Using NFFT 3 – a software library for various nonequispaced fast Fourier transforms

Jens Keiner
University of Lübeck
23560 Lübeck, Germany
keiner@math.uni-luebeck.de
and
Stefan Kunis
Chemnitz University of Technology
09107 Chemnitz, Germany
kunis@mathematik.tu-chemnitz.de
and
Daniel Potts
Chemnitz University of Technology
09107 Chemnitz, Germany
potts@mathematik.tu-chemnitz.de

NFFT 3 is a software library that implements the nonequispaced fast Fourier transform (NFFT) and a number of related algorithms, e.g. nonequispaced fast Fourier transforms on the sphere and iterative schemes for inversion. This is to provide a survey on the mathematical concepts behind the NFFT and its variants, as well as a general guideline for using the library. Numerical examples for a number of applications are given.

Categories and Subject Descriptors: G.1 [**F.2.1**]: Numerical analysis, Computation of transforms

General Terms: Algorithms, Documentation, Theory

Additional Key Words and Phrases: fast Fourier transforms, approximative algorithms

1. INTRODUCTION

NFFT 3 [Keiner et al. 2006b] is a C software library for computing the nonequispaced fast Fourier transform (NFFT); see Appendix D for alternative denominations, further approaches, and references. At the heart of the library is a fast algorithm that generalizes the ubiquitous FFT [Cooley and Tukey 1965] from equally spaced to arbitrary sampling points/spatial nodes, hence the name NFFT; see Appendix B. Our implementation is based on the popular FFTW library [Frigo and Johnson 2005a]. Unlike the FFT, the NFFT is not trivially inverted, which compells one to resort to iterative techniques. Our library also implements these iterative algorithms for inversion, as well as several other discrete transforms related to or based on the NFFT. This paper is to present an overview over NFFT 3, the underlying algorithmic concepts, and intents to serve as a general guideline.

We start in Section 2 with the definition of the nonequispaced discrete Fourier transform (NDFT) as a generalization of the discrete Fourier transform (DFT) to arbitrary nodes. The NFFT algorithm, its key properties, and differences to the classical FFT are discussed. Having settled the relevant definitions, the basic principles of using the NFFT 3 software library are described in Section 3. An overview over the software interface to the NFFT library routines concludes this section. In Section 4, several transforms that derive from the original NFFT are described. This includes, among others, the NFFT on the unit sphere and iterative schemes for inversion of the NFFT. Section 5 is to summarize the theoretical arithmetic costs of our algorithms and presents actual performance measurements for our implementation. In the final Section 6, a number of elementary numerical examples as well as more advanced applications are given.

We have restricted ourselves to describe brief recipes which should be sufficient for most users. For the experienced user, a comprehensive software interface documentation is provided within the downloadable software package [Keiner et al. 2006b, doc/refman.pdf].

2. NONEQUISPACED DISCRETE FOURIER TRANSFORM

Fast Fourier transforms (FFTs) have recently been generalized to arbitrary sampling situations; see [Dutt and Rokhlin 1993; Beylkin 1995; Anderson and Dahleh 1996; Steidl 1998; Ware 1998; Potts et al. 2001] and the references in Appendix D. A convenient approach is to define the discrete Fourier transform (DFT) as the evaluation of a trigonometric polynomial, given by its Fourier coefficients, at equispaced spatial nodes. This can then be generalized to arbitrary nodes.

2.1 Discrete Fourier transform

Following standard conventions [Van Loan 1992; Frigo and Johnson 2005a] the (forward) discrete Fourier transform (DFT) is defined as the calculation of the sums

$$f_j = \sum_{k=0}^{N-1} \hat{f}_k \mathrm{e}^{-2\pi \mathrm{i} jk/N} \qquad (j = 0, \ldots, N-1) \tag{2.1}$$

for $N \in \mathbb{N}$ and given coefficients $\hat{f}_k \in \mathbb{C}$. The transform is readily inverted, i.e.,

$$\hat{f}_k = \frac{1}{N} \sum_{j=0}^{N-1} f_j \mathrm{e}^{2\pi \mathrm{i} kj/N}. \tag{2.2}$$

Clearly, the computation of these transforms requires $\mathcal{O}(N^2)$ arithmetic operations in general. Fast Fourier transforms (FFTs) are $\mathcal{O}(N \log N)$ algorithms to compute the same result.

In matrix-vector notation, the DFT can be written as the matrix-vector product

$$\boldsymbol{f} = \boldsymbol{F} \hat{\boldsymbol{f}}, \quad \boldsymbol{F} = (\mathrm{e}^{-2\pi \mathrm{i} jk/N})_{j,k=0,\ldots,N}$$

with vectors $\boldsymbol{f} = (f_j)_{j=0,\ldots,N-1}$, $\hat{\boldsymbol{f}} = (\hat{f}_k)_{k=0,\ldots,N-1}$, and the Fourier matrix $\boldsymbol{F}$. The inverse transform is equivalent to $\hat{\boldsymbol{f}} = \frac{1}{N}\boldsymbol{F}^{\mathsf{H}} \boldsymbol{f}$. It is straightforward to generalize these concepts to multidimensional transforms; see Section 2.3 below. But before we define the NDFT, some notation is at order.

2.2 Notation

A d-dimensional nonequispaced discrete Fourier transform is defined by a set of arbitrary spatial nodes $\mathcal{X}$ and a frequency bandwidth vector $\boldsymbol{N}$. Each node $\boldsymbol{x}_j$ in the sampling set $\mathcal{X} := \{\boldsymbol{x}_j \in \mathbb{T}^d : j = 0, \ldots, M-1\}$ is drawn from the d-dimensional torus $\mathbb{T}^d \cong [-\frac{1}{2}, \frac{1}{2})^d$ with the number of nodes equal to $|\mathcal{X}| = M$.

For each dimension $t = 0, \ldots, d-1$, the bandwidth $N_t \in 2\mathbb{N}$ is defined as a fixed even number. These bandwidths are collected in the vector $\boldsymbol{N} := (N_0, \ldots, N_{d-1})^\top$. Let us define the multi-index set

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

Then the set $I_{\boldsymbol{N}}$ is a representation of all possible frequencies in a transform. For simplicity, we use the convention that a multi-index $\boldsymbol{k}$ can also reference elements in vectors or rows and columns of matrices as if it were linearized to $k := \sum_{t=0}^{d-1} (k_t + \frac{N_t}{2}) \prod_{t'=t+1}^{d-1} N_{t'}$.

The inner product between a frequency index $\boldsymbol{k}$ and a node $\boldsymbol{x}$ is defined as usual, $\boldsymbol{kx} := k_0 x_0 + \ldots + k_{d-1} x_{d-1}$. Furthermore, we define the component-wise vector product $\boldsymbol{\sigma} \odot \boldsymbol{N} := (\sigma_0 N_0, \ldots, \sigma_{d-1} N_{d-1},)^\top$, and the reciprocal of a vector $\boldsymbol{N}$ with nonzero components, $\boldsymbol{N}^{-1} := \left(N_0{}^{-1}, \ldots, N_{d-1}^{-1}\right)^\top$.

The class of functions $f : \mathbb{T}^d \to \mathbb{C}$ that is naturally associated with an index set $I_{\boldsymbol{N}}$ is the space of d-variate, one-periodic trigonometric polynomials of degree at most N_t along each dimension t, $T_{\boldsymbol{N}} := \operatorname{span}\left(\mathrm{e}^{-2\pi \mathrm{i} \boldsymbol{k} \cdot} : \boldsymbol{k} \in I_{\boldsymbol{N}}\right)$. This space has dimension $\dim T_{\boldsymbol{N}} = |I_{\boldsymbol{N}}| = N_0 \cdot \ldots \cdot N_{d-1}$.

2.3 Definition

We are now ready to define the *nonequispaced discrete Fourier transform (NDFT)* as a generalization of the forward DFT (2.1) to arbitrary nodes. Given Fourier coefficients $\hat{f}_{\boldsymbol{k}} \in \mathbb{C}$, $\boldsymbol{k} \in I_{\boldsymbol{N}}$, as input, the NDFT is defined as the evaluation of the corresponding trigonometric polynomial $f \in T_{\boldsymbol{N}}$ at the set of M arbitrary nodes $\mathcal{X}$, i.e., the calculation of the sums

$$f_j = \sum_{\boldsymbol{k} \in I_{\boldsymbol{N}}} \hat{f}_{\boldsymbol{k}} \mathrm{e}^{-2\pi \mathrm{i} \boldsymbol{k}\boldsymbol{x}_j} \qquad (j = 0, \ldots, M-1). \tag{2.3}$$

To see how this compares with the definition of the DFT, take $\boldsymbol{N} = (N, \ldots, N)^\top$, $N \in 2\mathbb{N}$, and N^d equispaced nodes $\boldsymbol{x_j} = \frac{1}{N}\boldsymbol{j}$, $\boldsymbol{j} \in I_{\boldsymbol{N}}$. Then (2.3) reduces (up to an index shift) to a usual multidimensional forward DFT.

The NDFT (2.3) too can be written as a matrix-vector product,

$$\boldsymbol{f} = \boldsymbol{A}\hat{\boldsymbol{f}},$$

with the vectors $\boldsymbol{f} := (f_j)_{j=0,\ldots,M-1}$, $\hat{\boldsymbol{f}} := (\hat{f}_{\boldsymbol{k}})_{\boldsymbol{k} \in I_{\boldsymbol{N}}}$, and the nonequispaced Fourier matrix $\boldsymbol{A} := \left(\mathrm{e}^{-2\pi \mathrm{i} \boldsymbol{k}\boldsymbol{x}_j}\right)_{j=0,\ldots,M-1;\, \boldsymbol{k} \in I_{\boldsymbol{N}}}$. Typically, the matrix $\boldsymbol{A}$ is not square. Even if this should be the case, it is usually neither orthogonal nor has an inverse. Therefore, the definition of an inverse NDFT transform is not canonical. Instead, it is customary to define the *adjoint NDFT* by the matrix-vector product

$$\hat{\boldsymbol{h}} = \boldsymbol{A}^{\mathsf{H}} \boldsymbol{f}.$$

This is equivalent to the sums

$$\hat{h}_{\boldsymbol{k}} = \sum_{j=0}^{M-1} \hat{f}_j \mathrm{e}^{2\pi \mathrm{i} \boldsymbol{k}\boldsymbol{x}_j} \qquad (\boldsymbol{k} \in I_{\boldsymbol{N}}). \tag{2.4}$$

The NFFT algorithm, implemented in our library, is a fast approximate algorithm to compute the sums in (2.3). It contains also a fast algorithm to compute the adjoint transform (2.4).

2.4 Key features

A d-dimensional DFT of degree N along each dimension has N^d input and output coefficients. This generally requires $\mathcal{O}(N^{d+1})$ to compute the transform. FFTs reduce this to $\mathcal{O}(N^d \log N)$.

The cost for nonequispaced fast transforms is similar. However, it is important to note that there the number of nodes as well as their configuration are arbitrary. This generally implies $M \neq |I_{\boldsymbol{N}}|$. The direct calculation of the sums in (2.3), an algorithm which we also denote NDFT, requires $\mathcal{O}(M|I_{\boldsymbol{N}}|)$ floating point operations. Since the number of nodes M is typically of order $|I_{\boldsymbol{N}}|$, this cost is considered too expensive for most applications. The NFFT 3 library implements a fast approximate algorithm, called NFFT, which computes the same result using only $\mathcal{O}\left(|I_{\boldsymbol{N}}| \log |I_{\boldsymbol{N}}| + |\log \varepsilon|^d M\right)$ operations. Here, ε is the desired accuracy of the computation. The constant contained in this notation depends on the amount of extra memory spent for precomputation; see Appendix C.

Let us briefly compare NDFT/NFFT and classical DFT/FFT to observe some key features and differences: While a multidimensional DFT/FFT is uniquely defined by the bandwidth vector $\boldsymbol{N}$ alone, i.e., the sampling set $\mathcal{X}$ is fixed, the NDFT/NFFT also depend on the actual choice of $\mathcal{X}$; see Subsection 2.3. FFT algorithms are usually exact algorithms with small errors in the result caused only by limitations of floating point arithmetic. The NFFT algorithm purposely introduces a systematic error in the computation to achieve its favorable arithmetic cost. This additional error can be controlled, and, if deemed necessary, can be reduced to the order of machine precision. To this end, the NFFT uses an *oversampled* FFT internally (in our case we use routines from FFTW) together with a pre- and post-processing step. This introduces two additional parameters, an *oversampling factor* and a *truncation parameter*, that control the accuracy of the NFFT. More details are found in Appendix B.

3. GUIDELINES FOR USING THE LIBRARY

NFFT 3 is a free software library completely written in C. The downloadable source code comes as a GNU-style package and follows standard conventions. Brief installation instructions and a source tree can be found in Appendix A. To compile and link programmes that use NFFT 3, you also need the FFTW library [Frigo and Johnson 2005b] in version 3.0.0 or better.

3.1 Interface

The interface of NFFT 3 resembles that of FFTW 3. A transform is specified by creating a *plan* – a data structure that contains the information to determine the characteristics of a transform. In contrast to FFTW 3, it is also necessary to call a precomputation routine, once the nodes $\mathcal{X}$ have been provided. After these initial steps, transforms are executed on input data by invoking an execution routine. At the end of its lifetime, a plan is finalized and all associated data structures are destroyed.

3.2 Typical workflow

This brief example is to illustrate the basic principles. It shows how to compute a one-dimensional NFFT.

Creating a plan. An NFFT plan `p` is declared by

```
nfft_plan p;
```

The simplest way to initialize the plan for a one-dimensional transform is to call

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

The arguments to `nfft_init_1d` are the pointer `&p` to the yet uninitialized plan, the number of Fourier coefficients `N0`, and the number of nodes `M`. This creates and initializes all data structures inside the plan. Memory for Fourier coefficients, nodes, and function values is allocated automatically.

Providing the nodes. The nodes have to be stored in the member array `p.x`. In one dimension, the coordinate of the jth node is a number in $[-\frac{1}{2}, \frac{1}{2})$ that is stored at the index j, i.e.,

```
p.x[j] = your choice in [-0.5,0.5);
```

Precomputation. The precomputation procedure for the plan is invoked (conditionally) by

```
if (p.nfft_flags & PRE_ONE_PSI)
  nfft_precompute_one_psi(&p);
```

NFFT 3 uses different strategies here which are free of choice by the user via passing the appropriate flag to one of the more advanced plan initialization routines. A default strategy is chosen by the simple initialization routines. For convenience, the flag `PRE_ONE_PSI` indicates that precomputation is necessary; see Appendix C for details.

Providing the Fourier coefficients. Fourier coefficients are stored in the one-dimensional member array `p.f_hat`. In the one-dimensional case this array contains the Fourier coefficients in the order $\hat{f}_{-N/2}, \ldots, \hat{f}_{N/2-1}$. They are accessed by

```
p.f_hat[k] = your choice in the complex numbers;
```

Execution. To compute (2.3) with the NFFT algorithm, invoke

```
nfft_trafo(&p);
```

The M output values f_j are stored in the one-dimensional member array `p.f`. To compute a second NFFT with different input data, simply update the Fourier coefficients in `p.f_hat` and invoke `nfft_trafo` a second time.

The adjoint transform (2.4) is computed by `nfft_adjoint`. Here, the input is put in `p.f` and the output is written to `p.f_hat`.

Finalization. Since the initialization routine for the plan has allocated memory, it is essential to finalize the plan, once it is no longer needed. This is done by

```
nfft_finalize(&p);
```

Example. Here is a brief example that computes a one-dimensional NFFT with 14 Fourier coefficients and 19 nodes. The routine is part of the example program `simple_test.c` which is found under `examples/nfft/` in the software package. It uses some auxilliary routines from the NFFT 3 header `util.h` to initialize vectors with random entries and to write them to the standard output.

```
void simple_test_nfft_1d()
{
  nfft_plan p; int N=14; int M=19;
  nfft_init_1d(&p,N,M);
  nfft_vrand_shifted_unit_double(p.x,p.M_total);
  if(p.nfft_flags & PRE_ONE_PSI) nfft_precompute_one_psi(&p);
  nfft_vrand_unit_complex(p.f_hat,p.N_total);
  nfft_vpr_complex(p.f_hat,p.N_total,"given Fourier coefficients, f_hat");
  ndft_trafo(&p);
  nfft_vpr_complex(p.f,p.M_total,"ndft, f");
  nfft_trafo(&p);
  nfft_vpr_complex(p.f,p.M_total,"nfft, f");
  nfft_finalize(&p);
}
```

3.3 Multivariate transforms

In d dimensions, the nodes $\boldsymbol{x}_j$ in the sampling set $\mathcal{X}$, as well as the frequency indices $\boldsymbol{k}$ in the set $I_{\boldsymbol{N}}$ are d-dimensional vectors. This affects the storage layout of the arrays `x` and `f_hat`. Generally, both remain one-dimensional and all d-dimensional quantities are linearized. For `x` this means that the tth coordinate of the jth node ($t = 0, \ldots, d-1$; $j = 0, \ldots, M-1$) is defined via

`p.x[d*j+t] =` *your choice in [-0.5,0.5)*;

This linear ordering has been chosen because the node configuration may be completely unstructured. Analogously, the Fourier coefficient $\hat{f}_{\boldsymbol{k}}$ is stored in `f_hat[k]` at the linearized index `k`$= \sum_{t=0}^{d-1}(k_t + \frac{N_t}{2}) \prod_{t'=t+1}^{d-1} N_{t'}$.

Due to software engineering considerations, we do not offer alternative data layouts, even if the nodes belong to a multidimensional structure, e.g. a tensor product grid. Moreover, NFFT 3 does currently not offer support for strided data or even more general FFTW-style i/o-tensors.

3.4 Programming interface and data structures

The programming interface for the NFFT module is summarized in Table 3.1. To allow for a comparison with the NDFT algorithm, the routine `ndft_trafo` computes the transform via direct evaluation of (2.3). The routines `ndft_adjoint` and `nfft_adjoint` compute adjoint transforms (2.4), respectively. The Initialization routines `nfft_init_1d`, `nfft_init_2d`, `nfft_init_3d` allow for convenient initialization of plans for one-, two-, and three-dimensional transforms with default parameters. Plans for general multivariate transforms are created using `nfft_init`. For the experienced user, the function `nfft_init_guru` allows control over all relevant parameters.

The most important members of the `nfft_plan` data type are listed in Table 3.2. Only `f_hat`, `f`, and `x` are to be modified by the user.

4. GENERALIZATIONS

This section surveys further generalizations of the NFFT algorithm that are implemented by the NFFT 3 library. The interfaces resemble that of the NFFT module described above, with the

Table 3.1. Interface to NDFT/NFFT transforms in NFFT 3.

```
void ndft_trafo(nfft_plan* p)
void ndft_adjoint(nfft_plan* p)
void nfft_trafo(nfft_plan* p)
void nfft_adjoint(nfft_plan* p)
void nfft_init_1d(nfft_plan* p, int N0, int M)
void nfft_init_2d(nfft_plan* p, int N0, int N1, int M)
void nfft_init_3d(nfft_plan* p, int N0, int N1, int N2, int M)
void nfft_init(nfft_plan* p, int d, int* N, int M)
void nfft_init_guru(nfft_plan* p, int d, int* N, int M, int* n,
                    int m, unsigned nfft_flags, unsigned fftw_flags)
void nfft_precompute_one_psi(nfft_plan* p)
void nfft_check(nfft_plan* p)
void nfft_finalize(nfft_plan* p)
```

Table 3.2. Most important members of the structure `nfft_plan`.

Type	Name	Size	Description
`int`	`d`	1	Spatial dimension d
`int*`	`N`	d	Multibandwidth $\boldsymbol{N}$
`int`	`N_total`	1	Number of coefficients $\lvert I_{\boldsymbol{N}}\rvert$
`int`	`M_total`	1	Number of nodes M
`double complex*`	`f_hat`	$\lvert I_{\boldsymbol{N}}\rvert$	Fourier coefficients $\hat{\boldsymbol{f}}$ or adjoint coefficients $\hat{\boldsymbol{h}}$
`double complex*`	`f`	M	Samples $\boldsymbol{f}$
`double*`	`x`	dM	Sampling set $\mathcal{X}$

prefix `nfft_` replaced by the acronym related to the specific transform in all names. Details can be found in the API documentation accompanying the library. All algorithms, with exception of the discrete polynomial transform, are based on the same computational procedure as the NFFT.

4.1 NFST/NFCT - Nonequispaced fast sine/cosine transforms

Discrete (co)sine transforms (DCT/DST) are closely related real variants of the discrete Fourier transform DFT. In the same way, nonequispaced discrete (co)sine transforms (NDCT/NDST) are real transforms related to the NDFT. They are parameterised by arbitrary nodes $\boldsymbol{x}_j \in [0, \frac{1}{2}]^d$. Frequencies $\boldsymbol{k}$ are taken from one of the index sets

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

where $I_{\boldsymbol{N}}^C$ corresponds to the NDCT and $I_{\boldsymbol{N}}^S$ to the NDST, respectively. For given real Fourier coefficients $\hat{f}_{\boldsymbol{k}} \in \mathbb{R}$, NDCT and NDST are defined by

$$f_j = \sum_{\boldsymbol{k}\in I_{\boldsymbol{N}}^C} \hat{f}_{\boldsymbol{k}} \cos(2\pi(\boldsymbol{k} \odot \boldsymbol{x}_j)) \quad (j = 0, \dots, M-1), \qquad f_j = \sum_{\boldsymbol{k}\in I_{\boldsymbol{N}}^S} \hat{f}_{\boldsymbol{k}} \sin(2\pi(\boldsymbol{k} \odot \boldsymbol{x}_j)),$$

respectively. Here, for notational convenience, we have defined $\cos(\boldsymbol{x}) := \cos(x_0) \cdot \ldots \cdot \cos(x_{d-1})$ and $\sin(\boldsymbol{x}) := \sin(x_0) \cdot \ldots \cdot \sin(x_{d-1})$.

Straightforward algorithms for these transforms need $\mathcal{O}(M|I_{\boldsymbol{N}}^C|)$ and $\mathcal{O}(M|I_{\boldsymbol{N}}^S|)$ arithmetic operations. The fast algorithms (NFST/NFCT) need $\mathcal{O}(|I_{\boldsymbol{N}}^S| \log |I_{\boldsymbol{N}}^S| + |\log \varepsilon|^d M)$ and $\mathcal{O}(|I_{\boldsymbol{N}}^C| \log |I_{\boldsymbol{N}}^C| + |\log \varepsilon|^d M)$ operations, respectively. Note that for these real transforms, the adjoint variants coincide with the transposed versions. Details about the implemented algorithms can be found in [Tian and Liu 2000; Potts 2003a; Fenn and Potts 2005].

4.2 NSFFT - Nonequispaced sparse fast Fourier transform

In higher dimensions, the so-called "curse of dimensionality" [Zenger 1991; Sprengel 2000; Bungartz and Griebel 2004] often requires the use of Fourier series where frequency indices are restricted to the hyperbolic cross

$$H_N^d := \bigcup_{\boldsymbol{N}\in\mathbb{N}^d,\, |I_{\boldsymbol{N}}|=N} I_{\boldsymbol{N}}, \qquad N = 2^{J+2},\ J \in \mathbb{N}_0,$$

which has only $\mathcal{O}(N \log^{d-1} N)$ degrees of freedom compared to $\mathcal{O}(N^d)$ in the plain case. If compared to ordinary Fourier series, the approximation error commited, which is typically measured in a norm of dominated mixed smoothness, can be shown to get worse by a factor of only $\log^{d-1} N$ (see [Sprengel 2000]). The *nonequispaced sparse discrete Fourier transform (NSDFT)* is defined as the evaluation of

$$f_j = \sum_{\boldsymbol{k}\in H_N^d} \hat{f}_{\boldsymbol{k}} \mathrm{e}^{-2\pi\mathrm{i}\boldsymbol{k}\boldsymbol{x}_j} \qquad (j = 0, \dots, M-1)$$

with given nodes $\boldsymbol{x}_j \in \mathbb{T}^d$ and Fourier coefficients $\hat{f}_{\boldsymbol{k}} \in \mathbb{C}$. The algorithms implemented in NFFT 3 reduce the number of arithmetic operations from $\mathcal{O}(MN \log^{d-1} N)$ to $\mathcal{O}(N \log^2 N + |\log\varepsilon|^2 M)$ in two dimensions ($d = 2$), and to $\mathcal{O}(N^{3/2} \log N + |\log\varepsilon|^3 M)$ in three dimensions ($d = 3$). Details are found in [Fenn et al. 2006] for details.

4.3 NNFFT - fast Fourier transform for nonequispaced data in space and frequency domain

The NNFFT is a variant of the NFFT where also frequencies are no longer equispaced integer multi-indices. The transform is defined by

$$f_j = \sum_{l=0}^{L-1} \hat{f}_l \mathrm{e}^{-2\pi\mathrm{i}(\boldsymbol{v}_l\odot\boldsymbol{N})\boldsymbol{x}_j} \qquad (j = 0, \dots, M-1)$$

with Fourier coefficients $\hat{f}_l \in \mathbb{C}$, and arbitrary frequencies $\boldsymbol{v}_l \in \mathbb{T}^d$ and nodes $\boldsymbol{x}_j \in \mathbb{T}^d$. Here, $\boldsymbol{N} \in \mathbb{N}^d$ is called the *nonharmonic bandwidth*. This transform is known as *fast Fourier transform for nonequispaced data in space and frequency domain (NNFFT)* [Dutt and Rokhlin 1993; Elbel and Steidl 1998; Potts et al. 2001] or as *type 3 nonuniform FFT* [Lee and Greengard 2005] and takes $\mathcal{O}(|I_{\boldsymbol{N}}| \log |I_{\boldsymbol{N}}| + |\log\varepsilon|^d (L + M))$ arithmetic operations to compute.

4.4 NFSFT - nonequispaced fast spherical Fourier transform

Denote by $\mathbb{S}^2 := \left\{\boldsymbol{x} \in \mathbb{R}^3 :\ \|\boldsymbol{x}\|_2 = 1\right\}$ the two-dimensional unit sphere. In spherical cordinates, any point on $\mathbb{S}^2$ is identified with a pair of angles $(\vartheta, \varphi)^\top \in [0, \pi] \times [0, 2\pi)$. Hence, a sampling set on the sphere can be defined by $\mathcal{X} := \{(\vartheta_j, \varphi_j) : j = 0, \dots, M-1\}$. A convenient basis of orthogonal functions on $\mathbb{S}^2$ are spherical harmonics $Y_k^n : \mathbb{S}^2 \to \mathbb{C}$, $k \in \mathbb{N}_0$, $|n| \le k$, defined by

$$Y_k^n(\vartheta, \varphi) := P_k^{|n|}(\cos\vartheta)\, \mathrm{e}^{\mathrm{i}n\varphi}.$$

Here, $P_k^{|n|}$ are the associated Legendre functions of degree k and order $|n|$ (see [Abramowitz and Stegun 1972, pp. 331]). Frequencies for a finite transform up to bandwidth N are collected in the index set

$$J_N := \{(k, n) \in \mathbb{Z}^2 :\ k = 0, \dots, N;\ n = -k, \dots, k\}.$$

The *nonequispaced discrete spherical Fourier transform (NDSFT)* is defined as the calculation of

$$f_j = \sum_{(k,n)\in J_N} \hat{f}_k^n\, Y_k^n\, (\vartheta_j, \varphi_j) \qquad (j = 0, \dots, M-1).$$

The direct method (NDSFT) clearly needs $\mathcal{O}(MN^2)$ arithmetic operations. The approximate $\mathcal{O}(N^2 \log^2 N + |\log\varepsilon|^2 M)$ algorithm implemented in the NFFT 3 library (*NFSFT*) is a combination of the fast polynomial transform (FPT; see next subsection) and the NFFT. It was introduced in [Kunis and Potts 2003] while the adjoint variant was developed in [Keiner and Potts 2008]. Other algorithms, mostly for particular sampling sets, exist (e.g. [Driscoll and Healy 1994; Suda and Takami 2002; Rokhlin and Tygert 2006]).

4.5 FPT - fast polynomial transform

A *discrete polynomial transform* (DPT) is a generalization of the DFT from complex exponentials $\mathrm{e}^{\mathrm{i}kx}$ to systems of algebraic polynomials that satisfy a three-term recurrence; see [Abramowitz and Stegun 1972, pp. 773]. More precisely, let $p_0, p_1, \ldots : [-1,1] \to \mathbb{R}$ be a sequence of polynomials satisfying the recurrence

$$p_{k+1}(x) = (\alpha_k x + \beta_k) p_k(x) + \gamma_k p_{k-1}(x) \qquad (k \geq 0), \tag{4.1}$$

where $p_{-1}(x) := 0$, $p_0(x) := 1$, and $\alpha_k \neq 0$, $k \geq 0$. Then every p_k is a degree-k polynomial. Typical examples are the Legendre polynomials $P_k(x)$ or the Chebyshev polynomials of first kind $T_k(x) := \cos(k \arccos x)$. Let a finite linear expansion of the form

$$f(x) = \sum_{k=0}^{N} a_k p_k(x)$$

be given. The DPT in our sense is defined as the computation of the coefficients b_k in the expansion

$$f(x) = \sum_{k=0}^{N} b_k T_k(x).$$

Implemented directly, this computation takes $\mathcal{O}(N^2)$ arithmetic operations. The *fast polynomial transform (FPT)* is an $\mathcal{O}(N \log^2 N)$ algorithm to compute the same result. It is implemented in the NFFT 3 library following the approach in [Potts et al. 1998]. This is based on using the three-term-recurrence relation (4.1) repeatedly, together with a fast method for polynomial multiplication in the Chebyshev basis, and a cascade-like summation process. More details can be found in [Driscoll and Healy 1994; Driscoll et al. 1996; Potts et al. 1998; Potts 2003a] and references therein.

4.6 Inversion and solver module

The NDFT differs from the DFT also in that an inversion formula similar to (2.2) generally does not exist; see Section 2.3. A more general view is at order to define what should be an inverse NDFT/NFFT transform. This section is devoted to explain our approach for the NFFT. But everything also applies to the mentioned NFFT variants.

Inversion of the NDFT means the computation of Fourier coefficients $\hat{f}_{\boldsymbol{k}}$, $\boldsymbol{k} \in I_{\boldsymbol{N}}$, from given function samples y_j at nodes $\boldsymbol{x}_j$, $j = 0, \ldots, M-1$. In matrix-vector notation, this is equivalent to solving the linear system

$$\boldsymbol{A}\hat{\boldsymbol{f}} \approx \boldsymbol{y}. \tag{4.2}$$

This can be either and overdetermined system if $|I_{\boldsymbol{N}}| \leq M$ (this includes the quadratic case) or an underdetermined system if $|I_{\boldsymbol{N}}| > M$. Generally, this compels us to look for a pseudo-inverse solution $\hat{\boldsymbol{f}}^{\dagger}$ (see e.g. [Björck 1996, p. 15]). For this, we also require that the nonequispaced Fourier matrix $\boldsymbol{A}$ have full rank. Eigenvalue estimates in [Feichtinger et al. 1995; Bass and Gröchenig 2004; Kunis and Potts 2007] indeed assert that this condition is satisfied if the sampling set $\mathcal{X}$ is uniformly dense or uniformly separated with respect to the inverse bandwidth $\boldsymbol{N}^{-1}$.

For the overdetermined case, we now consider a weighted least squares problem, while for the consistent underdetermined case, we look for a solution of an interpolation problem. Both can be solved by iterative algorithms using NFFT and adjoint NFFT to realize fast matrix-vector multiplications involving $\boldsymbol{A}$ and $\boldsymbol{A}^{\mathsf{H}}$, respectively.

Weighted least squares problem. If $|I_{\boldsymbol{N}}| \leq M$, the linear system (4.2) is overdetermined which typically implies that the given data $y_j \in \mathbb{C}$, $j = 0, \ldots, M-1$, can only be approximated up to a residual $\boldsymbol{r} := \boldsymbol{y} - \boldsymbol{A}\hat{\boldsymbol{f}}$. Therefore, we consider the *weighted least squares problem*

$$\sum_{j=0}^{M-1} w_j |y_j - f(x_j)|^2 \xrightarrow{\hat{\boldsymbol{f}}} \min$$

Input: $\boldsymbol{y} \in \mathbb{C}^M$, $\hat{\boldsymbol{f}}_0 \in \mathbb{C}^{|I_{\boldsymbol{N}}|}$.

CGNR (left):

$\boldsymbol{r}_0 = \boldsymbol{y} - \boldsymbol{A}\hat{\boldsymbol{f}}_0$
$\hat{\boldsymbol{z}}_0 = \boldsymbol{A}^{\mathsf{H}}\boldsymbol{W}\boldsymbol{r}_0$
$\hat{\boldsymbol{p}}_0 = \hat{\boldsymbol{z}}_0$
for $l = 0, \dots$ **do**
  $\boldsymbol{v}_l = \boldsymbol{A}\hat{\boldsymbol{W}}\hat{\boldsymbol{p}}_l$
  $\alpha_l = \frac{\hat{\boldsymbol{z}}_l^{\mathsf{H}}\hat{\boldsymbol{W}}\hat{\boldsymbol{z}}_l}{\boldsymbol{v}_l^{\mathsf{H}}\boldsymbol{W}\boldsymbol{v}_l}$
  $\hat{\boldsymbol{f}}_{l+1} = \hat{\boldsymbol{f}}_l + \alpha_l\hat{\boldsymbol{W}}\hat{\boldsymbol{p}}_l$
  $\boldsymbol{r}_{l+1} = \boldsymbol{r}_l - \alpha_l\boldsymbol{v}_l$
  $\hat{\boldsymbol{z}}_{l+1} = \boldsymbol{A}^{\mathsf{H}}\boldsymbol{W}\boldsymbol{r}_{l+1}$
  $\beta_l = \frac{\hat{\boldsymbol{z}}_{l+1}^{\mathsf{H}}\hat{\boldsymbol{W}}\hat{\boldsymbol{z}}_{l+1}}{\hat{\boldsymbol{z}}_l^{\mathsf{H}}\hat{\boldsymbol{W}}\hat{\boldsymbol{z}}_l}$
  $\hat{\boldsymbol{p}}_{l+1} = \beta_l\hat{\boldsymbol{p}}_l + \hat{\boldsymbol{z}}_{l+1}$
end for

CGNE (right):

$\boldsymbol{r}_0 = \boldsymbol{y} - \boldsymbol{A}\hat{\boldsymbol{f}}_0$
$\hat{\boldsymbol{p}}_0 = \boldsymbol{A}^{\mathsf{H}}\boldsymbol{W}\boldsymbol{r}_0$
for $l = 0, \dots$ **do**
  $\alpha_l = \frac{\boldsymbol{r}_l^{\mathsf{H}}\boldsymbol{W}\boldsymbol{r}_l}{\hat{\boldsymbol{p}}_l^{\mathsf{H}}\hat{\boldsymbol{W}}\hat{\boldsymbol{p}}_l}$
  $\hat{\boldsymbol{f}}_{l+1} = \hat{\boldsymbol{f}}_l + \alpha_l\hat{\boldsymbol{W}}\hat{\boldsymbol{p}}_l$
  $\boldsymbol{r}_{l+1} = \boldsymbol{r}_l - \alpha_l\boldsymbol{A}\hat{\boldsymbol{W}}\hat{\boldsymbol{p}}_l$
  $\beta_l = \frac{\boldsymbol{r}_{l+1}^{\mathsf{H}}\boldsymbol{W}\boldsymbol{r}_{l+1}}{\boldsymbol{r}_l^{\mathsf{H}}\boldsymbol{W}\boldsymbol{r}_l}$
  $\hat{\boldsymbol{p}}_{l+1} = \beta_l\hat{\boldsymbol{p}}_l + \boldsymbol{A}^{\mathsf{H}}\boldsymbol{W}\boldsymbol{r}_{l+1}$
end for

Output: The lth approximation to the solution vector $\hat{\boldsymbol{f}}_l \in \mathbb{C}^{|I_{\boldsymbol{N}}|}$.

Alg. 1. Variants of the conjugate gradient method for normal equations of first (CGNR = residual minimization; left) and second kind (CGNE = error minimization; right).

with weights $w_j > 0$, $\boldsymbol{W} := \operatorname{diag}(w_j)_{j=0,\dots,M-1}$. This weights might be used to compensate for clusters in the sampling set $\mathcal{X}$. The weighted least squares problem is equivalent to solving the *weighted normal equation of first kind*, $\boldsymbol{A}^{\mathsf{H}}\boldsymbol{W}\boldsymbol{A}\hat{\boldsymbol{f}} = \boldsymbol{A}^{\mathsf{H}}\boldsymbol{W}\boldsymbol{y}$. This can be done, e.g., by using the Landweber (or Richardson) iteration, the steepest descent method, or the conjugate gradient method for least squares problems. The latter method is given in pseudo-code in Algorithm 1 (left). The NFFT 3 library nevertheless implements all three algorithms.

Interpolation problem. If $|I_{\boldsymbol{N}}| > M$, and if the linear system (4.2) is consistent, the data $y_j \in \mathbb{C}$, $j = 0, \dots, M-1$, can be interpolated exactly. But since there exist multiple solutions, we consider the *constrained minimization problem*

$$\sum_{\boldsymbol{k}\in I_{\boldsymbol{N}}} \frac{|\hat{f}_{\boldsymbol{k}}|^2}{\hat{w}_{\boldsymbol{k}}} \xrightarrow{\hat{\boldsymbol{f}}} \min \quad \text{subject to} \quad \boldsymbol{A}\hat{\boldsymbol{f}} = \boldsymbol{y},$$

which incorporates "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, i.e., a solution with rapid decay of Fourier coefficients $\hat{f}_{\boldsymbol{k}}$, is favored if the damping factors $\hat{w}_{\boldsymbol{k}}$ are decreasing themselves. The interpolation problem is equivalent to the *damped normal equation of second kind* $\boldsymbol{A}\hat{\boldsymbol{W}}\boldsymbol{A}^{\mathsf{H}}\tilde{\boldsymbol{f}} = \boldsymbol{y}$, $\hat{\boldsymbol{f}} = \hat{\boldsymbol{W}}\boldsymbol{A}^{\mathsf{H}}\tilde{\boldsymbol{f}}$. The conjugate gradient method applied to this linear system is given in Algorithm 1 (right). Of course, the NFFT 3 library also implements this scheme.

5. SUMMARY AND PERFORMANCE ANALYSIS

This section is to provide a performance analysis of the NFFT algorithm. The asymptotic bounds on the number of floating point operations for the nonequispaced fast Fourier transform (NFFT) and its variant on the sphere (NFSFT) are compared to those for the discrete/fast Fourier transform in Table 5.1. The rest of the section is to provide results of actual performance measurements for our implementation of the NFFT in spatial dimensions $d = 1, 2, 3$. Computation time measurements and a numerical confirmation of the error estimates for the window functions (C.1-C.4) with respect to the so-called cut-off parameter are given.

For our tests, we used a computer system with an AMD Athlon™ XP 2700+ processor, 1GB memory, SuSE Linux with kernel 2.4.20-4GB-athlon, and the GCC 3.3 compiler. All computations were carried out in double precision using the compiler option `-O3`. For all tests with random input, the nodes $\boldsymbol{x}_j$ and the Fourier coefficients $\hat{f}_{\boldsymbol{k}}$ were drawn from a uniform pseudo-random

Table 5.1. Asymptotic number of floating point operations with respect to the polynomial degree N (so that $\boldsymbol{N} = (N, \ldots, N)^\top$), the number of nodes M, spatial dimension d, and target accuracy ε for different transforms.

	$M = \mathcal{O}(N^d)$, $\mathcal{X}$ grid		M, $\mathcal{X}$ arbitrary	
	DFT	FFT	NDFT	NFFT
Torus $\mathbb{T}^d$	$\mathcal{O}(N^{d+1})$	$\mathcal{O}(N^d \log N)$	$\mathcal{O}(MN^d)$	$\mathcal{O}(N^d \log N + \lvert\log \varepsilon\rvert^d M)$
Sphere $\mathbb{S}^2$	$\mathcal{O}(N^3)$	$\mathcal{O}(N^2 \log^2 N)$	$\mathcal{O}(MN^2)$	$\mathcal{O}(N^2 \log^2 N + \lvert\log \varepsilon\rvert^2 M)$

distribution in the respective domains, i.e., $\boldsymbol{x}_j \in \mathbb{T}^d$ and $\hat{f}_{\boldsymbol{k}} \in [0,1] \times [0,1]\mathrm{i}$. The code for these and more examples and applications can be found in the directories `examples/` and `applications/` of the NFFT 3 package.

5.1 Computation time with respect to problem size

We compared computation times for ordinary FFTs (using FFTW 3.0.0 [Frigo and Johnson 2005a; 2005b] in double precision with the flag `FFTW_MEASURE`), direct nonequispaced discrete Fourier transforms (NDFTs), and nonequispaced fast Fourier transforms (NFFTs) for different problem sizes $|I_{\boldsymbol{N}}|$ in dimensions $d = 1, 2, 3$. The multibandwidth $\boldsymbol{N}$ was chosen to $\boldsymbol{N} = (N, \ldots, N)^\top$, $N \in 2\mathbb{N}$. The nodes for FFTs are fixed and restricted to the lattice $\boldsymbol{N}^{-1} \odot I_{\boldsymbol{N}}$. For NDFTs and NFFTs, we chose $M = N^d$ random nodes and used an oversampling factor $\sigma = 2$, cut-off parameters $m = 2$ and $m = 4$, and the Kaiser-Bessel window function (precomputation flags `PRE_PSI` and `PRE_PHI_HUT`). With these options, the accuracy measured by

$$E_\infty := \max_{0 \le j < M} |f_j - s_j| / \sum_{\boldsymbol{k} \in I_{\boldsymbol{N}}} |\hat{f}_{\boldsymbol{k}}| \tag{5.1}$$

is approximately $E_\infty \approx 10^{-4}$ ($m = 2$) and $E_\infty \approx 10^{-8}$ ($m = 4$) for $d = 1, 2, 3$. Here, s_j denotes the NFFT-approximation to f_j; see the Appendix for details. The corresponding example program can be found under `examples/nfft/nfft_times`. As a byproduct of the measurements shown in Table 5.2, we also obtain a performance index measured in million floating point operations per second (MFLOPS). We assume that the number of floating point operations are

$$5|I_{\boldsymbol{N}}| \log_2 |I_{\boldsymbol{N}}|, \quad \text{and} \quad |I_{\boldsymbol{N}}| + 5 \cdot 2^d |I_{\boldsymbol{N}}| \log(2^d |I_{\boldsymbol{N}}|) + 4(2m+1)^d |I_{\boldsymbol{N}}|$$

for the FFT and the NFFT ($M = |I_{\boldsymbol{N}}|$ nodes, oversampling factor $\sigma = 2$, and precomputation flags `PRE_PSI` and `PRE_PHI_HUT`), respectively. More details are given in the Appendix.

Some conclusions can be drawn from Table 5.2: Both, FFT and NFFT, require a computation time of order $\mathcal{O}(|I_{\boldsymbol{N}}| \log |I_{\boldsymbol{N}}|)$ as expected. Doubling $|I_{\boldsymbol{N}}|$ indeed results in only slightly more than twice the computation time. The NDFT also scales as the expected $\mathcal{O}(|I_{\boldsymbol{N}}|^2)$. For FFT and NDFT, the constant in the $\mathcal{O}$-notation is independent of the space dimension d, whereas the computation time of the NFFT increases considerably with d fixing $|I_{\boldsymbol{N}}|$. Moreover, Figure 5.1 shows a relative performance gain of the FFT for small and moderate sized problems. In contrast, the overhead contained in the approximation scheme of the NFFT algorithm seems to attenuate this effect for the NFFT.

5.2 Accuracy

The actual choice of the window function and the cut-off parameter m (cf. Appendix) have direct influence on the accuracy achieved by the NFFT algorithm. To confirm this, we compare values f_j in (2.3) computed by the NDFT with approximations s_j computed using an NFFT with different choices of the window function and the cut-off parameter m. The error is measure is again (5.1). Figure 5.2 shows results for 1, 2, and 3-dimensional transforms with equal total numbers of Fourier coefficients $|I_{\boldsymbol{N}}|$.

6. APPLICATIONS

The rest of this tutorial is devoted to illustrating the usage of the NFFT 3 library for more advanced applications. We start with two small problems from scattered data approximation

Table 5.2. Computation times in seconds with respect to the problem size $l_N = \log_2 |I_{\boldsymbol{N}}|$. Numbers in parentheses give the penalty factor, i.e., the quotient of computation times for NFFT and FFT. Times for small l_N have been averaged over a series of transforms. An asterisk (∗) indicates values not calculated due to excessive computation time.

				$m=4,\ E_\infty \approx 10^{-8}$		$m=2,\ E_\infty \approx 10^{-4}$	
	l_N	FFT	NDFT	NFFT	penalty factor	NFFT	penalty factor
$d=1$	3	1.3e-07	8.6e-06	1.3e-06	(10.0)	1.1e-06	(8.6)
	4	2.1e-07	3.5e-05	2.7e-06	(12.9)	2.2e-06	(10.5)
	5	4.2e-07	1.4e-04	5.2e-06	(12.3)	4.2e-06	(11.4)
	6	9.2e-07	5.7e-04	1.1e-05	(11.5)	9.1e-06	(9.8)
	7	2.2e-06	2.3e-03	2.1e-05	(9.5)	1.9e-05	(9.0)
	8	5.3e-06	9.2e-03	4.4e-05	(8.3)	3.9e-05	(7.7)
	9	1.1e-05	3.7e-02	9.8e-05	(8.8)	8.3e-05	(7.4)
	10	2.5e-05	1.5e-01	2.0e-04	(8.1)	1.8e-04	(7.1)
	11	6.0e-05	6.0e-01	6.5e-04	(10.8)	5.4e-04	(8.9)
	12	1.5e-04	2.4e+00	1.5e-03	(9.8)	1.4e-03	(8.9)
	13	5.5e-04	9.6e+00	3.5e-03	(6.4)	3.3e-03	(5.6)
	14	1.7e-03	4.0e+01	7.8e-03	(4.6)	7.1e-03	(4.3)
	15	4.0e-03	1.6e+02	1.6e-02	(4.1)	1.5e-02	(3.7)
	16	8.4e-03	∗	3.4e-02	(4.1)	3.2e-02	(3.8)
	17	2.0e-02	∗	7.5e-02	(3.8)	7.2e-02	(3.6)
	18	4.6e-02	∗	1.6e-01	(3.4)	1.5e-01	(3.2)
	19	9.5e-02	∗	3.2e-01	(3.4)	3.0e-01	(3.1)
	20	2.1e-01	∗	6.9e-01	(3.3)	6.4e-01	(3.3)
	21	4.3e-01	∗	1.5e+00	(3.5)	1.4e+00	(3.2)
	22	1.0e+00	∗	3.2e+00	(3.1)	3.0e+00	(2.9)
$d=2$	6	9.0e-07	6.0e-04	6.3e-05	(70.4)	3.2e-05	(33.2)
	8	4.4e-06	9.5e-03	2.5e-04	(58.0)	1.3e-04	(29.0)
	10	2.2e-05	1.5e-01	1.2e-03	(55.0)	6.0e-04	(27.4)
	12	1.2e-04	2.4e+00	6.4e-03	(52.5)	4.0e-03	(32.2)
	14	1.7e-03	4.0e+01	4.0e-02	(23.4)	3.1e-02	(18.1)
	16	2.2e-02	∗	1.7e-01	(7.8)	1.3e-01	(6.1)
	18	8.7e-02	∗	6.7e-01	(7.7)	5.0e-01	(5.8)
	20	3.3e-01	∗	3.0e+00	(9.1)	2.1e+00	(6.4)
	22	1.4e+00	∗	1.4e+01	(10.1)	1.1e+01	(7.9)
$d=3$	9	1.0e-05	3.8e-02	4.4e-03	(423.4)	1.1e-03	(109.5)
	12	1.1e-04	2.4e+00	4.2e-02	(369.1)	1.4e-02	(115.8)
	15	3.5e-03	1.6e+02	3.9e-01	(110.8)	1.5e-01	(41.8)
	18	3.9e-02	∗	3.9e+00	(99.3)	1.7e+00	(42.4)
	21	9.4e-01	∗	8.4e+01	(89.0)	1.8e+01	(19.6)

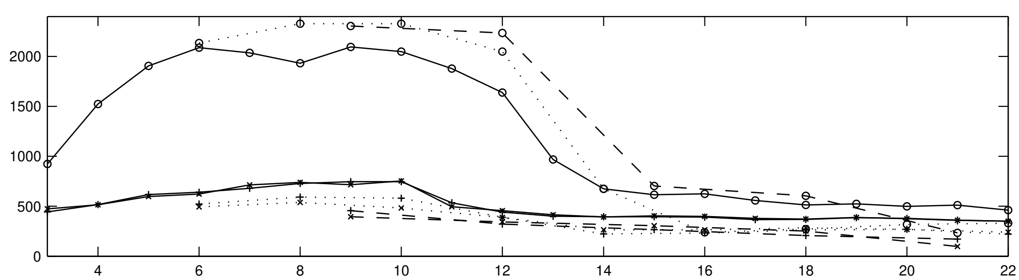
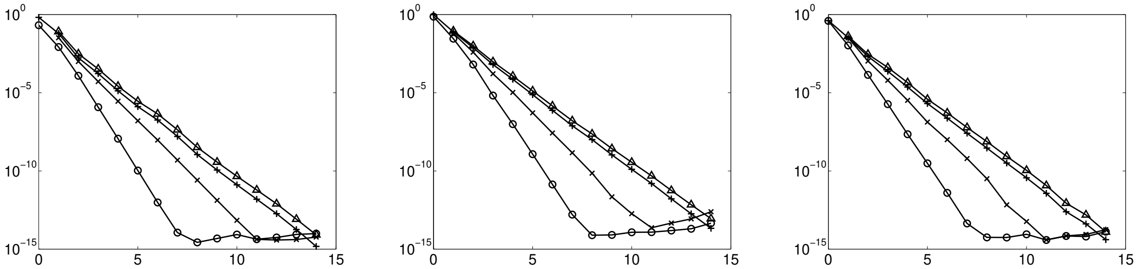
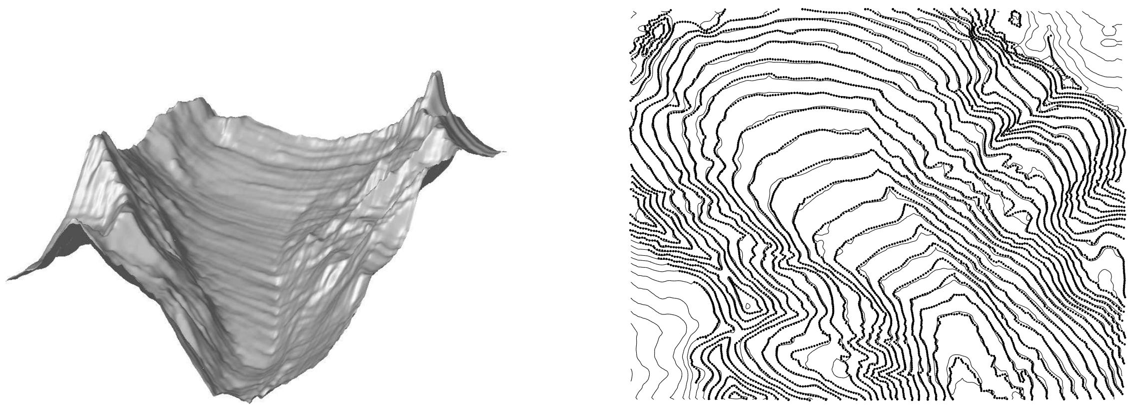


Fig. 5.1. Performance measured in MFLOPS achieved by the NFFT 3 library with respect to the logarithm of the total problem size $l_N = \log_2 |I_{\boldsymbol{N}}|$. Shown are results for $d = 1$ (solid), $d = 2$ (dotted), and $d = 3$ (dashed). The tested algorithms are the FFT (∘), the NFFT with $m = 4$ (×), and the NFFT with $m = 2$ (+).



(a) $d = 1$, $N = 2^{12}$. (b) $d = 2$, $N = 2^6$. (c) $d = 3$, $N = 2^4$.

Fig. 5.2. Error E_∞ for increasing cut-off parameter $m = 0, \ldots, 14$ and $d = 1, 2, 3$. In each case, the degree N was chosen to be equal along each dimension such that $|I_{\boldsymbol{N}}| = 2^{12}$. We fixed $\sigma = 2$ (cf. Appendix) and $M = 10000$. Shown are results for the Kaiser Bessel ($\circ$), the Sinc ($\times$), the B-spline ($+$), and the Gaussian window function ($\triangle$).



(a) Surface plot. (b) Contour plot.

Fig. 6.1. Reconstruction of the glacier from data on level sets at $M = 8345$with $N_0 = N_1 = 256$ after 40 iterations.

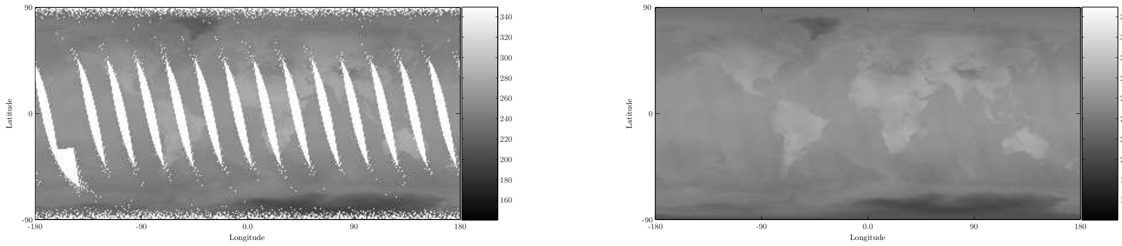
which use the solver module and the NFFT and NFSFT routines. Then we briefly sketch some more important applications - fast summation schemes and medical imaging. These and more examples and applications can be found in the directories `examples/` and `applications/` of the package.

6.1 Scattered data approximation - computing an inverse transform

Our first example demonstrates the reconstruction of the surface of a glacier from data available on level sets. This is done by solving a damped optimal interpolation problem. Figure 6.1 shows the result using the $M = 8345$ available data points and bandwidths $N_0 = N_1 = 256$ for the two dimensions after 40 iterations of the CGNE method. The corresponding example program `glacier` can be found in `examples/solver/`. There is also a simple test program called `simple_test` to illustrate basic principles.

The second example uses a map from the NASA AMSU mission [National Aeronautics and Space Administration 2007] containing global atmospheric temperature data of the earth from 5 November 2006. The map contains empty strips not covered by the trajectory of the satellite used to take the measurements. To fill these gaps, we solve a weighted linear least squares problem to obtain a spherical harmonics approximation of degree $N = 128$ to the given data (see [Keiner et al. 2007]). We then use the values of the approximation to fill the gaps. Figure 6.2 shows the original data and the map with the gaps filled.

The following subsections are devoted to the demonstration of further important applications



(a) Given data.

(b) Approximation.

Fig. 6.2. Original atmospheric temperature data in Kelvin (a) and the same data with values filled in from the least squares approximation at degree $N = 128$.

of NFFT 3 routines. The source code is available in the directory `applications/` in the package.

6.2 Fast summation

An important task in many applications is to efficiently evaluate a linear combinations of radial functions, i.e.,

$$g\left(\boldsymbol{y}_j\right) := \sum_{l=0}^{L-1} \alpha_l K\left(\left\|\boldsymbol{y}_j - \boldsymbol{x}_l\right\|_2\right) \qquad (j = 0, \ldots, M-1)$$

with source and target nodes $\boldsymbol{x}_l, \boldsymbol{y}_j \in \mathbb{R}^d$, and real coefficients $\alpha_l \in \mathbb{R}$.

For kernels K with a certain smoothness, e.g. the Gaussian $K(x) = \mathrm{e}^{-x^2/c^2}$, the multiquadric $K(x) = \sqrt{x^2 + c^2}$, or the inverse multiquadric $K(x) = 1/\sqrt{x^2 + c^2}$, all with a parameter $c > 0$, our algorithm requires $\mathcal{O}(L + M)$ arithmetic operations instead of $\mathcal{O}(LM)$ via direct calculation. For singular kernels K, e.g.

$$\frac{1}{x^2},\ \frac{1}{|x|},\ \log|x|,\ x^2 \log|x|,\ \frac{1}{x},\ \frac{\sin(cx)}{x},\ \frac{\cos(cx)}{x},\ \cot(cx)$$

it is imperative to apply an additional regularization procedure. The cost then becomes $\mathcal{O}(L \log L + M)$ or $\mathcal{O}(M \log M + L)$ provided that either the source nodes $\boldsymbol{x}_l$ or the target nodes $\boldsymbol{y}_j$ are uniformly distributed to a reasonable degree. This fast method, which was proposed in [Potts and Steidl 2003; Potts et al. 2004; Fenn and Steidl 2004], generalizes the diagonalization of convolution matrices by Fourier matrices to settings with arbitrary nodes. Also, our method yields nearly the same cost as the FMM [Beatson and Greengard 1997] while allowing for an easy exchange of the kernel K. A recent application in particle simulation is given in [Pöplau et al. 2006]. The directory `applications/fastsum` contains C and MatLab programs showing how to use the fast summation method.

As a special case, let us consider the fast Gauss transform. This is the calculation of sums of the form

$$g\left(y_j\right) := \sum_{l=0}^{L-1} \alpha_l \mathrm{e}^{-\sigma|y_j - x_l|^2} \qquad (j = 0, \ldots, M-1)$$

for source and target nodes $x_l, y_j \in [-\frac{1}{4}, \frac{1}{4}]$, complex coefficients $\alpha_l \in \mathbb{C}$, and a complex parameter $\sigma = a + \mathrm{i}b$, $a > 0$, $b \in \mathbb{R}$ with $\mathcal{O}(L + M)$ operations. For details see [Andersson and Beylkin 2005; Kunis et al. 2006]. Numerical examples found in [Kunis et al. 2006] can be reproduced with the programs found in the directory `applications/fastgauss`.

On the sphere $\mathbb{S}^2$, the equivalent of radial functions are *zonal functions*. Fast summation of zonal functions on the sphere is the computation of sums of the form

$$g\left(\boldsymbol{\xi}_j\right) := \sum_{l=0}^{L-1} \alpha_l K\left(\boldsymbol{\eta}_l \cdot \boldsymbol{\xi}_j\right) \qquad (j = 0, \ldots, M-1)$$

for source and target nodes $\boldsymbol{\eta}_l, \boldsymbol{\xi}_j \in \mathbb{S}^2$, and real coefficients $\alpha_l \in \mathbb{R}$ with $\mathcal{O}(L+M)$ operations. Our algorithm is based on the nonequispaced fast spherical Fourier transform. It and can be easily adapted to different kernels K, e.g. the Poisson kernel, smooth locally supported kernels, or the spherical Gaussian kernel. For details see [Keiner et al. 2006a]. Numerical examples are found under `applications/fastsumS2/`.

6.3 Applications in medical imaging

In magnetic resonance imaging (MRI), raw data is measured in *k-space*, the domain of spatial frequencies. Methods that use non-Cartesian sampling grids in k-space, e.g. a spiral, have received increasing attention recently. Reconstruction is usually done by resampling the data onto a Cartesian grid to use the standard FFT. This is often called *gridding*. A so-called inverse model is based on an implicit discretization and both methods can be implemented efficiently using the NFFT and the inverse NFFT, respectively (see [Knopp et al. 2007] and references therein). Furthermore, approaches to field inhomogeneity correction have been proposed in [Sutton et al. 2003; Eggers et al. 2007]. Numerical tests are found in `applications/mri/`.

In computerized tomography (CT) an $N \times N$ (medical) image is to be reconstructed from its Radon transform. The standard reconstruction algorithm, the filtered backprojection, yields reasonably good images at the expense of $\mathcal{O}(N^3)$ arithmetic operations. Fourier reconstruction methods reduce the number of arithmetic operations to $\mathcal{O}(N^2 \log N)$. Unfortunately, straightforward Fourier reconstruction algorithms suffer from unacceptable artifacts making them useless in practice. Better quality of the reconstructed images can be achieved by our algorithm which is based on NFFTs. For details see [Potts and Steidl 2001; 2000; 2002] and references therein, for examples we refer to `applications/radon/`.

Another application of the discrete Radon transform is the *discrete ridgelet transform*, see e.g. [Candes et al. 2006] and the references therein. A simple test program for denoising images by applying hard thresholding to ridgelet coefficients [Ma and Fenn 2006] can be found in `applications/radon`. It uses the NFFT-based discrete Radon transform and the translation-invariant discrete Wavelet transform from the MatLab toolbox WaveLab850 [Donoho et al. 2006].

Our last example to be mentioned here is the *polar FFT*, which is a special case of the NFFT for a particular set of nodes. Of course, the polar as well as a so-called pseudo-polar FFT can be calculated efficiently accurately with the NFFT. Furthermore, the reconstruction of a $2d$ signal from its Fourier transform samples on a (pseudo-)polar grid by means of the inverse nonequispaced FFT is possible under certain density assumptions. For details see [Averbuch et al. 2006; Fenn et al. 2007; Beylkin et al. 2007]. Numerical tests can be found in the directory `applications/polarFFT/`.

APPENDIX

A. INSTALLATION

The installation of the NFFT 3 software library follows the usual GNU principles. Download the most recent version from `http://www.tu-chemnitz.de/~potts/nfft`. We assume that you use a `bash` compatible shell. Uncompress the archive and change to the newly created directory:

```
tar xfvz nfft3.x.x.tar.gz
cd nfft3.x.x
```

Figure A.1 gives an overview over the directory structure of the NFFT 3 package. The package is now ready to be configured. To this end, run the configure script `configure` which will adapt the build process to your computer system. After that, compile the library by invoking the `make` utility:

```
./configure
make
```

When the build process has finished, you can run `make install` to install the library permanently on your system.

For more information about options offered by the `configure` script, run the script with the option `--help`. For example, to change the installation location you can use the `--prefix`. Here

- doc (API docs)
- include (C headers)
- util (utility functions)
- kernel
  - fpt (fast polynomial transform)
  - mri (transform in magnetic resonance imaging)
  - nfct (nonequispaced fast cosine transform)
  - nfft (nonequispaced fast Fourier transform)
  - nfsft (nonequispaced fast spherical Fourier transform)
  - nfst (nonequispaced fast sine transform)
  - nnfft (nonequispaced in space and frequency FFT)
  - nsfft (nonequispaced sparse fast Fourier transform)
  - solver (inverse transforms)
- examples (for each in kernel)
- applications
  - fastgauss (fast Gauss transform)
  - fastsum (summation schemes)
  - fastsumS2 (summation on the sphere)
  - mri (reconstruction in mri)
  - nfft_flags (time and memory requirements)
  - polarFFT (fast polar Fourier transform)
  - radon (radon transform)
  - stability (stability inverse nfft)

Fig. A.1. Directory structure of the NFFT 3 package.

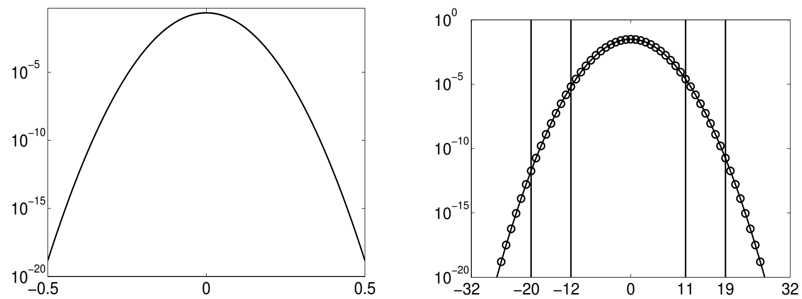
is a short list of some important options: By default, NFFT 3 uses the Kaiser-Bessel window function (see Appendix B). This can be changed by passing the option `--with-window=ARG`, where `ARG` is replaced by one of `kaiserbessel` (Kaiser-Bessel), `gaussian` (Gaussian), `bspline` (B-spline), or `sinc` (sinc power). This makes all routines use the respective window function. NFFT 3 routines can also be configured to measure the computation time for several steps during execution. You can enable this behavior with the options `--enable-measure-time` and/or `--enable-measure-time-fftw`.

B. NFFT - NONEQUISPACED FAST FOURIER TRANSFORM

In this section, we describe the approximate NFFT algorithm in more detail. To illustrate the principles, we restrict ourselves to the one-dimensional case, hence the computation of

$$f(x) = \sum_{k \in I_N} \hat{f}_k \mathrm{e}^{-2\pi \mathrm{i} k x} \qquad (j = 0, \ldots, M-1) \tag{B.1}$$

for nonequispaced nodes $x_j \in \mathbb{T}$. In this case, the NFFT has cost $\mathcal{O}(N \log N + |\log \varepsilon| M)$, where ε is the desired accuracy. The chief idea of the NFFT algorithm is to use standard FFTs in combination with an approximation scheme that is based on a window function φ. This function needs to be mutually well localized in time/spatial and frequency domain. Several such window functions have been proposed in [Dutt and Rokhlin 1993; Beylkin 1995; Steidl 1998; Fourmont 2003; Fessler and Sutton 2003].



(a) Window function. (b) Fourier transform.

Fig. B.1. Example for a suitable window function, in this case the Gaussian window function φ (left) and the sampled integral Fourier-transform $\hat{\varphi}$ (right) with pass, transition, and stop band for $N = 24$, $\sigma = \frac{4}{3}$, $n = 32$.

The ansatz

The first step is to approximate the trigonometric polynomial f of degree N in (B.1) by a linear combination s_1 of shifted one-periodic window functions $\tilde{\varphi}$,

$$s_1(x) := \sum_{l \in I_n} g_l \, \tilde{\varphi}\left(x - \frac{l}{n}\right), \tag{B.2}$$

where $n := \sigma N$ for some oversampling factor $\sigma > 1$. The value of n determines the length of the ordinary FFT used below.

The window function

The window function $\varphi : \mathbb{R} \to \mathbb{R}$ is chosen so that the one-periodic version $\tilde{\varphi}(x) := \sum_{r \in \mathbb{Z}} \varphi(x + r)$ has a uniformly convergent Fourier series $\tilde{\varphi}(x) = \sum_{k \in \mathbb{Z}} c_k(\tilde{\varphi}) \, \mathrm{e}^{-2\pi \mathrm{i} k x}$, and at the same time is well localized in the time/spatial domain $\mathbb{T}$ and in the frequency domain $\mathbb{Z}$. The Fourier coefficients $c_k(\tilde{\varphi})$ of this one-periodic version are samples of the Fourier transform $\hat{\varphi}$ of the original window function at integers points, i.e.,

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

An example is shown in Figure B.1.

The first approximation - cut-off in frequency domain

If we take the definition in (B.2) and switch to the frequency domain, we obtain

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

with discrete Fourier coefficients

$$\hat{g}_k = \sum_{l \in I_n} g_l \, \mathrm{e}^{2\pi \mathrm{i} \frac{kl}{n}}. \tag{B.3}$$

A comparison of (B.1) and (B.2) suggests to define

$$\hat{g}_k := \begin{cases} \frac{\hat{f}_k}{c_k(\tilde{\varphi})} & \text{for } k \in I_N, \\ 0 & \text{for } k \in I_n \setminus I_N, \end{cases} \tag{B.4}$$

assuming that $c_k(\tilde{\varphi})$ is small for $|k| \geq n - \frac{N}{2}$. The values g_l can now be computed by applying the Fourier inversion theorem to (B.3). This step can be realized by an FFT of length n. This first approximation causes an aliasing error.

The second approximation - cut-off in time/spatial domain

If the window function φ is well localized in time/spatial domain then it can be approximated by a truncated version

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

with $\operatorname{supp}\psi = \left[-\frac{m}{n},\frac{m}{n}\right]$, $m \ll n$, $m \in \mathbb{N}$. Here too, we can define a one periodic version $\tilde{\psi}$ with compact support in $\mathbb{T}$ by $\tilde{\psi}(x) = \sum_{r\in\mathbb{Z}} \psi(x+r)$. With the help of the index set $I_{n,m}(x_j) := \{l \in I_n : nx_j - m \le l \le nx_j + m\}$ a further approximation to the first approximation s_1 at the nodes x_j can be defined by

$$s(x_j) := \sum_{l\in I_{n,m}(x_j)} g_l\,\tilde{\varphi}\left(x_j - \frac{l}{n}\right) = \sum_{l\in I_n} g_l\,\tilde{\psi}\left(x_j - \frac{l}{n}\right)\,. \tag{B.5}$$

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

The case $d > 1$

The described approximation method can be generalized to arbitrary dimensions. But first, a few generalizations have to be introduced. A multivariate window function is now defined by $\varphi(\boldsymbol{x}) := \prod_{t=0}^{d-1} \varphi_t(x_t)$ with a univariate window function φ_t. A simple consequence is that the Fourier coefficients have the form $c_{\boldsymbol{k}}(\tilde{\varphi}) = \prod_{t=0}^{d-1} c_{k_t}(\tilde{\varphi}_t)$. The ansatz function s_1 reads now

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

The size of the FFT size is determined to $\boldsymbol{n} := \boldsymbol{\sigma}\odot\boldsymbol{N}$, and the oversampling factor is $\boldsymbol{\sigma} = (\sigma_0,\ldots,\sigma_{d-1})^\top$. Following (B.4) we define

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

The values $g_{\boldsymbol{l}}$ are obtained by a (multivariate) FFT of size $n_0\times n_1\times\ldots\times n_{d-1}$. Using the compactly supported function $\psi(\boldsymbol{x}) = \varphi(\boldsymbol{x})\,\chi_{[-\frac{m}{n},\frac{m}{n}]^d}(\boldsymbol{x})$ gives the final form,

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

Again, $\tilde{\psi}$ denotes the one-periodic version of ψ. Note that the multi-index set $I_{\boldsymbol{n},m}(\boldsymbol{x}_j) := \{\boldsymbol{l}\in I_{\boldsymbol{n}} : \boldsymbol{n}\odot\boldsymbol{x}_j - m\boldsymbol{1} \le \boldsymbol{l} \le \boldsymbol{n}\odot\boldsymbol{x}_j + m\boldsymbol{1}\}$ has at most $(2m+1)^d$ entries for every $\boldsymbol{x}_j$.

The final algorithm

The NFFT algorithm is given in pseudo-code in Algorithm 2. In addition to the evaluation of the window functions, it requires roughly $|I_{\boldsymbol{N}}| + |I_{\boldsymbol{n}}|\log|I_{\boldsymbol{n}}| + 2(2m+1)^d M$ floating point operations. In matrix-vector notation, Algorithm 2 can be written as $\boldsymbol{A}\hat{\boldsymbol{f}} \approx \boldsymbol{D}\boldsymbol{F}\boldsymbol{B}\hat{\boldsymbol{f}}$, where $\boldsymbol{B}$ denotes the sparse real $M\times|I_{\boldsymbol{n}}|$ matrix

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

The matrix $\boldsymbol{F} := (\mathrm{e}^{-2\pi\mathrm{i}\boldsymbol{k}(\boldsymbol{N}^{-1}\odot\boldsymbol{j})})_{\boldsymbol{j},\boldsymbol{k}\in I_{\boldsymbol{n}}}$ is the ordinary Fourier matrix of size $|I_{\boldsymbol{n}}|\times|I_{\boldsymbol{n}}|$ and $\boldsymbol{D}$ is a real $|I_{\boldsymbol{n}}|\times|I_{\boldsymbol{N}}|$ "diagonal" matrix defined by

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

with zero matrices $\boldsymbol{O}_t$ of size $N_t\times\frac{n_t-N_t}{2}$. The corresponding adjoint matrix-vector product is given by $\boldsymbol{A}^{\mathsf{H}}\hat{\boldsymbol{f}} \approx \boldsymbol{B}^\top\boldsymbol{F}^{\mathsf{H}}\boldsymbol{D}^\top\hat{\boldsymbol{f}}$. With the help of the transposed index set $I^\top_{\boldsymbol{n},m}(\boldsymbol{l}) := \{j = 0,\ldots,M-1 : \boldsymbol{l} - m\boldsymbol{1} \le \boldsymbol{n}\odot\boldsymbol{x}_j \le \boldsymbol{l} + m\boldsymbol{1}\}$, one obtains Algorithm 3 for the adjoint NFFT.

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

1: For $\boldsymbol{k} \in I_{\boldsymbol{N}}$ compute $\hat{g}_{\boldsymbol{k}} := |I_{\boldsymbol{n}}|^{-1} \cdot \hat{f}_{\boldsymbol{k}} / c_{\boldsymbol{k}}(\tilde{\varphi})$.

2: For $\boldsymbol{l} \in I_{\boldsymbol{n}}$ compute $g_{\boldsymbol{l}} := \sum_{\boldsymbol{k} \in I_{\boldsymbol{N}}} \hat{g}_{\boldsymbol{k}} \, \mathrm{e}^{-2\pi \mathrm{i} \boldsymbol{k}(\boldsymbol{n}^{-1} \odot \boldsymbol{l})}$ by a d-variate FFT.

3: For $j = 0, \dots, M-1$ compute $f_j := \sum_{\boldsymbol{l} \in I_{\boldsymbol{n},m}(\boldsymbol{x}_j)} g_{\boldsymbol{l}} \, \tilde{\varphi}(\boldsymbol{x}_j - \boldsymbol{n}^{-1} \odot \boldsymbol{l})$.

Output: Approximate function values f_j, $j = 0, \dots, M-1$.
Arithmetic cost: $|I_{\boldsymbol{N}}| + |I_{\boldsymbol{n}}| \log |I_{\boldsymbol{n}}| + 2(2m+1)^d M$ + evaluations of the window function.

Alg. 2. Nonequispaced fast Fourier transform (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 $f_j \in \mathbb{C}$, $j = 0, \dots, M-1$.

1: For $\boldsymbol{l} \in I_{\boldsymbol{n}}$ compute $g_{\boldsymbol{l}} := \sum\limits_{j \in I_{\boldsymbol{n},m}^{\top}(\boldsymbol{l})} f_j \, \tilde{\varphi}(\boldsymbol{x}_j - \boldsymbol{n}^{-1} \odot \boldsymbol{l})$.

2: For $\boldsymbol{k} \in I_{\boldsymbol{N}}$ compute $\hat{g}_{\boldsymbol{k}} := \sum\limits_{\boldsymbol{l} \in I_{\boldsymbol{n}}} g_{\boldsymbol{l}} \, \mathrm{e}^{+2\pi \mathrm{i} \boldsymbol{k}(\boldsymbol{n}^{-1} \odot \boldsymbol{l})}$ by d-variate (backward) FFT.

3: For $\boldsymbol{k} \in I_{\boldsymbol{N}}$ compute $\hat{h}_{\boldsymbol{k}} := |I_{\boldsymbol{n}}|^{-1} \cdot \hat{g}_{\boldsymbol{k}} / c_{\boldsymbol{k}}(\tilde{\varphi})$.

Output: Approximate coefficients $\hat{h}_{\boldsymbol{k}}$, $\boldsymbol{k} \in I_{\boldsymbol{N}}$.
Arithmetic cost: $|I_{\boldsymbol{N}}| + |I_{\boldsymbol{n}}| \log |I_{\boldsymbol{n}}| + 2(2m+1)^d M$ + evaluation of the window function.

Alg. 3. Adjoint nonequispaced fast Fourier transform (adjoint NFFT).

C. AVAILABLE WINDOW FUNCTIONS AND EVALUATION TECHNIQUES

For clarity of the presentation, we restrict ourselves to the case $d = 1$, again. To keep the aliasing error and the truncation error small, several functions φ with good localization in time and frequency domain have been proposed, e.g. the (dilated) *Gaussian* [Dutt and Rokhlin 1993; Steidl 1998; Duijndam and Schonewille 1999] (with shape parameter $b := \frac{2\sigma}{2\sigma-1} \frac{m}{\pi}$),

$$\varphi(x) = (\pi b)^{-1/2} \, \mathrm{e}^{-\frac{(nx)^2}{b}}, \qquad \hat{\varphi}(k) = \frac{1}{n} \mathrm{e}^{-b\left(\frac{\pi k}{n}\right)^2}, \tag{C.1}$$

(dilated) *cardinal central B–splines* [Beylkin 1995; Steidl 1998],

$$\varphi(x) = M_{2m}(nx), \qquad \hat{\varphi}(k) = \frac{1}{n} \mathrm{sinc}^{2m}(k\pi/n), \tag{C.2}$$

where M_{2m} denotes the centered cardinal B–Spline of order $2m$, (dilated) *Sinc functions* [Potts 2003b],

$$\varphi(x) = \frac{N(2\sigma-1)}{2m} \mathrm{sinc}^{2m}\left(\frac{(\pi N x (2\sigma-1))}{2m}\right), \qquad \hat{\varphi}(k) = M_{2m}\left(\frac{2mk}{(2\sigma-1)N}\right), \tag{C.3}$$

and (dilated) *Kaiser–Bessel functions* [Jackson et al. 1991; Fourmont 2003] (with shape parameter $b := \pi\left(2 - \frac{1}{\sigma}\right)$),

$$\begin{aligned}
\varphi(x) &= \frac{1}{\pi} \begin{cases} \dfrac{\sinh\left(b\sqrt{m^2 - n^2x^2}\right)}{\sqrt{m^2 - n^2x^2}}, & \text{for } |x| \le \frac{m}{n}, \\[2ex] \dfrac{\sin\left(b\sqrt{n^2x^2 - m^2}\right)}{\sqrt{n^2x^2 - m^2}}, & \text{otherwise}, \end{cases} \\
\hat{\varphi}(k) &= \frac{1}{n} \begin{cases} I_0\left(m\sqrt{b^2 - (2\pi k/n)^2}\right), & \text{for } k = -n\left(1 - \frac{1}{2\sigma}\right), \dots, n\left(1 - \frac{1}{2\sigma}\right), \\ 0, & \text{otherwise}, \end{cases}
\end{aligned} \tag{C.4}$$

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

$$|f(x_j) - s(x_j)| \le C(\sigma, m) \sum_{k \in I_N} |\hat{f}_k|,$$

where

$$C(\sigma, m) := \begin{cases} 4\,\mathrm{e}^{-m\pi(1-1/(2\sigma-1))}, & \text{for (C.1)},\\ 4\left(\frac{1}{2\sigma-1}\right)^{2m}, & \text{for (C.2)},\\ \frac{1}{m-1}\left(\frac{2}{\sigma^{2m}} + \left(\frac{\sigma}{2\sigma-1}\right)^{2m}\right), & \text{for (C.3)},\\ 4\pi\left(\sqrt{m}+m\right)\sqrt[4]{1-\frac{1}{\sigma}}\,\mathrm{e}^{-2\pi m\sqrt{1-1/\sigma}}, & \text{for (C.4)}. \end{cases}$$

Thus, for fixed $\sigma > 1$ the approximation error introduced by the NFFT decays exponentially with the number of summands m in (B.5). These estimates can of course be generalized to higher dimensions; see [Elbel and Steidl 1998].

It must be noted that the cost of the NFFT also increases with m. In the following, we will suggest different methods for efficient storage and application of the matrix $\boldsymbol{B}$. These are all available in the NFFT 3 library by choosing particular flags during the initialization phase.

Fully precomputed window function

An obvious possibility is to precompute all values $\varphi(\boldsymbol{x}_j - \boldsymbol{n}^{-1} \odot \boldsymbol{l})$ for $j = 0, \dots, M-1$ and $\boldsymbol{l} \in I_{\boldsymbol{n},m}(\boldsymbol{x}_j)$ which are needed for $\boldsymbol{B}$. This needs to store the possibly large amount of $(2m+1)^d M$ real numbers. On the other hand, no additional computation is required during the matrix-vector multiplication apart from the inevitable $2(2m+1)^d M$ floating point operations. This method, indicated by the flag `PRE_FULL_PSI`, is the fastest procedure but can only be used if enough memory for storage is available.

Tensor product based precomputation

Using the fact that the window functions are represented by tensor products one can resort to storing only the values $\varphi_t((\boldsymbol{x}_j)_t - \frac{l_t}{n_t})$ for $j = 0, \dots, M-1$, $t = 0, \dots, d-1$, and $l_t \in I_{n_t,m}((\boldsymbol{x}_j)_t)$, where $(\boldsymbol{x}_j)_t$ denotes the tth component of the jth node. This method needs to store $d(2m+1)M$ real numbers which is less than the first method needs. However, for each node one possibly needs up to $2(2m+1)^d$ extra multiplications to obtain the values of the multivariate window function $\varphi(\boldsymbol{x}_j - \boldsymbol{n}^{-1} \odot \boldsymbol{l})$ for $\boldsymbol{l} \in I_{\boldsymbol{n},m}(\boldsymbol{x}_j)$. This technique is available for every window function discussed and can be used by setting the flag `PRE_PSI`. This is also the default method used in our library.

Linear interpolation from a lookup table

For a large number of nodes M, the amount of memory can be further reduced by using lookup table techniques. For a recent example within the framework of gridding see [Beatty et al. 2005]. We suggest to precompute from the even window function the equidistant samples $\varphi_t(\frac{rm}{Kn_t})$ for $t = 0, \dots, d-1$ and $r = 0, \dots, K$, $K \in \mathbb{N}$. Then for the actual node $\boldsymbol{x}_j$ during the NFFT the values $\varphi_t((\boldsymbol{x}_j)_t - \frac{l_t}{n_t})$ for $t = 0, \dots, d-1$ and $l_t \in I_{n_t,m}((\boldsymbol{x}_j)_t)$ are computed by linear interpolation from its two neighboring values. This method needs to store only dK real numbers in total, where K depends only on the desired accuracy but neither on the number of nodes M nor on the multibandwidth $\boldsymbol{N}$. If we choose K to be a multiple of m, we can further reduce the computational costs during the interpolation step since the distance from $(\boldsymbol{x}_j)_t - \frac{l_t}{n_t}$ to the two neighboring values is the same for all $l_t \in I_{n_t,m}((\boldsymbol{x}_j)_t)$. During the actual multiplication with the matrix $\boldsymbol{B}$, this method requires $2(2m+1)^d$ extra multiplications per node. It can be used by setting the flag `PRE_LIN_PSI`. Error estimates for this additional approximation are given in [Kunis and Potts 2008].

Fast Gaussian gridding

Two useful properties of the Gaussian window function (C.1) that can be exploited within the presented framework have recently been reviewed in [Greengard and Lee 2004]. Beside the tensor product structure for $d > 1$, which is also valid for all other window functions, it is remarkable that the number of evaluations of the exponential function `exp()` can be reduced by a substantially. More precisely, for $d = 1$ and a fixed node x_j the evaluation of $\varphi(x_j - \frac{l'}{n})$, $l' \in I_{n,m}(x_j)$ can be computed more efficiently by writing this as

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

where $u = \min I_{n,m}(x_j)$ and $l = 0, \ldots, 2m$. The first factor and the exponential within the brackets are a constant for each fixed node x_j. Once we have evaluated the second exponential, its lth power can be computed by repeated multiplications only. Furthermore, the last exponential is independent of x_j such that these $2m + 1$ different values need to be precomputed only once – usually a negligible amount. Thus, it is sufficient to store or evaluate $2M$ exponentials for $d = 1$. The case $d > 1$ needs to store or evaluate $2dM$ numbers. This follows from the tensor product structure. The described method is employed when the flags `FG_PSI` and possibly `PRE_FG_PSI` are used.

No precomputation of the window function

The last method considered does not precompute any values of the window function at all. Instead, all values are calculated on-line. This needs to be done at most $\frac{3}{2}(2m + 1)^d M$ times. The computation time strongly depends on how fast one can evaluate the particular window function. Since this method does not need any additional storage, it is attractive whenever large problems are considered that would push the other methods beyond the memory limitations.

D. FURTHER NFFT APPROACHES

Besides [Keiner et al. 2006b], the MatLab toolbox [Fessler and Sutton 2002] also implements the NFFT. In several papers, approximations techniques similar to the NFFT have been described. Common names for these schemes are *non-uniform fast Fourier transform* [Fessler and Sutton 2003], *generalized fast Fourier transform* [Dutt and Rokhlin 1993], *unequally-spaced fast Fourier transform* [Beylkin 1995], *fast approximate Fourier transforms for irregularly spaced data* [Ware 1998], *non-equispaced fast Fourier transform* [Fourmont 2003] or *gridding* [Sramek and Schwab 1989; Jackson et al. 1991; Pelt 1997].

Also, different window functions have been considered in various papers, e.g. a Gaussian pulse tapered with a Hanning window [Duijndam and Schonewille 1999], Gaussian kernels combined with Sinc kernels [Pelt 1997], and particular optimized windows [Jackson et al. 1991; Duijndam and Schonewille 1999]. Special approaches based on scaling vectors [Nguyen and Liu 1999], minimizing the Frobenius norm of certain error matrices [Nieslony and Steidl 2003], or min-max interpolation [Fessler and Sutton 2003] have been proposed. While these approaches lead to better accuracy for the Gaussian or B-Spline windows, improvements for the Kaiser-Bessel window are marginal. Recently, the time consuming multiplication with the matrix $\boldsymbol{B}$ has been accelerated by using commodity graphics hardware [Sorensen et al. 2008].

The following two approaches have also been considered for the univariate case $d = 1$: The authors of [Anderson and Dahleh 1996] use for each node x_j an mth order Taylor expansion of the trigonometric polynomial (B.1) around the nearest neighboring point on an oversampled equispaced lattice I_n/n. Here again we have $n = \sigma N$, $\sigma \gg 1$. This algorithm utilizes m FFTs of size n (compared to only one in our approach) and uses a fair amount of extra memory; see also [Kunis and Potts 2008; Kunis 2008]. Another approach is considered in [Dutt and Rokhlin 1995]. It is based on a Lagrange interpolation technique. After computing an FFT of the vector $\hat{\boldsymbol{f}} \in \mathbb{C}^N$ in (2.3) an exact polynomial interpolation scheme is used to obtain the values of the trigonometric polynomial f at the nonequispaced nodes x_j. This is the most time consuming part of this method. In an approximate way, it can however be realized with the help of the fast multipole method. This approach is appealing since it allows also for computing an inverse transform. Nevertheless, numerical experiments in [Dutt and Rokhlin 1995] indicate that this approach is far more time consuming than Algorithm 2. Also, the inversion can only be computed in a stable way for almost equispaced nodes [Dutt and Rokhlin 1995].

For a comparison of different approaches to nonequispaced Fourier transforms, we also refer to [Ware 1998; Nieslony and Steidl 2003; Fessler and Sutton 2003; Kunis and Potts 2008].

ACKNOWLEDGMENTS

We thank G. Steidl for numerous fruitful and enlightning discussions. Contributions to the source code made by M. Böhme, M. Fenn, T. Knopp, and S. Klatt are gratefully acknowledged. For details, see the `AUTHORS` file in the package directory. We also thank the anonymous referees and the editor for many helpful comments and valuable suggestions.

REFERENCES

Abramowitz, M. and Stegun, I. A., Eds. 1972. *Handbook of Mathematical Functions*. National Bureau of Standards, Washington, DC, USA.

Anderson, C. and Dahleh, M. 1996. Rapid computation of the discrete Fourier transform. *SIAM J. Sci. Comput. 17*, 913 – 919.

Andersson, F. and Beylkin, G. 2005. The fast Gauss transform with complex parameters. *J. Comput. Physics 203*, 274 – 286.

Averbuch, A., Coifman, R., Donoho, D. L., Elad, M., and Israeli, M. 2006. Fast and accurate polar Fourier transform. *Appl. Comput. Harmon. Anal. 21*, 145 – 167.

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

Beatson, R. K. and Greengard, L. 1997. A short course on fast multipole methods. In *Wavelets, Multilevel Methods and Elliptic PDEs*, M. Ainsworth, J. Levesley, W. A. Light, and M. Marletta, Eds. Clarendon Press, Oxford, 1 – 37.

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

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

Beylkin, G., Kurcz, C., and Monzón, L. 2007. Grids and transforms for band-limited functions in a disk. *Inverse Problems 23*, 2059 – 2088.

Björck, Å. 1996. *Numerical Methods for Least Squares Problems*. SIAM, Philadelphia, PA, USA.

Bungartz, H.-J. and Griebel, M. 2004. Sparse grids. *Acta Numer. 13*, 147 – 269.

Candes, E. J., Demanet, L., Donoho, D. L., and Ying, L. 2006. Fast discrete curvelet transforms. *SIAM Multiscale Model. Simul. 3*, 861 – 899.

Cooley, J. W. and Tukey, J. W. 1965. An algorithm for machine calculation of complex Fourier series. *Math. Comput. 19*, 297 – 301.

Donoho, D., Maleki, A., and Shaharam, M. 2006. Wavelab 850. `http://www-stat.stanford.edu/~wavelab`.

Driscoll, J. R. and Healy, D. 1994. Computing Fourier transforms and convolutions on the 2–sphere. *Adv. in Appl. Math. 15*, 202 – 250.

Driscoll, J. R., Healy, D., and Rockmore, D. 1996. Fast discrete polynomial transforms with applications to data analysis for distance transitive graphs. *SIAM J. Comput. 26*, 1066 – 1099.

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

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

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

Eggers, H., Knopp, T., and Potts, D. 2007. Field inhomogeneity correction based on gridding reconstruction. IEEE *Trans. Med. Imag. 26*, 374 – 384.

Elbel, B. and Steidl, G. 1998. Fast Fourier transform for nonequispaced data. In *Approximation Theory IX*, C. K. Chui and L. L. Schumaker, Eds. Vanderbilt University Press, Nashville, 39 – 46.

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

Fenn, M., Kunis, S., and Potts, D. 2006. Fast evaluation of trigonometric polynomials from hyperbolic crosses. *Numer. Algorithms 41*, 339 – 352.

Fenn, M., Kunis, S., and Potts, D. 2007. On the computation of the polar FFT. *Appl. Comput. Harmon. Anal. 22*, 257 – 263.

Fenn, M. and Potts, D. 2005. Fast summation based on fast trigonometric transforms at nonequispaced nodes. *Numer. Linear Algebra Appl. 12*, 161 – 169.

Fenn, M. and Steidl, G. 2004. Fast NFFT based summation of radial functions. *Sampling Theory in Signal and Image Processing 3*, 1 – 28.

FESSLER, J. A. AND SUTTON, B. P. 2002. NUFFT - nonuniform FFT toolbox for Matlab. `http://www.eecs.umich.edu/~fessler/code`.

FESSLER, J. A. AND SUTTON, B. P. 2003. Nonuniform fast Fourier transforms using min-max interpolation. IEEE *Trans. Signal Process. 51*, 560 – 574.

FOURMONT, K. 2003. Non equispaced fast Fourier transforms with applications to tomography. *J. Fourier Anal. Appl. 9*, 431 – 450.

FRIGO, M. AND JOHNSON, S. G. 2005a. The design and implementation of FFTW3. *Proceedings of the* IEEE *93*, 216–231.

FRIGO, M. AND JOHNSON, S. G. 2005b. FFTW, C subroutine library. `http://www.fftw.org`.

GREENGARD, L. AND LEE, J.-Y. 2004. Accelerating the nonuniform fast Fourier transform. *SIAM Rev. 46*, 443 – 454.

JACKSON, J. I., MEYER, C. H., NISHIMURA, D. G., AND MACOVSKI, A. 1991. Selection of a convolution function for Fourier inversion using gridding. IEEE *Trans. Med. Imag. 10*, 473 – 478.

KEINER, J., KUNIS, S., AND POTTS, D. 2006a. Fast summation of radial functions on the sphere. *Computing 78*, 1 – 15.

KEINER, J., KUNIS, S., AND POTTS, D. 2006b. NFFT 3.0, C subroutine library. `http://www.tu-chemnitz.de/~potts/nfft`.

KEINER, J., KUNIS, S., AND POTTS, D. 2007. Efficient reconstruction of functions on the sphere from scattered data. *J. Fourier Anal. Appl. 13*, 435 – 458.

KEINER, J. AND POTTS, D. 2008. Fast evaluation of quadrature formulae on the sphere. *Math. Comput. 77*, 397 – 419.

KNOPP, T., KUNIS, S., AND POTTS, D. 2007. A note on the iterative MRI reconstruction from nonuniform k-space data. *Int. J. Biomed. Imag.*. ID 24727.

KUNIS, S. 2008. Nonequispaced fast Fourier transforms without oversampling. *Preprint*.

KUNIS, S. AND POTTS, D. 2003. Fast spherical Fourier algorithms. *J. Comput. Appl. Math. 161*, 75 – 98.

KUNIS, S. AND POTTS, D. 2007. Stability results for scattered data interpolation by trigonometric polynomials. *SIAM J. Sci. Comput. 29*, 1403 – 1419.

KUNIS, S. AND POTTS, D. 2008. Time and memory requirements of the nonequispaced FFT. *Sampl. Theory Signal Image Process. 7*, 77 – 100.

KUNIS, S., POTTS, D., AND STEIDL, G. 2006. Fast Gauss transform with complex parameters using NFFTs. *J. Numer. Math. 14*, 295 – 303.

LEE, J.-Y. AND GREENGARD, L. 2005. The type 3 nonuniform FFT and its applications. *J. Comput. Physics 206*, 1 – 5.

MA, J. AND FENN, M. 2006. Combined complex ridgelet shrinkage and total variation minimization. *SIAM J. Sci. Comput. 28*, 984 – 1000.

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION. 2007. NASA AIRS Homepage. `http://disc.gsfc.nasa.gov/AIRS/index.shtml`.

NGUYEN, N. AND LIU, Q. H. 1999. The regular Fourier matrices and nonuniform fast Fourier transforms. *SIAM J. Sci. Comput. 21*, 283 – 293.

NIESLONY, A. AND STEIDL, G. 2003. Approximate factorizations of Fourier matrices with nonequispaced knots. *Linear Algebra Appl. 266*, 337 – 351.

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

PÖPLAU, G., POTTS, D., AND VAN RIENEN, U. 2006. Calculation of 3D space-charge fields of bunches of charged particles by fast summation. In *Scientific Computing in Electrical Engineering*, A. Anile, G. Alì, and G. Mascaly, Eds. Springer, 241 – 246.

POTTS, D. 2003a. Fast algorithms for discrete polynomial transforms on arbitrary grids. *Linear Algebra Appl. 366*, 353 – 370.

POTTS, D. 2003b. Schnelle Fourier-Transformationen für nichtäquidistante Daten und Anwendungen. Habilitation, Universität zu Lübeck, `http://www.tu-chemnitz.de/~potts`.

POTTS, D. AND STEIDL, G. 2000. New Fourier reconstruction algorithms for computerized tomography. In *Proceedings of SPIE: Wavelet Applications in Signal and Image Processing VIII*, A. Aldroubi, A. Laine, and M. Unser, Eds. Vol. 4119. 13 – 23.

POTTS, D. AND STEIDL, G. 2001. A new linogram algorithm for computerized tomography. IMA *J. Numer. Anal. 21*, 769 – 782.

POTTS, D. AND STEIDL, G. 2002. Fourier reconstruction of functions from their nonstandard sampled Radon transform. *J. Fourier Anal. Appl. 8*, 513 – 533.

POTTS, D. AND STEIDL, G. 2003. Fast summation at nonequispaced knots by NFFTs. *SIAM J. Sci. Comput. 24*, 2013 – 2037.

POTTS, D., STEIDL, G., AND NIESLONY, A. 2004. Fast convolution with radial kernels at nonequispaced knots. *Numer. Math. 98*, 329 – 351.

POTTS, D., STEIDL, G., AND TASCHE, M. 1998. Fast algorithms for discrete polynomial transforms. *Math. Comput. 67*, 1577 – 1590.

POTTS, D., STEIDL, G., AND TASCHE, M. 2001. Fast Fourier transforms for nonequispaced data: A tutorial. In *Modern Sampling Theory: Mathematics and Applications*, J. J. Benedetto and P. J. S. G. Ferreira, Eds. Birkhäuser, Boston, 247 – 270.

ROKHLIN, V. AND TYGERT, M. 2006. Fast algorithms for spherical harmonic Expansions. *SIAM J. Sci. Comput. 27*, 1903 – 1928.

SORENSEN, T. S., SCHAEFFTER, T., NO, K. O., AND HANSEN, M. S. 2008. Accelerating the nonequispaced fast Fourier transform on commodity graphics hardware. IEEE *Trans. Med. Imag. 27*, 538 – 547.

SPRENGEL, F. 2000. A class of function spaces and interpolation on sparse grids. *Numer. Funct. Anal. Optim. 21*, 273 – 293.

SRAMEK, R. A. AND SCHWAB, F. R. 1989. Imaging. In *Synthesis Imaging in Radio Astronomy: A Collection of Lectures from the Third NRAO Synthesis Imaging Summer School*, R. Perley, F. R. Schwab, and A. Bridle, Eds. Astronomical Society of the Pacific, 83 – 138.

STEIDL, G. 1998. A note on fast Fourier transforms for nonequispaced grids. *Adv. Comput. Math. 9*, 337 – 353.

SUDA, R. AND TAKAMI, M. 2002. A fast spherical harmonics transform algorithm. *Math. Comput. 71*, 703 – 715.

SUTTON, B. P., NOLL, D. C., AND FESSLER, J. A. 2003. Fast, iterative, field-corrected image reconstruction for MRI in the presence of field inhomogeneities. IEEE *Trans. Med. Imag. 22*, 178 – 188.

TIAN, B. AND LIU, Q. H. 2000. Nonuniform fast cosine transform and Chebyshev PSTD algorithm. *J. Electromagnet. Waves Appl 14*, 797 – 798.

VAN LOAN, C. F. 1992. *Computational Frameworks for the Fast Fourier Transform.* SIAM, Philadelphia, PA, USA.

WARE, A. F. 1998. Fast approximate Fourier transforms for irregularly spaced data. *SIAM Rev. 40*, 838 – 856.

ZENGER, C. 1991. Sparse grids. In *Parallel algorithms for partial differential equations (Kiel, 1990).* Notes Numer. Fluid Mech., vol. 31. Vieweg, 241–251.

Received Month Year; revised Month Year; accepted Month Year