Time and memory requirements of the Nonequispaced FFT

Stefan Kunis* Daniel Potts†

We consider the fast Fourier transform at nonequispaced nodes (NFFT) and give detailed information on the time and memory requirements of its building blocks. This manuscript reviews the state of the art approaches and focuses within the most successful scheme on the computational most involved part. Beside a rigorous derivation of an lookup table technique, we compare a wide range of precomputation schemes which lead to substantially different computation times of the NFFT. In particular, we show how to balance accuracy, memory usage, and computation time.

Key words and phrases : Nonequispaced Fast Fourier Transform, FFT

2000 AMS Mathematics Subject Classification —

1 Introduction

This paper summarises algorithms for the discrete and fast Fourier transform at nonequispaced nodes. Generalising the famous fast Fourier transform (FFT), we evaluate for $N \in 2\mathbb{N}$, $M \in \mathbb{N}$, a vector of coefficients $\hat{\boldsymbol{f}} = (\hat{f}_k)_{k=-\frac{N}{2},\ldots,\frac{N}{2}-1} \in \mathbb{C}^N$, and a set of nodes $x_j \in \mathbb{R}$ the sums

$$f_j = \sum_{k=-\frac{N}{2}}^{\frac{N}{2}-1} \hat{f}_k \mathrm{e}^{-2\pi \mathrm{i} k x_j}, \qquad j = 0, \ldots, M-1.$$

In the multivariate setting, the discrete Fourier transform at nonequispaced nodes (NDFT) requires $\mathcal{O}(N_\pi M)$ operations for N_π equispaced frequencies and M nonequispaced sampling nodes $\boldsymbol{x}_j \in \mathbb{R}^d$. In contrast, the approximate NFFT takes only $\mathcal{O}(N_\pi \log N_\pi + M)$ floating point operations (flops), where the constant of proportionality depends in theory solely on the prescribed target accuracy and on the space dimension d.

*kunis@math.uni-luebeck.de, University of Lübeck, Institute of Mathematics, 23560 Lübeck, Germany
†potts@mathematik.tu-chemnitz.de, Chemnitz Universitiy of Technology, Department of Mathematics, 09107 Chemnitz, Germany

There is a variety of important applications which utilise the NDFT, e.g. in computerised tomography, for fast summation algorithms [25], as fast Fourier transform on the sphere [19], or as part of the ridgelet and curvelet transforms [11, 5]. Furthermore, the reconstruction from nonuniform samples is stated in [10, 1, 20] as inversion of the NDFT and used for example in magnetic resonance imaging [12]. In each of these applications, the actual computation of the NDFT is the computationally dominant task and one has to deal with different requirements on the NFFT with respect to the target accuracy, the usage of memory, and the actual computation time. An early review of several algorithms for the NFFT is given in [29]. Only later, a unified approach to fast algorithms for the present task was obtained in [27, 26] and recently, a particular property of the Gaussian window function was utilised in [15] to speed up computations when no precomputation is possible.

Despite the fact, that there exist a couple of tutorial papers for the NFFT (see also [16]), the aim of this manuscript is to give detailed information on the accuracy, memory, and time requirements of NFFT algorithms and to describe how to balance these factors. In particular, the $\mathcal{O}(M)$-step of the NFFT hides a significant constant and we focus on a variety of precomputation schemes which lead to substantially different computation times of the whole NFFT.

The outline of the paper is as follows. In Section 2 minor improvements in the direct calculation of the NDFT are discussed, cf. [2]. Furthermore the unified approach to the NFFT is reviewed and alternative NFFTs are discussed. In Section 3 we compare different methods for the fast evaluation and precomputation of the most popular window functions. Finally, we compare the different NFFTs numerically in Section 4 and draw conclusions. All used algorithms are available as pre-release of an upcoming version of our widely used software [18].

2 Notation, the NDFT and the NFFT

This section summarises the mathematical theory and ideas behind the NFFT. For d, $M \in \mathbb{N}$ let the torus $\mathbb{T}^d := \mathbb{R}^d/\mathbb{Z}^d \sim [-\frac{1}{2}, \frac{1}{2})^d$ and the sampling set $\mathcal{X} := \{\boldsymbol{x}_j \in \mathbb{T}^d : j = 0, \ldots, M-1\}$ be given. Furthermore, let the multi degree $\boldsymbol{N} = (N_0, N_1, \ldots, N_{d-1})^\top \in 2\mathbb{N}^d$ and the index set for possible frequencies $I_{\boldsymbol{N}} := \{-\frac{N_0}{2}, \frac{N_0}{2}-1\} \times \ldots \times \{-\frac{N_{d-1}}{2}, \frac{N_{d-1}}{2}-1\}$ be given. We define the space of d-variate trigonometric polynomials $T_{\boldsymbol{N}}$ of multi degree $\boldsymbol{N}$ by

$$T_{\boldsymbol{N}} := \operatorname{span}\left\{\mathrm{e}^{-2\pi\mathrm{i}\boldsymbol{k}\cdot} : \boldsymbol{k} \in I_{\boldsymbol{N}}\right\}$$

The dimension of this space and hence the total number of Fourier coefficients is $N_\pi = N_0 \cdot \ldots \cdot N_{d-1}$. Note, that we abbreviate the inner product between the frequency $\boldsymbol{k}$ and the time/spatial node $\boldsymbol{x}$ by $\boldsymbol{k}\boldsymbol{x} = \boldsymbol{k}^\top \boldsymbol{x} = k_0 x_0 + k_1 x_1 + \ldots + k_{d-1} x_{d-1}$. For clarity of presentation the multi index $\boldsymbol{k}$ addresses elements of vectors and matrices as well.

2.1 NDFT

For a finite number of given Fourier coefficients $\hat{f}_{\boldsymbol{k}} \in \mathbb{C},\ \boldsymbol{k} \in I_{\boldsymbol{N}}$, one wants to evaluate the trigonometric polynomial

$$f(\boldsymbol{x}) := \sum_{\boldsymbol{k} \in I_{\boldsymbol{N}}} \hat{f}_{\boldsymbol{k}} \mathrm{e}^{-2\pi \mathrm{i} \boldsymbol{k} \boldsymbol{x}} \tag{2.1}$$

at given nonequispaced nodes $\boldsymbol{x}_j \in \mathbb{T}^d,\ j = 0, \ldots, M-1$. Thus, our concern is the computation of the matrix vector product

$$\boldsymbol{f} = \boldsymbol{A} \hat{\boldsymbol{f}} \tag{2.2}$$

where

$$\boldsymbol{f} := (f(\boldsymbol{x}_j))_{j=0,\ldots,M-1}, \quad \boldsymbol{A} := \left(\mathrm{e}^{-2\pi \mathrm{i} \boldsymbol{k} \boldsymbol{x}_j}\right)_{j=0,\ldots,M-1;\ \boldsymbol{k} \in I_{\boldsymbol{N}}}, \quad \hat{\boldsymbol{f}} := \left(\hat{f}_{\boldsymbol{k}}\right)_{\boldsymbol{k} \in I_{\boldsymbol{N}}}.$$

The straight forward algorithm for this matrix vector product, which is called NDFT in Algorithm 1, takes $\mathcal{O}(MN_\pi)$ arithmetical operations and stores no matrix elements at all, but rather uses MN_π direct calls of the function `cexp()` to evaluate the complex exponentials $\mathrm{e}^{-2\pi \mathrm{i} \boldsymbol{k} \boldsymbol{x}_j}$.

Input: $d, M \in \mathbb{N},\ \boldsymbol{N} \in 2\mathbb{N}^d$,
$\boldsymbol{x}_j \in [-\frac{1}{2}, \frac{1}{2}]^d,\ j = 0, \ldots, M-1$, and $\hat{f}_{\boldsymbol{k}} \in \mathbb{C},\ \boldsymbol{k} \in I_{\boldsymbol{N}}$.

for $j = 0, \ldots, M-1$ **do**
  $f_j = 0$
  for $\boldsymbol{k} \in I_{\boldsymbol{N}}$ **do**
    $f_j += \hat{f}_{\boldsymbol{k}} \mathrm{e}^{-2\pi \mathrm{i} \boldsymbol{k} \boldsymbol{x}_j}$
  end for
end for

Output: values $f_j = f(\boldsymbol{x}_j),\ j = 0, \ldots, M-1$.

Algorithm 1: NDFT

Related matrix vector products are the adjoint NDFT

$$\hat{\boldsymbol{f}} = \boldsymbol{A}^{\mathsf{H}} \boldsymbol{f}, \qquad \hat{f}_{\boldsymbol{k}} = \sum_{j=0}^{M-1} f_j \mathrm{e}^{2\pi \mathrm{i} \boldsymbol{k} \boldsymbol{x}_j},$$

where the update step in Algorithm 1 is simply changed to $\hat{f}_{\boldsymbol{k}} += f_j \mathrm{e}^{2\pi \mathrm{i} \boldsymbol{k} \boldsymbol{x}_j}$, the conjugated NDFT $\boldsymbol{f} = \overline{\boldsymbol{A}} \hat{\boldsymbol{f}}$, and the transposed NDFT $\hat{\boldsymbol{f}} = \boldsymbol{A}^{\top} \boldsymbol{f}$ where $\boldsymbol{A}^{\mathsf{H}} = \overline{\boldsymbol{A}}^{\top}$. Note furthermore, that the inversion formula $\boldsymbol{F}^{-1} = \boldsymbol{F}^{\mathsf{H}}$ for the (equispaced and normalised) Fourier matrix $\boldsymbol{F}$ does **not** hold in the general situation of arbitrary sampling nodes for the matrix $\boldsymbol{A}$.

NDFT acceleration

Algorithm 1 evaluates MN_π complex exponentials. Due to the fact that these direct calls are more expensive than multiplications, we may basically change the update step to $f_j = f_j \mathrm{e}^{2\pi \mathrm{i} x_j} + \hat{f}_k$ ($d = 1$), i.e., do a Horner-like-scheme, see also [2]. Hence, in general $2dM$ direct calls are sufficient for the computations in Algorithm 1. Note however, that this approach looses numerical stability to some extend, cf. [28]. Trading even more memory for the acceleration of the computation, one might precompute all entries of the matrix $\boldsymbol{A}$, which is only feasible for small N_π and M, see Example 4.1 in Section 4.

NDFT method	memory	flops	evaluations
standard	-	MN_π	MN_π
Horner-like	-	MN_π	$2dM$
fully precomputed	MN_π	MN_π	-

Table 2.1: Number of precomputed and stored complex exponentials (memory), the order of magnitude for the number of floating point operations (flops), and the number of evaluations for the function `cexp()` (evaluations).

2.2 NFFT

The most successful approach for the fast computation of (2.2), cf. [7, 4, 27, 26, 13, 12, 15], is based on the usage of an oversampled FFT and a window function φ which is simultaneously localised in time/space and frequency. Basically, the scheme utilises the convolution theorem in the following three informal steps:

1. deconvolve the trigonometric polynomial f in (2.1) with the window function in frequency domain,
2. compute an oversampled FFT on the result of step 1., and
3. convolve the result of step 2. with the window function in time/spatial domain; evaluate this convolution at the nodes $\boldsymbol{x}_j$.

Throughout the rest of the paper $\sigma > 1$ and $n = \sigma N \in \mathbb{N}$ will denote the *oversampling factor* and the *FFT size*, respectively. Furthermore, for $d > 1$ let $\boldsymbol{\sigma} \in \mathbb{R}^d$, $\sigma_0, \ldots, \sigma_{d-1} > 1$, $\boldsymbol{n} = \boldsymbol{\sigma} \odot \boldsymbol{N}$, and $n_\pi = n_0 \cdot \ldots \cdot n_{d-1}$ denote the oversampling factor, the FFT size, and the total FFT size, respectively. Here, we use for notational convenience the pointwise product $\boldsymbol{\sigma} \odot \boldsymbol{N} := (\sigma_0 N_0, \sigma_1 N_1, \ldots, \sigma_{d-1} N_{d-1},)^\top$ with its inverse $\boldsymbol{N}^{-1} := \left(\frac{1}{N_0}, \frac{1}{N_1}, \ldots, \frac{1}{N_{d-1}}\right)^\top$.

The window function

Starting with a window function $\varphi \in L_2(\mathbb{R})$, which is well localised in the time/spatial domain $\mathbb{R}$ and in the frequency domain $\mathbb{R}$, respectively, one assumes that its 1-periodic

version $\tilde{\varphi}$, i.e.,

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

has an uniformly convergent Fourier series and is well localised in the time/spatial domain $\mathbb{T}$ and in the frequency domain $\mathbb{Z}$, respectively. Thus, the periodic window function $\tilde{\varphi}$ may be represented by its Fourier series

$$\tilde{\varphi}(x) = \sum_{k \in \mathbb{Z}} \hat{\varphi}(k) \, \mathrm{e}^{-2\pi \mathrm{i} k x}$$

with the Fourier coefficients

$$\hat{\varphi}(k) := \int_{\mathbb{T}} \tilde{\varphi}(x) \, \mathrm{e}^{2\pi \mathrm{i} k x} \, \mathrm{d}x = \int_{\mathbb{R}} \varphi(x) \, \mathrm{e}^{2\pi \mathrm{i} k x} \, \mathrm{d}x \,, \qquad k \in \mathbb{Z}.$$

We truncate this series at the FFT length n which causes a aliasing error.

If φ is furthermore well localised in time/spatial domain $\mathbb{R}$, it can be truncated with *truncation parameter* $m \in \mathbb{N}$, $m \ll n$ and approximated by the function $\varphi \cdot \chi_{[-\frac{m}{n}, \frac{m}{n}]}$ which has compact support within the interval $[-\frac{m}{n}, \frac{m}{n}]$. Furthermore, the periodic window function can be approximated by the periodic version of the truncated window function. For $d > 1$, univariate window functions $\varphi_0, \ldots, \varphi_{d-1}$, and a node $\boldsymbol{x} = (x_0, \ldots, x_{d-1})^\top$

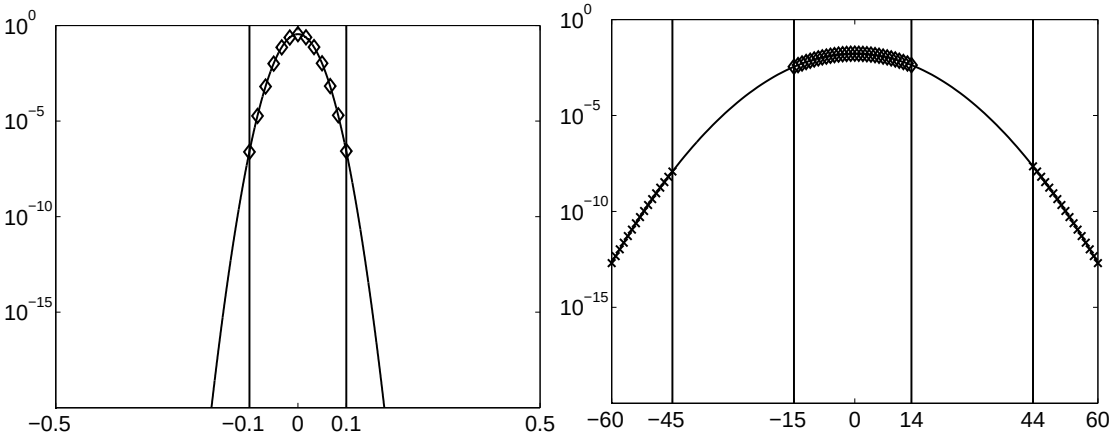

Figure 2.1: Left: Gaussian window function $\varphi(x) = c\,\mathrm{e}^{-\alpha x^2}$ (solid), cf. (3.1), and its 1-periodic version $\tilde{\varphi}$ sampled on $2m+1$ nodes $x_0 - \frac{m}{n}, x_0 - \frac{m-1}{n}, \ldots, x_0 + \frac{m}{n}$ denoted by $\diamond$ where $x_0 = 0$; Right: integral Fourier transform $\hat{\varphi}$ with pass ($\diamond$), transition, and stop band ($\times$); for the parameters $N = 30$, $\sigma = 2$, $n = 60$, $m = 6$.

the multivariate window function is simply given by

$$\varphi(\boldsymbol{x}) := \varphi_0(x_0) \, \varphi_1(x_1) \ldots \varphi_{d-1}(x_{d-1}) \,, \tag{2.3}$$

where $\tilde{\varphi}(\boldsymbol{x}) = \sum_{\boldsymbol{r}\in\mathbb{Z}^d} \varphi(\boldsymbol{x}+\boldsymbol{r})$ again denotes the 1-periodic version; an immediate observation is

$$\hat{\varphi}\left(\boldsymbol{k}\right) := \int\limits_{\mathbb{R}^d} \varphi\left(\boldsymbol{x}\right) \mathrm{e}^{2\pi\mathrm{i}\boldsymbol{k}\boldsymbol{x}}\,\mathrm{d}\boldsymbol{x} = \hat{\varphi}_0\left(k_0\right)\hat{\varphi}_1\left(k_1\right)\ldots\hat{\varphi}_{d-1}\left(k_{d-1}\right)\,.$$

For a single truncation parameter $m \in \mathbb{N}$ the window function is truncated to the cube $\boldsymbol{n}^{-1}\odot[-m,m]^d$.

We follow the general approach of [27, 26] and approximate the complex exponentials in the trigonometric polynomial (2.1) by

$$\mathrm{e}^{-2\pi\mathrm{i}\boldsymbol{k}\boldsymbol{x}} \approx \frac{1}{n_\pi\hat{\varphi}\left(\boldsymbol{k}\right)} \sum_{\boldsymbol{l}\in I_{\boldsymbol{n},m}(\boldsymbol{x})} \tilde{\varphi}\left(\boldsymbol{x}-\boldsymbol{n}^{-1}\odot\boldsymbol{l}\right)\mathrm{e}^{-2\pi\mathrm{i}\left(\boldsymbol{n}^{-1}\odot\boldsymbol{l}\right)\boldsymbol{k}} \tag{2.4}$$

where the set

$$I_{\boldsymbol{n},m}\left(\boldsymbol{x}\right) := \left\{\boldsymbol{l}\in I_{\boldsymbol{n}} : \boldsymbol{n}\odot\boldsymbol{x} - m\boldsymbol{1} \le \boldsymbol{l} \le \boldsymbol{n}\odot\boldsymbol{x} + m\boldsymbol{1}\right\}$$

collects these indices where the window function is mostly concentrated (the inequalities have to be fulfilled modulo $\boldsymbol{n}$ and for each component). After changing the order of summation in (2.1) we obtain for $\boldsymbol{x}_j \in \mathbb{T}^d$, $j = 0,\ldots,M-1$, the approximation

$$f\left(\boldsymbol{x}_j\right) \approx \sum_{\boldsymbol{l}\in I_{\boldsymbol{n},m}(\boldsymbol{x}_j)} \left(\sum_{\boldsymbol{k}\in I_{\boldsymbol{N}}} \frac{\hat{f}_{\boldsymbol{k}}}{n_\pi\hat{\varphi}\left(\boldsymbol{k}\right)}\mathrm{e}^{-2\pi\mathrm{i}\left(\boldsymbol{n}^{-1}\odot\boldsymbol{l}\right)\boldsymbol{k}}\right)\tilde{\varphi}\left(\boldsymbol{x}_j-\boldsymbol{n}^{-1}\odot\boldsymbol{l}\right)\,.$$

This causes a truncation and an aliasing error, see [26, 24] for details. As can be readily seen, after an initial deconvolution step, the expression in brackets can be computed via a d-variate FFT of total size n_π. The final step consists of the evaluation of sums having at most $(2m+1)^d$ summands where the window function is sampled only in the neighbourhood of the node $\boldsymbol{x}_j$.

The algorithm and its matrix notation

The proposed scheme reads in matrix vector notation as

$$\boldsymbol{A}\hat{\boldsymbol{f}} \approx \boldsymbol{B}\boldsymbol{F}\boldsymbol{D}\hat{\boldsymbol{f}}, \tag{2.5}$$

where $\boldsymbol{B}$ denotes the real $M\times n_\pi$ sparse matrix

$$\boldsymbol{B} := \left(\tilde{\varphi}\left(\boldsymbol{x}_j-\boldsymbol{n}^{-1}\odot\boldsymbol{l}\right)\cdot\chi_{I_{\boldsymbol{n},m}(\boldsymbol{x}_j)}\left(\boldsymbol{l}\right)\right)_{j=0,\ldots,M-1;\,\boldsymbol{l}\in I_{\boldsymbol{n}}}\,, \tag{2.6}$$

where $\boldsymbol{F}$ is the d-variate Fourier matrix of size $n_\pi\times n_\pi$, and where $\boldsymbol{D}$ is the real $n_\pi\times N_\pi$ 'diagonal' matrix

$$\boldsymbol{D} := \bigotimes_{t=0}^{d-1}\left(\boldsymbol{O}_t \,|\, \mathrm{diag}\left(1/\,\hat{\varphi}_t\left(k_t\right)\right)_{k_t\in I_{N_t}} \,|\, \boldsymbol{O}_t\right)^\top$$

Method	memory	flops	evaluations
-	-	N_π	N_π
PRE_PHI_HUT	$N_0 + \ldots + N_{d-1}$	N_π	-

Table 2.2: Computational requirements for the deconvolution step in Algorithm 2.

with zero matrices $\boldsymbol{O}_t$ of size $N_t \times \frac{n_t - N_t}{2}$. Obviously, the approximate matrix splitting can by applied to the adjoint matrix as $\boldsymbol{A}^{\mathsf{H}} \approx \boldsymbol{D}^\top \boldsymbol{F}^{\mathsf{H}} \boldsymbol{B}^\top$, where the multiplication with the sparse matrix $\boldsymbol{B}^\top$ is implemented in a 'transposed' way, summation as outer loop and only using the index sets $I_{\boldsymbol{n},m}(\boldsymbol{x}_j)$.

Table 2.2 shows the computational requirements for the deconvolution step of the NFFT. We give detailed information on the number of precomputed and stored real numbers (memory), the order of magnitude for the number of floating point operations (flops), and the number of evaluations for the univariate Fourier-transformed window function $\hat{\varphi}$. Precomputing the factors $\hat{\varphi}_t(k_t)$ for $t = 0, \ldots, d-1$ and $k_t \in I_{N_t}$ is denoted by its associated Flag PRE_PHI_HUT within the software library.

This is followed by one FFT of total size n_π. Hence, the computational complexity of the NFFT increases for a larger oversampling factor $\boldsymbol{\sigma}$, affecting both the 'deconvolution step' and the FFT. The time and memory requirements of the convolution and evaluation step are discussed in Section 3 in detail. In summary, we propose Algorithm 2 and Algorithm 3 for the computation of the nonequispaced FFT (2.2) and its adjoint, respectively.

Input: $d, M \in \mathbb{N}$, $\boldsymbol{N} \in 2\mathbb{N}^d$,
$\boldsymbol{x}_j \in [-\frac{1}{2}, \frac{1}{2}]^d$, $j = 0, \ldots, M-1$, and $\hat{f}_{\boldsymbol{k}} \in \mathbb{C}$, $\boldsymbol{k} \in I_{\boldsymbol{N}}$.

For $\boldsymbol{k} \in I_{\boldsymbol{N}}$ compute

$$\hat{g}_{\boldsymbol{k}} := \frac{\hat{f}_{\boldsymbol{k}}}{n_\pi c_{\boldsymbol{k}}(\tilde{\varphi})}.$$

For $\boldsymbol{l} \in I_{\boldsymbol{n}}$ compute by d-variate FFT

$$g_{\boldsymbol{l}} := \sum_{\boldsymbol{k} \in I_{\boldsymbol{N}}} \hat{g}_{\boldsymbol{k}} \, \mathrm{e}^{-2\pi \mathrm{i} \boldsymbol{k} \left(\boldsymbol{n}^{-1} \odot \boldsymbol{l}\right)}.$$

For $j = 0, \ldots, M-1$ compute

$$s_j := \sum_{\boldsymbol{l} \in I_{\boldsymbol{n},m}(\boldsymbol{x}_j)} g_{\boldsymbol{l}} \, \tilde{\varphi}\left(\boldsymbol{x}_j - \boldsymbol{n}^{-1} \odot \boldsymbol{l}\right).$$

Output: approximate values $s_j \approx f_j$, $j = 0, \ldots, M-1$.

Algorithm 2: NFFT

Input: $d, M \in \mathbb{N}$, $\boldsymbol{N} \in 2\mathbb{N}^d$,
$\boldsymbol{x}_j \in [-\frac{1}{2}, \frac{1}{2}]^d$ and $f_j \in \mathbb{C},\ j = 0, \ldots, M-1$.

Compute the sparse matrix vector product

$$\boldsymbol{g} := \boldsymbol{B}^\top \boldsymbol{f}\,.$$

Apply the d-variate IFFT as

$$\hat{\boldsymbol{g}} := \boldsymbol{F}^{\mathsf{H}} \boldsymbol{g}\,.$$

Multiply by the 'diagonal' matrix, i.e.,

$$\hat{\boldsymbol{s}} := \boldsymbol{D}^\top \hat{\boldsymbol{g}}\,.$$

Output: approximate values $\hat{s}_{\boldsymbol{k}},\ \boldsymbol{k} \in I_{\boldsymbol{N}}$.

Algorithm 3: NFFT$^{\mathsf{H}}$

Alternative NFFTs

Taylor based NFFT: A simple but nevertheless fast scheme for the computation of (2.2) in the univariate case $d = 1$ is presented in [1]. This approach uses for each node $x_j \in [-\frac{1}{2}, \frac{1}{2})$ a m-th order Taylor expansion of the trigonometric polynomial in (2.1) about the nearest neighbouring point on the oversampled equispaced lattice $\{n^{-1}k - \frac{1}{2}\}_{k=0,\ldots,n-1}$ where again $n = \sigma N$, $\sigma \gg 1$. Besides its simple structure and only $\mathcal{O}(N \log N + M)$ arithmetic operations, this algorithm utilises m FFTs of size n compared to only one in the NFFT approach, uses a medium amount of extra memory, and is not suited for highly accurate computations, see Example 4.2. Furthermore, its extension to higher dimensions has not been considered so far.

Multipole based NFFT: A second approach for the univariate case $d = 1$ is considered in [8] and based on a Lagrange interpolation technique. After taking a N-point FFT of the vector $\hat{\boldsymbol{f}}$ in (2.2) one uses an exact polynomial interpolation scheme to obtain the values of the trigonometric polynomial f at the nonequispaced nodes x_j. Here, the time consuming part is the exact polynomial interpolation scheme which can however be realised fast in an approximate way by means of the fast multipole method. This approach is appealing since it allows also for the inverse transform. Nevertheless, numerical experiments in [8] showed that this approach is far more time consuming than Algorithm 2 and the inversion can only be computed in a stable way for almost equispaced nodes [8].

Linear algebra based NFFT: Using a fully discrete approach, one might fix the entries of the 'diagonal' matrix $\boldsymbol{D}$ in (2.5) first and precompute optimised entries for the sparse matrix $\boldsymbol{B}$ to achieve higher accuracy, cf. [22, 23]. A similar approach, based on min-max interpolation, has been taken within [12]. While these approaches gain some accuracy for the Gaussian or B-Spline windows, no reasonable improvement is obtained

for the Kaiser-Bessel window function. Since it is more expensive to precompute these optimised entries of the matrix $\boldsymbol{B}$, we do not further consider these schemes.

3 Evaluation techniques for window functions

To keep the aliasing error and the truncation error small, several univariate functions φ with good localisation in time and frequency domain were proposed. For an oversampling factor $\sigma > 1$, a degree $N \in 2\mathbb{N}$, the FFT length $n = \sigma N$, and a cut-off parameter $m \in \mathbb{N}$, the following window functions are considered:

1. for a shape parameter $b = \frac{2\sigma}{2\sigma-1}\frac{m}{\pi}$ the dilated *Gaussian window* [7, 27, 6]
$$\varphi(x) = (\pi b)^{-1/2}\, \mathrm{e}^{-\frac{(nx)^2}{b}}, \tag{3.1}$$
2. for M_{2m} denoting the centred cardinal B-Spline of order $2m$ the dilated *B-Spline window* [4, 27]
$$\varphi(x) = M_{2m}(nx), \tag{3.2}$$
3. the dilated *Sinc window* [24]
$$\varphi(x) = \operatorname{sinc}^{2m}\left(\frac{(2\sigma-1)N}{2m}\pi x\right) \tag{3.3}$$
with $\operatorname{sinc}(x) := \sin(x)/x$ for $x \neq 0$ and $\operatorname{sinc}(0) := 1$
4. and for a shape parameter $b = \pi(2 - \frac{1}{\sigma})$ the dilated *Kaiser-Bessel window* [25]
$$\varphi(x) = \frac{1}{\pi}\begin{cases} \frac{\sinh\left(b\sqrt{m^2-n^2x^2}\right)}{\sqrt{m^2-n^2x^2}} & \text{for } |x| \leq \frac{m}{n}, \\ \frac{\sin\left(b\sqrt{n^2x^2-m^2}\right)}{\sqrt{n^2x^2-m^2}} & \text{otherwise.} \end{cases} \tag{3.4}$$

Note, that the latter two have compact support in frequency domain while the second one has compact support in time domain. Further references on the usage of (generalised) Kaiser-Bessel window functions include [17, 13, 21], where some authors prefer to interchange the role of time and frequency domain. For these univariate window functions φ, the error introduced by Algorithm 2 obeys

$$|f(x_j) - s_j| \leq C_{\sigma,m}\|\hat{\boldsymbol{f}}\|_1 \tag{3.5}$$

where

$$C_{\sigma,m} := \begin{cases} 4\,\mathrm{e}^{-m\pi(1-1/(2\sigma-1))} & \text{for (3.1), cf. [27]}, \\ 4\left(\frac{1}{2\sigma-1}\right)^{2m} & \text{for (3.2), cf. [27]}, \\ \frac{1}{m-1}\left(\frac{2}{\sigma^{2m}} + \left(\frac{\sigma}{2\sigma-1}\right)^{2m}\right) & \text{for (3.3), cf. [24]}, \\ 4\pi\left(\sqrt{m}+m\right)\sqrt[4]{1-\frac{1}{\sigma}}\,\mathrm{e}^{-2\pi m\sqrt{1-1/\sigma}} & \text{for (3.4), cf. [24]}. \end{cases}$$

Thus, for fixed $\sigma > 1$, the approximation error introduced by the NFFT decays exponentially with the number m of summands in (2.4). Using the tensor product approach, the above error estimates have been generalised for the multivariate setting in [9, 6]. Note furthermore, that it is convenient to replace the periodic window function $\tilde{\varphi}$ again by the original one φ within the actual computation. This causes an error for functions with large support in time/spatial domain. However, whenever the FFT-length n is reasonable 'large', e.g., $n \geq \max\{4m, 12\}$ for the Gaussian, an easy calculation shows that for $x \in [-\frac{m}{n}, \frac{m}{n}]$ the estimate

$$\frac{|\tilde{\varphi}(x) - \varphi(x)|}{|\varphi(x)|} = \sum_{r \in \mathbb{Z} \setminus \{0\}} \mathrm{e}^{-\frac{n^2}{b} r(r-2x)} < 10^{-16}$$

holds true, i.e., the made error is within machine precision. If the restriction on n is not fulfilled, the NDFT method is competitive, anyway.

In the following, we suggest different methods for the compressed storage and application of the matrix $\boldsymbol{B}$ which are all available within our NFFT library by choosing particular flags in a simple way during the initialisation phase. These methods do not yield a different asymptotic performance but rather yield a lower constant in the amount of computation.

3.1 Fully precomputed window function

One possibility is to precompute all values $\varphi(\boldsymbol{x}_j - \boldsymbol{n}^{-1} \odot \boldsymbol{l})$ for $j = 0, \ldots, M-1$ and $\boldsymbol{l} \in I_{\boldsymbol{n},m}(\boldsymbol{x}_j)$ explicitly. Thus, one has to store the large amount of $(2m+1)^d M$ real numbers but uses no extra floating point operations during the matrix vector multiplication beside the necessary $(2m+1)^d M$ flops. Furthermore, we store for this method explicitly the row and column for each nonzero entry of the matrix $\boldsymbol{B}$. This method, included by the flag `PRE_FULL_PSI`, is the fastest procedure but can only be used if enough main memory is available.

3.2 Tensor product based precomputation

Using the fact that the window functions are built as tensor products one can store $\varphi_t((\boldsymbol{x}_j)_t - \frac{l_t}{n_t})$ for $j = 0, \ldots, M-1$, $t = 0, \ldots, d-1$, and $l_t \in I_{n_t,m}((\boldsymbol{x}_j)_t)$ where $(\boldsymbol{x}_j)_t$ denotes the t-th component of the j-th node. This method uses a medium amount of memory to store $d(2m+1)M$ real numbers in total. However, one has to carry out for each node at most $2(2m+1)^d$ extra multiplications to compute from the factors the multivariate window function $\varphi(\boldsymbol{x}_j - \boldsymbol{n}^{-1} \odot \boldsymbol{l})$ for $\boldsymbol{l} \in I_{\boldsymbol{n},m}(\boldsymbol{x}_j)$. Note, that this technique is available for every window function discussed here and can be used by means of the flag `PRE_PSI` which is also the default method within our software library.

3.3 Linear interpolation from a lookup table

For a large number of nodes M, the amount of memory can by further reduced by the use of lookup table techniques. For a recent example within the framework of gridding see

[3]. We suggest to precompute from the even window function the equidistant samples $\varphi_t(\frac{rm}{Kn_t})$ for $t = 0,\ldots,d-1$ and $r = 0,\ldots,K,\ K \in \mathbb{N}$ and then compute for the actual node $\boldsymbol{x}_j$ during the NFFT the values $\varphi_t((\boldsymbol{x}_j)_t - \frac{l_t}{n_t})$ for $t = 0,\ldots,d-1$ and $l_t \in I_{n_t,m}((\boldsymbol{x}_j)_t)$ by means of the linear interpolation from its two neighbouring precomputed samples.

Lemma 3.1 *For the univariate window functions* (3.1) - (3.4) *and* $K \in \mathbb{N}$ *the linear interpolated window function, denoted by* φ_K*, fulfils*

$$\max_{|x|\leq\frac{m}{n}} |\varphi(x) - \varphi_K(x)| \leq \begin{cases} \left(\dfrac{2\sigma-1}{\sigma}\right)^{3/2} \dfrac{\pi\sqrt{2m}}{16K^2} & \textit{for } (3.1)\,, \\ \dfrac{m^2}{4K^2} & \textit{for } (3.2)\,, \\ \dfrac{m(2\sigma-1)^2\pi^2}{48\sigma^2K^2} & \textit{for } (3.3)\,, \\ \dfrac{\mathrm{e}^{2\pi m}}{8K^2} & \textit{for } (3.4)\,. \end{cases}$$

Proof: From standard error estimates, we know that the linear interpolated window function φ_K fulfils

$$\max_{|x|\leq\frac{m}{n}} |\varphi(x) - \varphi_K(x)| \leq \frac{m^2}{8K^2n^2} \max_{|\xi|\leq\frac{m}{n}} \left|\varphi''(\xi)\right|. \tag{3.6}$$

The maximum of this second derivative is met for the window functions (3.1) - (3.4) at $\xi = 0$. Thus, the assertion follows by

$$|\varphi''(0)| = \begin{cases} \left(\dfrac{2\sigma-1}{\sigma m}\right)^{3/2} \dfrac{\pi n^2}{\sqrt{2}} & \text{for } (3.1)\,, \\ 2n^2\left(M_{2m-2}(0) - M_{2m-2}(1)\right) & \text{for } (3.2)\,, \\ \dfrac{(2\sigma-1)^2\pi^2n^2}{6m\sigma^2} & \text{for } (3.3)\,, \\ \dfrac{n^2}{2m^3\pi}\left(bm\cosh(bm) - \sinh(bm)\right) & \text{for } (3.4)\,, \end{cases}$$

and the estimates $M_{2m-2}(0) - M_{2m-2}(1) \leq 1$ and $bm\cosh(bm) - \sinh(bm) \leq 2\pi m\,\mathrm{e}^{2\pi m}$. ■

This method needs only a storage of dK real numbers in total where K depends solely on the target accuracy but neither on the number of nodes M nor on the multi degree $\boldsymbol{N}$. Choosing K to be a multiple of m, we further reduce the computational costs during the interpolation since the distance from $(\boldsymbol{x}_j)_t - \frac{l_t}{n_t}$ to the two neighbouring interpolation nodes and hence the interpolation weights remain the same for all $l_t \in I_{n_t,m}((\boldsymbol{x}_j)_t)$. This method requires $2(2m+1)^d$ extra multiplications per node and is used within the NFFT by the flag `PRE_LIN_PSI`.

3.4 Fast Gaussian gridding

Two useful properties of the Gaussian window function (3.1) within the present framework were recently reviewed in [15]. Beside its tensor product structure for $d > 1$, which also holds for all other window functions, it is remarkable that the number of evaluations of the form `exp()` can be greatly decreased. More precisely, for $d = 1$ and a fixed node x_j the evaluations of $\varphi(x_j - \frac{l'}{n})$, $l' \in I_{n,m}(x_j)$, can be reduced by the splitting

$$\sqrt{\pi b}\varphi\left(x_j - \frac{l'}{n}\right) = \mathrm{e}^{-\frac{(nx_j - l')^2}{b}} = \mathrm{e}^{-\frac{(nx_j-u)^2}{b}} \left(\mathrm{e}^{-\frac{2(nx_j-u)}{b}}\right)^l \mathrm{e}^{-\frac{l^2}{b}}.$$

where $u = \min I_{n,m}(x_j)$ and $l = 0, \ldots, 2m$. Note, that the first factor and the exponential within the brackets are constant for each fixed node x_j. Once, we evaluate the second exponential, its l-th power can be computed consecutively by multiplications only. Furthermore, the last exponential is independent of x_j and these $2m+1$ values are computed only once within the NFFT and their amount is negligible. Thus, it is sufficient to store or evaluate $2M$ exponentials for $d = 1$. The case $d > 1$ uses $2dM$ storages or evaluations by using the general tensor product structure. This method is employed by the flags `FG_PSI` and `PRE_FG_PSI` for the evaluation or storage of 2 exponentials per node, respectively.

3.5 No precomputation of the window function

The last considered method uses no precomputation at all, but rather evaluates the univariate window function $(2m + 1)^d M$ times. Thus, the computational time depends on how fast we can evaluate the particular window function. However, no additional storage is necessary which suits this approach whenever the problem size reaches the memory limits of the used computer.

3.6 Summary on the computational costs

The multiplication with the sparse matrix $\boldsymbol{B}$ clearly takes $\mathcal{O}(m^d M)$ operations. Beside this, Table 3.1 summarises the memory requirements for different strategies to store the elements of this matrix and the extra costs it takes to multiply with this 'compressed' matrix.

4 Numerical experiments

We present numerical experiments in order to demonstrate the performance of our algorithms. All algorithms were implemented in C and tested on an AMD Athlon$^{\mathrm{TM}}$XP 2700+ with 2GB main memory, SuSe-Linux (kernel 2.4.20-4GB-athlon, gcc 3.3) using double precision arithmetic. Moreover, we have used the libraries FFTW 3.0.1 [14] and an extended version of NFFT 2.0.3 [18]. In order to reproduce all the numerical results, a pre-release of the upcoming NFFT library can be downloaded from our web page [18].

Method	memory	extra flops	evaluations
-	-	*	$m^d M$
FG_PSI	-	dm,*	dM
PRE_LIN_PSI	dK	dm,*	-
PRE_FG_PSI	dM	dm,*	-
PRE_PSI	dmM	*	-
PRE_FULL_PSI	$m^d M$	-	-

Table 3.1: Theoretical order of magnitude for memory requirements, extra floating point operations, and the evaluations of the window function φ. Furthermore, at most $2(2m+1)^d$ multiplications are used within each scheme besides PRE_FULL_PSI to compute the multivariate window function out of its univariate factors, denoted by *.

Example 4.1 *We start by examining accelerated NDFTs for $d = 1$. Using the three proposed possibilities to compute the matrix vector product by either MN direct calls of* `cexp()`*, a Horner-like scheme, or a fully precomputed matrix $\boldsymbol{A}$, we obtain the following timings for increasing N in Figure 4.1 (top-left). Clearly, the complete precomputation of the matrix $\boldsymbol{A}$ does not pay off. The Horner-like NDFT uses no extra memory and is considerably faster than the NDFT. Furthermore, it is faster than the default NFFT until an break even of $N = 128$.* □

Example 4.2 *Our second example concerns the Taylor expansion based NFFT, again only for $d = 1$. We note that this scheme actually provides a competitive NFFT with respect to the computation time relative to the problem size - at least within a factor, cf. Figure 4.1 (top-right). The main drawbacks of this approach are its instability, i.e., it is not possible to obtain high accurate results by increasing the order m of the Taylor expansion and hence the number of used FFTs. This fact remains even true for a very large oversampling factor $\sigma = 16$, see Figure 4.1 (bottom-left). Furthermore, even when the target accuracy $E_\infty = \|\boldsymbol{f} - \boldsymbol{s}\|_\infty / \|\boldsymbol{f}\|_\infty$, cf. [1], is somewhat larger, the NFFT needs considerable fewer arithmetic operations to reach it, cf. 4.1 (bottom-right).* □

Example 4.3 *We now compare the computation time for the three tasks within the NFFT, i.e., the deconvolution step, the oversampled FFT, and the convolution/evaluation step for space dimension $d = 1$. Figure 4.2 shows the timings for increasing degree N, $M = N$ nodes, and a fixed cut-off $m = 4$. The linear dependence of the computation time with respect to the problem size can be seen for the matrix-vector multiplication with the 'diagonal' matrix $\boldsymbol{D}$ and the sparse matrix $\boldsymbol{B}$ whereas the FFT takes $\mathcal{O}(N \log N)$ operations.*

For the deconvolution step we obtain a speed up of more than 3 by avoiding direct calls of the Fourier-transformed window function $\hat{\varphi}$, this method is default and turned on by the precomputation flag PRE_PHI_HUT*, cf. Figure 4.2 (top-left). Ways to speed up the FFT by a more exhaustive search of an optimal FFT-plan are discussed in [14],*

Figure 4.2 (top-right) shows for larger degree N a speed up of around 2 when we use the planner `FFTW_MEASURE`*, which is also default within the NFFT.*

The time to compute the last step of the NFFT differs from no precomputation of the matrix entries of $\boldsymbol{B}$ to explicitly precomputed entries with `PRE_FULL_PSI` *by a factor of 20 to 100 for small degrees $N \leq 2048$ and by a factor of 5 to 20 for larger degrees. Note however, that the use of this flag with 'maximal precomputation' is limited by the available memory, e.g. for $m = 4$, and $M = 2^{20}$ we already need 144 MByte just for storing the matrix entries and its indices.* □

Example 4.4 *Furthermore, we show the timings of the convolution/evaluation step for increasing N, the multi degree $\boldsymbol{N} = (N, \ldots, N)^\top$, $M = N^d$ nodes, a fixed cut-off $m = 4$, and space dimension $d = 2, 3$ in Figure 4.3. Note, that for $d = 2$ and $m = 4$ the matrix $\boldsymbol{B}$ has already 81 nonzero entries per row.* □

Example 4.5 *More detailed, we focus on the convolution/evaluation step for space dimension $d = 1$. Figure 4.4 shows the computation time with respect to achieved target accuracy $E_2 = \|\boldsymbol{f} - \boldsymbol{s}\|_2 / \|\boldsymbol{f}\|_2$, cf. [26], by increasing the cut-off m for fixed degree and number of nodes.*

We conclude, that if no additional memory is used for precomputing the entries of the matrix $\boldsymbol{B}$, the Gaussian window function in conjunction with the flag `FG_PSI` *performs best, cf. Figure 4.4 (top-left). If no precomputation is used, the particular bad behaviour of the B-Spline window function is due to the fact that evaluating this window function once already takes $\mathcal{O}(m)$ operations.*

When only a small amount of memory is used for precomputations, the decision between the linear interpolated Kaiser-Bessel window function and the fast Gaussian gridding with precomputation `PRE_FG_PSI` *depends on the accuracy one would like to achieve - here, the linear interpolated Kaiser-Bessel window performs better up to single precision (top-right).*

Whenever at least $2mM$ values can be precomputed, the Kaiser-Bessel window performs always best, i.e., needs the least time to achieve a given target accuracy, cf. Figure 4.4 (bottom). □

Example 4.6 *Finally, Figure 4.5 shows the quadratic decay of the error introduced by the linear interpolation of the window function if the method* `PRE_LIN_PSI` *is used. The decay of the error E_2 coincides for all window functions up to the accuracy, they actually can provide for a fixed cut-off $m = 10$.* □

5 Conclusions

Fast algorithms for the nonequispaced discrete Fourier transform are already known a couple of years. Besides their asymptotic computational complexity of $\mathcal{O}(N_\pi \log N_\pi +$

M) for N_π equispaced frequencies and M nonequispaced sampling nodes, NFFTs differ substantially in their computation time for interesting problem sizes. For its actual usage, we summarise:

1. If the problem size is really small, e.g. $N = M < 32$ for $d = 1$, just use the NDFT or its Horner-like derivative.

2. The simplest fast method is the Taylor expansion based NFFT, it achieves not even single precision, needs a somewhat larger oversampling factor, and is slower than window function based methods.

3. If the problem barely fits into your computer, you should use the fast Gaussian gridding NFFT, i.e., the Gaussian window function in conjunction with the flag `FG_PSI` which uses no extra memory.

4. Using only a small amount of memory for precomputation and asking for high accuracy, the fast Gaussian gridding NFFT with precomputation performs best while storing $2d$ real numbers per node. However, the Kaiser-Bessel window in conjunction with the lookup table method `PRE_LIN_PSI` with 2^{12} precomputed values suffices for single precision 10^{-8}, regardless of the problem size, and outperforms the fast Gaussian gridding. Furthermore, the lookup table is the only precomputation method which is independent of the actual sampling set $\{\boldsymbol{x}_j\}$.

5. If a medium amount of memory can be used for precomputation, the Kaiser-Bessel window function performs best. The tensor product based precomputation scheme `PRE_PSI` yields a faster NFFT than the lookup table method or the fast Gaussian gridding with precomputation, but stores for each node dm real numbers. For small to medium size problems, one can gain another factor 2 to 5 by means of an fully precomputed window function `PRE_FULL_PSI`. However, this causes a storage cost of m^d real numbers per sampling node.

6. Default precomputation methods, selected by the simple initialisation routine of the NFFT, are: `PRE_PHI_HUT` for the deconvolution step, the flag `FFTW_MEASURE` for planning the FFT, and the tensor product based precomputation scheme `PRE_PSI` for the convolution/evaluation step. Furthermore, the Kaiser-Bessel window function is selected as default at compilation.

Acknowledgement

The first author is grateful for support of this work by the German Academic Exchange Service (DAAD) and the warm hospitality during his stay at the Numerical Harmonic Analysis Group, University of Vienna. Furthermore, we would like to thank Matt Fulkerson for contributing an early version of the fast Gaussian gridding to the NFFT software.

References

[1] C. Anderson and M. Dahleh. Rapid computation on the discrete Fourier transform. SIAM *J. Sci. Comput.*, 17:913 – 919, 1996.

[2] S. Bagchi and S. Mitra. The nonuniform discrete Fourier transform. In F. Marvasti, editor, *Nonuniform Sampling: Theory and Practice*. Kluwer/Plenum, 2001.

[3] P. Beatty, D. Nishimura, and J. Pauly. Rapid gridding reconstruction with a minimal oversampling ratio. IEEE *Trans. Med. Imag.*, 24:799 – 808, 2005.

[4] G. Beylkin. On the fast Fourier transform of functions with singularities. *Appl. Comput. Harmon. Anal.*, 2:363 – 381, 1995.

[5] E. Candes, L. Demanet, D. Donoho, and L. Ying. Fast discrete curvelet transforms. SIAM *Multiscale Model. Simul.*, to appear.

[6] A. J. W. Duijndam and M. A. Schonewille. Nonuniform fast Fourier transform. *Geophysics*, 64:539 – 551, 1999.

[7] A. Dutt and V. Rokhlin. Fast fourier transforms for nonequispaced data. SIAM *J. Sci. Stat. Comput.*, 14:1368 – 1393, 1993.

[8] A. Dutt and V. Rokhlin. Fast Fourier transforms for nonequispaced data II. *Appl. Comput. Harmon. Anal.*, 2:85 – 100, 1995.

[9] B. Elbel and G. Steidl. Fast Fourier transform for nonequispaced data. In C. K. Chui and L. L. Schumaker, editors, *Approximation Theory IX*, Nashville, 1998. Vanderbilt University Press.

[10] H. Feichtinger, K. Gröchenig, and T. Strohmer. Efficient numerical methods in non-uniform sampling theory. *Numer. Math.*, 69:423 – 440, 1995.

[11] M. Fenn and J. Ma. Combined complex ridgelet shrinkage and total variation minimization. SIAM *J. Sci. Comput.*, 2005. accepted.

[12] J. A. Fessler and B. P. Sutton. Nonuniform fast Fourier transforms using min-max interpolation. IEEE *Trans. Signal Process.*, 51:560 – 574, 2003.

[13] K. Fourmont. Non equispaced fast Fourier transforms with applications to tomography. *J. Fourier Anal. Appl.*, 9:431 – 450, 2003.

[14] M. Frigo and S. G. Johnson. FFTW, a C subroutine library. http://www.fftw.org/.

[15] L. Greengard and J.-Y. Lee. Accelerating the nonuniform fast Fourier transform. SIAM *Rev.*, 46:443 – 454, 2004.

[16] J. A. Hogan and J. D. Lakey. *Time-Frequency and Time-Scale Methods: Wavelets, Sampling, Uncertainty Principles*. Applied and Numerical Harmonic Analysis series. Birkhauser, Boston, 2005.

[17] J. I. Jackson. Selection of a convolution function for Fourier inversion using gridding. IEEE *Trans. Med. Imag.*, 10:473 – 478, 1991.

[18] S. Kunis and D. Potts. NFFT, Softwarepackage, C subroutine library. http://www.math.uni-luebeck.de/potts/nfft, 2002 – 2005.

[19] S. Kunis and D. Potts. Fast spherical Fourier algorithms. *J. Comput. Appl. Math.*, 161:75 – 98, 2003.

[20] S. Kunis and D. Potts. Stability results for scattered data interpolation by trigonometric polynomials. *Preprint, Univ. Lübeck, A-04-12*, 2004.

[21] S. Matej, J. Fessler, and I. Kazantsev. Iterative tomographic image reconstruction using Fourier-based forward and back- projectors. *IEEE Trans. Med. Imag.*, 23:401 – 412, 2004.

[22] N. Nguyen and Q. H. Liu. The regular Fourier matrices and nonuniform fast Fourier transforms. SIAM *J. Sci. Comput.*, 21:283 – 293, 1999.

[23] A. Nieslony and G. Steidl. Approximate factorizations of Fourier matrices with nonequispaced knots. *Linear Algebra Appl.*, 266:337 – 351, 2003.

[24] D. Potts. Schnelle Fourier-Transformationen für nichtäquidistante Daten und Anwendungen. *Habilitation, Universität zu Lübeck*, 2003.

[25] D. Potts and G. Steidl. Fast summation at nonequispaced knots by NFFTs. SIAM *J. Sci. Comput.*, 24:2013 – 2037, 2003.

[26] D. Potts, G. Steidl, and M. Tasche. Fast Fourier transforms for nonequispaced data: A tutorial. In J. J. Benedetto and P. J. S. G. Ferreira, editors, *Modern Sampling Theory: Mathematics and Applications*, pages 247 – 270, Boston, 2001. Birkhäuser.

[27] G. Steidl. A note on fast Fourier transforms for nonequispaced grids. *Adv. Comput. Math.*, 9:337 – 353, 1998.

[28] M. Tasche and H. Zeuner. Roundoff error analysis for fast trigonometric transforms. In *Handbook of Analytic-Computational Methods in Applied Mathematics*, pages 357 – 406, CRC Press, Boca Raton, 2000.

[29] A. F. Ware. Fast approximate Fourier transforms for irregularly spaced data. SIAM *Rev.*, 40:838 – 856, 1998.

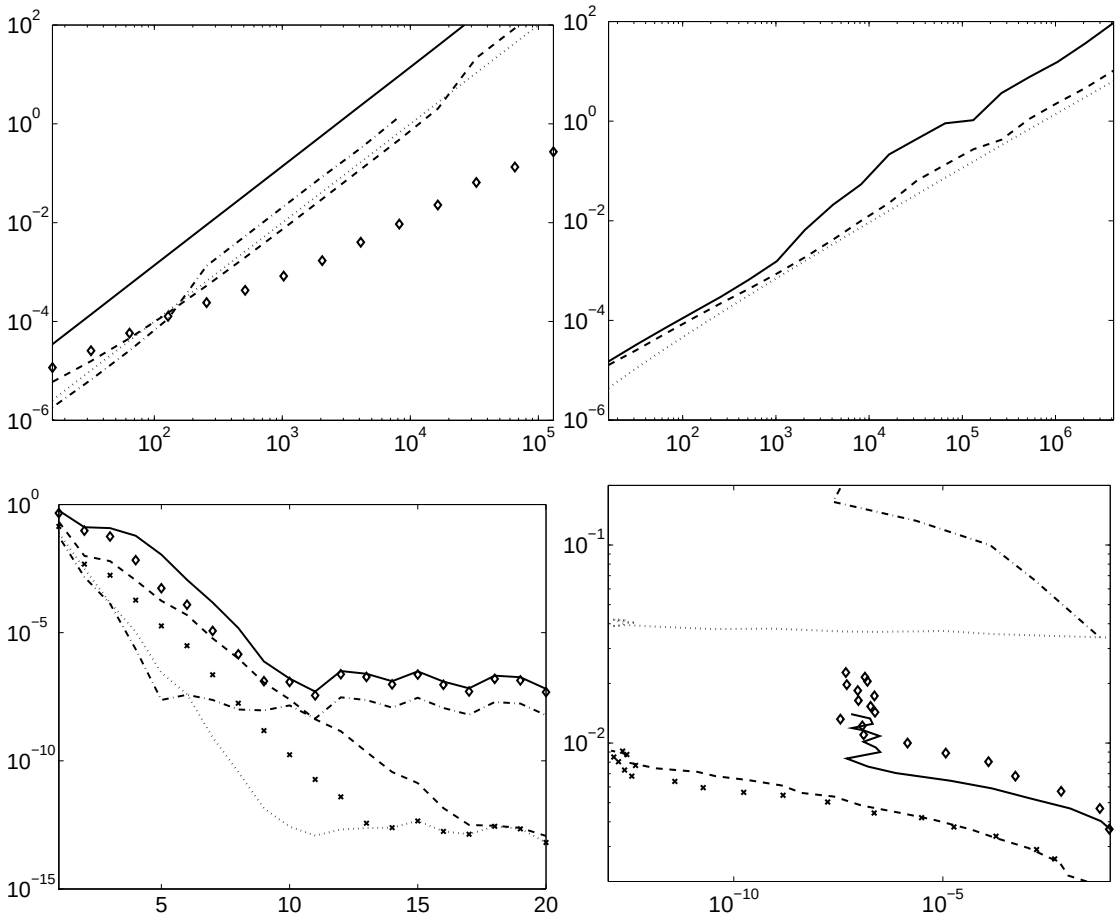

Figure 4.1: Comparison of different NDFTs and the Taylor expansion based NFFT with the (window function based) NFFT for the univariate case $d = 1$. **Top:** Computation time in seconds with respect to increasing degree $N = 2^4, \ldots, 2^{22}$ and $M = N$. Left: NDFT (solid), Horner-like NDFT (dashed), Multiplication with fully precomputed Matrix $\boldsymbol{A}$ (dash-dot), the curve $10^{-8}N^2$ (dotted), and default NFFT, i.e. Kaiser-Bessel window, $\sigma = 2$, $m = 6$ and precomputation methods `PRE_PHI_HUT` and `PRE_PSI` ($\diamond$). Right: Taylor expansion based NFFT with $\sigma = 4$, $m = 6$ (solid), NFFT with $\sigma = 2$, $m = 6$, and precomputed fast Gaussian gridding `PRE_FG_PSI`, which uses the same amount of memory (dashed), and the curve $10^{-7}N \log N$ (dotted). **Bottom:** Accuracy of the Taylor expansion based NFFT and the NFFT with respect to increasing Taylor-order/cut-off $m = 1, \ldots, 20$, fixed degree $N = 4096$ and $M = N$ nodes. Different oversampling factors are denoted for the Taylor expansion based NFFT as $\sigma = 1.5$ (solid), $\sigma = 2$ ($\diamond$), $\sigma = 16$ (dash-dot) and for the NFFT as $\sigma = 1.5$ (dashed), $\sigma = 2$ ($\times$), and $\sigma = 16$ (dotted). The NFFT is used with precomputed fast Gaussian gridding `PRE_FG_PSI`. Left: Accuracy of the Taylor expansion based NFFT with respect to increasing order m of the Taylor expansion and accuracy of the NFFT with respect to increasing cut-off m. Right: Computation time in seconds with respect to achieved target accuracy E_∞.

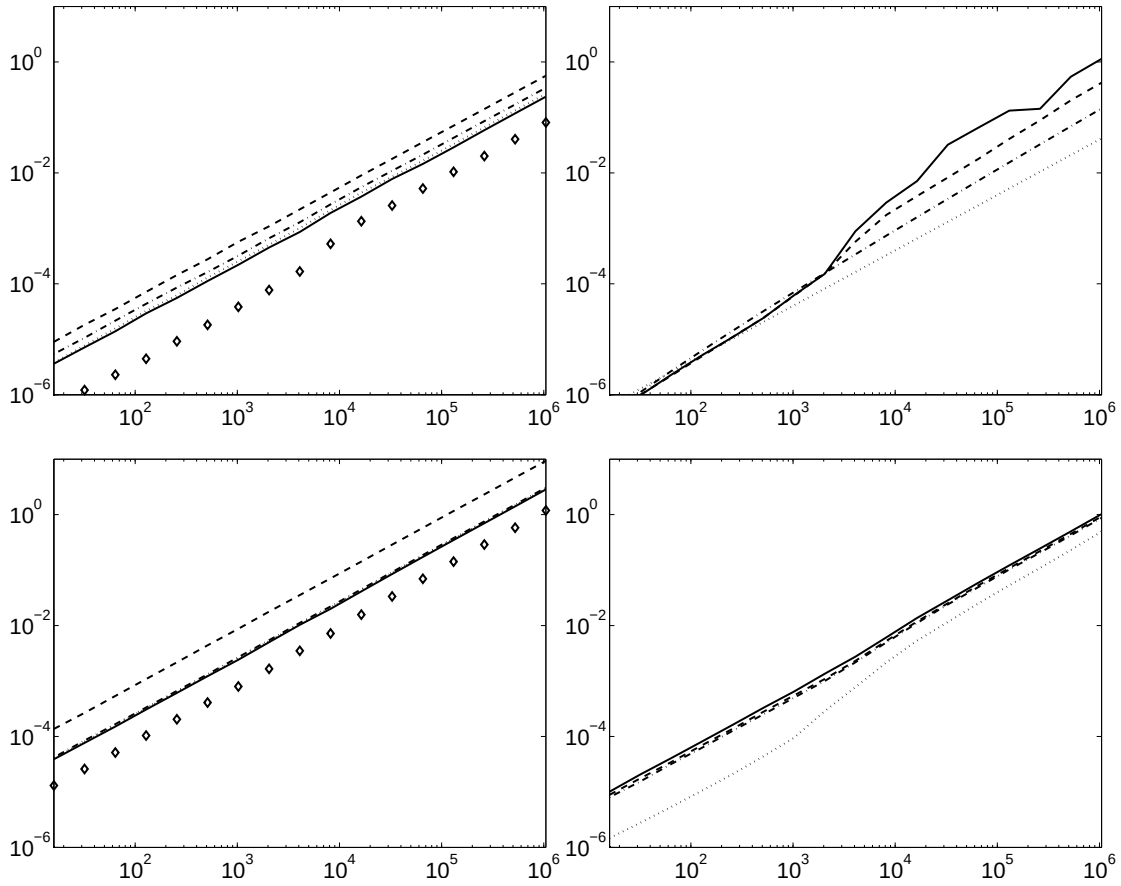

Figure 4.2: Computation time in seconds with respect to increasing degree $N = 2^4, \ldots, 2^{20}$, $M = N$ nodes, $d = 1$, cut-off $m = 4$, and oversampling factor $\sigma = 2$. Window functions are denoted (top-left) and (bottom-left) by: Gaussian (solid), Kaiser-Bessel (dashed), Sinc (dash-dot), and B-Spline (dotted). **Top:** Left: Deconvolution step, i.e., multiplication with the 'diagonal' matrix $\boldsymbol{D}$, where the method with precomputation `PRE_PHI_HUT` is denoted by $\diamond$. Right: Oversampled FFT of length $n = \sigma N$, planner flags are `FFTW_ESTIMATE` (solid) and `FFTW_MEASURE` (dashed). Furthermore, the curves $10^{-8} N \log N$ (dash-dot) and $4 \cdot 10^{-8} N$ (dotted) are shown. **Bottom:** Convolution/evaluation step, i.e., multiplication with the sparse matrix $\boldsymbol{B}$. Left: Comparing the different window functions without any precomputation, denoted as above and the fast Gaussian gridding `FG_PSI` ($\diamond$). Right: Precomputed Gaussian window function with all proposed methods, i.e., `PRE_LIN_PSI` (solid), `PRE_FG_PSI` (dashed), `PRE_PSI` (dash-dot), and `PRE_FULL_PSI` (dotted).

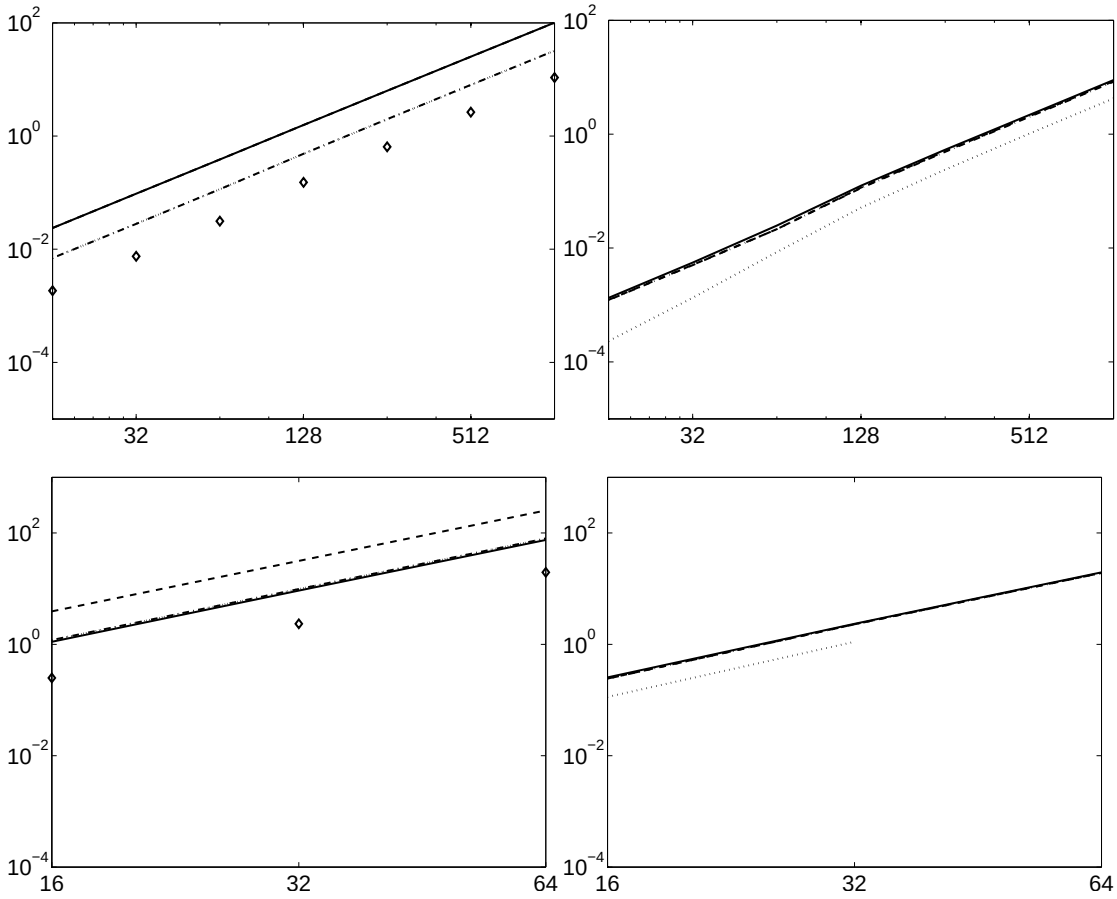

Figure 4.3: Computation time of the multivariate convolution/evaluation step in seconds with respect to increasing multi degree $\boldsymbol{N} = (N, \ldots, N)^\top$, cut-off $m = 4$, and oversampling factor $\sigma = 2$. **Top:** Space dimension $d = 2$, degree $N = 2^4, \ldots, 2^{10}$ and $M = N^2$ nodes. **Bottom:** Space dimension $d = 3$, degree $N = 2^4, 2^5, 2^6$ and $M = N^3$ nodes. Window functions and precomputations are shown as in Figure 4.2 (bottom).

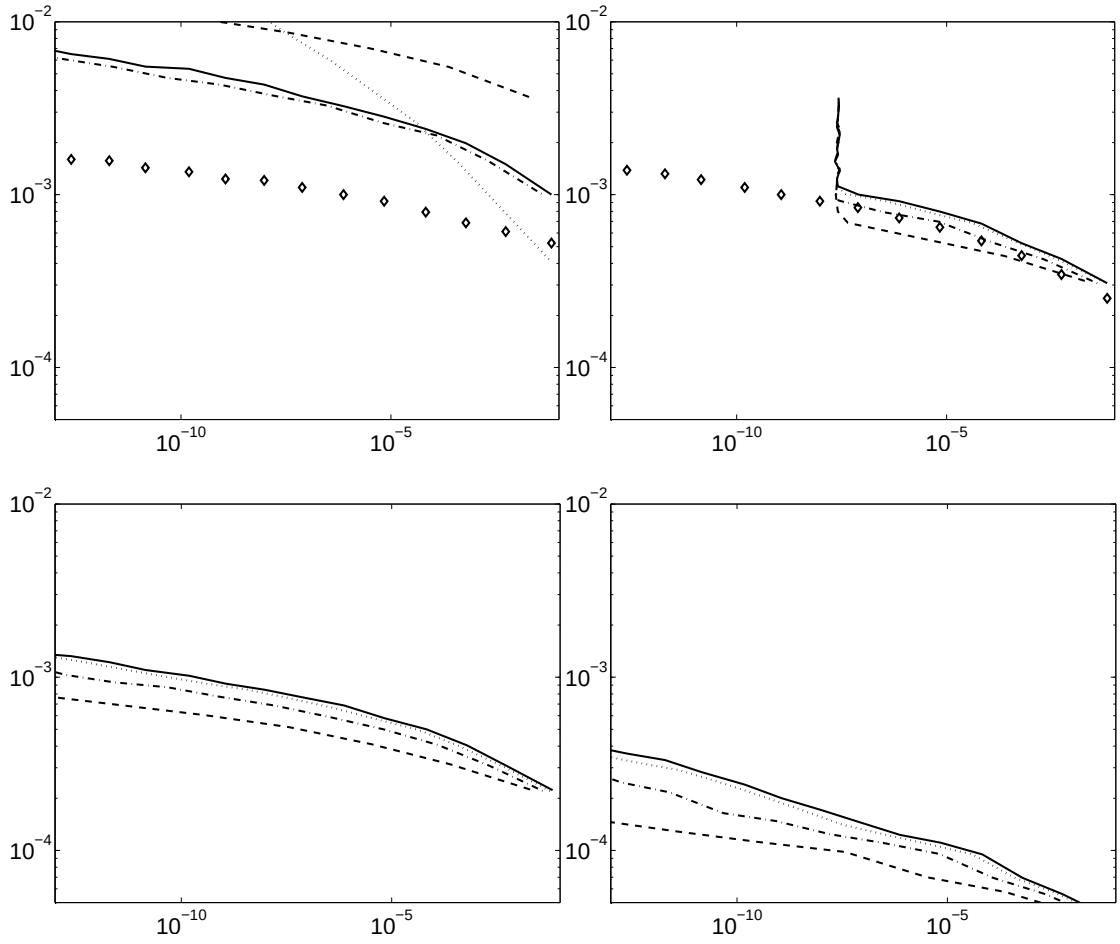

Figure 4.4: Computation time in seconds with respect to target accuracy for increasing cut-off $m = 1, \ldots, 20$, fixed degree $N = 1024$, $M = N$ nodes, and $d = 1$. Window functions: Gaussian (solid), Kaiser-Bessel (dashed), Sinc (dash-dot), and B-Spline (dotted). **Top:** Left: No precomputation, fast Gaussian gridding without precomputation `FG_PSI` is denoted by $\diamond$. Right: Linear interpolated window function `PRE_LIN_PSI` from lookup table with $K = 2^{11}m$ precomputed values achieving single precision 10^{-8}; and fast Gaussian gridding with precomputation `PRE_FG_PSI` ($\diamond$). **Bottom:** Left: Tensor product based precomputation `PRE_PSI`. Right: Fully precomputed matrix $\boldsymbol{B}$, i.e. `PRE_FULL_PSI`.

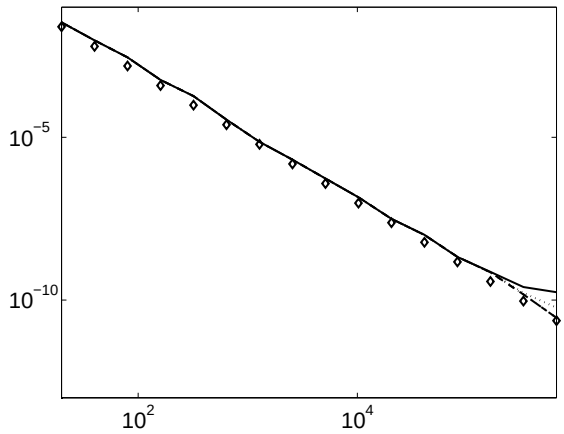

Figure 4.5: Accuracy of the NFFT with linear interpolated window function with respect to the size of the lookup table $K = 2m, 4m, \ldots, 2^{16}m$ for cut-off $m = 10$, fixed degree $N = 1024$, $M = N$ nodes, and $d = 1$.