{b-a} = \int_0^1 \frac{e^{i2\pi(j-k)y}}{b-a}(b-a)\mathrm{d}y = 0 & \text{otherwise} \end{cases}$$

because the integral of $\sin(x)$ and $\cos(x)$ functions over a multiple interval of their period is zero.

Lemma 2.1. *The following relation is true and it's called* discrete inner product's orthogonality relation. *If we define*

$$\langle \phi_j, \phi_k \rangle_N = \frac{b-a}{N} \sum_{n=1}^{N} \phi_j(x_n) \overline{\phi_k(x_n)} = \frac{b-a}{N} \sum_{n=1}^{N} e^{i2\pi(n-1)(j-k)/N},$$

then

$$\langle \phi_j, \phi_k \rangle_N = \delta_{jk}, \quad \textit{with } -N+1 \le j-k \le N-1 \textit{ and } \delta_{jk} \textit{ Knonecher's delta.} \tag{2.4}$$

Proof. From (2.3) we have:

$$\langle \phi_j, \phi_k \rangle_N = \begin{cases} \dfrac{b-a}{N} \displaystyle\sum_{n=1}^{N} \frac{1}{b-a} = \frac{b-a}{N}\frac{N}{b-a} = 1 & \text{if } j = k \\ \\ \dfrac{b-a}{N} \displaystyle\sum_{n=0}^{N-1} (e^{i2\pi(j-k)/N})^n = \frac{1-e^{i2\pi(j-k)}}{1-e^{i2\pi(j-k)/N}} = \frac{1-\cos(2\pi(j-k))}{1-e^{i2\pi(j-k)/N}} = 0 & \text{if } j \neq k \end{cases}$$

so (2.4) is shown becuase $\cos(2\pi(j-k))$ is 1 for every j such as $-N+1 \le j-k \le N-1$. □

2.2 DFT: Discrete Fourier Transform

We recall that, due to the orthonormality of $\{\phi_k\}_k$,

$$f(x) = \sum_{k=-\infty}^{+\infty} c_k \phi_k(x), \quad c_k = \int_a^b f(x)\overline{\phi_k(x)}\mathrm{d}x. \tag{2.5}$$

How to approximate c_k? Given a set of $N+1$ equispaced nodes, x_n on the interval $[a,b]$, with a simple linear transformation $y_n = (x_n - a)/(b-a)$ we obtain the new nodes $y_n = (n-1)/N \in [0,1]$. Recalling the composite trapezoidal quadrature formula for a periodic function g

$$\int_a^b g(x)\mathrm{d}x \approx \frac{b-a}{2N}\Big(g(x_1) + 2\sum_{k=2}^{N} g(x_k) + g(x_{N+1})\Big) = \frac{b-a}{N}\sum_{k=1}^{N} g(x_k), \tag{2.6}$$

we can approximate the coefficients in (2.5) obtaining:

$$\begin{aligned} c_k &= \int_a^b f(x)\overline{\phi_k(x)}\mathrm{d}x = \int_a^b f(x)\frac{e^{-i2\pi(k-1-N/2)(x-a)/(b-a)}}{\sqrt{b-a}}\mathrm{d}x = \\ &= \sqrt{b-a}\int_0^1 f((b-a)y+a)e^{-i2\pi(k-1)y}e^{iN\pi y}\mathrm{d}y \approx \\ &\approx \frac{\sqrt{b-a}}{N}\sum_{n=1}^{N}\Big(f(x_n)e^{iN\pi y_n}\Big)e^{-i2\pi(k-1)y_n} = \hat{f}_k. \end{aligned} \tag{2.7}$$

The transformation $[f(x_1), \ldots, f(x_N)]^T \to [\hat{f}_1, \ldots, \hat{f}_N]^T$ is called *Discrete Fourier Transform* (DFT) of f and $[\hat{f}_1, \ldots, \hat{f}_N]^T$ are called *Discrete Fourier coefficients* of the function f. On the other hand, if $\hat{f}_k$ are given complex values, with $k = 1, \dots, N$, we can consider the periodic function

$$F_N(x) = \sum_{n=1}^{N} \hat{f}_n \phi_n(x). \tag{2.8}$$

Evaluation of $F_N(x)$ at the nodes x_k, with $k = 1, \dots, N$, is

$$\begin{aligned}\hat{\hat{f}}_k = \sum_{n=1}^{N} \hat{f}_n \phi_n(x_k) = \sum_{n=1}^{N} \hat{f}_n \frac{e^{i2\pi(n-1-N/2)(x_k-a)/(b-a)}}{\sqrt{b-a}} = \\ = \frac{N}{\sqrt{b-a}} \frac{1}{N} \Big(\sum_{n=1}^{N} \hat{f}_n e^{i2\pi(n-1)y_k} \Big) e^{-iN\pi y_k}.\end{aligned} \tag{2.9}$$

The transformation $[\hat{f}_1, \dots, \hat{f}_N]^T \to [\hat{\hat{f}}_1, \dots, \hat{\hat{f}}_N]^T$ is called *Inverse Discrete Fourier Transform* (IDFT).

2.3 Error estimate

First, defining $J = \mathbb{Z} \backslash \{1, 2, \dots, N\}$, we can investigate the truncation error:

$$\begin{aligned}\int_a^b \left| f(x) - \sum_{j=1}^{N} c_j \phi_j(x) \right|^2 \mathrm{d}x = \int_a^b \left| \sum_{j \in J} c_j \phi_j(x) \right|^2 \mathrm{d}x = \\ = \int_a^b \Bigg(\sum_{j \in J} c_j \phi_j(x) \Bigg) \overline{\Bigg(\sum_{k \in J} c_k \phi_k(x) \Bigg)} \mathrm{d}x = \sum_{k \in J} |c_k|^2.\end{aligned}$$

We can now estimate c_k. For a C^1 function we have:

$$\begin{aligned} c_k &= \int_a^b f(x) \overline{\phi_k(x)} \mathrm{d}x = \\ &= -\frac{b-a}{i2\pi(k-1-N/2)} \left(\frac{f(b) - f(a)}{\sqrt{b-a}} \right) + \frac{b-a}{i2\pi(k-1-N/2)} \int_a^b f'(x) \overline{\phi_k(x)} \mathrm{d}x = \mathcal{O}(k^{-1}).\end{aligned}$$

If $f'(a) = f'(b)$ and $f'(x) \in C^1$, then integrating by parts, we obtain $c_k(x) = \mathcal{O}(k^{-2})$ and so on. Therefore, if $f(x)$ is an infinitely differentiable and periodic function (i.e. all the derivates are periodic), then c_k decays faster than every negative power of k. This property is called *exponential* or *spectral convergence*. Moreover, we have the following theorem.

Theorem 2.1. *Let $f_N(x)$ denote the truncation of the exact Fourier series. Then,*

$$|f(x) - f_N(x)| \le \sum_{k \in J} |c_k|, \tag{2.10}$$

that is to say, the error is bounded by the sum of the absolute value of all neglected coefficients. Let now F_N defined as in (2.8). Then

$$|f(x) - F_N(x)| \le 2 \sum_{k \in J} |c_k| \tag{2.11}$$

that is to say, the error is bounded by twice the sum of the absolute values of all the neglected coefficients. Comparing (2.10) and (2.11) we conclude: the penalty for using quadrature to approximate coefficients is at worst a factor of two (see [2]).

Since $\{\phi_k\}_k$ is orthonormal also for the discrete inner product for $1 \leq k \leq N$, then by Lemma (2.1) the trigonometric polynomial (2.8), which is the truncated approximated Fourier Series, is an *interpolation Fourier polynomial* of f on the set x_j, for all j used. In fact:

$$\begin{aligned} F_N(x_k) &= \sum_{n=1}^{N} \hat{f}_n \phi_n(x_k) = \\ &= \sum_{n=1}^{N} \left(\left(\frac{\sqrt{b-a}}{N} \sum_{m=1}^{N} f(x_m) e^{iN\pi y_m} \right) e^{-i2\pi(n-1)y_m} \right) \frac{e^{i2\pi(n-1-N/2)(x-a)/(b-a)}}{\sqrt{b-a}} = \\ &= \frac{1}{N} \sum_{m=1}^{N} f(x_m) e^{iN\pi(m-1)/N} e^{-iN\pi(k-1)/N} \sum_{n=1}^{N} e^{-i2\pi(n-1)(m-1)/N} e^{i2\pi(n-1)(k-1)/N} = \\ &= \frac{1}{N} \sum_{m=1}^{N} f(x_m) e^{i(m-k)\pi} \sum_{n=1}^{N} e^{i2\pi(n-1)(k-m)/N} = \frac{1}{N} f(x_k) N = f(x_k). \end{aligned}$$

This result means that trapezoidal quadrature's formula on N points is *exact* for the basis functions $\{\phi_k\}_{k=-N+1}^{N-1}$.

Chapter 3

Equispaced MFT and FFT

From now on, we speak of *Fourier coefficients* instead of *Discrete Fourier coefficients* $\{\hat{f}_k\}_k$, while $\{\hat{\hat{f}}_k\}_k$ stands for the values found by the transformation from the frequency domain to the space domain *Inverse Discrete Fourier Transform.*

Given M equispaced evaluation points and N complex values, the *Matrix Fourier Transform* (MFT) and the *Fast Fourier Transform* (FFT) are two ways to implement (2.7) in MATLAB. Obviously, also the inverse operation, *Inverse Discrete Fourier Transform* (IDFT), can be implemented through two ways: the *Inverse Matrix Fourier Transform* (IMFT) and the *Inverse Fast Fourier Transform* (IFFT). For hystorical reasons it is easy to misunderstand which transform we are referring to. In general we will talk about MFT and FFT and where is possible we will try to be more accurate. However the context will clarify our pourposes.

Our goal is to show that FFT is computationally more efficient than MFT, from a certain number of Fourier coefficients. Moreover, we will show how to evaluate the MFT and the FFT on M equispaced evaluation points with $M \neq N$.

3.1 M equispaced evaluation points, with $M = N$

This is the case with N equispaced sampling points and N complex values. To test this implementations, run the MATLAB's scripts `./matrixFT/simple_test_rebuilding.m` (for MFT) and `./fastFT/simple_test_rebuilding.m` (for FFT).

3.1.1 MFT: Matrix Fourier Transform

In MFT only matrix-vector multiplications are used, reported in (2.7) and (2.9). MFT is a straightforward implementation of (2.7) and (2.9) using a matrix, called F, defined as

$$(F)_{jk} = e^{-i2\pi(j-1)y_k}, \quad \text{with } y_k = (n-1)/N,\ n = 1, \dots, N+1,$$

which lets us to compute the matrix-vector multiplication version of the DFT and the IDFT:

- MFT: from the space domain to the frequency domain. The coefficients $[\hat{f}_1, \dots, \hat{f}_N]^T$ are:

$$\frac{\sqrt{b-a}}{N} \cdot F[f(x_1)e^{iN\pi y_1}, \dots, f(x_N)e^{iN\pi y_N}]^T \tag{3.1}$$

- IMFT: from the frequency domain to the space domain to space domain. The values $[\hat{\hat{f}}_1, \dots, \hat{\hat{f}}_N]^T$ are:

$$\frac{N}{\sqrt{b-a}}\Big(\frac{F^H[\hat{f}_1, \dots, \hat{f}_N]}{N}\Big) \circ [e^{-i\pi N y_1}, \dots, e^{-i\pi N y_N}], \tag{3.2}$$

where "$\circ$" is the Hadamar product: $(A \circ B)_{i,j} = A_{i,j} \cdot B_{i,j}$ and with "H" the Hermitian transpose.

3.1.2 FFT: Fast Fourer Transform

FFT is the most efficient algorithm to compute Fourier coefficients in one dimension. Invented by Gauss in 1805, to study *2 Pallas* and *3 Juno*'s trajectories, this algorithm became famous only in 1965 when *J. W. Cooley* of IBM and *John W. Tukey* of *University of Princeton* published a paper reinventing the algorithm and describing how to implement it on a PC. The basic idea of this algorithm is not to compute the N necessary coefficients, but to compute separately N_1 and N_2 coefficients, with $N = N_1 + N_2$. If we choose N as a power of 2, it is possible to set $N_1 = N_2$: this is a great advantage for the computational cost because every number can be expressed as a power of 2's summation. In general, given a sequence of $\{f(x_n)\}_n$, it's possible to decompose (1.5) into two parts, one even and one odd respectively:

$$\hat{f}_k = \sum_{n=0}^{N/2-1} f(x_{2n}) e^{-i2\pi \frac{2n}{N} k} + \sum_{n=0}^{N/2-1} f(x_{2n+1}) e^{-i2\pi \frac{2n+1}{N} k}.$$

Recalling $y_n = (n-1)/N$, denoting the DFT of the even-indexed inputs x_{2n} by E_k and the DFT of the odd-indexed inputs x_{2n+1} by O_k and collecting common factor $e^{-i2\pi \frac{1}{N} k}$, called *twiddle factor*, in the second summation, we obtain:

$$\hat{f}_k = \underbrace{\sum_{n=0}^{N/2-1} f(x_{2n}) e^{-i2\pi \frac{n}{N/2} k}}_{\text{DFT of even−indexed part of } \{y_n\}_n} + e^{-i2\pi \frac{1}{N} k} \underbrace{\sum_{n=0}^{N/2-1} f(x_{2n+1}) e^{-i2\pi \frac{n}{N/2} k}}_{\text{DFT of odd−indexed part of } \{y_n\}_n} = E_k + e^{-i2\pi \frac{1}{N} k} O_k.$$

These smaller DFTs have a length of $N/2$, so we need compute only $N/2$ outputs: thanks to the periodicity properties of the DFT, the outputs for $N/2 \le k < N$ from a DFT of length $N/2$ are identical to the outputs for $0 \le k < N/2$. That is,

$$E_{k+N/2} = E_k$$
$$O_{k+N/2} = O_k.$$

The *twiddle factor* obeys the relation

$$e^{-2\pi i \frac{(k+N/2)}{N}} = e^{-\pi i} e^{-2\pi i \frac{k}{N}} = -e^{-2\pi i \frac{k}{N}},$$

flipping the sign of the $O_{k+N/2}$ terms. Thus, the whole DFT can be calculated as follows:

$$\hat{f}_k = \begin{cases} E_k + e^{-\frac{2\pi i}{N} k} O_k & \text{if } k < N/2 \\ E_{k-N/2} - e^{-\frac{2\pi i}{N}(k-N/2)} O_{k-N/2} & \text{if } k \ge N/2. \end{cases}$$

This result, expressing the DFT of length N recursively in terms of two DFTs of size $N/2$, is the core of the Fast Fourier transform. The algorithm gains its speed by re-using the results of intermediate computations to compute multiple DFT outputs. Note that final outputs are obtained by a combination of E_k and $O_k e^{-2\pi i k/N}$. The overall cost of this procedure is $\mathcal{O}(N \log N)$ instead of $\mathcal{O}(N^2)$ of the direct MFT approach. This algorithm can be extended to arbitrary numbers N.

Fastest Fourier Transform in the West: a FFT library for MATLAB

In MATLAB, the FFT is implemented in a library written in C, called `FFTW` (see [3]), designed to compute the DFT in one or more dimensions, of arbitrary input size, real or complex, data sets. The acronym stands for *Fastest Fourier Transform in the West*, showing that the developers were quite optimistic on the results of their work. However, it works well and it's superior to other public available software. This library provide us some useful MATLAB's commands as `fft`, `ifft`, `fftshift`, `ifftshift`. The core of such good performance is not a special algorithm but the composition of different FFT algorithms and approaches, chosen at runtime on the basis of the type and the size of the problem. The library can also "remember" these choices, so that if many calls to `FFTW` have similar input, the first one will be slightly slower, and all the next ones will have a very high speedup. There is also a method, called wisdom, to save this information so that in a very specific task, that needs always similar transforms, it will be possible to create and distribute a wisdom just once. Moreover, also the architecture features and peculiarities are taken into account or specific instruction scheduling. In order to fully take advantage of this, part of the code is automatically generated, to produce highly optimized routines. For this purpose, a call is provided in order to allocate the memory so that these special instructions can work as fast as possible. The `FFTW` execution is divided in two parts: planning and execution. The most important is the planning, which consists of an analysis of the particular problem for choosing the best possible approach: this knowledge is called wisdom. Details such as the transform size, if the transform is "in-place" or "out-of-place", if it is a real, complex, or real-to-complex transform, are taken into account. However, the planning takes time. For this reason, different levels of planning are available, so that if the problem is small enough, or if just one transform has to be made, it is possible to avoid a long planning that would be useless. In such cases, an estimation is made in order to "guess", as quickly as possible, a good plan. On the other hand, when the problem needs many transforms of the same type, it is worth to use the planner in its whole, so that the loss of performance for the planning is well absorbed by the gain of speed in the transforms (for more info see [6]).

Remembering we are in the interval $[a, b)$ and that $y_n = (n-1)/N$, we can obtain the following algorithm to compute (2.7) and (2.9), using the `FFTW` library:

- FFT: from the space domain to the frequency domain. The coefficients $[\hat{f}_1, \ldots, \hat{f}_N]^T$ are:

$$\frac{\sqrt{b-a}}{N} \cdot \texttt{fft}\Big([f(x_1)e^{i\pi N y_1}, \ldots, f(x_n)e^{i\pi N y_N}]^T\Big)$$

or:

$$\frac{\sqrt{b-a}}{N} \cdot \texttt{fftshift}\Big(\texttt{fft}\Big([f(x_1), \ldots, f(x_n)]^T\Big)\Big). \tag{3.3}$$

Remark 3.1. The function `fftshift` translates frequency zero's value in the center of the spectrum. This is meaningfull the first Fourier coefficient is the signal's average value (it descend from the definition of $a_0/2$ or c_0) so we must translate the first frequency component in the center of our interval.

- IFFT: from the frequency domain to the space domain. The values $[\hat{\hat{f}}_1, \ldots, \hat{\hat{f}}_N]^T$ are:

$$\frac{N}{\sqrt{b-a}} \cdot \texttt{ifft}\Big([\hat{f}_1, \ldots, \hat{f}_N]\Big) \circ [e^{-i\pi N y_1}, \ldots, e^{-i\pi N y_N}]$$

or:

$$\frac{N}{\sqrt{b-a}} \cdot \texttt{ifft}\Big(\texttt{fftshift}\Big([\hat{f}_1, \ldots, \hat{f}_N]\Big)\Big). \tag{3.4}$$

3.1.3 Computational cost

If FFT, or IFFT, are computed on N equispaced evaluation points, with N power of 2, the computational cost is $\mathcal{O}(N \log_2 N)$. On the other hand, MFT, or IMFT, has a $\mathcal{O}(N^2)$ computational cost. However, these costs are asymptotic and can hide costant factors.

3.1.4 Comparison between MFT and FFT

In order to investingate the main differences between MFT and FFT, our calculations were made on a Personal Comuputer `Asus F3Jc, Intel Core Duo T2250 1,73GHz, 2MB L2 cache, 533MHz FSB` processor, running `Ubuntu 10.04 Lucid Lynx` OS and MATLAB 7.8.0.347 (R2009a). The test function is

```
f(x) = sin(2*pi*(x-a)/(b-a)) + 2*cos(4*2*pi*(x-a)/(b-a))
```

and the Figure 3.1 is the output of MATLAB's script `./MFTvsFFT.m`. From the plot, it appears that exist an N_0 (approximatively 64 on our PC) under which is computationally more convenient using the MFT instead of the FFT [2]. This is due also to the use of optimized `BLAS` (Basic Linear Algebra Subroutines) routines in matrix-vector operations. On the other hand the error, i.e. the difference between the exact value of the test function and the value found by MFT or FFT, from space domain to frequency domain and backward, evaluated in infinity norm, is always smaller using FFT rather than MFT: that is because FFT is a recursive algorithm which uses some simmetries to avoid unnecessary operations. We computed CPU Times as an average over 150 attemps.

Table 3.1: Output of `./MFTvsFFT.m`.

N Fourier coefficients	MFT Error $\|\cdot\|_\infty$	FFT Error $\|\cdot\|_\infty$
16	7.1346e-15	4.4409e-16
32	1.3950e-14	8.8818e-16
64	2.9571e-14	8.8818e-16
128	7.6440e-14	8.8818e-16
256	1.4631e-13	8.8818e-16
512	3.2998e-13	1.1102e-15
1024	7.0924e-13	1.1102e-15
2048	1.4485e-12	1.7764e-15

Remark 3.2. Note that DFT is using complex arithmetic even if the function to be transformed is real. So, if the function is real, the operations of transform and inverse transform can introduce a spurious imaginary part we can get rid of with the MATLAB command `real`.

In this test we didn't count the cost of the building of the matrix F because it has been supposed that this matrix can be build once for all.

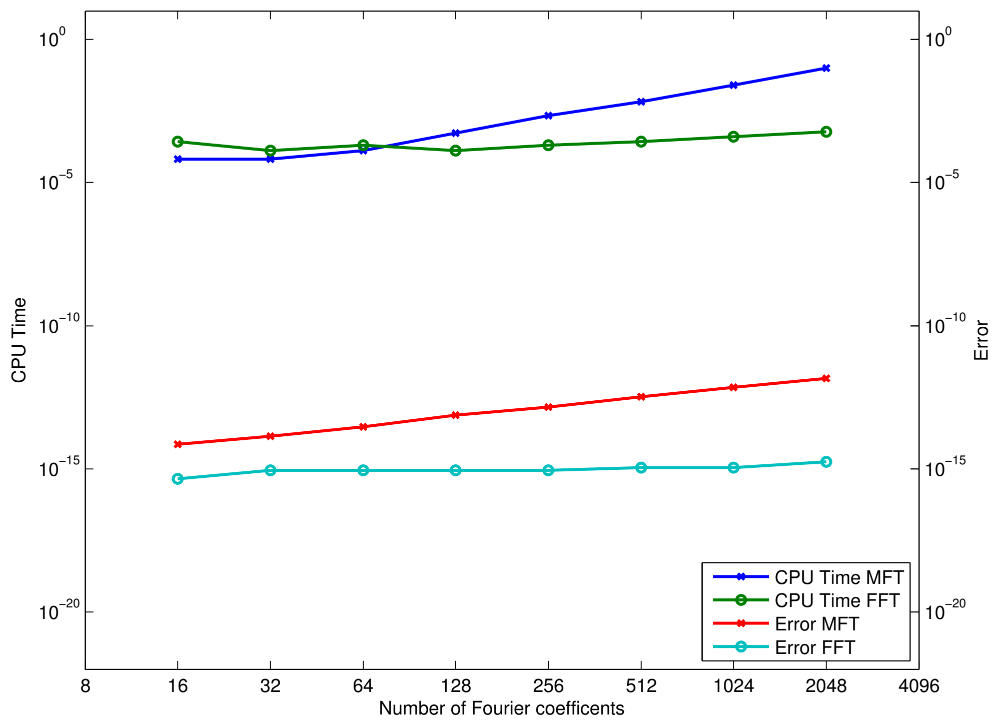

Figure 3.1: From the space domain to the frequency domain and backward. Comparison between FFT and MFT: CPU Time and Errors.

3.1.5 SpeedyFT

To improve the computational cost of DFT and IDFT in MATLAB we wrote a simple, but "clever", function `./speedyFT/speedyFT.m` for the PC where we have tested both alghoritms. This function implements the results shown above: if the input vector is length less or equal of 64 nodes (according to the results on our PC), which is the discriminating value to choose if it't better to use MFTor FFT, `./speedyFT/speedyft.m` follow MFT's way. On the other hand, if the input's vector is length more of 64 nodes, `./speedyFT/speedyft.m` follows FFT's way. Note that both functions `./speedyFT/sft.m`, to compute the DFT, and `./speedyFT/isft.m`, to compute IDFT, are "clever" themselves and that the returning values of these functions are already shifted. To improve further this algorithm, and its usage on others PC with differenct characteristics, we might use the same method of the FFTW library. For example, when the SpeedyFT is called for the first time, it is possible to test the alghoritm in order to find the switching N between MFT and FFT. Moreover it is possible to save the computed matrices a first time, from 2 to 4096 (powers of two) size, to speed up the alghoritm. When a computation of another matrix size, not previously saved, is made, it is a good idea to update the saved matrices.

3.2 M equispaced evaluation points, with $M \neq N$

This is the case with M evaluation values and N Fourier coefficients, with $M \neq N$. To test these implementations, run the MATLAB's scripts `/matrixFT/simple_test_evaluation.m` (for MFT) and `/fastFT/simple_test_evaluation.m` (for FFT).

3.2.1 MFT: Matrix Fourier Transform

- $M > N$: We can compute the Fourier coefficients in the traditional way (3.1) and compute (2.9) with the basis functions evaluated on an arbitrary set of equispaced points, i.e. (2.8) formula:

$$\hat{\hat{f}}_k = \sum_{n=1}^{N} \hat{f}_n \phi_n(x_k), \text{ with } k = 1, \dots, M$$

- $M < N$: There is no difference with the case discussed above.

Simply we can use an arbitrary number of equispaced points on where our function will be evaluated. So we will use a rectangular matrix in the matrix-vector operation.

3.2.2 FFT: Fast Fourier Transform

- $M > N$: We can compute the Fourier coefficients in the traditional way (3.3) and compute the IFFT with the following input:

$$\hat{f}^* = [\underbrace{0, \dots, 0,}_{\frac{\text{M}-\text{N}}{2} \text{ items}} \hat{f}, \underbrace{0, \dots, 0}_{\frac{\text{M}-\text{N}}{2} \text{ items}}].$$

- $M < N$: Given N Fourier coefficients, we can compute the IFFT on a different set of M equispaced points using with the following Fourier coefficients:

$$\hat{f}^* = [\underbrace{\hat{f}_1, \dots, \hat{f}_{\frac{M-N}{2}}}_{\text{items to be deleted}}, \hat{f}_{\frac{M-N}{2}+1}, \dots, \hat{f}_{\frac{M+N}{2}}, \underbrace{\hat{f}_{\frac{M+N}{2}+1}, \dots, \hat{f}_N}_{\text{items to be deleted}}].$$

In this case the only restriction is to not delete too many frequencies to obtain an acceptable result.

Chapter 4

Non Equispaced DFT and FFT

In Chapter 3 we discussed about the implementation in MATLAB of the DFT on a set of equispaced nodes. What if the nodes are not equispaced? This is what we want to investigate in this chapter because the limit of the FFT algorithm is that is based on symmetries which hold only with equispaced nodes. Before starting, when we talk about "random nodes", we must remember that we have chosen to implement them in MATLAB with the command `rand` with an arbitrary fixed seed (choosen `s=3` in our tests) to be able to repeat more than once experiments on the same "random" set.

4.1 The NDFT and NFFT library

Stephan Kunis wrote a dissertation thesis [5] about the computation of the *Nonequispaced Discrete Fourier Transform* (NDFT) and the *Nonequispaced Fast Fourier Transform* (NFFT), from the space domain to the frequency domain and backward, on a set of nonequispaced nodes. He used the *Fourier Transform* with a slightly different notation: we will see in Section 4.2 how to uniform his notations with the ours. Moreover he provided a library, written in C, which help us to compute the NDFT and the NFFT [7]. In Sections 4.1.1 and 4.1.2 we briefly show what this library is able to do in order to compute NDFT and NFFT.

4.1.1 NDFT: Non Equispaced Discrete Fourier Transform

The idea underlaying the NDFT is the same idea of MFT: both methods use the standard matrix-vector multiplication:

- NDFT: from the space domain to the frequency domain, on $x_n \in [-\frac{1}{2}, \frac{1}{2})$. The coefficients $[\hat{f}_1, \ldots, \hat{f}_N]^T$, defined by Kunis, are

$$\hat{f}_n = \sum_{k=-N/2}^{N/2-1} f(x) e^{i2\pi k x_n}, \tag{4.1}$$

  or, in MATLAB:

```
f_hat = sqrt(b-a)*exp(1i*2*pi*[-N/2:N/2-1].'*y.')*f(x)/N;.
```

- INDFT: from the frequency domain to the space domain on M (in general $\neq N$) nonequispaced points $y \in [-\frac{1}{2}, \frac{1}{2})$. The values $[\hat{\hat{f}}_1, \dots, \hat{\hat{f}}_M]^T$, defined by Kunis, are

$$\hat{\hat{f}}_m = \sum_{k=-N/2}^{N/2-1} \hat{f}_k e^{-i2\pi k y_m}, \tag{4.2}$$

or, in MATLAB:

```
f_hat.'  * exp(-1i*2*pi*[-N/2:N/2-1].'*y.')/sqrt(b-a);.
```

All these procediments are implemented in the MATLAB's script `./nFT/ndft.m` and `./nFT/indtf.m`.

4.1.2 NFFT: Non Equispaced Fast Fourier Transform

As we will see in Section 4.1.3, NDFT (with its *Inverse*) requires a too high computational cost: this is why some alghorithms have been recently developed in order to overcome this situation. `NFFT` library, written in C, implements one of them. This is library is avaible on the web [7] with a mexfile which lets us to use these functions in MATLAB.

The `NFFT` library, unlike the `FFTW` library (on which it is anyway based), is not a method that optimizes and provides a complete exploitation of the machine features, but simply is a new approximated algorithm that allows to bypass the bottleneck of the direct summation.

From now on, we speak about NFFT referring to the trasformation from given complex values in frequency domain to values in space domanin. The transformation from the space domain to the frequency domain is a problem of higher complexity (and out of our aims) than finding, *quickly*, the values in the space domain on nonequispaced points from given complex values.

Let us to assume $d = 1$ our dimension. Given

- a set of nonequispaced nodes $x_j \in [-\frac{1}{2}, \frac{1}{2})$, with $j = 1, \dots, N$,
- a set of possible frequencies $I_N = [-\frac{N}{2}, \frac{N}{2})$,
- an oversampling factor $\sigma > 1$, setting $n = \sigma N$,

the main idea is that $f(x)$ can be approximated with a linear combination of shifted 1-periodic window functions, that is:

$$f(x) = \sum_{k \in I_N} \hat{f}_k e^{-2\pi i k x} \approx s_1(x) = \sum_{l \in I_n} g_l \tilde{\varphi}\Big(x - \frac{l}{n}\Big) \tag{4.3}$$

Hence, starting with a (reasonable) window function $\varphi : \mathbb{R} \to \mathbb{R}$, its 1-periodic version is given by

$$\tilde{\varphi}(x) = \sum_{r \in \mathbb{Z}} \varphi(x + r).$$

The first approximation: cut-off in frequency domain

Using the property of the Fourier transform, we can switch to the frequency domain, so we get

$$s_1(x) = \sum_{k\in\mathbb{Z}} \hat{g}_k \hat{\varphi}_k e^{-2\pi ikx} = \sum_{k\in I_n} \hat{g}_k \hat{\varphi}_k e^{-2\pi ikx} + \sum_{r\in\mathbb{Z}\setminus\{0\}} \sum_{k\in I_n} \hat{g}_k \hat{\varphi}_{k+nr} e^{-2\pi i(k+nr)x}, \tag{4.4}$$

with

$$\hat{g}_k = \sum_{l\in I_n} g_l e^{2\pi ik\frac{l}{n}} \tag{4.5}$$

and the Fourier coefficients $\hat{\varphi}_k$ defined as

$$\hat{\varphi}_k = \int_{-1/2}^{1/2} \tilde{\varphi}(x) e^{2\pi ikx} \mathrm{d}x, \quad k \in \mathbb{Z}. \tag{4.6}$$

If $\hat{\varphi}_k$ are small enough for $k \in \mathbb{Z} \setminus I_n$, and if $\hat{\varphi}_k \neq 0$ for $k \in I_N$, then comparing (4.2) and (4.4) we can set

$$\hat{g}_k = \begin{cases} \hat{f}_k / \hat{\varphi}_k & \text{if } k \in I_N \\ 0 & \text{if } k \in I_n \setminus I_N \end{cases} \tag{4.7}$$

Now, each value g_l can be obtained using an IFFT of size $n = \sigma N$ (where the `FFTW` library is now used):

$$g_l = \frac{1}{n} \sum_{k\in I_N} \hat{g}_k e^{-2\pi ik\frac{l}{n}}, \quad \text{with } l \in I_n. \tag{4.8}$$

We immediatly observe that this first part of the approximation introduces an *aliasing error*.

The second approximation: cut-off in time/space domain

If φ is well localized in time domain, it can be approximated by a function

$$\psi(x) = \varphi(x)\chi_{[-\frac{m}{n},\frac{m}{n}]}(x)$$

with support in $[-\frac{m}{n}, \frac{m}{n}]$ and $m \in \mathbb{N}$, $m \ll n$. Again, we define its 1-periodic version as

$$\tilde{\psi} = \sum_{r\in\mathbb{Z}} \psi(x + r).$$

In this way, we define an approximation of $s_1(x)$ by

$$f(x_j) \approx s_1(x_j) \approx s(x_j) = \sum_{l\in I_{n,m}(x_j)} g_l \tilde{\psi}\Big(x_j - \frac{l}{n}\Big), \tag{4.9}$$

where $I_{n,m}(x_j) = \{l \in I_n : nx_j - m \leq l \leq nx_j + m\}$. For nodes $x_j \in [-\frac{1}{2}, \frac{1}{2})$ the above sum contains at most $2m + 1$ nonzero summands. This second part introduces a *truncation error*.

The algorithm

To summarize, given the input values $N \in \mathbb{N}$, $\sigma > 1$, $n = \sigma N$, $x_j \in [-\frac{1}{2}, \frac{1}{2})$ and $\hat{f}_k \in \mathbb{C}$ the algorithm can be described as follows.

1. Precompute $\hat{\varphi}_k$, with $k \in I_N$.
2. Precompute $\psi(x_j - \frac{l}{n})$, with $l \in I_{n,m}(x_j)$.
3. Generate $\hat{g}_k = \hat{f}_k / \hat{\varphi}_k$, with $k \in I_N$.
4. Compute g_l using a d-variate FFT, in our case $d = 1$ (at this point, `FFTW` library is used).
5. Set
$$s(x_j) = \sum_{l \in I_{n,m}(x_j)} g_l \tilde{\psi}\Big(x_j - \frac{l}{n}\Big).$$

The values $s(x_j)$ computed in the last step, are the output of the algorithm which represent the approximated value of $f(x_j)$.

4.1.3 Computational Cost

The NDFT can be directly implemented by MFT and requires $\mathcal{O}(NM)$ steps, with N representing the number of Fourier coefficients and M the number of nodes: such computational complexity is simply too high for most practical applications. The overall complexity of NFFT for a d-variate transform is $\mathcal{O}((\sigma N)^d \log(\sigma N) + m^d M)$ since the d-variate FFT requires $\mathcal{O}((\sigma N)^d \log(\sigma N))$ steps and the evaluation of the sum (4.9) requires $\mathcal{O}(mM)$ steps.

4.1.4 Error estimation

Since this is an approximated algorithm, we have to give also an estimate of the error (see [1]) which will be estimated by a sum of the *aliasing error* (E_a) and the *truncation error* (E_t). According to (4.9), the total error is splittend in two parts:

$$E(x_j) = |f(x_j) - s(x_j)| \leq E_a(x_j) + E_t(x_j) = C(\sigma, m)||\hat{f}||_1,$$

with $C(\sigma, m)$ depending on the window function choosen and $||\hat{f}||_1 = \sum_{k \in I_N} |\hat{f}_k|$.

4.1.5 Some window functions and their $C(\sigma, m)$

Now we present some window functions and their Fourier coefficients (see [4]).

Dilated Gaussian

$$\varphi(x) = (\pi b)^{\frac{1}{2}} e^{-\frac{(nx)^2}{b}},$$
$$\hat{\varphi}_k = \frac{1}{n} e^{-b(\frac{\pi k}{n})^2}.$$

A good choice parameter b is

$$b = \frac{2\sigma}{(2\sigma - 1)} \frac{m}{\pi}$$

as suggested in [1], and

$$C(\sigma,m) = 4e^{-m\pi(1-\frac{1}{2\sigma-1})}.$$

Cardinal central B-splines

$$\varphi(x) = M_{2m}(nx),$$
$$\hat{\varphi}_k = \frac{1}{n}\mathrm{sinc}^{2m}\Big(k\frac{\pi}{n}\Big),$$

with

$$C(\sigma,m) = 4\Big(\frac{1}{2\sigma-1}\Big)^{2m}.$$

Dilated Keiser–Bessel functions

$$\varphi(x) = \frac{1}{\pi}\begin{cases} \dfrac{\sinh(b\sqrt{m^2-n^2x^2})}{\sqrt{m^2-n^2x^2}} & \text{for } |x| \le \dfrac{m}{n}, \quad \text{with } b = \pi\Big(2-\dfrac{1}{\alpha}\Big), \\ \dfrac{\sinh(b\sqrt{n^2x^2-m^2})}{\sqrt{n^2x^2-m^2}} & \text{otherwise,} \end{cases}$$

$$\hat{\varphi}_k = \frac{1}{n}\begin{cases} I_0\left(m\sqrt{b^2-\Big(\dfrac{2\pi k}{n}\Big)^2}\right) & k = -n\Big(1-\dfrac{1}{2\sigma}\Big),\dots,n\Big(1-\dfrac{1}{2\sigma}\Big), \\ 0 \quad \text{otherwise,} \end{cases}$$

with

$$C(\sigma,m) = 4\pi(\sqrt{m}+m)\sqrt[4]{1-\frac{1}{\sigma}}e^{-2\pi m\sqrt{1-\frac{1}{\sigma}}}.$$

Remark 4.1. It is important to underline that the Dilated Keiser-Bessel functions are kept as default from the NFFT library if any window function is specified.

4.2 How to match Kunis' notations with the ours

Given x_j, an arbitrary set of nodes on the interval $[a,b)$, we want to adapt Kunis' definitons (4.1) and (4.2) to our way to intend the *Fourier Transform*. The main differences between the Kunis' notations and the ours are two:

- Kunis' nodes are defined on $[-\frac{1}{2},\frac{1}{2})$ but in our notation the nodes are defined on $[0,1)$.
- DFT and IDFT (or NDFT and INDFT) in Kunis' notation presents a "minus" factor in the exponential.

How to fix this issue? Here's an useful scheme:

- from our to Kunis' notation: given $x \in [a,b)$ we can use Kunis' scripts with
$$y = -\Big(\frac{x-a}{b-a} - 0.5\Big);$$
- from Kunis' to our notation: given $y \in [-\frac{1}{2},\frac{1}{2}]$ we can use our scipts with
$$x = -(b-a)(y-0.5)+a.$$

4.3 Comparison between MFT, FFT and NFFT

Now it seems useful to make a comparison between the MFT, FFT and the NFFT from the frequency domain to the space domain. We said that FFT works only for equispaced points: in order to make an useful comparison of CPU Time and Errors, nobody forbid us to choose equispaced points also for MFT and NFFT. This is the aim of the script `./MFTvsFFTvsNFFT.m`, based on the same test function of Section 3.1.4 and whose output is the Figure 4.1. Once again, the error computed in our test is the difference between the exact value of the test function and the value found by MFT, FFT and NFFT, evaluated in infinity norm.

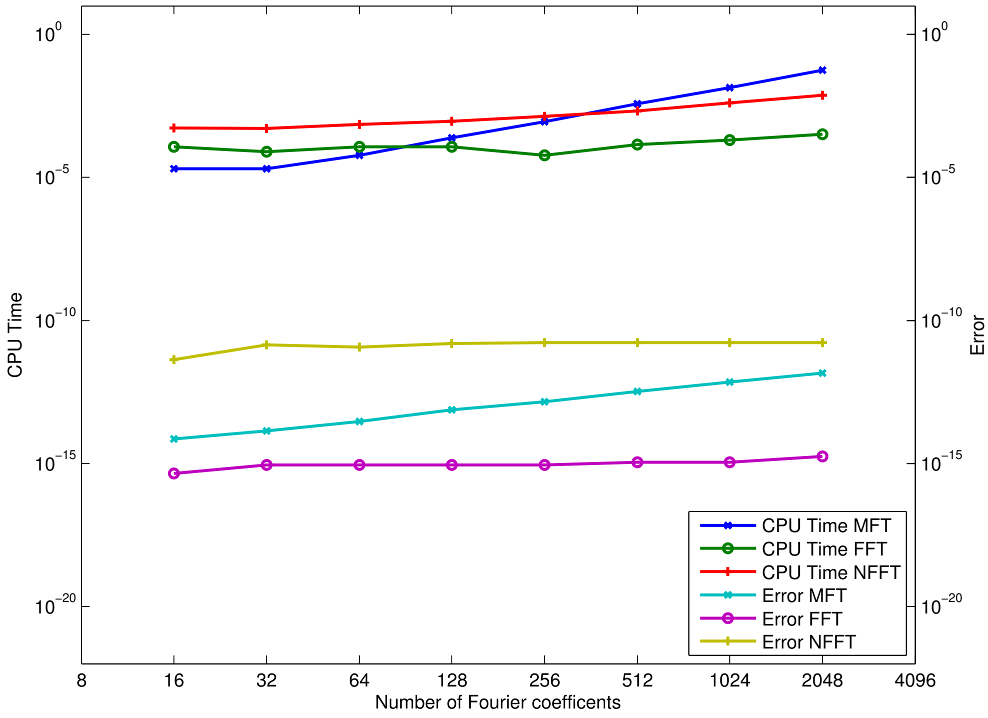

Figure 4.1: From the frequency domain to the space domain. Comparison between MFT, FFT and NFFT: CPU Time and Errors

Table 4.1: Output of `./MFTvsFFTvsNFFT.m`.

N Fourier coefficients	MFT Error $\|\|\cdot\|\|_\infty$	FFT Error $\|\|\cdot\|\|_\infty$	NFFT Error $\|\|\cdot\|\|_\infty$
16	7.1346e-15	4.4409e-16	4.3396e-12
32	1.3950e-14	8.8818e-16	1.4065e-11
64	2.9571e-14	8.8818e-16	1.1525e-11
128	7.6440e-14	8.8818e-16	1.5687e-11
256	1.4631e-13	8.8818e-16	1.6717e-11
512	3.2998e-13	1.1102e-15	1.6957e-11
1024	7.0924e-13	1.1102e-15	1.7022e-11
2048	1.4485e-12	1.7764e-15	1.7042e-11

4.4 Conclusions

In conclusion, this comparison show some interesting results:

- NFFT is faster than MFT;
- even if, in this case, FFT is faster than NFFT, we recall that FFT works *only* for equispaced points: NFFT works for any set of points, equispaced or not;
- The error using NFFT is higher than MFT and FFT but its order, in our test, is acceptable.

It's clear that the NFFT is slightly more expensive than FFT in computation time and not so accurate as the FFT. However the difference on computation time used and on the errors made are not so big despite of the great advantage to manage also nonequispaced nodes: that's way the NFFT algorithm is very powerful.

Chapter 5

NFFT in solving hyperbolic problems

Consider the follow equation

$$\begin{cases} \dfrac{\partial u}{\partial t} + a\dfrac{\partial u}{\partial x} = 0 & x \in \mathbb{R},\ t > 0 \\ u(x,0) = u_0(x) & x \in \mathbb{R} \end{cases} \tag{5.1}$$

with $a \neq 0$ costant, called *transport coefficient.* The solution of this problem in a point x is a wave travelling at the speed a given by

$$u(x,t) = u_0(x - at). \tag{5.2}$$

The curves $x(t)$ on the plane (x,t), solution of the ordinary differential equation (ODE)

$$\begin{cases} \dfrac{\mathrm{d}x}{\mathrm{d}t} = a & t > 0 \\ x(0) = x_0, & x_0 \in \mathbb{R}, \end{cases} \tag{5.3}$$

are called *characteristics curves* and the solution is *constant* along them because

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}t}u(x,t) &= \frac{\mathrm{d}t}{\mathrm{d}t}\frac{\partial u}{\partial t}(x,t) + \frac{\mathrm{d}x}{\mathrm{d}t}\frac{\partial u}{\partial x}(x,t) = \\ &= \frac{\partial u}{\partial t}(x,t) + a\frac{\partial u}{\partial x}(x,t) = 0. \end{aligned}$$

In general:

$$\begin{cases} \dfrac{\partial u}{\partial t} + a(x,t)\dfrac{\partial u}{\partial x} + a_0(x,t)u = f(x,t) & x \in \mathbb{R},\ t > 0 \\ u(x,0) = u_0(x) & x \in \mathbb{R} \end{cases} \tag{5.4}$$

where $a(x,t), a_0, f$ are assigned functions with the variables (x,t). The *Characteristics Curves* $x(t)$ are the solutions of Cauchy's problem

$$\begin{cases} \dfrac{\mathrm{d}x}{\mathrm{d}t} = a(x,t) & t > 0 \\ x(0) = x_0, & x_0 \in \mathbb{R}, \end{cases} \tag{5.5}$$

In this case, the solutions of (5.4) satisfy the ODE

$$\frac{\mathrm{d}}{\mathrm{d}t}u(x(t),t) = f(x(t),t) - a_0(x(t),t)u(x(t),t).$$

So, it's possible to find the solution solving an ODE over every *characteristic curve.* This approach is called *method of characteristics.*

5.1 Solution of an hyperbolic PDE on equispaced nodes

We suppose that $f(x) \equiv a_0(x,t) \equiv 0$ and the transport coefficient $a(x,t)$ is a 2π-periodic function. Given a set of equispaced nodes $\{x_e\}_e$, an initial solution $u_0(x)$ and a problem of type (5.1), our aim now is to find the solution through the method of characteristic at time t_f on $\{x_e\}_e$. This is not a trivial question because, when $a(x,t) \neq c$, the new nodes, solution of (5.5) are not, in general, equispaced so we are only able to build the solution at time t_f on a set of nodes different from those of interest.

Forward Characteristic

Forward characteristic is the standard method to solve the ODE (5.5), providing us, in general, a set of nonequispaced nodes called $\{x_{ne}\}_{ne}$. We can find the solution of the hyperbolic PDE at the time t_f only on $\{x_{ne}\}_{ne}$, different from those of interest, evaluating $u_0(x_s)$, with $x_s \in \{x_e\}_e$, and then making the assignment $u(x_n, t_f) = u_0(x_s)$, with x_n solution of (5.5) with initial value x_s. To find the solution at the time t_f on the set of equispaced nodes $\{x_e\}_e$ we should interpolate $u(x_n, t_f)$, $x_n \in \{x_{ne}\}_{ne}$, and rebuild the values on $\{x_e\}_e$. We could use NFFT from the space domain to the frequency domain and then use the IFFT from the frequency domain to the space domain in order to be able to perform the next timestep: however, we said that switching from the space domain to the frequency domain by NFFT is a difficult task.

Backward Characteristic

The right approach to reach our aim is considering the initial set of equispaced nodes $\{x_e\}_e$ as the nodes reached at the final time t_f. So we want to investigate from which points, called again $\{x_{ne}\}_{ne}$, to remark that in general they are nonequispaced, $\{x_e\}_e$ are coming from. This is the meaning of solving the system (5.5) backward, in time. It's easy to find out the new ODE to be solved. Given an interval $[t_0, t_f]$, a function $a(x(t), t)$ and the ODE

$$\begin{cases} \dfrac{\mathrm{d}x}{\mathrm{d}t} = a(x(t), t) \\ x(t_f) = x_s, \qquad x_s \in \{x_e\}_e \end{cases}$$

we want to find out the solution $x(t_0)$. Changing the variables in this way

$$\begin{cases} \tilde{t} = -t \\ \tilde{x}(\tilde{t}) = x(t) \end{cases} \quad \text{we have} \quad \begin{cases} t = -\tilde{t} \\ x(t) = \tilde{x}(-t). \end{cases}$$

So the new ODE to be solved is

$$\begin{cases} \dfrac{\mathrm{d}\tilde{x}}{\mathrm{d}\tilde{t}} = \dfrac{\mathrm{d}\tilde{x}}{\mathrm{d}t}\dfrac{\mathrm{d}t}{\mathrm{d}\tilde{t}} = \dfrac{\mathrm{d}\tilde{x}}{\mathrm{d}t} \cdot (-1) = -a(\tilde{x}(\tilde{t}), -\tilde{t}) = -a(\tilde{x}(t), t) \\ \tilde{x}(-t_f) = x_s. \end{cases} \text{so} \begin{cases} \dfrac{\mathrm{d}\tilde{x}}{\mathrm{d}t} = -a(\tilde{x}(t), t) \\ \tilde{x}(-t_f) = x_s. \end{cases} \tag{5.6}$$

Now it's possible to *solve the new ODE (5.6) forward in time*, with the meaning that solving forward in time is equal, in the original ODE, to have assigned the values at the time t_f and we want to find out the values at the time t_0. Relabelling $\tilde{x}$ with x we have:

$$\begin{cases} \dfrac{\mathrm{d}x}{\mathrm{d}t} = -a(x,t) \qquad t \in [-t_f, -t_0] \\ x(-t_f) = x_s, \qquad x_s \in \{x_e\}_e. \end{cases} \tag{5.7}$$

Now with the solution of (5.7), called for our conveniency $\{x_{ne}\}_{ne}$ and in general nonequispaced as we said, we can find the values at the time t_0 on $\{x_{ne}\}_{ne}$ so that

$$u(x_s, t_f) = u_0(x_n), \tag{5.8}$$

with $x_n \in \{x_{ne}\}_{ne}$ solution of (5.7) with initial value x_s. The meaning of (5.8) is that the values $u_0(x_n)$ are the values of the solution at the final time t_f on the set of equispaced nodes called $x_s \in \{x_e\}_e$. To solve this ODE we can use a Runge–Kutta–Fehlberg45 method, of order 5, which is adapative in time with

```
dt = min(2,max(0.6,abs(0.9*(tol/norm(err))^(1/5))))*dt;
```

and should reach the error required (in our tests 10^{-15}). This method is implemented in `./hyper/rk45.m` (in table (5.1) we can see the Butcher matrix, with Fehlberg coefficients) [8].

Table 5.1: Butcher matrix, with Fehlberg coefficients

c	a_1	a_2	a_3	a_4	a_5	a_6
0	0	0	0	0	0	0
$\frac{1}{4}$	$\frac{1}{4}$	0	0	0	0	0
$\frac{3}{8}$	$\frac{3}{32}$	$\frac{9}{32}$	0	0	0	0
$\frac{12}{13}$	$\frac{1932}{2197}$	$-\frac{7200}{2197}$	$\frac{7296}{2197}$	0	0	0
1	$\frac{439}{216}$	-8	$\frac{3680}{513}$	$-\frac{845}{4104}$	0	0
$\frac{1}{2}$	$-\frac{8}{27}$	2	$-\frac{3544}{2565}$	$\frac{1859}{4104}$	$-\frac{11}{40}$	0
b	$\frac{25}{216}$	0	$\frac{1408}{2565}$	$\frac{2197}{4104}$	$-\frac{1}{5}$	0
$\hat{b}$	$\frac{16}{135}$	0	$\frac{6656}{12825}$	$\frac{28561}{56430}$	$-\frac{9}{50}$	$\frac{2}{55}$
Error	$\frac{1}{360}$	0	$-\frac{128}{4275}$	$-\frac{2197}{75240}$	$\frac{1}{50}$	$\frac{2}{55}$

When the initial function $u_0(x)$ is given, it is easy to find $u_0(x_n)$, but when we know only the initial values *on* the equispaced nodes it is more difficult. To proceed in next timesteps it is very useful using FFT from the space domain to the frequency domain and the NFFT from the frequency domain to the space domain: in this way we are always able to know the values of the initial function everywhere in our space domain becuase the NFFT helps us to rebuild the initial function on the nonequispaced set found by solving (5.7).

This is an useful scheme to summarize:

1. Are given:
   - a set of $x_s \in \{x_e\}_e$, equispaced nodes;
   - initial values $u_0(x_s)$;
   - a problem of type (5.1).
2. We can find $x_n \in \{x_{ne}\}_{ne}$, set of nonequispaced nodes in general, solution of the ODE (5.7) with initial value x_s.

3. Using the FFT (3.3), from space domain to frequency domain on the initial values $u_0(x_s)$, lets us to compute the Fourier coefficients of the initial function.

4. Using the NFFT (4.9), from frequency domain to space domain on the set of nonequispaced nodes $\{x_{ne}\}_{ne}$, lets us to rebuild the initial value $u_0(x_n)$.

5. Now $u_0(x_n)$ is the value of the solution of the hyperbolic PDE (5.1) on the node x_s at the final time t_f, i.e. $u(x_s, t_f)$.

Example 5.1. In order to test how good is the method we can try to solve an 1D hyperbolic PDE which solution is known [9]. Given the 1D hyperbolic PDE:

$$\begin{cases} \dfrac{\partial u}{\partial t} - \sin(x)\dfrac{\partial u}{\partial x} = 0 & x \in [0, 2\pi),\ t \in (0, 1.571] \\ u(x, 0) = \sin(x) & x \in [0, 2\pi), \end{cases} \tag{5.9}$$

for which the exact solution is $u(x,t) = \sin\left(2\tan^{-1}\left(e^t \tan\dfrac{x}{2}\right)\right)$, we can solve the ODE

$$\begin{cases} \dfrac{\mathrm{d}x}{\mathrm{d}t} = \sin(x) & x \in [0, 2\pi),\ t \in (0, 1.571] \\ x(0) = x, & x \in [0, 2\pi) \end{cases} \tag{5.10}$$

in order to find from which nodes the equispaced nodes x are coming from backward in time.

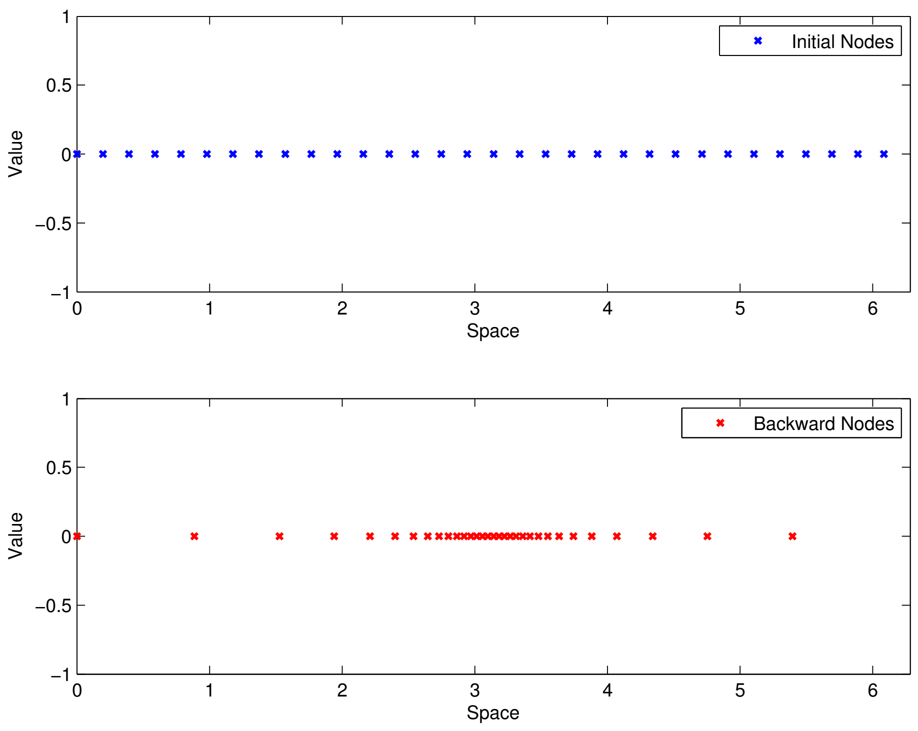

Figure 5.1: Location of the new nodes, solution of ODE (5.10)

.

Now, introduced an equispaced discretization of the nodes in $[a, b)$ we can solve in MATLAB the ODE (5.10): the solution, found with `./nFT/rk45.m`, is shown in Figure 5.1.

The top plot shows the location of the given equispaced nodes. The plot below shows the location of the new nodes, solution of the ODE (5.10). Then, calling the NFFT with input Fourier coefficients computed on the initial equispaced sampling nodes (using FFT) and with the new nodes, takes us to find out the values of the initial function, which we didn't know before, at the new nonequispaced nodes: these values will be the values of the solution $u(x,t)$ on the initial equispaced sampling nodes at the time t required. Note that, in general, the interval where the new nodes lie on might be not the same of the initial interval $[a,b)$ so we have to use the MATLAB command `mod(xn,b-a)`, where `xn` are the new nodes, solution of (5.10).

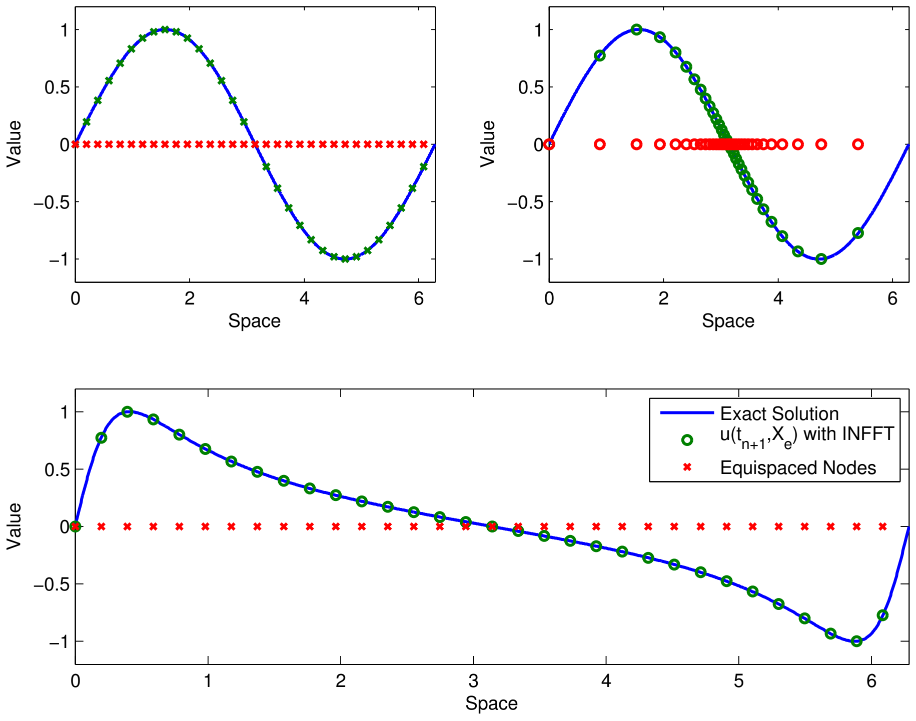

Figure 5.2: Output of `./hyper/nfft4hyper1dex1.m`.

In the Figure 5.2 we can see three plots: the top right plot shows the initial values and the initial equispaced nodes, the top left plot shows the values computed with NFFT from the frequency domain to the space domain on the nonequispaced nodes, solution of (5.7) and the bottom plot is the final solution on the initial equispaced nodes at the time $t = 1.571$. Note that the blue line is always referred to the reference values not in a discrete time but in continuous time. The error, computed as the difference between the exact solution, given by analysis, and the computed solution, evaluated in infinity norm, is 1.7148e-12.

Example 5.2. We can test our method on another PDE equation [9]:

$$\begin{cases} \dfrac{\partial u}{\partial t} - \dfrac{1}{2+\cos(x)}\dfrac{\partial u}{\partial x} = 0 & x \in [0, 2\pi),\ t \in (0, 50.27] \\ u(x,0) = \sin(x) & x \in [0, 2\pi), \end{cases} \tag{5.11}$$

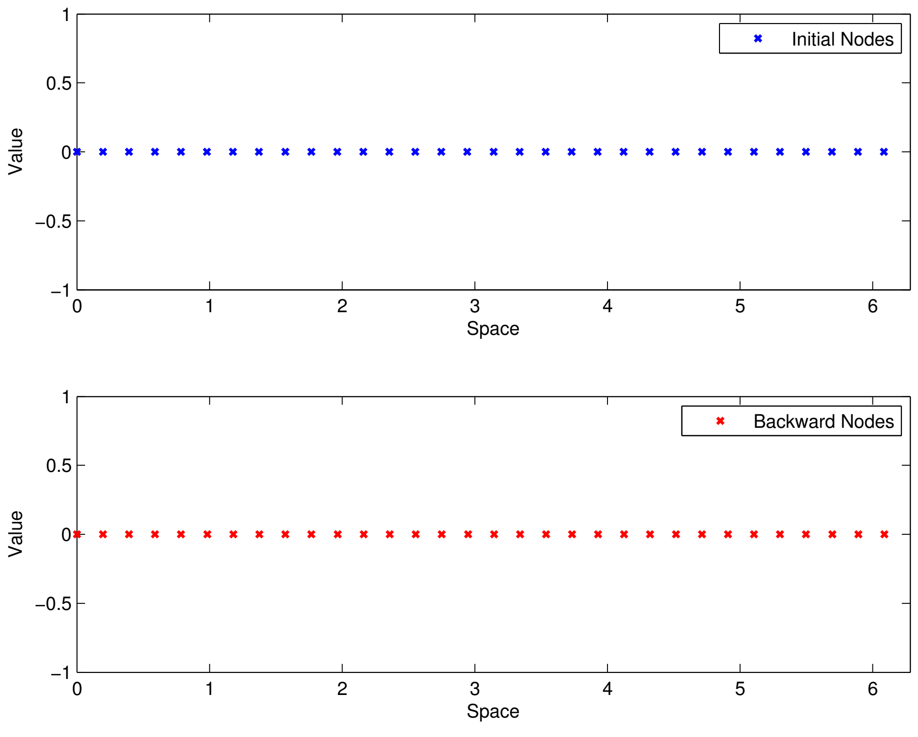

Figure 5.3: Location of the new nodes, solution of ODE (5.12)

.

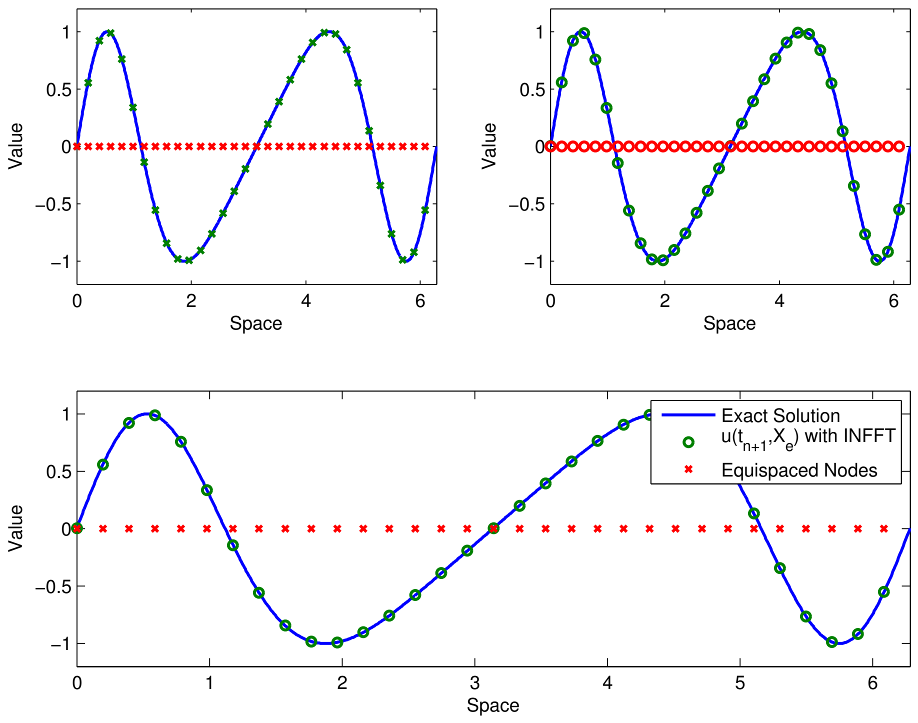

Figure 5.4: Output of `./hyper/nfft4hyper1Dex2.m`.

for which the exact solution is $u(x,t) = \sin(2x + \sin x + t)$, we can solve the ODE

$$\begin{cases} \dfrac{\mathrm{d}x}{\mathrm{d}t} = \dfrac{1}{2+\cos(x)} & t \in (0, 50.27] \\ x(0) = x, & x \in [0, 2\pi), \end{cases} \tag{5.12}$$

in order to find from which nodes the nodes x are coming. The meanings of Figures 5.3 and 5.4 are the same of Figures 5.1 and 5.2, respectively. This is not a trivial example because this is the case when the interval, where the new nodes lie on, does not correspond to the same initial interval $[a, b)$, due to transport coefficient. So we have to use the MATLAB command `mod(xn,b-a)`, with `xn` solution of (5.12), to find the solution on the nodes of interest. The error, computed as the difference between the exact solution, given by analysis, and the computed solution, evaluated in infinity norm, is 1.1516e-12.

Remark 5.1. In both examples we required a tolerance of 10^{-15} for `rk45.m`: from our tests we can conclude that this target tolerance is too high to be required despite of the error committed of order 10^{-12}: this is due to the use of `NFFT` algorithm, which order of tolerance is between 10^{-12} and 10^{-11} (see Section 4.3).

Chapter 6

Conclusions

The aim of our thesis has been reached: our tests in MATLAB showed that

- if the nodes are equispaced, the *Fast Fourier Transform* (FFT) is the most accurate transformation we can do, in terms of accuracy and the best in terms of computational cost (from a certain number N of coefficients);
- if the nodes are not equispaced, the *Nonequispaced Fast Fourier Transform* (NFFT) is faster than the direct computation and only slightly less accurate.

NFFT is very powerful in some applications like finding a solution of an hyperbolic Partial Differential Equations, with a periodic transport coefficient.

Bibliography

[1] J.J. Benedetto and P. Ferreira, *Modern Sampling Theory: Mathematics and Applications*, ch. 12, p. 249–274, Birkhäuser, 2001, available online at `http://www.tu-chemnitz.de/~potts/paper/ndft.pdf`.

[2] John P. Boyd, *Chebyshev and Fourier Spectral Methods: Second Revised Edition*, Dover Publications, December 2001.

[3] Matteo Frigo and Steven G. Johnson, *The FFTW web page*, available online at `http://www.fftw.org`.

[4] S. Kunis J. Keiner and D. Potts, *NFFT 3.0 - Tutorial*, 2006, available online at `http://www-user.tu-chemnitz.de/~potts/nfft/guide/nfft3.pdf`.

[5] Stephan Kunis, *Nonequispaced FFT, Generalisation and Inversion*, 2006, available online at `http://www-user.tu-chemnitz.de/~skunis/paper/KunisDiss.pdf`.

[6] Roberto Montagna, *Hyperinterpolation at XU points and interpolation at Padua points in the square: computational aspects*, 2004.

[7] Daniel Potts, Jens Keiner, and Stefan Kunis, *The NFFT webpage*, 2009, available online at `http://www-user.tu-chemnitz.de/~potts/nfft/download.php`.

[8] Alfio Quarteroni, Riccardo Sacco, and Fausto Saleri, *Matematica Numerica*, Springer, Milano, 2008.

[9] Endre Süli and Antony Ware, *A Spectral Method of Characteristics for Hyperbolic Problems*, SIAM J. Numer. Anal. **28** (1991), no. 2, 423–445.