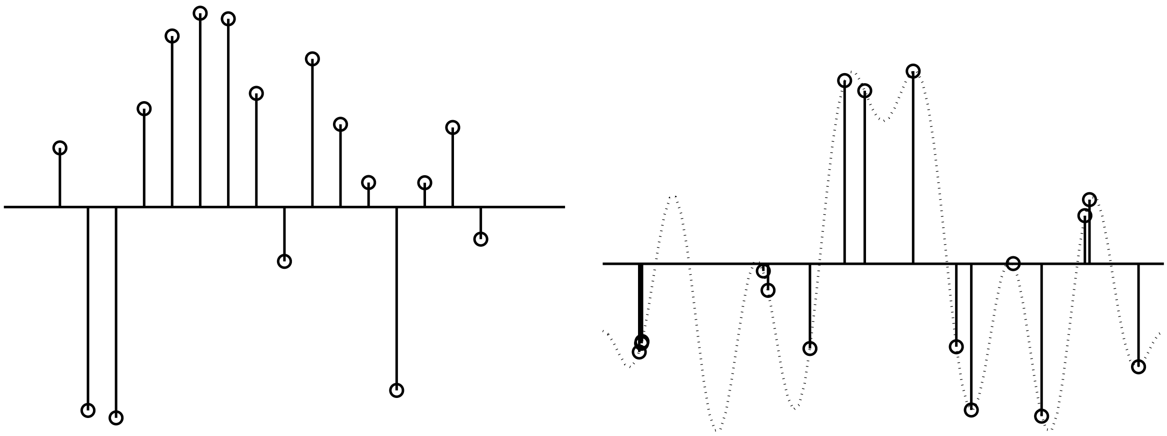
NFFT2.0
May 4, 2005

Stefan Kunis and Daniel Potts
University of Lübeck
Institute of Mathematics
D–23560 Lübeck, Germany
{kunis,potts}@math.uni-luebeck.de



1 Introduction

This technical report gives a short summary of the fast Fourier transform at non equispaced knots (NFFT) and its generalised inverse. An introduction to the NFFT, error estimates etc. can be found in [20]. The computational complexity of the proposed algorithms is $\mathcal{O}\left(N_\Pi \log N_\Pi + M\right)$ for N_Π equispaced frequencies and M non equispaced samples.

The algorithms are implemented in the C-library nfft2 which will be described in detail. The heart of the algorithms are (uniform) fast Fourier transforms for which the library fftw3 (see [12]) is used.

The remainder is organised as follows: in Section 2, the problems of the discrete Fourier transform at non equispaced knots (NDFT) are stated, the used notation is introduced, related works are classified, and the fast algorithm (NFFT) is deduced in pseudo code. Furthermore, an iterative scheme for inversion of the NFFT is presented. The implementation and usage of the library is described in Section 3. Finally, simple examples of usage and numerical results are presented in Section 4.

2 Notation, the NDFT and the NFFT

This section summarises the mathematical theory and ideas behind the NFFT.

2.1 NDFT

Let the torus

$$\mathbb{T}^d := \left\{ \boldsymbol{x} = (x_t)_{t=0,\ldots,d-1} \in \mathbb{R}^d : -\frac{1}{2} \le x_t < \frac{1}{2},\ t = 0, \ldots, d-1 \right\}$$

of dimension d be given. It will serve as domain where the non equispaced knots $\boldsymbol{x}$ are taken from. Thus, the sampling set is given by $\mathcal{X} := \{\boldsymbol{x}_j \in \mathbb{T}^d : j = 0, \ldots, M-1\}$.

The space of all (d-variate, one-periodic) functions $f : \mathbb{T}^d \to \mathbb{C}$ is restricted to the space of d-variate trigonometric polynomials

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

with degree N_t ($t = 0, \ldots, d-1$) in the t-th dimension. Possible frequencies $\boldsymbol{k}$ are collected in the multi index set

$$I_{\boldsymbol{N}} := \left\{ \boldsymbol{k} = (k_t)_{t=0,\ldots,d-1} \in \mathbb{Z}^d : -\frac{N_t}{2} \le k_t < \frac{N_t}{2},\ t = 0, \ldots, d-1 \right\},$$

where $\boldsymbol{N} = (N_t)_{t=0,\ldots,d-1}$ is the multi bandlimit. The dimension of the space of d-variate trigonometric polynomials is given by $N_\Pi = \prod_{t=0}^{d-1} N_t$.

The inner product between the frequency $\boldsymbol{k}$ and the time/spatial knot $\boldsymbol{x}$ is defined in the usual way by $\boldsymbol{k}\boldsymbol{x} := k_0x_0 + k_1x_1 + \ldots + k_{d-1}x_{d-1}$. Furthermore, two vectors may be linked by the pointwise product $\boldsymbol{\sigma} \odot \boldsymbol{N} := (\sigma_0 N_0, \sigma_1 N_1, \ldots, \sigma_{d-1} N_{d-1},)^{\mathrm{T}}$ with its inverse $\boldsymbol{N}^{-1} := \left(\frac{1}{N_0}, \frac{1}{N_1}, \ldots, \frac{1}{N_{d-1}}\right)^{\mathrm{T}}$.

For clarity of presentation the multiindex $\boldsymbol{k}$ adresses elements of vectors and matrices as well, i.e., the plain index $k_\Pi := \sum_{t=0}^{d-1} (k_t + \frac{N_t}{2}) \prod_{t'=t+1}^{d-1} N_{t'}$ is not used here.

Direct NDFT

The first problem to be adressed can be regarded as a matrix vector multiplication. 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 non equispaced knots $\boldsymbol{x}_j \in \mathbb{T}^d$. Thus, our concern is the evaluation of

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

In matrix vector notation this reads as

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

where

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

As already mentioned, all 'vec'-operation are omitted, and the matrix vector product is defined by $\boldsymbol{A}\hat{\boldsymbol{f}} := \operatorname{Vec}(\boldsymbol{A}) \operatorname{vec}\left(\hat{\boldsymbol{f}}\right)$. The straight forward algorithm of this matrix vector product, which is called ndft, takes $\mathcal{O}(MN_\Pi)$ arithmetical operations.

Related matrix vector products are the adjoint ndft

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

the conjugated ndft

$$\boldsymbol{f} = \overline{\boldsymbol{A}}\hat{\boldsymbol{f}}, \qquad f_j = \sum_{\boldsymbol{k}\in I_{\boldsymbol{N}}} \hat{f}_{\boldsymbol{k}} \mathrm{e}^{2\pi \mathrm{i} \boldsymbol{k}\boldsymbol{x}_j},$$

and the transposed ndft

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

where $\boldsymbol{A}^{\mathrm{H}} = \overline{\boldsymbol{A}}^{\mathrm{T}}$.

Equispaced knots

For $N_t = N$ $(t = 0, \ldots, d-1)$, $M = N^d$ and equispaced knots $\boldsymbol{x}_{\boldsymbol{j}} = \frac{1}{N}\boldsymbol{j}$ $(\boldsymbol{j} \in I_{\boldsymbol{N}})$ the computation of (2.2) is known as (multivariate) discrete Fourier transform (DFT). In this special case, the input data $\hat{f}_{\boldsymbol{k}}$ are called discrete Fourier coefficients and the samples $f_{\boldsymbol{j}}$ can be computed by the well known fast Fourier transform (FFT) with only $\mathcal{O}(N_\Pi \log N_\Pi)$ $(N_\Pi = N^d)$ arithmetic operations. Furthermore, one has an inversion formula

$$\boldsymbol{A}\boldsymbol{A}^{\mathrm{H}} = \boldsymbol{A}^{\mathrm{H}}\boldsymbol{A} = N_\Pi \boldsymbol{I},$$

which does **NOT** hold true for the non equispaced case in general.

Inverse NDFT

In general there is no simple inversion formula, hence one deals with the following reconstruction or recovery problem. Given the values $f_j \in \mathbb{C}$ $(j = 0, \ldots, M-1)$ at non equispaced knots $\boldsymbol{x}_j$ $(j = 0, \ldots, M-1)$, the aim is to reconstruct a trigonometric polynomial f resp. its Fourier-coefficients $\hat{f}_{\boldsymbol{k}}$ $(\boldsymbol{k} \in I_{\boldsymbol{N}})$ with

$$f(\boldsymbol{x}_j) = \sum_{\boldsymbol{k}\in I_{\boldsymbol{N}}} \hat{f}_{\boldsymbol{k}} \mathrm{e}^{-2\pi \mathrm{i} \boldsymbol{k}\boldsymbol{x}_j} \approx f_j \quad \text{resp.} \quad \boldsymbol{A}\hat{\boldsymbol{f}} \approx \boldsymbol{f}.$$

Often, the number of nodes M and the dimension of the space of the polynomials N_Π do not coincide, i.e., the matrix $\boldsymbol{A}$ is rectangular. A standard method is to use the Moore-Penrose-pseudoinverse solution $\hat{\boldsymbol{f}}^\dagger$ which solves the general linear least squares problem, see e.g. [3, p. 15],

$$\|\hat{\boldsymbol{f}}\|_2 \to \min \quad \text{subject to} \quad \|\boldsymbol{f} - \boldsymbol{A}\hat{\boldsymbol{f}}\|_2 = \min \quad \text{for} \quad \boldsymbol{f} \in \mathbb{C}^M.$$

Of course, computing the pseudoinverse by the singular value decomposition is very expensive here and no practical way at all.

For a comparative low polynomial degree $N_\Pi < M$ the linear system $\boldsymbol{A}\hat{\boldsymbol{f}} \approx \boldsymbol{f}$ is over-determined, so that in general the given data $f_j \in \mathbb{C},\ j = 0, \ldots, M-1$, will be only approximated up to the *residual* $\boldsymbol{r} := \boldsymbol{f} - \boldsymbol{A}\hat{\boldsymbol{f}}$. One considers the *weighted approximation problem*

$$\|\boldsymbol{f} - \boldsymbol{A}\hat{\boldsymbol{f}}\|_{\boldsymbol{W}} = \left(\sum_{j=0}^{M-1} w_j |f_j - f(x_j)|^2\right)^{1/2} \xrightarrow{\hat{\boldsymbol{f}}} \min,$$

which may incorporate weights $w_j > 0$, $\boldsymbol{W} := \operatorname{diag}(w_j)_{j=0,\ldots,M-1}$, to compensate for clusters in the sampling set $\mathcal{X}$. This problem is equivalent to the *weighted normal equation of first kind*

$$\boldsymbol{A}^{\mathrm{H}}\boldsymbol{W}\boldsymbol{A}\hat{\boldsymbol{f}} = \boldsymbol{A}^{\mathrm{H}}\boldsymbol{W}\boldsymbol{f}. \tag{2.3}$$

For a comparative high polynomial degree $N_\Pi > M$ one expects to interpolate the given data $f_j \in \mathbb{C},\ j = 0, \ldots, M-1$, exactly. The (consistent) linear system $\boldsymbol{A}\hat{\boldsymbol{f}} = \boldsymbol{f}$ is under-determined. One considers the *damped minimisation problem*

$$\|\hat{\boldsymbol{f}}\|_{\hat{\boldsymbol{W}}^{-1}} = \left(\sum_{\boldsymbol{k}\in I_{\boldsymbol{N}}} \hat{w}_{\boldsymbol{k}}^{-1} |\hat{f}_{\boldsymbol{k}}|^2\right)^{1/2} \xrightarrow{\hat{\boldsymbol{f}}} \min \quad \text{subject to} \quad \boldsymbol{A}\hat{\boldsymbol{f}} = \boldsymbol{f}, \tag{2.4}$$

which may incorporate 'damping factors' $\hat{w}_{\boldsymbol{k}} > 0$, $\hat{\boldsymbol{W}} := \operatorname{diag}(\hat{w}_{\boldsymbol{k}})_{\boldsymbol{k}\in I_{\boldsymbol{N}}}$. A smooth solution is favoured, i.e., a decay of the Fourier coefficients $\hat{f}_{\boldsymbol{k}}$, $\boldsymbol{k} \in I_{\boldsymbol{N}}$, for decaying damping factors $\hat{w}_{\boldsymbol{k}}$. This problem is equivalent to the *damped normal equation of second kind*

$$\boldsymbol{A}\hat{\boldsymbol{W}}\boldsymbol{A}^{\mathrm{H}}\tilde{\boldsymbol{f}} = \boldsymbol{f}, \qquad \hat{\boldsymbol{f}} = \hat{\boldsymbol{W}}\boldsymbol{A}^{\mathrm{H}}\tilde{\boldsymbol{f}}. \tag{2.5}$$

2.2 NFFT

For clarity of presentation the ideas behind the NFFT will be shown for the case $d = 1$ and the algorithm ndft. The generalisation of the FFT is an approximative algorithm and has computational complexity $\mathcal{O}\left(N \log N + \log\left(1/\varepsilon\right) M\right)$, where ε denotes the desired accuracy. The main idea is to use standard FFTs and a window function φ which is well localised in the time/spatial domain $\mathbb{R}$ and in the frequency domain $\mathbb{R}$. Several window functions were proposed, see [5, 2, 22, 9, 8].

The considered problem is the fast evaluation of

$$f\left(x\right) = \sum_{k\in I_N} \hat{f}_k \mathrm{e}^{-2\pi\mathrm{i}kx} \tag{2.6}$$

at arbitrary knots $x_j \in \mathbb{T},\ j = 0, \ldots, M-1$.

The ansatz

One wants to approximate the trigonometric polynomial f in (2.6) by a linear combination of shifted 1-periodic window functions $\tilde{\varphi}$ as

$$s_1\left(x\right) := \sum_{l\in I_n} g_l\, \tilde{\varphi}\left(x - \frac{l}{n}\right). \tag{2.7}$$

With the help of an oversampling factor $\sigma > 1$ the fft-length is given by $n := \sigma N$.

The window function

Starting with a window function $\varphi \in L_2(\mathbb{R})$, 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 $\mathbb{Z}$. The periodic window function $\tilde{\varphi}$ may be represented by its Fourier series

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

with the Fourier coefficients

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

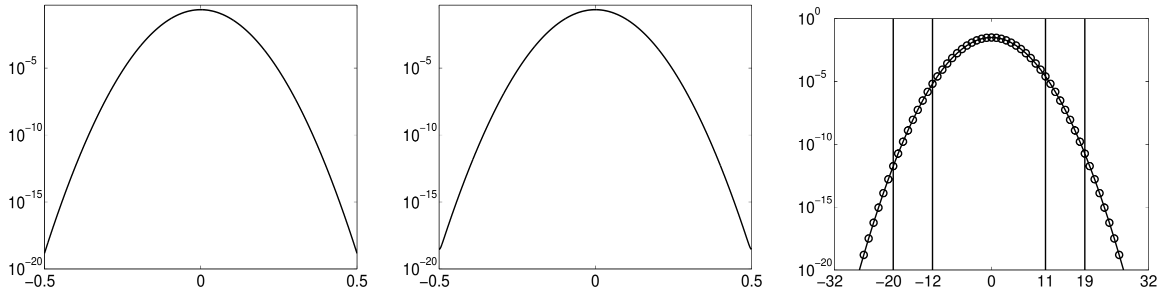


Figure 1: From left to right: Gaussian window funtion φ, its 1-periodic version $\tilde{\varphi}$, and the integral Fourier-transform $\hat{\varphi}$ (with pass, transition, and stop band) for $N = 24$, $\sigma = \frac{4}{3}$, $n = 32$.

The first approximation - cutting off in frequency domain

Switching from the definition (2.7) to the frequency domain, one obtains

$$s_1(x) = \sum_{k \in I_n} \hat{g}_k\, c_k(\tilde{\varphi})\, \mathrm{e}^{-2\pi \mathrm{i} k x} + \sum_{r \in \mathbb{Z} \setminus \{0\}} \sum_{k \in I_n} \hat{g}_k\, c_{k+nr}(\tilde{\varphi})\, \mathrm{e}^{-2\pi \mathrm{i}(k+nr)x} \tag{2.8}$$

with the discrete Fourier coefficients

$$\hat{g}_k := \sum_{l \in I_n} g_l\, \mathrm{e}^{2\pi \mathrm{i} \frac{kl}{n}}. \tag{2.9}$$

Comparing (2.6) to (2.7) and assuming $c_k(\tilde{\varphi})$ small for $|k| \geq n - \frac{N}{2}$ suggests to set

$$\hat{g}_k := \begin{cases} \frac{\hat{f}_k}{c_k(\tilde{\varphi})} & \text{for } k \in I_N, \\ 0 & \text{for } k \in I_n \setminus I_N. \end{cases} \tag{2.10}$$

Then the values g_l can be obtained from (2.9) by

$$g_l = \frac{1}{n} \sum_{k \in I_N} \hat{g}_k\, \mathrm{e}^{-2\pi \mathrm{i} \frac{kl}{n}} \qquad (l \in I_n),$$

a FFT of size n.

This approximation causes an aliasing error.

The second approximation - cutting off in time/spatial domain

If φ is well localised in time/space domain $\mathbb{R}$ it can be approximated by a function

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

with $\operatorname{supp}\psi\ \left[-\frac{m}{n},\frac{m}{n}\right]$, $m \ll n$, $m \in \mathbb{N}$. Again, one defines its one periodic version $\tilde{\psi}$ with compact support in $\mathbb{T}$ as

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

With the help of the index set

$$I_{n,m}(x_j) := \{l \in I_n : nx_j - m \le l \le nx_j + m\}$$

an approximation to s_1 is defined by

$$s(x_j) := \sum_{l\in I_{n,m}(x_j)} g_l\,\tilde{\psi}\left(x_j - \frac{l}{n}\right). \tag{2.11}$$

Note, that for fixed $x_j \in \mathbb{T}$, the above sum contains at most $(2m+1)$ nonzero summands.

This approximation causes a truncation error.

The case $d > 1$

Starting with the original problem of evaluating the multivariate trigonometric polynomial in (2.1) one has to do a few generalisations. The window function is given by

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

where φ_t is an univariate window function. Thus, a simple consequence is

$$c_{\boldsymbol{k}}(\tilde{\varphi}) = c_{k_0}(\tilde{\varphi}_0)\,c_{k_1}(\tilde{\varphi}_1)\dots c_{k_{d-1}}(\tilde{\varphi}_{d-1}).$$

The ansatz is generalised to

$$s_1(\boldsymbol{x}) := \sum_{\boldsymbol{l}\in I_{\boldsymbol{n}}} g_{\boldsymbol{l}}\,\tilde{\varphi}\left(\boldsymbol{x} - \boldsymbol{n}^{-1}\odot\boldsymbol{l}\right),$$

where the fft-size is given by $\boldsymbol{n} := \boldsymbol{\sigma}\odot\boldsymbol{N}$ and the oversampling factors by $\boldsymbol{\sigma} = (\sigma_0,\dots,\sigma_{d-1})^{\mathrm{T}}$. Along the lines of (2.10) one defines

$$\hat{g}_{\boldsymbol{k}} := \begin{cases} \frac{\hat{f}_{\boldsymbol{k}}}{c_{\boldsymbol{k}}(\tilde{\varphi})} & \text{for } \boldsymbol{k}\in I_{\boldsymbol{N}}, \\ 0 & \text{for } \boldsymbol{k}\in I_{\boldsymbol{n}}\backslash I_{\boldsymbol{N}}. \end{cases}$$

The values $g_{\boldsymbol{l}}$ can be obtained by a (multivariate) FFT of size $n_0\times n_1\times\dots\times n_{d-1}$ as

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

Using the compactly supported function $\psi(\boldsymbol{x}) = \varphi(\boldsymbol{x})\,\chi_{[-\frac{m}{n},\frac{m}{n}]^d}(\boldsymbol{x})$, one obtains

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

where $\tilde{\psi}$ again denotes the one periodic version of ψ and the multi index set is given by

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

The algorithm

In summary, the following Algorithm 1 is obtained for the fast computation of (2.2) with $\mathcal{O}\left(n_\Pi \log n_\Pi + mM\right)$ arithmetic operations.

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 $\hat{f}_{\boldsymbol{k}} \in \mathbb{C}$, $\boldsymbol{k} \in I_{\boldsymbol{N}}$,

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

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

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

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

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

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

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

Algorithm 1: `NFFT`

Algorithm 1 reads in matrix vector notation as

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

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

$$\boldsymbol{B} := \left(\tilde{\psi}\left(\boldsymbol{x}_j - \boldsymbol{n}^{-1} \odot \boldsymbol{l}\right)\right)_{j=0,\ldots,M-1;\, \boldsymbol{l} \in I_{\boldsymbol{n}}}, \tag{2.12}$$

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

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

with zero matrices $\boldsymbol{O}_t$ of size $N_t \times \frac{n_t - N_t}{2}$.

The corresponding computation of the adjoint matrix vector product reads as

$$\boldsymbol{A}^{\mathrm{H}}\hat{\boldsymbol{f}} \approx \boldsymbol{D}^{\mathrm{T}}\boldsymbol{F}^{\mathrm{H}}\boldsymbol{B}^{\mathrm{T}}\hat{\boldsymbol{f}}.$$

With the help of the transposed index set

$$I_{\boldsymbol{n},m}^{\mathrm{T}}(\boldsymbol{l}) := \left\{j = 0, \ldots, M-1 : \boldsymbol{l} - m\boldsymbol{1} \leq \boldsymbol{n} \odot \boldsymbol{x}_j \leq \boldsymbol{l} + m\boldsymbol{1}\right\},$$

one obtains Algorithm 2 for the adjoint nfft. Due to the characterisation of the non zero elements of the matrix $\boldsymbol{B}$, i.e.,

$$\bigcup_{j=0}^{M-1} j \times I_{\boldsymbol{n},m}\left(\boldsymbol{x}_j\right) = \bigcup_{\boldsymbol{l} \in I_{\boldsymbol{n}}} I_{\boldsymbol{n},m}^{\mathrm{T}}(\boldsymbol{l}) \times \boldsymbol{l}.$$

the multiplication with the sparse matrix $\boldsymbol{B}^{\mathrm{T}}$ is implemented in a 'transposed' way in the library, summation as outer loop and only using the multi index sets $I_{\boldsymbol{n},m}\left(\boldsymbol{x}_j\right)$.

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 $f_j \in \mathbb{C}$, $j = 0, \ldots, M-1$,

1: For $\boldsymbol{l} \in I_{\boldsymbol{n}}$ compute

$$g_{\boldsymbol{l}} := \sum_{j \in I^{\mathrm{T}}_{\boldsymbol{n},m}(\boldsymbol{l})} f_j \, \tilde{\psi}\left(\boldsymbol{x}_j - \boldsymbol{n}^{-1} \odot \boldsymbol{l}\right) .$$

2: For $\boldsymbol{k} \in I_{\boldsymbol{N}}$ compute by d-variate IFFT

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

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

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

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

Algorithm 2: $\mathtt{NFFT}^{\mathrm{H}}$

2.3 Window functions

Again, only the case $d = 1$ is presented. To keep the aliasing error and the truncation error small, several functions φ with good localisation in time and frequency domain were proposed, e.g. the (dilated) *Gaussian* [5, 22, 4]

$$\begin{aligned} \varphi(x) &= (\pi b)^{-1/2} \, \mathrm{e}^{-\frac{(nx)^2}{b}} \qquad \left(b := \frac{2\sigma}{2\sigma - 1} \frac{m}{\pi}\right), \\ \hat{\varphi}(k) &= \frac{1}{n} \, \mathrm{e}^{-b\left(\frac{\pi k}{n}\right)^2}, \end{aligned} \tag{2.13}$$

(dilated) *cardinal central B–splines* [2, 22]

$$\varphi(x) = M_{2m}(nx), \tag{2.14}$$

$$\hat{\varphi}(k) = \frac{1}{n} \mathrm{sinc}^{2m}(k\pi/n), \tag{2.15}$$

where M_{2m} denotes the centered cardinal B–Spline of order $2m$,
(dilated) *Sinc functions*

$$\begin{aligned} \varphi(x) &= \frac{N(2\sigma - 1)}{2m} \mathrm{sinc}^{2m}\left(\frac{(\pi N x (2\sigma - 1))}{2m}\right), \\ \hat{\varphi}(k) &= M_{2m}\left(\frac{2mk}{(2\sigma - 1) N}\right) \end{aligned} \tag{2.16}$$

and (dilated) *Kaiser–Bessel functions* [15, 9]

$$\begin{aligned}
\varphi(x) &= \frac{1}{\pi}\begin{cases} \dfrac{\sinh\left(b\sqrt{m^2-n^2x^2}\right)}{\sqrt{m^2-n^2x^2}} & \text{for } |x| \le \frac{m}{n} \qquad \left(b := \pi\left(2-\frac{1}{\sigma}\right)\right), \\[2ex] \dfrac{\sin\left(b\sqrt{n^2x^2-m^2}\right)}{\sqrt{n^2x^2-m^2}} & \text{otherwise}, \end{cases} \\
\hat{\varphi}(k) &= \frac{1}{n}\begin{cases} I_0\left(m\sqrt{b^2-(2\pi k/n)^2}\right) & \text{for } k = -n\left(1-\frac{1}{2\sigma}\right),\ldots,n\left(1-\frac{1}{2\sigma}\right), \\ 0 & \text{otherwise}, \end{cases}
\end{aligned} \tag{2.17}$$

where I_0 denotes the *modified zero–order Bessel function*. For these functions φ it has been proven that

$$|f(\boldsymbol{x}_j) - s(\boldsymbol{x}_j)| \le C(\sigma, m)\, \|\hat{\boldsymbol{f}}\|_1$$

where

$$C(\sigma, m) := \begin{cases} 4\, \mathrm{e}^{-m\pi(1-1/(2\sigma-1))} & \text{for } (2.13)\ [22], \\ 4\left(\frac{1}{2\sigma-1}\right)^{2m} & \text{for } (2.14)\ [22], \\ \frac{1}{m-1}\left(\frac{2}{\sigma^{2m}} + \left(\frac{\sigma}{2\sigma-1}\right)^{2m}\right) & \text{for } (2.16), \\ 4\pi\left(\sqrt{m}+m\right)\sqrt[4]{1-\frac{1}{\sigma}}\ \mathrm{e}^{-2\pi m\sqrt{1-1/\sigma}} & \text{for } (2.17). \end{cases}$$

Thus, for fixed $\sigma > 1$, the approximation error introduced by the NFFT decays exponentially with the number m of summands in (2.11). Using the tensor product approach the above error estimates can be generalised for the multivariate setting [6]. On the other hand, the complexity of the NFFT increases with m.

Further NFFT papers

Several papers have described fast approximations for the NFFT. Common names for NFFT are *non-uniform fast Fourier transform* [8], *generalized fast Fourier transform* [5], *unequally-spaced fast Fourier transform* [2], *fast approximate Fourier transforms for irregularly spaced data* [23], *non-equispaced fast Fourier transform* [10] or *gridding* [21, 15, 19].

In various papers, different window functions were considered, e.g. Gaussian pulse tapered with a Hanning window in [4], Gaussian kernels combined with sinc kernels in [19], and special optimised windows in [15, 4]. Furthermore, special approaches based on scaling vectors [17], based on minimising the Frobenius norm of certain error matrices [18] or based on min-max interpolation [8] are proposed. However, the numerical results in [18, 8] show that these approaches are not superior to the approach based on Kaiser-Bessel functions.

An even more general FFT, where both the knots in frequency and time/spatial domain are arbitrary, was developed in [6]. Up to now, the library does not support this extension.

Our algorithms are based on the approach in [20]. Here, one can change the window functions in a very simple way. See Section 3 and Section 4 for a suitable choice of the window function with respect to accuracy, speed and memory usage.

2.4 Inverse NFFT

As already mentioned, the reconstruction or recovery problem is to find for given data $\boldsymbol{f} \in \mathbb{C}^M$ a suitable vector of Fourier coefficients $\hat{\boldsymbol{f}} \in \mathbb{C}^{N_\Pi}$ satisfying $\boldsymbol{A}\hat{\boldsymbol{f}} \approx \boldsymbol{f}$. Starting from the normal equations (2.3) and (2.5) it has been proven, that these are well conditioned for (see [7, 1])

$$N_t < C_d \delta^{-1}\,, \qquad \delta := \max_{j,l=0,\ldots,M-1} \min_{l\neq j} \operatorname{dist}_\infty(\boldsymbol{x}_j, \boldsymbol{x}_l)\,,$$

or (see [16])

$$N_t > C_{\beta,d} q^{-1+\frac{1-d}{\beta}}\,, \qquad q := \min_{j,l=0,\ldots,M-1; l\neq j} \operatorname{dist}_\infty(\boldsymbol{x}_j, \boldsymbol{x}_l)\,,$$

where $t = 0, \ldots, d-1$. The mesh norm δ or the separation distance q have to be bounded with respect to the polynomial degree N_Π. Once, a suitable multi bandwidth $\boldsymbol{N}$ has been chosen, one may apply one of the following iterative algorithms. The implemented algorithms are given below in pseudocode, see also [3, 14]. Algorithm 3 is the only algorithm which computes the original residual $\boldsymbol{r}_l$ in each step, all other algorithms iterate the residual. Algorithm 1 and 2 are used for the matrix vector multiplication with $\boldsymbol{A}$ and $\boldsymbol{A}^{\mathrm{H}}$, respectively.

Input: $\boldsymbol{f} \in \mathbb{C}^M$, $\hat{\boldsymbol{f}}_0 \in \mathbb{C}^{N_\Pi}$
1: $\boldsymbol{r}_0 = \boldsymbol{f} - \boldsymbol{A}\hat{\boldsymbol{f}}_0$
2: $\hat{\boldsymbol{z}}_0 = \boldsymbol{A}^{\mathrm{H}}\boldsymbol{W}\boldsymbol{r}_0$
3: **for** $l = 0, \ldots$ **do**
4: $\quad \hat{\boldsymbol{f}}_{l+1} = \hat{\boldsymbol{f}}_l + \alpha \hat{\boldsymbol{W}}\hat{\boldsymbol{z}}_l$
5: $\quad \boldsymbol{r}_{l+1} = \boldsymbol{f} - \boldsymbol{A}\hat{\boldsymbol{f}}_{l+1}$
6: $\quad \hat{\boldsymbol{z}}_{l+1} = \boldsymbol{A}^{\mathrm{H}}\boldsymbol{W}\boldsymbol{r}_{l+1}$
7: **end for**
Output: $\hat{\boldsymbol{f}}_l$

Algorithm 3: `LANDWEBER`

Input: $\boldsymbol{f} \in \mathbb{C}^M$, $\hat{\boldsymbol{f}}_0 \in \mathbb{C}^{N_\Pi}$
1: $\boldsymbol{r}_0 = \boldsymbol{f} - \boldsymbol{A}\hat{\boldsymbol{f}}_0$
2: $\hat{\boldsymbol{z}}_0 = \boldsymbol{A}^{\mathrm{H}}\boldsymbol{W}\boldsymbol{r}_0$
3: **for** $l = 0, \ldots$ **do**
4: $\quad \boldsymbol{v}_l = \boldsymbol{A}\hat{\boldsymbol{W}}\hat{\boldsymbol{z}}_l$
5: $\quad \alpha_l = \dfrac{\hat{\boldsymbol{z}}_l^{\mathrm{H}}\hat{\boldsymbol{W}}\hat{\boldsymbol{z}}_l}{\boldsymbol{v}_l^{\mathrm{H}}\boldsymbol{W}\boldsymbol{v}_l}$
6: $\quad \hat{\boldsymbol{f}}_{l+1} = \hat{\boldsymbol{f}}_l + \alpha_l \hat{\boldsymbol{W}}\hat{\boldsymbol{z}}_l$
7: $\quad \boldsymbol{r}_{l+1} = \boldsymbol{r}_{l+1} - \alpha_l \boldsymbol{v}_l$
8: $\quad \hat{\boldsymbol{z}}_{l+1} = \boldsymbol{A}^{\mathrm{H}}\boldsymbol{W}\boldsymbol{r}_{l+1}$
9: **end for**
Output: $\hat{\boldsymbol{f}}_l$

Algorithm 4: `STEEPEST_DESCENT`

The memory usage of the iterative algorithms are given in the following Table 1.

Input: $\boldsymbol{f} \in \mathbb{C}^M$, $\hat{\boldsymbol{f}}_0 \in \mathbb{C}^{N_\Pi}$

if `ITERATE_2nd` **then**

  $\hat{\boldsymbol{f}}_0^{\mathrm{cgne}} = \hat{\boldsymbol{f}}_0$

end if

$\boldsymbol{r}_0 = \boldsymbol{f} - \boldsymbol{A}\hat{\boldsymbol{f}}_0$

$\hat{\boldsymbol{z}}_0 = \boldsymbol{A}^{\mathrm{H}}\boldsymbol{W}\boldsymbol{r}_0$

$\hat{\boldsymbol{p}}_0 = \hat{\boldsymbol{z}}_0$

for $l = 0, \dots$ **do**

  $\boldsymbol{v}_l = \boldsymbol{A}\hat{\boldsymbol{W}}\hat{\boldsymbol{p}}_l$

  $\alpha_l = \dfrac{\hat{\boldsymbol{z}}_l^{\mathrm{H}}\hat{\boldsymbol{W}}\hat{\boldsymbol{z}}_l}{\boldsymbol{v}_l^{\mathrm{H}}\boldsymbol{W}\boldsymbol{v}_l}$

  $\hat{\boldsymbol{f}}_{l+1} = \hat{\boldsymbol{f}}_l + \alpha_l\hat{\boldsymbol{W}}\hat{\boldsymbol{p}}_l$

  if `ITERATE_2nd` **then**

    $\alpha_l^{\mathrm{cgne}} = \dfrac{\boldsymbol{r}_l^{\mathrm{H}}\boldsymbol{W}\boldsymbol{r}_l}{\hat{\boldsymbol{z}}_l^{\mathrm{H}}\hat{\boldsymbol{W}}\hat{\boldsymbol{z}}_l}$

    $\hat{\boldsymbol{f}}_{l+1}^{\mathrm{cgne}} = \hat{\boldsymbol{f}}_l^{\mathrm{cgne}} + \alpha_l^{\mathrm{cgne}}\hat{\boldsymbol{W}}\hat{\boldsymbol{z}}_l$

  end if

  $\boldsymbol{r}_{l+1} = \boldsymbol{r}_l - \alpha_l\boldsymbol{v}_l$

  $\hat{\boldsymbol{z}}_{l+1} = \boldsymbol{A}^{\mathrm{H}}\boldsymbol{W}\boldsymbol{r}_{l+1}$

  $\beta_l = \dfrac{\hat{\boldsymbol{z}}_{l+1}^{\mathrm{H}}\hat{\boldsymbol{W}}\hat{\boldsymbol{z}}_{l+1}}{\hat{\boldsymbol{z}}_l^{\mathrm{H}}\hat{\boldsymbol{W}}\hat{\boldsymbol{z}}_l}$

  $\hat{\boldsymbol{p}}_{l+1} = \beta_l\hat{\boldsymbol{p}}_l + \hat{\boldsymbol{z}}_{l+1}$

end for

Output: $\hat{\boldsymbol{f}}_l$, $\hat{\boldsymbol{f}}_l^{\mathrm{cgne}}$

Algorithm 5: `CGNR_E`

Algorithm	Memory usage
`LANDWEBER`	$2M + 2N_\Pi$
`STEEPEST_DESCENT`	$3M + 2N_\Pi$
`CGNR_E`	$3M + (3\,(+1))\,N_\Pi$
`CGNE_R`	$2M + (2\,(+1))\,N_\Pi$

Table 1: Memory usage of the iterative schemes.

Input: $\boldsymbol{f} \in \mathbb{C}^{M}$, $\hat{\boldsymbol{f}}_0 \in \mathbb{C}^{N_\Pi}$

if `ITERATE_2nd` **then**

  $\gamma_0 = 1$

  $\hat{\boldsymbol{f}}_0^{\mathrm{cgnr}} = \hat{\boldsymbol{f}}_0$

end if

$\boldsymbol{r}_0 = \boldsymbol{f} - \boldsymbol{A}\hat{\boldsymbol{f}}_0$

$\hat{\boldsymbol{p}}_0 = \boldsymbol{A}^{\mathrm{H}}\boldsymbol{W}\boldsymbol{r}_0$

for $l = 0, \ldots$ **do**

  $\alpha_l = \dfrac{\boldsymbol{r}_l^{\mathrm{H}}\boldsymbol{W}\boldsymbol{r}_l}{\hat{\boldsymbol{p}}_l^{\mathrm{H}}\hat{\boldsymbol{W}}\hat{\boldsymbol{p}}_l}$

  $\hat{\boldsymbol{f}}_{l+1} = \hat{\boldsymbol{f}}_l + \alpha_l \hat{\boldsymbol{W}}\hat{\boldsymbol{p}}_l$

  $\boldsymbol{r}_{l+1} = \boldsymbol{r}_l - \alpha_l \boldsymbol{A}\hat{\boldsymbol{W}}\hat{\boldsymbol{p}}_l$

  $\beta_l = \dfrac{\boldsymbol{r}_{l+1}^{\mathrm{H}}\boldsymbol{W}\boldsymbol{r}_{l+1}}{\boldsymbol{r}_l^{\mathrm{H}}\boldsymbol{W}\boldsymbol{r}_l}$

  if `ITERATE_2nd` **then**

    $\gamma_{l+1} = \beta_l \gamma_l + 1$

    $\hat{\boldsymbol{f}}_{l+1}^{\mathrm{cgnr}} = \frac{1}{\gamma_{l+1}}\left(\beta_l \gamma_l \hat{\boldsymbol{f}}_l^{\mathrm{cgnr}} + \hat{\boldsymbol{f}}_{l+1}\right)$

  end if

  $\hat{\boldsymbol{p}}_{l+1} = \beta_l \hat{\boldsymbol{p}}_l + \boldsymbol{A}^{\mathrm{H}}\boldsymbol{W}\boldsymbol{r}_{l+1}$

end for

Output: $\hat{\boldsymbol{f}}_l$, $\hat{\boldsymbol{f}}_l^{\mathrm{cgnr}}$

Algorithm 6: `CGNE_R`

3 Library

The library is completely written in C and uses the *FFTW library* (see [12]), which has to be installed on your system. Algorithms 1 and 2 are implemented for arbitrary dimensions $d = 1, 2, \ldots$. The library has several options (determined at compile time) and parameters (determined at run time).

3.1 Installation

Download and make

1. Install the FFTW library (version 3), available from `www.fftw.org`. Generally this will mean
   `> ./configure --prefix <FFTW_PATH>`
   `> make`
   `> make install`

2. Download from `www.math.uni-luebeck.de/potts/nfft`
   `> tar xfv nfft2.tar` will create a directory `./nfft2`
   `./nfft2> ./configure <FFTW_PATH>` creates all makefiles
   `./nfft2> make` creates the library and all examples

The following options are determined at compile time, realised in the `Makefile`s and in the file `./include/options.h`, respectively.

Options

The adaptation of the C-library is done in `./include/options.h`. One chooses between window functions by defining **one** of the following constant symbols: `KAISER_BESSEL, SINC_2m, GAUSSIAN, B_SPLINE`.

Furthermore, the fast transforms can be instrumented such that the elapsed time for each step of Algorithm 1 and 2 is printed to `stderr`, respectively. This option should help to customise the library to one's needs. One can enable this option by defining `MEASURE_TIME` in `options.h`.

3.2 NFFT - General procedure

One has to follow certain steps to write a simple program using the NFFT library. The first argument of each function is a pointer to a application-owned variable of type `nfft_plan`. The aim of this structure is to keep interfaces small, it contains all parameters and data.

Initialisation

Initialisation of a plan is done by one of the `nfft_init`-functions. The simplest version for the univariate case $d = 1$ just specifies the number of Fourier coefficients N_0 and the number of non equispaced knots M. For an application-owned variable `nfft_plan my_plan` the function call is

```
nfft_init_1d(&my_plan,N0,M);
```

The first argument should be uninitialised. Memory allocation is completely done by the init routine.

Setting knots

One has to define the knots $\boldsymbol{x}_j \in \mathbb{T}^d$ for the transformation in the member variable `my_plan.x`. The t-th coordinate of the j-th knot $\boldsymbol{x}_j$ has to be assigned to

```
my_plan.x[d*j+t]= /*your choice*/;
```

Precompute $\tilde{\psi}$

The precomputation of the values $\tilde{\psi}\left(\boldsymbol{x}_j - \boldsymbol{n}^{-1} \odot \boldsymbol{l}\right)$ depends on the choice for `my_plan.x`. If and only if the nfft-flag `PRE_PSI` is set, the subroutine `nfft_precompute_psi(&my_plan)` has to be called, i.e.

```
if(my_plan.nfft_flags & PRE_PSI) nfft_precompute_psi(&my_plan);
```

Doing the transform

Algorithm 1 is implemented as

```
nfft_trafo(&my_plan);
```

One only needs one plan for several transforms of the same kind, i.e. transforms with equal initialisation parameters.

Finalisation

All memory allocated by the init routine is deallocated by

```
nfft_finalize(&my_plan);
```

Note, that almost all (de)allocation operations of the library are done by `fftw_malloc` and `fftw_free`. Additional data, declared and allocated by the application, have to be deallocated by the user's program as well.

Example

Thus, the following code computes a univariate nfft with 12 Fourier coefficients and 19 knots.

```
void simple_test_nfft_1d()
{
  int j,k;
  nfft_plan my_plan;

  nfft_init_1d(&my_plan,12,19);

  for(j=0;j<my_plan.M;j++)
    my_plan.x[j]=((double)rand())/RAND_MAX-0.5;

  if(my_plan.nfft_flags & PRE_PSI)
    nfft_precompute_psi(&my_plan);

  for(k=0;k<my_plan.N_L;k++)
    {
```

```
      my_plan.f_hat[k][0]=((double)rand())/RAND_MAX;
      my_plan.f_hat[k][1]=((double)rand())/RAND_MAX;
    }

  nfft_trafo(&my_plan);

  nfft_finalize(&my_plan);
}
```

3.3 NFFT - Parameter

All parameters are stored in the plan. Using the simple initialisation `nfft_init`, one has only to define the multi bandwidth $\boldsymbol{N}$ and the number of knots M. All other parameters are set to default values. The specific initialisation `nfft_init_specific` may be used to customise the library to one's needs.

name	default	description
$\boldsymbol{N}$		multivariate bandwidth
M		number of knots
$\boldsymbol{n}$	$2^{\lceil \log \boldsymbol{N} \rceil}$	FFTW-size
m	see Table 3	cut off parameter in time domain
`nfft_flags`	`PRE_PSI| PRE_PHI_HUT| MALLOC_X| MALLOC_F_HAT| MALLOC_F| FFTW_INIT| FFT_OUT_OF_PLACE`	disjunction of flags for memory allocation
`fftw_flags`	`FFTW_ESTIMATE| FFTW_DESTROY_INPUT`	disjunction of flags for FFTW

Table 2: Parameters

In Table 3 we give the default values for the cut off parameter m. This parameter depends on the window function and is choosen such that the error E_∞ (see Section 4) is smaller than 10^{-12}.

window function	Kaiser-Bessel	sinc^{2m}	B-spline	Gaussian
m	6	9	11	12

Table 3: Default values for the cut-off parameter m, see also Section 4,Figure 3.

Flags

The flags, set in `nfft_flags` concern memory allocation and precomputations. For the knot vector $\boldsymbol{x}$, the vector of Fourier coefficients $\hat{\boldsymbol{f}}$, and the vector of samples $\boldsymbol{f}$ memory is allocated iff `MALLOC_X`, `MALLOC_F_HAT`, and `MALLOC_F` are set, respectively.

The flag `FFTW_INIT` causes the library to initialise the plans for the fft; iff `FFT_OUT_OF_PLACE` is set, the vectors $\boldsymbol{g}$ and $\hat{\boldsymbol{g}}$ have their own allocated memory, otherwise they share.

Precomputation can be customised by the flags `PRE_PHI_HUT`, `PRE_PSI`, and `PRE_FULL_PSI`; `PRE_PHI_HUT` causes the initialisation routine to precompute the values $c_{\boldsymbol{k}}(\tilde{\phi})$ in their tensor product form. The flag `PRE_PSI` causes the allocation of memory for $\tilde{\psi}\left(\boldsymbol{x}_j - \boldsymbol{n}^{-1} \odot \boldsymbol{l}\right)$, used in the 3rd step of Algorithm 1. The routine nfft_precompute_psi has to be called for the actual precomputation. Again, the library uses the tensor product form, which leads to a memory usage of $d(2m+2)$ doubles per knot, but an arithmetic complexity of $d(2m+2)^d$ per knot; similar to what is called 'fast gridding' in [13], where the storage requirements are further decreased for the Gaussian window function. In contrast, if in addition the flag PRE_FULL_PSI is set, the library uses $(2m+2)^d$ doubles and $2(2m+2)^d$ integers (for indices) per knot, which leads to an arithmetic complexity of $(2m+2)^d$ per knot. In this case the nfft_full_psi has to be called.

3.4 NFFT - Data

Data layout

All data with multi index are stored *plain*, e.g. $\hat{f}_{\boldsymbol{k}}$ in my_plan.f_hat[k_Π], where the plain index again is given by $k_\Pi := \sum_{t=0}^{d-1}(k_t + \frac{N_t}{2})\prod_{t'=t+1}^{d-1} N_{t'}$ and the knots $(\boldsymbol{x}_j)_t$ in my_plan.x[$dj+t$].

Exchanging data

All routines work on the structure my_plan. One can exchange data with this structure if one wants to use several transforms. One can do this by declaring, allocating memory (fftw_malloc), initiatisation (optionally) and using the SWAPC macro, e.g.

```
SWAPC(my_plan.f_hat,new_f_hat).
```

Note that the data has the right type and size (see Table 4). Memory deallocation of all 'free' data has to be done by the user's program. Note, that the vectors my_plan.g and my_plan.g_hat must not be exchanged.

3.5 NFFT - Plan

The library defines the structure nfft_plan. The members are int d, int* N, int M, double* sigma, int* n, int m, double* b, int nfft_flags, int fftw_flags (already discussed, see Table 2). Furthermore, it contains all data vectors, see Table 4.

type	name	description	size (in *type)
double*	x	knots $\boldsymbol{x}_j \in \mathbb{T}^d$	dM
fftw_complex*	f_hat	Fourier coefficients	N_Π
fftw_complex*	f	samples	M
double**	c_phi_inv	precomputed $c_{\boldsymbol{k}}(\tilde{\varphi})^{-1}$	$\sum_{t=0}^{d-1} N_t$
double*	psi	precomputed $\tilde{\psi}\left(\boldsymbol{x}_j - \boldsymbol{n}^{-1}\odot\boldsymbol{l}\right)$ or (flags *_PSI)	$d\,(2m+2)\,M$ $(2m+2)^d\,M$
fftw_complex*	g	$g_{\boldsymbol{l}}$	n_Π
fftw_complex*	g_hat	$\hat{g}_{\boldsymbol{k}}$	(n_Π)

Table 4: Members of nfft_plan.

3.6 iNFFT - General procedure

The following Figure 2 shows how to use the inverse nfft. There is no general stopping rule implemented, since this task is highly dependent on the special application. A simple example can be found in example/simple_test/.

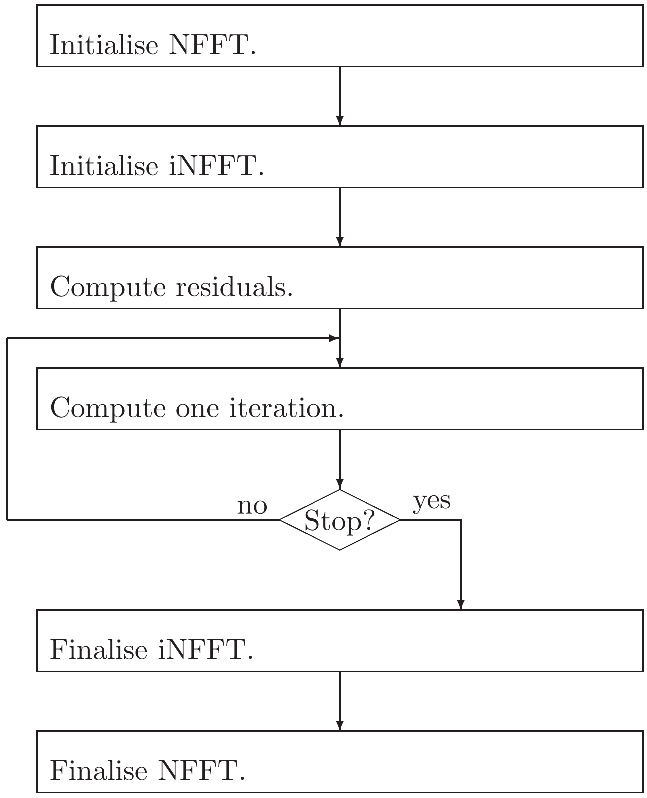


Figure 2: Control flow of the iNFFT.

3.7 iNFFT - Parameter

The inverse nfft basically wraps an already initialised direct nfft (`direct_plan`), i.e. one only specifies the type of iteration by setting one of

`LANDWEBER`, `STEEPEST_DESCENT`, `CGNR_E`, `CGNE_R`.

in `infft_flags`. The additional flag `NORMS_FOR_LANDWEBER`, only applicape if `LANDWEBER` is set, causes the library to compute the residuals $\|\boldsymbol{r}_l\|_{\boldsymbol{W}}^2$ and $\|\hat{\boldsymbol{z}}_l\|_{\hat{\boldsymbol{W}}}^2$; `ITERATE_2nd`, only applicable if `CGNR_E` or `CGNR_E` is set, computes the cgne/cgnr-iterate out of the cgnr/cgne-algorithm, see [14] for details. Weights and damping factors are used if the flags

`PRECOMPUTE_WEIGHT`, `PRECOMPUTE_DAMP`

are specified, respectively. One has to initialise the members `my_iplan.w`, `my_iplan.w_hat` in these cases. The utility library, see `lib/utils.c`, provides a set of functions for computing voronoi weights and certain damping factors, see Subsection 4.3. The default for `infft_flags` is `CGNR_E`.

3.8 iNFFT - Plan

The library defines the structure `infft_plan`. The members `direct_plan` and `infft_flags` are already discussed. Table 5 shows part of the remaining members.

type	name	description	size (in *type)
double*	w	weights $\boldsymbol{w}$	M
double*	w_hat	damping factors $\hat{\boldsymbol{w}}$	N_Π
fftw_complex*	given_f	right hand side	M
fftw_complex*	f_hat_iter	actual solution	N_Π
fftw_complex*	f_hat_iter_2nd	2nd solution (ITERATE_2nd)	N_Π
fftw_complex*	r_iter	residual vector $\boldsymbol{r}_{l+1}$	M
double	dot_r_iter	$\|\boldsymbol{r}_{l+1}\|_{\boldsymbol{W}}^2$	
double	dot_r_iter_old	$\|\boldsymbol{r}_l\|_{\boldsymbol{W}}^2$	

Table 5: More important members of infft_plan.

3.9 User functions

All user functions have return type void and their first argument is of type nfft_plan* or infft_plan*, respectively. Table 6 shows all available functions.

name	additional arguments
ndft_trafo	
ndft_conjugated	
ndft_adjoint	
ndft_transposed	
nfft_init_1d	int N0, int M
nfft_init_2d	int N0, int N1, int M
nfft_init_3d	int N0, int N1, int N2, int M
nfft_init	int d, int *N, int M
nfft_init_specific	int d, int *N, int M, int *n, int m, unsigned nfft_flags, unsigned fftw_flags
nfft_precompute_psi	
nfft_full_psi	double eps
nfft_trafo	
nfft_conjugated	
nfft_adjoint	
nfft_transposed	
nfft_finalize	
infft_init	nfft_plan *direct_plan
infft_init_specific	nfft_plan *direct_plan, unsigned infft_flags
infft_before_loop	
infft_loop_one_step	
infft_finalize	

Table 6: User functions.

4 Examples

One may start with `example/simple_test/`, where the usage of the library is presented for small problems. More elaborated examples in `example/error_decay, franke, glacier, interpolation_1d, lena` show the usage of the inverse nfft. Note, that these examples use MatLab for their visualisation.

The library was tested on a AMD Athlon(tm) XP 2700+, 1GB memory, SuSe-Linux, kernel 2.4.20-4GB-athlon, gcc version 3.3. In all test cases the knots $\boldsymbol{x}_j$ and the Fourier coefficients $\hat{f}_{\boldsymbol{k}}$ are chosen pseudo random with $\boldsymbol{x}_j \in [-0.5, 0.5]^d$ and $\hat{f}_{\boldsymbol{k}} \in [0,1] \times [0,1]\,\mathrm{i}$.

4.1 Accuracy & m

The accuracy of the Algorithm 1, measured by

$$E_2 = \frac{\|\boldsymbol{f} - \boldsymbol{s}\|_2}{\|\boldsymbol{f}\|_2} = \left(\sum_{j \in I_M^1} |f_j - s(\boldsymbol{x}_j)|^2 / \sum_{\boldsymbol{j} \in I_M^1} |f_j|^2 \right)^{\frac{1}{2}}$$

and

$$E_\infty = \frac{\|\boldsymbol{f} - \boldsymbol{s}\|_\infty}{\|\hat{\boldsymbol{f}}\|_1} = \max_{j \in I_M^1} |f_j - s(\boldsymbol{x}_j)| / \sum_{\boldsymbol{k} \in I_{\boldsymbol{N}}} |\hat{f}_{\boldsymbol{k}}|$$

is shown in Figure 3, see `./example/accuracy`.

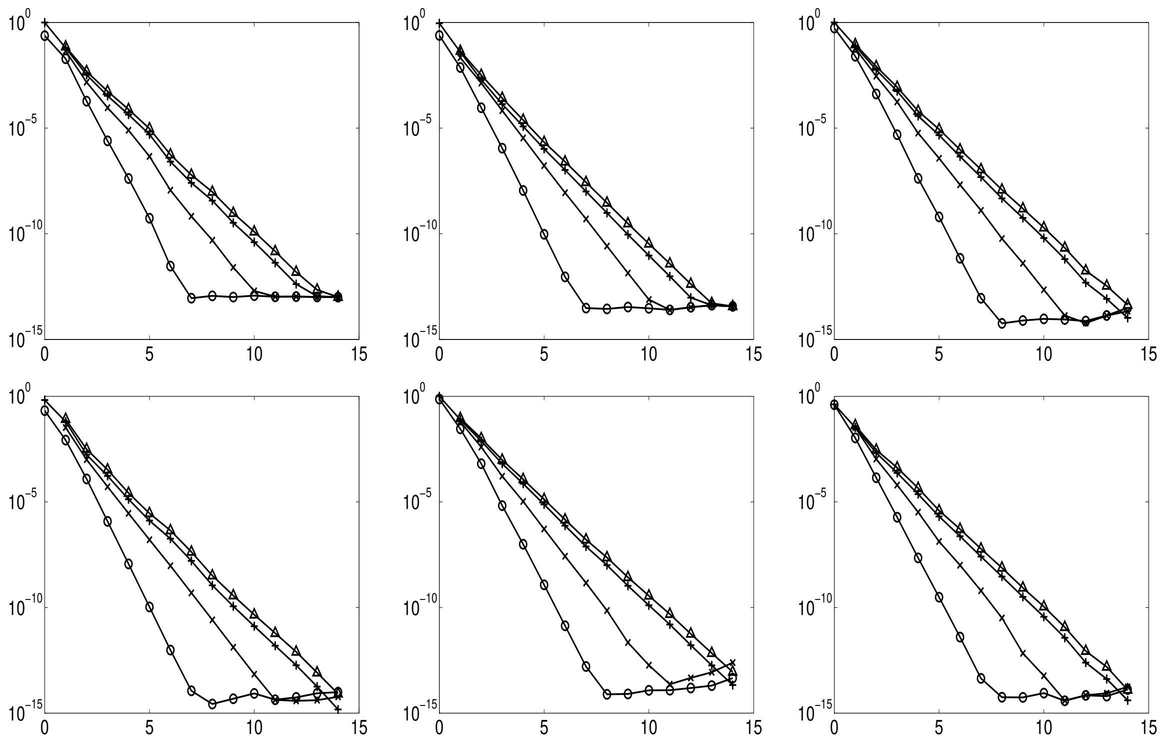


Figure 3: The error E_2 (top) and E_∞ (bottom) with respect to m, from left to right $d = 1, 2, 3$ ($N = 2^{12}, 2^6, 2^4$, $\sigma = 2$, $M = 10000$), for Kaiser Bessel- (circle), Sinc power- (x), B-Spline- (+), and Gaussian window (triangle).

4.2 CPU-time & N

Figure 4 compares computational times for both the NDFT and the NFFT, see `./example/timing`.

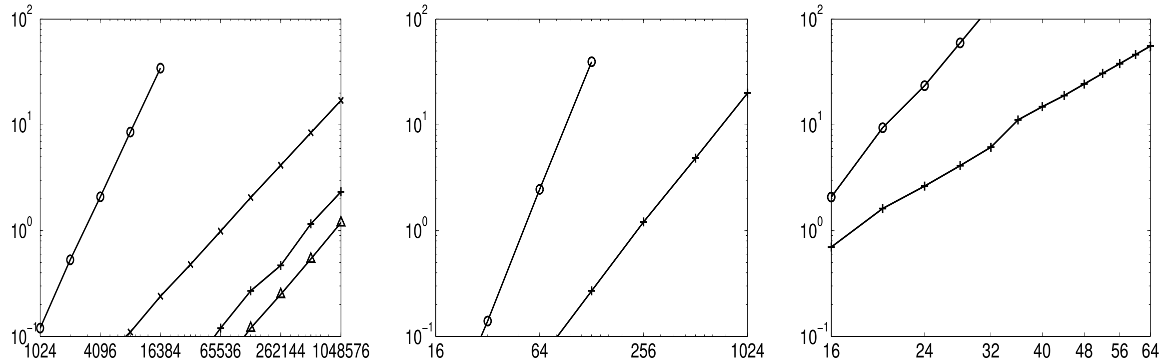


Figure 4: The elapsed CPU-time with respect to N for Kaiser Bessel window, parameters $m = 6$, $\sigma = 2$, $M = N$; left, $d = 1$: NDFT (circle), NFFT (no precomputation, x), NFFT (simple interface, +), NFFT (maximum precomputation, triangle); middle, $d = 2$: NDFT (circle), NFFT (simple interface, +); right, $d = 3$: NDFT (circle), NFFT (simple interface, +).

Figure 5 compares computational times for single steps of the NFFT, see `./example/timing`.

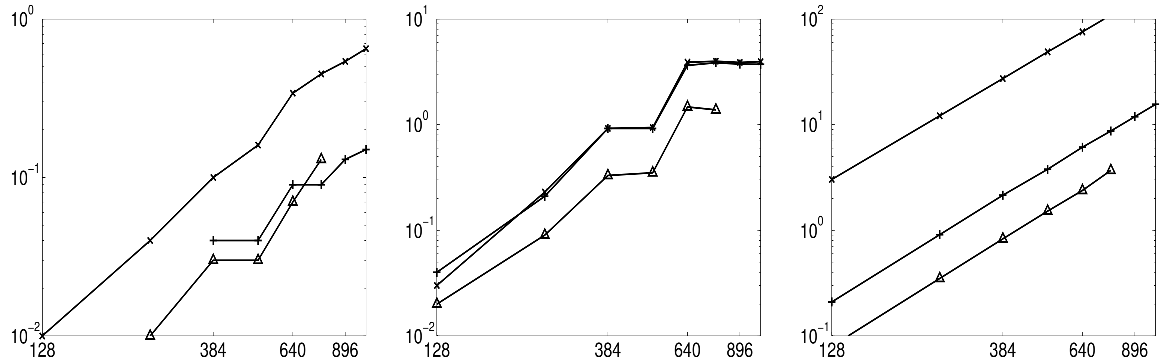


Figure 5: The elapsed CPU-time with respect to N for Kaiser Bessel window, parameters $d = 2$, $m = 6$, $\sigma \geq 2$, $M = N$; from left to right: multiplication with $\boldsymbol{D}$, $\boldsymbol{F}$, and $\boldsymbol{B}$ respectively; no precomputation (x), simple interface (+), and maximum precomputation (triangle).

4.3 Reconstruction of functions

The usage of the inverse NFFT is e.g. shown in `./example/interpolation_1d`, Figure 6, and `./example/glacier`, Figure 7. For theoretical result see [16].

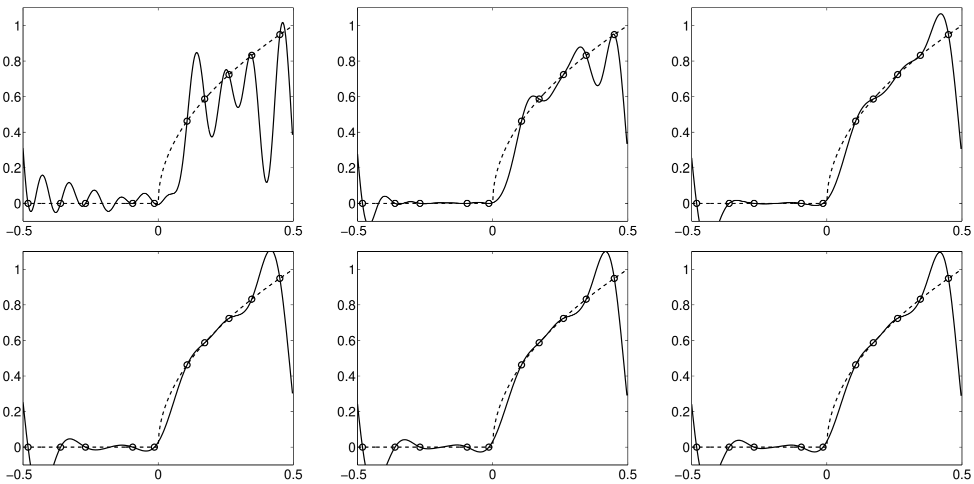


Figure 6: Real part of different interpolation polynomials for $N = 20$ for the function $\sqrt{2}x_+^{1/2}$ sampled at $M = 10$ nodes; 10 iterations and the six kernels: Dirichlet, Fejer, Jackson ($m = 2$), Jackson ($m = 4$), Sobolev ($\alpha = 1$), and inverse multi-quadric-type ($c = 1$, $\mu = 2$).

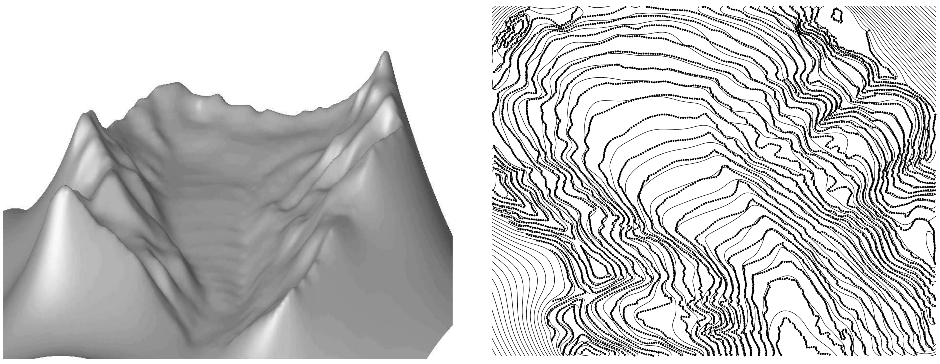


Figure 7: Reconstruction of the glacier data set `vol87.dat` from [11] with radial ($\hat{w}_{\boldsymbol{k}} = \hat{w}_{\|\boldsymbol{k}\|_2}$) inverse multi-quadric-type damping factors ($c = 1$, $\mu = 1.4$); $M = 8345$ nodes, $N = 256$, 40 iterations.

References

[1] R. Bass and K. Gröchenig. Random sampling of multivariate trigonometric polynomials. SIAM *J. Math. Anal.*, 36:773 – 795, 2004.

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

[3] A. Björck. *Numerical Methods for Least Squares Problems.* SIAM, Philadelphia, 1996.

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

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

[6] 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.

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

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

[9] K. Fourmont. Schnelle Fourier–Transformation bei nichtäquidistanten Gittern und tomographische Anwendungen. Dissertation, Universität Münster, 1999.

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

[11] R. Franke. http://www.math.nps.navy.mil/˜rfranke/README.

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

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

[14] M. Hanke. *Conjugate gradient type method for ill–posed problems.* Wiley, New York, 1995.

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

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

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

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

[19] J. Pelt. Fast computation of trigonometric sums with applications to frequency analysis of astronomical data. In D. Maoz, A. Sternberg, and E. Leibowitz, editors, *Astronomical Time Series*, pages 179 – 182, Kluwer, 1997.

[20] 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.

[21] R. A. Scramek and F. R. Schwab. Imaging. In R. Perley, F. R. Schwab, and A. Bridle, editors, *Astronomical Society of the Pacific Conference*, volume 6, pages 117 – 138, 1988.

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

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