NFFT

3.5.2

Generated by Doxygen 1.8.19

Chapter 1

Main Page

1.1 Introduction

Fast Fourier transforms (FFTs) belong to the '10 algorithms with the greatest influence on the development and practice of science and engineering in the 20th century'. The classic algorithm computes the discrete Fourier transform

$$f_j = \sum_{k=-\frac{N}{2}}^{\frac{N}{2}-1} \hat{f}_k \mathrm{e}^{-2\pi \mathrm{i} \frac{kj}{N}}$$

for $j = -\frac{N}{2}, \ldots, \frac{N}{2} - 1$ and given complex coefficients $\hat{f}_k \in \mathbb{C}$. Using a divide and conquer approach, the number of floating point operations is reduced from $\mathcal{O}(N^2)$ for a straightforward computation to only $\mathcal{O}(N \log N)$. In conjunction with publicly available efficient implementations the fast Fourier transform has become of great importance in scientific computing.

However, two shortcomings of traditional schemes are the need for equispaced sampling and the restriction to the system of complex exponential functions. The NFFT is a C subroutine library for computing the nonequispaced discrete Fourier transform (NDFT) and its generalisations in one or more dimensions, of arbitrary input size, and of complex data.

More precisely, we collect the possible frequencies $\mathbf{k} \in \mathbb{Z}^d$ in the multi-index set

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

where $\mathbf{N} = (N_t)_{t=0,\ldots,d-1}$ is the multibandlimit, i.e., $N_t \in 2\mathbb{N}$. For a finite number of given Fourier coefficients $\hat{f}_{\mathbf{k}} \in \mathbb{C}$, $\mathbf{k} \in I_{\mathbf{N}}$, we consider the fast evaluation of the trigonometric polynomial

$$f(\mathbf{x}) := \sum_{\mathbf{k} \in I_N} \hat{f}_{\mathbf{k}} \mathrm{e}^{-2\pi \mathrm{i} \mathbf{k} \mathbf{x}}$$

at given nonequispaced nodes $\mathbf{x}_j \in \mathbb{T}^d$, $j = 0, \ldots, M-1$, from the d-dimensional torus as well as the adjoint problem, the fast evaluation of sums of the form

$$\hat{h}_{\mathbf{k}} := \sum_{j=0}^{M-1} f_j \mathrm{e}^{2\pi \mathrm{i} \mathbf{k} \mathbf{x}_j}.$$

1.1.1 Generalisations

The generalisations of the NFFT include

- NNFFT - nonequispaced in time and frequency fast Fourier transform,
- NFCT/NFST - nonequispaced fast (co)sine transform,
- NSFFT - nonequispaced sparse fast Fourier transform,
- FPT - fast polynomial transform,
- NFSFT - nonequispaced fast spherical Fourier transform.

Furthermore, we consider the inversion of the above transforms by iterative methods.

1.2 FAQ - Frequently Asked Questions

see https://www.tu-chemnitz.de/~potts/nfft/faq.php

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 5

Module Documentation

5.1 FPT - Fast polynomial transform

This module implements fast polynomial transforms.

Macros

- #define FPT_NO_FAST_ALGORITHM (1U << 2)

  If set, TODO complete comment.
- #define FPT_NO_DIRECT_ALGORITHM (1U << 3)

  If set, TODO complete comment.
- #define FPT_NO_STABILIZATION (1U << 0)

  If set, no stabilization will be used.
- #define FPT_PERSISTENT_DATA (1U << 4)

  If set, TODO complete comment.
- #define FPT_FUNCTION_VALUES (1U << 5)

  If set, the output are function values at Chebyshev nodes rather than Chebyshev coefficients.
- #define FPT_AL_SYMMETRY (1U << 6)

  If set, TODO complete comment.

Functions

- fpt_set fpt_init (const int M, const int t, const unsigned int flags)
- void fpt_precompute (fpt_set set, const int m, double *alpha, double *beta, double *gam, int k_start, const double threshold)
- void fpt_transposed (fpt_set set, const int m, double _Complex *x, double _Complex *y, const int k_end, const unsigned int flags)

Variables

- *We expand this macro for each supported precision * X

5.1.1 Detailed Description

This module implements fast polynomial transforms.

In the following, we abbreviate the term "fast polynomial transforms" by FPT.

Let $\alpha_n,\ \beta_n,\ \gamma_n,\ n = 0,\ldots,N$ be given recursion coefficients of the polynomials P_n defined by $P_{-1}(x) = 0$, $P_0(x) = 1$ and

$$P_n(x) = (\alpha_n x + \beta_n)P_{n-1}(x) + \gamma_n P_{n-2}(x), \qquad n = 1, 2, \ldots$$

for $x \in [-1, 1]$. The Chebyshev polnyomials of the first kind are defined by

$$T_n(x) = \cos(n \arccos x).$$

Let $f\colon [-1, 1] \to \mathbb{R}$ be a polynomial of degree $N \in \mathbb{N}$. The FPT transforms the polynomial coefficients $[x_n]_{n=0..N}$ from

$$f = \sum_{n=0}^{N} x_n P_n$$

into Chebyshev coefficients $[y_n]_{n=0..N}$ from

$$f = \sum_{n=0}^{N} y_n T_n.$$

5.1.2 Function Documentation

5.1.2.1 fpt_init()

```
fpt_set fpt_init (
            const int M,
            const int t,
            const unsigned int flags )
```

Initializes a set of precomputed data for DPT transforms of equal length.

- M The maximum DPT transform index $M \in \mathbb{N}_0$. The individual transforms are addressed by and index number $m \in \mathbb{N}_0$ with range $m = 0,\ldots,M$. The total number of transforms is therefore $M + 1$.
- t The exponent $t \in \mathbb{N}, t \geq 2$ of the transform length $N = 2^t \in \mathbb{N}, N \geq 4$
- flags A bitwise combination of the flags FPT_NO_STABILIZATION,

Author

Jens Keiner

Definition at line 795 of file fpt.c.

References fpt_set_s_::flags, fpt_set_s_::M, fpt_set_s_::N, fpt_set_s_::t, and X.

Referenced by nfsft_precompute().

5.1.2.2 fpt_precompute()

```
void fpt_precompute (
             fpt_set set,
             const int m,
             double * alpha,
             double * beta,
             double * gam,
             int k_start,
             const double threshold )
```

Computes the data required for a single DPT transform.

- set The set of DPT transform data where the computed data will be stored.
- m The transform index $m \in \mathbb{N}, 0 \leq m \leq M$.
- alpha The three-term recurrence coefficients $\alpha_k \in \mathbb{R}$ for $k = 0, \ldots, N$ such that `alpha[k]` $= \alpha_k$.
- beta The three-term recurrence coefficients $\beta_k \in \mathbb{R}$ for $k = 0, \ldots, N$ such that `beta[k]` $= \beta_k$.
- gamma The three-term recurrence coefficients $\gamma_k \in \mathbb{R}$ for $k = 0, \ldots, N$ such that `gamma[k]` $= \gamma_k$.
- k_start The index $k_{\text{start}} \in \mathbb{N}_0, 0 \leq k_{\text{start}} \leq N$
- threshold The threshold $\kappa \in \mathbb{R}, \kappa > 0$.

Author

Jens Keiner

Definition at line 1307 of file fpt.c.

5.1.2.3 fpt_transposed()

```
void fpt_transposed (
             fpt_set set,
             const int m,
             double _Complex * x,
             double _Complex * y,
             const int k_end,
             const unsigned int flags )
```

Computes a single DPT transform.

- set
- m
- x
- y
- k_end
- flags

Definition at line 1740 of file fpt.c.

5.1.3 Variable Documentation

5.1.3.1 X

`void X`

Computes a single DPT transform.

- set
- m
- x
- y
- k_end
- flags

Initialisation of a transform plan, guru.

- ths The pointer to a nfft plan
- d The dimension
- N The multi bandwidth
- M The number of nodes
- n The oversampled multi bandwidth
- m The spatial cut-off
- flags NFFT flags to use
- fftw_flags_off FFTW flags to use

Author

Stefan Kunis, Daniel Potts

Definition at line 94 of file nfft3.h.

Referenced by fpt_init(), nfsft_precompute(), nfsoft_adjoint(), and nfsoft_trafo().

5.2 MRI - Transforms in magnetic resonance imaging

Data Structures

- struct mri_inh_2d1d_plan
- struct mri_inh_3d_plan

Functions

- void mri_inh_2d1d_trafo (mri_inh_2d1d_plan *that)
- void mri_inh_2d1d_init_guru (mri_inh_2d1d_plan *ths, int *N, int M, int *n, int m, double sigma, unsigned nfft_flags, unsigned fftw_flags)
- void mri_inh_2d1d_finalize (mri_inh_2d1d_plan *ths)
- void mri_inh_3d_trafo (mri_inh_3d_plan *that)
- void mri_inh_3d_adjoint (mri_inh_3d_plan *that)
- void mri_inh_3d_finalize (mri_inh_3d_plan *ths)

5.2.1 Detailed Description

5.2.2 Function Documentation

5.2.2.1 mri_inh_2d1d_trafo()

```
mri_inh_2d1d_trafo (
            mri_inh_2d1d_plan * ths )
```

Executes a mri transformation considering the field inhomogeneity with the 2d1d method, i.e. computes for $j = 0, ..., M_{total} - 1$

$$f(x_j) = \sum_{k \in I_N^2} \hat{f}(k) \mathrm{e}^{\mathrm{i} t_j \omega(k)} \mathrm{e}^{-2\pi \mathrm{i} k x_j}$$

- ths_plan The plan

Author

Tobias Knopp

Definition at line 57 of file mri.c.

Referenced by mri_inh_2d1d_init_guru().

5.2.2.2 mri_inh_2d1d_init_guru()

```
void mri_inh_2d1d_init_guru (
             mri_inh_2d1d_plan * ths,
             int * N,
             int M,
             int * n,
             int m,
             double sigma,
             unsigned nfft_flags,
             unsigned fftw_flags )
```

Creates a transform plan.

- ths_plan The plan for the transform
- N The bandwidth N
- M_total The number of nodes x
- n The oversampled bandwidth N
- m The cut-off parameter
- sigma The oversampling factor
- nnfft_flags The flags

Author

Tobias Knopp

Definition at line 156 of file mri.c.

References mri_inh_2d1d_trafo().

5.2.2.3 mri_inh_2d1d_finalize()

```
mri_inh_2d1d_finalize (
             mri_inh_2d1d_plan * ths )
```

Destroys a plan.

- ths_plan The plan

Author

Tobias Knopp

Definition at line 174 of file mri.c.

5.2.2.4 mri_inh_3d_trafo()

```
mri_inh_3d_trafo (
            mri_inh_3d_plan * ths )
```

Executes a mri transformation considering the field inhomogeneity with the 3d method, i.e. computes for $j = 0, ..., M_{total} - 1$

$$f(x_j) = \sum_{k \in I_N^2} \hat{f}(k) \mathrm{e}^{\mathrm{i} t_j \omega(k)} \mathrm{e}^{-2\pi \mathrm{i} k x_j}$$

- ths_plan The plan

Author

Tobias Knopp

Definition at line 189 of file mri.c.

5.2.2.5 mri_inh_3d_adjoint()

```
mri_inh_3d_adjoint (
            mri_inh_3d_plan * ths )
```

Executes an adjoint mri transformation considering the field inhomogeneity with the 3d method, i.e. computes for $k \in I_N^2$

$$\hat{f}(k) = \sum_{j=0}^{M_{total}-1} f(x_j) \mathrm{e}^{\mathrm{i} t_j \omega(k)} \mathrm{e}^{-2\pi \mathrm{i} k x_j}$$

- ths_plan The plan

Author

Tobias Knopp

Definition at line 221 of file mri.c.

5.2.2.6 mri_inh_3d_finalize()

```
mri_inh_3d_finalize (
            mri_inh_3d_plan * ths )
```

Destroys a plan.

- ths_plan The plan

Author

Tobias Knopp

Definition at line 266 of file mri.c.

5.3 NFCT - Nonequispaced fast cosine transform

Direct and fast computation of the discrete nonequispaced cosine transform.

Data Structures

- struct nfct_plan

5.3.1 Detailed Description

Direct and fast computation of the discrete nonequispaced cosine transform.

5.4 NFFT - Nonequispaced fast Fourier transform

Direct and fast computation of the nonequispaced discrete Fourier transform.

Data Structures

- struct nfft_plan

Macros

- #define PRE_PHI_HUT (1U<<0)
- #define FG_PSI (1U<<1)
- #define PRE_LIN_PSI (1U<<2)
- #define PRE_FG_PSI (1U<<3)
- #define PRE_PSI (1U<<4)
- #define PRE_FULL_PSI (1U<<5)
- #define MALLOC_X (1U<<6)
- #define MALLOC_F_HAT (1U<<7)
- #define MALLOC_F (1U<<8)
- #define FFT_OUT_OF_PLACE (1U<<9)
- #define FFTW_INIT (1U<<10)
- #define PRE_ONE_PSI (PRE_LIN_PSI| PRE_FG_PSI| PRE_PSI| PRE_FULL_PSI)

5.4.1 Detailed Description

Direct and fast computation of the nonequispaced discrete Fourier transform.

This module implements the nonequispaced fast Fourier transforms. In the following, we abbreviate the term "nonequispaced fast Fourier transform" by NFFT.

We introduce our notation and nomenclature for discrete Fourier transforms. Let the torus

$$\mathbb{T}^d := \left\{ \mathbf{x} = (x_t)_{t=0,\dots,d-1} \in \mathbb{R}^d : \; -\frac{1}{2} \leq x_t < \frac{1}{2}, \; t = 0, \dots, d-1 \right\}$$

of dimension d be given. It will serve as domain from which the nonequispaced nodes $\mathbf{x}$ are taken. The sampling set is given by $\mathcal{X} := \{\mathbf{x}_j \in \mathbb{T}^d : \; j = 0, \dots, M-1\}$. Possible frequencies $\mathbf{k}$ are collected in the multi index set

$$I_{\mathbf{N}} := \left\{ \mathbf{k} = (k_t)_{t=0,\dots,d-1} \in \mathbb{Z}^d : -\frac{N_t}{2} \leq k_t < \frac{N_t}{2}, \; t = 0, \dots, d-1 \right\}.$$

Our concern is the computation of the *nonequispaced* discrete Fourier transform (NDFT)

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

The corresponding adjoint NDFT is the computation of

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

Direct implementations are given by nfft_direct_trafo and nfft_direct_adjoint taking $\mathcal{O}(|I_{\mathbf{N}}| M)$ floating point operations. Approximative realisations take only $\mathcal{O}(|I_{\mathbf{N}}| \log |I_{\mathbf{N}}| + M)$ floating point operations. These are provided by nfft_trafo and nfft_adjoint, respectively.

5.4.2 Macro Definition Documentation

5.4.2.1 PRE_PHI_HUT

```
#define PRE_PHI_HUT (1U<<0)
```

If this flag is set, the deconvolution step (the multiplication with the diagonal matrix $\mathbf{D}$) uses precomputed values of the Fourier transformed window function.

See also

nfft_init

nfft_init_advanced

nfft_init_guru

Author

Stefan Kunis

Definition at line 181 of file nfft3.h.

5.4.2.2 FG_PSI

```
#define FG_PSI (1U<<1)
```

If this flag is set, the convolution step (the multiplication with the sparse matrix $\mathbf{B}$) uses particular properties of the Gaussian window function to trade multiplications for direct calls to exponential function.

See also

nfft_init

nfft_init_advanced

nfft_init_guru

Author

Stefan Kunis

Definition at line 182 of file nfft3.h.

5.4.2.3 PRE_LIN_PSI

```
#define PRE_LIN_PSI (1U<<2)
```

If this flag is set, the convolution step (the multiplication with the sparse matrix $\mathbf{B}$) uses linear interpolation from a lookup table of equispaced samples of the window function instead of exact values of the window function. At the moment a table of size $(K+1)d$ is used, where $K = 2^{10}(m+1)$. An estimate for the size of the lookup table with respect to the target accuracy should be implemented.

See also

nfft_init

nfft_init_advanced

nfft_init_guru

Author

Stefan Kunis

Definition at line 183 of file nfft3.h.

5.4.2.4 PRE_FG_PSI

```
#define PRE_FG_PSI (1U<<3)
```

If this flag is set, the convolution step (the multiplication with the sparse matrix $\mathbf{B}$) uses particular properties of the Gaussian window function to trade multiplications for direct calls to exponential function (the remaining $2dM$ direct calls are precomputed).

See also

nfft_init

nfft_init_advanced

nfft_init_guru

Author

Stefan Kunis

Definition at line 184 of file nfft3.h.

5.4.2.5 PRE_PSI

```
#define PRE_PSI (1U<<4)
```

If this flag is set, the convolution step (the multiplication with the sparse matrix $\mathbf{B}$) uses $(2m+2)dM$ precomputed values of the window function.

See also

nfft_init

nfft_init_advanced

nfft_init_guru

Author

Stefan Kunis

Definition at line 185 of file nfft3.h.

5.4.2.6 PRE_FULL_PSI

```
#define PRE_FULL_PSI (1U<<5)
```

If this flag is set, the convolution step (the multiplication with the sparse matrix $\mathbf{B}$) uses $(2m+2)^dM$ precomputed values of the window function, in addition indices of source and target vectors are stored.

See also

nfft_init

nfft_init_advanced

nfft_init_guru

Author

Stefan Kunis

Definition at line 186 of file nfft3.h.

5.4.2.7 MALLOC_X

```
#define MALLOC_X (1U<<6)
```

If this flag is set, (de)allocation of the node vector is done.

See also

nfft_init

nfft_init_advanced

nfft_init_guru

nfft_finalize

Author

Stefan Kunis

Definition at line 187 of file nfft3.h.

5.4.2.8 MALLOC_F_HAT

```
#define MALLOC_F_HAT (1U<<7)
```

If this flag is set, (de)allocation of the vector of Fourier coefficients is done.

See also

nfft_init

nfft_init_advanced

nfft_init_guru

nfft_finalize

Author

Stefan Kunis

Definition at line 188 of file nfft3.h.

5.4.2.9 MALLOC_F

```
#define MALLOC_F (1U<<8)
```

If this flag is set, (de)allocation of the vector of samples is done.

See also

nfft_init

nfft_init_advanced

nfft_init_guru

nfft_finalize

Author

Stefan Kunis

Definition at line 189 of file nfft3.h.

5.4.2.10 FFT_OUT_OF_PLACE

```
#define FFT_OUT_OF_PLACE (1U<<9)
```

If this flag is set, FFTW uses disjoint input/output vectors.

See also

nfft_init

nfft_init_advanced

nfft_init_guru

nfft_finalize

Author

Stefan Kunis

Definition at line 190 of file nfft3.h.

5.4.2.11 FFTW_INIT

```
#define FFTW_INIT (1U<<10)
```

If this flag is set, fftw_init/fftw_finalize is called.

See also

nfft_init

nfft_init_advanced

nfft_init_guru

nfft_finalize

Author

Stefan Kunis

Definition at line 191 of file nfft3.h.

5.4.2.12 PRE_ONE_PSI

```
#define PRE_ONE_PSI (PRE_LIN_PSI| PRE_FG_PSI| PRE_PSI| PRE_FULL_PSI)
```

Summarises if precomputation is used within the convolution step (the multiplication with the sparse matrix $\mathbf{B}$). If testing against this flag is positive, nfft_precompute_one_psi has to be called.

See also

nfft_init

nfft_init_advanced

nfft_init_guru

nfft_precompute_one_psi

nfft_finalize

Author

Stefan Kunis

Definition at line 194 of file nfft3.h.

5.5 NFSFT - Nonequispaced fast spherical Fourier transform

This module implements nonuniform fast spherical Fourier transforms.

Data Structures

- struct nfsft_wisdom

  Wisdom structure.
- struct nfsft_plan

Macros

- #define NFSFT_NORMALIZED (1U << 0)
- #define NFSFT_USE_NDFT (1U << 1)
- #define NFSFT_USE_DPT (1U << 2)
- #define NFSFT_MALLOC_X (1U << 3)
- #define NFSFT_MALLOC_F_HAT (1U << 5)
- #define NFSFT_MALLOC_F (1U << 6)
- #define NFSFT_PRESERVE_F_HAT (1U << 7)
- #define NFSFT_PRESERVE_X (1U << 8)
- #define NFSFT_PRESERVE_F (1U << 9)
- #define NFSFT_DESTROY_F_HAT (1U << 10)
- #define NFSFT_DESTROY_X (1U << 11)
- #define NFSFT_DESTROY_F (1U << 12)
- #define NFSFT_NO_DIRECT_ALGORITHM (1U << 13)
- #define NFSFT_NO_FAST_ALGORITHM (1U << 14)
- #define NFSFT_ZERO_F_HAT (1U << 16)
- #define NFSFT_EQUISPACED (1U << 17)
- #define NFSFT_INDEX(k, n, plan) ((2∗(plan)->N+2)∗((plan)->N-n+1)+(plan)->N+k+1)
- #define NFSFT_F_HAT_SIZE(N) ((2∗N+2)∗(2∗N+2))
- #define **BWEXP_MAX** 10
- #define **BW_MAX** 1024
- #define **ROW**(k) (k∗(wisdom.N_MAX+2))
- #define **ROWK**(k) (k∗(wisdom.N_MAX+2)+k)
- #define NFSFT_DEFAULT_NFFT_CUTOFF 6

  The default NFFT cutoff parameter.
- #define NFSFT_DEFAULT_THRESHOLD 1000

  The default threshold for the FPT.
- #define NFSFT_BREAK_EVEN 5

  The break-even bandwidth $N \in \mathbb{N}_0$.

Enumerations

- enum **bool** { **false** = 0, **true** = 1 }

Functions

- void **alpha_al_row** (R *alpha, const int N, const int n)
- void **beta_al_row** (R *beta, const int N, const int n)
- void **gamma_al_row** (R *gamma, const int N, const int n)
- void alpha_al_all (R *alpha, const int N)

  Compute three-term-recurrence coefficients α_{k-1}^n of associated Legendre functions for $k, n = 0, 1, \ldots, N$.
- void beta_al_all (R *beta, const int N)

  Compute three-term-recurrence coefficients β_{k-1}^n of associated Legendre functions for $k, n = 0, 1, \ldots, N$.
- void gamma_al_all (R *gamma, const int N)

  Compute three-term-recurrence coefficients γ_{k-1}^n of associated Legendre functions for $k, n = 0, 1, \ldots, N$.
- void eval_al (R *x, R *y, const int size, const int k, R *alpha, R *beta, R *gamma)

  Evaluates an associated Legendre polynomials $P_k^n(x, c)$ using the Clenshaw-algorithm.
- int eval_al_thresh (R *x, R *y, const int size, const int k, R *alpha, R *beta, R *gamma, R threshold)

  Evaluates an associated Legendre polynomials $P_k^n(x, c)$ using the Clenshaw-algorithm if it no exceeds a given threshold.
- static void c2e (nfsft_plan *plan)

  Converts coefficients $(b_k^n)_{k=0}^M$ with $M \in \mathbb{N}_0$, $-M \leq n \leq M$ from a linear combination of Chebyshev polynomials.
- static void c2e_transposed (nfsft_plan *plan)

  Transposed version of the function c2e.
- void nfsft_init (nfsft_plan *plan, int N, int M)
- void nfsft_init_advanced (nfsft_plan *plan, int N, int M, unsigned int flags)
- void **nfsft_init_guru** (nfsft_plan *plan, int N, int M, unsigned int flags, unsigned int nfft_flags, int nfft_cutoff)
- void nfsft_precompute (int N, double kappa, unsigned int nfsft_flags, unsigned int fpt_flags)
- void nfsft_forget (void)
- void nfsft_finalize (nfsft_plan *plan)
- static void **nfsft_set_f_nan** (nfsft_plan *plan)
- void **nfsft_trafo_direct** (nfsft_plan *plan)
- static void **nfsft_set_f_hat_nan** (nfsft_plan *plan)
- void **nfsft_adjoint_direct** (nfsft_plan *plan)
- void nfsft_trafo (nfsft_plan *plan)
- void nfsft_adjoint (nfsft_plan *plan)
- void **nfsft_precompute_x** (nfsft_plan *plan)

Variables

- bool nfsft_wisdom::initialized

  Indicates wether the structure has been initialized.
- unsigned int **nfsft_wisdom::flags**
- int nfsft_wisdom::N_MAX

  Stores precomputation flags.
- int nfsft_wisdom::T_MAX

  The logarithm /f$t = \log_2 N_{\text{max}}/f$ of the maximum bandwidth.
- double * nfsft_wisdom::alpha

  Precomputed recursion coefficients /fα_k^n/f for /f$k = 0,/ldots, N_{\text{max}}; n=-k,/ldots,k/f$ of associated Legendre-functions /fP_k^n/f.
- double * nfsft_wisdom::beta

  Precomputed recursion coefficients /fβ_k^n/f for /f$k = 0,/ldots, N_{\text{max}}; n=-k,/ldots,k/f$ of associated Legendre-functions /fP_k^n/f.
- double * nfsft_wisdom::gamma

  Precomputed recursion coefficients /fγ_k^n/f for /f$k = 0,/ldots, N_{\text{max}}; n=-k,/ldots,k/f$ of associated Legendre-functions /fP_k^n/f.

- double nfsft_wisdom::threshold

  The threshold /fκ/f.
- fpt_set nfsft_wisdom::set

  Structure for discrete polynomial transform (DPT)
- static struct nfsft_wisdom wisdom = {false,0U,-1,-1,0,0,0,0,0}

  The global wisdom structure for precomputed data.

5.5.1 Detailed Description

This module implements nonuniform fast spherical Fourier transforms.

In the following, we abbreviate the term "nonuniform fast spherical Fourier transform" by NFSFT.

5.5.2 Preliminaries

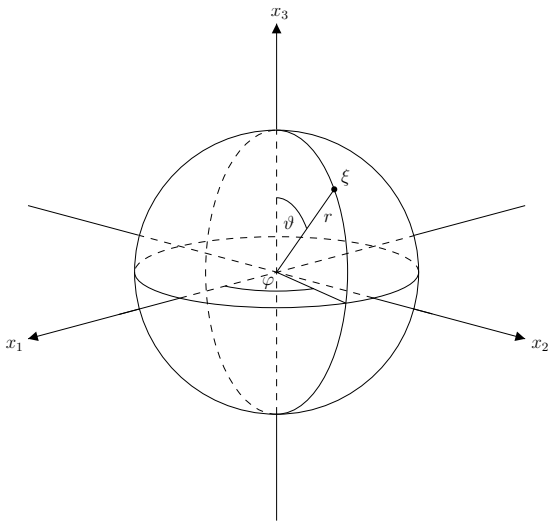
This section summarises basic definitions and properties related to spherical Fourier transforms.

5.5.2.1 Spherical Coordinates

Every point in $\mathbb{R}^3$ can be described in *spherical coordinates* by a vector $(r, \vartheta, \varphi)^{\mathrm{T}}$ with the radius $r \in \mathbb{R}^+$ and two angles $\vartheta \in [0, \pi]$, $\varphi \in [-\pi, \pi)$. We denote by $\mathbb{S}^2$ the two-dimensional unit sphere embedded into $\mathbb{R}^3$, i.e.

$$\mathbb{S}^2 := \left\{ \mathbf{x} \in \mathbb{R}^3 : \ \|\mathbf{x}\|_2 = 1 \right\}$$

and identify a point from $\mathbb{S}^2$ with the corresponding vector $(\vartheta, \varphi)^{\mathrm{T}}$. The spherical coordinate system is illustrated in the following figure:



For consistency with the other modules and the conventions used there, we also use *swapped scaled spherical coordinates* $x_1 := \frac{\varphi}{2\pi}$, $x_2 := \frac{\vartheta}{2\pi}$ and identify a point from $\mathbb{S}^2$ with the vector $\mathbf{x} := (x_1, x_2) \in [-\frac{1}{2}, \frac{1}{2}) \times [0, \frac{1}{2}]$.

5.5.2.2 Legendre Polynomials

The *Legendre polynomials* $P_k : [-1,1] \to \mathbb{R}$, $k \in \mathbb{N}_0$ as *classical orthogonal polynomials* are given by their corresponding *Rodrigues formula*

$$P_k(t) := \frac{1}{2^k k!} \frac{\mathsf{d}^k}{\mathsf{d}t^k} \left(t^2 - 1\right)^k.$$

The corresponding three-term recurrence relation is

$$(k+1)P_{k+1}(t) = (2k+1)xP_k(t) - kP_{k-1}(t) \quad (k \in \mathbb{N}_0).$$

With

$$\langle f, g \rangle_{\mathsf{L}^2([-1,1])} := \int_{-1}^{1} f(t)g(t)\mathsf{d}t$$

being the usual $\mathsf{L}^2([-1,1])$ inner product, the Legendre polynomials obey the orthogonality condition

$$\langle P_k, P_l \rangle_{\mathsf{L}^2([-1,1])} = \frac{2}{2k+1}\delta_{k,l}.$$

Remarks

The normalisation constant $c_k := \sqrt{\frac{2k+1}{2}}$ renders the scaled Legendre polynomials $c_k P_k$ orthonormal with respect to the induced $\mathsf{L}^2([-1,1])$ norm

$$\|f\|_{\mathsf{L}^2([-1,1])} := \left(< f, f >_{\mathsf{L}^2([-1,1])}\right)^{1/2} = \left(\int_{-1}^{1} |f(t)|^2 \, \mathsf{d}t\right)^{1/2}.$$

5.5.2.3 Associated Legendre Functions

The *associated Legendre functions* $P_k^n : [-1,1] \to \mathbb{R}$, $n \in \mathbb{N}_0$, $k \geq n$ are defined by

$$P_k^n(t) := \left(\frac{(k-n)!}{(k+n)!}\right)^{1/2} \left(1 - t^2\right)^{n/2} \frac{\mathsf{d}^n}{\mathsf{d}t^n} P_k(t).$$

For $n = 0$, they coincide with the Legendre polynomials, i.e. $P_k^0 = P_k$. The associated Legendre functions obey the three-term recurrence relation

$$P_{k+1}^n(t) = v_k^n t P_k^n(t) + w_k^n P_{k-1}^n(t) \quad (k \geq n),$$

with $P_{n-1}^n(t) := 0$, $P_n^n(t) := \frac{\sqrt{(2n)!}}{2^n n!}\left(1 - t^2\right)^{n/2}$, and

$$v_k^n := \frac{2k+1}{((k-n+1)(k+n+1))^{1/2}}, \qquad w_k^n := -\frac{((k-n)(k+n))^{1/2}}{((k-n+1)(k+n+1))^{1/2}}.$$

For fixed n, the set $\{P_k^n : k \geq n\}$ forms a complete set of orthogonal functions in $\mathsf{L}^2([-1,1])$ with

$$\langle P_k^n, P_l^n \rangle_{\mathsf{L}^2([-1,1])} = \frac{2}{2k+1}\delta_{k,l} \quad (0 \leq n \leq k, l).$$

Remarks

The normalisation constant $c_k = \sqrt{\frac{2k+1}{2}}$ renders the scaled associated Legendre functions $c_k P_k^n$ orthonormal with respect to the induced $\mathsf{L}^2([-1,1])$ norm

$$\|f\|_{\mathsf{L}^2([-1,1])} := \left(< f, f >_{\mathsf{L}^2([-1,1])}\right)^{1/2} = \left(\int_{-1}^{1} |f(t)|^2 \, \mathsf{d}t\right)^{1/2}.$$

5.5.2.4 Spherical Harmonics

The standard orthogonal basis of spherical harmonics for $\mathrm{L}^2\left(\mathbb{S}^2\right)$ with yet unnormalised basis functions $\tilde{Y}_k^n : \mathbb{S}^2 \to \mathbb{C}$ is given by

$$\tilde{Y}_k^n(\vartheta,\varphi) := P_k^{|n|}(\cos\vartheta)\mathrm{e}^{\mathrm{i}n\varphi}$$

with the usual $\mathrm{L}^2\left(\mathbb{S}^2\right)$ inner product

$$\langle f,g\rangle_{\mathrm{L}^2(\mathbb{S}^2)} := \int_{\mathbb{S}^2} f(\vartheta,\varphi)\overline{g(\vartheta,\varphi)}\,\mathrm{d}\xi := \int_{-\pi}^{\pi}\int_0^{\pi} f(\vartheta,\varphi)\overline{g(\vartheta,\varphi)}\sin\vartheta\,\mathrm{d}\vartheta\,\mathrm{d}\varphi.$$

The normalisation constant $c_k^n := \sqrt{\frac{2k+1}{4\pi}}$ renders the scaled basis functions

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

orthonormal with respect to the induced $\mathrm{L}^2\left(\mathbb{S}^2\right)$ norm

$$\|f\|_{\mathrm{L}^2(\mathbb{S}^2)} = \left(< f,f >_{\mathrm{L}^2(\mathbb{S}^2)}\right)^{1/2} = \left(\int_{-\pi}^{\pi}\int_0^{\pi} |f(\vartheta,\varphi)|^2\sin\vartheta\,\mathrm{d}\vartheta\,\mathrm{d}\varphi\right)^{1/2}.$$

A function $f \in \mathrm{L}^2\left(\mathbb{S}^2\right)$ has the orthogonal expansion

$$f = \sum_{k=0}^{\infty}\sum_{n=-k}^{k} \hat{f}(k,n)Y_k^n,$$

where the coefficients $\hat{f}(k,n) := \langle f, Y_k^n\rangle_{\mathrm{L}^2(\mathbb{S}^2)}$ are the *spherical Fourier coefficients* and the equivalence is understood in the L^2-sense.

5.5.3 Nonuniform Fast Spherical Fourier Transforms

This section describes the input and output relation of the spherical Fourier transform algorithms and the layout of the corresponding plan structure.

5.5.3.1 Nonuniform Discrete Spherical Fourier Transform

The *nonuniform discrete spherical Fourier transform* (*NDSFT*) is defined as follows:

Input : coefficients $\hat{f}(k,n) \in \mathbb{C}$ for $k = 0,\ldots,N,\ n = -k,\ldots,k,\ N \in \mathbb{N}_0$,
arbitrary nodes $\mathbf{x}(m) \in [-\frac{1}{2},\frac{1}{2}] \times [0,\frac{1}{2}]$ for $m = 0,\ldots,M-1, M \in \mathbb{N}$.

Task : evaluate $f(m) := f\left(\mathbf{x}(m)\right) = \sum_{k=0}^{N}\sum_{n=-k}^{k} \hat{f}_k^n Y_k^n\left(\mathbf{x}(m)\right)$ for $m = 0,\ldots,M-1$.

Output : coefficients $f(m) \in \mathbb{C}$ for $m = 0,\ldots,M-1$.

5.5.3.2 Adjoint Nonuniform Discrete Spherical Fourier Transform

The *adjoint nonuniform discrete spherical Fourier transform* (*adjoint NDSFT*) is defined as follows:

Input : coefficients $f(m) \in \mathbb{C}$ for $m = 0,\ldots,M-1, M \in \mathbb{N}$,
arbitrary nodes $\mathbf{x}(m) \in [-\frac{1}{2},\frac{1}{2}] \times [0,\frac{1}{2}]$ for $m = 0,\ldots,M-1, N \in \mathbb{N}_0$.

Task : evaluate $\hat{f}(k,n) := \sum_{m=0}^{M-1} f(m)\overline{Y_k^n\left(\mathbf{x}(m)\right)}cDo$ for $k = 0,\ldots,N,\ n = -k,\ldots,k$.

Output : coefficients $\hat{f}(k,n) \in \mathbb{C}$ for $k = 0,\ldots,N,\ n = -k,\ldots,k$.

5.5.3.3 Data Layout

This section describes the public layout of the nfsft_plan structure which contains all data for the computation of the aforementioned spherical Fourier transforms. The structure contains private (no read or write allowed), public read-only (only read access permitted), and public read-write (read and write access allowed) members. In the following, we indicate read and write access by `read` and `write`. The public members are structured as follows:

- `N_total` (`read`) The total number of components in `f_hat`. If the bandwidth is $N \in \mathbb{N}_0$, the total number of components in `f_hat` is `N_total` $= (2N+2)^2$.
- `M_total` (`read`) the total number of samples M
- `f_hat` (`read-write`) The flattened array of spherical Fourier coefficents. The array has length $(2N+2)^2$ such that valid indices $i \in \mathbb{N}_0$ for array access `f_hat [` i `]` are $i = 0, 1, \ldots, (2N+2)^2 - 1$. However, only read and write access to indices corresponding to spherical Fourier coefficients $\hat{f}(k,n)$ is defined. The index i corresponding to the spherical Fourier coefficient $\hat{f}(k,n)$ with $0 \leq k \leq M$, $-k \leq n \leq k$ is $i = (N+2)(N-n+1) + N + k + 1$. For convenience, the helper macro NFSFT_INDEX(k,n) provides the necessary index calculations such that one can write `f_hat[ NFSFT_INDEX(` k, n`)] = ...` to access the component corresponding to $\hat{f}(k,n)$. The data layout is due to implementation details.
- `f` (`read-write`) the array of coefficients $f(m)$ for $m = 0, \ldots, M-1$ such that `f[` m`]` = $f(m)$
- `N` (`read`) the bandwidth $N \in \mathbb{N}_0$
- `x` the array of nodes $\mathbf{x}(m) \in [-\frac{1}{2}, \frac{1}{2}] \times [0, \frac{1}{2}]$ for $m = 0, \ldots, M-1$ such that `f[` $2m$`]` = x_1 and `f[` $2m+1$`]` = x_2

5.5.3.4 Good to know...

When using the routines of this module you should bear in mind the following:

- The bandwidth N_{max} up to which precomputation is performed is always chosen as the next power of two with respect to the specified maximum bandwidth.
- By default, the NDSFT transforms (see nfsft_direct_trafo, nfsft_trafo) are allowed to destroy the input `f_hat` while the input `x` is preserved. On the contrary, the adjoint NDSFT transforms (see nfsft_direct_adjoint, nfsft_adjoint) do not destroy the input `f` and `x` by default. The desired behaviour can be assured by using the NFSFT_PRESERVE_F_HAT, NFSFT_PRESERVE_X, NFSFT_PRESERVE_F and NFSFT_DESTROY_F_HAT, NFSFT_DESTROY_X, NFSFT_DESTROY_F flags.

5.5.4 Macro Definition Documentation

5.5.4.1 NFSFT_NORMALIZED

```
#define NFSFT_NORMALIZED (1U << 0)
```

By default, all computations are performed with respect to the unnormalized basis functions

$$\tilde{Y}_k^n(\vartheta, \varphi) = P_k^{|n|}(\cos\vartheta)\mathrm{e}^{\mathrm{i}n\varphi}.$$

If this flag is set, all computations are carried out using the L_2- normalized basis functions

$$Y_k^n(\vartheta, \varphi) = \sqrt{\frac{2k+1}{4\pi}} P_k^{|n|}(\cos\vartheta)\mathrm{e}^{\mathrm{i}n\varphi}.$$

See also

nfsft_init

nfsft_init_advanced

nfsft_init_guru

Author

Jens Keiner

Definition at line 561 of file nfft3.h.

5.5.4.2 NFSFT_USE_NDFT

```
#define NFSFT_USE_NDFT (1U << 1)
```

If this flag is set, the fast NFSFT algorithms (see nfsft_trafo, nfsft_adjoint) will use internally the exact but usually slower direct NDFT algorithm in favor of fast but approximative NFFT algorithm.

See also

nfsft_init

nfsft_init_advanced

nfsft_init_guru

Author

Jens Keiner

Definition at line 562 of file nfft3.h.

5.5.4.3 NFSFT_USE_DPT

```
#define NFSFT_USE_DPT (1U << 2)
```

If this flag is set, the fast NFSFT algorithms (see nfsft_trafo, nfsft_adjoint) will use internally the usually slower direct DPT algorithm in favor of the fast FPT algorithm.

See also

nfsft_init

nfsft_init_advanced

nfsft_init_guru

Author

Jens Keiner

Definition at line 563 of file nfft3.h.

5.5.4.4 NFSFT_MALLOC_X

```
#define NFSFT_MALLOC_X (1U << 3)
```

If this flag is set, the init methods (see nfsft_init , nfsft_init_advanced , and nfsft_init_guru) will allocate memory and the method nfsft_finalize will free the array `x` for you. Otherwise, you have to assure by yourself that `x` points to an array of proper size before excuting a transform and you are responsible for freeing the corresponding memory before program termination.

See also

nfsft_init

nfsft_init_advanced

nfsft_init_guru

Author

Jens Keiner

Definition at line 564 of file nfft3.h.

5.5.4.5 NFSFT_MALLOC_F_HAT

```
#define NFSFT_MALLOC_F_HAT (1U << 5)
```

If this flag is set, the init methods (see nfsft_init , nfsft_init_advanced , and nfsft_init_guru) will allocate memory and the method nfsft_finalize will free the array `f_hat` for you. Otherwise, you have to assure by yourself that `f_hat` points to an array of proper size before excuting a transform and you are responsible for freeing the corresponding memory before program termination.

See also

nfsft_init

nfsft_init_advanced

nfsft_init_guru

Author

Jens Keiner

Definition at line 565 of file nfft3.h.

5.5.4.6 NFSFT_MALLOC_F

```
#define NFSFT_MALLOC_F (1U << 6)
```

If this flag is set, the init methods (see nfsft_init , nfsft_init_advanced , and nfsft_init_guru) will allocate memory and the method nfsft_finalize will free the array `f` for you. Otherwise, you have to assure by yourself that `f` points to an array of proper size before excuting a transform and you are responsible for freeing the corresponding memory before program termination.

See also

nfsft_init

nfsft_init_advanced

nfsft_init_guru

Author

Jens Keiner

Definition at line 566 of file nfft3.h.

5.5.4.7 NFSFT_PRESERVE_F_HAT

```
#define NFSFT_PRESERVE_F_HAT (1U << 7)
```

If this flag is set, it is guaranteed that during an execution of nfsft_direct_trafo or nfsft_trafo the content of `f_hat` remains unchanged.

See also

nfsft_init

nfsft_init_advanced

nfsft_init_guru

Author

Jens Keiner

Definition at line 567 of file nfft3.h.

5.5.4.8 NFSFT_PRESERVE_X

```
#define NFSFT_PRESERVE_X (1U << 8)
```

If this flag is set, it is guaranteed that during an execution of nfsft_direct_trafo, nfsft_trafo or nfsft_direct_adjoint, nfsft_adjoint the content of `x` remains unchanged.

See also

nfsft_init

nfsft_init_advanced

nfsft_init_guru

Author

Jens Keiner

Definition at line 568 of file nfft3.h.

5.5.4.9 NFSFT_PRESERVE_F

```
#define NFSFT_PRESERVE_F (1U << 9)
```

If this flag is set, it is guaranteed that during an execution of nfsft_direct_adjoint or nfsft_adjoint the content of `f` remains unchanged.

See also

nfsft_init

nfsft_init_advanced

nfsft_init_guru

Author

Jens Keiner

Definition at line 569 of file nfft3.h.

5.5.4.10 NFSFT_DESTROY_F_HAT

```
#define NFSFT_DESTROY_F_HAT (1U << 10)
```

If this flag is set, it is explicitely allowed that during an execution of nfsft_direct_trafo or nfsft_trafo the content of `f_hat` may be changed.

See also

nfsft_init

nfsft_init_advanced

nfsft_init_guru

Author

Jens Keiner

Definition at line 570 of file nfft3.h.

5.5.4.11 NFSFT_DESTROY_X

```
#define NFSFT_DESTROY_X (1U << 11)
```

If this flag is set, it is explicitely allowed that during an execution of nfsft_direct_trafo, nfsft_trafo or nfsft_direct_↩ adjoint, nfsft_adjoint the content of `x` may be changed.

See also

nfsft_init

nfsft_init_advanced

nfsft_init_guru

Author

Jens Keiner

Definition at line 571 of file nfft3.h.

5.5.4.12 NFSFT_DESTROY_F

```
#define NFSFT_DESTROY_F (1U << 12)
```

If this flag is set, it is explicitely allowed that during an execution of nfsft_direct_adjoint or nfsft_adjoint the content of `f` may be changed.

See also

nfsft_init

nfsft_init_advanced

nfsft_init_guru

Author

Jens Keiner

Definition at line 572 of file nfft3.h.

5.5.4.13 NFSFT_NO_DIRECT_ALGORITHM

```
#define NFSFT_NO_DIRECT_ALGORITHM (1U << 13)
```

If this flag is set, the transforms nfsft_direct_trafo and nfsft_direct_adjoint do not work. Setting this flag saves some memory for precomputed data.

See also

nfsft_precompute

nfsft_direct_trafo

nfsft_direct_adjoint

Author

Jens Keiner

Definition at line 576 of file nfft3.h.

5.5.4.14 NFSFT_NO_FAST_ALGORITHM

```
#define NFSFT_NO_FAST_ALGORITHM (1U << 14)
```

If this flag is set, the transforms nfsft_trafo and nfsft_adjoint do not work. Setting this flag saves memory for precomputed data.

See also

nfsft_precompute

nfsft_trafo

nfsft_adjoint

Author

Jens Keiner

Definition at line 577 of file nfft3.h.

5.5.4.15 NFSFT_ZERO_F_HAT

```
#define NFSFT_ZERO_F_HAT (1U << 16)
```

If this flag is set, the transforms nfsft_adjoint and nfsft_direct_adjoint set all unused entries in `f_hat` not corresponding to spherical Fourier coefficients to zero.

Author

Jens Keiner

Definition at line 578 of file nfft3.h.

5.5.4.16 NFSFT_EQUISPACED

```
#define NFSFT_EQUISPACED (1U << 17)
```

If this flag is set, we use the equispaced FFT instead of the NFFT. This implies that the nodes are fixed to

$$\varphi_i = 2\pi \frac{i}{2N+2}, \qquad i = -N-1, \ldots, N,$$

$$\vartheta_j = 2\pi \frac{j}{2N+2}, \qquad j = 0, \ldots, N.$$

Author

Michael Quellmalz

Definition at line 573 of file nfft3.h.

5.5.4.17 NFSFT_INDEX

```
#define NFSFT_INDEX(
              k,
              n,
              plan ) ((2*(plan)->N+2)*((plan)->N-n+1)+(plan)->N+k+1)
```

This helper macro expands to the index i corresponding to the spherical Fourier coefficient `f_hat`(k,n) for $0 \leq k \leq N$, $-k \leq n \leq k$ with

$$(N+2)(N-n+1)+N+k+1$$

Definition at line 581 of file nfft3.h.

5.5.4.18 NFSFT_F_HAT_SIZE

```
#define NFSFT_F_HAT_SIZE(
              N ) ((2*N+2)*(2*N+2))
```

This helper macro expands to the logical size of a spherical Fourier coefficients array for a bandwidth N.

Definition at line 582 of file nfft3.h.

5.5.4.19 NFSFT_DEFAULT_NFFT_CUTOFF

```
#define NFSFT_DEFAULT_NFFT_CUTOFF 6
```

The default NFFT cutoff parameter.

Author

Jens Keiner

Definition at line 65 of file nfsft.c.

5.5.4.20 NFSFT_DEFAULT_THRESHOLD

```
#define NFSFT_DEFAULT_THRESHOLD 1000
```

The default threshold for the FPT.

Author

Jens Keiner

Definition at line 72 of file nfsft.c.

5.5.4.21 NFSFT_BREAK_EVEN

```
#define NFSFT_BREAK_EVEN 5
```

The break-even bandwidth $N \in \mathbb{N}_0$.

Author

Jens Keiner

Definition at line 79 of file nfsft.c.

5.5.5 Function Documentation

5.5.5.1 alpha_al_all()

```
void alpha_al_all (
            R * alpha,
            const int N )  [inline]
```

Compute three-term-recurrence coefficients α_{k-1}^n of associated Legendre functions for $k, n = 0, 1, \ldots, N$.

- alpha A pointer to an array of doubles of size $(N + 1)^2$ where the coefficients will be stored such that alpha[n+(N+1)+k] = α_{k-1}^n.
- N The upper bound N.

Definition at line 89 of file legendre.c.

Referenced by nfsft_precompute().

5.5.5.2 beta_al_all()

```
void beta_al_all (
            R * beta,
            const int N )  [inline]
```

Compute three-term-recurrence coefficients β_{k-1}^n of associated Legendre functions for $k, n = 0, 1, \ldots, N$.

- beta A pointer to an array of doubles of size $(N + 1)^2$ where the coefficients will be stored such that beta[n+(N+1)+k] = β_{k-1}^n.
- N The upper bound N.

Definition at line 98 of file legendre.c.

Referenced by nfsft_precompute().

5.5.5.3 gamma_al_all()

```
void gamma_al_all (
            R * gamma,
            const int N )  [inline]
```

Compute three-term-recurrence coefficients γ_{k-1}^n of associated Legendre functions for $k, n = 0, 1, \ldots, N$.

- beta A pointer to an array of doubles of size $(N+1)^2$ where the coefficients will be stored such that gamma[n+(N+1)+k] = γ_{k-1}^n.
- N The upper bound N.

Definition at line 107 of file legendre.c.

Referenced by nfsft_precompute().

5.5.5.4 eval_al()

```
void eval_al (
            R * x,
            R * y,
            const int size,
            const int k,
            R * alpha,
            R * beta,
            R * gamma )
```

Evaluates an associated Legendre polynomials $P_k^n(x, c)$ using the Clenshaw-algorithm.

- x A pointer to an array of nodes where the function is to be evaluated
- y A pointer to an array where the function values are returned
- size The length of x and y
- k The index k
- alpha A pointer to an array containing the recurrence coefficients $\alpha_c^n, \ldots, \alpha_{c+k}^n$
- beta A pointer to an array containing the recurrence coefficients $\beta_c^n, \ldots, \beta_{c+k}^n$
- gamma A pointer to an array containing the recurrence coefficients $\gamma_c^n, \ldots, \gamma_{c+k}^n$

Definition at line 116 of file legendre.c.

5.5.5.5 eval_al_thresh()

```
int eval_al_thresh (
            R * x,
            R * y,
            const int size,
            const int k,
            R * alpha,
            R * beta,
            R * gamma,
            R threshold )
```

Evaluates an associated Legendre polynomials $P_k^n(x, c)$ using the Clenshaw-algorithm if it no exceeds a given threshold.

- x A pointer to an array of nodes where the function is to be evaluated
- y A pointer to an array where the function values are returned
- size The length of x and y
- k The index k
- alpha A pointer to an array containing the recurrence coefficients $\alpha_c^n, \ldots, \alpha_{c+k}^n$
- beta A pointer to an array containing the recurrence coefficients $\beta_c^n, \ldots, \beta_{c+k}^n$
- gamma A pointer to an array containing the recurrence coefficients $\gamma_c^n, \ldots, \gamma_{c+k}^n$
- threshold The threshold

Definition at line 161 of file legendre.c.

5.5.5.6 c2e()

```
static void c2e (
            nfsft_plan * plan )  [inline], [static]
```

Converts coefficients $(b_k^n)_{k=0}^M$ with $M \in \mathbb{N}_0$, $-M \leq n \leq M$ from a linear combination of Chebyshev polynomials.

$$f(\cos\vartheta) = \sum_{k=0}^{2\lfloor \frac{M}{2} \rfloor} a_k (\sin\vartheta)^{n \bmod 2} T_k(\cos\vartheta)$$

to coefficients $(c_k^n)_{k=0}^M$ matching the representation by complex exponentials

$$f(\cos\vartheta) = \sum_{k=-M}^{M} c_k \mathrm{e}^{\mathrm{i}k\vartheta}$$

for each order $n = -M, \ldots, M$.

- plan The nfsft_plan containing the coefficients $(b_k^n)_{k=0,\ldots,M;n=-M,\ldots,M}$

Remarks

The transformation is computed in place.

Author

Jens Keiner

Definition at line 115 of file nfsft.c.

References NFSFT_INDEX.

5.5.5.7 c2e_transposed()

```
static void c2e_transposed (
            nfsft_plan * plan )  [inline], [static]
```

Transposed version of the function c2e.

- plan The nfsft_plan containing the coefficients $(c_k^n)_{k=-M,\dots,M;n=-M,\dots,M}$

Remarks

The transformation is computed in place.

Author

Jens Keiner

Definition at line 193 of file nfsft.c.

References NFSFT_INDEX.

5.5.5.8 nfsft_init()

```
void nfsft_init (
            nfsft_plan * plan,
            int N,
            int M )
```

Creates a transform plan.

- plan a pointer to a nfsft_plan structure
- N the bandwidth $N \in \mathbb{N}_0$
- M the number of nodes $M \in \mathbb{N}$

Author

Jens Keiner

Definition at line 260 of file nfsft.c.

References nfsft_init_advanced(), NFSFT_MALLOC_F, NFSFT_MALLOC_F_HAT, and NFSFT_MALLOC_X.

5.5.5.9 nfsft_init_advanced()

```
void nfsft_init_advanced (
            nfsft_plan * plan,
            int N,
            int M,
            unsigned int nfsft_flags )
```

Creates a transform plan.

- plan a pointer to a

  `nfsft_plan`

  structure
- N the bandwidth $N \in \mathbb{N}_0$
- M the number of nodes $M \in \mathbb{N}$
- nfsft_flags the NFSFT flags

Author

Jens Keiner

Definition at line 267 of file nfsft.c.

Referenced by nfsft_init().

5.5.5.10 nfsft_precompute()

```
void nfsft_precompute (
            int N,
            double kappa,
            unsigned int nfsft_flags,
            unsigned int fpt_flags )
```

Performes precomputation up to the next power of two with respect to a given bandwidth $N \in \mathbb{N}_2$. The threshold parameter $\kappa \in \mathbb{R}^+$ determines the number of stabilization steps computed in the discrete polynomial transform and thereby its accuracy.

- N the bandwidth $N \in \mathbb{N}_0$
- threshold the threshold $\kappa \in \mathbb{R}^+$
- nfsft_precomputation_flags the NFSFT precomputation flags
- fpt_precomputation_flags the FPT precomputation flags

Author

Jens Keiner

Definition at line 376 of file nfsft.c.

References nfsft_wisdom::alpha, alpha_al_all(), nfsft_wisdom::beta, beta_al_all(), FPT_AL_SYMMETRY, fpt_↩ init(), FPT_PERSISTENT_DATA, nfsft_wisdom::gamma, gamma_al_all(), nfsft_wisdom::initialized, nfsft_wisdom↩ ::N_MAX, NFSFT_BREAK_EVEN, NFSFT_NO_DIRECT_ALGORITHM, NFSFT_NO_FAST_ALGORITHM, nfsft↩ _wisdom::T_MAX, wisdom, and X.

5.5.5.11 nfsft_forget()

```
void nfsft_forget (
            void  )
```

Forgets all precomputed data.

Author

Jens Keiner

Definition at line 579 of file nfsft.c.

References nfsft_wisdom::alpha, nfsft_wisdom::beta, nfsft_wisdom::gamma, nfsft_wisdom::initialized, nfsft_↩ wisdom::N_MAX, NFSFT_BREAK_EVEN, NFSFT_NO_DIRECT_ALGORITHM, NFSFT_NO_FAST_ALGORITHM, and wisdom.

5.5.5.12 nfsft_finalize()

```
void nfsft_finalize (
            nfsft_plan * plan )
```

Destroys a plan.

- plan the plan to be destroyed

Author

Jens Keiner

Definition at line 625 of file nfsft.c.

References NFSFT_EQUISPACED, NFSFT_MALLOC_F, NFSFT_MALLOC_F_HAT, NFSFT_MALLOC_X, NFS↩ FT_NO_FAST_ALGORITHM, and NFSFT_PRESERVE_F_HAT.

5.5.5.13 nfsft_trafo()

```
void nfsft_trafo (
            nfsft_plan * plan )
```

Executes a NFSFT, i.e. computes for $m = 0, \ldots, M-1$

$$f(m) = \sum_{k=0}^{N} \sum_{n=-k}^{k} \hat{f}(k,n) Y_k^n \left(2\pi x_1(m), 2\pi x_2(m)\right).$$

- plan the plan

Author

Jens Keiner

Definition at line 1068 of file nfsft.c.

References NFSFT_NO_FAST_ALGORITHM, and wisdom.

5.5.5.14 nfsft_adjoint()

```
void nfsft_adjoint (
            nfsft_plan * plan )
```

Executes an adjoint NFSFT, i.e. computes for $k = 0, \ldots, N; n = -k, \ldots, k$

$$\hat{f}(k,n) = \sum_{m=0}^{M-1} f(m) Y_k^n \left(2\pi x_1(m), 2\pi x_2(m)\right).$$

- plan the plan

Author

Jens Keiner

Definition at line 1316 of file nfsft.c.

References NFSFT_NO_FAST_ALGORITHM, and wisdom.

5.5.6 Variable Documentation

5.5.6.1 N_MAX

```
int nfsft_wisdom::N_MAX
```

Stores precomputation flags.

The maximum bandwidth /f$N_{\text{max}} \in \mathbb{N}_0/f$

Definition at line 63 of file kernel/nfsft/api.h.

Referenced by nfsft_forget(), and nfsft_precompute().

5.5.6.2 wisdom

```
struct nfsft_wisdom wisdom = {false,0U,-1,-1,0,0,0,0,0}  [static]
```

The global wisdom structure for precomputed data.

`wisdom.initialized` is set to `false` and `wisdom.flags` is set to `0U`.

Author

Jens Keiner

Definition at line 1 of file nfsft.c.

Referenced by nfsft_adjoint(), nfsft_forget(), nfsft_precompute(), and nfsft_trafo().

5.6 NFSOFT - Nonequispaced fast SO(3) Fourier transform

This module implements nonuniform fast SO(3) Fourier transforms.

Macros

- #define NFSOFT_NORMALIZED (1U << 0)
- #define NFSOFT_USE_NDFT (1U << 1)
- #define NFSOFT_USE_DPT (1U << 2)
- #define NFSOFT_MALLOC_X (1U << 3)
- #define NFSOFT_REPRESENT (1U << 4)
- #define NFSOFT_MALLOC_F_HAT (1U << 5)
- #define NFSOFT_MALLOC_F (1U << 6)
- #define NFSOFT_PRESERVE_F_HAT (1U << 7)
- #define NFSOFT_PRESERVE_X (1U << 8)
- #define NFSOFT_PRESERVE_F (1U << 9)
- #define NFSOFT_DESTROY_F_HAT (1U << 10)
- #define NFSOFT_DESTROY_X (1U << 11)
- #define NFSOFT_DESTROY_F (1U << 12)
- #define NFSOFT_NO_STABILIZATION (1U << 13)
- #define NFSOFT_CHOOSE_DPT (1U << 14)
- #define NFSOFT_SOFT (1U << 15)
- #define NFSOFT_ZERO_F_HAT (1U << 16)
- #define NFSOFT_INDEX(m, n, l, B) ((((l)+((B)+1))+(2*(B)+2)*(((n)+((B)+1))+(2*(B)+2)*((m)+((B)+1)))))
- #define NFSOFT_F_HAT_SIZE(B) (((B)+1)*(4*((B)+1)*((B)+1)-1)/3)

Functions

- void nfsoft_precompute (nfsoft_plan *plan3D)
- void nfsoft_init (nfsoft_plan *plan, int N, int M)
- void nfsoft_init_advanced (nfsoft_plan *plan, int N, int M, unsigned int nfsoft_flags)
- void nfsoft_init_guru (nfsoft_plan *plan, int B, int M, unsigned int nfsoft_flags, unsigned int nfft_flags, int nfft_cutoff, int fpt_kappa)
- void nfsoft_init_guru_advanced (nfsoft_plan *plan, int B, int M, unsigned int nfsoft_flags, unsigned int nfft_↩ flags, int nfft_cutoff, int fpt_kappa, int nn_oversampled)
- void nfsoft_trafo (nfsoft_plan *plan3D)
- void nfsoft_adjoint (nfsoft_plan *plan3D)
- void nfsoft_finalize (nfsoft_plan *plan)

5.6.1 Detailed Description

This module implements nonuniform fast SO(3) Fourier transforms.

In the following, we abbreviate the term "nonuniform fast SO(3) Fourier transform" by NFSOFT.

5.6.2 Macro Definition Documentation

5.6.2.1 NFSOFT_NORMALIZED

```
#define NFSOFT_NORMALIZED (1U << 0)
```

By default, all computations are performed with respect to the unnormalized basis functions

$$D^l_{mn}(\alpha,\beta,\gamma) = d^{mn}_l(\cos\beta)\mathrm{e}^{-\mathrm{i}m\alpha}\mathrm{e}^{-\mathrm{i}n\gamma}.$$

If this flag is set, all computations are carried out using the L_2- normalized basis functions

$$\tilde{D}^l_{mn}(\alpha,\beta,\gamma) = \sqrt{\frac{2l+1}{8\pi^2}}d^{mn}_l(\cos\beta)\mathrm{e}^{-\mathrm{i}m\alpha}\mathrm{e}^{-\mathrm{i}n\gamma}$$

See also

nfsoft_init

nfsoft_init_advanced

nfsoft_init_guru

Author

Antje Vollrath

Definition at line 673 of file nfft3.h.

5.6.2.2 NFSOFT_USE_NDFT

```
#define NFSOFT_USE_NDFT (1U << 1)
```

If this flag is set, the fast NFSOFT algorithms (see nfsoft_trafo, nfsoft_adjoint) will use internally the exact but usually slower direct NDFT algorithm in favor of fast but approximative NFFT algorithm.

See also

nfsoft_init

nfsoft_init_advanced

nfsoft_init_guru

Author

Antje Vollrath

Definition at line 674 of file nfft3.h.

5.6.2.3 NFSOFT_USE_DPT

```
#define NFSOFT_USE_DPT (1U << 2)
```

If this flag is set, the fast NFSOFT algorithms (see nfsoft_trafo, nfsoft_adjoint) will use internally the usually slower direct DPT algorithm in favor of the fast FPT algorithm.

See also

nfsoft_init

nfsoft_init_advanced

nfsoft_init_guru

Author

Antje Vollrath

Definition at line 675 of file nfft3.h.

5.6.2.4 NFSOFT_MALLOC_X

```
#define NFSOFT_MALLOC_X (1U << 3)
```

If this flag is set, the init methods (see nfsoft_init , nfsoft_init_advanced , and nfsoft_init_guru) will allocate memory and the method nfsoft_finalize will free the array x for you. Otherwise, you have to assure by yourself that x points to an array of proper size before excuting a transform and you are responsible for freeing the corresponding memory before program termination.

See also

nfsoft_init

nfsoft_init_advanced

nfsoft_init_guru

Author

Antje Vollrath

Definition at line 676 of file nfft3.h.

5.6.2.5 NFSOFT_REPRESENT

```
#define NFSOFT_REPRESENT (1U << 4)
```

If this flag is set, the Wigner-D functions will be normed such that they satisfy the representation property of the spherical harmonics as defined in the NFFT software package, i.e. for every rotation matrix `A` with Euler angles α, β, γ and every unit vector `x` the Wigner-D functions will be normed such that

$$\sum_{m=-l}^{l} D_{mn}^{l}(\alpha,\beta,\gamma) Y_m^l(x) = Y_n^l(A^{-1}x)$$

Author

Antje Vollrath

Definition at line 677 of file nfft3.h.

5.6.2.6 NFSOFT_MALLOC_F_HAT

```
#define NFSOFT_MALLOC_F_HAT (1U << 5)
```

If this flag is set, the init methods (see nfsoft_init , nfsoft_init_advanced , and nfsoft_init_guru) will allocate memory and the method nfsoft_finalize will free the array `f_hat` for you. Otherwise, you have to assure by yourself that `f_hat` points to an array of proper size before excuting a transform and you are responsible for freeing the corresponding memory before program termination.

See also

nfsoft_init

nfsoft_init_advanced

nfsoft_init_guru

Author

Antje Vollrath

Definition at line 678 of file nfft3.h.

5.6.2.7 NFSOFT_MALLOC_F

```
#define NFSOFT_MALLOC_F (1U << 6)
```

If this flag is set, the init methods (see nfsoft_init , nfsoft_init_advanced , and nfsoft_init_guru) will allocate memory and the method nfsoft_finalize will free the array `f` for you. Otherwise, you have to assure by yourself that `f` points to an array of proper size before excuting a transform and you are responsible for freeing the corresponding memory before program termination.

See also

nfsoft_init

nfsoft_init_advanced

nfsoft_init_guru

Author

Antje Vollrath

Definition at line 679 of file nfft3.h.

5.6.2.8 NFSOFT_PRESERVE_F_HAT

```
#define NFSOFT_PRESERVE_F_HAT (1U << 7)
```

If this flag is set, it is guaranteed that during an execution of nfsoft_trafo the content of `f_hat` remains unchanged.

See also

nfsoft_init

nfsoft_init_advanced

nfsoft_init_guru

Author

Antje Vollrath

Definition at line 680 of file nfft3.h.

5.6.2.9 NFSOFT_PRESERVE_X

```
#define NFSOFT_PRESERVE_X (1U << 8)
```

If this flag is set, it is guaranteed that during an execution of nfsoft_trafo or nfsoft_adjoint the content of `x` remains unchanged.

See also

nfsoft_init

nfsoft_init_advanced

nfsoft_init_guru

Author

Antje Vollrath

Definition at line 681 of file nfft3.h.

5.6.2.10 NFSOFT_PRESERVE_F

```
#define NFSOFT_PRESERVE_F (1U << 9)
```

If this flag is set, it is guaranteed that during an execution of ndsoft_adjoint or nfsoft_adjoint the content of `f` remains unchanged.

See also

nfsoft_init

nfsoft_init_advanced

nfsoft_init_guru

Author

Antje Vollrath

Definition at line 682 of file nfft3.h.

5.6.2.11 NFSOFT_DESTROY_F_HAT

```
#define NFSOFT_DESTROY_F_HAT (1U << 10)
```

If this flag is set, it is explicitely allowed that during an execution of nfsoft_trafo the content of `f_hat` may be changed.

See also

nfsoft_init

nfsoft_init_advanced

nfsoft_init_guru

Author

Antje Vollrath

Definition at line 683 of file nfft3.h.

5.6.2.12 NFSOFT_DESTROY_X

```
#define NFSOFT_DESTROY_X (1U << 11)
```

If this flag is set, it is explicitely allowed that during an execution of nfsoft_trafo or nfsoft_adjoint the content of `x` may be changed.

See also

nfsoft_init

nfsoft_init_advanced

nfsoft_init_guru

Author

Antje Vollrath

Definition at line 684 of file nfft3.h.

5.6.2.13 NFSOFT_DESTROY_F

```
#define NFSOFT_DESTROY_F (1U << 12)
```

If this flag is set, it is explicitely allowed that during an execution of ndsoft_adjoint or nfsoft_adjoint the content of `f` may be changed.

See also

nfsoft_init

nfsoft_init_advanced

nfsoft_init_guru

Author

Antje Vollrath

Definition at line 685 of file nfft3.h.

5.6.2.14 NFSOFT_NO_STABILIZATION

```
#define NFSOFT_NO_STABILIZATION (1U << 13)
```

If this flag is set, the fast NFSOFT algorithms (see nfsoft_trafo, nfsoft_adjoint) will use internally the FPT algorithm without the stabilization scheme and thus making bigger errors for higher bandwidth but becoming significantly faster

Author

Antje Vollrath

Definition at line 688 of file nfft3.h.

5.6.2.15 NFSOFT_CHOOSE_DPT

```
#define NFSOFT_CHOOSE_DPT (1U << 14)
```

If this flag is set, the fast NFSOFT algorithms (see nfsoft_trafo, nfsoft_adjoint) will decide whether to use the DPT or FPT algorithm depending on which is faster for the chosen orders.

not yet included in the checked-in version

Author

Antje Vollrath

Definition at line 689 of file nfft3.h.

5.6.2.16 NFSOFT_SOFT

```
#define NFSOFT_SOFT (1U << 15)
```

If this flag is set, the fast NFSOFT algorithms (see nfsoft_trafo, nfsoft_adjoint) becomes a SOFT, i.e., we use equispaced nodes. The FFTW will be used instead of the NFFT.-->not included yet

See also

nfsoft_init

nfsoft_init_advanced

nfsoft_init_guru

Author

Antje Vollrath

Definition at line 690 of file nfft3.h.

5.6.2.17 NFSOFT_ZERO_F_HAT

```
#define NFSOFT_ZERO_F_HAT (1U << 16)
```

If this flag is set, the transform nfsoft_adjoint sets all unused entries in `f_hat` not corresponding to SO(3) Fourier coefficients to zero.

Author

Antje Vollrath

Definition at line 691 of file nfft3.h.

5.6.2.18 NFSOFT_INDEX

```
#define NFSOFT_INDEX(
              m,
              n,
              l,
              B ) (((l)+((B)+1))+(2*(B)+2)*(((n)+((B)+1))+(2*(B)+2)*((m)+((B)+1))))
```

This macro expands to the index i corresponding to the SO(3) Fourier coefficient $\hat{f}_l^{mn}$ for $l = 0, ..., B$, $m, n = -l, ..., l$ with

Definition at line 694 of file nfft3.h.

5.6.2.19 NFSOFT_F_HAT_SIZE

```
#define NFSOFT_F_HAT_SIZE(
              B ) (((B)+1)*(4*((B)+1)*((B)+1)-1)/3)
```

This macro expands to the logical size of a SO(3) Fourier coefficients array for a bandwidth B.

Definition at line 695 of file nfft3.h.

5.6.3 Function Documentation

5.6.3.1 nfsoft_precompute()

```
void nfsoft_precompute (
             nfsoft_plan * plan )
```

Does all node-dependent and node-independent precomputations needed for the NFSOFT.

- plan a pointer to a nfsoft_plan structure

Definition at line 382 of file nfsoft.c.

5.6.3.2 nfsoft_init()

```
void nfsoft_init (
             nfsoft_plan * plan,
             int N,
             int M )
```

Creates a NFSOFT transform plan.

- plan a pointer to a nfsoft_plan structure
- N the bandwidth $N \in \mathbb{N}_0$
- M the number of nodes $M \in \mathbb{N}$

Author

Antje Vollrath

Definition at line 45 of file nfsoft.c.

References nfsoft_init_advanced(), NFSOFT_MALLOC_F, NFSOFT_MALLOC_F_HAT, and NFSOFT_MALLOC_X.

5.6.3.3 nfsoft_init_advanced()

```
void nfsoft_init_advanced (
             nfsoft_plan * plan,
             int N,
             int M,
             unsigned int nfsoft_flags )
```

Creates a NFSOFT transform plan.

- plan a pointer to a nfsoft_plan structure
- N the bandwidth $N \in \mathbb{N}_0$
- M the number of nodes $M \in \mathbb{N}$
- nfsoft_flags the NFSOFT flags

Author

Antje Vollrath

Definition at line 51 of file nfsoft.c.

References MALLOC_X, nfsoft_init_guru(), PRE_PHI_HUT, and PRE_PSI.

Referenced by nfsoft_init().

5.6.3.4 nfsoft_init_guru()

```
void nfsoft_init_guru (
             nfsoft_plan * plan,
             int N,
             int M,
             unsigned int nfsoft_flags,
             unsigned int nfft_flags,
             int nfft_cutoff,
             int fpt_kappa )
```

Creates a NFSOFT transform plan.

- plan a pointer to a nfsoft_plan structure
- N the bandwidth $N \in \mathbb{N}_0$
- M the number of nodes $M \in \mathbb{N}$
- nfsoft_flags the NFSFT flags
- nfft_flags the NFFT flags
- fpt_kappa a parameter contolling the accuracy of the FPT
- nfft_cutoff the NFFT cutoff parameter

Author

Antje Vollrath

Definition at line 59 of file nfsoft.c.

References nfsoft_init_guru_advanced().

Referenced by nfsoft_init_advanced().

5.6.3.5 nfsoft_init_guru_advanced()

```
void nfsoft_init_guru_advanced (
             nfsoft_plan * plan,
             int N,
             int M,
             unsigned int nfsoft_flags,
             unsigned int nfft_flags,
             int nfft_cutoff,
             int fpt_kappa,
             int fftw_size )
```

Creates a NFSOFT transform plan.

- plan a pointer to a nfsoft_plan structure
- N the bandwidth $N \in \mathbb{N}_0$
- M the number of nodes $M \in \mathbb{N}$
- nfsoft_flags the NFSFT flags
- nfft_flags the NFFT flags
- fpt_kappa a parameter contolling the accuracy of the FPT
- nfft_cutoff the NFFT cutoff parameter
- fftw_size the size of the 3D FFTW transform inside the NFFT (fftw_size $= (2N + 2)\,\sigma$, where $\sigma \geq 1$ is the oversampling factor)

Definition at line 66 of file nfsoft.c.

References nfsoft_adjoint(), NFSOFT_MALLOC_F, NFSOFT_MALLOC_F_HAT, NFSOFT_MALLOC_X, nfsoft_↩trafo(), and PRE_LIN_PSI.

Referenced by nfsoft_init_guru().

5.6.3.6 nfsoft_trafo()

```
void nfsoft_trafo (
             nfsoft_plan * plan_nfsoft )
```

Executes a NFSOFT, i.e. computes for $m = 0, \ldots, M-1$

$$f(g_m) = \sum_{l=0}^{B} \sum_{m=-l}^{l} \sum_{n=-l}^{l} \hat{f}_l^{mn} D_l^{mn}\left(\alpha_m, \beta_m, \gamma_m\right).$$

- plan_nfsoft the plan

Author

Antje Vollrath

Definition at line 416 of file nfsoft.c.

References X.

Referenced by nfsoft_init_guru_advanced().

5.6.3.7 nfsoft_adjoint()

```
void nfsoft_adjoint (
            nfsoft_plan * plan_nfsoft )
```

Executes an adjoint NFSOFT, i.e. computes for $l = 0, \ldots, B; m, n = -l, \ldots, l$

$$\hat{f}_l^{mn} = \sum_{m=0}^{M-1} f(g_m) D_l^{mn}(\alpha_m, \beta_m, \gamma_m)$$

- plan_nfsoft the plan

Author

Antje Vollrath

Definition at line 545 of file nfsoft.c.

References NFSOFT_USE_NDFT, and X.

Referenced by nfsoft_init_guru_advanced().

5.6.3.8 nfsoft_finalize()

```
void nfsoft_finalize (
            nfsoft_plan * plan )
```

Destroys a plan.

- plan the plan to be destroyed

Author

Antje Vollrath

Definition at line 645 of file nfsoft.c.

5.7 NFST - Nonequispaced fast sine transform

Direct and fast computation of the discrete nonequispaced sine transform.

Data Structures

- struct nfst_plan

5.7.1 Detailed Description

Direct and fast computation of the discrete nonequispaced sine transform.

5.8 NNFFT - Nonequispaced in time and frequency FFT

Direct and fast computation of the discrete nonequispaced in time and frequency Fourier transform.

Data Structures

- struct nnfft_plan

Macros

- #define MALLOC_V (1U<< 11)

Functions

- void nnfft_init (nnfft_plan *ths, int d, int N_total, int M_total, int *N)
- void nnfft_init_guru (nnfft_plan *ths, int d, int N_total, int M_total, int *N, int *N1, int m, unsigned nnfft_flags)
- void nnfft_trafo (nnfft_plan *ths)

  user routines
- void nnfft_adjoint (nnfft_plan *ths)
- void nnfft_precompute_lin_psi (nnfft_plan *ths)

  create a lookup table
- void nnfft_precompute_psi (nnfft_plan *ths)
- void nnfft_precompute_full_psi (nnfft_plan *ths)

  computes all entries of B explicitly
- void nnfft_precompute_phi_hut (nnfft_plan *ths)

  initialisation of direct transform
- void nnfft_finalize (nnfft_plan *ths)

5.8.1 Detailed Description

Direct and fast computation of the discrete nonequispaced in time and frequency Fourier transform.

5.8.2 Macro Definition Documentation

5.8.2.1 MALLOC_V

```
#define MALLOC_V (1U<< 11)
```

If this flag is set, (de)allocation of the frequency node vector is done.

See also

nnfft_init

nnfft_init_guru

nnfft_finalize

Author

Tobias Knopp

Definition at line 414 of file nfft3.h.

5.8.3 Function Documentation

5.8.3.1 nnfft_init()

```
nnfft_init (
              nnfft_plan * ths_plan,
              int d,
              int N_total,
              int M_total,
              int * N )
```

Creates a transform plan.

- ths_plan The plan for the transform
- d The dimension
- N_total The number of nodes v
- M_total The number of nodes x
- N The bandwidth N

Author

Tobias Knopp

Definition at line 613 of file nnfft.c.

5.8.3.2 nnfft_init_guru()

```
void nnfft_init_guru (
              nnfft_plan * ths_plan,
              int d,
              int N_total,
              int M_total,
              int * N,
              int * N1,
              int m,
              unsigned nnfft_flags )
```

Creates a transform plan.

- ths_plan The plan for the transform
- d The dimension
- N_total The number of nodes v
- M_total The number of nodes x
- N The bandwidth N
- N1 The oversampled bandwidth N
- m The cut-off parameter
- nnfft_flags The flags

Author

Tobias Knopp

Definition at line 577 of file nnfft.c.

References FFT_OUT_OF_PLACE, FFTW_INIT, MALLOC_F_HAT, and PRE_PHI_HUT.

5.8.3.3 nnfft_trafo()

```
void nnfft_trafo (
            nnfft_plan * ths_plan )
```

user routines

Executes a NNFFT, i.e. computes for $j = 0, ..., M_{total} - 1$

$$f(x_j) = \sum_{k=0}^{N_{total}-1} \hat{f}(v_k) \mathrm{e}^{-2\pi \mathrm{i} v_k x_j \odot N}$$

- ths_plan The plan

Author

Tobias Knopp

Definition at line 291 of file nnfft.c.

5.8.3.4 nnfft_adjoint()

```
void nnfft_adjoint (
            nnfft_plan * ths_plan )
```

Executes a adjoint NNFFT, i.e. computes for $k = 0, ..., N_{total} - 1$

$$\hat{f}(v_k) = \sum_{j=0}^{M_{total}-1} f(x_j) \mathrm{e}^{2\pi \mathrm{i} v_k x_j \odot N}$$

- ths_plan The plan

Author

Tobias Knopp

Definition at line 319 of file nnfft.c.

5.8.3.5 nnfft_precompute_lin_psi()

```
void nnfft_precompute_lin_psi (
            nnfft_plan * ths_plan )
```

create a lookup table

Precomputation for a transform plan.

- ths_plan The pointer to a nfft plan

Author

Tobias Knopp

precomputes equally spaced values of the window function psi

if PRE_LIN_PSI is set the application program has to call this routine

Definition at line 367 of file nnfft.c.

5.8.3.6 nnfft_precompute_psi()

```
void nnfft_precompute_psi (
            nnfft_plan * ths_plan )
```

Precomputation for a transform plan.

- ths_plan The pointer to a nfft plan

Author

Tobias Knopp

precomputes the values of the window function psi in a tensor product form

if PRE_PSI is set the application program has to call this routine after setting the nodes x

Definition at line 385 of file nnfft.c.

Referenced by nnfft_precompute_full_psi().

5.8.3.7 nnfft_precompute_full_psi()

```
void nnfft_precompute_full_psi (
             nnfft_plan * ths_plan )
```

computes all entries of B explicitly

Precomputation for a transform plan.

- ths_plan The pointer to a nfft plan

Author

Tobias Knopp

precomputes the values of the window function psi and their indices in non tensor product form

if PRE_FULL_PSI is set the application program has to call this routine after setting the nodes x

Definition at line 424 of file nnfft.c.

References nnfft_precompute_psi().

5.8.3.8 nnfft_precompute_phi_hut()

```
void nnfft_precompute_phi_hut (
             nnfft_plan * ths_plan )
```

initialisation of direct transform

Precomputation for a transform plan.

- ths_plan The pointer to a nfft plan

Author

Tobias Knopp

precomputes the values of the fourier transformed window function, i.e. phi_hut

if PRE_PHI_HUT is set the application program has to call this routine after setting the nodes v

Definition at line 347 of file nnfft.c.

5.8.3.9 nnfft_finalize()

```
void nnfft_finalize (
             nnfft_plan * ths_plan )
```

Destroys a plan.

- ths_plan The plan

Author

Tobias Knopp

Definition at line 657 of file nnfft.c.

References MALLOC_F, MALLOC_F_HAT, MALLOC_V, MALLOC_X, PRE_FULL_PSI, PRE_LIN_PSI, PRE_PHI_HUT, and PRE_PSI.

5.9 NSFFT - Nonequispaced sparse FFT

Direct and fast computation of the nonequispaced FFT on the hyperbolic cross.

Data Structures

- struct nsfft_plan

Macros

- #define NSDFT (1U<< 12)

Functions

- void nsfft_trafo (nsfft_plan *ths)
- void nsfft_adjoint (nsfft_plan *ths)
- void nsfft_cp (nsfft_plan *ths, nfft_plan *ths_full_plan)
- void nsfft_init_random_nodes_coeffs (nsfft_plan *ths)
- void nsfft_init (nsfft_plan *ths, int d, int J, int M, int m, unsigned flags)
- void nsfft_finalize (nsfft_plan *ths)

5.9.1 Detailed Description

Direct and fast computation of the nonequispaced FFT on the hyperbolic cross.

5.9.2 Macro Definition Documentation

5.9.2.1 NSDFT

```
#define NSDFT (1U<< 12)
```

If this flag is set, the member index_sparse_to_full is (de)allocated and initialised for the use in the routine nsfft_↩ direct_trafo and nsdft_adjoint.

See also

nsfft_init

Author

Stefan Kunis

Definition at line 464 of file nfft3.h.

5.9.3 Function Documentation

5.9.3.1 nsfft_trafo()

```
void nsfft_trafo (
            nsfft_plan * ths )
```

Executes an NSFFT, computes **fast** and **approximate** for $j = 0, \ldots, M-1$:

$$f_j = \sum_{k \in H_N^d} \hat{f}_k \mathrm{e}^{-2\pi \mathrm{i} k x_j}$$

- ths The pointer to a nsfft plan

Author

Markus Fenn, Stefan Kunis

Definition at line 1541 of file nsfft.c.

5.9.3.2 nsfft_adjoint()

```
void nsfft_adjoint (
            nsfft_plan * ths )
```

Executes an adjoint NSFFT, computes **fast** and **approximate** for $k \in H_N^d$:

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

- ths The pointer to a nsfft plan

Author

Stefan Kunis

Definition at line 1549 of file nsfft.c.

5.9.3.3 nsfft_cp()

```
void nsfft_cp (
            nsfft_plan * ths,
            nfft_plan * ths_nfft )
```

Copy coefficients from nsfft plan to a nfft plan.

- ths Pointers to a nsfft plan and to a nfft plan

Author

Markus Fenn, Stefan Kunis

Definition at line 599 of file nsfft.c.

5.9.3.4 nsfft_init_random_nodes_coeffs()

```
void nsfft_init_random_nodes_coeffs (
            nsfft_plan * ths )
```

Initialisation of pseudo random nodes and coefficients.

- ths The pointer to a nsfft plan

Author

Markus Fenn, Stefan Kunis

Definition at line 723 of file nsfft.c.

5.9.3.5 nsfft_init()

```
void nsfft_init (
            nsfft_plan * ths,
            int d,
            int J,
            int M,
            int m,
            unsigned flags )
```

Initialisation of a transform plan.

- ths The pointer to a nsfft plan
- d The dimension
- J The problem size
- M The number of nodes
- m nfft cut-off parameter
- flags

Author

Markus Fenn, Stefan Kunis

Definition at line 1778 of file nsfft.c.

References UNUSED.

5.9.3.6 nsfft_finalize()

```
void nsfft_finalize (
            nsfft_plan * ths )
```

Destroys a transform plan.

- ths The pointer to a nsfft plan

Author

Markus Fenn, Stefan Kunis

Definition at line 1885 of file nsfft.c.

5.10 Solver - Inverse transforms

Macros

- #define LANDWEBER (1U<< 0)
- #define STEEPEST_DESCENT (1U<< 1)
- #define CGNR (1U<< 2)
- #define CGNE (1U<< 3)
- #define NORMS_FOR_LANDWEBER (1U<< 4)
- #define PRECOMPUTE_WEIGHT (1U<< 5)
- #define PRECOMPUTE_DAMP (1U<< 6)

5.10.1 Detailed Description

5.10.2 Macro Definition Documentation

5.10.2.1 LANDWEBER

```
#define LANDWEBER (1U<< 0)
```

If this flag is set, the Landweber (Richardson) iteration is used to compute an inverse transform.

Author

Stefan Kunis

Definition at line 773 of file nfft3.h.

5.10.2.2 STEEPEST_DESCENT

```
#define STEEPEST_DESCENT (1U<< 1)
```

If this flag is set, the method of steepest descent (gradient) is used to compute an inverse transform.

Author

Stefan Kunis

Definition at line 774 of file nfft3.h.

5.10.2.3 CGNR

```
#define CGNR (1U<< 2)
```

If this flag is set, the conjugate gradient method for the normal equation of first kind is used to compute an inverse transform. Each iterate minimises the residual in the current Krylov subspace.

Author

Stefan Kunis

Definition at line 775 of file nfft3.h.

5.10.2.4 CGNE

```
#define CGNE (1U<< 3)
```

If this flag is set, the conjugate gradient method for the normal equation of second kind is used to compute an inverse transform. Each iterate minimises the error in the current Krylov subspace.

Author

Stefan Kunis

Definition at line 776 of file nfft3.h.

5.10.2.5 NORMS_FOR_LANDWEBER

```
#define NORMS_FOR_LANDWEBER (1U<< 4)
```

If this flag is set, the Landweber iteration updates the member dot_r_iter.

Author

Stefan Kunis

Definition at line 777 of file nfft3.h.

5.10.2.6 PRECOMPUTE_WEIGHT

```
#define PRECOMPUTE_WEIGHT (1U<< 5)
```

If this flag is set, the samples are weighted, eg to cope with varying sampling density.

Author

Stefan Kunis

Definition at line 778 of file nfft3.h.

5.10.2.7 PRECOMPUTE_DAMP

```
#define PRECOMPUTE_DAMP (1U<< 6)
```

If this flag is set, the Fourier coefficients are damped, eg to favour fast decaying coefficients.

Author

Stefan Kunis

Definition at line 779 of file nfft3.h.

5.11 Util - Auxiliary functions

This module implements frequently used utility functions.

Macros

- #define **CONCAT**(prefix, name) prefix ## name
- #define **Y**(name) CONCAT(nfft_,name)
- #define **FFTW**(name) CONCAT(fftw_,name)
- #define **NFFT**(name) CONCAT(nfft_,name)
- #define **NFCT**(name) CONCAT(nfct_,name)
- #define **NFST**(name) CONCAT(nfst_,name)
- #define **NFSFT**(name) CONCAT(nfsft_,name)
- #define **SOLVER**(name) CONCAT(solver_,name)
- #define **X**(name) Y(name)
- #define **STRINGIZEx**(x) #x
- #define **STRINGIZE**(x) STRINGIZEx(x)
- #define **K**(x) ((R) x)
- #define **DK**(name, value) const R name = K(value)
- #define **KPI** K(3.14159265358979323846264338327950288419716939937510)
- #define **K2PI** K(6.28318530717958647692528676655900576839433879875021)
- #define **K4PI** K(12.56637061435917295385057353311801153678867759750042)
- #define **KE** K(2.71828182845904523536028747135266249775724709369995)
- #define **IF**(x, a, b) ((x)?(a):(b))
- #define **MIN**(a, b) (((a)<(b))?(a):(b))
- #define **MAX**(a, b) (((a)>(b))?(a):(b))
- #define **ABS**(x) (((x)>K(0.0))?(x):(-(x)))
- #define **SIGN**(a) (((a)>=0)?1:-1)
- #define **SIGN**(a) (((a)>=0)?1:-1)
- #define **SIGNF**(a) IF((a)<K(0.0),K(-1.0),K(1.0))
- #define **SIZE**(x) sizeof(x)/sizeof(x[0])
- #define CSWAP(x, y)

  Swap two vectors.
- #define RSWAP(x, y)

  Swap two vectors.
- #define **PHI_HUT**(n, k, d) (Y(bessel_i0)((R)(ths->m) * SQRT(ths->b[d] * ths->b[d] - (K(2.0) * KPI * (R)(k) / (R)(n)) * (K(2.0) * KPI * (R)(k) / (R)(n)))))
- #define **PHI**(n, x, d)
- #define **WINDOW_HELP_INIT**
- #define **WINDOW_HELP_FINALIZE** {Y(free)(ths->b);}
- #define **WINDOW_HELP_ESTIMATE_m** 8
- #define **COPYSIGN** copysign
- #define **NEXTAFTER** nextafter
- #define **MKNAN** nan
- #define **CEIL** ceil
- #define **FLOOR** floor
- #define **NEARBYINT** nearbyint
- #define **RINT** rint
- #define **ROUND** round
- #define **LRINT** lrint
- #define **LROUND** lround
- #define **LLRINT** llrint
- #define **LLROUND** llround

- #define **TRUNC** trunc
- #define **FMOD** fmod
- #define **REMAINDER** remainder
- #define **REMQUO** remquo
- #define **FDIM** fdim
- #define **FMAX** fmax
- #define **FMIN** fmin
- #define **FFMA** fma
- #define **FABS** fabs
- #define **SQRT** sqrt
- #define **CBRT** cbrt
- #define **HYPOT** hypot
- #define **EXP** exp
- #define **EXP2** exp2
- #define **EXPM1** expm1
- #define **LOG** log
- #define **LOG2** log2
- #define **LOG10** log10
- #define **LOG1P** log1p
- #define **LOGB** logb
- #define **ILOGB** ilogb
- #define **MODF** modf
- #define **FREXP** frexp
- #define **LDEXP** ldexp
- #define **SCALBN** scalbn
- #define **SCALBLN** scalbln
- #define **POW** pow
- #define **COS** cos
- #define **SIN** sin
- #define **TAN** tan
- #define **COSH** cosh
- #define **SINH** sinh
- #define **TANH** tanh
- #define **ACOS** acos
- #define **ASIN** asin
- #define **ATAN** atan
- #define **ATAN2** atan2
- #define **ACOSH** acosh
- #define **ASINH** asinh
- #define **ATANH** atanh
- #define **TGAMMA** tgamma
- #define **LGAMMA** lgamma
- #define **J0** j0
- #define **J1** j1
- #define **JN** jn
- #define **Y0** y0
- #define **Y1** y1
- #define **YN** yn
- #define **ERF** erf
- #define **ERFC** erfc
- #define **CREAL** creal
- #define **CIMAG** cimag
- #define **CABS** cabs
- #define **CARG** carg
- #define **CONJ** conj

- #define **CPROJ** cproj
- #define **CSQRT** csqrt
- #define **CEXP** cexp
- #define **CLOG** clog
- #define **CPOW** cpow
- #define **CSIN** csin
- #define **CCOS** ccos
- #define **CTAN** ctan
- #define **CASIN** casin
- #define **CACOS** cacos
- #define **CATAN** catan
- #define **CSINH** csinh
- #define **CCOSH** ccosh
- #define **CTANH** ctanh
- #define **CASINH** casinh
- #define **CACOSH** cacosh
- #define **CATANH** catanh
- #define **MANT_DIG** DBL_MANT_DIG
- #define **MIN_EXP** DBL_MIN_EXP
- #define **MAX_EXP** DBL_MAX_EXP
- #define **EPSILON** DBL_EPSILON
- #define **FLTROUND** 0.0
- #define **R_RADIX** FLT_RADIX
- #define **II** _Complex_I
- #define **__FGS__** "lg"
- #define **__FES__** "lE"
- #define **__FE__** "% 20.16lE"
- #define **__FI__** "%lf"
- #define **__FIS__** "lf"
- #define **__FR__** "%le"
- #define **TRUE** 1
- #define **FALSE** 0
- #define **__D__** "%td"
- #define UNUSED(x) (void)x

  Dummy use of unused parameters to silence compiler warnings.
- #define UNUSED(x) (void)x

  Dummy use of unused parameters to silence compiler warnings.
- #define **STACK_MALLOC**(T, p, x) p = (T)alloca(x)
- #define **STACK_FREE**(x) /∗ Nothing. Cleanup done automatically. ∗/
- #define TIC(a)

  Timing, method works since the inaccurate timer is updated mostly in the measured function.
- #define **TOC**(a)
- #define **TIC_FFTW**(a)
- #define **TOC_FFTW**(a)
- #define **CK**(ex) (void)((ex) || (Y(assertion_failed)(#ex, __LINE__, __FILE__), 0))
- #define **A**(ex) /∗ nothing ∗/

Typedefs

- typedef double **R**
- typedef double _Complex **C**
- typedef ptrdiff_t **INT**

Enumerations

- enum **float_property** {
  NFFT_EPSILON = 0, **NFFT_SAFE__MIN** = 1, **NFFT_BASE** = 2, **NFFT_PRECISION** = 3,
  NFFT_MANT_DIG = 4, **NFFT_FLTROUND** = 5, **NFFT_E_MIN** = 6, **NFFT_R_MIN** = 7,
  NFFT_E_MAX = 8, **NFFT_R_MAX** = 9 }

Functions

- INT **nfft_m2K** (const INT m)
- double **drand48** (void)
- void **srand48** (long int)
- void ∗ **alloca** (size_t)
- R nfft_elapsed_seconds (ticks t1, ticks t0)

  Return number of elapsed seconds between two time points.
- R **nfft_sinc** (R x)
- R **nfft_lambda** (R z, R eps)
- R **nfft_lambda2** (R mu, R nu)
- R **nfft_bessel_i0** (R x)
- R **nfft_bsplines** (const INT, const R x)
- R **nfft_float_property** (float_property)
- R **nfft_prod_real** (R ∗vec, INT d)
- INT **nfft_log2i** (const INT m)
- void **nfft_next_power_of_2_exp** (const INT N, INT ∗N2, INT ∗t)
- void **nfft_next_power_of_2_exp_int** (const int N, int ∗N2, int ∗t)
- R **nfft_error_l_infty_double** (const R ∗x, const R ∗y, const INT n)
- R **nfft_error_l_infty_1_double** (const R ∗x, const R ∗y, const INT n, const R ∗z, const INT m)
- R **nfft_error_l_2_complex** (const C ∗x, const C ∗y, const INT n)
- R **nfft_error_l_2_double** (const R ∗x, const R ∗y, const INT n)
- void **nfft_sort_node_indices_radix_msdf** (INT n, INT ∗keys0, INT ∗keys1, INT rhigh)
- void **nfft_sort_node_indices_radix_lsdf** (INT n, INT ∗keys0, INT ∗keys1, INT rhigh)
- void **nfft_assertion_failed** (const char ∗s, int line, const char ∗file)
- R nfft_dot_double (R ∗x, INT n)

  Computes the inner/dot product $x^H x$.
- R nfft_dot_w_complex (C ∗x, R ∗w, INT n)

  Computes the weighted inner/dot product $x^H (w \odot x)$.
- R nfft_dot_w_double (R ∗x, R ∗w, INT n)

  Computes the weighted inner/dot product $x^H (w \odot x)$.
- R nfft_dot_w_w2_complex (C ∗x, R ∗w, R ∗w2, INT n)

  Computes the weighted inner/dot product $x^H (w \odot w2 \odot w2 \odot x)$.
- R nfft_dot_w2_complex (C ∗x, R ∗w2, INT n)

  Computes the weighted inner/dot product $x^H (w2 \odot w2 \odot x)$.
- void nfft_cp_complex (C ∗x, C ∗y, INT n)

  Copies $x \leftarrow y$.
- void nfft_cp_double (R ∗x, R ∗y, INT n)

  Copies $x \leftarrow y$.
- void nfft_cp_a_complex (C ∗x, R a, C ∗y, INT n)

  Copies $x \leftarrow ay$.
- void nfft_cp_a_double (R ∗x, R a, R ∗y, INT n)

  Copies $x \leftarrow ay$.
- void nfft_cp_w_complex (C ∗x, R ∗w, C ∗y, INT n)

  Copies $x \leftarrow w \odot y$.

- void nfft_cp_w_double (R *x, R *w, R *y, INT n)

  Copies $x \leftarrow w \odot y$.
- void nfft_upd_axpy_double (R *x, R a, R *y, INT n)

  Updates $x \leftarrow ax + y$.
- void nfft_upd_xpay_complex (C *x, R a, C *y, INT n)

  Updates $x \leftarrow x + ay$.
- void nfft_upd_xpay_double (R *x, R a, R *y, INT n)

  Updates $x \leftarrow x + ay$.
- void nfft_upd_axpby_complex (C *x, R a, C *y, R b, INT n)

  Updates $x \leftarrow ax + by$.
- void nfft_upd_axpby_double (R *x, R a, R *y, R b, INT n)

  Updates $x \leftarrow ax + by$.
- void nfft_upd_xpawy_complex (C *x, R a, R *w, C *y, INT n)

  Updates $x \leftarrow x + aw \odot y$.
- void nfft_upd_xpawy_double (R *x, R a, R *w, R *y, INT n)

  Updates $x \leftarrow x + aw \odot y$.
- void nfft_upd_axpwy_complex (C *x, R a, R *w, C *y, INT n)

  Updates $x \leftarrow ax + w \odot y$.
- void nfft_upd_axpwy_double (R *x, R a, R *w, R *y, INT n)

  Updates $x \leftarrow ax + w \odot y$.
- void **nfft_voronoi_weights_1d** (R *w, R *x, const INT M)
- R nfft_modified_fejer (const INT N, const INT kk)

  Compute damping factor for modified Fejer kernel: /f$\frac{2}{N}\left(1-\frac{\left|2k+1\right|}{N}\right)/f$.
- R nfft_modified_jackson2 (const INT N, const INT kk)

  Compute damping factor for modified Jackson kernel.
- R nfft_modified_jackson4 (const INT N, const INT kk)

  Compute damping factor for modified generalised Jackson kernel.
- R nfft_modified_sobolev (const R mu, const INT kk)

  Compute damping factor for modified Sobolev kernel.
- R nfft_modified_multiquadric (const R mu, const R c, const INT kk)

  Comput damping factor for modified multiquadric kernel.

5.11.1 Detailed Description

This module implements frequently used utility functions.

In particular, this includes simple measurement of resources, evaluation of window functions, vector routines for basic linear algebra tasks, and computation of weights for the inverse transforms.

5.11.2 Macro Definition Documentation

5.11.2.1 CSWAP

```
#define CSWAP(
              x,
              y )
```

Value:

```
  {C* NFFT_SWAP_temp__; \
  NFFT_SWAP_temp__=(x); (x)=(y); (y)=NFFT_SWAP_temp__;}
```

Swap two vectors.

Definition at line 139 of file infft.h.

5.11.2.2 RSWAP

```
#define RSWAP(
              x,
              y )
```

Value:

```
  {R* NFFT_SWAP_temp__; NFFT_SWAP_temp__=(x); \
  (x)=(y); (y)=NFFT_SWAP_temp__;}
```

Swap two vectors.

Definition at line 143 of file infft.h.

5.11.2.3 PHI

```
#define PHI(
              n,
              x,
              d )
```

Value:

```
( (((R)(ths->m) * (R)(ths->m) - (x) * (R)(n) * (x) * (R)(n)) > K(0.0)) \
?  SINH(ths->b[d] * SQRT((R)(ths->m) * (R)(ths->m) - (x) * (R)(n) * (x) * (R)(n))) \
  / (KPI * SQRT((R)(ths->m) * (R)(ths->m) - (x) * (R)(n) * (x) * (R)(n))) \
:  ((((R)(ths->m) * (R)(ths->m) - (x) * (R)(n) * (x) * (R)(n)) < K(0.0)) \
  ?  SIN(ths->b[d] * SQRT((x) * (R)(n) * (x) * (R)(n) - (R)(ths->m) * (R)(ths->m))) \
    / (KPI * SQRT((x) * (R)(n) * (x) * (R)(n) - (R)(ths->m) * (R)(ths->m))) \
  : ths->b[d] / KPI))
```

Definition at line 209 of file infft.h.

5.11.2.4 WINDOW_HELP_INIT

```
#define WINDOW_HELP_INIT
```

Value:

```
    { \
      int WINDOW_idx; \
      ths->b = (R*) Y(malloc)((size_t)(ths->d) * sizeof(R)); \
      for (WINDOW_idx = 0; WINDOW_idx < ths->d; WINDOW_idx++) \
        ths->b[WINDOW_idx] = (KPI * (K(2.0) - K(1.0) / ths->sigma[WINDOW_idx])); \
  }
```

Definition at line 216 of file infft.h.

5.11.2.5 TIC

```
#define TIC(
              a )
```

Timing, method works since the inaccurate timer is updated mostly in the measured function.

For small times not every call of the measured function will also produce a 'unit' time step. Measuring the fftw might cause a wrong output vector due to the repeated ffts.

Definition at line 1400 of file infft.h.

5.12 Examples

Modules

- Solver component

5.12.1 Detailed Description

5.13 Solver component

Modules

- Reconstruction of a glacier fromscattered data

5.13.1 Detailed Description

5.14 Reconstruction of a glacier fromscattered data

Functions

- static NFFT_R my_weight (NFFT_R z, NFFT_R a, NFFT_R b, NFFT_R c)

  Generalised Sobolev weight.
- static void glacier (int N, int M)

  Reconstruction routine.
- static void glacier_cv (int N, int M, int M_cv, unsigned solver_flags)

  Reconstruction routine with cross validation.
- int main (int argc, char ∗∗argv)

  Main routine.

5.14.1 Detailed Description

5.15 Applications

Modules

- Fast Gauss transfrom with complex parameter
- Fast summation

  Direct and fast summation (convolution)
- Fast summation of radial functions on the sphere
- Transforms in magnetic resonance imaging
- Polar FFT
- Fast evaluation of quadrature formulae on the sphere

5.15.1 Detailed Description

5.16 Fast Gauss transfrom with complex parameter

Data Structures

- struct fgt_plan

  Structure for the Gauss transform.

Macros

- #define **NFFT_PRECISION_DOUBLE**
- #define DGT_PRE_CEXP (1U<< 0)

  If this flag is set, the whole matrix is precomputed and stored for the discrete Gauss transfrom.
- #define FGT_NDFT (1U<< 1)

  If this flag is set, the fast Gauss transform uses the discrete instead of the fast Fourier transform.
- #define FGT_APPROX_B (1U<< 2)

  If this flag is set, the discrete Fourier coefficients of the uniformly sampled Gaussian are used instead of the sampled continuous Fourier transform.

Functions

- static void dgt_trafo (fgt_plan *ths)

  Executes the discrete Gauss transform.
- static void fgt_trafo (fgt_plan *ths)

  Executes the fast Gauss transform.
- static void fgt_init_guru (fgt_plan *ths, int N, int M, NFFT_C sigma, int n, NFFT_R p, int m, unsigned flags)

  Initialisation of a transform plan, guru.
- static void fgt_init (fgt_plan *ths, int N, int M, NFFT_C sigma, NFFT_R eps)

  Initialisation of a transform plan, simple.
- static void fgt_init_node_dependent (fgt_plan *ths)

  Initialisation of a transform plan, depends on source and target nodes.
- static void fgt_finalize (fgt_plan *ths)

  Destroys the transform plan.
- static void fgt_test_init_rand (fgt_plan *ths)

  Random initialisation of a fgt plan.
- static NFFT_R fgt_test_measure_time (fgt_plan *ths, unsigned dgt)

  Compares execution times for the fast and discrete Gauss transform.
- static void fgt_test_simple (int N, int M, NFFT_C sigma, NFFT_R eps)

  Simple example that computes fast and discrete Gauss transforms.
- static void fgt_test_andersson (void)

  Compares accuracy and execution time of the fast Gauss transform with increasing expansion degree.
- static void fgt_test_error (void)

  Compares accuracy of the fast Gauss transform with increasing expansion degree.
- static void fgt_test_error_p (void)

  Compares accuracy of the fast Gauss transform with increasing expansion degree and different periodisation lengths.
- int main (int argc, char **argv)

  Different tests of the fast Gauss transform.

Variables

- int fgt_plan::N
  number of source nodes
- int fgt_plan::M
  number of target nodes
- NFFT_C * fgt_plan::alpha
  source coefficients
- NFFT_C * fgt_plan::f
  target evaluations
- unsigned fgt_plan::flags
  flags for precomputation and approximation type
- NFFT_C fgt_plan::sigma
  parameter of the Gaussian
- NFFT_R * fgt_plan::x
  source nodes in $[-1/4, 1/4]$
- NFFT_R * fgt_plan::y
  target nodes in $[-1/4, 1/4]$
- NFFT_C * fgt_plan::pre_cexp
  precomputed values for dgt
- int fgt_plan::n
  expansion degree
- NFFT_R fgt_plan::p
  period, at least 1
- NFFT_C * fgt_plan::b
  expansion coefficients

5.16.1 Detailed Description

5.16.2 Macro Definition Documentation

5.16.2.1 DGT_PRE_CEXP

```
#define DGT_PRE_CEXP (1U<< 0)
```

If this flag is set, the whole matrix is precomputed and stored for the discrete Gauss transfrom.

See also

fgt_init_node_dependent

fgt_init

Author

Stefan Kunis

Definition at line 42 of file fastgauss.c.

5.16.2.2 FGT_NDFT

```
#define FGT_NDFT (1U<< 1)
```

If this flag is set, the fast Gauss transform uses the discrete instead of the fast Fourier transform.

See also

fgt_init

nfft_trafo_direct

nfft_trafo

Author

Stefan Kunis

Definition at line 53 of file fastgauss.c.

5.16.2.3 FGT_APPROX_B

```
#define FGT_APPROX_B (1U<< 2)
```

If this flag is set, the discrete Fourier coefficients of the uniformly sampled Gaussian are used instead of the sampled continuous Fourier transform.

See also

fgt_init

Author

Stefan Kunis

Definition at line 63 of file fastgauss.c.

5.16.3 Function Documentation

5.16.3.1 dgt_trafo()

```
static void dgt_trafo (
            fgt_plan * ths )  [static]
```

Executes the discrete Gauss transform.

- ths The pointer to a fgt plan

Author

Stefan Kunis

Definition at line 101 of file fastgauss.c.

References fgt_plan::alpha, DGT_PRE_CEXP, fgt_plan::f, fgt_plan::flags, fgt_plan::M, fgt_plan::N, fgt_plan::pre_↩cexp, fgt_plan::sigma, fgt_plan::x, and fgt_plan::y.

5.16.3.2 fgt_trafo()

```
static void fgt_trafo (
            fgt_plan * ths )  [static]
```

Executes the fast Gauss transform.

- ths The pointer to a fgt plan

Author

Stefan Kunis

Definition at line 128 of file fastgauss.c.

References FGT_NDFT, and fgt_plan::flags.

5.16.3.3 fgt_init_guru()

```
static void fgt_init_guru (
             fgt_plan * ths,
             int N,
             int M,
             NFFT_C sigma,
             int n,
             NFFT_R p,
             int m,
             unsigned flags )  [static]
```

Initialisation of a transform plan, guru.

- ths The pointer to a fpt plan
- N The number of source nodes
- M The number of target nodes
- sigma The parameter of the Gaussian
- n The polynomial expansion degree
- p the periodisation length, at least 1
- m The spatial cut-off of the nfft
- flags FGT flags to use

Author

Stefan Kunis

Definition at line 166 of file fastgauss.c.

Referenced by fgt_init().

5.16.3.4 fgt_init()

```
static void fgt_init (
             fgt_plan * ths,
             int N,
             int M,
             NFFT_C sigma,
             NFFT_R eps )  [static]
```

Initialisation of a transform plan, simple.

- ths The pointer to a fpt plan
- N The number of source nodes
- M The number of target nodes
- sigma The parameter of the Gaussian
- eps The target accuracy

Author

Stefan Kunis

Definition at line 240 of file fastgauss.c.

References DGT_PRE_CEXP, and fgt_init_guru().

Referenced by fgt_test_simple().

5.16.3.5 fgt_init_node_dependent()

```
static void fgt_init_node_dependent (
             fgt_plan * ths )  [static]
```

Initialisation of a transform plan, depends on source and target nodes.

- ths The pointer to a fpt plan

Author

Stefan Kunis

Definition at line 261 of file fastgauss.c.

References DGT_PRE_CEXP, fgt_plan::flags, and fgt_plan::pre_cexp.

5.16.3.6 fgt_finalize()

```
static void fgt_finalize (
             fgt_plan * ths )  [static]
```

Destroys the transform plan.

- ths The pointer to the fgt plan

Author

Stefan Kunis

Definition at line 292 of file fastgauss.c.

5.16.3.7 fgt_test_init_rand()

```
static void fgt_test_init_rand (
            fgt_plan * ths )  [static]
```

Random initialisation of a fgt plan.

- ths The pointer to the fgt plan

Author

Stefan Kunis

Definition at line 315 of file fastgauss.c.

References fgt_plan::N, and fgt_plan::x.

5.16.3.8 fgt_test_measure_time()

```
static NFFT_R fgt_test_measure_time (
            fgt_plan * ths,
            unsigned dgt )  [static]
```

Compares execution times for the fast and discrete Gauss transform.

- ths The pointer to the fgt plan
- dgt If this parameter is set dgt_trafo is called as well

Author

Stefan Kunis

Definition at line 338 of file fastgauss.c.

5.16.3.9 fgt_test_simple()

```
static void fgt_test_simple (
            int N,
            int M,
            NFFT_C sigma,
            NFFT_R eps )  [static]
```

Simple example that computes fast and discrete Gauss transforms.

- ths The pointer to the fgt plan
- sigma The parameter of the Gaussian
- eps The target accuracy

Author

Stefan Kunis

Definition at line 373 of file fastgauss.c.

References fgt_init().

Referenced by main().

5.16.3.10 fgt_test_andersson()

```
static void fgt_test_andersson (
            void  )  [static]
```

Compares accuracy and execution time of the fast Gauss transform with increasing expansion degree.

Similar to the test in F. Andersson and G. Beylkin. The fast Gauss transform with complex parameters. J. Comput. Physics 203 (2005) 274-286

Author

Stefan Kunis

Definition at line 409 of file fastgauss.c.

Referenced by main().

5.16.3.11 fgt_test_error()

```
static void fgt_test_error (
            void  )  [static]
```

Compares accuracy of the fast Gauss transform with increasing expansion degree.

Author

Stefan Kunis

Definition at line 475 of file fastgauss.c.

Referenced by main().

5.16.3.12 fgt_test_error_p()

```
static void fgt_test_error_p (
            void  )  [static]
```

Compares accuracy of the fast Gauss transform with increasing expansion degree and different periodisation lengths.

Author

Stefan Kunis

Definition at line 527 of file fastgauss.c.

Referenced by main().

5.16.3.13 main()

```
int main (
            int argc,
            char ** argv )
```

Different tests of the fast Gauss transform.

Author

Stefan Kunis

Definition at line 576 of file fastgauss.c.

References fgt_test_andersson(), fgt_test_error(), fgt_test_error_p(), and fgt_test_simple().

5.17 Fast summation

Direct and fast summation (convolution)

Modules

- fastsum_matlab
- fastsum_test

Data Structures

- struct fastsum_plan_

  plan for fast summation algorithm

Macros

- #define X(name) NFFT(name)

  Include header for C99 complex datatype.
- #define **NF_KUB**
- #define EXACT_NEARFIELD (1U<< 0)

  Constant symbols.
- #define **NEARFIELD_BOXES** (1U<< 1)
- #define STORE_PERMUTATION_X_ALPHA (1U<< 2)

  If this flag is set, and eps_I > 0.0 and NEARFIELD_BOXES is not set, then the vector permutation_x_alpha is stored.

Typedefs

- typedef C(∗ **kernel**) (R, int, const R ∗)
- typedef struct fastsum_plan_ fastsum_plan

  plan for fast summation algorithm

Functions

- static int max_i (int a, int b)

  max
- static R fak (int n)

  factorial
- static R binom (int n, int m)

  binomial coefficient
- static R BasisPoly (int m, int r, R xx)

  basis polynomial for regularized kernel
- C regkern (kernel k, R xx, int p, const R ∗param, R a, R b)

  regularized kernel with K_I arbitrary and K_B smooth to zero
- static C regkern1 (kernel k, R xx, int p, const R ∗param, R a, R b)

  regularized kernel with K_I arbitrary and K_B periodized (used in 1D)
- static C regkern3 (kernel k, R xx, int p, const R ∗param, R a, R b)

  regularized kernel for even kernels with K_I even and K_B mirrored

- C kubintkern (const R x, const C *Add, const int Ad, const R a)

  linear spline interpolation in near field with even kernels
- static C kubintkern1 (const R x, const C *Add, const int Ad, const R a)

  cubic spline interpolation in near field with arbitrary kernels
- static void quicksort (int d, int t, R *x, C *alpha, int *permutation_x_alpha, int N)

  quicksort algorithm for source knots and associated coefficients
- static void BuildBox (fastsum_plan *ths)

  initialize box-based search data structures
- static C calc_SearchBox (int d, R *y, R *x, C *alpha, int start, int end_lt, const C *Add, const int Ad, int p, R a, const kernel k, const R *param, const unsigned flags)

  inner computation function for box-based near field correction
- static C SearchBox (R *y, fastsum_plan *ths)

  box-based near field correction
- static void BuildTree (int d, int t, R *x, C *alpha, int *permutation_x_alpha, int N)

  recursive sort of source knots dimension by dimension to get tree structure
- static C SearchTree (const int d, const int t, const R *x, const C *alpha, const R *xmin, const R *xmax, const int N, const kernel k, const R *param, const int Ad, const C *Add, const int p, const unsigned flags)

  fast search in tree of source knots for near field computation
- static void **fastsum_precompute_kernel** (fastsum_plan *ths)
- void fastsum_init_guru_kernel (fastsum_plan *ths, int d, kernel k, R *param, unsigned flags, int nn, int p, R eps_I, R eps_B)

  initialize node independent part of fast summation plan
- void fastsum_init_guru_source_nodes (fastsum_plan *ths, int N_total, int nn_oversampled, int m)

  initialize source nodes dependent part of fast summation plan
- void fastsum_init_guru_target_nodes (fastsum_plan *ths, int M_total, int nn_oversampled, int m)

  initialize target nodes dependent part of fast summation plan
- void fastsum_init_guru (fastsum_plan *ths, int d, int N_total, int M_total, kernel k, R *param, unsigned flags, int nn, int m, int p, R eps_I, R eps_B)

  initialization of fastsum plan
- void fastsum_finalize_source_nodes (fastsum_plan *ths)

  finalization of fastsum plan
- void fastsum_finalize_target_nodes (fastsum_plan *ths)

  finalization of fastsum plan
- void fastsum_finalize_kernel (fastsum_plan *ths)

  finalization of fastsum plan
- void fastsum_finalize (fastsum_plan *ths)

  finalization of fastsum plan
- void fastsum_exact (fastsum_plan *ths)

  direct computation of sums
- void fastsum_precompute_source_nodes (fastsum_plan *ths)

  precomputation for fastsum
- void fastsum_precompute_target_nodes (fastsum_plan *ths)

  precomputation for fastsum
- void fastsum_precompute (fastsum_plan *ths)

  precomputation for fastsum
- void fastsum_trafo (fastsum_plan *ths)

  fast NFFT-based summation
- fastsum_plan_::NFFT (plan) mv1

  source nfft plan

- **fastsum_plan_::FFTW** (plan) fft_plan

- C gaussian (R x, int der, const R *param)
  K(x)=exp(-x^2/c^2)
- C multiquadric (R x, int der, const R *param)
  K(x)=sqrt(x^2+c^2)
- C inverse_multiquadric (R x, int der, const R *param)
  K(x)=1/sqrt(x^2+c^2)
- C logarithm (R x, int der, const R *param)
  K(x)=log |x|.
- C thinplate_spline (R x, int der, const R *param)
  K(x) = x^2 log |x|.
- C one_over_square (R x, int der, const R *param)
  K(x) = 1/x^2.
- C one_over_modulus (R x, int der, const R *param)
  K(x) = 1/|x|.
- C one_over_x (R x, int der, const R *param)
  K(x) = 1/x.
- C inverse_multiquadric3 (R x, int der, const R *param)
  K(x) = 1/sqrt(x^2+c^2)^3.
- C sinc_kernel (R x, int der, const R *param)
  K(x) = sin(cx)/x.
- C cosc (R x, int der, const R *param)
  K(x) = cos(cx)/x.
- C kcot (R x, int der, const R *param)
  K(x) = cot(cx)
- C one_over_cube (R x, int der, const R *param)
  K(x) = 1/x^3.
- C log_sin (R x, int der, const R *param)
  K(x) = log(|sin(cx)|)
- C laplacian_rbf (R x, int der, const R *param)
  K(x) = exp(-|x|/c)

Variables

- int fastsum_plan_::d
  api
- int fastsum_plan_::N_total
  number of source knots
- int fastsum_plan_::M_total
  number of target knots
- C * fastsum_plan_::alpha
  source coefficients
- C * fastsum_plan_::f
  target evaluations
- R * fastsum_plan_::x
  source knots in d-ball with radius 1/4-eps_b/2
- R * fastsum_plan_::y
  target knots in d-ball with radius 1/4-eps_b/2

- kernel fastsum_plan_::k
  kernel function
- R * fastsum_plan_::kernel_param
  parameters for kernel function
- unsigned fastsum_plan_::flags
  flags precomp.
- C * fastsum_plan_::pre_K
  internal
- int fastsum_plan_::n
  FS__ - fast summation.
- C * fastsum_plan_::b
  expansion coefficients
- C * fastsum_plan_::f_hat
  Fourier coefficients of nfft plans.
- int fastsum_plan_::p
  degree of smoothness of regularization
- R fastsum_plan_::eps_I
  inner boundary
- R fastsum_plan_::eps_B
  outer boundary
- int fastsum_plan_::Ad
  near field
- C * fastsum_plan_::Add
  spline values
- int **fastsum_plan_::box_count**
- int **fastsum_plan_::box_count_per_dim**
- int * **fastsum_plan_::box_offset**
- R * **fastsum_plan_::box_x**
- C * **fastsum_plan_::box_alpha**
- int * fastsum_plan_::permutation_x_alpha
  permutation vector of source nodes if STORE_PERMUTATION_X_ALPHA is set
- R fastsum_plan_::MEASURE_TIME_t [8]
  Measured time for each step if MEASURE_TIME is set.

5.17.1 Detailed Description

Direct and fast summation (convolution)

Computes the sums

$$f(y_j) = \sum_{k=1}^{N} \alpha_k K(x_k - y_j), \quad j = 1 \dots M.$$

5.17.2 Macro Definition Documentation

5.17.2.1 X

```
#define X(
              name ) NFFT(name)
```

Include header for C99 complex datatype.

Include header for utils from NFFT3 library. Include header for NFFT3 library.

Definition at line 57 of file fastsum.h.

5.17.3 Function Documentation

5.17.3.1 regkern3()

```
static C regkern3 (
             kernel k,
             R xx,
             int p,
             const R * param,
             R a,
             R b )  [static]
```

regularized kernel for even kernels with K_I even and K_B mirrored

regularized kernel for even kernels with K_I even and K_B mirrored smooth to K(1/2) (used in dD, d>1)

Definition at line 230 of file fastsum.c.

5.17.3.2 kubintkern()

```
C kubintkern (
             const R x,
             const C * Add,
             const int Ad,
             const R a )
```

linear spline interpolation in near field with even kernels

cubic spline interpolation in near field with even kernels

Definition at line 318 of file fastsum.c.

5.17.3.3 fastsum_init_guru_kernel()

```
void fastsum_init_guru_kernel (
             fastsum_plan * ths,
             int d,
             kernel k,
             R * param,
             unsigned flags,
             int nn,
             int p,
             R eps_I,
             R eps_B )
```

initialize node independent part of fast summation plan

Parameters

ths	The pointer to a fastsum plan.
d	The dimension of the problem.
kernel	The kernel function.
param	The parameters for the kernel function.
flags	Fastsum flags.
nn	The expansion degree.
p	The degree of smoothness.
eps_I	The inner boundary.
eps_B	the outer boundary.

Definition at line 779 of file fastsum.c.

5.17.3.4 fastsum_init_guru_source_nodes()

```
void fastsum_init_guru_source_nodes (
             fastsum_plan * ths,
             int N_total,
             int nn_oversampled,
             int m )
```

initialize source nodes dependent part of fast summation plan

Parameters

ths	The pointer to a fastsum plan.
N_total	The number of source knots x.
nn_oversampled	The oversampled expansion degree for nfft.
m	The cut-off parameter for the NFFT.

Definition at line 887 of file fastsum.c.

References fastsum_plan_::d.

5.17.3.5 fastsum_init_guru_target_nodes()

```
void fastsum_init_guru_target_nodes (
             fastsum_plan * ths,
             int M_total,
             int nn_oversampled,
             int m )
```

initialize target nodes dependent part of fast summation plan

Parameters

ths	The pointer to a fastsum plan.
M_total	The number of target knots y.
nn_oversampled	The oversampled expansion degree for nfft.
m	The cut-off parameter for the NFFT.

Definition at line 955 of file fastsum.c.

References fastsum_plan_::d.

5.17.3.6 fastsum_init_guru()

```
void fastsum_init_guru (
              fastsum_plan * ths,
              int d,
              int N_total,
              int M_total,
              kernel k,
              R * param,
              unsigned flags,
              int nn,
              int m,
              int p,
              R eps_I,
              R eps_B )
```

initialization of fastsum plan

initialize fast summation plan

Parameters

ths	The pointer to a fastsum plan.
d	The dimension of the problem.
N_total	The number of source knots x.
M_total	The number of target knots y.
kernel	The kernel function.
param	The parameters for the kernel function.
flags	Fastsum flags.
nn	The expansion degree.
m	The cut-off parameter for the NFFT.
p	The degree of smoothness.
eps_I	The inner boundary.
eps_B	the outer boundary.

Definition at line 987 of file fastsum.c.

5.17.3.7 fastsum_finalize_source_nodes()

```
void fastsum_finalize_source_nodes (
            fastsum_plan * ths )
```

finalization of fastsum plan

finalize source nodes dependent part of plan

Parameters

ths	The pointer to a fastsum plan.

Definition at line 996 of file fastsum.c.

Referenced by fastsum_finalize().

5.17.3.8 fastsum_finalize_target_nodes()

```
void fastsum_finalize_target_nodes (
            fastsum_plan * ths )
```

finalization of fastsum plan

finalize target nodes dependent part of plan

Parameters

ths	The pointer to a fastsum plan.

Definition at line 1020 of file fastsum.c.

Referenced by fastsum_finalize().

5.17.3.9 fastsum_finalize_kernel()

```
void fastsum_finalize_kernel (
            fastsum_plan * ths )
```

finalization of fastsum plan

finalize node independent part of plan

Parameters

ths	The pointer to a fastsum plan.

Definition at line 1029 of file fastsum.c.

References fastsum_plan_::eps_I, EXACT_NEARFIELD, and fastsum_plan_::flags.

Referenced by fastsum_finalize().

5.17.3.10 fastsum_finalize()

```
void fastsum_finalize (
            fastsum_plan * ths )
```

finalization of fastsum plan

finalize plan

Parameters

ths	The pointer to a fastsum plan.

Definition at line 1048 of file fastsum.c.

References fastsum_finalize_kernel(), fastsum_finalize_source_nodes(), and fastsum_finalize_target_nodes().

5.17.3.11 fastsum_exact()

```
void fastsum_exact (
            fastsum_plan * ths )
```

direct computation of sums

direct summation

Parameters

ths	The pointer to a fastsum plan.

Definition at line 1056 of file fastsum.c.

References fastsum_plan_::f, and fastsum_plan_::M_total.

5.17.3.12 fastsum_precompute_source_nodes()

```
void fastsum_precompute_source_nodes (
            fastsum_plan * ths )
```

precomputation for fastsum

sort source nodes, precompute nfft source plan.

Parameters

ths	The pointer to a fastsum plan.

Definition at line 1086 of file fastsum.c.

References fastsum_plan_::MEASURE_TIME_t.

Referenced by fastsum_precompute().

5.17.3.13 fastsum_precompute_target_nodes()

```
void fastsum_precompute_target_nodes (
             fastsum_plan * ths )
```

precomputation for fastsum

precompute nfft target plan.

Parameters

ths	The pointer to a fastsum plan.

Definition at line 1141 of file fastsum.c.

References fastsum_plan_::MEASURE_TIME_t.

Referenced by fastsum_precompute().

5.17.3.14 fastsum_precompute()

```
void fastsum_precompute (
             fastsum_plan * ths )
```

precomputation for fastsum

sort source nodes, precompute nfft plans etc.

Parameters

ths	The pointer to a fastsum plan.

Definition at line 1173 of file fastsum.c.

References fastsum_precompute_source_nodes(), and fastsum_precompute_target_nodes().

5.17.3.15 fastsum_trafo()

```
void fastsum_trafo (
            fastsum_plan * ths )
```

fast NFFT-based summation

fast NFFT-based summation algorithm

Parameters

ths	The pointer to a fastsum plan.

Definition at line 1180 of file fastsum.c.

References fastsum_plan_::MEASURE_TIME_t.

5.17.4 Variable Documentation

5.17.4.1 d

```
int fastsum_plan_::d
```

api

number of dimensions

Definition at line 86 of file fastsum.h.

Referenced by BuildBox(), fastsum_init_guru_source_nodes(), and fastsum_init_guru_target_nodes().

5.17.4.2 flags

```
unsigned fastsum_plan_::flags
```

flags precomp.

and approx.type

Definition at line 100 of file fastsum.h.

Referenced by fastsum_finalize_kernel().

5.17.4.3 pre_K

`C* fastsum_plan_::pre_K`

internal

DS_PRE - direct summation precomputed K(x_j-y_l)

Definition at line 105 of file fastsum.h.

5.17.4.4 n

`int fastsum_plan_::n`

FS__ - fast summation.

expansion degree

Definition at line 108 of file fastsum.h.

5.17.4.5 Ad

`int fastsum_plan_::Ad`

near field

number of spline knots for nearfield computation of regularized kernel

Definition at line 120 of file fastsum.h.

5.18 fastsum_matlab

Functions

- int **main** (int argc, char ∗∗argv)

5.18.1 Detailed Description

5.19 fastsum_test

Functions

- int **main** (int argc, char ∗∗argv)

5.19.1 Detailed Description

5.20 Fast summation of radial functions on the sphere

Modules

- fastsumS2_matlab

5.20.1 Detailed Description

5.21 fastsumS2_matlab

Macros

- #define **SYMBOL_ABEL_POISSON**(k, h) (pow(h,k))
- #define **SYMBOL_SINGULARITY**(k, h) ((2.0/(2∗k+1))∗pow(h,k))
- #define KT_ABEL_POISSON (0)

  Abel-Poisson kernel.
- #define KT_SINGULARITY (1)

  Singularity kernel.
- #define KT_LOC_SUPP (2)

  Locally supported kernel.
- #define KT_GAUSSIAN (3)

  Gaussian kernel.

Enumerations

- enum pvalue { **NO** = 0, **YES** = 1, **BOTH** = 2 }

  Enumeration type for yes/no/both-type parameters.

Functions

- static int **scaled_modified_bessel_i_series** (const R x, const R alpha, const int nb, const int ize, R ∗b)
- static void **scaled_modified_bessel_i_normalize** (const R x, const R alpha, const int nb, const int ize, R ∗b, const R sum_)
- static int smbi (const R x, const R alpha, const int nb, const int ize, R ∗b)

  Calculates the modified bessel function $I_{n+\alpha}(x)$, possibly scaled by e^{-x}, for real non-negative $x, alpha$ with $0 \leq \alpha < 1$, and $n = 0, 1, \ldots, nb - 1$.
- static double innerProduct (const double phi1, const double theta1, const double phi2, const double theta2)

  Computes the $\mathbb{R}^3$ standard inner product between two vectors on the unit sphere $\mathbb{S}^2$ given in spherical coordinates.
- static double poissonKernel (const double x, const double h)

  Evaluates the Poisson kernel $Q_h : [-1, 1] \to \mathbb{R}$ at a node $x \in [-1, 1]$.
- static double singularityKernel (const double x, const double h)

  Evaluates the singularity kernel $S_h : [-1, 1] \to \mathbb{R}$ at a node $x \in [-1, 1]$.
- static double locallySupportedKernel (const double x, const double h, const double lambda)

  Evaluates the locally supported kernel $L_{h,\lambda} : [-1, 1] \to \mathbb{R}$ at a node $x \in [-1, 1]$.
- static double gaussianKernel (const double x, const double sigma)

  Evaluates the spherical Gaussian kernel $G_\sigma : [-1, 1] \to \mathbb{R}$ at a node $x \in [-1, 1]$.
- int main (int argc, char ∗∗argv)

  The main program.

5.21.1 Detailed Description

5.21.2 Function Documentation

5.21.2.1 smbi()

```
static int smbi (
             const R x,
             const R alpha,
             const int nb,
             const int ize,
             R * b )  [static]
```

Calculates the modified bessel function $I_{n+\alpha}(x)$, possibly scaled by e^{-x}, for real non-negative $x, alpha$ with $0 \leq \alpha < 1$, and $n = 0, 1, \ldots, nb - 1$.

- [in] x non-negative real number in $I_{n+\alpha}(x)$
- [in] alpha non-negative real number with $0 \leq \alpha < 1$ in $I_{n+\alpha}(x)$
- [in] nb number of functions to be calculated
- [in] ize switch between no scaling (ize = 1) and exponential scaling (ize = 2)
- [out] b real output vector to contain $I_{n+\alpha}(x)$, $n = 0, 1, \ldots, nb - 1$

Returns

error indicator. Only if this value is identical to nb, then all values in b have been calculated to full accuracy. If not, errors are indicated using the following scheme:

- ncalc $<$ 0: At least one of the arguments was out of range (e.g. nb $<=$ 0, ize neither equals 1 nor 2, $|x| \geq expar g$). In this case, the output vector b is not calculated and ncalc is set to $\min(nb, 0) - 1$.
- 0 $<$ ncalc $<$ nb: Not all requested functions could be calculated to full accuracy. This can occur when nb is much larger than |x|. in this case, the values $I_{n+\alpha}(x)$ are calculated to full accuracy for $n = 0, 1, \ldots, ncalc$. The rest of the values up to $n = 0, 1, \ldots, nb - 1$ is calculated to a lower accuracy.

\acknowledgement

This program is based on a program written by David J. Sookne [2] that computes values of the Bessel functions $J_\nu(x)$ or $I_\nu(x)$ for real argument x and integer order ν. modifications include the restriction of the computation to the Bessel function $I_\nu(x)$ for non-negative real argument, the extension of the computation to arbitrary non-negative orders ν, and the elimination of most underflow.

References: [1] F. W. J. Olver and D. J. Sookne, A note on backward recurrence algorithms", Math. Comput. (26), 1972, pp 125 -- 132. [2] D. J. Sookne, "Bessel functions of real argument and int order", NBS Jour. of Res. B. (77B), 1973, pp. 125 – 132.

Modified by W. J. Cody, Applied Mathematics Division, Argonne National Laboratory, Argonne, IL, 60439, USA

Modified by Jens Keiner, Institute of Mathematics, University of Lübeck, 23560 Lübeck, Germany

Definition at line 192 of file fastsumS2.c.

5.21.2.2 innerProduct()

```
static double innerProduct (
             const double phi1,
             const double theta1,
             const double phi2,
             const double theta2 )  [inline], [static]
```

Computes the $\mathbb{R}^3$ standard inner product between two vectors on the unit sphere $\mathbb{S}^2$ given in spherical coordinates.

- phi1 The angle $\varphi_1 \in [-\pi, \pi)$ of the first vector
- theta1 The angle $\vartheta_1 \in [0, \pi]$ of the first vector
- phi2 The angle $\varphi_2 \in [-\pi, \pi)$ of the second vector
- theta2 The angle $\vartheta_2 \in [0, \pi]$ of the second vector

Returns

The inner product $\cos\vartheta_1 \cos\vartheta_2 + \sin\vartheta_1 \sin(\vartheta_2 \cos(\varphi_1 - \varphi_2)$

Author

Jens Keiner

Definition at line 449 of file fastsumS2.c.

5.21.2.3 poissonKernel()

```
static double poissonKernel (
             const double x,
             const double h )  [inline], [static]
```

Evaluates the Poisson kernel $Q_h : [-1, 1] \to \mathbb{R}$ at a node $x \in [-1, 1]$.

- x The node $x \in [-1, 1]$
- h The parameter $h \in (0, 1)$

Returns

The value of the Poisson kernel $Q_h(x)$ at the node x

Author

Jens Keiner

Definition at line 468 of file fastsumS2.c.

5.21.2.4 singularityKernel()

```
static double singularityKernel (
            const double x,
            const double h )  [inline], [static]
```

Evaluates the singularity kernel $S_h : [-1,1] \to \mathbb{R}$ at a node $x \in [-1,1]$.

- x The node $x \in [-1,1]$
- h The parameter $h \in (0,1)$

Returns

The value of the Poisson kernel $S_h(x)$ at the node x

Author

Jens Keiner

Definition at line 484 of file fastsumS2.c.

5.21.2.5 locallySupportedKernel()

```
static double locallySupportedKernel (
            const double x,
            const double h,
            const double lambda )  [inline], [static]
```

Evaluates the locally supported kernel $L_{h,\lambda} : [-1,1] \to \mathbb{R}$ at a node $x \in [-1,1]$.

- x The node $x \in [-1,1]$
- h The parameter $h \in (0,1)$
- lambda The parameter $\lambda \in \mathbb{N}_0$

Returns

The value of the locally supported kernel $L_{h,\lambda}(x)$ at the node x

Author

Jens Keiner

Definition at line 502 of file fastsumS2.c.

5.21.2.6 gaussianKernel()

```
static double gaussianKernel (
             const double x,
             const double sigma )  [inline], [static]
```

Evaluates the spherical Gaussian kernel $G_\sigma : [-1,1] \to \mathbb{R}$ at a node $x \in [-1,1]$.

- x The node $x \in [-1,1]$
- sigma The parameter $\sigma \in \mathbb{R}_+$

Returns

The value of the pherical Gaussian kernel $G_\sigma(x)$ at the node x

Author

Jens Keiner

Definition at line 520 of file fastsumS2.c.

5.21.2.7 main()

```
int main (
             int argc,
             char ** argv )
```

The main program.

Parameters

argc	The number of arguments
argv	An array containing the arguments as C-strings

Returns

Exit code

Author

Jens Keiner

Definition at line 535 of file fastsumS2.c.

5.22 Transforms in magnetic resonance imaging

Modules

- 2D transforms
- 3D transforms

5.22.1 Detailed Description

5.23 construct_data_2d

Functions

- static void construct (char ∗file, int N, int M)

  construct makes an 2d-nfft
- int **main** (int argc, char ∗∗argv)

5.23.1 Detailed Description

5.24 construct_data__inh_2d1d

Functions

- static void construct (char ∗file, int N, int M)

  construct
- int **main** (int argc, char ∗∗argv)

5.24.1 Detailed Description

5.25 construct_data_inh_3d

Functions

- static void construct (char ∗file, int N, int M)

  construct
- int **main** (int argc, char ∗∗argv)

5.25.1 Detailed Description

5.26 2D transforms

Modules

- construct_data_2d
- construct_data__inh_2d1d
- construct_data_inh_3d
- reconstruct_data_2d
- construct_data_gridding
- reconstruct_data__inh_2d1d
- reconstruct_data_inh_3d
- construct_data_inh_nnfft

5.26.1 Detailed Description

5.27 reconstruct_data_2d

Functions

- static void reconstruct (char ∗filename, int N, int M, int iteration, int weight)

  reconstruct makes an inverse 2d nfft
- int **main** (int argc, char ∗∗argv)

5.27.1 Detailed Description

5.28 construct_data_gridding

Functions

- static void reconstruct (char ∗filename, int N, int M, int weight)

  reconstruct makes a 2d-adjoint-nfft
- int **main** (int argc, char ∗∗argv)

5.28.1 Detailed Description

5.29 reconstruct_data__inh_2d1d

Functions

- static void **reconstruct** (char ∗filename, int N, int M, int iteration, int weight)
- int **main** (int argc, char ∗∗argv)

5.29.1 Detailed Description

5.30 reconstruct_data_inh_3d

Functions

- static void **reconstruct** (char ∗filename, int N, int M, int iteration, int weight)
- int **main** (int argc, char ∗∗argv)

5.30.1 Detailed Description

5.31 construct_data_inh_nnfft

Functions

- static void reconstruct (char ∗filename, int N, int M, int iteration, int weight)

  reconstruct
- int **main** (int argc, char ∗∗argv)

5.31.1 Detailed Description

5.32 construct_data_1d2d

Functions

- static void construct (char ∗file, int N, int M, int Z, fftw_complex ∗mem)

  construct makes an 2d-nfft for every slice
- static void fft (int N, int M, int Z, fftw_complex ∗mem)

  fft makes an 1D-ftt for every knot through all layers
- static void read_data (int N, int M, int Z, fftw_complex ∗mem)

  read fills the memory with the file input_data_f.dat as the real part of f and with zeros for the imag part of f
- int **main** (int argc, char ∗∗argv)

5.32.1 Detailed Description

5.33 construct_data_3d

Functions

- static void **construct** (char ∗file, int N, int M, int Z)
- int **main** (int argc, char ∗∗argv)

5.33.1 Detailed Description

5.34 3D transforms

Modules

- construct_data_1d2d
- construct_data_3d
- reconstruct_data_1d2d
- reconstruct_data_3d
- reconstruct_data_gridding

5.34.1 Detailed Description

5.35 reconstruct_data_1d2d

Functions

- static void reconstruct (char ∗filename, int N, int M, int Z, int iteration, int weight, fftw_complex ∗mem)

  reconstruct makes an inverse 2d-nfft for every slice
- static void print (int N, int M, int Z, fftw_complex ∗mem)

  print writes the memory back in a file output_real.dat for the real part and output_imag.dat for the imaginary part
- int **main** (int argc, char ∗∗argv)

5.35.1 Detailed Description

5.36 reconstruct_data_3d

Functions

- static void reconstruct (char ∗filename, int N, int M, int Z, int iteration, int weight)

  reconstruct makes an inverse 3d-nfft
- int **main** (int argc, char ∗∗argv)

5.36.1 Detailed Description

5.37 reconstruct_data_gridding

Functions

- static void reconstruct (char ∗filename, int N, int M, int Z, int weight, fftw_complex ∗mem)

  reconstruct makes an 2d-adjoint-nfft for every slice
- static void print (int N, int M, int Z, fftw_complex ∗mem)

  print writes the memory back in a file output_real.dat for the real part and output_imag.dat for the imaginary part
- int **main** (int argc, char ∗∗argv)

5.37.1 Detailed Description

5.38 Polar FFT

Modules

- linogram_fft_test
- mpolar_fft_test
- polar_fft_test

5.38.1 Detailed Description

5.39 linogram_fft_test

Functions

- static int linogram_grid (int T, int rr, NFFT_R ∗x, NFFT_R ∗w)

  Generates the points x with weights w for the linogram grid with T slopes and R offsets.
- static int linogram_dft (NFFT_C ∗f_hat, int NN, NFFT_C ∗f, int T, int rr, int m)

  discrete pseudo-polar FFT
- static int linogram_fft (NFFT_C ∗f_hat, int NN, NFFT_C ∗f, int T, int rr, int m)

  NFFT-based pseudo-polar FFT.
- static int inverse_linogram_fft (NFFT_C ∗f, int T, int rr, NFFT_C ∗f_hat, int NN, int max_i, int m)

  NFFT-based inverse pseudo-polar FFT.
- static int comparison_fft (FILE ∗fp, int N, int T, int rr)

  Comparison of the FFTW, linogram FFT, and inverse linogram FFT.
- int main (int argc, char ∗∗argv)

  test program for various parameters

Variables

- NFFT_R **GLOBAL_elapsed_time**

5.39.1 Detailed Description

5.40 mpolar_fft_test

Functions

- static int mpolar_grid (int T, int S, NFFT_R *x, NFFT_R *w)

  Generates the points $x_{t,j}$ with weights $w_{t,j}$ for the modified polar grid with T angles and R offsets.
- static int mpolar_dft (NFFT_C *f_hat, int NN, NFFT_C *f, int T, int S, int m)

  discrete mpolar FFT
- static int mpolar_fft (NFFT_C *f_hat, int NN, NFFT_C *f, int T, int S, int m)

  NFFT-based mpolar FFT.
- static int inverse_mpolar_fft (NFFT_C *f, int T, int S, NFFT_C *f_hat, int NN, int max_i, int m)

  inverse NFFT-based mpolar FFT
- static int comparison_fft (FILE *fp, int N, int T, int S)

  Comparison of the FFTW, mpolar FFT, and inverse mpolar FFT.
- int main (int argc, char **argv)

  test program for various parameters

Variables

- NFFT_R **GLOBAL_elapsed_time**

5.40.1 Detailed Description

5.40.2 Function Documentation

5.40.2.1 mpolar_grid()

```
static int mpolar_grid (
            int T,
            int S,
            NFFT_R * x,
            NFFT_R * w )  [static]
```

Generates the points $x_{t,j}$ with weights $w_{t,j}$ for the modified polar grid with T angles and R offsets.

We add more concentric circles to the polar grid and exclude those nodes not located in the unit square, i.e.,

$$x_{t,j} := r_j \left(\cos\theta_t, \sin\theta_t\right)^\top , \qquad (j,t)^\top \in I_{\sqrt{2}R} \times I_T \,.$$

with r_j and θ_t as for the polar grid. The number of nodes for the modified polar grid can be estimated as $M \approx \frac{4}{\pi}\log(1+\sqrt{2})TR$.

Definition at line 58 of file mpolar_fft_test.c.

5.41 polar_fft_test

Functions

- static int polar_grid (int T, int S, NFFT_R *x, NFFT_R *w)

  Generates the points $x_{t,j}$ with weights $w_{t,j}$ for the polar grid with T angles and R offsets.
- static int polar_dft (NFFT_C *f_hat, int NN, NFFT_C *f, int T, int S, int m)

  discrete polar FFT
- static int polar_fft (NFFT_C *f_hat, int NN, NFFT_C *f, int T, int S, int m)

  NFFT-based polar FFT.
- static int inverse_polar_fft (NFFT_C *f, int T, int S, NFFT_C *f_hat, int NN, int max_i, int m)

  inverse NFFT-based polar FFT
- int main (int argc, char **argv)

  test program for various parameters

5.41.1 Detailed Description

5.41.2 Function Documentation

5.41.2.1 polar_grid()

```
static int polar_grid (
            int T,
            int S,
            NFFT_R * x,
            NFFT_R * w )  [static]
```

Generates the points $x_{t,j}$ with weights $w_{t,j}$ for the polar grid with T angles and R offsets.

The nodes of the polar grid lie on concentric circles around the origin. They are given for $(j,t)^\top \in I_R \times I_T$ by a signed radius $r_j := \frac{j}{R} \in [-\frac{1}{2}, \frac{1}{2})$ and an angle $\theta_t := \frac{\pi t}{T} \in [-\frac{\pi}{2}, \frac{\pi}{2})$ as

$$x_{t,j} := r_j \left(\cos\theta_t, \sin\theta_t\right)^\top .$$

The total number of nodes is $M = TR$, whereas the origin is included multiple times.

Weights are introduced to compensate for local sampling density variations. For every point in the sampling set, we associate a small surrounding area. In case of the polar grid, we choose small ring segments. The area of such a ring segment around $x_{t,j}$ ($j \neq 0$) is

$$w_{t,j} = \frac{\pi}{2TR^2}\left(\left(|j| + \frac{1}{2}\right)^2 - \left(|j| - \frac{1}{2}\right)^2\right) = \frac{\pi|j|}{TR^2} .$$

The area of the small circle of radius $\frac{1}{2R}$ around the origin is $\frac{\pi}{4R^2}$. Divided by the multiplicity of the origin in the sampling set, we get $w_{t,0} := \frac{\pi}{4TR^2}$. Thus, the sum of all weights is $\frac{\pi}{4}(1 + \frac{1}{R^2})$ and we divide by this value for normalization.

Definition at line 73 of file polar_fft_test.c.

5.42 Fast evaluation of quadrature formulae on the sphere

Modules

- quadratureS2_test

5.42.1 Detailed Description

5.43 quadratureS2_test

Enumerations

- enum boolean { **NO** = 0, **YES** = 1 }

  Enumeration for parameter values.
- enum testtype { **ERROR** = 0, **TIMING** = 1 }

  Enumeration for test modes.
- enum gridtype {
  GRID_GAUSS_LEGENDRE = 0, **GRID_CLENSHAW_CURTIS** = 1, **GRID_HEALPIX** = 2, **GRID_EQUIDI↩**
  STRIBUTION = 3,
  GRID_EQUIDISTRIBUTION_UNIFORM = 4 }

  Enumeration for quadrature grid types.
- enum functiontype {
  FUNCTION_RANDOM_BANDLIMITED = 0, **FUNCTION_F1** = 1, **FUNCTION_F2** = 2, **FUNCTION_F3** = 3,
  FUNCTION_F4 = 4, **FUNCTION_F5** = 5, **FUNCTION_F6** = 6 }

  Enumeration for test functions.
- enum modes { **USE_GRID** = 0, **RANDOM** = 1 }

  TODO Add comment here.

Functions

- int main (int argc, char ∗∗argv)

  The main program.

5.43.1 Detailed Description

5.43.2 Function Documentation

5.43.2.1 main()

```
int main (
            int argc,
            char ** argv )
```

The main program.

Parameters

argc	The number of arguments
argv	An array containing the arguments as C-strings

Returns

Exit code

Definition at line 69 of file quadratureS2.c.

References nfsft_wisdom::threshold.

Chapter 6

Data Structure Documentation

6.1 fastsum_plan_ Struct Reference

plan for fast summation algorithm

`#include <fastsum.h>`

Public Member Functions

- NFFT (plan) mv1
  source nfft plan
- NFFT (plan) mv2
  target nfft plan
- **FFTW** (plan) fft_plan

Data Fields

- int d
  api
- int N_total
  number of source knots
- int M_total
  number of target knots
- C ∗ alpha
  source coefficients
- C ∗ f
  target evaluations
- R ∗ x
  source knots in d-ball with radius 1/4-eps_b/2
- R ∗ y

target knots in d-ball with radius 1/4-eps_b/2

- kernel k

  kernel function

- R * kernel_param

  parameters for kernel function

- unsigned flags

  flags precomp.

- C * pre_K

  internal

- int n

  FS__ - fast summation.

- C * b

  expansion coefficients

- C * f_hat

  Fourier coefficients of nfft plans.

- int p

  degree of smoothness of regularization

- R eps_I

  inner boundary

- R eps_B

  outer boundary

- int Ad

  near field

- C * Add

  spline values

- int **box_count**
- int **box_count_per_dim**
- int * **box_offset**
- R * **box_x**
- C * **box_alpha**
- int * permutation_x_alpha

  permutation vector of source nodes if STORE_PERMUTATION_X_ALPHA is set

- R MEASURE_TIME_t [8]

  Measured time for each step if MEASURE_TIME is set.

6.1.1 Detailed Description

plan for fast summation algorithm

Definition at line 82 of file fastsum.h.

The documentation for this struct was generated from the following file:

- fastsum.h

6.2 fgt_plan Struct Reference

Structure for the Gauss transform.

Data Fields

- int N
  number of source nodes
- int M
  number of target nodes
- NFFT_C * alpha
  source coefficients
- NFFT_C * f
  target evaluations
- unsigned flags
  flags for precomputation and approximation type
- NFFT_C sigma
  parameter of the Gaussian
- NFFT_R * x
  source nodes in $[-1/4, 1/4]$
- NFFT_R * y
  target nodes in $[-1/4, 1/4]$
- NFFT_C * pre_cexp
  precomputed values for dgt
- int n
  expansion degree
- NFFT_R p
  period, at least 1
- NFFT_C * b
  expansion coefficients

6.2.1 Detailed Description

Structure for the Gauss transform.

Definition at line 66 of file fastgauss.c.

The documentation for this struct was generated from the following file:

- fastgauss.c

6.3 fpt_data_ Struct Reference

Holds data for a single cascade summation.

`#include <fpt.h>`

Data Fields

- fpt_step ** steps

  The cascade summation steps

- int k_start

  TODO Add comment here.

- double * alphaN

  TODO Add comment here.

- double * betaN

  TODO Add comment here.

- double * gammaN

  TODO Add comment here.

- double alpha_0

  TODO Add comment here.

- double beta_0

  TODO Add comment here.

- double gamma_m1

  TODO Add comment here.

- double * _alpha

  < *TODO Add comment here.*

- double * _beta

  TODO Add comment here.

- double * _gamma

  TODO Add comment here.

- bool **precomputed**

6.3.1 Detailed Description

Holds data for a single cascade summation.

Definition at line 45 of file fpt.h.

6.3.2 Field Documentation

6.3.2.1 k_start

```
int fpt_data_::k_start
```

TODO Add comment here.

Definition at line 48 of file fpt.h.

6.3.2.2 alphaN

```
double* fpt_data_::alphaN
```

TODO Add comment here.

Definition at line 49 of file fpt.h.

6.3.2.3 betaN

```
double* fpt_data_::betaN
```

TODO Add comment here.

Definition at line 50 of file fpt.h.

6.3.2.4 gammaN

```
double* fpt_data_::gammaN
```

TODO Add comment here.

Definition at line 51 of file fpt.h.

6.3.2.5 alpha_0

```
double fpt_data_::alpha_0
```

TODO Add comment here.

Definition at line 52 of file fpt.h.

6.3.2.6 beta_0

```
double fpt_data_::beta_0
```

TODO Add comment here.

Definition at line 53 of file fpt.h.

6.3.2.7 gamma_m1

```
double fpt_data_::gamma_m1
```

TODO Add comment here.

Definition at line 54 of file fpt.h.

6.3.2.8 _alpha

```
double* fpt_data_::_alpha
```

< TODO Add comment here.

TODO Add comment here.

Definition at line 56 of file fpt.h.

6.3.2.9 _beta

```
double* fpt_data_::_beta
```

TODO Add comment here.

Definition at line 57 of file fpt.h.

6.3.2.10 _gamma

```
double* fpt_data_::_gamma
```

TODO Add comment here.

Definition at line 58 of file fpt.h.

The documentation for this struct was generated from the following file:

- fpt.h

6.4 fpt_set_s_ Struct Reference

Holds data for a set of cascade summations.

```
#include <fpt.h>
```

Data Fields

- unsigned int flags
  The flags
- int M
  The number of DPT transforms
- int N
  The transform length.
- int t
  The exponent of N
- fpt_data * dpt
  The DPT transform data
- double ** xcvecs

Array of pointers to arrays containing the Chebyshev nodes

- double * xc

  Array for Chebychev-nodes.
- double _Complex * **temp**
- double _Complex * **work**
- double _Complex * **result**
- double _Complex * **vec3**
- double _Complex * **vec4**
- double _Complex * **z**
- fftw_plan * plans_dct3

  Transform plans for the fftw library
- fftw_plan * plans_dct2

  Transform plans for the fftw library
- fftw_r2r_kind * kinds

  Transform kinds for fftw library
- fftw_r2r_kind * kindsr

  Transform kinds for fftw library
- double * **xc_slow**

6.4.1 Detailed Description

Holds data for a set of cascade summations.

Definition at line 65 of file fpt.h.

6.4.2 Field Documentation

6.4.2.1 N

```
int fpt_set_s_::N
```

The transform length.

Must be a power of two.

Definition at line 69 of file fpt.h.

Referenced by fpt_init().

6.4.2.2 xc

```
double* fpt_set_s_::xc
```

Array for Chebychev-nodes.

Definition at line 76 of file fpt.h.

The documentation for this struct was generated from the following file:

- fpt.h

6.5 fpt_step_ Struct Reference

Holds data for a single multiplication step in the cascade summation.

```
#include <fpt.h>
```

Data Fields

- bool stable
  Indicates if the values contained represent a fast or a slow stabilized step.
- int Ns
  TODO Add comment here.
- int ts
  TODO Add comment here.
- double * a
  The matrix components
- double **g**

6.5.1 Detailed Description

Holds data for a single multiplication step in the cascade summation.

Definition at line 30 of file fpt.h.

6.5.2 Field Documentation

6.5.2.1 stable

```
bool fpt_step_::stable
```

Indicates if the values contained represent a fast or a slow stabilized step.

Definition at line 32 of file fpt.h.

6.5.2.2 Ns

```
int fpt_step_::Ns
```

TODO Add comment here.

Definition at line 35 of file fpt.h.

6.5.2.3 ts

```
int fpt_step_::ts
```

TODO Add comment here.

Definition at line 36 of file fpt.h.

The documentation for this struct was generated from the following file:

- fpt.h

6.6 mri_inh_2d1d_plan Struct Reference

6.6.1 Detailed Description

The structure for the transform plan.

The documentation for this struct was generated from the following file:

- mri.dox

6.7 mri_inh_3d_plan Struct Reference

6.7.1 Detailed Description

The structure for the transform plan.

The documentation for this struct was generated from the following file:

- mri.dox

6.8 mrif_inh_2d1d_plan Struct Reference

Data Fields

- NFFT_INT N_total

  Total number of Fourier coefficients.
- NFFT_INT M_total

  Total number of samples.
- fftwf_complex ∗ f_hat

  Fourier coefficients.
- fftwf_complex ∗ f

  Samples.
- void(∗ mv_trafo)(void ∗)

  Transform.
- void(∗ mv_adjoint)(void ∗)

  Adjoint transform.
- nfftf_plan **plan**
- int **N3**
- float **sigma3**
- float ∗ **t**
- float ∗ **w**

6.8.1 Detailed Description

Definition at line 513 of file nfft3.h.

The documentation for this struct was generated from the following file:

- nfft3.h

6.9 mrif_inh_3d_plan Struct Reference

Data Fields

- NFFT_INT N_total
  Total number of Fourier coefficients.
- NFFT_INT M_total
  Total number of samples.
- fftwf_complex * f_hat
  Fourier coefficients.
- fftwf_complex * f
  Samples.
- void(* mv_trafo)(void *)
  Transform.
- void(* mv_adjoint)(void *)
  Adjoint transform.
- nfftf_plan **plan**
- int **N3**
- float **sigma3**
- float * **t**
- float * **w**

6.9.1 Detailed Description

Definition at line 513 of file nfft3.h.

The documentation for this struct was generated from the following file:

- nfft3.h

6.10 mril_inh_3d_plan Struct Reference

Data Fields

- NFFT_INT N_total
  Total number of Fourier coefficients.
- NFFT_INT M_total
  Total number of samples.
- fftwl_complex * f_hat
  Fourier coefficients.
- fftwl_complex * f
  Samples.
- void(* mv_trafo)(void *)
  Transform.
- void(* mv_adjoint)(void *)
  Adjoint transform.
- nfftl_plan **plan**
- int **N3**
- long double **sigma3**
- long double * **t**
- long double * **w**

6.10.1 Detailed Description

Definition at line 514 of file nfft3.h.

The documentation for this struct was generated from the following file:

- nfft3.h

6.11 nfct_plan Struct Reference

6.11.1 Detailed Description

NFCT transform plan

The documentation for this struct was generated from the following file:

- nfct.dox

6.12 nfctf_plan Struct Reference

data structure for an NFCT (nonequispaced fast cosine transform) plan with float precision

`#include <nfft3.h>`

Public Member Functions

- FFTW_MANGLE_FLOAT (plan) my_fftw_r2r_plan
  fftw_plan
- FFTW_MANGLE_FLOAT (r2r_kind) *r2r_kind
  r2r transform type (DCT-I)

Data Fields

- NFFT_INT N_total
  Total number of Fourier coefficients.
- NFFT_INT M_total
  Total number of samples.
- float * f_hat
  Fourier coefficients.
- float * f
  Samples.
- void(* mv_trafo)(void *)
  Transform.
- void(* mv_adjoint)(void *)
  Adjoint transform.
- NFFT_INT d

dimension, rank

- NFFT_INT * N

  cut-off-frequencies (kernel)
- NFFT_INT * n

  length of DCT-I
- NFFT_INT n_total

  Combined total length of FFTW transforms.
- float * sigma

  oversampling-factor
- NFFT_INT m

  cut-off parameter in time-domain
- float * b

  shape parameters
- NFFT_INT K

  Number of equispaced samples of window function.
- unsigned flags

  flags for precomputation, malloc
- unsigned fftw_flags

  flags for the fftw
- float * x

  nodes (in time/spatial domain)
- double MEASURE_TIME_t [3]

  measured time for each step
- float ** c_phi_inv

  precomputed data, matrix D
- float * psi

  precomputed data, matrix B
- NFFT_INT size_psi

  only for thin B
- NFFT_INT * psi_index_g

  only for thin B
- NFFT_INT * psi_index_f

  only for thin B
- float * **g**
- float * **g_hat**
- float * g1

  input of fftw
- float * g2

  output of fftw
- float * spline_coeffs

  input for de Boor algorithm, if B_SPLINE or SINC_2m is defined

6.12.1 Detailed Description

data structure for an NFCT (nonequispaced fast cosine transform) plan with float precision

Definition at line 271 of file nfft3.h.

6.12.2 Field Documentation

6.12.2.1 K

`NFFT_INT nfctf_plan::K`

Number of equispaced samples of window function.

Used for flag PRE_LIN_PSI.

Definition at line 271 of file nfft3.h.

The documentation for this struct was generated from the following file:

- nfft3.h

6.13 nfft_plan Struct Reference

6.13.1 Detailed Description

NFFT transform plan

The documentation for this struct was generated from the following file:

- nfft.dox

6.14 nfftf_mv_plan_complex Struct Reference

Data Fields

- NFFT_INT N_total
  Total number of Fourier coefficients.
- NFFT_INT M_total
  Total number of samples.
- fftwf_complex * f_hat
  Fourier coefficients.
- fftwf_complex * f
  Samples.
- void(* mv_trafo)(void *)
  Transform.
- void(* mv_adjoint)(void *)
  Adjoint transform.

6.14.1 Detailed Description

Definition at line 179 of file nfft3.h.

The documentation for this struct was generated from the following file:

- nfft3.h

6.15 nfftf_mv_plan_double Struct Reference

Data Fields

- NFFT_INT N_total
  Total number of Fourier coefficients.
- NFFT_INT M_total
  Total number of samples.
- float * f_hat
  Fourier coefficients.
- float * f
  Samples.
- void(* mv_trafo)(void *)
  Transform.
- void(* mv_adjoint)(void *)
  Adjoint transform.

6.15.1 Detailed Description

Definition at line 179 of file nfft3.h.

The documentation for this struct was generated from the following file:

- nfft3.h

6.16 nfftf_plan Struct Reference

data structure for an NFFT (nonequispaced fast Fourier transform) plan with float precision

`#include <nfft3.h>`

Public Member Functions

- FFTW_MANGLE_FLOAT (plan) my_fftw_plan1
  Forward FFTW plan.
- FFTW_MANGLE_FLOAT (plan) my_fftw_plan2
  Backward FFTW plan.

Data Fields

- NFFT_INT N_total

  Total number of Fourier coefficients.
- NFFT_INT M_total

  Total number of samples.
- fftwf_complex * f_hat

  Fourier coefficients.
- fftwf_complex * f

  Samples.
- void(* mv_trafo)(void *)

  Transform.
- void(* mv_adjoint)(void *)

  Adjoint transform.
- NFFT_INT d

  Dimension (rank).
- NFFT_INT * N

  Multi-bandwidth.
- float * sigma

  Oversampling factor.
- NFFT_INT * n

  Length of FFTW transforms.
- NFFT_INT n_total

  Combined total length of FFTW transforms.
- NFFT_INT m

  Cut-off parameter for window function.
- float * b

  Shape parameter for window function.
- NFFT_INT K

  Number of equispaced samples of window function.
- unsigned flags

  Flags for precomputation, (de)allocation, and FFTW usage, default setting is PRE_PHI_HUT | PRE_PSI | MALLO↩ C_X | MALLOC_F_HAT | MALLOC_F | FFTW_INIT | FFT_OUT_OF_PLACE.
- unsigned fftw_flags

  Flags for the FFTW, default is FFTW_ESTIMATE | FFTW_DESTROY_INPUT.
- float * x

  Nodes in time/spatial domain, size is dM floats.
- float MEASURE_TIME_t [3]

  Measured time for each step if MEASURE_TIME is set.
- float ** c_phi_inv

  Precomputed data for the diagonal matrix D, size is $N_0 + \cdots + N_{d-1}$ doubles.
- float * psi

  Precomputed data for the sparse matrix B, size depends on precomputation scheme.
- NFFT_INT * psi_index_g

  Indices in source/target vector for PRE_FULL_PSI.
- NFFT_INT * psi_index_f

  Indices in source/target vector for PRE_FULL_PSI.
- fftwf_complex * g

  Oversampled vector of samples, size is n_total double complex.
- fftwf_complex * g_hat

  Zero-padded vector of Fourier coefficients, size is n_total fftw_complex.

- fftwf_complex * g1

  Input of fftw.
- fftwf_complex * g2

  Output of fftw.
- float * spline_coeffs

  Input for de Boor algorithm if B_SPLINE or SINC_POWER is defined.
- NFFT_INT * index_x

  Index array for nodes x used when flag NFFT_SORT_NODES is set.

6.16.1 Detailed Description

data structure for an NFFT (nonequispaced fast Fourier transform) plan with float precision

Definition at line 179 of file nfft3.h.

6.16.2 Field Documentation

6.16.2.1 n

```
NFFT_INT* nfftf_plan::n
```

Length of FFTW transforms.

This is equal to sigma*N. The default is to use a power of two that satifies $2 \leq \sigma < 4$.

Definition at line 179 of file nfft3.h.

6.16.2.2 m

```
NFFT_INT nfftf_plan::m
```

Cut-off parameter for window function.

Default values for the different window functions are - 6 (KAISER_BESSEL), - 9 (SINC_POWER), - 11 (B_SPLINE), - 12 (GAUSSIAN)

Definition at line 179 of file nfft3.h.

6.16.2.3 K

```
NFFT_INT nfftf_plan::K
```

Number of equispaced samples of window function.

Used for flag PRE_LIN_PSI.

Definition at line 179 of file nfft3.h.

The documentation for this struct was generated from the following file:

- nfft3.h

6.17 nfftl_mv_plan_double Struct Reference

Data Fields

- NFFT_INT N_total
  Total number of Fourier coefficients.
- NFFT_INT M_total
  Total number of samples.
- long double * f_hat
  Fourier coefficients.
- long double * f
  Samples.
- void(* mv_trafo)(void *)
  Transform.
- void(* mv_adjoint)(void *)
  Adjoint transform.

6.17.1 Detailed Description

Definition at line 180 of file nfft3.h.

The documentation for this struct was generated from the following file:

- nfft3.h

6.18 nfftl_plan Struct Reference

data structure for an NFFT (nonequispaced fast Fourier transform) plan with long double precision

```
#include <nfft3.h>
```

Public Member Functions

- FFTW_MANGLE_LONG_DOUBLE (plan) my_fftw_plan1

  Forward FFTW plan.
- FFTW_MANGLE_LONG_DOUBLE (plan) my_fftw_plan2

  Backward FFTW plan.

Data Fields

- NFFT_INT N_total

  Total number of Fourier coefficients.
- NFFT_INT M_total

  Total number of samples.
- fftwl_complex $*$ f_hat

  Fourier coefficients.
- fftwl_complex $*$ f

  Samples.
- void($*$ mv_trafo)(void $*$)

  Transform.
- void($*$ mv_adjoint)(void $*$)

  Adjoint transform.
- NFFT_INT d

  Dimension (rank).
- NFFT_INT $*$ N

  Multi-bandwidth.
- long double $*$ sigma

  Oversampling factor.
- NFFT_INT $*$ n

  Length of FFTW transforms.
- NFFT_INT n_total

  Combined total length of FFTW transforms.
- NFFT_INT m

  Cut-off parameter for window function.
- long double $*$ b

  Shape parameter for window function.
- NFFT_INT K

  Number of equispaced samples of window function.
- unsigned flags

  Flags for precomputation, (de)allocation, and FFTW usage, default setting is PRE_PHI_HUT | PRE_PSI | MALLO↩ C_X | MALLOC_F_HAT | MALLOC_F | FFTW_INIT | FFT_OUT_OF_PLACE.
- unsigned fftw_flags

  Flags for the FFTW, default is FFTW_ESTIMATE | FFTW_DESTROY_INPUT.
- long double $*$ x

  Nodes in time/spatial domain, size is dM long doubles.
- long double MEASURE_TIME_t [3]

  Measured time for each step if MEASURE_TIME is set.
- long double $**$ c_phi_inv

  Precomputed data for the diagonal matrix D, size is $N_0 + \cdots + N_{d-1}$ doubles.
- long double $*$ psi

  Precomputed data for the sparse matrix B, size depends on precomputation scheme.

- NFFT_INT ∗ psi_index_g
  Indices in source/target vector for PRE_FULL_PSI.
- NFFT_INT ∗ psi_index_f
  Indices in source/target vector for PRE_FULL_PSI.
- fftwl_complex ∗ g
  Oversampled vector of samples, size is n_total double complex.
- fftwl_complex ∗ g_hat
  Zero-padded vector of Fourier coefficients, size is n_total fftw_complex.
- fftwl_complex ∗ g1
  Input of fftw.
- fftwl_complex ∗ g2
  Output of fftw.
- long double ∗ spline_coeffs
  Input for de Boor algorithm if B_SPLINE or SINC_POWER is defined.
- NFFT_INT ∗ index_x
  Index array for nodes x used when flag NFFT_SORT_NODES is set.

6.18.1 Detailed Description

data structure for an NFFT (nonequispaced fast Fourier transform) plan with long double precision

Definition at line 180 of file nfft3.h.

6.18.2 Field Documentation

6.18.2.1 n

```
NFFT_INT* nfftl_plan::n
```

Length of FFTW transforms.

This is equal to sigma∗N. The default is to use a power of two that satifies $2 \leq \sigma < 4$.

Definition at line 180 of file nfft3.h.

6.18.2.2 m

```
NFFT_INT nfftl_plan::m
```

Cut-off parameter for window function.

Default values for the different window functions are - 6 (KAISER_BESSEL), - 9 (SINC_POWER), - 11 (B_SPLINE), - 12 (GAUSSIAN)

Definition at line 180 of file nfft3.h.

6.18.2.3 K

`NFFT_INT nfftl_plan::K`

Number of equispaced samples of window function.

Used for flag PRE_LIN_PSI.

Definition at line 180 of file nfft3.h.

The documentation for this struct was generated from the following file:

- nfft3.h

6.19 nfsft_plan Struct Reference

6.19.1 Detailed Description

NFSFT transform plan

The documentation for this struct was generated from the following file:

- nfsft.dox

6.20 nfsft_wisdom Struct Reference

Wisdom structure.

`#include <api.h>`

Data Fields

- bool initialized

  Indicates wether the structure has been initialized.
- unsigned int **flags**
- int N_MAX

  Stores precomputation flags.
- int T_MAX

  The logarithm /f$t = \log_2 N_{\text{max}}/f$ of the maximum bandwidth.
- double ∗ alpha

  Precomputed recursion coefficients /fα_k^n/f for /f$k = 0,/ldots, N_{\text{max}}; n=-k,/ldots,k/f$ of associated Legendre-functions /fP_k^n/f.
- double ∗ beta

  Precomputed recursion coefficients /fβ_k^n/f for /f$k = 0,/ldots, N_{\text{max}}; n=-k,/ldots,k/f$ of associated Legendre-functions /fP_k^n/f.
- double ∗ gamma

  Precomputed recursion coefficients /fγ_k^n/f for /f$k = 0,/ldots, N_{\text{max}}; n=-k,/ldots,k/f$ of associated Legendre-functions /fP_k^n/f.
- double threshold

  The threshold /fκ/f.
- fpt_set set

  Structure for discrete polynomial transform (DPT)

6.20.1 Detailed Description

Wisdom structure.

Definition at line 56 of file kernel/nfsft/api.h.

The documentation for this struct was generated from the following file:

- kernel/nfsft/api.h

6.21 nfsftf_plan Struct Reference

data structure for an NFSFT (nonequispaced fast spherical Fourier transform) plan with float precision

`#include <nfft3.h>`

Data Fields

- NFFT_INT N_total
  Total number of Fourier coefficients.
- NFFT_INT M_total
  Total number of samples.
- fftwf_complex * f_hat
  Fourier coefficients.
- fftwf_complex * f
  Samples.
- void(* mv_trafo)(void *)
  Transform.
- void(* mv_adjoint)(void *)
  Adjoint transform.
- int N
  the bandwidth N
- float * x
  the nodes $\mathbf{x}(m) = (x_1, x_2) \in [-\frac{1}{2}, \frac{1}{2}) \times [0, \frac{1}{2}]$ for $m = 0, \ldots, M-1$, $M \in \mathbb{N}$,
- int t
  the logarithm of NPT with respect to the basis 2
- unsigned int flags
  the planner flags
- nfftf_plan plan_nfft
  the internal NFFT plan
- fftwf_complex * f_hat_intern
  Internally used pointer to spherical Fourier coefficients.
- double MEASURE_TIME_t [3]
  Measured time for each step if MEASURE_TIME is set.

6.21.1 Detailed Description

data structure for an NFSFT (nonequispaced fast spherical Fourier transform) plan with float precision

Definition at line 559 of file nfft3.h.

The documentation for this struct was generated from the following file:

- nfft3.h

6.22 nfsoftf_plan_ Struct Reference

Data Fields

- NFFT_INT N_total
  Total number of Fourier coefficients.
- NFFT_INT M_total
  Total number of samples.
- fftwf_complex * f_hat
  Fourier coefficients.
- fftwf_complex * f
  Samples.
- void(* mv_trafo)(void *)
  Transform.
- void(* mv_adjoint)(void *)
  Adjoint transform.
- float * x
  input nodes
- fftwf_complex * wig_coeffs
  deprecated variable
- fftwf_complex * cheby
  deprecated variable
- fftwf_complex * aux
  deprecated variable
- int t
  the logarithm of NPT with respect to the basis 2
- unsigned int flags
  the planner flags
- nfftf_plan p_nfft
  the internal NFFT plan
- fptf_set * internal_fpt_set
  the internal FPT plan
- int nthreads
  the number of threads

6.22.1 Detailed Description

Definition at line 671 of file nfft3.h.

The documentation for this struct was generated from the following file:

- nfft3.h

6.23 nfst_plan Struct Reference

6.23.1 Detailed Description

Structure for a transform plan

The documentation for this struct was generated from the following file:

- nfst.dox

6.24 nfstf_plan Struct Reference

data structure for an NFST (nonequispaced fast sine transform) plan with float precision

`#include <nfft3.h>`

Public Member Functions

- FFTW_MANGLE_FLOAT (plan) my_fftw_r2r_plan
  fftw_plan forward
- FFTW_MANGLE_FLOAT (r2r_kind) *r2r_kind
  r2r transform type (dct-i)

Data Fields

- NFFT_INT N_total
  Total number of Fourier coefficients.
- NFFT_INT M_total
  Total number of samples.
- float * f_hat
  Fourier coefficients.
- float * f
  Samples.
- void(* mv_trafo)(void *)
  Transform.
- void(* mv_adjoint)(void *)
  Adjoint transform.
- NFFT_INT d

dimension, rank

- NFFT_INT ∗ N

  bandwidth

- NFFT_INT ∗ n

  length of DST-I

- NFFT_INT n_total

  Combined total length of FFTW transforms.

- float ∗ sigma

  oversampling-factor

- NFFT_INT m

  cut-off parameter in time-domain

- float ∗ b

  shape parameters

- NFFT_INT K

  Number of equispaced samples of window function.

- unsigned flags

  flags for precomputation, malloc

- unsigned fftw_flags

  flags for the fftw

- float ∗ x

  nodes (in time/spatial domain)

- double MEASURE_TIME_t [3]

  measured time for each step

- float ∗∗ c_phi_inv

  precomputed data, matrix D

- float ∗ psi

  precomputed data, matrix B

- NFFT_INT size_psi

  only for thin B

- NFFT_INT ∗ psi_index_g

  only for thin B

- NFFT_INT ∗ psi_index_f

  only for thin B

- float ∗ **g**
- float ∗ **g_hat**
- float ∗ g1

  input of fftw

- float ∗ g2

  output of fftw

- float ∗ spline_coeffs

  input for de Boor algorithm, if B_SPLINE or SINC_2m is defined

- float **nfstf_full_psi_eps**

6.24.1 Detailed Description

data structure for an NFST (nonequispaced fast sine transform) plan with float precision

Definition at line 349 of file nfft3.h.

6.24.2 Field Documentation

6.24.2.1 K

`NFFT_INT nfstf_plan::K`

Number of equispaced samples of window function.

Used for flag PRE_LIN_PSI.

Definition at line 349 of file nfft3.h.

The documentation for this struct was generated from the following file:

- nfft3.h

6.25 nnfft_plan Struct Reference

6.25.1 Detailed Description

NNFFT transform plan

The documentation for this struct was generated from the following file:

- nnfft.dox

6.26 nnfftf_plan Struct Reference

data structure for an NNFFT (nonequispaced in time and frequency fast Fourier transform) plan with float precision

`#include <nfft3.h>`

Data Fields

- NFFT_INT N_total
  Total number of Fourier coefficients.
- NFFT_INT M_total
  Total number of samples.
- fftwf_complex * f_hat
  Fourier coefficients.
- fftwf_complex * f
  Samples.
- void(* mv_trafo)(void *)
  Transform.
- void(* mv_adjoint)(void *)
  Adjoint transform.
- int d
  dimension, rank
- float * sigma
  oversampling-factor
- float * a
  *1 + 2*m/N1*
- int * N
  cut-off-frequencies
- int * N1
  *sigma*N*
- int * aN1
  *sigma*a*N*
- int m
  cut-off parameter in time-domain
- float * b
  shape parameters
- int K
  number of precomp.
- int aN1_total
  aN1_total=aN1[0] ...*
- nfftf_plan * direct_plan
  plan for the nfft
- unsigned nnfft_flags
  flags for precomputation, malloc
- int * n
  n=N1, for the window function
- float * x
  nodes (in time/spatial domain)
- float * v
  nodes (in fourier domain)
- float * c_phi_inv
  precomputed data, matrix D
- float * psi
  precomputed data, matrix B
- int size_psi
  only for thin B
- int * psi_index_g

only for thin B

- int ∗ psi_index_f

  only for thin B

- fftwf_complex ∗ **F**
- float ∗ spline_coeffs

  input for de Boor algorithm, if B_SPLINE or SINC_2m is defined

6.26.1 Detailed Description

data structure for an NNFFT (nonequispaced in time and frequency fast Fourier transform) plan with float precision

Definition at line 412 of file nfft3.h.

6.26.2 Field Documentation

6.26.2.1 K

```
int nnfftf_plan::K
```

number of precomp.

uniform psi

Definition at line 412 of file nfft3.h.

6.26.2.2 aN1_total

```
int nnfftf_plan::aN1_total
```

aN1_total=aN1[0]∗ ...

∗aN1[d-1]

Definition at line 412 of file nfft3.h.

The documentation for this struct was generated from the following file:

- nfft3.h

6.27 nsfft_plan Struct Reference

6.27.1 Detailed Description

Structure for a NFFT plan

The documentation for this struct was generated from the following file:

- nsfft.dox

6.28 nsfftf_plan Struct Reference

data structure for an NSFFT (nonequispaced sparse fast Fourier transform) plan with float precision

`#include <nfft3.h>`

Public Member Functions

- FFTW_MANGLE_FLOAT (plan) ∗set_fftw_plan1
  fftw plan for the nfft blocks
- FFTW_MANGLE_FLOAT (plan) ∗set_fftw_plan2
  fftw plan for the nfft blocks

Data Fields

- NFFT_INT N_total
  Total number of Fourier coefficients.
- NFFT_INT M_total
  Total number of samples.
- fftwf_complex ∗ f_hat
  Fourier coefficients.
- fftwf_complex ∗ f
  Samples.
- void(∗ mv_trafo)(void ∗)
  Transform.
- void(∗ mv_adjoint)(void ∗)
  Adjoint transform.
- int d
  dimension, rank; d = 2, 3
- int J
  problem size, i.e., d=2: N_total=(J+4) 2^(J+1) d=3: N_total=2^J 6(2^((J+1)/2+1)-1)+2^(3(J/2+1))
- int sigma
  oversampling-factor
- unsigned flags
  flags for precomputation, malloc
- int ∗ index_sparse_to_full
  index conversation, overflow for d=3, J=9!

- int r_act_nfft_plan
  index of current nfft block
- nfftf_plan * act_nfft_plan
  current nfft block
- nfftf_plan * center_nfft_plan
  central nfft block
- nfftf_plan * set_nfft_plan_1d
  nfft plans for short nffts
- nfftf_plan * set_nfft_plan_2d
  nfft plans for short nffts
- float * x_transposed
  coordinate exchanged nodes, d = 2
- float * **x_102**
- float * **x_201**
- float * **x_120**
- float * x_021
  coordinate exchanged nodes, d=3

6.28.1 Detailed Description

data structure for an NSFFT (nonequispaced sparse fast Fourier transform) plan with float precision

Definition at line 462 of file nfft3.h.

The documentation for this struct was generated from the following file:

- nfft3.h

6.29 s_param Struct Reference

Data Fields

- int **d**
- int **trafo_adjoint**
- int **N**
- int **M**
- double **sigma**
- int **m**
- int **flags**
- int **nfsft_flags**
- int **psi_flags**
- int **L**
- int **n**
- int **p**
- char * **kernel_name**
- R **c**
- R **eps_I**
- R **eps_B**

6.29.1 Detailed Description

Definition at line 120 of file nfft_benchomp.c.

The documentation for this struct was generated from the following files:

- nfft_benchomp.c
- nfsft_benchomp.c
- fastsum_benchomp.c

6.30 s_result Struct Reference

Data Fields

- int **nthreads**
- s_resval **resval** [6]

6.30.1 Detailed Description

Definition at line 138 of file nfft_benchomp.c.

The documentation for this struct was generated from the following files:

- nfft_benchomp.c
- nfsft_benchomp.c
- fastsum_benchomp.c

6.31 s_resval Struct Reference

Data Fields

- double **avg**
- double **min**
- double **max**
- R **avg**
- R **min**
- R **max**

6.31.1 Detailed Description

Definition at line 131 of file nfft_benchomp.c.

The documentation for this struct was generated from the following files:

- nfft_benchomp.c
- nfsft_benchomp.c
- fastsum_benchomp.c

6.32 s_testset Struct Reference

Data Fields

- s_param **param**
- s_result * **results**
- int **nresults**

6.32.1 Detailed Description

Definition at line 144 of file nfft_benchomp.c.

The documentation for this struct was generated from the following files:

- nfft_benchomp.c
- nfsft_benchomp.c
- fastsum_benchomp.c

6.33 solverf_plan_complex Struct Reference

data structure for an inverse NFFT plan with float precision

`#include <nfft3.h>`

Data Fields

- nfftf_mv_plan_complex * mv
  matrix vector multiplication
- unsigned flags
  iteration type
- float * w
  weighting factors
- float * w_hat
  damping factors
- fftwf_complex * y
  right hand side, samples
- fftwf_complex * f_hat_iter
  iterative solution
- fftwf_complex * r_iter
  iterated residual vector
- fftwf_complex * z_hat_iter
  residual of normal equation of first kind
- fftwf_complex * p_hat_iter
  search direction
- fftwf_complex * v_iter
  residual vector update
- float alpha_iter

step size for search direction

- float beta_iter

  step size for search correction
- float dot_r_iter

  weighted dotproduct of r_iter
- float dot_r_iter_old

  previous dot_r_iter
- float dot_z_hat_iter

  weighted dotproduct of z_hat_iter
- float dot_z_hat_iter_old

  previous dot_z_hat_iter
- float dot_p_hat_iter

  weighted dotproduct of p_hat_iter
- float dot_v_iter

  weighted dotproduct of v_iter

6.33.1 Detailed Description

data structure for an inverse NFFT plan with float precision

Definition at line 771 of file nfft3.h.

The documentation for this struct was generated from the following file:

- nfft3.h

6.34 solverf_plan_double Struct Reference

data structure for an inverse NFFT plan with float precision

`#include <nfft3.h>`

Data Fields

- nfftf_mv_plan_double * mv

  matrix vector multiplication

- unsigned flags

  iteration type
- float * w

  weighting factors
- float * w_hat

  damping factors
- float * y

  right hand side, samples
- float * f_hat_iter

  iterative solution
- float * r_iter

iterated residual vector

- float * z_hat_iter

  residual of normal equation of first kind

- float * p_hat_iter

  search direction

- float * v_iter

  residual vector update

- float alpha_iter

  step size for search direction

- float beta_iter

  step size for search correction

- float dot_r_iter

  weighted dotproduct of r_iter

- float dot_r_iter_old

  previous dot_r_iter

- float dot_z_hat_iter

  weighted dotproduct of z_hat_iter

- float dot_z_hat_iter_old

  previous dot_z_hat_iter

- float dot_p_hat_iter

  weighted dotproduct of p_hat_iter

- float dot_v_iter

  weighted dotproduct of v_iter

6.34.1 Detailed Description

data structure for an inverse NFFT plan with float precision

Definition at line 771 of file nfft3.h.

The documentation for this struct was generated from the following file:

- nfft3.h

6.35 solverl_plan_double Struct Reference

data structure for an inverse NFFT plan with long double precision

```
#include <nfft3.h>
```

Data Fields

- nfftl_mv_plan_double * mv
  matrix vector multiplication
- unsigned flags
  iteration type
- long double * w
  weighting factors
- long double * w_hat
  damping factors
- long double * y
  right hand side, samples
- long double * f_hat_iter
  iterative solution
- long double * r_iter
  iterated residual vector
- long double * z_hat_iter
  residual of normal equation of first kind
- long double * p_hat_iter
  search direction
- long double * v_iter
  residual vector update
- long double alpha_iter
  step size for search direction
- long double beta_iter
  step size for search correction
- long double dot_r_iter
  weighted dotproduct of r_iter
- long double dot_r_iter_old
  previous dot_r_iter
- long double dot_z_hat_iter
  weighted dotproduct of z_hat_iter
- long double dot_z_hat_iter_old
  previous dot_z_hat_iter
- long double dot_p_hat_iter
  weighted dotproduct of p_hat_iter
- long double dot_v_iter
  weighted dotproduct of v_iter

6.35.1 Detailed Description

data structure for an inverse NFFT plan with long double precision

Definition at line 772 of file nfft3.h.

The documentation for this struct was generated from the following file:

- nfft3.h

6.36 taylor_plan Struct Reference

6.36.1 Detailed Description

Definition at line 41 of file taylor_nfft.c.

The documentation for this struct was generated from the following file:

- taylor_nfft.c

6.37 window_funct_plan_ Struct Reference

window_funct_plan is a plan to use the window functions independent of the nfft

Data Fields

- int **d**
- int **m**
- int **n** [1]
- double **sigma** [1]
- double ∗ **b**

6.37.1 Detailed Description

window_funct_plan is a plan to use the window functions independent of the nfft

Definition at line 34 of file mri.c.

The documentation for this struct was generated from the following file:

- mri.c

Chapter 7

File Documentation

7.1 api.h File Reference

Header file with internal API of the NFSFT module.

```
#include "config.h"
#include "nfft3.h"
```

Data Structures

- struct nfsft_wisdom

  Wisdom structure.

Macros

- #define **BWEXP_MAX** 10
- #define **BW_MAX** 1024
- #define **ROW**(k) (k∗(wisdom.N_MAX+2))
- #define **ROWK**(k) (k∗(wisdom.N_MAX+2)+k)

Enumerations

- enum **bool** { **false** = 0, **true** = 1 }

7.1.1 Detailed Description

Header file with internal API of the NFSFT module.

Author

Jens Keiner

7.2 fastsum.c File Reference

Fast NFFT-based summation algorithm.

```
#include "config.h"
#include <stdlib.h>
#include <math.h>
#include "nfft3.h"
#include "fastsum.h"
#include "infft.h"
#include "kernels.h"
```

Functions

- static int max_i (int a, int b)

  max
- static R fak (int n)

  factorial
- static R binom (int n, int m)

  binomial coefficient
- static R BasisPoly (int m, int r, R xx)

  basis polynomial for regularized kernel
- C regkern (kernel k, R xx, int p, const R *param, R a, R b)

  regularized kernel with K_I arbitrary and K_B smooth to zero
- static C regkern1 (kernel k, R xx, int p, const R *param, R a, R b)

  regularized kernel with K_I arbitrary and K_B periodized (used in 1D)
- static C regkern3 (kernel k, R xx, int p, const R *param, R a, R b)

  regularized kernel for even kernels with K_I even and K_B mirrored
- C kubintkern (const R x, const C *Add, const int Ad, const R a)

  linear spline interpolation in near field with even kernels
- static C kubintkern1 (const R x, const C *Add, const int Ad, const R a)

  cubic spline interpolation in near field with arbitrary kernels
- static void quicksort (int d, int t, R *x, C *alpha, int *permutation_x_alpha, int N)

  quicksort algorithm for source knots and associated coefficients
- static void BuildBox (fastsum_plan *ths)

  initialize box-based search data structures
- static C calc_SearchBox (int d, R *y, R *x, C *alpha, int start, int end_lt, const C *Add, const int Ad, int p, R a, const kernel k, const R *param, const unsigned flags)

  inner computation function for box-based near field correction
- static C SearchBox (R *y, fastsum_plan *ths)

  box-based near field correction
- static void BuildTree (int d, int t, R *x, C *alpha, int *permutation_x_alpha, int N)

  recursive sort of source knots dimension by dimension to get tree structure
- static C SearchTree (const int d, const int t, const R *x, const C *alpha, const R *xmin, const R *xmax, const int N, const kernel k, const R *param, const int Ad, const C *Add, const int p, const unsigned flags)

  fast search in tree of source knots for near field computation
- static void **fastsum_precompute_kernel** (fastsum_plan *ths)
- void fastsum_init_guru_kernel (fastsum_plan *ths, int d, kernel k, R *param, unsigned flags, int nn, int p, R eps_I, R eps_B)

  initialize node independent part of fast summation plan

- void fastsum_init_guru_source_nodes (fastsum_plan *ths, int N_total, int nn_oversampled, int m)

  initialize source nodes dependent part of fast summation plan
- void fastsum_init_guru_target_nodes (fastsum_plan *ths, int M_total, int nn_oversampled, int m)

  initialize target nodes dependent part of fast summation plan
- void fastsum_init_guru (fastsum_plan *ths, int d, int N_total, int M_total, kernel k, R *param, unsigned flags, int nn, int m, int p, R eps_I, R eps_B)

  initialization of fastsum plan
- void fastsum_finalize_source_nodes (fastsum_plan *ths)

  finalization of fastsum plan
- void fastsum_finalize_target_nodes (fastsum_plan *ths)

  finalization of fastsum plan
- void fastsum_finalize_kernel (fastsum_plan *ths)

  finalization of fastsum plan
- void fastsum_finalize (fastsum_plan *ths)

  finalization of fastsum plan
- void fastsum_exact (fastsum_plan *ths)

  direct computation of sums
- void fastsum_precompute_source_nodes (fastsum_plan *ths)

  precomputation for fastsum
- void fastsum_precompute_target_nodes (fastsum_plan *ths)

  precomputation for fastsum
- void fastsum_precompute (fastsum_plan *ths)

  precomputation for fastsum
- void fastsum_trafo (fastsum_plan *ths)

  fast NFFT-based summation

7.2.1 Detailed Description

Fast NFFT-based summation algorithm.

Author

Markus Fenn

Date

2003-2006

7.3 fastsum.h File Reference

Header file for the fast NFFT-based summation algorithm.

```
#include "config.h"
#include "nfft3.h"
#include "infft.h"
```

Data Structures

- struct fastsum_plan_

  plan for fast summation algorithm

Macros

- #define X(name) NFFT(name)

  Include header for C99 complex datatype.
- #define **NF_KUB**
- #define EXACT_NEARFIELD (1U<< 0)

  Constant symbols.
- #define **NEARFIELD_BOXES** (1U<< 1)
- #define STORE_PERMUTATION_X_ALPHA (1U<< 2)

  If this flag is set, and eps_I > 0.0 and NEARFIELD_BOXES is not set, then the vector permutation_x_alpha is stored.

Typedefs

- typedef C(* **kernel**) (R, int, const R *)
- typedef struct fastsum_plan_ fastsum_plan

  plan for fast summation algorithm

Functions

- void fastsum_init_guru (fastsum_plan *ths, int d, int N_total, int M_total, kernel k, R *param, unsigned flags, int nn, int m, int p, R eps_I, R eps_B)

  initialization of fastsum plan
- void fastsum_init_guru_kernel (fastsum_plan *ths, int d, kernel k, R *param, unsigned flags, int nn, int p, R eps_I, R eps_B)

  initialize node independent part of fast summation plan
- void fastsum_init_guru_source_nodes (fastsum_plan *ths, int N_total, int nn_oversampled, int m)

  initialize source nodes dependent part of fast summation plan
- void fastsum_init_guru_target_nodes (fastsum_plan *ths, int M_total, int nn_oversampled, int m)

  initialize target nodes dependent part of fast summation plan
- void fastsum_finalize (fastsum_plan *ths)

  finalization of fastsum plan
- void fastsum_finalize_source_nodes (fastsum_plan *ths)

  finalization of fastsum plan
- void fastsum_finalize_target_nodes (fastsum_plan *ths)

  finalization of fastsum plan
- void fastsum_finalize_kernel (fastsum_plan *ths)

  finalization of fastsum plan
- void fastsum_exact (fastsum_plan *ths)

  direct computation of sums
- void fastsum_precompute_source_nodes (fastsum_plan *ths)

  precomputation for fastsum
- void fastsum_precompute_target_nodes (fastsum_plan *ths)

  precomputation for fastsum
- void fastsum_precompute (fastsum_plan *ths)

precomputation for fastsum

- void fastsum_trafo (fastsum_plan *ths)

  fast NFFT-based summation

- C regkern (kernel k, R xx, int p, const R *param, R a, R b)

  regularized kernel with K_I arbitrary and K_B smooth to zero

- C kubintkern (const R x, const C *Add, const int Ad, const R a)

  linear spline interpolation in near field with even kernels

7.3.1 Detailed Description

Header file for the fast NFFT-based summation algorithm.

reference: M. Fenn, G. Steidl, Fast NFFT based summation of radial functions. Sampl. Theory Signal Image Process., 3, 1-28, 2004.

Author

Markus Fenn

Date

2003-2006

7.4 fastsum_matlab.c File Reference

Simple test program for the fast NFFT-based summation algorithm, called by fastsum.m.

```
#include "config.h"
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include "fastsum.h"
#include "kernels.h"
#include "infft.h"
```

Functions

- int **main** (int argc, char **argv)

7.4.1 Detailed Description

Simple test program for the fast NFFT-based summation algorithm, called by fastsum.m.

Author

Markus Fenn

Date

2006

7.5 fastsum_test.c File Reference

Simple test program for the fast NFFT-based summation algorithm.

```
#include "config.h"
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include "fastsum.h"
#include "kernels.h"
#include "infft.h"
```

Functions

- int **main** (int argc, char ∗∗argv)

7.5.1 Detailed Description

Simple test program for the fast NFFT-based summation algorithm.

Author

Markus Fenn

Date

2006

7.6 flags.c File Reference

Testing the nfft.

```
#include "config.h"
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <stdlib.h>
#include "nfft3.h"
#include "infft.h"
```

Functions

- static void **flags_cp** (NFFT(plan) ∗dst, NFFT(plan) ∗src)
- static void **time_accuracy** (int d, int N, int M, int n, int m, unsigned test_ndft, unsigned test_pre_full_psi)
- static void **accuracy_pre_lin_psi** (int d, int N, int M, int n, int m, int K)
- int **main** (int argc, char ∗∗argv)

Variables

- unsigned **test_fg** =0
- unsigned **test_fftw** =0
- unsigned **test** =0

7.6.1 Detailed Description

Testing the nfft.

Author

Stefan Kunis

References: Time and Memory Requirements of the Nonequispaced FFT

7.7 fpt.c File Reference

Implementation file for the FPT module.

```
#include "config.h"
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <math.h>
#include "nfft3.h"
#include "infft.h"
#include "fpt.h"
```

Macros

- #define K_START_TILDE(x, y) (MAX(MIN(x,y-2),0))

  If defined, perform critical computations in three-term recurrence always in long double instead of using adaptive version that starts in double and switches to long double if required.
- #define K_END_TILDE(x, y) MIN(x,y-1)

  Maximum degree at top of a cascade.
- #define FIRST_L(x, y) (LRINT(floor((x)/(double)y)))

  Index of first block of four functions at level.
- #define LAST_L(x, y) (LRINT(ceil(((x)+1)/(double)y))-1)

  Index of last block of four functions at level.
- #define **N_TILDE**(y) (y-1)
- #define **IS_SYMMETRIC**(x, y, z) (x >= ((y-1.0)/z))
- #define **FPT_BREAK_EVEN** 4
- #define **ABUVXPWY_SYMMETRIC**(NAME, S1, S2)
- #define **ABUVXPWY_SYMMETRIC_1**(NAME, S1)
- #define **ABUVXPWY_SYMMETRIC_2**(NAME, S1)
- #define **FPT_DO_STEP**(NAME, M1_FUNCTION, M2_FUNCTION)
- #define **FPT_DO_STEP_TRANSPOSED**(NAME, M1_FUNCTION, M2_FUNCTION)

Functions

- static void **abuvxpwy** (double a, double b, double _Complex *u, double _Complex *x, double *v, double _Complex *y, double *w, int n)
- static void **abuvxpwy_symmetric1** (double a, double b, double _Complex *u, double _Complex *x, double *v, double _Complex *y, double *w, int n)
- static void **abuvxpwy_symmetric2** (double a, double b, double _Complex *u, double _Complex *x, double *v, double _Complex *y, double *w, int n)
- static void **abuvxpwy_symmetric1_1** (double a, double b, double _Complex *u, double _Complex *x, double *v, double _Complex *y, int n, double *xx)
- static void **abuvxpwy_symmetric1_2** (double a, double b, double _Complex *u, double _Complex *x, double *v, double _Complex *y, int n, double *xx)
- static void **abuvxpwy_symmetric2_1** (double a, double b, double _Complex *u, double _Complex *x, double _Complex *y, double *w, int n, double *xx)
- static void **abuvxpwy_symmetric2_2** (double a, double b, double _Complex *u, double _Complex *x, double _Complex *y, double *w, int n, double *xx)
- static void **auvxpwy** (double a, double _Complex *u, double _Complex *x, double *v, double _Complex *y, double *w, int n)
- static void **auvxpwy_symmetric** (double a, double _Complex *u, double _Complex *x, double *v, double _Complex *y, double *w, int n)
- static void **auvxpwy_symmetric_1** (double a, double _Complex *u, double _Complex *x, double *v, double _Complex *y, double *w, int n, double *xx)
- static void **auvxpwy_symmetric_2** (double a, double _Complex *u, double _Complex *x, double *v, double _Complex *y, double *w, int n, double *xx)
- static void **fpt_do_step** (double _Complex *a, double _Complex *b, double *a11, double *a12, double *a21, double *a22, double g, int tau, fpt_set set)
- static void **fpt_do_step_symmetric** (double _Complex *a, double _Complex *b, double *a11, double *a12, double *a21, double *a22, double g, int tau, fpt_set set)
- static void **fpt_do_step_symmetric_u** (double _Complex *a, double _Complex *b, double *a11, double *a12, double *a21, double *a22, double *x, double gam, int tau, fpt_set set)
- static void **fpt_do_step_symmetric_l** (double _Complex *a, double _Complex *b, double *a11, double *a12, double *a21, double *a22, double *x, double gam, int tau, fpt_set set)
- static void **fpt_do_step_t** (double _Complex *a, double _Complex *b, double *a11, double *a12, double *a21, double *a22, double g, int tau, fpt_set set)
- static void **fpt_do_step_t_symmetric** (double _Complex *a, double _Complex *b, double *a11, double *a12, double *a21, double *a22, double g, int tau, fpt_set set)
- static void **fpt_do_step_t_symmetric_u** (double _Complex *a, double _Complex *b, double *a11, double *a12, double *x, double gam, int tau, fpt_set set)
- static void **fpt_do_step_t_symmetric_l** (double _Complex *a, double _Complex *b, double *a21, double *a22, double *x, double gam, int tau, fpt_set set)
- static void **eval_clenshaw** (const double *x, double *y, int size, int k, const double *alpha, const double *beta, const double *gam)
- static void **eval_clenshaw2** (const double *x, double *z, double *y, int size1, int size, int k, const double *alpha, const double *beta, const double *gam)
- static int **eval_clenshaw_thresh2** (const double *x, double *z, double *y, int size, int k, const double *alpha, const double *beta, const double *gam, const double threshold)
- static void **eval_sum_clenshaw_fast** (const int N, const int M, const double _Complex *a, const double *x, double _Complex *y, const double *alpha, const double *beta, const double *gam, const double lambda)
- static void eval_sum_clenshaw_transposed (int N, int M, double _Complex *a, double *x, double _Complex *y, double _Complex *temp, double *alpha, double *beta, double *gam, double lambda)

  Clenshaw algorithm.
- static void **eval_sum_clenshaw_transposed_ld** (int N, int M, double _Complex *a, double *x, double _↩ Complex *y, double _Complex *temp, double *alpha, double *beta, double *gam, double lambda)
- fpt_set fpt_init (const int M, const int t, const unsigned int flags)
- void **fpt_precompute_1** (fpt_set set, const int m, int k_start)

- void **fpt_precompute_2** (fpt_set set, const int m, double *alpha, double *beta, double *gam, int k_start, const double threshold)
- void fpt_precompute (fpt_set set, const int m, double *alpha, double *beta, double *gam, int k_start, const double threshold)
- void **fpt_trafo_direct** (fpt_set set, const int m, const double _Complex *x, double _Complex *y, const int k_end, const unsigned int flags)
- void **fpt_trafo** (fpt_set set, const int m, const double _Complex *x, double _Complex *y, const int k_end, const unsigned int flags)
- void **fpt_transposed_direct** (fpt_set set, const int m, double _Complex *x, double _Complex *y, const int k_end, const unsigned int flags)
- void fpt_transposed (fpt_set set, const int m, double _Complex *x, double _Complex *y, const int k_end, const unsigned int flags)
- void **fpt_finalize** (fpt_set set)

7.7.1 Detailed Description

Implementation file for the FPT module.

Author

Jens Keiner

7.7.2 Macro Definition Documentation

7.7.2.1 K_START_TILDE

```
#define K_START_TILDE(
              x,
              y ) (MAX(MIN(x,y-2),0))
```

If defined, perform critical computations in three-term recurrence always in long double instead of using adaptive version that starts in double and switches to long double if required.

Minimum degree at top of a cascade

Definition at line 48 of file fpt.c.

7.7.2.2 ABUVXPWY_SYMMETRIC

```
#define ABUVXPWY_SYMMETRIC(
              NAME,
              S1,
              S2 )
```

Value:

```
static inline void NAME(double a, double b, double _Complex* u, double _Complex* x, \
  double* v, double _Complex* y, double* w, int n) \
{ \
  const int n2 = n>>1; \
  int l; double _Complex *u_ptr = u, *x_ptr = x, *y_ptr = y; \
  double *v_ptr = v, *w_ptr = w; \
  for (l = 0; l < n2; l++) \
    *u_ptr++ = a * (b * (*v_ptr++) * (*x_ptr++) + (*w_ptr++) * (*y_ptr++)); \
  v_ptr--; w_ptr--; \
  for (l = 0; l < n2; l++) \
    *u_ptr++ = a * (b * S1 * (*v_ptr--) * (*x_ptr++) + S2 * (*w_ptr--) * (*y_ptr++)); \
}
```

Definition at line 77 of file fpt.c.

7.7.2.3 ABUVXPWY_SYMMETRIC_1

```
#define ABUVXPWY_SYMMETRIC_1(
                NAME,
                S1 )
```

Value:

```
static inline void NAME(double a, double b, double _Complex* u, double _Complex* x, \
  double* v, double _Complex* y, int n, double *xx) \
{ \
  const int n2 = n»1; \
  int l; double _Complex *u_ptr = u, *x_ptr = x, *y_ptr = y; \
  double *v_ptr = v, *xx_ptr = xx; \
  for (l = 0; l < n2; l++, v_ptr++) \
    *u_ptr++ = a * (b * (*v_ptr) * (*x_ptr++) + ((*v_ptr)*(1.0+*xx_ptr++)) * (*y_ptr++)); \
  v_ptr--; \
  for (l = 0; l < n2; l++, v_ptr--) \
    *u_ptr++ = a * (b * S1 * (*v_ptr) * (*x_ptr++) + (S1 * (*v_ptr) * (1.0+*xx_ptr++)) * (*y_ptr++)); \
}
```

Definition at line 94 of file fpt.c.

7.7.2.4 ABUVXPWY_SYMMETRIC_2

```
#define ABUVXPWY_SYMMETRIC_2(
                NAME,
                S1 )
```

Value:

```
static inline void NAME(double a, double b, double _Complex* u, double _Complex* x, \
  double _Complex* y, double* w, int n, double *xx) \
{ \
  const int n2 = n»1; \
  int l; double _Complex *u_ptr = u, *x_ptr = x, *y_ptr = y; \
  double *w_ptr = w, *xx_ptr = xx; \
  for (l = 0; l < n2; l++, w_ptr++) \
    *u_ptr++ = a * (b * (*w_ptr/(1.0+*xx_ptr++)) * (*x_ptr++) + (*w_ptr) * (*y_ptr++)); \
  w_ptr--; \
  for (l = 0; l < n2; l++, w_ptr--) \
    *u_ptr++ = a * (b * (S1 * (*w_ptr)/(1.0+*xx_ptr++) ) * (*x_ptr++) + S1 * (*w_ptr) * (*y_ptr++)); \
}
```

Definition at line 111 of file fpt.c.

7.7.2.5 FPT_DO_STEP_TRANSPOSED

```
#define FPT_DO_STEP_TRANSPOSED(
                NAME,
                M1_FUNCTION,
                M2_FUNCTION )
```

Value:

```
static inline void NAME(double _Complex  *a, double _Complex *b, double *a11, \
  double *a12, double *a21, double *a22, double g, int tau, fpt_set set) \
{ \ \
  int length = 1«(tau+1); \ \
  double norm = 1.0/(length«1); \
  \
  /* Compute function values from Chebyshev-coefficients using a DCT-III. */ \
  fftw_execute_r2r(set->plans_dct3[tau-1],(double*)a,(double*)a); \
  fftw_execute_r2r(set->plans_dct3[tau-1],(double*)b,(double*)b); \
  \
  /* Perform matrix multiplication. */ \
  M1_FUNCTION(norm,g,set->z,a,a11,b,a21,length); \
  M2_FUNCTION(norm,g,b,a,a12,b,a22,length); \
  memcpy(a,set->z,length*sizeof(double _Complex)); \
  \
  /* Compute Chebyshev-coefficients using a DCT-II. */ \
  fftw_execute_r2r(set->plans_dct2[tau-1],(double*)a,(double*)a); \
  fftw_execute_r2r(set->plans_dct2[tau-1],(double*)b,(double*)b); \
}
```

Definition at line 335 of file fpt.c.

7.7.3 Function Documentation

7.7.3.1 eval_sum_clenshaw_transposed()

```
static void eval_sum_clenshaw_transposed (
            int N,
            int M,
            double _Complex * a,
            double * x,
            double _Complex * y,
            double _Complex * temp,
            double * alpha,
            double * beta,
            double * gam,
            double lambda )  [static]
```

Clenshaw algorithm.

Evaluates a sum of real-valued functions $P_k : \mathbb{R} \to \mathbb{R}$

$$f(x) = \sum_{k=0}^{N} a_k P_k(x) \quad (N \in \mathbb{N}_0)$$

obeying a three-term recurrence relation

$$P_{k+1}(x) = (alpha_k * x + beta_k) * P_k(x) + gamma_k P_{k-1}(x) \quad (alpha_k, beta_k, gamma_k \in \mathbb{R},\ k \geq 0)$$

with initial conditions $P_{-1}(x) := 0$, $P_0(x) := \lambda$ for given double _Complex coefficients $(a_k)_{k=0}^{N} \in \mathbb{C}^{N+1}$ at given nodes $(x_j)_{j=0}^{M} \in \mathbb{R}^{M+1}$, $M \in \mathbb{N}_0$.

Definition at line 717 of file fpt.c.

7.8 inverse_radon.c File Reference

NFFT-based discrete inverse Radon transform.

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>
#include <complex.h>
#include "nfft3mp.h"
```

Macros

- #define **NFFT_PRECISION_DOUBLE**
- #define KERNEL(r) (NFFT_K(1.0)-NFFT_M(fabs)((NFFT_R)(r))/((NFFT_R)S/2))

  define weights of kernel function for discrete Radon transform

Functions

- static int polar_grid (int T, int S, NFFT_R *x, NFFT_R *w)

  generates the points x with weights w for the polar grid with T angles and R offsets
- static int linogram_grid (int T, int S, NFFT_R *x, NFFT_R *w)

  generates the points x with weights w for the linogram grid with T slopes and R offsets
- static int inverse_radon_trafo (int(*gridfcn)(), int T, int S, NFFT_R *Rf, int NN, NFFT_R *f, int max_i)

  computes the inverse discrete Radon transform of Rf on the grid given by gridfcn() with T angles and R offsets by a NFFT-based CG-type algorithm
- int main (int argc, char **argv)

  simple test program for the inverse discrete Radon transform

7.8.1 Detailed Description

NFFT-based discrete inverse Radon transform.

Computes the inverse of the discrete Radon transform

$$R_{\theta_t} f\left(\frac{s}{R}\right) = \sum_{r \in I_R} w_r \sum_{k \in I_N^2} f_k \mathrm{e}^{-2\pi \mathrm{i} k \left(\frac{r}{R}\theta_t\right)} \mathrm{e}^{2\pi \mathrm{i} r s / R} \qquad (t \in I_T, s \in I_R).$$

given at the points $\frac{r}{R}\theta_t$ of the polar or linogram grid and where w_r are the weights of the Dirichlet- or Fejer-kernel by 1D-FFTs and the 2D-iNFFT.

Author

Markus Fenn

Date

2005

7.9 kernels.c File Reference

File with predefined kernels for the fast summation algorithm.

```
#include "config.h"
#include <stdio.h>
#include <math.h>
#include <float.h>
#include "kernels.h"
```

Functions

- C gaussian (R x, int der, const R *param)

  K(x)=exp(-x^2/c^2)
- C multiquadric (R x, int der, const R *param)

  K(x)=sqrt(x^2+c^2)
- C inverse_multiquadric (R x, int der, const R *param)

  K(x)=1/sqrt(x^2+c^2)
- C logarithm (R x, int der, const R *param)

  K(x)=log |x|.
- C thinplate_spline (R x, int der, const R *param)

  K(x) = x^2 log |x|.
- C one_over_square (R x, int der, const R *param)

  K(x) = 1/x^2.
- C one_over_modulus (R x, int der, const R *param)

  K(x) = 1/|x|.
- C one_over_x (R x, int der, const R *param)

  K(x) = 1/x.
- C inverse_multiquadric3 (R x, int der, const R *param)

  K(x) = 1/sqrt(x^2+c^2)^3.
- C sinc_kernel (R x, int der, const R *param)

  K(x) = sin(cx)/x.
- C cosc (R x, int der, const R *param)

  K(x) = cos(cx)/x.
- C kcot (R x, int der, const R *param)

  K(x) = cot(cx)
- C one_over_cube (R x, int der, const R *param)

  K(x) = 1/x^3.
- C log_sin (R x, int der, const R *param)

  K(x) = log(|sin(cx)|)
- C laplacian_rbf (R x, int der, const R *param)

  K(x) = exp(-|x|/c)

7.9.1 Detailed Description

File with predefined kernels for the fast summation algorithm.

7.10 kernels.h File Reference

Header file with predefined kernels for the fast summation algorithm.

```
#include "config.h"
#include "nfft3.h"
#include "infft.h"
```

Functions

- C gaussian (R x, int der, const R *param)
  K(x)=exp(-x^2/c^2)
- C multiquadric (R x, int der, const R *param)
  K(x)=sqrt(x^2+c^2)
- C inverse_multiquadric (R x, int der, const R *param)
  K(x)=1/sqrt(x^2+c^2)
- C logarithm (R x, int der, const R *param)
  K(x)=log |x|.
- C thinplate_spline (R x, int der, const R *param)
  K(x) = x^2 log |x|.
- C one_over_square (R x, int der, const R *param)
  K(x) = 1/x^2.
- C one_over_modulus (R x, int der, const R *param)
  K(x) = 1/|x|.
- C one_over_x (R x, int der, const R *param)
  K(x) = 1/x.
- C inverse_multiquadric3 (R x, int der, const R *param)
  K(x) = 1/sqrt(x^2+c^2)^3.
- C sinc_kernel (R x, int der, const R *param)
  K(x) = sin(cx)/x.
- C cosc (R x, int der, const R *param)
  K(x) = cos(cx)/x.
- C kcot (R x, int der, const R *param)
  K(x) = cot(cx)
- C one_over_cube (R x, int der, const R *param)
  K(x) = 1/x^3.
- C log_sin (R x, int der, const R *param)
  K(x) = log(|sin(cx)|)
- C laplacian_rbf (R x, int der, const R *param)
  K(x) = exp(-|x|/c)

7.10.1 Detailed Description

Header file with predefined kernels for the fast summation algorithm.

7.11 linogram_fft_test.c File Reference

NFFT-based pseudo-polar FFT and inverse.

```
#include <math.h>
#include <stdlib.h>
#include <complex.h>
#include "nfft3mp.h"
```

Functions

- static int linogram_grid (int T, int rr, NFFT_R ∗x, NFFT_R ∗w)

  Generates the points x with weights w for the linogram grid with T slopes and R offsets.
- static int linogram_dft (NFFT_C ∗f_hat, int NN, NFFT_C ∗f, int T, int rr, int m)

  discrete pseudo-polar FFT
- static int linogram_fft (NFFT_C ∗f_hat, int NN, NFFT_C ∗f, int T, int rr, int m)

  NFFT-based pseudo-polar FFT.
- static int inverse_linogram_fft (NFFT_C ∗f, int T, int rr, NFFT_C ∗f_hat, int NN, int max_i, int m)

  NFFT-based inverse pseudo-polar FFT.
- static int comparison_fft (FILE ∗fp, int N, int T, int rr)

  Comparison of the FFTW, linogram FFT, and inverse linogram FFT.
- int main (int argc, char ∗∗argv)

  test program for various parameters

Variables

- NFFT_R **GLOBAL_elapsed_time**

7.11.1 Detailed Description

NFFT-based pseudo-polar FFT and inverse.

Computes the NFFT-based pseudo-polar FFT and its inverse.

Author

Markus Fenn

Date

2006

7.12 mpolar_fft_test.c File Reference

NFFT-based polar FFT and inverse on modified polar grid.

```
#include <math.h>
#include <stdlib.h>
#include <complex.h>
#include "nfft3mp.h"
```

Functions

- static int mpolar_grid (int T, int S, NFFT_R *x, NFFT_R *w)

  Generates the points $x_{t,j}$ with weights $w_{t,j}$ for the modified polar grid with T angles and R offsets.
- static int mpolar_dft (NFFT_C *f_hat, int NN, NFFT_C *f, int T, int S, int m)

  discrete mpolar FFT
- static int mpolar_fft (NFFT_C *f_hat, int NN, NFFT_C *f, int T, int S, int m)

  NFFT-based mpolar FFT.
- static int inverse_mpolar_fft (NFFT_C *f, int T, int S, NFFT_C *f_hat, int NN, int max_i, int m)

  inverse NFFT-based mpolar FFT
- static int comparison_fft (FILE *fp, int N, int T, int S)

  Comparison of the FFTW, mpolar FFT, and inverse mpolar FFT.
- int main (int argc, char **argv)

  test program for various parameters

Variables

- NFFT_R **GLOBAL_elapsed_time**

7.12.1 Detailed Description

NFFT-based polar FFT and inverse on modified polar grid.

Computes the NFFT-based polar FFT and its inverse on a modified polar grid for various parameters.

Author

Markus Fenn

Date

2006

7.13 ndft_fast.c File Reference

Testing ndft, Horner-like ndft, and fully precomputed ndft.

```
#include "config.h"
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <stdlib.h>
#include "nfft3.h"
#include "infft.h"
```

Functions

- static void **ndft_horner_trafo** (NFFT(plan) *ths)
- static void **ndft_pre_full_trafo** (NFFT(plan) *ths, C *A)
- static void **ndft_pre_full_init** (NFFT(plan) *ths, C *A)
- static void **ndft_time** (int N, int M, unsigned test_ndft, unsigned test_pre_full)
- int **main** (int argc, char **argv)

7.13.1 Detailed Description

Testing ndft, Horner-like ndft, and fully precomputed ndft.

Author

Stefan Kunis

7.14 nfft3.h File Reference

Header file for the nfft3 library.

```
#include <fftw3.h>
```

Data Structures

- struct nfftf_mv_plan_complex
- struct nfftf_mv_plan_double
- struct nfftf_plan

  data structure for an NFFT (nonequispaced fast Fourier transform) plan with float precision
- struct nfftl_mv_plan_double
- struct nfftl_plan

  data structure for an NFFT (nonequispaced fast Fourier transform) plan with long double precision
- struct nfctf_plan

  data structure for an NFCT (nonequispaced fast cosine transform) plan with float precision
- struct nfstf_plan

  data structure for an NFST (nonequispaced fast sine transform) plan with float precision
- struct nnfftf_plan

  data structure for an NNFFT (nonequispaced in time and frequency fast Fourier transform) plan with float precision
- struct nsfftf_plan

  data structure for an NSFFT (nonequispaced sparse fast Fourier transform) plan with float precision
- struct mrif_inh_2d1d_plan
- struct mrif_inh_3d_plan
- struct mril_inh_3d_plan
- struct nfsftf_plan

  data structure for an NFSFT (nonequispaced fast spherical Fourier transform) plan with float precision
- struct nfsoftf_plan_
- struct solverf_plan_complex

  data structure for an inverse NFFT plan with float precision
- struct solverf_plan_double

  data structure for an inverse NFFT plan with float precision
- struct solverl_plan_double

  data structure for an inverse NFFT plan with long double precision

Macros

- #define **NFFT_CONCAT**(prefix, name) prefix ## name
- #define **NFFT_EXTERN** extern
- #define MACRO_MV_PLAN(RC)

  Adjoint transform.
- #define **NFFT_MANGLE_DOUBLE**(name) NFFT_CONCAT(nfft_, name)
- #define **NFFT_MANGLE_FLOAT**(name) NFFT_CONCAT(nfftf_, name)
- #define **NFFT_MANGLE_LONG_DOUBLE**(name) NFFT_CONCAT(nfftl_, name)
- #define **NFFT_DEFINE_MALLOC_API**(X)
- #define **NFFT_DEFINE_API**(X, Y, R, C)
- #define PRE_PHI_HUT (1U<<0)
- #define FG_PSI (1U<<1)
- #define PRE_LIN_PSI (1U<<2)
- #define PRE_FG_PSI (1U<<3)
- #define PRE_PSI (1U<<4)
- #define PRE_FULL_PSI (1U<<5)
- #define MALLOC_X (1U<<6)
- #define MALLOC_F_HAT (1U<<7)
- #define MALLOC_F (1U<<8)
- #define FFT_OUT_OF_PLACE (1U<<9)
- #define FFTW_INIT (1U<<10)
- #define **NFFT_SORT_NODES** (1U<<11)
- #define **NFFT_OMP_BLOCKWISE_ADJOINT** (1U<<12)
- #define PRE_ONE_PSI (PRE_LIN_PSI| PRE_FG_PSI| PRE_PSI| PRE_FULL_PSI)
- #define **NFCT_MANGLE_DOUBLE**(name) NFFT_CONCAT(nfct_, name)
- #define **NFCT_MANGLE_FLOAT**(name) NFFT_CONCAT(nfctf_, name)
- #define **NFCT_MANGLE_LONG_DOUBLE**(name) NFFT_CONCAT(nfctl_, name)
- #define **NFCT_DEFINE_API**(X, Y, R, C)
- #define **NFST_MANGLE_DOUBLE**(name) NFFT_CONCAT(nfst_, name)
- #define **NFST_MANGLE_FLOAT**(name) NFFT_CONCAT(nfstf_, name)
- #define **NFST_MANGLE_LONG_DOUBLE**(name) NFFT_CONCAT(nfstl_, name)
- #define **NFST_DEFINE_API**(X, Y, R, C)
- #define **NNFFT_MANGLE_DOUBLE**(name) NFFT_CONCAT(nnfft_, name)
- #define **NNFFT_MANGLE_FLOAT**(name) NFFT_CONCAT(nnfftf_, name)
- #define **NNFFT_MANGLE_LONG_DOUBLE**(name) NFFT_CONCAT(nnfftl_, name)
- #define **NNFFT_DEFINE_API**(X, Y, Z, R, C)
- #define MALLOC_V (1U<< 11)
- #define **NSFFT_MANGLE_DOUBLE**(name) NFFT_CONCAT(nsfft_, name)
- #define **NSFFT_MANGLE_FLOAT**(name) NFFT_CONCAT(nsfftf_, name)
- #define **NSFFT_MANGLE_LONG_DOUBLE**(name) NFFT_CONCAT(nsfftl_, name)
- #define **NSFFT_DEFINE_API**(X, Y, Z, R, C)
- #define NSDFT (1U<< 12)
- #define **MRI_MANGLE_DOUBLE**(name) NFFT_CONCAT(mri_, name)
- #define **MRI_MANGLE_FLOAT**(name) NFFT_CONCAT(mrif_, name)
- #define **MRI_MANGLE_LONG_DOUBLE**(name) NFFT_CONCAT(mril_, name)
- #define **MRI_DEFINE_API**(X, Z, R, C)
- #define **NFSFT_MANGLE_DOUBLE**(name) NFFT_CONCAT(nfsft_, name)
- #define **NFSFT_MANGLE_FLOAT**(name) NFFT_CONCAT(nfsftf_, name)
- #define **NFSFT_MANGLE_LONG_DOUBLE**(name) NFFT_CONCAT(nfsftl_, name)
- #define **NFSFT_DEFINE_API**(X, Z, R, C)
- #define NFSFT_NORMALIZED (1U << 0)
- #define NFSFT_USE_NDFT (1U << 1)
- #define NFSFT_USE_DPT (1U << 2)
- #define NFSFT_MALLOC_X (1U << 3)

- #define NFSFT_MALLOC_F_HAT (1U << 5)
- #define NFSFT_MALLOC_F (1U << 6)
- #define NFSFT_PRESERVE_F_HAT (1U << 7)
- #define NFSFT_PRESERVE_X (1U << 8)
- #define NFSFT_PRESERVE_F (1U << 9)
- #define NFSFT_DESTROY_F_HAT (1U << 10)
- #define NFSFT_DESTROY_X (1U << 11)
- #define NFSFT_DESTROY_F (1U << 12)
- #define NFSFT_EQUISPACED (1U << 17)
- #define NFSFT_NO_DIRECT_ALGORITHM (1U << 13)
- #define NFSFT_NO_FAST_ALGORITHM (1U << 14)
- #define NFSFT_ZERO_F_HAT (1U << 16)
- #define NFSFT_INDEX(k, n, plan) ((2∗(plan)->N+2)∗((plan)->N-n+1)+(plan)->N+k+1)
- #define NFSFT_F_HAT_SIZE(N) ((2∗N+2)∗(2∗N+2))
- #define **FPT_MANGLE_DOUBLE**(name) NFFT_CONCAT(fpt_, name)
- #define **FPT_MANGLE_FLOAT**(name) NFFT_CONCAT(fptf_, name)
- #define **FPT_MANGLE_LONG_DOUBLE**(name) NFFT_CONCAT(fptl_, name)
- #define **FPT_DEFINE_API**(X, Y, R, C)
- #define FPT_NO_STABILIZATION (1U << 0)

  If set, no stabilization will be used.
- #define FPT_NO_FAST_ALGORITHM (1U << 2)

  If set, TODO complete comment.
- #define FPT_NO_DIRECT_ALGORITHM (1U << 3)

  If set, TODO complete comment.
- #define FPT_PERSISTENT_DATA (1U << 4)

  If set, TODO complete comment.
- #define **FPT_NO_INIT_FPT_DATA** (1U << 7)
- #define FPT_FUNCTION_VALUES (1U << 5)

  If set, the output are function values at Chebyshev nodes rather than Chebyshev coefficients.
- #define FPT_AL_SYMMETRY (1U << 6)

  If set, TODO complete comment.
- #define **NFSOFT_MANGLE_DOUBLE**(name) NFFT_CONCAT(nfsoft_, name)
- #define **NFSOFT_MANGLE_FLOAT**(name) NFFT_CONCAT(nfsoftf_, name)
- #define **NFSOFT_MANGLE_LONG_DOUBLE**(name) NFFT_CONCAT(nfsoftl_, name)
- #define **NFSOFT_DEFINE_API**(X, Y, Z, R, C)
- #define NFSOFT_NORMALIZED (1U << 0)
- #define NFSOFT_USE_NDFT (1U << 1)
- #define NFSOFT_USE_DPT (1U << 2)
- #define NFSOFT_MALLOC_X (1U << 3)
- #define NFSOFT_REPRESENT (1U << 4)
- #define NFSOFT_MALLOC_F_HAT (1U << 5)
- #define NFSOFT_MALLOC_F (1U << 6)
- #define NFSOFT_PRESERVE_F_HAT (1U << 7)
- #define NFSOFT_PRESERVE_X (1U << 8)
- #define NFSOFT_PRESERVE_F (1U << 9)
- #define NFSOFT_DESTROY_F_HAT (1U << 10)
- #define NFSOFT_DESTROY_X (1U << 11)
- #define NFSOFT_DESTROY_F (1U << 12)
- #define NFSOFT_NO_STABILIZATION (1U << 13)
- #define NFSOFT_CHOOSE_DPT (1U << 14)
- #define NFSOFT_SOFT (1U << 15)
- #define NFSOFT_ZERO_F_HAT (1U << 16)
- #define NFSOFT_INDEX(m, n, l, B) (((l)+((B)+1))+(2∗(B)+2)∗(((n)+((B)+1))+(2∗(B)+2)∗((m)+((B)+1))))
- #define NFSOFT_F_HAT_SIZE(B) (((B)+1)∗(4∗((B)+1)∗((B)+1)-1)/3)

- #define **SOLVER_MANGLE_DOUBLE**(name) NFFT_CONCAT(solver_, name)
- #define **SOLVER_MANGLE_FLOAT**(name) NFFT_CONCAT(solverf_, name)
- #define **SOLVER_MANGLE_LONG_DOUBLE**(name) NFFT_CONCAT(solverl_, name)
- #define **SOLVER_DEFINE_API**(X, Y, R, C)
- #define LANDWEBER (1U<< 0)
- #define STEEPEST_DESCENT (1U<< 1)
- #define CGNR (1U<< 2)
- #define CGNE (1U<< 3)
- #define NORMS_FOR_LANDWEBER (1U<< 4)
- #define PRECOMPUTE_WEIGHT (1U<< 5)
- #define PRECOMPUTE_DAMP (1U<< 6)
- #define **NFFT_DEFINE_UTIL_API**(Y, R, C)

Typedefs

- typedef ptrdiff_t **NFFT_INT**
- typedef void *(* **nfftf_malloc_type_function**) (size_t n)
- typedef void(* **nfftf_free_type_function**) (void *p)
- typedef void(* **nfftf_die_type_function**) (const char *errString)
- typedef void *(* **nfftl_malloc_type_function**) (size_t n)
- typedef void(* **nfftl_free_type_function**) (void *p)
- typedef void(* **nfftl_die_type_function**) (const char *errString)
- typedef struct fptf_set_s_ * fptf_set

  A set of precomputed data for a set of DPT transforms of equal maximum length.
- typedef struct nfsoftf_plan_ **nfsoftf_plan**

Functions

- void * **nfftf_malloc** (size_t n)
- void **nfftf_free** (void *p)
- void **nfftf_die** (const char *s)
- **NNNNNNNNNNNNNFFT_DEFINE_MALLOC_API** (NFFT_MANGLE_DOUBLE) extern void *nfftl_↩ malloc(size_t n)
- void **nfftl_free** (void *p)
- void **nfftl_die** (const char *s)
- void **nfftf_trafo_direct** (const nfftf_plan *ths)
- void **nfftf_adjoint_direct** (const nfftf_plan *ths)
- void **nfftf_trafo** (nfftf_plan *ths)
- void **nfftf_trafo_1d** (nfftf_plan *ths)
- void **nfftf_trafo_2d** (nfftf_plan *ths)
- void **nfftf_trafo_3d** (nfftf_plan *ths)
- void **nfftf_adjoint** (nfftf_plan *ths)
- void **nfftf_adjoint_1d** (nfftf_plan *ths)
- void **nfftf_adjoint_2d** (nfftf_plan *ths)
- void **nfftf_adjoint_3d** (nfftf_plan *ths)
- void **nfftf_init_1d** (nfftf_plan *ths, int N1, int M)
- void **nfftf_init_2d** (nfftf_plan *ths, int N1, int N2, int M)
- void **nfftf_init_3d** (nfftf_plan *ths, int N1, int N2, int N3, int M)
- void **nfftf_init** (nfftf_plan *ths, int d, int *N, int M)
- void **nfftf_init_guru** (nfftf_plan *ths, int d, int *N, int M, int *n, int m, unsigned flags, unsigned fftw_flags)
- void **nfftf_init_lin** (nfftf_plan *ths, int d, int *N, int M, int *n, int m, int K, unsigned flags, unsigned fftw_flags)
- void **nfftf_precompute_one_psi** (nfftf_plan *ths)

- void **nfftf_precompute_psi** (nfftf_plan *ths)
- void **nfftf_precompute_full_psi** (nfftf_plan *ths)
- void **nfftf_precompute_fg_psi** (nfftf_plan *ths)
- void **nfftf_precompute_lin_psi** (nfftf_plan *ths)
- const char * **nfftf_check** (nfftf_plan *ths)
- void **nfftf_finalize** (nfftf_plan *ths)
- **NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNFFT_DEFINE_API** (NF↩FT_MANGLE_DOUBLE, FFTW_MANGLE_DOUBLE, double, fftw_complex) typedef struct
- void **nfftl_trafo_direct** (const nfftl_plan *ths)
- void **nfftl_adjoint_direct** (const nfftl_plan *ths)
- void **nfftl_trafo** (nfftl_plan *ths)
- void **nfftl_trafo_1d** (nfftl_plan *ths)
- void **nfftl_trafo_2d** (nfftl_plan *ths)
- void **nfftl_trafo_3d** (nfftl_plan *ths)
- void **nfftl_adjoint** (nfftl_plan *ths)
- void **nfftl_adjoint_1d** (nfftl_plan *ths)
- void **nfftl_adjoint_2d** (nfftl_plan *ths)
- void **nfftl_adjoint_3d** (nfftl_plan *ths)
- void **nfftl_init_1d** (nfftl_plan *ths, int N1, int M)
- void **nfftl_init_2d** (nfftl_plan *ths, int N1, int N2, int M)
- void **nfftl_init_3d** (nfftl_plan *ths, int N1, int N2, int N3, int M)
- void **nfftl_init** (nfftl_plan *ths, int d, int *N, int M)
- void **nfftl_init_guru** (nfftl_plan *ths, int d, int *N, int M, int *n, int m, unsigned flags, unsigned fftw_flags)
- void **nfftl_init_lin** (nfftl_plan *ths, int d, int *N, int M, int *n, int m, int K, unsigned flags, unsigned fftw_flags)
- void **nfftl_precompute_one_psi** (nfftl_plan *ths)
- void **nfftl_precompute_psi** (nfftl_plan *ths)
- void **nfftl_precompute_full_psi** (nfftl_plan *ths)
- void **nfftl_precompute_fg_psi** (nfftl_plan *ths)
- void **nfftl_precompute_lin_psi** (nfftl_plan *ths)
- const char * **nfftl_check** (nfftl_plan *ths)
- void **nfftl_finalize** (nfftl_plan *ths)
- void **nfctf_init_1d** (nfctf_plan *ths_plan, int N0, int M_total)
- void **nfctf_init_2d** (nfctf_plan *ths_plan, int N0, int N1, int M_total)
- void **nfctf_init_3d** (nfctf_plan *ths_plan, int N0, int N1, int N2, int M_total)
- void **nfctf_init** (nfctf_plan *ths_plan, int d, int *N, int M_total)
- void **nfctf_init_guru** (nfctf_plan *ths_plan, int d, int *N, int M_total, int *n, int m, unsigned flags, unsigned fftw_flags)
- void **nfctf_precompute_one_psi** (nfctf_plan *ths)
- void **nfctf_precompute_psi** (nfctf_plan *ths)
- void **nfctf_precompute_full_psi** (nfctf_plan *ths)
- void **nfctf_precompute_fg_psi** (nfctf_plan *ths)
- void **nfctf_precompute_lin_psi** (nfctf_plan *ths)
- void **nfctf_trafo** (nfctf_plan *ths_plan)
- void **nfctf_trafo_direct** (const nfctf_plan *ths_plan)
- void **nfctf_adjoint** (nfctf_plan *ths_plan)
- void **nfctf_adjoint_direct** (const nfctf_plan *ths_plan)
- const char * **nfctf_check** (nfctf_plan *ths)
- void **nfctf_finalize** (nfctf_plan *ths_plan)
- NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNFCT_DEFINE_API (NFCT_MANGLE_DOUBLE, FFTW_MANGLE_DOUBLE, double, fftw_complex) typedef struct

  data structure for an NFCT (nonequispaced fast cosine transform) plan with long double precision
- void **nfctl_init_1d** (nfctl_plan *ths_plan, int N0, int M_total)
- void **nfctl_init_2d** (nfctl_plan *ths_plan, int N0, int N1, int M_total)
- void **nfctl_init_3d** (nfctl_plan *ths_plan, int N0, int N1, int N2, int M_total)
- void **nfctl_init** (nfctl_plan *ths_plan, int d, int *N, int M_total)

- void **nfctl_init_guru** (nfctl_plan *ths_plan, int d, int *N, int M_total, int *n, int m, unsigned flags, unsigned fftw_flags)
- void **nfctl_precompute_one_psi** (nfctl_plan *ths)
- void **nfctl_precompute_psi** (nfctl_plan *ths)
- void **nfctl_precompute_full_psi** (nfctl_plan *ths)
- void **nfctl_precompute_fg_psi** (nfctl_plan *ths)
- void **nfctl_precompute_lin_psi** (nfctl_plan *ths)
- void **nfctl_trafo** (nfctl_plan *ths_plan)
- void **nfctl_trafo_direct** (const nfctl_plan *ths_plan)
- void **nfctl_adjoint** (nfctl_plan *ths_plan)
- void **nfctl_adjoint_direct** (const nfctl_plan *ths_plan)
- const char * **nfctl_check** (nfctl_plan *ths)
- void **nfctl_finalize** (nfctl_plan *ths_plan)
- void **nfstf_init_1d** (nfstf_plan *ths_plan, int N0, int M_total)
- void **nfstf_init_2d** (nfstf_plan *ths_plan, int N0, int N1, int M_total)
- void **nfstf_init_3d** (nfstf_plan *ths_plan, int N0, int N1, int N2, int M_total)
- void **nfstf_init** (nfstf_plan *ths_plan, int d, int *N, int M_total)
- void **nfstf_init_guru** (nfstf_plan *ths_plan, int d, int *N, int M_total, int *n, int m, unsigned flags, unsigned fftw_flags)
- void **nfstf_precompute_one_psi** (nfstf_plan *ths)
- void **nfstf_precompute_psi** (nfstf_plan *ths)
- void **nfstf_precompute_full_psi** (nfstf_plan *ths)
- void **nfstf_precompute_fg_psi** (nfstf_plan *ths)
- void **nfstf_precompute_lin_psi** (nfstf_plan *ths)
- void **nfstf_trafo** (nfstf_plan *ths_plan)
- void **nfstf_trafo_direct** (const nfstf_plan *ths_plan)
- void **nfstf_adjoint** (nfstf_plan *ths_plan)
- void **nfstf_adjoint_direct** (const nfstf_plan *ths_plan)
- const char * **nfstf_check** (nfstf_plan *ths)
- void **nfstf_finalize** (nfstf_plan *ths_plan)
- NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNFST_DEFINE_API (NFST_MANGLE_DOUBLE, FFTW_MANGLE_DOUBLE, double, fftw_complex) typedef struct

  data structure for an NFST (nonequispaced fast sine transform) plan with long double precision
- void **nfstl_init_1d** (nfstl_plan *ths_plan, int N0, int M_total)
- void **nfstl_init_2d** (nfstl_plan *ths_plan, int N0, int N1, int M_total)
- void **nfstl_init_3d** (nfstl_plan *ths_plan, int N0, int N1, int N2, int M_total)
- void **nfstl_init** (nfstl_plan *ths_plan, int d, int *N, int M_total)
- void **nfstl_init_guru** (nfstl_plan *ths_plan, int d, int *N, int M_total, int *n, int m, unsigned flags, unsigned fftw_flags)
- void **nfstl_precompute_one_psi** (nfstl_plan *ths)
- void **nfstl_precompute_psi** (nfstl_plan *ths)
- void **nfstl_precompute_full_psi** (nfstl_plan *ths)
- void **nfstl_precompute_fg_psi** (nfstl_plan *ths)
- void **nfstl_precompute_lin_psi** (nfstl_plan *ths)
- void **nfstl_trafo** (nfstl_plan *ths_plan)
- void **nfstl_trafo_direct** (const nfstl_plan *ths_plan)
- void **nfstl_adjoint** (nfstl_plan *ths_plan)
- void **nfstl_adjoint_direct** (const nfstl_plan *ths_plan)
- const char * **nfstl_check** (nfstl_plan *ths)
- void **nfstl_finalize** (nfstl_plan *ths_plan)
- void **nnfftf_init** (nnfftf_plan *ths_plan, int d, int N_total, int M_total, int *N)
- void **nnfftf_init_1d** (nnfftf_plan *ths_plan, int N, int M_total)
- void **nnfftf_init_guru** (nnfftf_plan *ths_plan, int d, int N_total, int M_total, int *N, int *N1, int m, unsigned nnfft_flags)
- void **nnfftf_trafo_direct** (nnfftf_plan *ths_plan)

- void **nnfftf_adjoint_direct** (nnfftf_plan *ths_plan)
- void **nnfftf_trafo** (nnfftf_plan *ths_plan)
- void **nnfftf_adjoint** (nnfftf_plan *ths_plan)
- void **nnfftf_precompute_lin_psi** (nnfftf_plan *ths_plan)
- void **nnfftf_precompute_psi** (nnfftf_plan *ths_plan)
- void **nnfftf_precompute_full_psi** (nnfftf_plan *ths_plan)
- void **nnfftf_precompute_phi_hut** (nnfftf_plan *ths_plan)
- void **nnfftf_precompute_one_psi** (nnfftf_plan *ths)
- void **nnfftf_finalize** (nnfftf_plan *ths_plan)
- NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNFFT_DEFINE_API (NNFFT_MANGLE_DOUBLE, FFTW↩ _MANGLE_DOUBLE, NFFT_MANGLE_DOUBLE, double, fftw_complex) typedef struct

  data structure for an NNFFT (nonequispaced in time and frequency fast Fourier transform) plan with long double precision
- void **nnfftl_init** (nnfftl_plan *ths_plan, int d, int N_total, int M_total, int *N)
- void **nnfftl_init_1d** (nnfftl_plan *ths_plan, int N, int M_total)
- void **nnfftl_init_guru** (nnfftl_plan *ths_plan, int d, int N_total, int M_total, int *N, int *N1, int m, unsigned nnfft_flags)
- void **nnfftl_trafo_direct** (nnfftl_plan *ths_plan)
- void **nnfftl_adjoint_direct** (nnfftl_plan *ths_plan)
- void **nnfftl_trafo** (nnfftl_plan *ths_plan)
- void **nnfftl_adjoint** (nnfftl_plan *ths_plan)
- void **nnfftl_precompute_lin_psi** (nnfftl_plan *ths_plan)
- void **nnfftl_precompute_psi** (nnfftl_plan *ths_plan)
- void **nnfftl_precompute_full_psi** (nnfftl_plan *ths_plan)
- void **nnfftl_precompute_phi_hut** (nnfftl_plan *ths_plan)
- void **nnfftl_precompute_one_psi** (nnfftl_plan *ths)
- void **nnfftl_finalize** (nnfftl_plan *ths_plan)
- void **nsfftf_trafo_direct** (nsfftf_plan *ths)
- void **nsfftf_adjoint_direct** (nsfftf_plan *ths)
- void **nsfftf_trafo** (nsfftf_plan *ths)
- void **nsfftf_adjoint** (nsfftf_plan *ths)
- void **nsfftf_cp** (nsfftf_plan *ths, nfftf_plan *ths_nfft)
- void **nsfftf_init_random_nodes_coeffs** (nsfftf_plan *ths)
- void **nsfftf_init** (nsfftf_plan *ths, int d, int J, int M, int m, unsigned flags)
- void **nsfftf_finalize** (nsfftf_plan *ths)
- NNNNNNNNNNNNNNNNNNNNNNNNNNNSFFT_DEFINE_API (NSFFT_MANGLE_DOUBLE, FFTW_MANGL↩ E_DOUBLE, NFFT_MANGLE_DOUBLE, double, fftw_complex) typedef struct

  data structure for an NSFFT (nonequispaced sparse fast Fourier transform) plan with long double precision
- void **nsfftl_trafo_direct** (nsfftl_plan *ths)
- void **nsfftl_adjoint_direct** (nsfftl_plan *ths)
- void **nsfftl_trafo** (nsfftl_plan *ths)
- void **nsfftl_adjoint** (nsfftl_plan *ths)
- void **nsfftl_cp** (nsfftl_plan *ths, nfftl_plan *ths_nfft)
- void **nsfftl_init_random_nodes_coeffs** (nsfftl_plan *ths)
- void **nsfftl_init** (nsfftl_plan *ths, int d, int J, int M, int m, unsigned flags)
- void **nsfftl_finalize** (nsfftl_plan *ths)
- void **mrif_inh_2d1d_trafo** (mrif_inh_2d1d_plan *ths)
- void **mrif_inh_2d1d_adjoint** (mrif_inh_2d1d_plan *ths)
- void **mrif_inh_2d1d_init_guru** (mrif_inh_2d1d_plan *ths, int *N, int M, int *n, int m, float sigma, unsigned nfft_flags, unsigned fftw_flags)
- void **mrif_inh_2d1d_finalize** (mrif_inh_2d1d_plan *ths)
- void **mrif_inh_3d_trafo** (mrif_inh_3d_plan *ths)
- void **mrif_inh_3d_adjoint** (mrif_inh_3d_plan *ths)
- void **mrif_inh_3d_init_guru** (mrif_inh_3d_plan *ths, int *N, int M, int *n, int m, float sigma, unsigned nfft↩ _flags, unsigned fftw_flags)

- void **mrif_inh_3d_finalize** (mrif_inh_3d_plan *ths)
- **MMMMMMMMMMMMMMMMMMMMMRI_DEFINE_API** (MRI_MANGLE_DOUBLE, NFFT_MANGLE_D↩ OUBLE, double, fftw_complex) typedef struct
- void **mril_inh_2d1d_trafo** (mril_inh_2d1d_plan *ths)
- void **mril_inh_2d1d_adjoint** (mril_inh_2d1d_plan *ths)
- void **mril_inh_2d1d_init_guru** (mril_inh_2d1d_plan *ths, int *N, int M, int *n, int m, long double sigma, unsigned nfft_flags, unsigned fftw_flags)
- void **mril_inh_2d1d_finalize** (mril_inh_2d1d_plan *ths)
- void **mril_inh_3d_trafo** (mril_inh_3d_plan *ths)
- void **mril_inh_3d_adjoint** (mril_inh_3d_plan *ths)
- void **mril_inh_3d_init_guru** (mril_inh_3d_plan *ths, int *N, int M, int *n, int m, long double sigma, unsigned nfft_flags, unsigned fftw_flags)
- void **mril_inh_3d_finalize** (mril_inh_3d_plan *ths)
- void **nfsftf_init** (nfsftf_plan *plan, int N, int M)
- void **nfsftf_init_advanced** (nfsftf_plan *plan, int N, int M, unsigned int nfsft_flags)
- void **nfsftf_init_guru** (nfsftf_plan *plan, int N, int M, unsigned int nfsft_flags, unsigned int nfft_flags, int nfft_cutoff)
- void **nfsftf_precompute** (int N, float kappa, unsigned int nfsft_flags, unsigned int fpt_flags)
- void **nfsftf_forget** (void)
- void **nfsftf_trafo_direct** (nfsftf_plan *plan)
- void **nfsftf_adjoint_direct** (nfsftf_plan *plan)
- void **nfsftf_trafo** (nfsftf_plan *plan)
- void **nfsftf_adjoint** (nfsftf_plan *plan)
- void **nfsftf_finalize** (nfsftf_plan *plan)
- void **nfsftf_precompute_x** (nfsftf_plan *plan)
- NNNNNNNNNNNNNNNNNNNNNNNFSFT_DEFINE_API (NFSFT_MANGLE_DOUBLE, NFFT_MANGLE↩ _DOUBLE, double, fftw_complex) typedef struct

  data structure for an NFSFT (nonequispaced fast spherical Fourier transform) plan with long double precision
- void **nfsftl_init** (nfsftl_plan *plan, int N, int M)
- void **nfsftl_init_advanced** (nfsftl_plan *plan, int N, int M, unsigned int nfsft_flags)
- void **nfsftl_init_guru** (nfsftl_plan *plan, int N, int M, unsigned int nfsft_flags, unsigned int nfft_flags, int nfft_cutoff)
- void **nfsftl_precompute** (int N, long double kappa, unsigned int nfsft_flags, unsigned int fpt_flags)
- void **nfsftl_forget** (void)
- void **nfsftl_trafo_direct** (nfsftl_plan *plan)
- void **nfsftl_adjoint_direct** (nfsftl_plan *plan)
- void **nfsftl_trafo** (nfsftl_plan *plan)
- void **nfsftl_adjoint** (nfsftl_plan *plan)
- void **nfsftl_finalize** (nfsftl_plan *plan)
- void **nfsftl_precompute_x** (nfsftl_plan *plan)
- fptf_set **fptf_init** (const int M, const int t, const unsigned int flags)
- void **fptf_precompute** (fptf_set set, const int m, float *alpha, float *beta, float *gam, int k_start, const float threshold)
- void **fptf_trafo_direct** (fptf_set set, const int m, const fftwf_complex *x, fftwf_complex *y, const int k_end, const unsigned int flags)
- void **fptf_trafo** (fptf_set set, const int m, const fftwf_complex *x, fftwf_complex *y, const int k_end, const unsigned int flags)
- void **fptf_transposed_direct** (fptf_set set, const int m, fftwf_complex *x, fftwf_complex *y, const int k_end, const unsigned int flags)
- void **fptf_transposed** (fptf_set set, const int m, fftwf_complex *x, fftwf_complex *y, const int k_end, const unsigned int flags)
- void **fptf_finalize** (fptf_set set)
- FFFFFFFFFFFFFFFFFPT_DEFINE_API (FPT_MANGLE_DOUBLE, FFTW_MANGLE_DOUBLE, double, fftw_complex) typedef struct fptl_set_s_ *fptl_set

  A set of precomputed data for a set of DPT transforms of equal maximum length.

- fptl_set **fptl_init** (const int M, const int t, const unsigned int flags)
- void **fptl_precompute** (fptl_set set, const int m, long double *alpha, long double *beta, long double *gam, int k_start, const long double threshold)
- void **fptl_trafo_direct** (fptl_set set, const int m, const fftwl_complex *x, fftwl_complex *y, const int k_end, const unsigned int flags)
- void **fptl_trafo** (fptl_set set, const int m, const fftwl_complex *x, fftwl_complex *y, const int k_end, const unsigned int flags)
- void **fptl_transposed_direct** (fptl_set set, const int m, fftwl_complex *x, fftwl_complex *y, const int k_end, const unsigned int flags)
- void **fptl_transposed** (fptl_set set, const int m, fftwl_complex *x, fftwl_complex *y, const int k_end, const unsigned int flags)
- void **fptl_finalize** (fptl_set set)
- void **nfsoftf_precompute** (nfsoftf_plan *plan)
- fptf_set **nfsoftf_SO3_single_fpt_init** (int l, int k, int m, unsigned int flags, int kappa)
- void **nfsoftf_SO3_fpt** (fftwf_complex *coeffs, fptf_set set, int l, int k, int m, unsigned int nfsoft_flags)
- void **nfsoftf_SO3_fpt_transposed** (fftwf_complex *coeffs, fptf_set set, int l, int k, int m, unsigned int nfsoft↩ _flags)
- void **nfsoftf_init** (nfsoftf_plan *plan, int N, int M)
- void **nfsoftf_init_advanced** (nfsoftf_plan *plan, int N, int M, unsigned int nfsoft_flags)
- void **nfsoftf_init_guru** (nfsoftf_plan *plan, int N, int M, unsigned int nfsoft_flags, unsigned int nfft_flags, int nfft_cutoff, int fpt_kappa)
- void **nfsoftf_init_guru_advanced** (nfsoftf_plan *plan, int N, int M, unsigned int nfsoft_flags, unsigned int nfft_flags, int nfft_cutoff, int fpt_kappa, int nn_oversampled)
- void **nfsoftf_trafo** (nfsoftf_plan *plan_nfsoft)
- void **nfsoftf_adjoint** (nfsoftf_plan *plan_nfsoft)
- void **nfsoftf_finalize** (nfsoftf_plan *plan)
- int **nfsoftf_posN** (int n, int m, int B)
- **NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNFSOFT_DEFINE_API** (NFSOFT_MANGLE_DOUBLE, NFF↩ T_MANGLE_DOUBLE, FPT_MANGLE_DOUBLE, double, fftw_complex) typedef struct nfsoftl_plan_
- void **nfsoftl_precompute** (nfsoftl_plan *plan)
- fptl_set **nfsoftl_SO3_single_fpt_init** (int l, int k, int m, unsigned int flags, int kappa)
- void **nfsoftl_SO3_fpt** (fftwl_complex *coeffs, fptl_set set, int l, int k, int m, unsigned int nfsoft_flags)
- void **nfsoftl_SO3_fpt_transposed** (fftwl_complex *coeffs, fptl_set set, int l, int k, int m, unsigned int nfsoft↩ _flags)
- void **nfsoftl_init** (nfsoftl_plan *plan, int N, int M)
- void **nfsoftl_init_advanced** (nfsoftl_plan *plan, int N, int M, unsigned int nfsoft_flags)
- void **nfsoftl_init_guru** (nfsoftl_plan *plan, int N, int M, unsigned int nfsoft_flags, unsigned int nfft_flags, int nfft_cutoff, int fpt_kappa)
- void **nfsoftl_init_guru_advanced** (nfsoftl_plan *plan, int N, int M, unsigned int nfsoft_flags, unsigned int nfft_flags, int nfft_cutoff, int fpt_kappa, int nn_oversampled)
- void **nfsoftl_trafo** (nfsoftl_plan *plan_nfsoft)
- void **nfsoftl_adjoint** (nfsoftl_plan *plan_nfsoft)
- void **nfsoftl_finalize** (nfsoftl_plan *plan)
- int **nfsoftl_posN** (int n, int m, int B)
- void **solverf_init_advanced_complex** (solverf_plan_complex *ths, nfftf_mv_plan_complex *mv, unsigned flags)
- void **solverf_init_complex** (solverf_plan_complex *ths, nfftf_mv_plan_complex *mv)
- void **solverf_before_loop_complex** (solverf_plan_complex *ths)
- void **solverf_loop_one_step_complex** (solverf_plan_complex *ths)
- void **solverf_finalize_complex** (solverf_plan_complex *ths)
- void **solverf_init_advanced_double** (solverf_plan_double *ths, nfftf_mv_plan_double *mv, unsigned flags)
- void **solverf_init_double** (solverf_plan_double *ths, nfftf_mv_plan_double *mv)
- void **solverf_before_loop_double** (solverf_plan_double *ths)
- void **solverf_loop_one_step_double** (solverf_plan_double *ths)
- void **solverf_finalize_double** (solverf_plan_double *ths)

- SSSSSSSSSSSSSSSSSSSSSSSSSSSSSOLVER_DEFINE_API (SOLVER_MANGLE_DOUBLE, NFFT↩ _MANGLE_DOUBLE, double, fftw_complex) typedef struct

  data structure for an inverse NFFT plan with long double precision
- void **solverl_init_advanced_complex** (solverl_plan_complex *ths, nfftl_mv_plan_complex *mv, unsigned flags)
- void **solverl_init_complex** (solverl_plan_complex *ths, nfftl_mv_plan_complex *mv)
- void **solverl_before_loop_complex** (solverl_plan_complex *ths)
- void **solverl_loop_one_step_complex** (solverl_plan_complex *ths)
- void **solverl_finalize_complex** (solverl_plan_complex *ths)
- void **solverl_init_advanced_double** (solverl_plan_double *ths, nfftl_mv_plan_double *mv, unsigned flags)
- void **solverl_init_double** (solverl_plan_double *ths, nfftl_mv_plan_double *mv)
- void **solverl_before_loop_double** (solverl_plan_double *ths)
- void **solverl_loop_one_step_double** (solverl_plan_double *ths)
- void **solverl_finalize_double** (solverl_plan_double *ths)
- float **nfftf_drand48** (void)
- void **nfftf_srand48** (long int seed)
- void nfftf_vrand_unit_complex (fftwf_complex *x, const NFFT_INT n)

  Inits a vector of random complex numbers in $[0,1] \times [0,1]\mathrm{i}$.
- void nfftf_vrand_shifted_unit_double (float *x, const NFFT_INT n)

  Inits a vector of random double numbers in $[-1/2, 1/2]$.
- void **nfftf_vrand_real** (float *x, const NFFT_INT n, const float a, const float b)
- void nfftf_vpr_double (float *x, const NFFT_INT n, const char *text)

  Print real vector to standard output.
- void nfftf_vpr_complex (fftwf_complex *x, const NFFT_INT n, const char *text)

  Print complex vector to standard output.
- NFFT_INT **nfftf_get_num_threads** (void)
- float **nfftf_clock_gettime_seconds** (void)
- float **nfftf_error_l_infty_complex** (const fftwf_complex *x, const fftwf_complex *y, const NFFT_INT n)
- float **nfftf_error_l_infty_1_complex** (const fftwf_complex *x, const fftwf_complex *y, const NFFT_INT n, const fftwf_complex *z, const NFFT_INT m)
- NFFT_INT **nfftf_exp2i** (const NFFT_INT a)
- NFFT_INT **nfftf_next_power_of_2** (const NFFT_INT N)
- float nfftf_dot_complex (fftwf_complex *x, NFFT_INT n)

  Computes the inner/dot product $x^H x$.
- void nfftf_upd_axpy_complex (fftwf_complex *x, float a, fftwf_complex *y, NFFT_INT n)

  Updates $x \leftarrow ax + y$.
- void nfftf_fftshift_complex (fftwf_complex *x, NFFT_INT d, NFFT_INT *N)

  Swaps each half over N[d]/2.
- void **nfftf_fftshift_complex_int** (fftwf_complex *x, int d, int *N)
- void nfftf_get_version (unsigned *major, unsigned *minor, unsigned *patch)

  Return library version.
- const char * nfftf_get_window_name ()
- NFFT_INT **nfftf_get_default_window_cut_off** ()
- **NNNNNNNNNNNNNNNNNNNNNFFT_DEFINE_UTIL_API** (NFFT_MANGLE_DOUBLE, double, fftw_↩ complex) long double nfftl_drand48(void)
- void **nfftl_srand48** (long int seed)
- void nfftl_vrand_unit_complex (fftwl_complex *x, const NFFT_INT n)

  Inits a vector of random complex numbers in $[0,1] \times [0,1]\mathrm{i}$.
- void nfftl_vrand_shifted_unit_double (long double *x, const NFFT_INT n)

  Inits a vector of random double numbers in $[-1/2, 1/2]$.
- void **nfftl_vrand_real** (long double *x, const NFFT_INT n, const long double a, const long double b)
- void nfftl_vpr_double (long double *x, const NFFT_INT n, const char *text)

  Print real vector to standard output.

- void nfftl_vpr_complex (fftwl_complex *x, const NFFT_INT n, const char *text)

  Print complex vector to standard output.
- NFFT_INT **nfftl_get_num_threads** (void)
- long double **nfftl_clock_gettime_seconds** (void)
- long double **nfftl_error_l_infty_complex** (const fftwl_complex *x, const fftwl_complex *y, const NFFT_INT n)
- long double **nfftl_error_l_infty_1_complex** (const fftwl_complex *x, const fftwl_complex *y, const NFFT_INT n, const fftwl_complex *z, const NFFT_INT m)
- NFFT_INT **nfftl_exp2i** (const NFFT_INT a)
- NFFT_INT **nfftl_next_power_of_2** (const NFFT_INT N)
- long double nfftl_dot_complex (fftwl_complex *x, NFFT_INT n)

  Computes the inner/dot product $x^H x$.
- void nfftl_upd_axpy_complex (fftwl_complex *x, long double a, fftwl_complex *y, NFFT_INT n)

  Updates $x \leftarrow ax + y$.
- void nfftl_fftshift_complex (fftwl_complex *x, NFFT_INT d, NFFT_INT *N)

  Swaps each half over N[d]/2.
- void **nfftl_fftshift_complex_int** (fftwl_complex *x, int d, int *N)
- void nfftl_get_version (unsigned *major, unsigned *minor, unsigned *patch)

  Return library version.
- const char * nfftl_get_window_name ()
- NFFT_INT **nfftl_get_default_window_cut_off** ()

Variables

- nfftf_malloc_type_function **nfftf_malloc_hook**
- nfftf_free_type_function **nfftf_free_hook**
- nfftf_die_type_function **nfftf_die_hook**
- nfftl_malloc_type_function **nfftl_malloc_hook**
- nfftl_free_type_function **nfftl_free_hook**
- nfftl_die_type_function **nfftl_die_hook**
- *We expand this macro for each supported precision * X
- *We expand this macro for each supported precision **Y**
- *We expand this macro for each supported precision **R**
- **nfftl_mv_plan_complex**
- **nfctl_plan**
- **nfstl_plan**
- **nnfftl_plan**
- **nsfftl_plan**
- **mril_inh_2d1d_plan**
- **nfsftl_plan**
- **nfsoftl_plan**
- **solverl_plan_complex**

7.14.1 Detailed Description

Header file for the nfft3 library.

7.14.2 Macro Definition Documentation

7.14.2.1 MACRO_MV_PLAN

```
#define MACRO_MV_PLAN(
                RC )
```

Value:

```
  NFFT_INT N_total; \
  NFFT_INT M_total; \
  RC *f_hat; \
  RC *f; \
  void (*mv_trafo)(void*); \
  void (*mv_adjoint)(void*);
```

Adjoint transform.

Macros for public members inherited by all plan structures.

Definition at line 66 of file nfft3.h.

7.14.2.2 NFFT_DEFINE_MALLOC_API

```
#define NFFT_DEFINE_MALLOC_API(
                X )
```

Value:

```
/* our own memory allocation and exit functions */ \
NFFT_EXTERN void *X(malloc)(size_t n); \
NFFT_EXTERN void X(free)(void *p); \
NFFT_EXTERN void X(die)(const char *s); \
\
/* You can replace the hooks with your own functions, if necessary. We */ \
/* need this for the Matlab interface. */ \
typedef void *(*X(malloc_type_function)) (size_t n); \
typedef void  (*X(free_type_function)) (void *p); \
typedef void  (*X(die_type_function)) (const char *errString); \
NFFT_EXTERN X(malloc_type_function) X(malloc_hook); \
NFFT_EXTERN X(free_type_function) X(free_hook); \
NFFT_EXTERN X(die_type_function) X(die_hook);
```

Definition at line 75 of file nfft3.h.

7.14.2.3 MRI_DEFINE_API

```
#define MRI_DEFINE_API(
                X,
                Z,
                R,
                C )
```

Value:

```
typedef struct\
{\
  MACRO_MV_PLAN(C)\
  Z(plan) plan;\
  int N3;\
  R sigma3;\
  R *t;\
  R *w;\
} X(inh_2d1d_plan);\
\
typedef struct\
```

```
{\
  MACRO_MV_PLAN(C)\
  Z(plan) plan;\
  int N3;\
  R sigma3;\
  R *t;\
  R *w;\
} X(inh_3d_plan);\
\
void X(inh_2d1d_trafo)(X(inh_2d1d_plan) *ths); \
void X(inh_2d1d_adjoint)(X(inh_2d1d_plan) *ths); \
void X(inh_2d1d_init_guru)(X(inh_2d1d_plan) *ths, int *N, int M, int *n, \
  int m, R sigma, unsigned nfft_flags, unsigned fftw_flags); \
void X(inh_2d1d_finalize)(X(inh_2d1d_plan) *ths); \
void X(inh_3d_trafo)(X(inh_3d_plan) *ths); \
void X(inh_3d_adjoint)(X(inh_3d_plan) *ths); \
void X(inh_3d_init_guru)(X(inh_3d_plan) *ths, int *N, int M, int *n, \
  int m, R sigma, unsigned nfft_flags, unsigned fftw_flags); \
void X(inh_3d_finalize)(X(inh_3d_plan) *ths);
```

Definition at line 480 of file nfft3.h.

7.14.2.4 FPT_DEFINE_API

```
#define FPT_DEFINE_API(
                X,
                Y,
                R,
                C )
```

Value:

```
typedef struct X(set_s_) *X(set); \
\
NFFT_EXTERN X(set) X(init)(const int M, const int t, const unsigned int flags); \
NFFT_EXTERN void X(precompute)(X(set) set, const int m, R *alpha, R *beta, \
  R *gam, int k_start, const R threshold); \
NFFT_EXTERN void X(trafo_direct)(X(set) set, const int m, const C *x, C *y, \
  const int k_end, const unsigned int flags); \
NFFT_EXTERN void X(trafo)(X(set) set, const int m, const C *x, C *y, \
  const int k_end, const unsigned int flags); \
NFFT_EXTERN void X(transposed_direct)(X(set) set, const int m, C *x, \
  C *y, const int k_end, const unsigned int flags); \
NFFT_EXTERN void X(transposed)(X(set) set, const int m, C *x, \
  C *y, const int k_end, const unsigned int flags); \
NFFT_EXTERN void X(finalize)(X(set) set);
```

Definition at line 597 of file nfft3.h.

7.14.2.5 NFSOFT_DEFINE_API

```
#define NFSOFT_DEFINE_API(
                X,
                Y,
                Z,
                R,
                C )
```

Value:

```
typedef struct X(plan_)\
{\
  MACRO_MV_PLAN(C) \
  R *x; \
  /* internal use only */\
  C *wig_coeffs; \
```

```
  C *cheby; \
  C *aux; \
  int t; \
  unsigned int flags; \
  Y(plan) p_nfft; \
  Z(set) *internal_fpt_set; \
  int nthreads; \
} X(plan);\
\
NFFT_EXTERN void X(precompute)(X(plan) *plan); \
NFFT_EXTERN Z(set) X(SO3_single_fpt_init)(int l, int k, int m, unsigned int flags, int kappa); \
NFFT_EXTERN void X(SO3_fpt)(C *coeffs, Z(set) set, int l, int k, int m, unsigned int nfsoft_flags); \
NFFT_EXTERN void X(SO3_fpt_transposed)(C *coeffs, Z(set) set,int l, int k, int m,unsigned int nfsoft_flags);
      \
NFFT_EXTERN void X(init)(X(plan) *plan, int N, int M); \
NFFT_EXTERN void X(init_advanced)(X(plan) *plan, int N, int M,unsigned int nfsoft_flags); \
NFFT_EXTERN void X(init_guru)(X(plan) *plan, int N, int M,unsigned int nfsoft_flags,unsigned int
      nfft_flags,int nfft_cutoff,int fpt_kappa); \
NFFT_EXTERN void X(init_guru_advanced)(X(plan) *plan, int N, int M,unsigned int nfsoft_flags,unsigned int
      nfft_flags,int nfft_cutoff,int fpt_kappa, int nn_oversampled); \
NFFT_EXTERN void X(trafo)(X(plan) *plan_nfsoft); \
NFFT_EXTERN void X(adjoint)(X(plan) *plan_nfsoft); \
NFFT_EXTERN void X(finalize)(X(plan) *plan); \
NFFT_EXTERN int X(posN)(int n,int m, int B);
```

Definition at line 641 of file nfft3.h.

7.14.3 Function Documentation

7.14.3.1 nfftf_vrand_unit_complex()

```
void nfftf_vrand_unit_complex (
              fftwf_complex * x,
              const NFFT_INT n )
```

Inits a vector of random complex numbers in $[0,1] \times [0,1]\mathrm{i}$.

7.14.3.2 nfftf_vrand_shifted_unit_double()

```
void nfftf_vrand_shifted_unit_double (
              float * x,
              const NFFT_INT n )
```

Inits a vector of random double numbers in $[-1/2, 1/2]$.

7.14.3.3 nfftf_get_window_name()

```
const char* nfftf_get_window_name ( )
```

- Return name of window function. $**$ The window function to be used is configured at compile time.

7.14.3.4 nfftl_vrand_unit_complex()

```
void nfftl_vrand_unit_complex (
            fftwl_complex * x,
            const NFFT_INT n )
```

Inits a vector of random complex numbers in $[0,1] \times [0,1]\mathrm{i}$.

7.14.3.5 nfftl_vrand_shifted_unit_double()

```
void nfftl_vrand_shifted_unit_double (
            long double * x,
            const NFFT_INT n )
```

Inits a vector of random double numbers in $[-1/2, 1/2]$.

7.14.3.6 nfftl_get_window_name()

```
const char* nfftl_get_window_name ( )
```

- Return name of window function. $**$ The window function to be used is configured at compile time.

7.15 nfsft.c File Reference

Implementation file for the NFSFT module.

```
#include "config.h"
#include <math.h>
#include <stdlib.h>
#include <string.h>
#include "nfft3.h"
#include "infft.h"
#include "legendre.h"
#include "api.h"
```

Macros

- #define NFSFT_DEFAULT_NFFT_CUTOFF 6

  The default NFFT cutoff parameter.
- #define NFSFT_DEFAULT_THRESHOLD 1000

  The default threshold for the FPT.
- #define NFSFT_BREAK_EVEN 5

  The break-even bandwidth $N \in \mathbb{N}_0$.

Functions

- static void c2e (nfsft_plan *plan)

  Converts coefficients $(b_k^n)_{k=0}^M$ with $M \in \mathbb{N}_0$, $-M \leq n \leq M$ from a linear combination of Chebyshev polynomials.
- static void c2e_transposed (nfsft_plan *plan)

  Transposed version of the function c2e.
- void nfsft_init (nfsft_plan *plan, int N, int M)
- void nfsft_init_advanced (nfsft_plan *plan, int N, int M, unsigned int flags)
- void **nfsft_init_guru** (nfsft_plan *plan, int N, int M, unsigned int flags, unsigned int nfft_flags, int nfft_cutoff)
- void nfsft_precompute (int N, double kappa, unsigned int nfsft_flags, unsigned int fpt_flags)
- void nfsft_forget (void)
- void nfsft_finalize (nfsft_plan *plan)
- static void **nfsft_set_f_nan** (nfsft_plan *plan)
- void **nfsft_trafo_direct** (nfsft_plan *plan)
- static void **nfsft_set_f_hat_nan** (nfsft_plan *plan)
- void **nfsft_adjoint_direct** (nfsft_plan *plan)
- void nfsft_trafo (nfsft_plan *plan)
- void nfsft_adjoint (nfsft_plan *plan)
- void **nfsft_precompute_x** (nfsft_plan *plan)

Variables

- static struct nfsft_wisdom wisdom = {false,0U,-1,-1,0,0,0,0,0}

  The global wisdom structure for precomputed data.

7.15.1 Detailed Description

Implementation file for the NFSFT module.

Author

Jens Keiner

7.16 polar_fft_test.c File Reference

NFFT-based polar FFT and inverse.

```
#include <math.h>
#include <stdlib.h>
#include <complex.h>
#include "nfft3mp.h"
```

Functions

- static int polar_grid (int T, int S, NFFT_R *x, NFFT_R *w)

  Generates the points $x_{t,j}$ with weights $w_{t,j}$ for the polar grid with T angles and R offsets.
- static int polar_dft (NFFT_C *f_hat, int NN, NFFT_C *f, int T, int S, int m)

  discrete polar FFT
- static int polar_fft (NFFT_C *f_hat, int NN, NFFT_C *f, int T, int S, int m)

  NFFT-based polar FFT.
- static int inverse_polar_fft (NFFT_C *f, int T, int S, NFFT_C *f_hat, int NN, int max_i, int m)

  inverse NFFT-based polar FFT
- int main (int argc, char **argv)

  test program for various parameters

7.16.1 Detailed Description

NFFT-based polar FFT and inverse.

Computes the NFFT-based polar FFT and its inverse for various parameters.

Author

Markus Fenn

Date

2006

7.17 radon.c File Reference

NFFT-based discrete Radon transform.

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>
#include <complex.h>
#include "nfft3mp.h"
```

Macros

- #define **NFFT_PRECISION_DOUBLE**
- #define KERNEL(r) (NFFT_K(1.0)-NFFT_M(fabs)((NFFT_R)(r))/((NFFT_R)S/2))

  define weights of kernel function for discrete Radon transform

Functions

- static int polar_grid (int T, int S, NFFT_R *x, NFFT_R *w)

  generates the points x with weights w for the polar grid with T angles and R offsets
- static int linogram_grid (int T, int S, NFFT_R *x, NFFT_R *w)

  generates the points x with weights w for the linogram grid with T slopes and R offsets
- static int Radon_trafo (int(*gridfcn)(), int T, int S, NFFT_R *f, int NN, NFFT_R *Rf)

  computes the NFFT-based discrete Radon transform of f on the grid given by gridfcn() with T angles and R offsets
- int main (int argc, char **argv)

  simple test program for the discrete Radon transform

7.17.1 Detailed Description

NFFT-based discrete Radon transform.

Computes the discrete Radon transform

$$R_{\theta_t} f\left(\frac{s}{R}\right) = \sum_{r \in I_R} w_r \sum_{k \in I_N^2} f_k \mathrm{e}^{-2\pi \mathrm{I} k \left(\frac{r}{R}\theta_t\right)} \mathrm{e}^{2\pi \mathrm{i} r s/R} \qquad (t \in I_T, s \in I_R).$$

by taking the 2D-NFFT of f_k ($k \in I_N^2$) at the points $\frac{r}{R}\theta_t$ of the polar or linogram grid followed by 1D-iFFTs for every direction $t \in T$, where w_r are the weights of the Dirichlet- or Fejer-kernel.

Author

Markus Fenn

Date

2005

7.18 solver.c File Reference

Implementation file for the solver module.

```
#include "config.h"
#include "nfft3.h"
#include "infft.h"
```

Macros

- #define **X**(name) CONCAT(solver_,name)

Functions

- void **CONCAT** (CONCAT(solver_, init_advanced_complex)
- void **CONCAT** (CONCAT(solver_, init_complex)
- void **CONCAT** (CONCAT(solver_, before_loop_complex)
- static void solver_loop_one_step_landweber_complex (CONCAT(solver_, plan_complex) *ths)

  void solver_loop_one_step_landweber
- static void solver_loop_one_step_steepest_descent_complex (CONCAT(solver_, plan_complex) *ths)

  void solver_loop_one_step_steepest_descent
- static void solver_loop_one_step_cgnr_complex (CONCAT(solver_, plan_complex) *ths)

  void solver_loop_one_step_cgnr
- static void solver_loop_one_step_cgne_complex (CONCAT(solver_, plan_complex) *ths)

  void solver_loop_one_step_cgne
- void CONCAT (CONCAT(solver_, loop_one_step_complex)

  void solver_loop_one_step
- void CONCAT (CONCAT(solver_, finalize_complex)

  void solver_finalize
- void CONCAT (CONCAT(solver_, init_advanced_double)

  void solver_finalize
- void **CONCAT** (CONCAT(solver_, init_double)
- void **CONCAT** (CONCAT(solver_, before_loop_double)
- static void solver_loop_one_step_landweber_double (CONCAT(solver_, plan_double) *ths)

  void solver_loop_one_step_landweber
- static void solver_loop_one_step_steepest_descent_double (CONCAT(solver_, plan_double) *ths)

  void solver_loop_one_step_steepest_descent
- static void solver_loop_one_step_cgnr_double (CONCAT(solver_, plan_double) *ths)

  void solver_loop_one_step_cgnr
- static void solver_loop_one_step_cgne_double (CONCAT(solver_, plan_double) *ths)

  void solver_loop_one_step_cgne
- void CONCAT (CONCAT(solver_, loop_one_step_double)

  void solver_loop_one_step
- void CONCAT (CONCAT(solver_, finalize_double)

  void solver_finalize

7.18.1 Detailed Description

Implementation file for the solver module.

Author

Stefan Kunis

7.19 taylor_nfft.c File Reference

Testing the nfft againt a Taylor expansion based version.

```
#include "config.h"
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <stdlib.h>
#include "nfft3.h"
#include "infft.h"
```

Data Structures

- struct taylor_plan

Functions

- static void taylor_init (taylor_plan *ths, int N, int M, int n, int m)

  Initialisation of a transform plan.
- static void taylor_precompute (taylor_plan *ths)

  Precomputation of weights and indices in Taylor expansion.
- static void taylor_finalize (taylor_plan *ths)

  Destroys a transform plan.
- static void taylor_trafo (taylor_plan *ths)

  *Executes a Taylor-NFFT, see equation (1.1) in [Guide], computes fast and approximate by means of a Taylor expansion for j=0,...,M-1 f[j] = sum_{k in I_N^ d} f_hat[k] * exp(-2 (pi) k x[j])*
- static void taylor_time_accuracy (int N, int M, int n, int m, int n_taylor, int m_taylor, unsigned test_accuracy)

  Compares NDFT, NFFT, and Taylor-NFFT.
- int **main** (int argc, char **argv)

7.19.1 Detailed Description

Testing the nfft againt a Taylor expansion based version.

Author

Stefan Kunis

References: Time and memory requirements of the Nonequispaced FFT

7.19.2 Function Documentation

7.19.2.1 taylor_init()

```
static void taylor_init (
             taylor_plan * ths,
             int N,
             int M,
             int n,
             int m )  [static]
```

Initialisation of a transform plan.

- ths The pointer to a taylor plan
- N The multi bandwidth
- M The number of nodes
- n The fft length
- m The order of the Taylor expansion

Author

Stefan Kunis

Definition at line 59 of file taylor_nfft.c.

7.19.2.2 taylor_precompute()

```
static void taylor_precompute (
            taylor_plan * ths )  [static]
```

Precomputation of weights and indices in Taylor expansion.

- ths The pointer to a taylor plan

Author

Stefan Kunis

Definition at line 77 of file taylor_nfft.c.

7.19.2.3 taylor_finalize()

```
static void taylor_finalize (
            taylor_plan * ths )  [static]
```

Destroys a transform plan.

- ths The pointer to a taylor plan

Author

Stefan Kunis, Daniel Potts

Definition at line 99 of file taylor_nfft.c.

7.19.2.4 taylor_trafo()

```
static void taylor_trafo (
            taylor_plan * ths )  [static]
```

Executes a Taylor-NFFT, see equation (1.1) in [Guide], computes fast and approximate by means of a Taylor expansion for j=0,...,M-1 f[j] = sum_{k in I_N^d} f_hat[k] * exp(-2 (pi) k x[j])

- ths The pointer to a taylor plan

Author

Stefan Kunis

Definition at line 117 of file taylor_nfft.c.

7.19.2.5 taylor_time_accuracy()

```
static void taylor_time_accuracy (
             int N,
             int M,
             int n,
             int m,
             int n_taylor,
             int m_taylor,
             unsigned test_accuracy )  [static]
```

Compares NDFT, NFFT, and Taylor-NFFT.

- N The bandwidth
- N The number of nodes
- n The FFT-size for the NFFT
- m The cut-off for window function
- n_taylor The FFT-size for the Taylor-NFFT
- m_taylor The order of the Taylor approximation
- test_accuracy Flag for NDFT computation

Author

Stefan Kunis

Definition at line 174 of file taylor_nfft.c.

7.20 wigner.h File Reference

Header file for functions related to Wigner-d/D functions.

Functions

- double SO3_alpha (int k, int m, int l)
  Computes three-term recurrence coefficients α_l^{km} of Wigner-d functions.
- double SO3_beta (int k, int m, int l)
  Computes three-term recurrence coefficients β_l^{km} of Wigner-d functions.
- double SO3_gamma (int k, int m, int l)
  Computes three-term recurrence coefficients γ_l^{km} of Wigner-d functions.
- void SO3_alpha_row (double *alpha, int N, int m, int n)
  Compute three-term-recurrence coefficients α_l^{km} of Wigner-d functions for all degrees $l = 0, \ldots, N$.
- void SO3_beta_row (double *beta, int N, int m, int n)
  Compute three-term-recurrence coefficients β_l^{km} of Wigner-d functions for all degrees $l = 0, \ldots, N$.
- void SO3_gamma_row (double *gamma, int N, int m, int n)
  Compute three-term-recurrence coefficients γ_l^{km} of Wigner-d functions for all degrees $l = 0, \ldots, N$.
- void SO3_alpha_matrix (double *alpha, int N, int n)

Compute three-term-recurrence coefficients α_l^{km} of Wigner-d functions for all order $m = -N, \ldots, N$ and degrees $l = 0, \ldots, N$.

- void SO3_beta_matrix (double *beta, int N, int n)

  Compute three-term-recurrence coefficients β_l^{km} of Wigner-d functions for all order $m = -N, \ldots, N$ and degrees $l = 0, \ldots, N$.
- void SO3_gamma_matrix (double *gamma, int N, int n)

  Compute three-term-recurrence coefficients γ_l^{km} of Wigner-d functions for all order $m = -N, \ldots, N$ and degrees $l = 0, \ldots, N$.
- void SO3_alpha_all (double *alpha, int N)

  Compute three-term-recurrence coefficients α_l^{km} of Wigner-d functions for all $k, m = -N, \ldots, N$ and $l = 0, \ldots, N$.
- void SO3_beta_all (double *beta, int N)

  Compute three-term-recurrence coefficients β_l^{km} of Wigner-d functions for all $k, m = -N, \ldots, N$ and $l = 0, \ldots, N$.
- void SO3_gamma_all (double *gamma, int N)

  Compute three-term-recurrence coefficients γ_l^{km} of Wigner-d functions for all $k, m = -N, \ldots, N$ and $l = 0, \ldots, N$.
- void eval_wigner (double *x, double *y, int size, int l, double *alpha, double *beta, double *gamma)

  Evaluates Wigner-d functions $d_l^{km}(x, c)$ using the Clenshaw-algorithm.
- int eval_wigner_thresh (double *x, double *y, int size, int l, double *alpha, double *beta, double *gamma, double threshold)

  Evaluates Wigner-d functions $d_l^{km}(x, c)$ using the Clenshaw-algorithm if it not exceeds a given threshold.
- double wigner_start (int n1, int n2, double theta)

  A method used for debugging, gives the values to start the "old" three-term recurrence generates $d_l^{km}(cos(theta))$ WHERE THE DEGREE l OF THE FUNCTION IS EQUAL TO THE MAXIMUM OF ITS ORDERS.

7.20.1 Detailed Description

Header file for functions related to Wigner-d/D functions.

Author

Antje Vollrath

7.20.2 Function Documentation

7.20.2.1 SO3_alpha()

```
double SO3_alpha (
            int k,
            int m,
            int l )
```

Computes three-term recurrence coefficients α_l^{km} of Wigner-d functions.

- k The order k
- m The order m
- l The degree l

Definition at line 25 of file wigner.c.

Referenced by SO3_alpha_all(), SO3_alpha_matrix(), and SO3_alpha_row().

7.20.2.2 SO3_beta()

```
double SO3_beta (
            int k,
            int m,
            int l )
```

Computes three-term recurrence coefficients β_l^{km} of Wigner-d functions.

- k The order k
- m The order m
- l The degree l

Definition at line 52 of file wigner.c.

Referenced by SO3_beta_all(), SO3_beta_matrix(), and SO3_beta_row().

7.20.2.3 SO3_gamma()

```
double SO3_gamma (
            int k,
            int m,
            int l )
```

Computes three-term recurrence coefficients γ_l^{km} of Wigner-d functions.

- k The order k
- m The order m
- l The degree l

Definition at line 73 of file wigner.c.

Referenced by SO3_gamma_all(), SO3_gamma_matrix(), and SO3_gamma_row().

7.20.2.4 SO3_alpha_row()

```
void SO3_alpha_row (
            double * alpha,
            int N,
            int m,
            int n )  [inline]
```

Compute three-term-recurrence coefficients α_l^{km} of Wigner-d functions for all degrees $l = 0, \ldots, N$.

- alpha A pointer to an array of doubles of size $(2N+1)^2(N+1)$
- m the first order
- n the second order
- N The upper bound N.

Definition at line 88 of file wigner.c.

References SO3_alpha().

7.20.2.5 SO3_beta_row()

```
void SO3_beta_row (
             double * beta,
             int N,
             int m,
             int n )  [inline]
```

Compute three-term-recurrence coefficients β_l^{km} of Wigner-d functions for all degrees $l = 0, \ldots, N$.

- alpha A pointer to an array of doubles of size $(2N+1)^2(N+1)$
- m the first order
- n the second order
- N The upper bound N.

Definition at line 96 of file wigner.c.

References SO3_beta().

7.20.2.6 SO3_gamma_row()

```
void SO3_gamma_row (
             double * gamma,
             int N,
             int m,
             int n )  [inline]
```

Compute three-term-recurrence coefficients γ_l^{km} of Wigner-d functions for all degrees $l = 0, \ldots, N$.

- alpha A pointer to an array of doubles of size $(2N+1)^2(N+1)$
- m the first order
- n the second order
- N The upper bound N.

Definition at line 104 of file wigner.c.

References SO3_gamma().

7.20.2.7 SO3_alpha_matrix()

```
void SO3_alpha_matrix (
            double * alpha,
            int N,
            int n )  [inline]
```

Compute three-term-recurrence coefficients α_l^{km} of Wigner-d functions for all order $m = -N, \ldots, N$ and degrees $l = 0, \ldots, N$.

- alpha A pointer to an array of doubles of size $(2N+1)^2(N+1)$
- n the second order
- N The upper bound N.

Definition at line 114 of file wigner.c.

References SO3_alpha().

7.20.2.8 SO3_beta_matrix()

```
void SO3_beta_matrix (
            double * beta,
            int N,
            int n )  [inline]
```

Compute three-term-recurrence coefficients β_l^{km} of Wigner-d functions for all order $m = -N, \ldots, N$ and degrees $l = 0, \ldots, N$.

- alpha A pointer to an array of doubles of size $(2N+1)^2(N+1)$
- n the second order
- N The upper bound N.

Definition at line 128 of file wigner.c.

References SO3_beta().

7.20.2.9 SO3_gamma_matrix()

```
void SO3_gamma_matrix (
            double * gamma,
            int N,
            int n )  [inline]
```

Compute three-term-recurrence coefficients γ_l^{km} of Wigner-d functions for all order $m = -N, \ldots, N$ and degrees $l = 0, \ldots, N$.

- alpha A pointer to an array of doubles of size $(2N+1)^2(N+1)$
- n the second order
- N The upper bound N.

Definition at line 142 of file wigner.c.

References SO3_gamma().

7.20.2.10 SO3_alpha_all()

```
void SO3_alpha_all (
            double * alpha,
            int N )  [inline]
```

Compute three-term-recurrence coefficients α_l^{km} of Wigner-d functions for all $k, m = -N, \ldots, N$ and $l = 0, \ldots, N$.

- alpha A pointer to an array of doubles of size $(2N+1)^2(N+1)$
- N The upper bound N.

Definition at line 158 of file wigner.c.

References SO3_alpha().

7.20.2.11 SO3_beta_all()

```
void SO3_beta_all (
            double * beta,
            int N )  [inline]
```

Compute three-term-recurrence coefficients β_l^{km} of Wigner-d functions for all $k, m = -N, \ldots, N$ and $l = 0, \ldots, N$.

- alpha A pointer to an array of doubles of size $(2N+1)^2(N+1)$
- N The upper bound N.

Definition at line 181 of file wigner.c.

References SO3_beta().

7.20.2.12 SO3_gamma_all()

```
void SO3_gamma_all (
            double * gamma,
            int N )  [inline]
```

Compute three-term-recurrence coefficients γ_l^{km} of Wigner-d functions for all $k, m = -N, \ldots, N$ and $l = 0, \ldots, N$.

- alpha A pointer to an array of doubles of size $(2N+1)^2(N+1)$
- N The upper bound N.

Definition at line 198 of file wigner.c.

References SO3_gamma().

7.20.2.13 eval_wigner()

```
void eval_wigner (
            double * x,
            double * y,
            int size,
            int l,
            double * alpha,
            double * beta,
            double * gamma )  [inline]
```

Evaluates Wigner-d functions $d_l^{km}(x, c)$ using the Clenshaw-algorithm.

- x A pointer to an array of nodes where the function is to be evaluated
- y A pointer to an array where the function values are returned
- size The length of x and y
- l The degree l
- alpha A pointer to an array containing the recurrence coefficients $\alpha_c^{km}, \ldots, \alpha_{c+l}^{km}$
- beta A pointer to an array containing the recurrence coefficients $\beta_c^{km}, \ldots, \beta_{c+l}^{km}$
- gamma A pointer to an array containing the recurrence coefficients $\gamma_c^{km}, \ldots, \gamma_{c+l}^{km}$

Definition at line 215 of file wigner.c.

7.20.2.14 eval_wigner_thresh()

```
int eval_wigner_thresh (
            double * x,
            double * y,
            int size,
            int l,
            double * alpha,
            double * beta,
            double * gamma,
            double threshold )  [inline]
```

Evaluates Wigner-d functions $d_l^{km}(x, c)$ using the Clenshaw-algorithm if it not exceeds a given threshold.

- x A pointer to an array of nodes where the function is to be evaluated
- y A pointer to an array where the function values are returned
- size The length of x and y
- l The degree l
- alpha A pointer to an array containing the recurrence coefficients $\alpha_c^{km}, \ldots, \alpha_{c+l}^{km}$
- beta A pointer to an array containing the recurrence coefficients $\beta_c^{km}, \ldots, \beta_{c+l}^{km}$
- gamma A pointer to an array containing the recurrence coefficients $\gamma_c^{km}, \ldots, \gamma_{c+l}^{km}$
- threshold The threshold

Definition at line 260 of file wigner.c.

7.20.2.15 wigner_start()

```
double wigner_start (
            int n1,
            int n2,
            double theta )
```

A method used for debugging, gives the values to start the "old" three-term recurrence generates $d_l^{km}(cos(theta))$ WHERE THE DEGREE l OF THE FUNCTION IS EQUAL TO THE MAXIMUM OF ITS ORDERS.

- theta the argument of
- n1 the first order
- n2 the second order

Returns

the function value $d_l^{km}(cos(theta))$

Definition at line 317 of file wigner.c.

Index