NFFT Reference Manual

3.0

Generated by Doxygen 1.4.4

Wed Nov 1 21:58:49 2006

Contents

Chapter 1

NFFT 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

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

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.

Chapter 2

NFFT Module Index

2.1 NFFT Modules

Here is a list of all modules:

Chapter 3

NFFT Directory Hierarchy

3.1 NFFT Directories

This directory hierarchy is sorted roughly, but not completely, alphabetically:

Chapter 4

NFFT Data Structure Index

4.1 NFFT Data Structures

Here are the data structures with brief descriptions:

Chapter 5

NFFT File Index

5.1 NFFT File List

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

Chapter 6

NFFT Module Documentation

6.1 NFFT - Nonequispaced fast Fourier transform

Direct and fast computation of the nonequispaced discrete Fourier transform.

Data Structures

- struct nfft_plan

  Structure for a NFFT plan.

Defines

- #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.*

- #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.*

- #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.*

- #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).*

- #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.*

- #define PRE_FULL_PSI (1U<< 5)

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

- #define MALLOC_X (1U<< 6)

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

- #define MALLOC_F_HAT (1U<< 7)

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

- #define MALLOC_F (1U<< 8)

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

- #define FFT_OUT_OF_PLACE (1U<< 9)

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

- #define FFTW_INIT (1U<< 10)

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

- #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 $\mathbf{B}$ *).*

Functions

- void ndft_trafo (nfft_plan *ths)

  Computes a NDFT, see the definition.

- void ndft_adjoint (nfft_plan *ths)

  Computes an adjoint NDFT, see the definition.

- void nfft_trafo (nfft_plan *ths)

  Computes a NFFT, see the definition.

- void nfft_adjoint (nfft_plan *ths)

  Computes an adjoint NFFT, see the definition.

- void nfft_init_1d (nfft_plan *ths, int N1, int M)

  Initialisation of a transform plan, wrapper d=1.

- void nfft_init_2d (nfft_plan *ths, int N1, int N2, int M)

  Initialisation of a transform plan, wrapper d=2.

- void nfft_init_3d (nfft_plan *ths, int N1, int N2, int N3, int M)

  Initialisation of a transform plan, wrapper d=3.

- void nfft_init (nfft_plan *ths, int d, int *N, int M)

  Initialisation of a transform plan, simple.

- void nfft_init_advanced (nfft_plan *ths, int d, int *N, int M, unsigned nfft_flags_on, unsigned nfft_flags_off)

  Initialisation of a transform plan, advanced.

- void nfft_init_guru (nfft_plan *ths, int d, int *N, int M, int *n, int m, unsigned nfft_flags, unsigned fftw_flags)

  Initialisation of a transform plan, guru.

- void nfft_precompute_one_psi (nfft_plan *ths)

  Precomputation for a transform plan.

- void nfft_precompute_full_psi (nfft_plan *ths)

  Superceded by nfft_precompute_one_psi.

- void nfft_precompute_psi (nfft_plan *ths)

  Superceded by nfft_precompute_one_psi.

- void nfft_precompute_lin_psi (nfft_plan *ths)

  Superceded by nfft_precompute_one_psi.

- void nfft_check (nfft_plan *ths)

  Checks a transform plan for frequently used bad parameter.

- void nfft_finalize (nfft_plan *ths)

  Destroys a transform plan.

6.1.1 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

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

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 $\mathcal{X} := \{\mathbf{x}_j \in \mathbb{T}^d : \ j = 0, \ldots, M-1\}$. Possible frequencies $\mathbf{k}$ are collected in the multi index set

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

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

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

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 ndft_trafo and ndft_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.

6.1.2 Define Documentation

6.1.2.1 #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 92 of file nfft3.h.

Referenced by accuracy_pre_lin_psi(), construct(), fastsum_init_guru(), fgt_init_guru(), glacier(), inverse_linogram_fft(), inverse_mpolar_fft(), inverse_polar_fft(), Inverse_Radon_trafo(), linogram_dft(), linogram_fft(), main(), mpolar_dft(), mpolar_fft(), nfct_finalize(), nfft_finalize(), nfft_init(), nfsft_init_advanced(), nfst_finalize(), nnfft_finalize(), nnfft_init(), nnfft_init_guru(), polar_dft(), polar_fft(), Radon_trafo(), reconstruct(), taylor_time_accuracy(), and time_accuracy().

6.1.2.2 #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 105 of file nfft3.h.

Referenced by time_accuracy().

6.1.2.3 #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 122 of file nfft3.h.

Referenced by accuracy_pre_lin_psi(), fastsum_precompute(), inverse_linogram_fft(), inverse_mpolar_fft(), inverse_polar_fft(), Inverse_Radon_trafo(), linogram_fft(), mpolar_fft(), nfft_finalize(), nfft_precompute_one_psi(), nnfft_finalize(), nnfft_init_guru(), polar_fft(), Radon_trafo(), reconstruct(), and time_accuracy().

6.1.2.4 #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 135 of file nfft3.h.

Referenced by nfft_finalize(), nfft_precompute_one_psi(), taylor_time_accuracy(), and time_accuracy().

6.1.2.5 #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 147 of file nfft3.h.

Referenced by construct(), fastsum_init_guru(), fastsum_precompute(), fgt_init_guru(), fgt_init_node_dependent(), inverse_linogram_fft(), inverse_mpolar_fft(), inverse_polar_fft(), Inverse_Radon_trafo(), linogram_dft(), linogram_fft(), main(), mpolar_dft(), mpolar_fft(), nfct_finalize(), nfft_finalize(), nfft_init(), nfft_precompute_one_psi(), nfsft_init_advanced(), nfst_finalize(), nfst_full_psi(), nnfft_finalize(), nnfft_init(), nnfft_init_guru(), polar_dft(), polar_fft(), Radon_trafo(), reconstruct(), and time_accuracy().

6.1.2.6 #define PRE_FULL_PSI (1U<< 5)

If this flag is set, the convolution step (the multiplication with the sparse matrix $\mathbf{B}$) uses $(2m+2)^d M$ 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 160 of file nfft3.h.

Referenced by fastsum_precompute(), glacier(), inverse_linogram_fft(), inverse_mpolar_fft(), inverse_polar_fft(), Inverse_Radon_trafo(), linogram_fft(), mpolar_fft(), nfct_adjoint(), nfct_trafo(), nfft_finalize(), nfft_precompute_one_psi(), nfst_finalize(), nfst_precompute_psi(), nnfft_finalize(), nnfft_init_guru(), polar_fft(), Radon_trafo(), reconstruct(), and time_accuracy().

6.1.2.7 #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 171 of file nfft3.h.

Referenced by accuracy_pre_lin_psi(), construct(), fastsum_init_guru(), fgt_init_guru(), glacier(), inverse_linogram_fft(), inverse_mpolar_fft(), inverse_polar_fft(), Inverse_Radon_trafo(), linogram_dft(), linogram_fft(), mpolar_dft(), mpolar_fft(), nfct_finalize(), nfft_finalize(), nfft_init(), nfst_finalize(), nnfft_finalize(), nnfft_init(), polar_dft(), polar_fft(), Radon_trafo(), reconstruct(), taylor_init(), and time_accuracy().

6.1.2.8 #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 183 of file nfft3.h.

Referenced by accuracy_pre_lin_psi(), construct(), fastsum_init_guru(), fgt_init_guru(), glacier(), inverse_linogram_fft(), inverse_mpolar_fft(), inverse_polar_fft(), Inverse_Radon_trafo(), linogram_dft(), linogram_fft(), mpolar_dft(), mpolar_fft(), nfct_finalize(), nfft_finalize(), nfft_init(), nfst_finalize(), nnfft_finalize(), nnfft_init(), nnfft_init_guru(), polar_dft(), polar_fft(), Radon_trafo(), reconstruct(), taylor_init(), and time_accuracy().

6.1.2.9 #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 194 of file nfft3.h.

Referenced by accuracy_pre_lin_psi(), construct(), fastsum_init_guru(), glacier(), inverse_linogram_fft(), inverse_mpolar_fft(), inverse_polar_fft(), Inverse_Radon_trafo(), linogram_dft(), linogram_fft(), mpolar_dft(), mpolar_fft(), nfct_finalize(), nfft_finalize(), nfft_init(), nfst_finalize(), nnfft_finalize(), nnfft_init(), polar_dft(), polar_fft(), Radon_trafo(), reconstruct(), taylor_init(), and time_accuracy().

6.1.2.10 #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 205 of file nfft3.h.

Referenced by accuracy_pre_lin_psi(), construct(), fastsum_init_guru(), glacier(), inverse_linogram_fft(), inverse_mpolar_fft(), inverse_polar_fft(), Inverse_Radon_trafo(), linogram_dft(), linogram_fft(), main(), mpolar_dft(), mpolar_fft(), nfct_finalize(), nfft_finalize(), nfft_init(), nfsft_init_advanced(), nfst_finalize(), nnfft_init(), nnfft_init_guru(), polar_dft(), polar_fft(), Radon_trafo(), reconstruct(), taylor_init(), taylor_time_accuracy(), and time_accuracy().

6.1.2.11 #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 216 of file nfft3.h.

Referenced by accuracy_pre_lin_psi(), construct(), fastsum_init_guru(), fgt_init_guru(), glacier(), inverse_linogram_fft(), inverse_mpolar_fft(), inverse_polar_fft(), Inverse_Radon_trafo(), linogram_dft(), linogram_fft(), main(), mpolar_dft(), mpolar_fft(), nfct_finalize(), nfft_finalize(), nfft_init(), nfsft_init_advanced(), nfst_finalize(), nnfft_init(), nnfft_init_guru(), polar_dft(), polar_fft(), Radon_trafo(), reconstruct(), taylor_init(), taylor_time_accuracy(), and time_accuracy().

6.1.2.12 #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 231 of file nfft3.h.

Referenced by glacier(), nfsft_precompute_x(), simple_test_infft_1d(), and taylor_time_accuracy().

6.1.3 Function Documentation

6.1.3.1 void ndft_trafo (nfft_plan * *ths*)

Computes a NDFT, see the definition.

< index over all nodes

< index for dimensions

< plain index for summation

< dito

< actual Fourier coefficient

< actual sample

< actual node x[d*j+t]

< multi index for summation

< sign times 2∗pi∗k∗x

Definition at line 129 of file nfft.c.

Referenced by accuracy_pre_lin_psi(), fgt_trafo(), linogram_dft(), mpolar_dft(), ndft_time(), nfsft_trafo(), polar_dft(), taylor_time_accuracy(), and time_accuracy().

6.1.3.2 void ndft_adjoint (nfft_plan ∗ *ths*)

Computes an adjoint NDFT, see the definition.

< index over all nodes

< index for dimensions

< plain index for summation

< dito

< actual Fourier coefficient

< actual sample

< actual node x[d∗j+t]

< multi index for summation

< sign times 2∗pi∗k∗x

Definition at line 130 of file nfft.c.

Referenced by fgt_trafo(), and nfsft_adjoint().

6.1.3.3 void nfft_trafo (nfft_plan ∗ *ths*)

Computes a NFFT, see the definition.

form $\hat{g}_k = \frac{\hat{f}_k}{c_k(\phi)}$ for $k \in I_N$

compute by d-variate discrete Fourier transform $g_l = \sum_{k \in I_N} \hat{g}_k \mathrm{e}^{-2\pi \mathrm{i} \frac{kl}{n}}$ for $l \in I_n$

set $f_j = \sum_{l \in I_n, m(x_j)} g_l \psi\left(x_j - \frac{l}{n}\right)$ for $j = 0, \ldots, M_total - 1$

Definition at line 550 of file nfft.c.

Referenced by accuracy_pre_lin_psi(), construct(), fastsum_trafo(), fgt_trafo(), linogram_fft(), mpolar_fft(), mri_inh_2d1d_trafo(), mri_inh_3d_trafo(), ndft_time(), nfsft_trafo(), nnfft_trafo(), polar_fft(), Radon_trafo(), simple_test_infft_1d(), taylor_time_accuracy(), and time_accuracy().

6.1.3.4 void nfft_adjoint (nfft_plan ∗ *ths*)

Computes an adjoint NFFT, see the definition.

set $g_l = \sum_{j=0}^{M_total-1} f_j \psi\left(x_j - \frac{l}{n}\right)$ for $l \in I_n, m(x_j)$

compute by d-variate discrete Fourier transform $\hat{g}_k = \sum_{l \in I_n} g_l \mathrm{e}^{+2\pi \mathrm{i} \frac{kl}{n}}$ for $k \in I_N$

form $\hat{f}_k = \frac{\hat{g}_k}{c_k(\phi)}$ for $k \in I_N$

Definition at line 579 of file nfft.c.

References nfft_plan::g, nfft_plan::g1, nfft_plan::g2, nfft_plan::g_hat, and nfft_plan::my_fftw_plan2.

Referenced by fastsum_trafo(), fgt_trafo(), mri_inh_2d1d_adjoint(), mri_inh_3d_adjoint(), nfsft_adjoint(), nnfft_adjoint(), and reconstruct().

6.1.3.5 void nfft_init_1d (nfft_plan * *ths*, int *N1*, int *M*)

Initialisation of a transform plan, wrapper d=1.

- ths The pointer to a nfft plan
- N1 bandwidth
- M The number of nodes

Author:
Stefan Kunis, Daniel Potts

Definition at line 855 of file nfft.c.

References nfft_init().

Referenced by ndft_time(), and simple_test_infft_1d().

6.1.3.6 void nfft_init_2d (nfft_plan * *ths*, int *N1*, int *N2*, int *M*)

Initialisation of a transform plan, wrapper d=2.

- ths The pointer to a nfft plan
- N1 bandwidth
- N2 bandwidth
- M The number of nodes

Author:
Stefan Kunis, Daniel Potts

Definition at line 863 of file nfft.c.

References nfft_init().

Referenced by construct().

6.1.3.7 void nfft_init_3d (nfft_plan * *ths*, int *N1*, int *N2*, int *N3*, int *M*)

Initialisation of a transform plan, wrapper d=3.

- ths The pointer to a nfft plan
- N1 bandwidth
- N2 bandwidth
- N3 bandwidth

- M The number of nodes

Author:
Stefan Kunis, Daniel Potts

Definition at line 872 of file nfft.c.

References nfft_init().

6.1.3.8 void nfft_init (nfft_plan * *ths*, int *d*, int * *N*, int *M_total*)

Initialisation of a transform plan, simple.

< index over all dimensions

Definition at line 810 of file nfft.c.

References nfft_plan::d, FFT_OUT_OF_PLACE, nfft_plan::fftw_flags, FFTW_INIT, nfft_plan::M_total, MALLOC_F, MALLOC_F_HAT, MALLOC_X, nfft_plan::N, nfft_plan::n, nfft_plan::nfft_flags, nfft_next_power_of_2(), PRE_PHI_HUT, and PRE_PSI.

Referenced by nfft_init_1d(), nfft_init_2d(), and nfft_init_3d().

6.1.3.9 void nfft_init_advanced (nfft_plan * *ths*, int *d*, int * *N*, int *M*, unsigned *nfft_flags_on*, unsigned *nfft_flags_off*)

Initialisation of a transform plan, advanced.

NOT YET IMPLEMENTED!!

- ths The pointer to a nfft plan
- d The dimension
- N The multi bandwidth
- M The number of nodes
- nfft_flags_on NFFT flags to switch on
- nfft_flags_off NFFT flags to switch off

Author:
Stefan Kunis, Daniel Potts

6.1.3.10 void nfft_init_guru (nfft_plan * *ths*, int *d*, int * *N*, int *M_total*, int * *n*, int *m*, unsigned *nfft_flags*, unsigned *fftw_flags*)

Initialisation of a transform plan, guru.

< index over all dimensions

Definition at line 835 of file nfft.c.

References nfft_plan::d, nfft_plan::fftw_flags, nfft_plan::m, nfft_plan::M_total, nfft_plan::N, nfft_plan::n, and nfft_plan::nfft_flags.

Referenced by accuracy_pre_lin_psi(), construct(), fastsum_init_guru(), fgt_init_guru(), glacier(), inverse_linogram_fft(), inverse_mpolar_fft(), inverse_polar_fft(), Inverse_Radon_trafo(), linogram_dft(), linogram_fft(), mpolar_dft(), mpolar_fft(), mri_inh_2d1d_init_guru(), mri_inh_3d_init_guru(), nfsft_init_guru(), polar_dft(), polar_fft(), Radon_trafo(), reconstruct(), taylor_init(), taylor_time_accuracy(), and time_accuracy().

6.1.3.11 void nfft_precompute_one_psi (nfft_plan * *ths*)

Precomputation for a transform plan.

- ths The pointer to a nfft plan

Author:
Stefan Kunis

wrapper for precompute*_psi

if PRE_*_PSI is set the application program has to call this routine (after) setting the nodes x

Definition at line 727 of file nfft.c.

References nfft_plan::nfft_flags, nfft_precompute_full_psi(), nfft_precompute_lin_psi(), nfft_precompute_psi(), PRE_FG_PSI, PRE_FULL_PSI, PRE_LIN_PSI, and PRE_PSI.

Referenced by accuracy_pre_lin_psi(), glacier(), nfsft_precompute_x(), simple_test_infft_1d(), taylor_time_accuracy(), and time_accuracy().

6.1.3.12 void nfft_precompute_full_psi (nfft_plan * *ths*)

Superceded by nfft_precompute_one_psi.

< index over all dimensions

< index over all nodes

< plain index 0<=l_L<lprod

< multi index u<=l<=o

< multi index 0<=lj<u+o+1

< postfix plain index

< 'bandwidth' of matrix B

< depends on x_j

Definition at line 686 of file nfft.c.

References nfft_plan::d, nfft_plan::m, nfft_plan::M_total, nfft_plan::psi, nfft_plan::psi_index_f, and nfft_plan::psi_index_g.

Referenced by fastsum_precompute(), inverse_linogram_fft(), inverse_mpolar_fft(), inverse_polar_fft(), Inverse_Radon_trafo(), linogram_fft(), mpolar_fft(), nfft_precompute_one_psi(), nnfft_precompute_full_psi(), polar_fft(), Radon_trafo(), and reconstruct().

6.1.3.13 void nfft_precompute_psi (nfft_plan * *ths*)

Superceded by nfft_precompute_one_psi.

< index over all dimensions

< index over all nodes

< index u<=l<=o

< index 0<=lj<u+o+1

< depends on x_j

Definition at line 666 of file nfft.c.

References nfft_plan::d, nfft_plan::m, nfft_plan::M_total, nfft_plan::n, nfft_plan::psi, and nfft_plan::x.

Referenced by construct(), fastsum_precompute(), fgt_init_node_dependent(), inverse_linogram_fft(), inverse_mpolar_fft(), inverse_polar_fft(), Inverse_Radon_trafo(), linogram_fft(), mpolar_fft(), nfft_precompute_one_psi(), nnfft_precompute_psi(), polar_fft(), Radon_trafo(), and reconstruct().

6.1.3.14 void nfft_precompute_lin_psi (nfft_plan * *ths*)

Superceded by nfft_precompute_one_psi.

< index over all dimensions

< index over all nodes

< step size in [0,(m+1)/n]

Definition at line 630 of file nfft.c.

References nfft_plan::K, nfft_plan::m, nfft_plan::n, and nfft_plan::psi.

Referenced by fastsum_precompute(), inverse_linogram_fft(), inverse_mpolar_fft(), inverse_polar_fft(), Inverse_Radon_trafo(), linogram_fft(), mpolar_fft(), nfft_precompute_one_psi(), nnfft_precompute_lin_psi(), polar_fft(), Radon_trafo(), and reconstruct().

6.1.3.15 void nfft_check (nfft_plan * *ths*)

Checks a transform plan for frequently used bad parameter.

- ths The pointer to a nfft plan

Author:
Stefan Kunis, Daniel Potts

Definition at line 882 of file nfft.c.

References nfft_plan::d, nfft_plan::m, nfft_plan::M_total, nfft_plan::N, nfft_plan::sigma, and nfft_plan::x.

6.1.3.16 void nfft_finalize (nfft_plan * *ths*)

Destroys a transform plan.

- ths The pointer to a nfft plan

Author:
Stefan Kunis, Daniel Potts

Definition at line 905 of file nfft.c.

References nfft_plan::c_phi_inv, nfft_plan::d, nfft_plan::f, nfft_plan::f_hat, FFT_OUT_OF_PLACE, FFTW_INIT, nfft_plan::g1, nfft_plan::g2, MALLOC_F, MALLOC_F_HAT, MALLOC_X, nfft_plan::my_fftw_plan1, nfft_plan::my_fftw_plan2, nfft_plan::n, nfft_plan::N, nfft_plan::nfft_flags, PRE_FG_PSI, PRE_FULL_PSI, PRE_LIN_PSI, PRE_PHI_HUT, PRE_PSI, nfft_plan::psi, nfft_plan::psi_index_f, nfft_plan::psi_index_g, nfft_plan::sigma, and nfft_plan::x.

Referenced by accuracy_pre_lin_psi(), construct(), fastsum_finalize(), fgt_finalize(), glacier(), inverse_linogram_fft(), inverse_mpolar_fft(), inverse_polar_fft(), Inverse_Radon_trafo(), linogram_dft(), linogram_fft(), mpolar_dft(), mpolar_fft(), mri_inh_2d1d_finalize(), mri_inh_3d_finalize(), ndft_time(), nfsft_finalize(), nnfft_finalize(), polar_dft(), polar_fft(), Radon_trafo(), reconstruct(), simple_test_infft_1d(), taylor_finalize(), taylor_time_accuracy(), and time_accuracy().

6.2 NFCT/NFST - Nonequispaced fast (co)sine transform

Direct and fast computation of the discrete nonequispaced (co)sine transform.

Data Structures

- struct nfct_plan

  Structure for a transform plan.

- struct nfst_plan

  Structure for a transform plan.

Functions

- void nfct_init_1d (nfct_plan *ths_plan, int N0, int M_total)

  Creates a 1-dimensional transform plan.

- void nfct_init_2d (nfct_plan *ths_plan, int N0, int N1, int M_total)

  Creates a 3-dimensional transform plan.

- void nfct_init_3d (nfct_plan *ths_plan, int N0, int N1, int N2, int M_total)

  Creates a 3-dimensional transform plan.

- void nfct_init (nfct_plan *ths_plan, int d, int *N, int M_total)

  Creates a d-dimensional transform plan.

- void nfct_init_guru (nfct_plan *ths_plan, int d, int *N, int M_total, int *n, int m, unsigned nfct_flags, unsigned fftw_flags)

  Creates a d-dimensional transform plan.

- void nfct_precompute_psi (nfct_plan *ths_plan)

  precomputes the values psi if the PRE_PSI is set the application program has to call this routine after setting the nodes this_plan->x

- void nfct_trafo (nfct_plan *ths_plan)

  executes a NFCT (approximate,fast), computes for $j = 0, ..., M\_total - 1$ $f_j^C(x_j) = \sum_{k \in I_0^{N,d}} \hat{f}_k^C * cos(2\pi k x_j)$

- void ndct_trafo (nfct_plan *ths_plan)

  executes a NDCT (exact,slow), computes for $j = 0, ..., M\_total - 1$ $f_j^C(x_j) = \sum_{k \in I_0^{N,d}} \hat{f}_k^C * cos(2\pi k x_j)$

- void nfct_adjoint (nfct_plan *ths_plan)

  executes a transposed NFCT (approximate,fast), computes for $k \in I_0^{N,d}$ $h^C(k) = \sum_{j \in I_0^{(M\_total,1)}} f_j^C * cos(2\pi k x_j)$

- void ndct_adjoint (nfct_plan *ths_plan)

  executes a direct transposed NDCT (exact,slow), computes for $k \in I_0^{N,d}$ $h^C(k) = \sum_{j \in I_0^{(M\_total,1)}} f_j^C * cos(2\pi k x_j)$

- void nfct_finalize (nfct_plan *ths_plan)

  Destroys a plan.

- double nfct_phi_hut (nfct_plan *ths_plan, int k, int d)

  do some adjustments (N,n) then compute PHI_HUT

- double nfct_phi (nfct_plan *ths_plan, double x, int d)

  do some adjustments (N,n) then compute PHI

- int nfct_fftw_2N (int n)

  returns 2(n-1), fftw related issue

- int nfct_fftw_2N_rev (int n)

  returns 0.5n+1, fftw related issue

- void nfst_init_1d (nfst_plan *ths_plan, int N0, int M_total)

  Creates a 1-dimensional transform plan.

- void nfst_init_2d (nfst_plan *ths_plan, int N0, int N1, int M_total)

  Creates a 3-dimensional transform plan.

- void nfst_init_3d (nfst_plan *ths_plan, int N0, int N1, int N2, int M_total)

  Creates a 3-dimensional transform plan.

- void nfst_init (nfst_plan *ths_plan, int d, int *N, int M_total)

  Creates a d-dimensional transform plan.

- void nfst_init_m (nfst_plan *ths_plan, int d, int *N, int M_total, int m)

  Creates a d-dimensional transform plan with pcific m.

- void nfst_init_guru (nfst_plan *ths_plan, int d, int *N, int M_total, int *n, int m, unsigned nfst_flags, unsigned fftw_flags)

  Creates a d-dimensional transform plan.

- void nfst_precompute_psi (nfst_plan *ths_plan)

  precomputes the values psi if the PRE_PSI is set the application program has to call this routine after setting the nodes this_plan->x

- void nfst_trafo (nfst_plan *ths_plan)

  executes a NFST (approximate,fast), computes for $j = 0, ..., M\_total - 1$ $f_j^S(x_j) = \sum_{k \in I_1^{N,d}} \hat{f}_k^S * sin(2\pi k x_j)$

- void ndst_trafo (nfst_plan *ths_plan)

  executes a NDST (exact,slow), computes for $j = 0, ..., M\_total - 1$ $f_j^S(x_j) = \sum_{k \in I_1^{N,d}} \hat{f}_k^S * sin(2\pi k x_j)$

- void nfst_adjoint (nfst_plan *ths_plan)

  executes a transposed NFST (approximate,fast), computes for $k \in I_1^{N,d}$ $h^S(k) = \sum_{j \in I_0^{M\_total,1}} f_j^S * cos(2\pi k x_j)$

- void ndst_adjoint (nfst_plan *ths_plan)

  executes a direct transposed NDST (exact,slow), computes for $k \in I_1^{N,d}$ $h^S(k) = \sum_{j \in I_0^{M\_total,1}} f_j^S * cos(2\pi k x_j)$

- void nfst_finalize (nfst_plan *ths_plan)

  Destroys a plan.

- void nfst_full_psi (nfst_plan *ths_plan, double eps)

  more memory usage, a bit faster

- double nfst_phi_hut (nfst_plan *ths_plan, int k, int d)

  do some adjustments (N,n) then compute PHI_HUT

- double nfst_phi (nfst_plan *ths_plan, double x, int d)

  do some adjustments (N,n) then compute PHI

- int nfst_fftw_2N (int n)

  returns 2(n+1), fftw related issue

- int nfst_fftw_2N_rev (int n)

  returns 0.5n-1, fftw related issue

6.2.1 Detailed Description

Direct and fast computation of the discrete nonequispaced (co)sine transform.

6.2.2 Function Documentation

6.2.2.1 void nfct_init_1d (nfct_plan * *ths_plan*, int *N0*, int *M_total*)

Creates a 1-dimensional transform plan.

- ths_plan The plan for the transform
- N0 The bandwidth N
- M_total The number of nodes x

Author:
Steffen Klatt

Definition at line 996 of file nfct.c.

References nfct_init().

6.2.2.2 void nfct_init_2d (nfct_plan * *ths_plan*, int *N0*, int *N1*, int *M_total*)

Creates a 3-dimensional transform plan.

- ths_plan The plan for the transform
- N0 The bandwidth of dimension 1
- N1 The bandwidth of dimension 2
- M_total The number of nodes x

Author:
Steffen Klatt

Definition at line 1004 of file nfct.c.

References nfct_init().

6.2.2.3 void nfct_init_3d (nfct_plan * *ths_plan*, int *N0*, int *N1*, int *N2*, int *M_total*)

Creates a 3-dimensional transform plan.

- ths_plan The plan for the transform
- N0 The bandwidth of dimension 1
- N1 The bandwidth of dimension 2
- N2 The bandwidth of dimension 3
- M_total The number of nodes x

Author:
Steffen Klatt

Definition at line 1013 of file nfct.c.

References nfct_init().

6.2.2.4 void nfct_init (nfct_plan * *ths_plan*, int *d*, int * *N*, int *M_total*)

Creates a d-dimensional transform plan.

- ths_plan The plan for the transform
- d the dimension
- N The bandwidths
- M_total The number of nodes x

Author:
Steffen Klatt

Definition at line 931 of file nfct.c.

References nfct_plan::d, nfct_plan::fftw_flags, nfct_plan::M_total, nfct_plan::N, nfct_plan::n, nfct_fftw_-2N(), nfct_plan::nfct_flags, and nfft_next_power_of_2().

Referenced by nfct_init_1d(), nfct_init_2d(), and nfct_init_3d().

6.2.2.5 void nfct_init_guru (nfct_plan * *ths*, int *d*, int * *N*, int *M_total*, int * *n*, int *m*, unsigned *nfct_flags*, unsigned *fftw_flags*)

Creates a d-dimensional transform plan.

< index over all dimensions

Definition at line 968 of file nfct.c.

References nfct_plan::d, nfct_plan::fftw_flags, nfct_plan::m, nfct_plan::M_total, nfct_plan::N, nfct_plan::n, and nfct_plan::nfct_flags.

6.2.2.6 void nfct_precompute_psi (nfct_plan * *ths*)

precomputes the values psi if the PRE_PSI is set the application program has to call this routine after setting the nodes this_plan->x

< index over all dimensions

< index over all nodes

< index 0<=lj<u+o+1

< depends on x_j

Definition at line 838 of file nfct.c.

References nfct_plan::d, nfct_plan::M_total, and nfct_plan::psi.

6.2.2.7 void nfct_trafo (nfct_plan * *ths*)

executes a NFCT (approximate,fast), computes for $j = 0, ..., M\_total - 1$ $f_j^C(x_j) = \sum_{k \in I_0^{N,d}} \hat{f}_k^C * cos(2\pi k x_j)$

use ths->my_fftw_r2r_plan

form $\hat{g}_k = \frac{\hat{f}_k}{c_k(\phi)}$ for $k \in I_N$

compute by d-variate discrete Fourier transform $g_l = \sum_{k \in I_N} \hat{g}_k \mathrm{e}^{-2\pi \mathrm{i} \frac{kl}{n}}$ for $l \in I_n$

set $f_j = \sum_{l \in I_n, m(x_j)} g_l \psi \left(x_j - \frac{l}{n}\right)$ for $j = 0, \ldots, M-1$

Definition at line 711 of file nfct.c.

References PRE_FULL_PSI.

6.2.2.8 void ndct_trafo (nfct_plan * *ths_plan*)

executes a NDCT (exact,slow), computes for $j = 0, ..., M\_total - 1$ $f_j^C(x_j) = \sum_{k \in I_0^{N,d}} \hat{f}_k^C * cos(2\pi k x_j)$

- ths_plan The plan for the transform

Author:
Steffen Klatt

Definition at line 256 of file nfct.c.

6.2.2.9 void nfct_adjoint (nfct_plan * *ths*)

executes a transposed NFCT (approximate,fast), computes for $k \in I_0^{N,d}$ $h^C(k) = \sum_{j \in I_0^{(M\_total,1)}} f_j^C * cos(2\pi k x_j)$

use ths->my_fftw_plan

set $g_l = \sum_{j=0}^{M-1} f_j \psi\left(x_j - \frac{l}{n}\right)$ for $l \in I_n, m(x_j)$

compute by d-variate discrete cosine transform $\hat{g}_k = \sum_{l \in I_n} g_l \mathrm{e}^{-2\pi \mathrm{i} \frac{kl}{n}}$ for $k \in I_N$

form $\hat{f}_k = \frac{\hat{g}_k}{c_k(\phi)}$ for $k \in I_N$

Definition at line 765 of file nfct.c.

References nfct_plan::g, nfct_plan::g1, nfct_plan::g2, nfct_plan::g_hat, nfct_plan::my_fftw_r2r_plan, nfct_plan::nfct_flags, PRE_FULL_PSI, nfct_plan::psi_index_f, and nfct_plan::psi_index_g.

6.2.2.10 void ndct_adjoint (nfct_plan * *ths_plan*)

executes a direct transposed NDCT (exact,slow), computes for $k \in I_0^{N,d}$ $h^C(k) = \sum_{j \in I_0^{(M\_total,1)}} f_j^C * cos(2\pi k x_j)$

- ths_plan The plan for the transform

Author:
Steffen Klatt

Definition at line 257 of file nfct.c.

References nfct_fftw_2N().

6.2.2.11 void nfct_finalize (nfct_plan * *ths*)

Destroys a plan.

index over dimensions

Definition at line 1027 of file nfct.c.

References nfct_plan::c_phi_inv, nfct_plan::d, nfct_plan::f, nfct_plan::f_hat, FFT_OUT_OF_PLACE, FFTW_INIT, nfct_plan::g1, nfct_plan::g2, MALLOC_F, MALLOC_F_HAT, MALLOC_X, nfct_plan::my_fftw_r2r_plan, nfct_plan::N, nfct_plan::n, nfct_plan::nfct_flags, PRE_PHI_HUT, PRE_PSI, nfct_plan::psi, nfct_plan::sigma, and nfct_plan::x.

6.2.2.12 double nfct_phi_hut (nfct_plan * *ths_plan*, int *k*, int *d*)

do some adjustments (N,n) then compute PHI_HUT

- ths_plan the plan for the transform
- k index of c_phi
- d dimension

Author:
Steffen Klatt

Definition at line 63 of file nfct.c.

6.2.2.13 double nfct_phi (nfct_plan * *ths_plan*, double *x*, int *d*)

do some adjustments (N,n) then compute PHI

- ths_plan the plan for the transform
- x node x
- d dimension

Author:
Steffen Klatt

Definition at line 72 of file nfct.c.

6.2.2.14 int nfct_fftw_2N (int *n*)

returns 2(n-1), fftw related issue

- n i.e. length of dct-1

Author:
Steffen Klatt

Definition at line 81 of file nfct.c.

Referenced by ndct_adjoint(), and nfct_init().

6.2.2.15 int nfct_fftw_2N_rev (int *n*)

returns 0.5n+1, fftw related issue

- n i.e. length of dct-1

Author:
Steffen Klatt

Definition at line 86 of file nfct.c.

6.2.2.16 void nfst_init_1d (nfst_plan * *ths_plan*, int *N0*, int *M_total*)

Creates a 1-dimensional transform plan.

- ths_plan The plan for the transform
- N0 The bandwidth N
- M_total The number of nodes x

Author:
Steffen Klatt

Definition at line 979 of file nfst.c.

References nfst_init().

6.2.2.17 void nfst_init_2d (nfst_plan * *ths_plan*, int *N0*, int *N1*, int *M_total*)

Creates a 3-dimensional transform plan.

- ths_plan The plan for the transform
- N0 The bandwidth of dimension 1
- N1 The bandwidth of dimension 2
- M_total The number of nodes x

Author:
Steffen Klatt

Definition at line 987 of file nfst.c.

References nfst_init().

6.2.2.18 void nfst_init_3d (nfst_plan * *ths_plan*, int *N0*, int *N1*, int *N2*, int *M_total*)

Creates a 3-dimensional transform plan.

- ths_plan The plan for the transform
- N0 The bandwidth of dimension 1
- N1 The bandwidth of dimension 2
- N2 The bandwidth of dimension 3
- M_total The number of nodes x

Author:
Steffen Klatt

Definition at line 996 of file nfst.c.

References nfst_init().

6.2.2.19 void nfst_init (nfst_plan * *ths*, int *d*, int * *N*, int *M_total*)

Creates a d-dimensional transform plan.

$<$ index over all dimensions

Definition at line 913 of file nfst.c.

References nfst_plan::d, nfst_plan::fftw_flags, nfst_plan::M_total, nfst_plan::N, nfst_plan::n, nfft_next_power_of_2(), and nfst_plan::nfst_flags.

Referenced by nfst_init_1d(), nfst_init_2d(), and nfst_init_3d().

6.2.2.20 void nfst_init_m (nfst_plan * *ths_plan*, int *d*, int * *N*, int *M_total*, int *m*)

Creates a d-dimensional transform plan with pcific m.

(just for convenience)

- ths_plan The plan for the transform
- d the dimension
- N The bandwidths
- M_total The number of nodes x
- m cut-off parameter

Author:
Steffen Klatt

Definition at line 940 of file nfst.c.

References nfft_next_power_of_2(), nfst_fftw_2N(), and nfst_init_guru().

6.2.2.21 void nfst_init_guru (nfst_plan * *ths*, int *d*, int * *N*, int *M_total*, int * *n*, int *m*, unsigned *nfst_flags*, unsigned *fftw_flags*)

Creates a d-dimensional transform plan.

< index over all dimensions

Definition at line 951 of file nfst.c.

References nfst_plan::d, nfst_plan::fftw_flags, nfst_plan::m, nfst_plan::M_total, nfst_plan::N, nfst_plan::n, and nfst_plan::nfst_flags.

Referenced by nfst_init_m().

6.2.2.22 void nfst_precompute_psi (nfst_plan * *ths*)

precomputes the values psi if the PRE_PSI is set the application program has to call this routine after setting the nodes this_plan->x

< index over all dimensions

< index over all nodes

< index 0<=lj<u+o+1

< depends on x_j

Definition at line 743 of file nfst.c.

References nfst_plan::d, nfst_plan::M_total, nfst_plan::nfst_flags, nfst_full_psi(), nfst_plan::nfst_full_psi_eps, PRE_FULL_PSI, and nfst_plan::psi.

6.2.2.23 void nfst_trafo (nfst_plan * *ths*)

executes a NFST (approximate,fast), computes for $j = 0, ..., M\_total - 1$ $f_j^S(x_j) = \sum_{k \in I_1^{N,d}} \hat{f}_k^S * sin(2\pi k x_j)$

use ths->my_fftw_r2r_plan

form $\hat{g}_k = \frac{\hat{f}_k}{c_k(\phi)}$ for $k \in I_N$

compute by d-variate discrete Fourier transform $g_l = \sum_{k \in I_N} \hat{g}_k \mathrm{e}^{-2\pi \mathrm{i} \frac{kl}{n}}$ for $l \in I_n$

set $f_j = \sum_{l \in I_n, m(x_j)} g_l \psi\left(x_j - \frac{l}{n}\right)$ for $j = 0, \ldots, M-1$

Definition at line 630 of file nfst.c.

6.2.2.24 void ndst_trafo (nfst_plan * *ths_plan*)

executes a NDST (exact,slow), computes for $j = 0, ..., M\_total - 1$ $f_j^S(x_j) = \sum_{k \in I_1^{N,d}} \hat{f}_k^S * sin(2\pi k x_j)$

- ths_plan The plan for the transform

Author:
Steffen Klatt

Definition at line 254 of file nfst.c.

6.2.2.25 void nfst_adjoint (nfst_plan * *ths*)

executes a transposed NFST (approximate,fast), computes for $k \in I_1^{N,d}$ $h^S(k) = \sum_{j \in I_0^{M\_total,1}} f_j^S * cos(2\pi k x_j)$

use ths->my_fftw_plan

set $g_l = \sum_{j=0}^{M-1} f_j \psi\left(x_j - \frac{l}{n}\right)$ for $l \in I_n, m(x_j)$

compute by d-variate discrete cosine transform $\hat{g}_k = \sum_{l \in I_n} g_l \mathrm{e}^{-2\pi \mathrm{i} \frac{kl}{n}}$ for $k \in I_N$

form $\hat{f}_k = \frac{\hat{g}_k}{c_k(\phi)}$ for $k \in I_N$

Definition at line 675 of file nfst.c.

References nfst_plan::g, nfst_plan::g1, nfst_plan::g2, nfst_plan::g_hat, and nfst_plan::my_fftw_r2r_plan.

6.2.2.26 void ndst_adjoint (nfst_plan * *ths_plan*)

executes a direct transposed NDST (exact,slow), computes for $k \in I_1^{N,d}$ $h^S(k) = \sum_{j \in I_0^{M\_total,1}} f_j^S * cos(2\pi k x_j)$

- ths_plan The plan for the transform

Author:
Steffen Klatt

Definition at line 255 of file nfst.c.

References nfst_fftw_2N().

6.2.2.27 void nfst_finalize (nfst_plan * *ths_plan*)

Destroys a plan.

- ths_plan The plan for the transform

Author:
Steffen Klatt

Definition at line 1006 of file nfst.c.

References nfst_plan::c_phi_inv, nfst_plan::d, nfst_plan::f, nfst_plan::f_hat, FFT_OUT_OF_PLACE, FFTW_INIT, nfst_plan::g1, nfst_plan::g2, MALLOC_F, MALLOC_F_HAT, MALLOC_X, nfst_plan::my_fftw_r2r_plan, nfst_plan::N, nfst_plan::n, nfst_plan::nfst_flags, PRE_FULL_PSI, PRE_PHI_HUT, PRE_PSI, nfst_plan::psi, nfst_plan::psi_index_f, nfst_plan::psi_index_g, nfst_plan::sigma, and nfst_plan::x.

6.2.2.28 void nfst_full_psi (nfst_plan * *ths*, double *eps*)

more memory usage, a bit faster

< index over all dimensions

< index over all nodes

< plain index 0<=l_L<lprod

< multi index 0<=lj<u+o+1

< postfix plain index

< 'bandwidth' of matrix B

< depends on x_j

Definition at line 771 of file nfst.c.

References nfst_plan::d, nfst_plan::M_total, nfst_plan::nfst_flags, PRE_PSI, nfst_plan::psi, nfst_plan::psi_index_f, nfst_plan::psi_index_g, and nfst_plan::size_psi.

Referenced by nfst_precompute_psi().

6.2.2.29 double nfst_phi_hut (nfst_plan * *ths_plan*, int *k*, int *d*)

do some adjustments (N,n) then compute PHI_HUT

- ths_plan the plan for the transform
- k index of c_phi
- d dimension

Author:
Steffen Klatt

Definition at line 63 of file nfst.c.

6.2.2.30 double nfst_phi (nfst_plan * *ths_plan*, double *x*, int *d*)

do some adjustments (N,n) then compute PHI

- ths_plan the plan for the transform
- x node x
- d dimension

Author:
Steffen Klatt

Definition at line 72 of file nfst.c.

6.2.2.31 int nfst_fftw_2N (int *n*)

returns 2(n+1), fftw related issue

- n i.e. length of dst-1

Author:
Steffen Klatt

Definition at line 81 of file nfst.c.

Referenced by ndst_adjoint(), and nfst_init_m().

6.2.2.32 int nfst_fftw_2N_rev (int *n*)

returns 0.5n-1, fftw related issue

- n i.e. length of dct-1

Author:
Steffen Klatt

Definition at line 86 of file nfst.c.

6.3 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

  Structure for a transform plan.

Defines

- #define MALLOC_V (1U<< 11)

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

Functions

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

  Creates a transform plan.

- 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.

- void nndft_trafo (nnfft_plan *ths_plan)

  Executes a direct NNDFT, i.e.

- void nndft_adjoint (nnfft_plan *ths_plan)

  Executes a direct adjoint NNDFT, i.e.

- void nnfft_trafo (nnfft_plan *ths_plan)

  Executes a NNFFT, i.e.

- void nnfft_adjoint (nnfft_plan *ths_plan)

  Executes a adjoint NNFFT, i.e.

- void nnfft_precompute_lin_psi (nnfft_plan *ths_plan)

  Precomputation for a transform plan.

- void nnfft_precompute_psi (nnfft_plan *ths_plan)

  Precomputation for a transform plan.

- void nnfft_precompute_full_psi (nnfft_plan *ths_plan)

  Precomputation for a transform plan.

- void nnfft_precompute_phi_hut (nnfft_plan *ths_plan)

  Precomputation for a transform plan.

- void nnfft_finalize (nnfft_plan *ths_plan)

  Destroys a plan.

6.3.1 Detailed Description

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

6.3.2 Define Documentation

6.3.2.1 #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 958 of file nfft3.h.

Referenced by nnfft_finalize(), nnfft_init(), and reconstruct().

6.3.3 Function Documentation

6.3.3.1 void nnfft_init (nnfft_plan * *ths*, int *d*, int *N_total*, int *M_total*, int * *N*)

Creates a transform plan.

< index over all dimensions

Definition at line 573 of file nnfft.c.

References nnfft_plan::d, FFT_OUT_OF_PLACE, FFTW_INIT, nnfft_plan::m, nnfft_plan::M_total, MALLOC_F, MALLOC_F_HAT, MALLOC_V, MALLOC_X, nnfft_plan::N, nnfft_plan::N1, nnfft_plan::N_total, nnfft_plan::nnfft_flags, PRE_PHI_HUT, and PRE_PSI.

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

Creates a transform plan.

< index over all dimensions

Definition at line 538 of file nnfft.c.

References nnfft_plan::d, FFT_OUT_OF_PLACE, FFTW_INIT, nnfft_plan::m, nnfft_plan::M_total, MALLOC_F_HAT, nnfft_plan::N, nnfft_plan::N1, nnfft_plan::N_total, nnfft_plan::nnfft_flags, PRE_FULL_PSI, PRE_LIN_PSI, PRE_PHI_HUT, and PRE_PSI.

Referenced by reconstruct().

6.3.3.3 void nndft_trafo (nnfft_plan * *ths*)

Executes a direct NNDFT, i.e.

< index over all nodes (time)

< index for dimensions

< index over all nodes (fourier)

< dito

< actual Fourier coefficient

< actual sample

< sign times 2*pi*k*x

Definition at line 64 of file nnfft.c.

6.3.3.4 void nndft_adjoint (nnfft_plan * *ths*)

Executes a direct adjoint NNDFT, i.e.

< index over all nodes (time)

< index for dimensions

< index over all nodes (fourier)

< dito

< actual Fourier coefficient

< actual sample

< sign times 2*pi*k*x

Definition at line 65 of file nnfft.c.

6.3.3.5 void nnfft_trafo (nnfft_plan * *ths_plan*)

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 270 of file nnfft.c.

References nnfft_plan::d, nnfft_plan::direct_plan, nfft_plan::f, nnfft_plan::f, nnfft_plan::M_total, nfft_trafo(), nnfft_plan::sigma, and nnfft_plan::x.

6.3.3.6 void nnfft_adjoint (nnfft_plan * *ths_plan*)

Executes a adjoint NNFFT, i.e.

computes for $k = 0, ..., N_{total} - 1$

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

- ths_plan The plan

Author:
Tobias Knopp

Definition at line 294 of file nnfft.c.

References nnfft_plan::d, nnfft_plan::direct_plan, nfft_plan::f, nnfft_plan::f, nnfft_plan::M_total, nfft_adjoint(), nnfft_plan::sigma, and nnfft_plan::x.

6.3.3.7 void nnfft_precompute_lin_psi (nnfft_plan * *ths*)

Precomputation for a transform plan.

< index over all dimensions

< index over all nodes

< step size in [0,(m+1)/n]

Definition at line 342 of file nnfft.c.

References nnfft_plan::direct_plan, nnfft_plan::K, nnfft_plan::m, nnfft_plan::N1, nfft_precompute_lin_psi(), and nnfft_plan::psi.

Referenced by reconstruct().

6.3.3.8 void nnfft_precompute_psi (nnfft_plan * *ths*)

Precomputation for a transform plan.

< index over all dimensions

< index over all nodes

< index u<=l<=o

< index 0<=lj<u+o+1

< depends on v_j

Definition at line 360 of file nnfft.c.

References nnfft_plan::d, nnfft_plan::direct_plan, nnfft_plan::m, nnfft_plan::M_total, nnfft_plan::N1, nnfft_plan::N_total, nfft_precompute_psi(), nnfft_plan::psi, nnfft_plan::sigma, nnfft_plan::v, and nnfft_plan::x.

Referenced by nnfft_precompute_full_psi(), and reconstruct().

6.3.3.9 void nnfft_precompute_full_psi (nnfft_plan * *ths*)

Precomputation for a transform plan.

< index over all dimensions

< index over all nodes

< plain index 0<=l_L<lprod

< multi index u<=l<=o

< multi index 0<=lj<u+o+1

< postfix plain index

< 'bandwidth' of matrix B

< depends on x_j

Definition at line 399 of file nnfft.c.

References nnfft_plan::d, nnfft_plan::direct_plan, nnfft_plan::m, nnfft_plan::M_total, nnfft_plan::N_total, nfft_precompute_full_psi(), nnfft_precompute_psi(), nnfft_plan::psi, nnfft_plan::psi_index_f, nnfft_plan::psi_index_g, nnfft_plan::sigma, and nnfft_plan::x.

Referenced by reconstruct().

6.3.3.10 void nnfft_precompute_phi_hut (nnfft_plan * *ths*)

Precomputation for a transform plan.

< index over all frequencies

< index over all dimensions

Definition at line 320 of file nnfft.c.

References nnfft_plan::c_phi_inv, nnfft_plan::d, nnfft_plan::M_total, nnfft_plan::N, and nnfft_plan::x.

Referenced by reconstruct().

6.3.3.11 void nnfft_finalize (nnfft_plan * *ths_plan*)

Destroys a plan.

- ths_plan The plan

Author:
Tobias Knopp

Definition at line 608 of file nnfft.c.

References nnfft_plan::aN1, nnfft_plan::c_phi_inv, nnfft_plan::direct_plan, nnfft_plan::f, nnfft_plan::f_hat, MALLOC_F, MALLOC_F_HAT, MALLOC_V, MALLOC_X, nnfft_plan::N, nnfft_plan::N1, nfft_finalize(), nnfft_plan::nnfft_flags, PRE_FULL_PSI, PRE_LIN_PSI, PRE_PHI_HUT, PRE_PSI, nnfft_plan::psi, nnfft_plan::psi_index_f, nnfft_plan::psi_index_g, nnfft_plan::v, and nnfft_plan::x.

Referenced by reconstruct().

6.4 NSFFT - Nonequispaced sparse FFT

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

Data Structures

- struct nsfft_plan

  Structure for a NFFT plan.

Defines

- #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 nsdft_trafo and nsdft_adjoint.

Functions

- void nsdft_trafo (nsfft_plan *ths)

  Executes an NSDFT, computes for $j = 0, \ldots, M-1$ *:*

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

  .

- void nsdft_adjoint (nsfft_plan *ths)

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

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

  .

- void nsfft_trafo (nsfft_plan *ths)

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

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

  .

- void nsfft_adjoint (nsfft_plan *ths)

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

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

  .

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

  Copy coefficients from nsfft plan to a nfft plan.

- void nsfft_init_random_nodes_coeffs (nsfft_plan *ths)

Initialisation of pseudo random nodes and coefficients.

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

  Initialisation of a transform plan.

- void nsfft_finalize (nsfft_plan *ths)

  Destroys a transform plan.

6.4.1 Detailed Description

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

6.4.2 Define Documentation

6.4.2.1 #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 nsdft_trafo and nsdft_adjoint.

See also:
nsfft_init

Author:
Stefan Kunis

Definition at line 1168 of file nfft3.h.

6.4.3 Function Documentation

6.4.3.1 void nsdft_trafo (nsfft_plan * *ths*)

Executes an NSDFT, computes 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 772 of file nsfft.c.

References nsfft_plan::d.

6.4.3.2 void nsdft_adjoint (nsfft_plan * *ths*)

Executes an adjoint NSFFT, computes for $k \in H_N^d$:

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

.

- ths The pointer to a nsfft plan

Author:
Stefan Kunis

Definition at line 832 of file nsfft.c.

References nsfft_plan::d.

6.4.3.3 void nsfft_trafo (nsfft_plan * *ths*)

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

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

.

- ths The pointer to a nsfft plan

Author:
Markus Fenn, Stefan Kunis

Definition at line 1501 of file nsfft.c.

References nsfft_plan::d.

6.4.3.4 void nsfft_adjoint (nsfft_plan * *ths*)

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

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

.

- ths The pointer to a nsfft plan

Author:
Stefan Kunis

Definition at line 1509 of file nsfft.c.

References nsfft_plan::d.

6.4.3.5 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 565 of file nsfft.c.

References nsfft_plan::act_nfft_plan, nsfft_plan::d, nfft_plan::f_hat, nsfft_plan::f_hat, nsfft_plan::index_-sparse_to_full, nsfft_plan::M_total, nfft_plan::N_total, nsfft_plan::N_total, and nfft_plan::x.

6.4.3.6 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 683 of file nsfft.c.

References nsfft_plan::act_nfft_plan, nsfft_plan::d, nsfft_plan::f_hat, nsfft_plan::M_total, nsfft_plan::N_-total, nfft_vrand_shifted_unit_double(), nfft_vrand_unit_complex(), nfft_plan::x, nsfft_plan::x_021, nsfft_plan::x_102, nsfft_plan::x_120, nsfft_plan::x_201, and nsfft_plan::x_transposed.

6.4.3.7 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 1730 of file nsfft.c.

6.4.3.8 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 1831 of file nsfft.c.

References nsfft_plan::d.

6.5 MRI - Transforms in magnetic resonance imaging

Data Structures

- struct mri_inh_2d1d_plan

  The structure for the transform plan.

- struct mri_inh_3d_plan

  The structure for the transform plan.

Functions

- void mri_inh_2d1d_trafo (mri_inh_2d1d_plan *ths)

  Executes a mri transformation considering the field inhomogeneity with the 2d1d method, i.e.

- void mri_inh_2d1d_adjoint (mri_inh_2d1d_plan *ths)

  Executes an adjoint mri transformation considering the field inhomogeneity with the 2d1d method, i.e.

- 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.

- void mri_inh_2d1d_finalize (mri_inh_2d1d_plan *ths)

  Destroys a plan.

- void mri_inh_3d_trafo (mri_inh_3d_plan *ths)

  Executes a mri transformation considering the field inhomogeneity with the 3d method, i.e.

- void mri_inh_3d_adjoint (mri_inh_3d_plan *ths)

  Executes an adjoint mri transformation considering the field inhomogeneity with the 3d method, i.e.

- void mri_inh_3d_init_guru (mri_inh_3d_plan *ths, int *N, int M, int *n, int m, double sigma, unsigned nfft_flags, unsigned fftw_flags)
- void mri_inh_3d_finalize (mri_inh_3d_plan *ths)

  Destroys a plan.

6.5.1 Function Documentation

6.5.1.1 void 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 40 of file mri.c.

References nfft_plan::f, mri_inh_2d1d_plan::f, nfft_plan::f_hat, mri_inh_2d1d_plan::f_hat, nfft_plan::m, mri_inh_2d1d_plan::M_total, window_funct_plan_::n, mri_inh_2d1d_plan::N3, mri_inh_2d1d_plan::N_total, nfft_trafo(), PI, mri_inh_2d1d_plan::plan, mri_inh_2d1d_plan::sigma3, mri_inh_2d1d_plan::t, and mri_inh_2d1d_plan::w.

Referenced by construct().

6.5.1.2 void mri_inh_2d1d_adjoint (mri_inh_2d1d_plan * *ths*)

Executes an adjoint mri transformation considering the field inhomogeneity with the 2d1d 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 82 of file mri.c.

References nfft_plan::f, mri_inh_2d1d_plan::f, nfft_plan::f_hat, mri_inh_2d1d_plan::f_hat, nfft_plan::m, mri_inh_2d1d_plan::M_total, window_funct_plan_::n, mri_inh_2d1d_plan::N3, mri_inh_2d1d_plan::N_total, nfft_adjoint(), PI, mri_inh_2d1d_plan::plan, mri_inh_2d1d_plan::sigma3, mri_inh_2d1d_plan::t, and mri_inh_2d1d_plan::w.

6.5.1.3 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 134 of file mri.c.

References mri_inh_2d1d_plan::f, nfft_plan::f, mri_inh_2d1d_plan::f_hat, nfft_plan::f_hat, mri_inh_2d1d_plan::M_total, nfft_plan::M_total, mri_inh_2d1d_plan::N3, mri_inh_2d1d_plan::N_total, nfft_plan::N_total, nfft_init_guru(), mri_inh_2d1d_plan::plan, mri_inh_2d1d_plan::sigma3, mri_inh_2d1d_plan::t, and mri_inh_2d1d_plan::w.

Referenced by construct(), and reconstruct().

6.5.1.4 void mri_inh_2d1d_finalize (mri_inh_2d1d_plan * *ths*)

Destroys a plan.

- ths_plan The plan

Author:
Tobias Knopp

Definition at line 149 of file mri.c.

References nfft_plan::f, mri_inh_2d1d_plan::f, nfft_plan::f_hat, mri_inh_2d1d_plan::f_hat, nfft_finalize(), mri_inh_2d1d_plan::plan, mri_inh_2d1d_plan::t, and mri_inh_2d1d_plan::w.

Referenced by construct(), and reconstruct().

6.5.1.5 void 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 164 of file mri.c.

References nfft_plan::f, mri_inh_3d_plan::f, nfft_plan::f_hat, mri_inh_3d_plan::f_hat, nfft_plan::m, window_funct_plan_::m, mri_inh_3d_plan::M_total, window_funct_plan_::n, mri_inh_3d_plan::N3, mri_inh_3d_plan::N_total, nfft_trafo(), mri_inh_3d_plan::plan, mri_inh_3d_plan::sigma3, mri_inh_3d_plan::w, and nfft_plan::x.

Referenced by construct().

6.5.1.6 void 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 196 of file mri.c.

References nfft_plan::f, mri_inh_3d_plan::f, mri_inh_3d_plan::f_hat, nfft_plan::f_hat, nfft_plan::m, window_funct_plan_::m, mri_inh_3d_plan::M_total, window_funct_plan_::n, mri_inh_3d_plan::N3, mri_inh_3d_plan::N_total, nfft_adjoint(), mri_inh_3d_plan::plan, mri_inh_3d_plan::sigma3, mri_inh_3d_plan::w, and nfft_plan::x.

6.5.1.7 void mri_inh_3d_finalize (mri_inh_3d_plan * *ths*)

Destroys a plan.

- ths_plan The plan

Author:
Tobias Knopp

Definition at line 238 of file mri.c.

References mri_inh_3d_plan::f_hat, nfft_finalize(), mri_inh_3d_plan::plan, and mri_inh_3d_plan::w.

Referenced by construct(), and reconstruct().

6.6 NFSFT - Nonequispaced fast spherical Fourier transform

This module implements nonuniform fast spherical Fourier transforms.

Data Structures

- struct nfsft_plan

  Structure for a NFSFT transform plan.

- struct nfsft_wisdom

  Wisdom structure.

Defines

- #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

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

  .

- #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.

- #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.

- #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.*

- #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.*

- #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.*

- #define NFSFT_PRESERVE_F_HAT (1U << 7)

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

- #define NFSFT_PRESERVE_X (1U << 8)

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

- #define NFSFT_PRESERVE_F (1U << 9)

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

- #define NFSFT_DESTROY_F_HAT (1U << 10)

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

- #define NFSFT_DESTROY_X (1U << 11)

  If this flag is set, it is explicitely allowed that during an execution of ndsft_trafo, nfsft_trafo or ndsft_adjoint, nfsft_adjoint the content of `x` *may be changed.*

- #define NFSFT_DESTROY_F (1U << 12)

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

- #define NFSFT_NO_DIRECT_ALGORITHM (1U << 13)

  If this flag is set, the transforms ndsft_trafo and ndsft_adjoint do not work.

- #define NFSFT_NO_FAST_ALGORITHM (1U << 14)

  If this flag is set, the transforms nfsft_trafo and nfsft_adjoint do not work.

- #define NFSFT_ZERO_F_HAT (1U << 16)

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

- #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

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

  .

- #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.

- #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 nfsft_init (nfsft_plan *plan, int N, int M)

  Creates a transform plan.

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

  Creates a transform plan.

- void nfsft_init_guru (nfsft_plan *plan, int N, int M, unsigned int nfsft_flags, int nfft_flags, int nfft_cutoff)

  Creates a transform plan.

- 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 $N \in \mathbb{N}_2$.

- void nfsft_forget ()

  Forgets all precomputed data.

- void ndsft_trafo (nfsft_plan *plan)

  Executes a direct NDSFT, i.e.

- void ndsft_adjoint (nfsft_plan *plan)

  Executes a direct adjoint NDSFT, i.e.

- void nfsft_trafo (nfsft_plan *plan)

  Executes a NFSFT, i.e.

- void nfsft_adjoint (nfsft_plan *plan)

  Executes an adjoint NFSFT, i.e.

- void nfsft_finalize (nfsft_plan *plan)

  Destroys a plan.

- void nfsft_precompute_x (nfsft_plan *plan)
- double alpha_al (int k, int n)

  Computes three-term recurrence coefficients α_k^n of associated Legendre functions.

- double beta_al (int k, int n)

  Computes three-term recurrence coefficients β_k^n of associated Legendre functions.

- double gamma_al (int k, int n)

  Computes three-term recurrence coefficients γ_k^n of associated Legendre functions.

- void alpha_al_row (double *alpha, int N, int n)
- void beta_al_row (double *beta, int N, int n)
- void gamma_al_row (double *gamma, int N, int n)

- void alpha_al_all (double *alpha, 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 (double *beta, 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 (double *gamma, 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 (double *x, double *y, int size, int k, double *alpha, double *beta, double *gamma)

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

- int eval_al_thresh (double *x, double *y, int size, int k, double *alpha, double *beta, double *gamma, double threshold)

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

- 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

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

  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 $n = -M, \ldots, M$.

- void c2e_transposed (nfsft_plan *plan)

  Transposed version of the function c2e.

Variables

- static struct nfsft_wisdom wisdom = {false,0U}

  The global wisdom structure for precomputed data.

6.6.1 Detailed Description

This module implements nonuniform fast spherical Fourier transforms.

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

6.6.2 Preliminaries

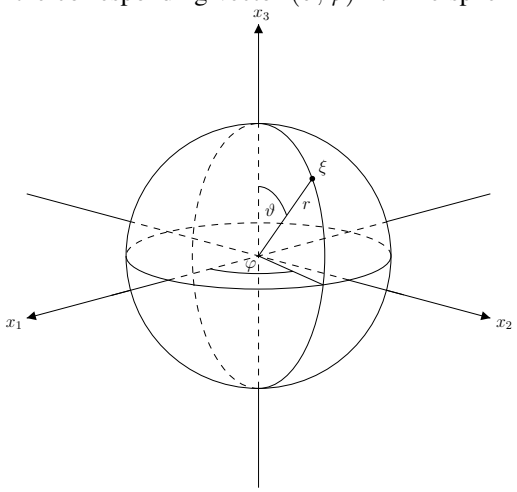
This section summarises basic definitions and properties related to spherical Fourier transforms.

6.6.2.1 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 $\mathbf{x} := (x_1, x_2) \in [-\frac{1}{2}, \frac{1}{2}) \times [0, \frac{1}{2}]$.

6.6.2.2 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*

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

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

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

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

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

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 $\mathrm{L}^2([-1, 1])$ norm

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

6.6.2.3 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

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

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 $\mathrm{L}^2([-1,1])$ with

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

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 $\mathrm{L}^2([-1,1])$ norm

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

6.6.2.4 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.

6.6.3 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.

6.6.3.1 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 $k = 0,\dots,N,\ n = -k,\dots,k,\ N \in \mathbb{N}_0$,
arbitrary nodes $\mathbf{x}(m) \in [-\frac{1}{2},\frac{1}{2}] \times [0,\frac{1}{2}]$ for $m = 0,\dots,M-1, M \in \mathbb{N}$.

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

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

6.6.3.2 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 $m = 0,\dots,M-1, M \in \mathbb{N}$,
arbitrary nodes $\mathbf{x}(m) \in [-\frac{1}{2},\frac{1}{2}] \times [0,\frac{1}{2}]$ for $m = 0,\dots,M-1, N \in \mathbb{N}_0$.

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

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

6.6.3.3 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 $i = 0,1,\dots,(2N+2)^2-1$. 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 $0 \le k \le M$, $-k \le n \le k$ 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 $m = 0,\dots,M-1$ 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 $m = 0,\dots,M-1$ such that `f[`$2m$`]` = x_1 and `f[`$2m+1$`]` = x_2

6.6.3.4 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 ndsft_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 ndsft_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.

6.6.4 Define Documentation

6.6.4.1 #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 1718 of file nfft3.h.

Referenced by main(), ndsft_adjoint(), ndsft_trafo(), nfsft_adjoint(), and nfsft_trafo().

6.6.4.2 #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 1730 of file nfft3.h.

Referenced by main(), nfsft_adjoint(), and nfsft_trafo().

6.6.4.3 #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

Warning:
This feature is not implemented yet!

Definition at line 1743 of file nfft3.h.

Referenced by main(), nfsft_adjoint(), and nfsft_trafo().

6.6.4.4 #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 1758 of file nfft3.h.

Referenced by nfsft_finalize(), nfsft_init(), and nfsft_init_guru().

6.6.4.5 #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 1773 of file nfft3.h.

Referenced by nfsft_finalize(), nfsft_init(), and nfsft_init_guru().

6.6.4.6 #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 1788 of file nfft3.h.

Referenced by nfsft_finalize(), nfsft_init(), and nfsft_init_guru().

6.6.4.7 #define NFSFT_PRESERVE_F_HAT (1U << 7)

If this flag is set, it is guaranteed that during an execution of ndsft_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 1800 of file nfft3.h.

Referenced by ndsft_trafo(), nfsft_finalize(), nfsft_init_guru(), and nfsft_trafo().

6.6.4.8 #define NFSFT_PRESERVE_X (1U << 8)

If this flag is set, it is guaranteed that during an execution of ndsft_trafo, nfsft_trafo or ndsft_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 1813 of file nfft3.h.

6.6.4.9 #define NFSFT_PRESERVE_F (1U << 9)

If this flag is set, it is guaranteed that during an execution of ndsft_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 1825 of file nfft3.h.

6.6.4.10 #define NFSFT_DESTROY_F_HAT (1U << 10)

If this flag is set, it is explicitely allowed that during an execution of ndsft_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 1836 of file nfft3.h.

6.6.4.11 #define NFSFT_DESTROY_X (1U << 11)

If this flag is set, it is explicitely allowed that during an execution of ndsft_trafo, nfsft_trafo or ndsft_-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 1848 of file nfft3.h.

6.6.4.12 #define NFSFT_DESTROY_F (1U << 12)

If this flag is set, it is explicitely allowed that during an execution of ndsft_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 1859 of file nfft3.h.

6.6.4.13 #define NFSFT_NO_DIRECT_ALGORITHM (1U << 13)

If this flag is set, the transforms ndsft_trafo and ndsft_adjoint do not work.

Setting this flag saves some memory for precomputed data.

See also:
nfsft_precompute
ndsft_trafo
ndsft_adjoint

Author:
Jens Keiner

Definition at line 1872 of file nfft3.h.

Referenced by ndsft_adjoint(), ndsft_trafo(), nfsft_forget(), and nfsft_precompute().

6.6.4.14 #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 1883 of file nfft3.h.

Referenced by main(), nfsft_adjoint(), nfsft_forget(), nfsft_init_guru(), nfsft_precompute(), and nfsft_-trafo().

6.6.4.15 #define NFSFT_ZERO_F_HAT (1U << 16)

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

Author:
Jens Keiner

Definition at line 1892 of file nfft3.h.

Referenced by ndsft_adjoint(), and nfsft_adjoint().

6.6.4.16 #define NFSFT_DEFAULT_NFFT_CUTOFF 6

The default NFFT cutoff parameter.

Author:
Jens Keiner

Definition at line 34 of file nfsft.c.

Referenced by nfsft_init_advanced().

6.6.4.17 #define NFSFT_DEFAULT_THRESHOLD 1000

The default threshold for the FPT.

Author:
Jens Keiner

Definition at line 41 of file nfsft.c.

6.6.4.18 #define NFSFT_BREAK_EVEN 5

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

Author:
Jens Keiner

Definition at line 48 of file nfsft.c.

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

6.6.5 Function Documentation

6.6.5.1 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 225 of file nfsft.c.

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

6.6.5.2 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 232 of file nfsft.c.

References FFT_OUT_OF_PLACE, FFTW_INIT, NFSFT_DEFAULT_NFFT_CUTOFF, nfsft_init_guru(), PRE_PHI_HUT, and PRE_PSI.

Referenced by nfsft_init().

6.6.5.3 void nfsft_init_guru (nfsft_plan * *plan*, int *N*, int *M*, unsigned int *flags*, int *nfft_flags*, int *nfft_cutoff*)

Creates a transform plan.

Definition at line 240 of file nfsft.c.

References nfft_plan::f, nfsft_plan::f, nfft_plan::f_hat, nfsft_plan::f_hat, nfsft_plan::f_hat_intern, nfsft_plan::flags, nfsft_plan::M_total, nfsft_plan::N, nfsft_plan::N_total, nfft_init_guru(), NFSFT_MALLOC_F, NFSFT_MALLOC_F_HAT, NFSFT_MALLOC_X, NFSFT_NO_FAST_ALGORITHM, NFSFT_PRESERVE_F_HAT, nfsft_plan::plan_nfft, nfft_plan::x, and nfsft_plan::x.

Referenced by main(), and nfsft_init_advanced().

6.6.5.4 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 $N \in \mathbb{N}_2$.

Definition at line 326 of file nfsft.c.

References nfsft_wisdom::alpha, alpha_al_all(), alpha_al_row(), nfsft_wisdom::beta, beta_al_all(), beta_al_row(), nfsft_wisdom::flags, FPT_AL_SYMMETRY, fpt_init(), FPT_PERSISTENT_DATA, fpt_precompute(), nfsft_wisdom::gamma, gamma_al_all(), gamma_al_row(), nfsft_wisdom::initialized, nfsft_wisdom::N_MAX, nfft_next_power_of_2_exp(), NFSFT_BREAK_EVEN, NFSFT_NO_DIRECT_ALGORITHM, NFSFT_NO_FAST_ALGORITHM, ROW, nfsft_wisdom::set, and nfsft_wisdom::T_MAX.

Referenced by main().

6.6.5.5 void nfsft_forget ()

Forgets all precomputed data.

Author:
Jens Keiner

Definition at line 428 of file nfsft.c.

References nfsft_wisdom::alpha, nfsft_wisdom::beta, nfsft_wisdom::flags, fpt_finalize(), nfsft_wisdom::gamma, nfsft_wisdom::initialized, nfsft_wisdom::N_MAX, NFSFT_BREAK_EVEN, NFSFT_NO_DIRECT_ALGORITHM, NFSFT_NO_FAST_ALGORITHM, and nfsft_wisdom::set.

Referenced by main().

6.6.5.6 void ndsft_trafo (nfsft_plan * *plan*)

Executes a direct NDSFT, 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 501 of file nfsft.c.

References nfsft_wisdom::alpha, nfsft_plan::f, nfsft_plan::f_hat, nfsft_plan::f_hat_intern, nfsft_plan::flags, nfsft_wisdom::flags, nfsft_wisdom::gamma, nfsft_plan::N, nfsft_plan::N_total, NFSFT_INDEX, NFSFT_NO_DIRECT_ALGORITHM, NFSFT_NORMALIZED, NFSFT_PRESERVE_F_HAT, PI, ROW, ROWK, and nfsft_plan::x.

Referenced by main(), and nfsft_trafo().

6.6.5.7 void ndsft_adjoint (nfsft_plan * *plan*)

Executes a direct adjoint NDSFT, 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 633 of file nfsft.c.

References nfsft_wisdom::alpha, nfsft_plan::f, nfsft_plan::f_hat, nfsft_plan::flags, nfsft_wisdom::flags, nfsft_wisdom::gamma, nfsft_plan::N, nfsft_plan::N_total, NFSFT_INDEX, NFSFT_NO_DIRECT_ALGORITHM, NFSFT_NORMALIZED, NFSFT_ZERO_F_HAT, PI, ROW, ROWK, and nfsft_plan::x.

Referenced by main(), and nfsft_adjoint().

6.6.5.8 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 745 of file nfsft.c.

References c2e(), dpt_trafo(), nfsft_plan::f, nfft_plan::f, nfft_plan::f_hat, nfsft_plan::f_hat, nfsft_plan::f_hat_intern, nfsft_plan::flags, nfsft_wisdom::flags, fpt_trafo(), nfsft_wisdom::initialized, nfsft_plan::N, nfsft_wisdom::N_MAX, nfsft_plan::N_total, ndft_trafo(), ndsft_trafo(), nfft_trafo(), NFSFT_BREAK_EVEN, NFSFT_INDEX, NFSFT_NO_FAST_ALGORITHM, NFSFT_NORMALIZED, NFSFT_PRESERVE_F_HAT, NFSFT_USE_DPT, NFSFT_USE_NDFT, PI, nfsft_plan::plan_nfft, nfsft_wisdom::set, nfsft_plan::x, and nfft_plan::x.

Referenced by main().

6.6.5.9 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 864 of file nfsft.c.

References c2e_transposed(), dpt_transposed(), nfsft_plan::f, nfft_plan::f, nfsft_plan::f_hat, nfft_plan::f_hat, nfsft_plan::flags, nfsft_wisdom::flags, fpt_transposed(), nfsft_wisdom::initialized, nfsft_plan::N, nfsft_wisdom::N_MAX, ndft_adjoint(), ndsft_adjoint(), nfft_adjoint(), NFSFT_BREAK_EVEN, NFSFT_INDEX, NFSFT_NO_FAST_ALGORITHM, NFSFT_NORMALIZED, NFSFT_USE_DPT, NFSFT_USE_NDFT, NFSFT_ZERO_F_HAT, PI, nfsft_plan::plan_nfft, nfsft_wisdom::set, nfsft_plan::x, and nfft_plan::x.

Referenced by main().

6.6.5.10 void nfsft_finalize (nfsft_plan * *plan*)

Destroys a plan.

- plan the plan to be destroyed

Author:
Jens Keiner

Definition at line 467 of file nfsft.c.

References nfsft_plan::f, nfsft_plan::f_hat, nfsft_plan::f_hat_intern, nfsft_plan::flags, nfft_finalize(), NFSFT_MALLOC_F, NFSFT_MALLOC_F_HAT, NFSFT_MALLOC_X, NFSFT_PRESERVE_F_HAT, nfsft_plan::plan_nfft, and nfsft_plan::x.

Referenced by main().

6.6.5.11 double alpha_al (int *k*, int *n*) [inline]

Computes three-term recurrence coefficients α_k^n of associated Legendre functions.

- k The index k
- n The index n

Definition at line 9 of file legendre.c.

Referenced by alpha_al_all(), and alpha_al_row().

6.6.5.12 double beta_al (int *k*, int *n*) [inline]

Computes three-term recurrence coefficients β_k^n of associated Legendre functions.

- k The index k
- n The index n

Definition at line 36 of file legendre.c.

Referenced by beta_al_all(), and beta_al_row().

6.6.5.13 double gamma_al (int *k*, int *n*) [inline]

Computes three-term recurrence coefficients γ_k^n of associated Legendre functions.

- k The index k
- n The index n

Definition at line 48 of file legendre.c.

Referenced by gamma_al_all(), and gamma_al_row().

6.6.5.14 void alpha_al_all (double * *alpha*, 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 106 of file legendre.c.

References alpha_al().

Referenced by nfsft_precompute().

6.6.5.15 void beta_al_all (double * *beta*, 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 120 of file legendre.c.

References beta_al().

Referenced by nfsft_precompute().

6.6.5.16 void gamma_al_all (double * *gamma*, 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 134 of file legendre.c.

References gamma_al().

Referenced by nfsft_precompute().

6.6.5.17 void eval_al (double * *x*, double * *y*, int *size*, int *k*, double * *alpha*, double * *beta*, double * *gamma*) [inline]

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 148 of file legendre.c.

6.6.5.18 int eval_al_thresh (double * *x*, double * *y*, int *size*, int *k*, double * *alpha*, double * *beta*, double * *gamma*, double *threshold*) [inline]

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 193 of file legendre.c.

6.6.5.19 void c2e (nfsft_plan * *plan*) [inline]

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

< The degree k

< The order k

< Stores temporary values

< Stores temporary values

< Auxilliary pointer

< Auxilliary pointer

< Lower loop bound

< Upper loop bound

< Lower loop bound for even terms

< Upper loop bound for even terms

Definition at line 80 of file nfsft.c.

References nfsft_plan::f_hat_intern, nfsft_plan::N, and NFSFT_INDEX.

Referenced by nfsft_trafo().

6.6.5.20 void c2e_transposed (nfsft_plan * *plan*) [inline]

Transposed version of the function c2e.

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

Remarks:
The transformation is computed in place.

Author:
Jens Keiner

< The degree k

< The order k

< Stores temporary values

< Stores temporary values

< Auxilliary pointer

< Auxilliary pointer

< Lower loop bound

< Upper loop bound

< Lower loop bound for even terms

< Upper loop bound for even terms

Definition at line 158 of file nfsft.c.

References nfsft_plan::f_hat, nfsft_plan::N, and NFSFT_INDEX.

Referenced by nfsft_adjoint().

6.6.6 Variable Documentation

6.6.6.1 struct nfsft_wisdom wisdom = {false,0U} [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 56 of file nfsft.c.

6.7 FPT - Fast polynomial transform

This module implements fast polynomial transforms.

Defines

- #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)

  TODO Don't use this flag!

Typedefs

- typedef fpt_set_s_ * fpt_set

  A set of precomputed data for a set of DPT transforms of equal maximum length.

Functions

- 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.

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

  Computes the data required for a single DPT transform.

- void dpt_trafo (fpt_set set, const int m, const double complex *x, double complex *y, const int k_end, const unsigned int flags)

  Computes a single DPT transform.

- void fpt_trafo (fpt_set set, const int m, const double complex *x, double complex *y, const int k_end, const unsigned int flags)

  Computes a single DPT transform.

- void dpt_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.

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

  Computes a single DPT transform.

- void fpt_finalize (fpt_set set)

6.7.1 Detailed Description

This module implements fast polynomial transforms.

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

6.7.2 Function Documentation

6.7.2.1 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.

Polynomial length

Cascade level

Index m

Allocate memory for auxilliary arrays.

Allocate memory for auxilliary arrays.

Initialize FFTW plans.

Definition at line 586 of file fpt.c.

References fpt_set_s_::dpt, fpt_set_s_::flags, FPT_NO_DIRECT_ALGORITHM, FPT_NO_FAST_ALGORITHM, fpt_set_s_::kinds, fpt_set_s_::kindsr, fpt_set_s_::lengths, fpt_set_s_::M, fpt_set_s_::N, PI, fpt_set_s_::plans_dct2, fpt_set_s_::plans_dct3, fpt_set_s_::result, fpt_data_::steps, fpt_set_s_::t, fpt_set_s_::temp, fpt_set_s_::vec3, fpt_set_s_::vec4, fpt_set_s_::work, fpt_set_s_::xc_slow, fpt_set_s_::xcvecs, and fpt_set_s_::z.

Referenced by nfsft_precompute().

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

Computes the data required for a single DPT transform.

< Cascade level

< Level index

< Length of polynomials for the next level in the cascade

< Degree of polynomials for the current level in the cascade

< First index l for current cascade level

< Last index l for current cascade level and current

< Length of polynomials for the next level in the cascade for stabilization

< Degree of polynomials for the current level in the cascade for stabilization

< Array containing function values of the (1,1)-component of U_k^n.

< Array containing function values of the (1,2)-component of U_k^n.

< Array containing function values of the (2,1)-component of U_k^n.

< Array containing function values of the (2,2)-component of U_k^n.

< Used to indicate that stabilization is neccessary.

Increase polynomial degree to next power of two.

Definition at line 698 of file fpt.c.

References fpt_step_::a11, fpt_step_::a12, fpt_step_::a21, fpt_step_::a22, fpt_data_::alpha, fpt_data_::alpha_0, fpt_data_::alphaN, fpt_data_::beta, fpt_data_::beta_0, fpt_data_::betaN, fpt_set_s_::dpt, eval_clenshaw(), eval_clenshaw_thresh(), FIRST_L, fpt_set_s_::flags, FPT_AL_SYMMETRY, FPT_NO_DIRECT_ALGORITHM, FPT_NO_FAST_ALGORITHM, FPT_NO_STABILIZATION, FPT_PERSISTENT_DATA, fpt_step_::gamma, fpt_data_::gamma, fpt_data_::gamma_m1, fpt_data_::gammaN, IS_SYMMETRIC, fpt_data_::k_start, K_START_TILDE, LAST_L, fpt_set_s_::N, fpt_step_::N_stab, N_TILDE, nfft_next_power_of_2(), nfft_next_power_of_2_exp(), fpt_step_::stable, fpt_data_::steps, fpt_set_s_::t, fpt_step_::t_stab, and fpt_set_s_::xcvecs.

Referenced by nfsft_precompute().

6.7.2.3 void dpt_trafo (fpt_set *set*, const int *m*, const 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 988 of file fpt.c.

References fpt_data_::alpha, fpt_data_::beta, fpt_set_s_::dpt, eval_sum_clenshaw(), fpt_set_s_::flags, FPT_FUNCTION_VALUES, FPT_NO_DIRECT_ALGORITHM, fpt_data_::gamma, fpt_data_::gamma_m1, fpt_data_::k_start, nfft_next_power_of_2_exp(), PI, fpt_set_s_::plans_dct2, fpt_set_s_::result, fpt_set_s_::temp, fpt_set_s_::work, fpt_set_s_::xc_slow, and fpt_set_s_::xcvecs.

Referenced by nfsft_trafo().

6.7.2.4 void fpt_trafo (fpt_set *set*, const int *m*, const double complex * *x*, double complex * *y*, const int *k_end*, const unsigned int *flags*)

Computes a single DPT transform.

Level index tau

Index of first block at current level

Index of last block at current level

Block index l

Length of polynomial coefficient arrays at next level

Polynomial array length for stabilization

Current matrix $U_{n,tau,l}$

Loop counter

Definition at line 1042 of file fpt.c.

References fpt_step_::a11, fpt_step_::a12, fpt_step_::a21, fpt_step_::a22, fpt_data_::alpha_0, fpt_data_::alphaN, fpt_data_::beta_0, fpt_data_::betaN, fpt_set_s_::dpt, FIRST_L, fpt_set_s_::flags, FPT_AL_SYMMETRY, fpt_do_step(), fpt_do_step_symmetric(), fpt_do_step_symmetric_l(), fpt_do_step_symmetric_u(), FPT_FUNCTION_VALUES, FPT_NO_FAST_ALGORITHM, fpt_step_::gamma, fpt_data_::gamma_m1, fpt_data_::gammaN, IS_SYMMETRIC, K_END_TILDE, fpt_data_::k_start, K_START_TILDE, LAST_L, fpt_step_::N_stab, nfft_next_power_of_2_exp(), fpt_set_s_::result, fpt_step_::stable, fpt_data_::steps, fpt_step_::t_stab, fpt_set_s_::vec3, fpt_set_s_::vec4, and fpt_set_s_::work.

Referenced by nfsft_trafo().

6.7.2.5 void dpt_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 1297 of file fpt.c.

References fpt_data_::alpha, fpt_data_::beta, fpt_set_s_::dpt, eval_sum_clenshaw_transposed(), fpt_set_s_::flags, FPT_FUNCTION_VALUES, FPT_NO_DIRECT_ALGORITHM, fpt_data_::gamma, fpt_data_::gamma_m1, fpt_data_::k_start, nfft_next_power_of_2_exp(), PI, fpt_set_s_::plans_dct3, fpt_set_s_::result, fpt_set_s_::temp, fpt_set_s_::work, fpt_set_s_::xc_slow, and fpt_set_s_::xcvecs.

Referenced by nfsft_adjoint().

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

Computes a single DPT transform.

Level index tau

Index of first block at current level

Index of last block at current level

Block index l

Length of polynomial coefficient arrays at next level

Polynomial array length for stabilization

Current matrix $U_{n,tau,l}$

Loop counter

Save copy of inpute data for stabilization steps.

Definition at line 1349 of file fpt.c.

References fpt_step_::a11, fpt_step_::a12, fpt_step_::a21, fpt_step_::a22, fpt_data_::alpha_0, fpt_data_::alphaN, fpt_data_::beta_0, fpt_data_::betaN, fpt_set_s_::dpt, FIRST_L, fpt_set_s_::flags, FPT_AL_SYMMETRY, fpt_do_step_transposed(), fpt_do_step_transposed_symmetric(), fpt_do_step_transposed_symmetric_l(), fpt_do_step_transposed_symmetric_u(), FPT_FUNCTION_VALUES, FPT_NO_FAST_ALGORITHM, fpt_step_::gamma, fpt_data_::gamma_m1, fpt_data_::gammaN, IS_SYMMETRIC, K_END_TILDE, fpt_data_::k_start, K_START_TILDE, LAST_L, fpt_step_::N_stab, nfft_next_power_of_2_exp(), fpt_set_s_::result, fpt_step_::stable, fpt_data_::steps, fpt_step_::t_stab, fpt_set_s_::vec3, fpt_set_s_::vec4, and fpt_set_s_::work.

Referenced by nfsft_adjoint().

6.8 Solver - Inverse transforms

Data Structures

- struct infft_plan

  Structure for an inverse transform plan.

- struct infct_plan

  Structure for an inverse transform plan.

- struct infst_plan

  Structure for an inverse transform plan.

- struct innfft_plan

  Structure for an inverse transform plan.

- struct imri_inh_2d1d_plan

  Structure for an inverse transform plan.

- struct imri_inh_3d_plan

  Structure for an inverse transform plan.

- struct infsft_plan

  Structure for an inverse transform plan.

Defines

- #define LANDWEBER (1U<< 0)

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

- #define STEEPEST_DESCENT (1U<< 1)

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

- #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.

- #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.

- #define NORMS_FOR_LANDWEBER (1U<< 4)

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

- #define PRECOMPUTE_WEIGHT (1U<< 5)

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

- #define PRECOMPUTE_DAMP (1U<< 6)

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

- #define MACRO_SOLVER_PLAN(MV, FLT, FLT_TYPE)

  Complete macro for mangling an inverse transform.

Functions

- void infft_init (infft_plan *ths, nfft_plan *mv)

  Simple initialisation.

- void infft_init_advanced (infft_plan *ths, nfft_plan *mv, unsigned infft_flags)

  Advanced initialisation.

- void infft_before_loop (infft_plan *ths)

  Setting up residuals before the actual iteration.

- void infft_loop_one_step (infft_plan *ths)

  Doing one step in the iteration.

- void infft_finalize (infft_plan *ths)

  Destroys the plan for the inverse transform.

- void infct_init (infct_plan *ths, nfct_plan *mv)

  Simple initialisation.

- void infct_init_advanced (infct_plan *ths, nfct_plan *mv, unsigned infct_flags)

  Advanced initialisation.

- void infct_before_loop (infct_plan *ths)

  Setting up residuals before the actual iteration.

- void infct_loop_one_step (infct_plan *ths)

  Doing one step in the iteration.

- void infct_finalize (infct_plan *ths)

  Destroys the plan for the inverse transform.

- void infst_init (infst_plan *ths, nfst_plan *mv)

  Simple initialisation.

- void infst_init_advanced (infst_plan *ths, nfst_plan *mv, unsigned infst_flags)

  Advanced initialisation.

- void infst_before_loop (infst_plan *ths)

  Setting up residuals before the actual iteration.

- void infst_loop_one_step (infst_plan *ths)

  Doing one step in the iteration.

- void infst_finalize (infst_plan *ths)

  Destroys the plan for the inverse transform.

- void innfft_init (innfft_plan *ths, nnfft_plan *mv)

  Simple initialisation.

- void innfft_init_advanced (innfft_plan *ths, nnfft_plan *mv, unsigned innfft_flags)

  Advanced initialisation.

- void innfft_before_loop (innfft_plan *ths)

  Setting up residuals before the actual iteration.

- void innfft_loop_one_step (innfft_plan *ths)

  Doing one step in the iteration.

- void innfft_finalize (innfft_plan *ths)

  Destroys the plan for the inverse transform.

- void imri_inh_2d1d_init (imri_inh_2d1d_plan *ths, mri_inh_2d1d_plan *mv)

  Simple initialisation.

- void imri_inh_2d1d_init_advanced (imri_inh_2d1d_plan *ths, mri_inh_2d1d_plan *mv, unsigned imri_inh_2d1d_flags)

  Advanced initialisation.

- void imri_inh_2d1d_before_loop (imri_inh_2d1d_plan *ths)

  Setting up residuals before the actual iteration.

- void imri_inh_2d1d_loop_one_step (imri_inh_2d1d_plan *ths)

  Doing one step in the iteration.

- void imri_inh_2d1d_finalize (imri_inh_2d1d_plan *ths)

  Destroys the plan for the inverse transform.

- void imri_inh_3d_init (imri_inh_3d_plan *ths, mri_inh_3d_plan *mv)

  Simple initialisation.

- void imri_inh_3d_init_advanced (imri_inh_3d_plan *ths, mri_inh_3d_plan *mv, unsigned imri_inh_3d_flags)

  Advanced initialisation.

- void imri_inh_3d_before_loop (imri_inh_3d_plan *ths)

  Setting up residuals before the actual iteration.

- void imri_inh_3d_loop_one_step (imri_inh_3d_plan *ths)

  Doing one step in the iteration.

- void imri_inh_3d_finalize (imri_inh_3d_plan *ths)

  Destroys the plan for the inverse transform.

- void infsft_init (infsft_plan *ths, nfsft_plan *mv)
  Simple initialisation.
- void infsft_init_advanced (infsft_plan *ths, nfsft_plan *mv, unsigned infsft_flags)
  Advanced initialisation.
- void infsft_before_loop (infsft_plan *ths)
  Setting up residuals before the actual iteration.
- void infsft_loop_one_step (infsft_plan *ths)
  Doing one step in the iteration.
- void infsft_finalize (infsft_plan *ths)
  Destroys the plan for the inverse transform.

6.8.1 Define Documentation

6.8.1.1 #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 2213 of file nfft3.h.

6.8.1.2 #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 2221 of file nfft3.h.

6.8.1.3 #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 2230 of file nfft3.h.

Referenced by inverse_linogram_fft(), inverse_mpolar_fft(), inverse_polar_fft(), Inverse_Radon_trafo(), and reconstruct().

6.8.1.4 #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 2239 of file nfft3.h.

Referenced by glacier().

6.8.1.5 #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 2247 of file nfft3.h.

6.8.1.6 #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 2255 of file nfft3.h.

Referenced by inverse_linogram_fft(), inverse_mpolar_fft(), inverse_polar_fft(), Inverse_Radon_trafo(), and reconstruct().

6.8.1.7 #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 2263 of file nfft3.h.

Referenced by glacier(), inverse_linogram_fft(), inverse_mpolar_fft(), inverse_polar_fft(), and reconstruct().

6.8.1.8 #define MACRO_SOLVER_PLAN(MV, FLT, FLT_TYPE)

Complete macro for mangling an inverse transform.

- MV Matrix vector multiplication type (eg nfft, nfct)

- FLT Float used as prefix for function names (double or complex)
- FLT_TYPE Float type (double or double complex)

Author:
Stefan Kunis

Definition at line 2274 of file nfft3.h.

6.9 Util - Auxilliary functions

This module implements frequently used utility functions.

Defines

- #define NFFT_SWAP_complex(x, y) {double complex* temp; temp=(x); (x)=(y); (y)=temp;}
  Swapping of two vectors.
- #define NFFT_SWAP_double(x, y) {double* temp; temp=(x); (x)=(y); (y)=temp;}
  Swapping of two vectors.
- #define PI 3.1415926535897932384
  Formerly known to be an irrational number.
- #define NFFT_MAX(a, b) ((a)>(b)? (a) : (b))
  Maximum of its two arguments.
- #define NFFT_MIN(a, b) ((a)<(b)? (a) : (b))
  Mimimum of its two arguments.

Functions

- double nfft_second ()
  Actual used CPU time in seconds; calls getrusage, limited accuracy.
- int nfft_total_used_memory ()
  Actual used memory in bytes; calls mallinfo if define HAVE_MALLOC_H.
- int nfft_ld (int m)
  Integer logarithm of 2.
- int nfft_int_2_pow (int a)
  Integer power of 2.
- int nfft_next_power_of_2 (int N)
  Computes $n \geq N$ such that $n = 2^j$, $j \in \mathbb{N}_0$.
- void nfft_next_power_of_2_exp (int N, int *N2, int *t)
  Computes ?
- double nfft_sinc (double x)
  Computes the sinus cardinalis $\frac{sin(x)}{x}$.
- double nfft_bspline_old (int k, double x, double *A)
  To test the new one.
- double nfft_bspline (int k, double x, double *scratch)

Computes the B-spline $M_{k,0}(x)$, scratch is used for de Boor's scheme.

- double nfft_i0 (double x)

  Modified Bessel function of order zero; adapted from Stephen Moshier's Cephes Math Library Release 2.8.

- int nfft_prod_int (int *vec, int d)

  Computes integer $\prod_{t=0}^{d-1} v_t$.

- int nfct_prod_int (int *vec, int d)

  Computes integer $\prod_{t=0}^{d-1} v_t$.

- int nfst_prod_minus_a_int (int *vec, int a, int d)

  Computes integer $\prod_{t=0}^{d-1} v_t - a$.

- int nfft_plain_loop (int *idx, int *N, int d)

  Computes $\sum_{t=0}^{d-1} i_t \prod_{t'=t+1}^{d-1} N_{t'}$.

- double nfft_prod_real (double *vec, int d)

  Computes double $\prod_{t=0}^{d-1} v_t$.

- double nfft_dot_complex (double complex *x, int n)

  Computes the inner/dot product $x^H x$.

- double nfft_dot_double (double *x, int n)

  Computes the inner/dot product $x^H x$.

- double nfft_dot_w_complex (double complex *x, double *w, int n)

  Computes the weighted inner/dot product $x^H (w \odot x)$.

- double nfft_dot_w_double (double *x, double *w, int n)

  Computes the weighted inner/dot product $x^H (w \odot x)$.

- double nfft_dot_w_w2_complex (double complex *x, double *w, double *w2, int n)

  Computes the weighted inner/dot product $x^H (w1 \odot w2 \odot w2 \odot x)$.

- double nfft_dot_w2_complex (double complex *x, double *w2, int n)

  Computes the weighted inner/dot product $x^H (w2 \odot w2 \odot x)$.

- void nfft_cp_complex (double complex *x, double complex *y, int n)

  Copies $x \leftarrow y$.

- void nfft_cp_double (double *x, double *y, int n)

  Copies $x \leftarrow y$.

- void nfft_cp_a_complex (double complex *x, double a, double complex *y, int n)

  Copies $x \leftarrow ay$.

- void nfft_cp_w_complex (double complex *x, double *w, double complex *y, int n)

  Copies $x \leftarrow w \odot y$.

- void nfft_cp_w_double (double *x, double *w, double *y, int n)
  Copies $x \leftarrow w \odot y$.
- void nfft_upd_axpy_complex (double complex *x, double a, double complex *y, int n)
  Updates $x \leftarrow ax + y$.
- void nfft_upd_axpy_double (double *x, double a, double *y, int n)
  Updates $x \leftarrow ax + y$.
- void nfft_upd_xpay_complex (double complex *x, double a, double complex *y, int n)
  Updates $x \leftarrow x + ay$.
- void nfft_upd_xpay_double (double *x, double a, double *y, int n)
  Updates $x \leftarrow x + ay$.
- void nfft_upd_axpby_complex (double complex *x, double a, double complex *y, double b, int n)
  Updates $x \leftarrow ax + by$.
- void nfft_upd_axpby_double (double *x, double a, double *y, double b, int n)
  Updates $x \leftarrow ax + by$.
- void nfft_upd_xpawy_complex (double complex *x, double a, double *w, double complex *y, int n)
  Updates $x \leftarrow x + aw \odot y$.
- void nfft_upd_xpawy_double (double *x, double a, double *w, double *y, int n)
  Updates $x \leftarrow x + aw \odot y$.
- void nfft_upd_axpwy_complex (double complex *x, double a, double *w, double complex *y, int n)
  Updates $x \leftarrow ax + w \odot y$.
- void nfft_upd_axpwy_double (double *x, double a, double *w, double *y, int n)
  Updates $x \leftarrow ax + w \odot y$.
- void nfft_fftshift_complex (double complex *x, int d, int *N)
  Swaps each half over N[d]/2.
- double nfft_error_l_infty_complex (double complex *x, double complex *y, int n)
  Computes $\frac{\|x-y\|_\infty}{\|x\|_\infty}$.
- double nfft_error_l_infty_double (double *x, double *y, int n)
  Computes $\frac{\|x-y\|_\infty}{\|x\|_\infty}$.
- double nfft_error_l_infty_1_complex (double complex *x, double complex *y, int n, double complex *z, int m)
  Computes $\frac{\|x-y\|_\infty}{\|z\|_1}$.
- double nfft_error_l_infty_1_double (double *x, double *y, int n, double *z, int m)
  Computes $\frac{\|x-y\|_\infty}{\|z\|_1}$.

- double nfft_error_l_2_complex (double complex *x, double complex *y, int n)

  Computes $\frac{\|x-y\|_2}{\|x\|_2}$.

- double nfft_error_l_2_double (double *x, double *y, int n)

  Computes $\frac{\|x-y\|_2}{\|x\|_2}$.

- void nfft_vpr_int (int *x, int n, char *text)

  Prints a vector of integer numbers.

- void nfft_vpr_double (double *x, int n, char *text)

  Prints a vector of doubles numbers.

- void nfft_vpr_complex (double complex *x, int n, char *text)

  Prints a vector of complex numbers.

- void nfft_vrand_unit_complex (double complex *x, int n)

  Inits a vector of random complex numbers in $[0,1] \times [0,1]$i .

- void nfft_vrand_shifted_unit_double (double *x, int n)

  Inits a vector of random double numbers in $[-1/2, 1/2]$.

- void nfft_voronoi_weights_1d (double *w, double *x, int M)

  Computes non periodic voronoi weights, assumes ordered nodes x_j.

- double nfft_modified_fejer (int N, int kk)

  Computes the damping factor for the modified Fejer kernel, ie $\frac{2}{N}\left(1-\frac{|2k+1|}{N}\right)$.

- double nfft_modified_jackson2 (int N, int kk)

  Computes the damping factor for the modified Jackson kernel.

- double nfft_modified_jackson4 (int N, int kk)

  Computes the damping factor for the modified generalised Jackson kernel.

- double nfft_modified_sobolev (double mu, int kk)

  Computes the damping factor for the modified Sobolev kernel.

- double nfft_modified_multiquadric (double mu, double c, int kk)

  Computes the damping factor for the modified multiquadric kernel.

6.9.1 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.

6.9.2 Function Documentation

6.9.2.1 double nfft_second ()

Actual used CPU time in seconds; calls getrusage, limited accuracy.

Calls getrusage, limited accuracy

Definition at line 25 of file util.c.

Referenced by comparison_fft(), fgt_test_measure_time(), inverse_linogram_fft(), inverse_mpolar_fft(), linogram_dft(), linogram_fft(), main(), mpolar_dft(), mpolar_fft(), ndft_time(), reconstruct(), taylor_time_accuracy(), and time_accuracy().

6.9.2.2 double nfft_bspline (int *k*, double *x*, double * *scratch*)

Computes the B-spline $M_{k,0}(x)$, scratch is used for de Boor's scheme.

< M_{k,0}(x)

$< x \in \mathrm{supp}(M_{0,r})$

< boundaries

< indices

< alpha of the de Boor scheme

Definition at line 237 of file util.c.

6.9.2.3 double nfft_i0 (double *x*)

Modified Bessel function of order zero; adapted from Stephen Moshier's Cephes Math Library Release 2.8.

Cephes Math Library Release 2.8: June, 2000 Copyright 1984, 1987, 2000 by Stephen L. Moshier

Definition at line 861 of file util.c.

6.9.2.4 double nfft_modified_fejer (int *N*, int *kk*)

Computes the damping factor for the modified Fejer kernel, ie $\frac{2}{N}\left(1-\frac{|2k+1|}{N}\right)$.

/f${2}{N}(1-{|2k+1|}{N})/f$

Definition at line 1428 of file util.c.

6.10 Examples

Modules

- Solver component

6.11 Solver component

Modules

- Reconstruction of a glacier from scattered data
- Simple inverse nfft

6.12 Reconstruction of a glacier from scattered data

Functions

- double my_weight (double z, double a, double b, double c)

  Generalised Sobolev weight.

- void glacier (int N, int M)

  Reconstruction routine.

- int main (int argc, char ∗∗argv)

  Main routine.

6.13 Simple inverse nfft

Functions

- void simple_test_infft_1d (int N, int M, int iter)
  Simple test routine for the inverse nfft.
- int main ()
  Main routine.

6.13.1 Function Documentation

6.13.1.1 void simple_test_infft_1d (int *N*, int *M*, int *iter*)

Simple test routine for the inverse nfft.

< index for nodes, freqencies,iter

< plan for the nfft

< plan for the inverse nfft

initialise an one dimensional plan

init pseudo random nodes

precompute psi, the entries of the matrix B

initialise inverse plan

init pseudo random samples and show them

initialise some guess f_hat_0 and solve

Definition at line 15 of file solver/simple_test.c.

References infft_plan::dot_r_iter, nfft_plan::f, nfft_plan::f_hat, infft_plan::f_hat_iter, infft_before_loop(), infft_finalize(), infft_init(), infft_loop_one_step(), nfft_plan::M_total, nfft_plan::N_total, nfft_finalize(), nfft_plan::nfft_flags, nfft_init_1d(), nfft_precompute_one_psi(), NFFT_SWAP_complex, nfft_trafo(), nfft_vpr_complex(), nfft_vrand_shifted_unit_double(), nfft_vrand_unit_complex(), PRE_ONE_PSI, nfft_plan::x, and infft_plan::y.

Referenced by main().

6.14 Applications

Modules

- Fast Gauss transfrom with complex parameter
- Fast summation
- Fast summation of radial functions on the sphere
- Transforms in magnetic resonance imaging
- Polar FFT
- Fast evaluation of quadrature formulae on the sphere

6.15 Fast Gauss transfrom with complex parameter

Data Structures

- struct fgt_plan

  Structure for the Gauss transform.

Defines

- #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

- void dgt_trafo (fgt_plan *ths)

  Executes the discrete Gauss transform.

- void fgt_trafo (fgt_plan *ths)

  Executes the fast Gauss transform.

- void fgt_init_guru (fgt_plan *ths, int N, int M, double complex sigma, int n, double p, int m, unsigned flags)

  Initialisation of a transform plan, guru.

- void fgt_init (fgt_plan *ths, int N, int M, double complex sigma, double eps)

  Initialisation of a transform plan, simple.

- void fgt_init_node_dependent (fgt_plan *ths)

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

- void fgt_finalize (fgt_plan *ths)

  Destroys the transform plan.

- void fgt_test_init_rand (fgt_plan *ths)

  Random initialisation of a fgt plan.

- double fgt_test_measure_time (fgt_plan *ths, unsigned dgt)

  Compares execution times for the fast and discrete Gauss transform.

- void fgt_test_simple (int N, int M, double complex sigma, double eps)

Simple example that computes fast and discrete Gauss transforms.

- void fgt_test_andersson ()

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

- void fgt_test_error ()

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

- void fgt_test_error_p ()

  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.

6.15.1 Define Documentation

6.15.1.1 #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 27 of file fastgauss.c.

Referenced by dgt_trafo(), fgt_init(), fgt_init_node_dependent(), and fgt_test_andersson().

6.15.1.2 #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
ndft_trafo
nfft_trafo

Author:
Stefan Kunis

Definition at line 38 of file fastgauss.c.

Referenced by fgt_test_andersson(), and fgt_trafo().

6.15.1.3 #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 48 of file fastgauss.c.

Referenced by fgt_init_guru().

6.15.2 Function Documentation

6.15.2.1 void dgt_trafo (fgt_plan * *ths*)

Executes the discrete Gauss transform.

- ths The pointer to a fgt plan

Author:
Stefan Kunis

Definition at line 86 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.

Referenced by fgt_test_error(), fgt_test_error_p(), fgt_test_measure_time(), and fgt_test_simple().

6.15.2.2 void fgt_trafo (fgt_plan * *ths*)

Executes the fast Gauss transform.

- ths The pointer to a fgt plan

Author:
Stefan Kunis

Definition at line 111 of file fastgauss.c.

References fgt_plan::b, nfft_plan::f_hat, FGT_NDFT, fgt_plan::flags, fgt_plan::n, ndft_adjoint(), ndft_trafo(), nfft_adjoint(), nfft_trafo(), fgt_plan::nplan1, and fgt_plan::nplan2.

Referenced by fgt_test_error(), fgt_test_error_p(), fgt_test_measure_time(), and fgt_test_simple().

6.15.2.3 void fgt_init_guru (fgt_plan * *ths*, int *N*, int *M*, double complex *sigma*, int *n*, double *p*, int *m*, unsigned *flags*)

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 149 of file fastgauss.c.

References fgt_plan::alpha, fgt_plan::b, nfft_plan::f, fgt_plan::f, nfft_plan::f_hat, FFTW_INIT, FGT_APPROX_B, fgt_plan::flags, fgt_plan::M, MALLOC_F_HAT, MALLOC_X, fgt_plan::n, fgt_plan::N, nfft_fftshift_complex(), nfft_init_guru(), nfft_next_power_of_2(), fgt_plan::nplan1, fgt_plan::nplan2, fgt_plan::p, PI, PRE_PHI_HUT, PRE_PSI, fgt_plan::sigma, fgt_plan::x, and fgt_plan::y.

Referenced by fgt_init(), fgt_test_andersson(), fgt_test_error(), and fgt_test_error_p().

6.15.2.4 void fgt_init (fgt_plan * *ths*, int *N*, int *M*, double complex *sigma*, double *eps*)

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 219 of file fastgauss.c.

References DGT_PRE_CEXP, fgt_init_guru(), and PI.

Referenced by fgt_test_simple().

6.15.2.5 void fgt_init_node_dependent (fgt_plan * *ths*)

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

- ths The pointer to a fpt plan

Author:
Stefan Kunis

Definition at line 242 of file fastgauss.c.

References DGT_PRE_CEXP, fgt_plan::flags, fgt_plan::M, nfft_plan::M_total, fgt_plan::N, nfft_plan::nfft_flags, nfft_precompute_psi(), fgt_plan::nplan1, fgt_plan::nplan2, fgt_plan::p, fgt_plan::pre_cexp, PRE_PSI, fgt_plan::sigma, nfft_plan::x, fgt_plan::x, and fgt_plan::y.

Referenced by fgt_test_andersson(), fgt_test_error(), fgt_test_error_p(), and fgt_test_simple().

6.15.2.6 void fgt_finalize (fgt_plan * *ths*)

Destroys the transform plan.

- ths The pointer to the fgt plan

Author:
Stefan Kunis

Definition at line 274 of file fastgauss.c.

References fgt_plan::alpha, fgt_plan::b, fgt_plan::f, nfft_finalize(), fgt_plan::nplan1, fgt_plan::nplan2, fgt_plan::x, and fgt_plan::y.

Referenced by fgt_test_andersson(), fgt_test_error(), fgt_test_error_p(), and fgt_test_simple().

6.15.2.7 void fgt_test_init_rand (fgt_plan * *ths*)

Random initialisation of a fgt plan.

- ths The pointer to the fgt plan

Author:
Stefan Kunis

Definition at line 297 of file fastgauss.c.

References fgt_plan::alpha, fgt_plan::M, fgt_plan::N, fgt_plan::x, and fgt_plan::y.

Referenced by fgt_test_andersson(), fgt_test_error(), fgt_test_error_p(), and fgt_test_simple().

6.15.2.8 double fgt_test_measure_time (fgt_plan * *ths*, unsigned *dgt*)

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 320 of file fastgauss.c.

References dgt_trafo(), fgt_trafo(), and nfft_second().

Referenced by fgt_test_andersson().

6.15.2.9 void fgt_test_simple (int *N*, int *M*, double complex *sigma*, double *eps*)

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 357 of file fastgauss.c.

References fgt_plan::alpha, dgt_trafo(), fgt_plan::f, fgt_finalize(), fgt_init(), fgt_init_node_dependent(), fgt_test_init_rand(), fgt_trafo(), fgt_plan::M, fgt_plan::N, nfft_error_l_infty_1_complex(), NFFT_SWAP_complex, and nfft_vpr_complex().

Referenced by main().

6.15.2.10 void fgt_test_andersson ()

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 double complex parameters. J. Comput. Physics 203 (2005) 274-286

Author:
Stefan Kunis

Definition at line 392 of file fastgauss.c.

References fgt_plan::alpha, DGT_PRE_CEXP, fgt_plan::f, fgt_finalize(), fgt_init_guru(), fgt_init_node_dependent(), FGT_NDFT, fgt_test_init_rand(), fgt_test_measure_time(), fgt_plan::flags, fgt_plan::M, fgt_plan::N, nfft_error_l_infty_1_complex(), and NFFT_SWAP_complex.

Referenced by main().

6.15.2.11 void fgt_test_error ()

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

Author:
Stefan Kunis

Definition at line 463 of file fastgauss.c.

References fgt_plan::alpha, dgt_trafo(), fgt_plan::f, fgt_finalize(), fgt_init_guru(), fgt_init_node_dependent(), fgt_test_init_rand(), fgt_trafo(), fgt_plan::M, fgt_plan::N, nfft_error_l_infty_1_complex(), and NFFT_SWAP_complex.

Referenced by main().

6.15.2.12 void fgt_test_error_p ()

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

Author:
Stefan Kunis

Definition at line 514 of file fastgauss.c.

References fgt_plan::alpha, dgt_trafo(), fgt_plan::f, fgt_finalize(), fgt_init_guru(), fgt_init_node_dependent(), fgt_test_init_rand(), fgt_trafo(), fgt_plan::M, fgt_plan::N, nfft_error_l_infty_1_complex(), and NFFT_SWAP_complex.

Referenced by main().

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

Different tests of the fast Gauss transform.

Author:
Stefan Kunis

Definition at line 562 of file fastgauss.c.

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

6.16 Fast summation

Modules

- fastsum_matlab
- fastsum_test

Data Structures

- struct fastsum_plan_

  plan for fast summation algorithm

Defines

- #define EXACT_NEARFIELD (1U<< 0)

  Constant symbols.

Typedefs

- typedef fastsum_plan_ fastsum_plan

  plan for fast summation algorithm

Functions

- double fak (int n)

  factorial

- double binom (int n, int m)

  binomial coefficient

- double BasisPoly (int m, int r, double xx)

  basis polynomial for regularized kernel

- double regkern (complex(∗kernel)(double, int, const double ∗), double xx, int p, const double ∗param, double a, double b)

  regularized kernel with K_I arbitrary and K_B smooth to zero

- double regkern1 (complex(∗kernel)(double, int, const double ∗), double xx, int p, const double ∗param, double a, double b)

  regularized kernel with K_I arbitrary and K_B periodized (used in 1D)

- double regkern2 (complex(∗kernel)(double, int, const double ∗), double xx, int p, const double ∗param, double a, double b)

  regularized kernel for even kernels with K_I even and K_B mirrored

- double regkern3 (complex(∗kernel)(double, int, const double ∗), double xx, int p, const double ∗param, double a, double b)

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

- double kubintkern (double x, double ∗Add, int Ad, double a)

  cubic spline interpolation in near field with even kernels

- double kubintkern1 (double x, double ∗Add, int Ad, double a)

  cubic spline interpolation in near field with arbitrary kernels

- void quicksort (int d, int t, double ∗x, complex ∗alpha, int N)

  quicksort algorithm for source knots and associated coefficients

- void BuildTree (int d, int t, double ∗x, complex ∗alpha, int N)

  recursive sort of source knots dimension by dimension to get tree structure

- complex SearchTree (int d, int t, double ∗x, complex ∗alpha, double ∗xmin, double ∗xmax, int N, complex(∗kernel)(double, int, const double ∗), const double ∗param, int Ad, double ∗Add, int p, unsigned flags)

  fast search in tree of source knots for near field computation

- void fastsum_init_guru (fastsum_plan ∗ths, int d, int N_total, int M_total, complex(∗kernel)(), double ∗param, unsigned flags, int nn, int m, int p, double eps_I, double eps_B)

  initialization 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 (fastsum_plan ∗ths)

  precomputation for fastsum

- void fastsum_trafo (fastsum_plan ∗ths)

  fast NFFT-based summation

- complex gaussian (double x, int der, const double ∗param)
- complex multiquadric (double x, int der, const double ∗param)
- complex inverse_multiquadric (double x, int der, const double ∗param)
- complex logarithm (double x, int der, const double ∗param)
- complex thinplate_spline (double x, int der, const double ∗param)
- complex one_over_square (double x, int der, const double ∗param)
- complex one_over_modulus (double x, int der, const double ∗param)
- complex one_over_x (double x, int der, const double ∗param)
- complex inverse_multiquadric3 (double x, int der, const double ∗param)
- complex sinc_kernel (double x, int der, const double ∗param)
- complex cosc (double x, int der, const double ∗param)
- complex cot (double x, int der, const double ∗param)

6.16.1 Function Documentation

6.16.1.1 void fastsum_init_guru (fastsum_plan * *ths*, int *d*, int *N_total*, int *M_total*, complex(*)() *kernel*, double * *param*, unsigned *flags*, int *nn*, int *m*, int *p*, double *eps_I*, double *eps_B*)

initialization of fastsum plan

inner boundary

outer boundary

init spline for near field computation

init d-dimensional NFFT plan

init d-dimensional FFTW plan

Definition at line 370 of file fastsum.c.

References fastsum_plan_::Ad, fastsum_plan_::Add, fastsum_plan_::alpha, fastsum_plan_::b, fastsum_plan_::d, fastsum_plan_::eps_B, fastsum_plan_::eps_I, EXACT_NEARFIELD, fastsum_plan_::f, FFT_OUT_OF_PLACE, fastsum_plan_::fft_plan, FFTW_INIT, fastsum_plan_::flags, fastsum_plan_::kernel, fastsum_plan_::kernel_param, fastsum_plan_::M_total, MALLOC_F, MALLOC_F_HAT, MALLOC_X, fastsum_plan_::mv1, fastsum_plan_::mv2, fastsum_plan_::n, fastsum_plan_::N_total, nfft_init_guru(), fastsum_plan_::p, PRE_PHI_HUT, PRE_PSI, fastsum_plan_::x, and fastsum_plan_::y.

Referenced by main().

6.16.1.2 void fastsum_finalize (fastsum_plan * *ths*)

finalization of fastsum plan

Parameters:
ths The pointer to a fastsum plan.

Definition at line 436 of file fastsum.c.

References fastsum_plan_::Add, fastsum_plan_::alpha, fastsum_plan_::b, EXACT_NEARFIELD, fastsum_plan_::f, fastsum_plan_::fft_plan, fastsum_plan_::flags, fastsum_plan_::mv1, fastsum_plan_::mv2, nfft_finalize(), fastsum_plan_::x, and fastsum_plan_::y.

Referenced by main().

6.16.1.3 void fastsum_exact (fastsum_plan * *ths*)

direct computation of sums

Parameters:
ths The pointer to a fastsum plan.

Definition at line 454 of file fastsum.c.

References fastsum_plan_::alpha, fastsum_plan_::d, fastsum_plan_::f, fastsum_plan_::kernel, fastsum_plan_::kernel_param, fastsum_plan_::x, and fastsum_plan_::y.

Referenced by main().

6.16.1.4 void fastsum_precompute (fastsum_plan * *ths*)

precomputation for fastsum

sort source knots

precompute spline values for near field

init NFFT plan for transposed transform in first step

precompute psi, the entries of the matrix B

init Fourier coefficients

init NFFT plan for transform in third step

precompute psi, the entries of the matrix B

precompute Fourier coefficients of regularised kernel

Definition at line 480 of file fastsum.c.

References fastsum_plan_::Ad, fastsum_plan_::Add, fastsum_plan_::alpha, fastsum_plan_::b, BuildTree(), nfft_plan::d, fastsum_plan_::d, fastsum_plan_::eps_B, fastsum_plan_::eps_I, EXACT_NEARFIELD, nfft_plan::f, fastsum_plan_::fft_plan, fastsum_plan_::flags, fastsum_plan_::kernel, fastsum_plan_::kernel_param, nfft_plan::M_total, fastsum_plan_::mv1, fastsum_plan_::mv2, nfft_plan::N, fastsum_plan_::n, fastsum_plan_::N_total, nfft_fftshift_complex(), nfft_plan::nfft_flags, nfft_precompute_full_psi(), nfft_precompute_lin_psi(), nfft_precompute_psi(), fastsum_plan_::p, PRE_FULL_PSI, PRE_LIN_PSI, PRE_PSI, regkern1(), regkern3(), nfft_plan::x, fastsum_plan_::x, and fastsum_plan_::y.

Referenced by main().

6.16.1.5 void fastsum_trafo (fastsum_plan * *ths*)

fast NFFT-based summation

limits for d-dimensional near field box

first step of algorithm

second step of algorithm

third step of algorithm

add near field

Definition at line 563 of file fastsum.c.

References fastsum_plan_::Ad, fastsum_plan_::Add, fastsum_plan_::alpha, fastsum_plan_::b, fastsum_plan_::d, fastsum_plan_::eps_I, nfft_plan::f, fastsum_plan_::f, nfft_plan::f_hat, fastsum_plan_::flags, fastsum_plan_::kernel, fastsum_plan_::kernel_param, fastsum_plan_::mv1, fastsum_plan_::mv2, fastsum_plan_::N_total, nfft_adjoint(), nfft_trafo(), fastsum_plan_::p, SearchTree(), fastsum_plan_::x, and fastsum_plan_::y.

Referenced by main().

6.17 fastsum_matlab

Functions

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

6.17.1 Function Documentation

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

< indices

< number of dimensions

< number of source nodes

< number of target nodes

< expansion degree

< cut-off parameter

< degree of smoothness

< name of kernel

< kernel function

< parameter for kernel

< plan for fast summation

< array for direct computation

< for time measurement

< for error computation

< inner boundary

< outer boundary

init two dimensional fastsum plan

load source knots and coefficients

load target knots

direct computation

copy result

precomputation

fast computation

compute max error

write result to file

finalise the plan

Definition at line 23 of file fastsum_matlab.c.

References fastsum_plan_::alpha, cosc(), cot(), fastsum_plan_::f, fastsum_exact(), fastsum_finalize(), fastsum_init_guru(), fastsum_precompute(), fastsum_trafo(), gaussian(), inverse_multiquadric(), inverse_multiquadric3(), logarithm(), fastsum_plan_::M_total, multiquadric(), nfft_second(), one_over_modulus(),

one_over_square(), one_over_x(), sinc_kernel(), thinplate_spline(), fastsum_plan_::x, and fastsum_plan_-::y.

6.18 fastsum_test

Functions

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

6.18.1 Function Documentation

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

< indices

< number of dimensions

< number of source nodes

< number of target nodes

< expansion degree

< cut-off parameter

< degree of smoothness

< name of kernel

< kernel function

< parameter for kernel

< plan for fast summation

< array for direct computation

< for time measurement

< for error computation

< inner boundary

< outer boundary

init two dimensional fastsum plan

init source knots in a d-ball with radius 0.25-eps_b/2

init target knots in a d-ball with radius 0.25-eps_b/2

direct computation

copy result

precomputation

fast computation

compute max error

finalise the plan

Definition at line 23 of file fastsum_test.c.

References fastsum_plan_::alpha, cosc(), cot(), fastsum_plan_::eps_B, fastsum_plan_::f, fastsum_exact(), fastsum_finalize(), fastsum_init_guru(), fastsum_precompute(), fastsum_trafo(), gaussian(), inverse_multiquadric(), inverse_multiquadric3(), logarithm(), fastsum_plan_::M_total, multiquadric(), nfft_second(), one_over_modulus(), one_over_square(), one_over_x(), PI, sinc_kernel(), thinplate_spline(),

fastsum_plan_::x, and fastsum_plan_::y.

6.19 Fast summation of radial functions on the sphere

Modules

- fastsumS2_matlab

6.20 fastsumS2_matlab

Defines

- #define SYMBOL_ABEL_POISSON(k, h) (pow(h,k))

  Macro for the Fourier-Legendre coefficients of the Abel-Poisson kernel.

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

  Macro for the Fourier-Legendre coefficients of the singularity kernel.

- #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

- 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.

- double poissonKernel (const double x, const double h)

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

- double singularityKernel (const double x, const double h)

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

- 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 $x \in [-1,1]$.

- 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.

6.20.1 Function Documentation

6.20.1.1 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.

- 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 76 of file fastsumS2.c.

Referenced by main().

6.20.1.2 double poissonKernel (const double *x*, const double *h*)

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

- 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 93 of file fastsumS2.c.

References PI.

Referenced by main().

6.20.1.3 double singularityKernel (const double *x*, const double *h*)

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

- 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 109 of file fastsumS2.c.

References PI.

Referenced by main().

6.20.1.4 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 $x \in [-1,1]$.

- 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 127 of file fastsumS2.c.

Referenced by main().

6.20.1.5 double gaussianKernel (const double *x*, const double *sigma*)

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

- 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 145 of file fastsumS2.c.

Referenced by main().

6.20.1.6 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 160 of file fastsumS2.c.

References nfsft_plan::f, nfsft_plan::f_hat, FFT_OUT_OF_PLACE, FFTW_INIT, gaussianKernel(), innerProduct(), KT_ABEL_POISSON, KT_GAUSSIAN, KT_LOC_SUPP, KT_SINGULARITY, locallySupportedKernel(), ndsft_adjoint(), ndsft_trafo(), nfft_error_l_infty_1_complex(), NFFT_MAX, nfft_second(), nfsft_adjoint(), NFSFT_F_HAT_SIZE, nfsft_finalize(), nfsft_forget(), NFSFT_INDEX, nfsft_init_guru(), NFSFT_NO_FAST_ALGORITHM, nfsft_precompute(), nfsft_precompute_x(), nfsft_trafo(), NFSFT_USE_DPT, NFSFT_USE_NDFT, PI, poissonKernel(), PRE_PHI_HUT, PRE_PSI, singularityKernel(), SYMBOL_ABEL_POISSON, SYMBOL_SINGULARITY, and nfsft_plan::x.

6.21 Transforms in magnetic resonance imaging

Modules

- 2D transforms
- 3D transforms

6.22 construct_data_2d

Functions

- void construct (char *file, int N, int M)

  construct makes an 2d-nfft

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

6.23 construct_data__inh_2d1d

Functions

- void construct (char *file, int N, int M)

  construct

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

6.24 construct_data_inh_3d

Functions

- void construct (char *file, int N, int M)

  construct

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

6.25 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

6.26 reconstruct_data_2d

Functions

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

  reconstruct makes an inverse 2d nfft

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

6.27 construct_data_gridding

Functions

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

  reconstruct makes a 2d-adjoint-nfft

- int main (int argc, char ∗∗argv)

6.28 reconstruct_data__inh_2d1d

Functions

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

6.29 reconstruct_data_inh_3d

Functions

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

6.30 construct_data_inh_nnfft

Functions

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

  reconstruct

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

6.31 construct_data_1d2d

Functions

- void construct (char *file, int N, int M, int Z, fftw_complex *mem)
  construct makes an 2d-nfft for every slice
- void fft (int N, int M, int Z, fftw_complex *mem)
  fft makes an 1D-ftt for every knot through all layers
- 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)

6.32 construct_data_3d

Functions

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

6.33 3D transforms

Modules

- construct_data_1d2d
- construct_data_3d
- reconstruct_data_1d2d
- reconstruct_data_3d
- reconstruct_data_gridding

6.34 reconstruct_data_1d2d

Functions

- 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

- 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)

6.35 reconstruct_data_3d

Functions

- 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)

6.36 reconstruct_data_gridding

Functions

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

  reconstruct makes an 2d-adjoint-nfft for every slice

- 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)

6.37 Polar FFT

Modules

- linogram_fft_test
- mpolar_fft_test
- polar_fft_test

6.38 linogram_fft_test

Functions

- int linogram_grid (int T, int R, double *x, double *w)

  Generates the points x with weights w for the linogram grid with T slopes and R offsets.

- int linogram_dft (fftw_complex *f_hat, int NN, fftw_complex *f, int T, int R, int m)

  discrete pseudo-polar FFT

- int linogram_fft (fftw_complex *f_hat, int NN, fftw_complex *f, int T, int R, int m)

  NFFT-based pseudo-polar FFT.

- int inverse_linogram_fft (fftw_complex *f, int T, int R, fftw_complex *f_hat, int NN, int max_i, int m)

  NFFT-based inverse pseudo-polar FFT.

- int comparison_fft (FILE *fp, int N, int T, int R)

  Comparison of the FFTW, linogram FFT, and inverse linogram FFT.

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

  test program for various parameters

Variables

- double GLOBAL_elapsed_time

6.38.1 Function Documentation

6.38.1.1 int linogram_grid (int *T*, int *R*, double * *x*, double * *w*)

Generates the points x with weights w for the linogram grid with T slopes and R offsets.

return the number of knots

Definition at line 25 of file linogram_fft_test.c.

Referenced by inverse_linogram_fft(), linogram_dft(), linogram_fft(), and main().

6.38.1.2 int linogram_dft (fftw_complex * *f_hat*, int *NN*, fftw_complex * *f*, int *T*, int *R*, int *m*)

discrete pseudo-polar FFT

< index for nodes and freqencies

< plan for the nfft-2D

< knots and associated weights

< number of knots

< oversampling factor sigma=2

< oversampling factor sigma=2

init two dimensional NFFT plan

init nodes from linogram grid

init Fourier coefficients from given image

NFFT-2D

copy result

finalise the plans and free the variables

Definition at line 55 of file linogram_fft_test.c.

References nfft_plan::f, nfft_plan::f_hat, FFT_OUT_OF_PLACE, FFTW_INIT, GLOBAL_elapsed_time, linogram_grid(), nfft_plan::M_total, MALLOC_F, MALLOC_F_HAT, MALLOC_X, nfft_plan::N_total, ndft_trafo(), nfft_finalize(), nfft_init_guru(), nfft_second(), PRE_PHI_HUT, PRE_PSI, and nfft_plan::x.

Referenced by comparison_fft(), and main().

6.38.1.3 int linogram_fft (fftw_complex * *f_hat*, int *NN*, fftw_complex * *f*, int *T*, int *R*, int *m*)

NFFT-based pseudo-polar FFT.

< index for nodes and freqencies

< plan for the nfft-2D

< knots and associated weights

< number of knots

< oversampling factor sigma=2

< oversampling factor sigma=2

init two dimensional NFFT plan

init nodes from linogram grid

precompute psi, the entries of the matrix B

init Fourier coefficients from given image

NFFT-2D

copy result

finalise the plans and free the variables

Definition at line 112 of file linogram_fft_test.c.

References nfft_plan::f, nfft_plan::f_hat, FFT_OUT_OF_PLACE, FFTW_INIT, GLOBAL_elapsed_time, linogram_grid(), nfft_plan::M_total, MALLOC_F, MALLOC_F_HAT, MALLOC_X, nfft_plan::N_total, nfft_finalize(), nfft_plan::nfft_flags, nfft_init_guru(), nfft_precompute_full_psi(), nfft_precompute_lin_psi(), nfft_precompute_psi(), nfft_second(), nfft_trafo(), PRE_FULL_PSI, PRE_LIN_PSI, PRE_PHI_HUT, PRE_PSI, and nfft_plan::x.

Referenced by comparison_fft(), and main().

6.38.1.4 int inverse_linogram_fft (fftw_complex * *f*, int *T*, int *R*, fftw_complex * *f_hat*, int *NN*, int *max_i*, int *m*)

NFFT-based inverse pseudo-polar FFT.

< index for nodes and freqencies

< plan for the nfft-2D

< plan for the inverse nfft

< knots and associated weights

< index for iterations

< number of knots

< oversampling factor sigma=2

< oversampling factor sigma=2

init two dimensional NFFT plan

init two dimensional infft plan

init nodes, given samples and weights

precompute psi, the entries of the matrix B

initialise damping factors

initialise some guess f_hat_0

solve the system

copy result

finalise the plans and free the variables

Definition at line 178 of file linogram_fft_test.c.

References CGNR, infft_plan::f_hat_iter, FFT_OUT_OF_PLACE, FFTW_INIT, infft_plan::flags, GLOBAL_elapsed_time, infft_before_loop(), infft_finalize(), infft_init_advanced(), infft_loop_one_step(), linogram_grid(), nfft_plan::M_total, MALLOC_F, MALLOC_F_HAT, MALLOC_X, infft_plan::mv, nfft_plan::N, nfft_plan::N_total, nfft_finalize(), nfft_plan::nfft_flags, nfft_init_guru(), nfft_precompute_full_psi(), nfft_precompute_lin_psi(), nfft_precompute_psi(), nfft_second(), infft_plan::p_hat_iter, PRE_FULL_PSI, PRE_LIN_PSI, PRE_PHI_HUT, PRE_PSI, PRECOMPUTE_DAMP, PRECOMPUTE_WEIGHT, infft_plan::w, infft_plan::w_hat, nfft_plan::x, and infft_plan::y.

Referenced by comparison_fft(), and main().

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

test program for various parameters

< linogram FFT size NxN

< number of directions/offsets

< number of knots of linogram grid

< knots and associated weights

< number of iterations

Hence, comparison of the FFTW, linogram FFT, and inverse linogram FFT

generate knots of linogram grid

load data

direct linogram FFT

Test of the linogram FFT with different m

fast linogram FFT

error of fast linogram FFT

Test of the inverse linogram FFT for different m in dependece of the iteration number

inverse linogram FFT

compute maximum error

free the variables

Definition at line 339 of file linogram_fft_test.c.

References comparison_fft(), inverse_linogram_fft(), linogram_dft(), linogram_fft(), linogram_grid(), and nfft_error_l_infty_complex().

6.39 mpolar_fft_test

Functions

- int mpolar_grid (int T, int R, double *x, double *w)

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

- int mpolar_dft (fftw_complex *f_hat, int NN, fftw_complex *f, int T, int R, int m)

  discrete mpolar FFT

- int mpolar_fft (fftw_complex *f_hat, int NN, fftw_complex *f, int T, int R, int m)

  NFFT-based mpolar FFT.

- int inverse_mpolar_fft (fftw_complex *f, int T, int R, fftw_complex *f_hat, int NN, int max_i, int m)

  inverse NFFT-based mpolar FFT

- int comparison_fft (FILE *fp, int N, int T, int R)

  Comparison of the FFTW, mpolar FFT, and inverse mpolar FFT.

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

  test program for various parameters

Variables

- double GLOBAL_elapsed_time

6.39.1 Function Documentation

6.39.1.1 int mpolar_grid (int *T*, int *R*, double * *x*, double * *w*)

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 $M \approx \frac{4}{\pi} \log(1+\sqrt{2}) TR$.

count the knots

normalize the weights

return the number of knots

Definition at line 36 of file mpolar_fft_test.c.

References PI.

Referenced by inverse_mpolar_fft(), main(), mpolar_dft(), and mpolar_fft().

6.39.1.2 int mpolar_dft (fftw_complex * *f_hat*, int *NN*, fftw_complex * *f*, int *T*, int *R*, int *m*)

discrete mpolar FFT

< index for nodes and freqencies

< plan for the nfft-2D

< knots and associated weights

< number of knots

< oversampling factor sigma=2

< oversampling factor sigma=2

init two dimensional NFFT plan

init nodes from mpolar grid

init Fourier coefficients from given image

NDFT-2D

copy result

finalise the plans and free the variables

Definition at line 79 of file mpolar_fft_test.c.

References nfft_plan::f, nfft_plan::f_hat, FFT_OUT_OF_PLACE, FFTW_INIT, GLOBAL_elapsed_time, nfft_plan::M_total, MALLOC_F, MALLOC_F_HAT, MALLOC_X, mpolar_grid(), nfft_plan::N_total, ndft_trafo(), nfft_finalize(), nfft_init_guru(), nfft_second(), PRE_PHI_HUT, PRE_PSI, and nfft_plan::x.

Referenced by comparison_fft(), and main().

6.39.1.3 int mpolar_fft (fftw_complex * *f_hat*, int *NN*, fftw_complex * *f*, int *T*, int *R*, int *m*)

NFFT-based mpolar FFT.

< index for nodes and freqencies

< plan for the nfft-2D

< knots and associated weights

< number of knots

< oversampling factor sigma=2

< oversampling factor sigma=2

init two dimensional NFFT plan

init nodes from mpolar grid

precompute psi, the entries of the matrix B

init Fourier coefficients from given image

NFFT-2D

copy result

finalise the plans and free the variables

Definition at line 137 of file mpolar_fft_test.c.

References nfft_plan::f, nfft_plan::f_hat, FFT_OUT_OF_PLACE, FFTW_INIT, GLOBAL_elapsed_time,

nfft_plan::M_total, MALLOC_F, MALLOC_F_HAT, MALLOC_X, mpolar_grid(), nfft_plan::N_total, nfft_finalize(), nfft_plan::nfft_flags, nfft_init_guru(), nfft_precompute_full_psi(), nfft_precompute_lin_psi(), nfft_precompute_psi(), nfft_second(), nfft_trafo(), PRE_FULL_PSI, PRE_LIN_PSI, PRE_PHI_HUT, PRE_PSI, and nfft_plan::x.

Referenced by comparison_fft(), and main().

6.39.1.4 int inverse_mpolar_fft (fftw_complex * *f*, int *T*, int *R*, fftw_complex * *f_hat*, int *NN*, int *max_i*, int *m*)

inverse NFFT-based mpolar FFT

< index for nodes and freqencies

< plan for the nfft-2D

< plan for the inverse nfft

< knots and associated weights

< index for iterations

< number of knots

< oversampling factor sigma=2

< oversampling factor sigma=2

init two dimensional NFFT plan

init two dimensional infft plan

init nodes, given samples and weights

precompute psi, the entries of the matrix B

initialise damping factors

initialise some guess f_hat_0

solve the system

copy result

finalise the plans and free the variables

Definition at line 206 of file mpolar_fft_test.c.

References CGNR, infft_plan::f_hat_iter, FFT_OUT_OF_PLACE, FFTW_INIT, infft_plan::flags, GLOBAL_elapsed_time, infft_before_loop(), infft_finalize(), infft_init_advanced(), infft_loop_one_step(), nfft_plan::M_total, MALLOC_F, MALLOC_F_HAT, MALLOC_X, mpolar_grid(), infft_plan::mv, nfft_plan::N, nfft_plan::N_total, nfft_finalize(), nfft_plan::nfft_flags, nfft_init_guru(), nfft_precompute_full_psi(), nfft_precompute_lin_psi(), nfft_precompute_psi(), nfft_second(), infft_plan::p_hat_iter, PRE_FULL_PSI, PRE_LIN_PSI, PRE_PHI_HUT, PRE_PSI, PRECOMPUTE_DAMP, PRECOMPUTE_WEIGHT, infft_plan::w, infft_plan::w_hat, nfft_plan::x, and infft_plan::y.

Referenced by comparison_fft(), and main().

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

test program for various parameters

< mpolar FFT size NxN

< number of directions/offsets

< number of knots of mpolar grid

< knots and associated weights

< number of iterations

Hence, comparison of the FFTW, mpolar FFT, and inverse mpolar FFT

generate knots of mpolar grid

load data

direct mpolar FFT

Test of the mpolar FFT with different m

fast mpolar FFT

compute error of fast mpolar FFT

Test of the inverse mpolar FFT for different m in dependece of the iteration number

inverse mpolar FFT

compute maximum relativ error

free the variables

Definition at line 369 of file mpolar_fft_test.c.

References comparison_fft(), inverse_mpolar_fft(), mpolar_dft(), mpolar_fft(), mpolar_grid(), and nfft_error_l_infty_complex().

6.40 polar_fft_test

Functions

- int polar_grid (int T, int R, double *x, double *w)

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

- int polar_dft (fftw_complex *f_hat, int NN, fftw_complex *f, int T, int R, int m)

  discrete polar FFT

- int polar_fft (fftw_complex *f_hat, int NN, fftw_complex *f, int T, int R, int m)

  NFFT-based polar FFT.

- int inverse_polar_fft (fftw_complex *f, int T, int R, fftw_complex *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

6.40.1 Function Documentation

6.40.1.1 int polar_grid (int *T*, int *R*, double * *x*, double * *w*)

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.

return the number of knots

Definition at line 51 of file polar_fft_test.c.

References PI.

Referenced by inverse_polar_fft(), main(), polar_dft(), and polar_fft().

6.40.1.2 int polar_dft (fftw_complex * *f_hat*, int *NN*, fftw_complex * *f*, int *T*, int *R*, int *m*)

discrete polar FFT

< index for nodes and frequencies

< plan for the nfft-2D

< knots and associated weights

< number of knots

< oversampling factor sigma=2

< oversampling factor sigma=2

init two dimensional NFFT plan

init nodes from polar grid

init Fourier coefficients from given image

NDFT-2D

copy result

finalise the plans and free the variables

Definition at line 73 of file polar_fft_test.c.

References nfft_plan::f, nfft_plan::f_hat, FFT_OUT_OF_PLACE, FFTW_INIT, nfft_plan::M_total, MALLOC_F, MALLOC_F_HAT, MALLOC_X, nfft_plan::N_total, ndft_trafo(), nfft_finalize(), nfft_init_guru(), polar_grid(), PRE_PHI_HUT, PRE_PSI, and nfft_plan::x.

Referenced by main().

6.40.1.3 int polar_fft (fftw_complex * *f_hat*, int *NN*, fftw_complex * *f*, int *T*, int *R*, int *m*)

NFFT-based polar FFT.

< index for nodes and freqencies

< plan for the nfft-2D

< knots and associated weights

< number of knots

< oversampling factor sigma=2

< oversampling factor sigma=2

init two dimensional NFFT plan

init nodes from polar grid

precompute psi, the entries of the matrix B

init Fourier coefficients from given image

NFFT-2D

copy result

finalise the plans and free the variables

Definition at line 127 of file polar_fft_test.c.

References nfft_plan::f, nfft_plan::f_hat, FFT_OUT_OF_PLACE, FFTW_INIT, nfft_plan::M_total,

MALLOC_F, MALLOC_F_HAT, MALLOC_X, nfft_plan::N_total, nfft_finalize(), nfft_plan::nfft_flags, nfft_init_guru(), nfft_precompute_full_psi(), nfft_precompute_lin_psi(), nfft_precompute_psi(), nfft_trafo(), polar_grid(), PRE_FULL_PSI, PRE_LIN_PSI, PRE_PHI_HUT, PRE_PSI, and nfft_plan::x.

Referenced by main().

6.40.1.4 int inverse_polar_fft (fftw_complex * *f*, int *T*, int *R*, fftw_complex * *f_hat*, int *NN*, int *max_i*, int *m*)

inverse NFFT-based polar FFT

< index for nodes and freqencies

< plan for the nfft-2D

< plan for the inverse nfft

< knots and associated weights

< index for iterations

< number of knots

< oversampling factor sigma=2

< oversampling factor sigma=2

init two dimensional NFFT plan

init two dimensional infft plan

init nodes, given samples and weights

precompute psi, the entries of the matrix B

initialise damping factors

initialise some guess f_hat_0

solve the system

copy result

finalise the plans and free the variables

Definition at line 191 of file polar_fft_test.c.

References CGNR, infft_plan::f_hat_iter, FFT_OUT_OF_PLACE, FFTW_INIT, infft_plan::flags, infft_before_loop(), infft_finalize(), infft_init_advanced(), infft_loop_one_step(), nfft_plan::M_total, MALLOC_F, MALLOC_F_HAT, MALLOC_X, infft_plan::mv, nfft_plan::N, nfft_plan::N_total, nfft_finalize(), nfft_plan::nfft_flags, nfft_init_guru(), nfft_precompute_full_psi(), nfft_precompute_lin_psi(), nfft_precompute_psi(), infft_plan::p_hat_iter, polar_grid(), PRE_FULL_PSI, PRE_LIN_PSI, PRE_PHI_HUT, PRE_PSI, PRECOMPUTE_DAMP, PRECOMPUTE_WEIGHT, infft_plan::w, infft_plan::w_hat, nfft_plan::x, and infft_plan::y.

Referenced by main().

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

test program for various parameters

< mpolar FFT size NxN

< number of directions/offsets

< number of knots of mpolar grid

< knots and associated weights

< number of iterations

generate knots of mpolar grid

load data

direct polar FFT

Test of the polar FFT with different m

fast polar FFT

compute error of fast polar FFT

Test of the inverse polar FFT for different m in dependece of the iteration number

inverse polar FFT

compute maximum relative error

free the variables

Definition at line 286 of file polar_fft_test.c.

References inverse_polar_fft(), nfft_error_l_infty_complex(), polar_dft(), polar_fft(), and polar_grid().

6.41 Fast evaluation of quadrature formulae on the sphere

Modules

- quadratureS2_test

6.42 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_EQUIDISTRIBUTION** = 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.

6.42.1 Function Documentation

6.42.1.1 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

$<$ The index variable for testcases

$<$ The number of testcases

$<$ The array containing the cut-off degrees $*$ N

< The maximum cut-off degree N for the$*$ current testcase

< The array containing the grid size parameters

< The maximum grid size parameter

< The array containing the grid size parameters

< Index variable for cut-off degrees

< The maximum number of cut-off degrees

< The testfunction

< The test function's bandwidth

< Whether to use the NFSFT algorithm or not

< Whether to use the NFFT algorithm or not

< Whether to use the FPT algorithm or not

< The current NFFT cut-off parameter

< The current NFSFT threshold parameter

< The type of quadrature grid to be used

< The number of repetitions to be performed

< The number of repetitions to be performed

< The quadrature weights

< The quadrature nodes

< The quadrature nodes

< The reference function values

< The function values

< The function values

< The reference spherical Fourier $*$ coefficients

< The spherical Fourier coefficients

< The NFSFT plan

< The NFSFT plan

< The NFSFT plan

< The computation time needed for a single $*$ run

< The average computation time needed

< The average error E_∞

< The average error E_2

< A loop variable

< A loop variable

< A loop variable

< A loop variable

< The current number of different $*$ colatitudinal angles (for grids)

< The current number of different ∗ longitudinal angles (for grids).

< The total number nodes.

< An array for saving the angles theta of a ∗ grid

< An array for saving the angles phi of a ∗ grid

< An FFTW plan for computing Clenshaw-Curtis quadrature weights

Definition at line 71 of file quadratureS2.c.

References nfsft_plan::f, nfsft_plan::f_hat, FFT_OUT_OF_PLACE, FFTW_INIT, nfsft_plan::N_total, ndsft_adjoint(), ndsft_trafo(), nfft_error_l_2_complex(), nfft_error_l_infty_complex(), NFFT_MAX, nfft_second(), nfsft_adjoint(), NFSFT_F_HAT_SIZE, nfsft_finalize(), nfsft_forget(), NFSFT_INDEX, nfsft_init_guru(), NFSFT_NO_FAST_ALGORITHM, NFSFT_NORMALIZED, nfsft_precompute(), nfsft_precompute_x(), nfsft_trafo(), NFSFT_USE_DPT, NFSFT_USE_NDFT, PI, PRE_PHI_HUT, PRE_-PSI, nfsft_wisdom::threshold, and nfsft_plan::x.

Chapter 7

NFFT Directory Documentation

7.1 examples/nnfft/accuracy/ Directory Reference

Files

- file **accuracy.c**

7.2 applications/ Directory Reference

Directories

- directory fastgauss
- directory fastsum
- directory fastsumS2
- directory mri
- directory polarFFT
- directory quadratureS2
- directory radon

Files

- file **applications/doxygen.c**

7.3 examples/ Directory Reference

Directories

- directory fpt
- directory nfct
- directory nfft
- directory nfsft
- directory nfst
- directory nnfft
- directory nsfft
- directory solver

Files

- file **examples/doxygen.c**

7.4 applications/fastgauss/ Directory Reference

Files

- file **fastgauss.c**

7.5 applications/fastsum/ Directory Reference

Files

- file fastsum.c

  Fast NFFT-based summation algorithm.

- file fastsum.h

  Header file for the fast NFFT-based summation algorithm.

- file fastsum_matlab.c

  Simple test program for the fast NFFT-based summation algorithm, called by fastsum.m.

- file fastsum_test.c

  Simple test program for the fast NFFT-based summation algorithm.

- file kernels.c

  File with predefined kernels for the fast summation algorithm.

- file kernels.h

  Header file with predefined kernels for the fast summation algorithm.

7.6 applications/fastsumS2/ Directory Reference

Files

- file **applications/fastsumS2/doxygen.c**
- file **fastsumS2.c**

7.7 examples/fpt/ Directory Reference

Files

- file **fpt/simple_test.c**

7.8 kernel/fpt/ Directory Reference

Files

- file fpt.c

  Implementation file for the FPT module.

7.9 include/ Directory Reference

Files

- file **config.h**
- file nfft3.h

  Header file for the nfft3 library.

- file **options.h**
- file util.h

  Header file for utility functions used by the nfft3 library.

- file **window_defines.h**

7.10 kernel/ Directory Reference

Directories

- directory fpt
- directory mri
- directory nfct
- directory nfft
- directory nfsft
- directory nfst
- directory nnfft
- directory nsfft
- directory solver

7.11 kernel/mri/ Directory Reference

Files

- file **mri.c**

7.12 applications/mri/ Directory Reference

Directories

- directory mri2d
- directory mri3d

Files

- file **applications/mri/doxygen.c**

7.13 applications/mri/mri2d/ Directory Reference

Files

- file **construct_data_2d.c**
- file **construct_data_inh_2d1d.c**
- file **construct_data_inh_3d.c**
- file **applications/mri/mri2d/doxygen.c**
- file **reconstruct_data_2d.c**
- file **mri2d/reconstruct_data_gridding.c**
- file **reconstruct_data_inh_2d1d.c**
- file **reconstruct_data_inh_3d.c**
- file **reconstruct_data_inh_nnfft.c**

7.14 applications/mri/mri3d/ Directory Reference

Files

- file **construct_data_2d1d.c**
- file **construct_data_3d.c**
- file **applications/mri/mri3d/doxygen.c**
- file **reconstruct_data_2d1d.c**
- file **reconstruct_data_3d.c**
- file **mri3d/reconstruct_data_gridding.c**

7.15 examples/nfct/ Directory Reference

Files

- file **nfct/simple_test.c**

7.16 kernel/nfct/ Directory Reference

Files

- file **nfct.c**

7.17 kernel/nfft/ Directory Reference

Files

- file **nfft.c**

7.18 examples/nfft/ Directory Reference

Files

- file flags.c

  Testing the nfft.

- file ndft_fast.c

  Testing ndft, Horner-like ndft, and fully precomputed ndft.

- file **nfft_times.c**
- file **nfft/simple_test.c**
- file taylor_nfft.c

  Testing the nfft againt a Taylor expansion based version.

7.19 examples/nfsft/ Directory Reference

Files

- file **nfsft/simple_test.c**

7.20 kernel/nfsft/ Directory Reference

Files

- file api.h

  Header file with internal API of the NFSFT module.

- file **legendre.c**
- file **legendre.h**
- file nfsft.c

  Implementation file for the NFSFT module.

7.21 examples/nfst/ Directory Reference

Files

- file **nfst/simple_test.c**

7.22 kernel/nfst/ Directory Reference

Files

- file **nfst.c**

7.23 kernel/nnfft/ Directory Reference

Files

- file **nnfft.c**

7.24 examples/nnfft/ Directory Reference

Directories

- directory accuracy
- directory simple_test

7.25 examples/nsfft/ Directory Reference

Files

- file **nsfft_test.c**
- file **nsfft/simple_test.c**

7.26 kernel/nsfft/ Directory Reference

Files

- file **nsfft.c**

7.27 applications/polarFFT/ Directory Reference

Files

- file **applications/polarFFT/doxygen.c**
- file linogram_fft_test.c

  NFFT-based pseudo-polar FFT and inverse.

- file mpolar_fft_test.c

  NFFT-based polar FFT and inverse on modified polar grid.

- file polar_fft_test.c

  NFFT-based polar FFT and inverse.

7.28 applications/quadratureS2/ Directory Reference

Files

- file **applications/quadratureS2/doxygen.c**
- file **quadratureS2.c**

7.29 applications/radon/ Directory Reference

Files

- file inverse_radon.c

  NFFT-based discrete inverse Radon transform.

- file radon.c

  NFFT-based discrete Radon transform.

7.30 examples/nnfft/simple_test/ Directory Reference

Files

- file **nnfft/simple_test/simple_test.c**

7.31 kernel/solver/ Directory Reference

Files

- file **solver.c**

7.32 examples/solver/ Directory Reference

Files

- file **examples/solver/doxygen.c**
- file **glacier.c**
- file **solver/simple_test.c**

7.33 util/ Directory Reference

Files

- file **util.c**

Chapter 8

NFFT Data Structure Documentation

8.1 fastsum_plan_ Struct Reference

plan for fast summation algorithm

`#include <fastsum.h>`

Data Fields

- int d
  api number of dimensions
- int N_total
  number of source knots
- int M_total
  number of target knots
- complex * alpha
  source coefficients
- complex * f
  target evaluations
- double * x
  source knots in d-ball with radius 1/4-eps_b/2
- double * y
  target knots in d-ball with radius 1/4-eps_b/2
- complex(* kernel)(double, int, const double *)
  kernel function
- double * kernel_param
  parameters for kernel function

- unsigned flags
  flags precomp.
- complex * pre_K
  DS_PRE - direct summation precomputed K(x_j-y_l).
- int n
  FS__ - fast summation expansion degree.
- fftw_complex * b
  expansion coefficients
- int p
  degree of smoothness of regularization
- double eps_I
  inner boundary
- double eps_B
  outer boundary
- nfft_plan mv1
  source nfft plan
- nfft_plan mv2
  target nfft plan
- int Ad
  near field number of spline knots for nearfield computation of regularized kernel
- double * Add
  spline values
- fftw_plan fft_plan

8.1.1 Detailed Description

plan for fast summation algorithm

Definition at line 34 of file fastsum.h.

8.1.2 Field Documentation

8.1.2.1 unsigned fastsum_plan_::flags

flags precomp.

and approx.type

Definition at line 52 of file fastsum.h.

Referenced by fastsum_finalize(), fastsum_init_guru(), fastsum_precompute(), and fastsum_trafo().

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

- fastsum.h

8.2 fgt_plan Struct Reference

Structure for the Gauss transform.

Data Fields

- int N
  number of source nodes
- int M
  number of target nodes
- double complex * alpha
  source coefficients
- double complex * f
  target evaluations
- unsigned flags
  flags for precomputation and approximation type
- double complex sigma
  parameter of the Gaussian
- double * x
  source nodes in $[-1/4, 1/4]$
- double * y
  target nodes in $[-1/4, 1/4]$
- double complex * pre_cexp
  precomputed values for dgt
- int n
  expansion degree
- double p
  period, at least 1
- double complex * b
  expansion coefficients
- nfft_plan * nplan1
  source nfft plan
- nfft_plan * nplan2
  target nfft plan

8.2.1 Detailed Description

Structure for the Gauss transform.

Definition at line 51 of file fastgauss.c.

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

- fastgauss.c

8.3 fpt_data_ Struct Reference

Holds data for a single cascade summation.

Data Fields

- fpt_step ** steps

  The cascade summation steps.

- int k_start
- double * alphaN
- double * betaN
- double * gammaN
- double alpha_0
- double beta_0
- double gamma_m1
- double * alpha
- double * beta
- double * gamma

8.3.1 Detailed Description

Holds data for a single cascade summation.

Definition at line 62 of file fpt.c.

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

- fpt.c

8.4 fpt_set_s_ Struct Reference

Holds data for a set of cascade summations.

Data Fields

- 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.
- int * lengths
  Transform lengths for fftw library.
- double * xc_slow

8.4.1 Detailed Description

Holds data for a set of cascade summations.

Definition at line 81 of file fpt.c.

8.4.2 Field Documentation

8.4.2.1 int fpt_set_s_::N

The transform length.

Must be a power of two.

Definition at line 85 of file fpt.c.

Referenced by fpt_finalize(), fpt_init(), and fpt_precompute().

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

- fpt.c

8.5 fpt_step_ Struct Reference

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

Data Fields

- bool stable

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

- int N_stab

  TODO Add comment here.

- int t_stab

  TODO Add comment here.

- double ** a11
- double ** a12
- double ** a21
- double ** a22

  The matrix components.

- double * gamma

8.5.1 Detailed Description

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

Definition at line 48 of file fpt.c.

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

- fpt.c

8.6 imri_inh_2d1d_plan Struct Reference

Structure for an inverse transform plan.

`#include <nfft3.h>`

Data Fields

- mri_inh_2d1d_plan * mv
  matrix vector multiplication
- unsigned flags
  iteration type
- double * w
  weighting factors
- double * w_hat
  damping factors
- double complex * y
  right hand side, samples
- double complex * f_hat_iter
  iterative solution
- double complex * r_iter
  iterated residual vector
- double complex * z_hat_iter
  residual of normal equation of \ first kind
- double complex * p_hat_iter
  search direction
- double complex * v_iter
  residual vector update
- double alpha_iter
  step size for search direction
- double beta_iter
  step size for search correction
- double dot_r_iter
  weighted dotproduct of r_iter
- double dot_r_iter_old
  previous dot_r_iter

- double dot_z_hat_iter
  weighted dotproduct of \ z_hat_iter
- double dot_z_hat_iter_old
  previous dot_z_hat_iter
- double dot_p_hat_iter
  weighted dotproduct of \ p_hat_iter
- double dot_v_iter
  weighted dotproduct of v_iter

8.6.1 Detailed Description

Structure for an inverse transform plan.

Definition at line 2343 of file nfft3.h.

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

- nfft3.h

8.7 imri_inh_3d_plan Struct Reference

Structure for an inverse transform plan.

`#include <nfft3.h>`

Data Fields

- mri_inh_3d_plan * mv
  matrix vector multiplication
- unsigned flags
  iteration type
- double * w
  weighting factors
- double * w_hat
  damping factors
- double complex * y
  right hand side, samples
- double complex * f_hat_iter
  iterative solution
- double complex * r_iter
  iterated residual vector
- double complex * z_hat_iter
  residual of normal equation of \ first kind
- double complex * p_hat_iter
  search direction
- double complex * v_iter
  residual vector update
- double alpha_iter
  step size for search direction
- double beta_iter
  step size for search correction
- double dot_r_iter
  weighted dotproduct of r_iter
- double dot_r_iter_old
  previous dot_r_iter

- double dot_z_hat_iter
  weighted dotproduct of \ z_hat_iter
- double dot_z_hat_iter_old
  previous dot_z_hat_iter
- double dot_p_hat_iter
  weighted dotproduct of \ p_hat_iter
- double dot_v_iter
  weighted dotproduct of v_iter

8.7.1 Detailed Description

Structure for an inverse transform plan.

Definition at line 2347 of file nfft3.h.

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

- nfft3.h

8.8 infct_plan Struct Reference

Structure for an inverse transform plan.

`#include <nfft3.h>`

Data Fields

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

- double dot_z_hat_iter
  weighted dotproduct of \ z_hat_iter
- double dot_z_hat_iter_old
  previous dot_z_hat_iter
- double dot_p_hat_iter
  weighted dotproduct of \ p_hat_iter
- double dot_v_iter
  weighted dotproduct of v_iter

8.8.1 Detailed Description

Structure for an inverse transform plan.

Definition at line 2331 of file nfft3.h.

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

- nfft3.h

8.9 infft_plan Struct Reference

Structure for an inverse transform plan.

`#include <nfft3.h>`

Data Fields

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

- double dot_z_hat_iter
  weighted dotproduct of \ z_hat_iter
- double dot_z_hat_iter_old
  previous dot_z_hat_iter
- double dot_p_hat_iter
  weighted dotproduct of \ p_hat_iter
- double dot_v_iter
  weighted dotproduct of v_iter

8.9.1 Detailed Description

Structure for an inverse transform plan.

Definition at line 2327 of file nfft3.h.

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

- nfft3.h

8.10 infsft_plan Struct Reference

Structure for an inverse transform plan.

`#include <nfft3.h>`

Data Fields

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

- double dot_z_hat_iter
  weighted dotproduct of \ z_hat_iter
- double dot_z_hat_iter_old
  previous dot_z_hat_iter
- double dot_p_hat_iter
  weighted dotproduct of \ p_hat_iter
- double dot_v_iter
  weighted dotproduct of v_iter

8.10.1 Detailed Description

Structure for an inverse transform plan.

Definition at line 2351 of file nfft3.h.

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

- nfft3.h

8.11 infst_plan Struct Reference

Structure for an inverse transform plan.

`#include <nfft3.h>`

Data Fields

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

- double dot_z_hat_iter
  weighted dotproduct of \ z_hat_iter
- double dot_z_hat_iter_old
  previous dot_z_hat_iter
- double dot_p_hat_iter
  weighted dotproduct of \ p_hat_iter
- double dot_v_iter
  weighted dotproduct of v_iter

8.11.1 Detailed Description

Structure for an inverse transform plan.

Definition at line 2335 of file nfft3.h.

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

- nfft3.h

8.12 innfft_plan Struct Reference

Structure for an inverse transform plan.

`#include <nfft3.h>`

Data Fields

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

- double dot_z_hat_iter
  weighted dotproduct of \ z_hat_iter
- double dot_z_hat_iter_old
  previous dot_z_hat_iter
- double dot_p_hat_iter
  weighted dotproduct of \ p_hat_iter
- double dot_v_iter
  weighted dotproduct of v_iter

8.12.1 Detailed Description

Structure for an inverse transform plan.

Definition at line 2339 of file nfft3.h.

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

- nfft3.h

8.13 mri_inh_2d1d_plan Struct Reference

The structure for the transform plan.

`#include <nfft3.h>`

Data Fields

- int N_total

  Total number of Fourier \ coefficients.

- int M_total

  Total number of samples.

- double complex * f_hat

  Vector of Fourier coefficients, \ size is N_total float_types.

- double complex * f

  Vector of samples, \ size is M_total float types.

- nfft_plan plan
- int N3
- double sigma3
- double * t
- double * w

8.13.1 Detailed Description

The structure for the transform plan.

Definition at line 1310 of file nfft3.h.

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

- nfft3.h

8.14 mri_inh_3d_plan Struct Reference

The structure for the transform plan.

`#include <nfft3.h>`

Data Fields

- int N_total

  Total number of Fourier \ coefficients.

- int M_total

  Total number of samples.

- double complex * f_hat

  Vector of Fourier coefficients, \ size is N_total float_types.

- double complex * f

  Vector of samples, \ size is M_total float types.

- nfft_plan plan
- int N3
- double sigma3
- double * t
- double * w

8.14.1 Detailed Description

The structure for the transform plan.

Definition at line 1329 of file nfft3.h.

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

- nfft3.h

8.15 nfct_plan Struct Reference

Structure for a transform plan.

`#include <nfft3.h>`

Data Fields

- int N_total
  Total number of Fourier \ coefficients.
- int M_total
  Total number of samples.
- double * f_hat
  Vector of Fourier coefficients, \ size is N_total float_types.
- double * f
  Vector of samples, \ size is M_total float types.
- int d
  dimension, rank
- int * N
  cut-off-frequencies (kernel)
- int * n
  length of dct-i
- double * sigma
  oversampling-factor
- int m
  cut-off parameter in time-domain
- double nfct_full_psi_eps
- double * b
  shape parameters
- unsigned nfct_flags
  flags for precomputation, malloc
- unsigned fftw_flags
  flags for the fftw
- double * x
  nodes (in time/spatial domain)
- double MEASURE_TIME_t [3]
  measured time for each step

- fftw_plan my_fftw_r2r_plan
  internal fftw_plan
- fftw_r2r_kind * r2r_kind
  r2r transform type (dct-i)
- double ** c_phi_inv
  precomputed data, matrix D
- double * 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
- double * g
- double * g_hat
- double * g1
  input of fftw
- double * g2
  output of fftw
- double * spline_coeffs
  input for de Boor algorithm, if B_SPLINE or SINC_2m is defined

8.15.1 Detailed Description

Structure for a transform plan.

Definition at line 490 of file nfft3.h.

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

- nfft3.h

8.16 nfft_plan Struct Reference

Structure for a NFFT plan.

`#include <nfft3.h>`

Data Fields

- int N_total

  Total number of Fourier \ coefficients.

- int M_total

  Total number of samples.

- double complex * f_hat

  Vector of Fourier coefficients, \ size is N_total float_types.

- double complex * f

  Vector of samples, \ size is M_total float types.

- int d

  Dimension, rank.

- int * N

  Multi bandwidth.

- double * sigma

  Oversampling-factor.

- int * n

  *FFTW length, equal to sigma*N, default is the power of 2 such that* $2 \leq \sigma < 4$.

- int n_total

  Total size of FFTW.

- int m

  Cut-off parameter of the window function, default value is 6 (KAISER_BESSEL), 9 (SINC_POWER), 11 (B_SPLINE), 12 (GAUSSIAN).

- double * b

  Shape parameter of the window function.

- int K

  Number of equispaced samples of the window function for PRE_LIN_PSI.

- unsigned nfft_flags

  Flags for precomputation, (de)allocation, and FFTW usage, default setting is PRE_PHI_HUT| PRE_PSI| MALLOC_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.

- double * x
  Nodes in time/spatial domain, size is dM doubles.
- double MEASURE_TIME_t [3]
  Measured time for each step if MEASURE_TIME is set.
- fftw_plan my_fftw_plan1
  Forward FFTW plan.
- fftw_plan my_fftw_plan2
  Backward FFTW plan.
- double ** c_phi_inv
  Precomputed data for the diagonal matrix D , size is $N_0 + \ldots + N_{d-1}$ doubles.
- double * psi
  Precomputed data for the sparse matrix B , size depends on precomputation scheme.
- int * psi_index_g
  Indices in source/target vector for PRE_FULL_PSI.
- int * psi_index_f
  Indices in source/target vector for PRE_FULL_PSI.
- double complex * g
  Oversampled vector of samples, size is n_total double complex.
- double complex * g_hat
  Zero-padded vector of Fourier coefficients, size is n_total double complex.
- double complex * g1
  Input of fftw.
- double complex * g2
  Output of fftw.
- double * spline_coeffs
  Input for de Boor algorithm if B_SPLINE or SINC_POWER is defined.

8.16.1 Detailed Description

Structure for a NFFT plan.

Definition at line 234 of file nfft3.h.

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

- nfft3.h

8.17 nfsft_plan Struct Reference

Structure for a NFSFT transform plan.

`#include <nfft3.h>`

Data Fields

- int N_total

  Inherited public members Total number of Fourier \ coefficients.

- int M_total

  Total number of samples.

- double complex * f_hat

  Vector of Fourier coefficients, \ size is N_total float_types.

- double complex * f

  Vector of samples, \ size is M_total float types.

- int N

  the bandwidth N

- double * 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 next greater power of two with * respect to N the logaritm of NPT with * respect to the basis 2*

- unsigned int flags

  the planner flags

- nfft_plan plan_nfft

  the internal NFFT plan

- double complex * f_hat_intern

  *Internally used pointer to * spherical Fourier coefficients.*

8.17.1 Detailed Description

Structure for a NFSFT transform plan.

Definition at line 1913 of file nfft3.h.

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

- nfft3.h

8.18 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

  The maximum bandwidth /f$N_{{max}} {N}_0/f$.

- int T_MAX

  The logarithm /f$t = N_{{max}}/f$ of the maximum bandwidth.

- double * alpha

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

- double * beta

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

- double * gamma

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

- double threshold

  The threshold /f$/f$.

- fpt_set set

  Structure for discrete polynomial transform *(*DPT*).*

8.18.1 Detailed Description

Wisdom structure.

Definition at line 40 of file api.h.

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

- api.h

8.19 nfst_plan Struct Reference

Structure for a transform plan.

`#include <nfft3.h>`

Data Fields

- int N_total
  Total number of Fourier \ coefficients.
- int M_total
  Total number of samples.
- double * f_hat
  Vector of Fourier coefficients, \ size is N_total float_types.
- double * f
  Vector of samples, \ size is M_total float types.
- int d
  dimension, rank
- int * N
  bandwidth
- int * n
  length of dst-1
- double * sigma
  oversampling-factor
- int m
  cut-off parameter in time-domain
- double nfst_full_psi_eps
- double * b
  shape parameters
- unsigned nfst_flags
  flags for precomputation, malloc
- unsigned fftw_flags
  flags for the fftw
- double * x
  nodes (in time/spatial domain)
- double MEASURE_TIME_t [3]
  measured time for each step

- fftw_plan my_fftw_r2r_plan
  internal fftw_plan forward
- fftw_r2r_kind * r2r_kind
  r2r transform type (dct-i)
- double ** c_phi_inv
  precomputed data, matrix D
- double * 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
- double * g
- double * g_hat
- double * g1
  input of fftw
- double * g2
  output of fftw
- double * spline_coeffs
  input for de Boor algorithm, if B_SPLINE or SINC_2m is defined

8.19.1 Detailed Description

Structure for a transform plan.

Definition at line 703 of file nfft3.h.

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

- nfft3.h

8.20 nnfft_plan Struct Reference

Structure for a transform plan.

`#include <nfft3.h>`

Data Fields

- int N_total
  Total number of Fourier \ coefficients.
- int M_total
  Total number of samples.
- double complex * f_hat
  Vector of Fourier coefficients, \ size is N_total float_types.
- double complex * f
  Vector of samples, \ size is M_total float types.
- int d
  dimension, rank
- double * sigma
  oversampling-factor
- double * 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
- double * b
  shape parameters
- int K
  number of precomp.
- int aN1_total
  aN1_total=aN1[0] .*

- nfft_plan * direct_plan
  plan for the nfft
- unsigned nnfft_flags
  flags for precomputation, malloc
- int * n
  n=N1, for the window function
- double * x
  nodes (in time/spatial domain)
- double * v
  nodes (in fourier domain)
- double * c_phi_inv
  precomputed data, matrix D
- double * 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
- double complex * F
- double * spline_coeffs
  input for de Boor algorithm, if B_SPLINE or SINC_2m is defined

8.20.1 Detailed Description

Structure for a transform plan.

Definition at line 961 of file nfft3.h.

8.20.2 Field Documentation

8.20.2.1 int nnfft_plan::K

number of precomp.

uniform psi

Definition at line 977 of file nfft3.h.

Referenced by nnfft_precompute_lin_psi().

8.20.2.2 int nnfft_plan::aN1_total

aN1_total=aN1[0]∗ .

.. ∗aN1[d-1]

Definition at line 979 of file nfft3.h.

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

- nfft3.h

8.21 nsfft_plan Struct Reference

Structure for a NFFT plan.

`#include <nfft3.h>`

Data Fields

- int N_total
  Total number of Fourier \ coefficients.
- int M_total
  Total number of samples.
- double complex * f_hat
  Vector of Fourier coefficients, \ size is N_total float_types.
- double complex * f
  Vector of samples, \ size is M_total float types.
- int d
  dimension, rank; d=2,3
- int J
  problem size, i.e.
- 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
- nfft_plan * act_nfft_plan
  current nfft block
- nfft_plan * center_nfft_plan
  central nfft block
- fftw_plan * set_fftw_plan1
  fftw plan for the nfft blocks
- fftw_plan * set_fftw_plan2
  fftw plan for the nfft blocks

- nfft_plan * set_nfft_plan_1d
  nfft plans for short nffts
- nfft_plan * set_nfft_plan_2d
  nfft plans for short nffts
- double * x_transposed
  coordinate exchanged nodes, d=2
- double * x_102
- double * x_201
- double * x_120
- double * x_021
  coordinate exchanged nodes, d=3

8.21.1 Detailed Description

Structure for a NFFT plan.

Definition at line 1171 of file nfft3.h.

8.21.2 Field Documentation

8.21.2.1 int nsfft_plan::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))

Definition at line 1179 of file nfft3.h.

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

- nfft3.h

8.22 taylor_plan Struct Reference

Data Fields

- nfft_plan p
  used for fftw and data
- int * idx0
  index of next neighbour of x_j on the oversampled regular grid
- double * deltax0
  distance to the grid point

8.22.1 Detailed Description

Definition at line 17 of file taylor_nfft.c.

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

- taylor_nfft.c

8.23 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
- double * spline_coeffs

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

8.23.1 Detailed Description

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

Definition at line 14 of file mri.c.

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

- mri.c

Chapter 9

NFFT File Documentation

9.1 api.h File Reference

Header file with internal API of the NFSFT module.

```
#include "config.h"
#include "nfft3.h"
#include <fftw3.h>
```

Data Structures

- struct nfsft_wisdom
  Wisdom structure.

Defines

- #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 }

9.1.1 Detailed Description

Header file with internal API of the NFSFT module.

Author:
Jens Keiner

Definition in file api.h.

9.2 fastsum.c File Reference

Fast NFFT-based summation algorithm.

```
#include <stdlib.h>
#include <complex.h>
#include <math.h>
#include "util.h"
#include "nfft3.h"
#include "fastsum.h"
```

Functions

- double fak (int n)
  factorial
- double binom (int n, int m)
  binomial coefficient
- double BasisPoly (int m, int r, double xx)
  basis polynomial for regularized kernel
- double regkern (complex(∗kernel)(double, int, const double ∗), double xx, int p, const double ∗param, double a, double b)
  regularized kernel with K_I arbitrary and K_B smooth to zero
- double regkern1 (complex(∗kernel)(double, int, const double ∗), double xx, int p, const double ∗param, double a, double b)
  regularized kernel with K_I arbitrary and K_B periodized (used in 1D)
- double regkern2 (complex(∗kernel)(double, int, const double ∗), double xx, int p, const double ∗param, double a, double b)
  regularized kernel for even kernels with K_I even and K_B mirrored
- double regkern3 (complex(∗kernel)(double, int, const double ∗), double xx, int p, const double ∗param, double a, double b)
  regularized kernel for even kernels with K_I even and K_B mirrored smooth to K(1/2) (used in dD, d>1)
- double kubintkern (double x, double ∗Add, int Ad, double a)
  cubic spline interpolation in near field with even kernels
- double kubintkern1 (double x, double ∗Add, int Ad, double a)
  cubic spline interpolation in near field with arbitrary kernels
- void quicksort (int d, int t, double ∗x, complex ∗alpha, int N)
  quicksort algorithm for source knots and associated coefficients
- void BuildTree (int d, int t, double ∗x, complex ∗alpha, int N)

recursive sort of source knots dimension by dimension to get tree structure

- complex SearchTree (int d, int t, double ∗x, complex ∗alpha, double ∗xmin, double ∗xmax, int N, complex(∗kernel)(double, int, const double ∗), const double ∗param, int Ad, double ∗Add, int p, unsigned flags)

  fast search in tree of source knots for near field computation

- void fastsum_init_guru (fastsum_plan ∗ths, int d, int N_total, int M_total, complex(∗kernel)(), double ∗param, unsigned flags, int nn, int m, int p, double eps_I, double eps_B)

  initialization 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 (fastsum_plan ∗ths)

  precomputation for fastsum

- void fastsum_trafo (fastsum_plan ∗ths)

  fast NFFT-based summation

9.2.1 Detailed Description

Fast NFFT-based summation algorithm.

Author:
Markus Fenn

Date:
2003-2006

Definition in file fastsum.c.

9.3 fastsum.h File Reference

Header file for the fast NFFT-based summation algorithm.

```
#include <complex.h>
#include "util.h"
#include "nfft3.h"
```

Data Structures

- struct fastsum_plan_

  plan for fast summation algorithm

Defines

- #define EXACT_NEARFIELD (1U<< 0)

  Constant symbols.

Typedefs

- typedef 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, complex(*kernel)(), double *param, unsigned flags, int nn, int m, int p, double eps_I, double eps_B)

  initialization 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 (fastsum_plan *ths)

  precomputation for fastsum

- void fastsum_trafo (fastsum_plan *ths)

  fast NFFT-based summation

9.3.1 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

Definition in file fastsum.h.

9.4 fastsum_matlab.c File Reference

Simple test program for the fast NFFT-based summation algorithm, called by fastsum.m.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <complex.h>
#include <math.h>
#include "fastsum.h"
#include "kernels.h"
```

Functions

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

9.4.1 Detailed Description

Simple test program for the fast NFFT-based summation algorithm, called by fastsum.m.

Author:
Markus Fenn

Date:
2006

Definition in file fastsum_matlab.c.

9.5 fastsum_test.c File Reference

Simple test program for the fast NFFT-based summation algorithm.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <complex.h>
#include <math.h>
#include "fastsum.h"
#include "kernels.h"
```

Functions

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

9.5.1 Detailed Description

Simple test program for the fast NFFT-based summation algorithm.

Author:
Markus Fenn

Date:
2006

Definition in file fastsum_test.c.

9.6 flags.c File Reference

Testing the nfft.

```
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <stdlib.h>
#include "util.h"
#include "nfft3.h"
#include "options.h"
```

Functions

- void flags_cp (nfft_plan *dst, nfft_plan *src)
- void time_accuracy (int d, int N, int M, int n, int m, unsigned test_ndft, unsigned test_pre_full_psi)
- 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

9.6.1 Detailed Description

Testing the nfft.

Author:
Stefan Kunis

References: Time and Memory Requirements of the Nonequispaced FFT

Definition in file flags.c.

9.6.2 Function Documentation

9.6.2.1 void time_accuracy (int *d*, int *N*, int *M*, int *n*, int *m*, unsigned *test_ndft*, unsigned *test_pre_full_psi*)

output vector ndft

init pseudo random nodes

init pseudo random Fourier coefficients

NDFT

NFFTs

finalise

Definition at line 46 of file flags.c.

References nfft_plan::d, nfft_plan::f, nfft_plan::f_hat, FFT_OUT_OF_PLACE, FFTW_INIT, FG_PSI, flags_cp(), nfft_plan::M_total, MALLOC_F, MALLOC_F_HAT, MALLOC_X, nfft_plan::MEASURE_-TIME_t, nfft_plan::N_total, ndft_trafo(), nfft_error_l_2_complex(), nfft_finalize(), nfft_init_guru(), nfft_-precompute_one_psi(), nfft_second(), NFFT_SWAP_complex, nfft_trafo(), nfft_vrand_shifted_unit_-double(), nfft_vrand_unit_complex(), PRE_FG_PSI, PRE_FULL_PSI, PRE_LIN_PSI, PRE_PHI_HUT, PRE_PSI, test_fftw, test_fg, and nfft_plan::x.

Referenced by main().

9.6.2.2 void accuracy_pre_lin_psi (int *d*, int *N*, int *M*, int *n*, int *m*, int *K*)

output vector ndft

realloc psi

precomputation can be done before the nodes are initialised

init pseudo random nodes

init pseudo random Fourier coefficients

compute exact result

NFFT

finalise

Definition at line 199 of file flags.c.

References nfft_plan::d, nfft_plan::f, nfft_plan::f_hat, FFT_OUT_OF_PLACE, FFTW_INIT, nfft_-plan::K, nfft_plan::M_total, MALLOC_F, MALLOC_F_HAT, MALLOC_X, nfft_plan::N_total, ndft_-trafo(), nfft_error_l_2_complex(), nfft_finalize(), nfft_init_guru(), nfft_precompute_one_psi(), NFFT_-SWAP_complex, nfft_trafo(), nfft_vrand_shifted_unit_double(), nfft_vrand_unit_complex(), PRE_LIN_-PSI, PRE_PHI_HUT, nfft_plan::psi, and nfft_plan::x.

Referenced by main().

9.7 fpt.c File Reference

Implementation file for the FPT module.

```
#include <math.h>
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>
#include "nfft3.h"
#include "util.h"
```

Data Structures

- struct fpt_step_

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

- struct fpt_data_

  Holds data for a single cascade summation.

- struct fpt_set_s_

  Holds data for a set of cascade summations.

Defines

- #define K_START_TILDE(x, y) (NFFT_MAX(NFFT_MIN(x,y-2),0))

  Computes the minimum degree at the top of a cascade.

- #define K_END_TILDE(x, y) NFFT_MIN(x,y-1)

  Computes the maximum degree at the top of a cascade.

- #define FIRST_L(x, y) ((int)floor((x)/(double)y))

  Computes the index of the first block of four functions in a cascade level.

- #define LAST_L(x, y) ((int)ceil(((x)+1)/(double)y)-1)

  Computes the index of the last block of four functions in a cascade level.

- #define N_TILDE(y) (y-1)
- #define IS_SYMMETRIC(x, y, z) (x >= ((y-1.0)/z))
- #define ABUVXPWY_SYMMETRIC(NAME, S1, S2)
- #define ABUVXPWY_SYMMETRIC_1(NAME, S1)
- #define ABUVXPWY_SYMMETRIC_2(NAME, S1)
- #define AUVXPWY_SYMMETRIC(NAME, S1, S2)
- #define FPT_DO_STEP(NAME, M1_FUNCTION, M2_FUNCTION)
- #define FPT_DO_STEP_TRANSPOSED(NAME, M1_FUNCTION, M2_FUNCTION)

Typedefs

- typedef fpt_step_ fpt_step

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

- typedef fpt_data_ fpt_data

  Holds data for a single cascade summation.

- typedef fpt_set_s_ fpt_set_s

  Holds data for a set of cascade summations.

Functions

- void abuvxpwy (double a, double b, double complex *u, double complex *x, double *v, double complex *y, double *w, int n)
- void abuvxpwy_symmetric1 (double a, double b, double complex *u, double complex *x, double *v, double complex *y, double *w, int n)
- void abuvxpwy_symmetric2 (double a, double b, double complex *u, double complex *x, double *v, double complex *y, double *w, int n)
- void abuvxpwy_symmetric1_1 (double a, double b, double complex *u, double complex *x, double *v, double complex *y, double *w, int n)
- void abuvxpwy_symmetric1_2 (double a, double b, double complex *u, double complex *x, double *v, double complex *y, double *w, int n)
- void abuvxpwy_symmetric2_1 (double a, double b, double complex *u, double complex *x, double *v, double complex *y, double *w, int n)
- void abuvxpwy_symmetric2_2 (double a, double b, double complex *u, double complex *x, double *v, double complex *y, double *w, int n)
- void auvxpwy (double a, double complex *u, double complex *x, double *v, double complex *y, double *w, int n)
- void auvxpwy_symmetric (double a, double complex *u, double complex *x, double *v, double complex *y, double *w, int n)
- void fpt_do_step (double complex *a, double complex *b, double *a11, double *a12, double *a21, double *a22, double gamma, int tau, fpt_set set)
- void fpt_do_step_symmetric (double complex *a, double complex *b, double *a11, double *a12, double *a21, double *a22, double gamma, int tau, fpt_set set)
- void fpt_do_step_symmetric_u (double complex *a, double complex *b, double *a11, double *a12, double *a21, double *a22, double gamma, int tau, fpt_set set)
- void fpt_do_step_symmetric_l (double complex *a, double complex *b, double *a11, double *a12, double *a21, double *a22, double gamma, int tau, fpt_set set)
- void fpt_do_step_transposed (double complex *a, double complex *b, double *a11, double *a12, double *a21, double *a22, double gamma, int tau, fpt_set set)
- void fpt_do_step_transposed_symmetric (double complex *a, double complex *b, double *a11, double *a12, double *a21, double *a22, double gamma, int tau, fpt_set set)
- void fpt_do_step_transposed_symmetric_u (double complex *a, double complex *b, double *a11, double *a12, double *a21, double *a22, double gamma, int tau, fpt_set set)
- void fpt_do_step_transposed_symmetric_l (double complex *a, double complex *b, double *a11, double *a12, double *a21, double *a22, double gamma, int tau, fpt_set set)
- void eval_clenshaw (const double *x, double *y, int size, int k, const double *alpha, const double *beta, const double *gamma)

- int eval_clenshaw_thresh (const double *x, double *y, int size, int k, const double *alpha, const double *beta, const double *gamma, const double threshold)
- void eval_sum_clenshaw (int N, int M, double complex *a, double *x, double complex *y, double complex *temp, double *alpha, double *beta, double *gamma, double lambda)

  Clenshaw algorithm.

- 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 *gamma, double lambda)

  Clenshaw algorithm.

- 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.

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

  Computes the data required for a single DPT transform.

- void dpt_trafo (fpt_set set, const int m, const double complex *x, double complex *y, const int k_end, const unsigned int flags)

  Computes a single DPT transform.

- void fpt_trafo (fpt_set set, const int m, const double complex *x, double complex *y, const int k_end, const unsigned int flags)

  Computes a single DPT transform.

- void dpt_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.

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

  Computes a single DPT transform.

- void fpt_finalize (fpt_set set)

9.7.1 Detailed Description

Implementation file for the FPT module.

Author:
Jens Keiner

Definition in file fpt.c.

9.7.2 Define Documentation

9.7.2.1 #define ABUVXPWY_SYMMETRIC(NAME, S1, S2)

Value:

```
inline void NAME(double a, double b, double complex* u, double complex* x, double* v, \
  double complex* y, double* w, int n) \
{ \
  int l; \
  double complex *u_ptr, *x_ptr, *y_ptr; \
  double *v_ptr, *w_ptr; \
  \
  u_ptr = u; \
  x_ptr = x; \
  v_ptr = v; \
  y_ptr = y; \
  w_ptr = w; \
  \
  for (l = 0; l < n/2; l++) \
  { \
    *u_ptr++ = a * (b * (*v_ptr++) * (*x_ptr++) + (*w_ptr++) * (*y_ptr++)); \
  } \
  v_ptr--; \
  w_ptr--; \
  for (l = 0; l < n/2; l++) \
  { \
    *u_ptr++ = a * (b * S1 * (*v_ptr--) * (*x_ptr++) + S2 * (*w_ptr--) * (*y_ptr++)); \
  } \
}
```

Definition at line 133 of file fpt.c.

9.7.2.2 #define ABUVXPWY_SYMMETRIC_1(NAME, S1)

Value:

```
inline void NAME(double a, double b, double complex* u, double complex* x, double* v, \
  double complex* y, double* w, int n) \
{ \
  int l; \
  double complex *u_ptr, *x_ptr, *y_ptr; \
  double *v_ptr, *w_ptr; \
  \
  u_ptr = u; \
  x_ptr = x; \
  v_ptr = v; \
  y_ptr = y; \
  w_ptr = w; \
  \
  for (l = 0; l < n/2; l++) \
  { \
    *u_ptr++ = a * (b * (*v_ptr++) * (*x_ptr++) + (*w_ptr++) * (*y_ptr++)); \
  } \
  v_ptr--; \
  /*w_ptr--;*/ \
  for (l = 0; l < n/2; l++) \
  { \
    *u_ptr++ = a * (b * S1 * (*v_ptr--) * (*x_ptr++) + (*w_ptr++) * (*y_ptr++)); \
  } \
}
```

Definition at line 162 of file fpt.c.

9.7.2.3 #define ABUVXPWY_SYMMETRIC_2(NAME, S1)

Value:

```
inline void NAME(double a, double b, double complex* u, double complex* x, double* v, \
  double complex* y, double* w, int n) \
{ \
  int l; \
  double complex *u_ptr, *x_ptr, *y_ptr; \
  double *v_ptr, *w_ptr; \
  \
  u_ptr = u; \
  x_ptr = x; \
  v_ptr = v; \
  y_ptr = y; \
  w_ptr = w; \
  \
  for (l = 0; l < n/2; l++) \
  { \
    *u_ptr++ = a * (b * (*v_ptr++) * (*x_ptr++) + (*w_ptr++) * (*y_ptr++)); \
  } \
  /*v_ptr--;*/ \
  w_ptr--; \
  for (l = 0; l < n/2; l++) \
  { \
    *u_ptr++ = a * (b * (*v_ptr++) * (*x_ptr++) + S1 * (*w_ptr--) * (*y_ptr++)); \
  } \
}
```

Definition at line 191 of file fpt.c.

9.7.2.4 #define FPT_DO_STEP_TRANSPOSED(NAME, M1_FUNCTION, M2_FUNCTION)

Value:

```
inline void NAME(double complex  *a, double complex *b, double *a11, \
  double *a12, double *a21, double *a22, double gamma, 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,gamma,set->z,a,a11,b,a21,length); \
  M2_FUNCTION(norm,gamma,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 333 of file fpt.c.

9.7.3 Function Documentation

9.7.3.1 void fpt_do_step (double complex * *a*, double complex * *b*, double * *a11*, double * *a12*, double * *a21*, double * *a22*, double *gamma*, int *tau*, fpt_set *set*) [inline]

The length of the coefficient arrays.

Twice the length of the coefficient arrays.

Definition at line 328 of file fpt.c.

Referenced by fpt_trafo().

9.7.3.2 void fpt_do_step_symmetric (double complex * *a*, double complex * *b*, double * *a11*, double * *a12*, double * *a21*, double * *a22*, double *gamma*, int *tau*, fpt_set *set*) [inline]

The length of the coefficient arrays.

Twice the length of the coefficient arrays.

Definition at line 329 of file fpt.c.

Referenced by fpt_trafo().

9.7.3.3 void fpt_do_step_symmetric_u (double complex * *a*, double complex * *b*, double * *a11*, double * *a12*, double * *a21*, double * *a22*, double *gamma*, int *tau*, fpt_set *set*) [inline]

The length of the coefficient arrays.

Twice the length of the coefficient arrays.

Definition at line 330 of file fpt.c.

Referenced by fpt_trafo().

9.7.3.4 void fpt_do_step_symmetric_l (double complex * *a*, double complex * *b*, double * *a11*, double * *a12*, double * *a21*, double * *a22*, double *gamma*, int *tau*, fpt_set *set*) [inline]

The length of the coefficient arrays.

Twice the length of the coefficient arrays.

Definition at line 331 of file fpt.c.

Referenced by fpt_trafo().

9.7.3.5 void fpt_do_step_transposed (double complex * *a*, double complex * *b*, double * *a11*, double * *a12*, double * *a21*, double * *a22*, double *gamma*, int *tau*, fpt_set *set*) [inline]

The length of the coefficient arrays.

Twice the length of the coefficient arrays.

Definition at line 356 of file fpt.c.

Referenced by fpt_transposed().

9.7.3.6 void fpt_do_step_transposed_symmetric (double complex * *a*, double complex * *b*, double * *a11*, double * *a12*, double * *a21*, double * *a22*, double *gamma*, int *tau*, fpt_set *set*) [inline]

The length of the coefficient arrays.

Twice the length of the coefficient arrays.

Definition at line 357 of file fpt.c.

Referenced by fpt_transposed().

9.7.3.7 void fpt_do_step_transposed_symmetric_u (double complex * *a*, double complex * *b*, double * *a11*, double * *a12*, double * *a21*, double * *a22*, double *gamma*, int *tau*, fpt_set *set*) [inline]

The length of the coefficient arrays.

Twice the length of the coefficient arrays.

Definition at line 358 of file fpt.c.

Referenced by fpt_transposed().

9.7.3.8 void fpt_do_step_transposed_symmetric_l (double complex * *a*, double complex * *b*, double * *a11*, double * *a12*, double * *a21*, double * *a22*, double *gamma*, int *tau*, fpt_set *set*) [inline]

The length of the coefficient arrays.

Twice the length of the coefficient arrays.

Definition at line 359 of file fpt.c.

Referenced by fpt_transposed().

9.7.3.9 void eval_sum_clenshaw (int *N*, int *M*, double complex * *a*, double * *x*, double complex * *y*, double complex * *temp*, double * *alpha*, double * *beta*, double * *gamma*, double *lambda*)

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 477 of file fpt.c.

Referenced by dpt_trafo().

9.7.3.10 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 * *gamma*, double *lambda*)

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 543 of file fpt.c.

Referenced by dpt_transposed().

9.8 inverse_radon.c File Reference

NFFT-based discrete inverse Radon transform.

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>
#include "util.h"
#include "nfft3.h"
```

Defines

- #define KERNEL(r) (1.0-fabs((double)(r))/((double)R/2))
  define weights of kernel function for discrete Radon transform

Functions

- int polar_grid (int T, int R, double *x, double *w)
  generates the points x with weights w for the polar grid with T angles and R offsets
- int linogram_grid (int T, int R, double *x, double *w)
  generates the points x with weights w for the linogram grid with T slopes and R offsets
- int Inverse_Radon_trafo (int(*gridfcn)(), int T, int R, double *Rf, int NN, double *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

9.8.1 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

Definition in file inverse_radon.c.

9.8.2 Function Documentation

9.8.2.1 int Inverse_Radon_trafo (int(∗)() *gridfcn*, int *T*, int *R*, double ∗ *Rf*, int *NN*, double ∗ *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

< index for nodes and freqencies

< plan for the nfft-2D

< plan for the inverse nfft

< variable for the fftw-1Ds

< plan for the fftw-1Ds

< index for directions and offsets

< knots and associated weights

< index for iterations

init two dimensional NFFT plan

init two dimensional infft plan

init nodes and weights of grid

precompute psi, the entries of the matrix B

compute 1D-ffts and init given samples and weights

initialise some guess f_hat_0

solve the system

copy result

finalise the plans and free the variables

Definition at line 91 of file inverse_radon.c.

References CGNR, infft_plan::f_hat_iter, fft(), FFT_OUT_OF_PLACE, FFTW_INIT, infft_before_loop(), infft_finalize(), infft_init_advanced(), infft_loop_one_step(), KERNEL, nfft_plan::M_total, MALLOC_F, MALLOC_F_HAT, MALLOC_X, nfft_plan::N_total, nfft_fftshift_complex(), nfft_finalize(), nfft_plan::nfft_flags, nfft_init_guru(), nfft_precompute_full_psi(), nfft_precompute_lin_psi(), nfft_precompute_psi(), infft_plan::p_hat_iter, PRE_FULL_PSI, PRE_LIN_PSI, PRE_PHI_HUT, PRE_PSI, PRECOMPUTE_WEIGHT, infft_plan::w, nfft_plan::x, and infft_plan::y.

Referenced by main().

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

simple test program for the inverse discrete Radon transform

< grid generating function

< number of directions/offsets

< image size

< number of iterations

load data

inverse Radon transform

write result

free the variables

Definition at line 211 of file inverse_radon.c.

References Inverse_Radon_trafo(), linogram_grid(), and polar_grid().

9.9 kernels.c File Reference

File with predefined kernels for the fast summation algorithm.

```
#include <stdio.h>
#include <complex.h>
#include <math.h>
#include <float.h>
```

Functions

- complex gaussian (double x, int der, const double *param)
- complex multiquadric (double x, int der, const double *param)
- complex inverse_multiquadric (double x, int der, const double *param)
- complex logarithm (double x, int der, const double *param)
- complex thinplate_spline (double x, int der, const double *param)
- complex one_over_square (double x, int der, const double *param)
- complex one_over_modulus (double x, int der, const double *param)
- complex one_over_x (double x, int der, const double *param)
- complex inverse_multiquadric3 (double x, int der, const double *param)
- complex sinc_kernel (double x, int der, const double *param)
- complex cosc (double x, int der, const double *param)
- complex cot (double x, int der, const double *param)

9.9.1 Detailed Description

File with predefined kernels for the fast summation algorithm.

Definition in file kernels.c.

9.10 kernels.h File Reference

Header file with predefined kernels for the fast summation algorithm.

Functions

- complex gaussian (double x, int der, const double *param)
- complex multiquadric (double x, int der, const double *param)
- complex inverse_multiquadric (double x, int der, const double *param)
- complex logarithm (double x, int der, const double *param)
- complex thinplate_spline (double x, int der, const double *param)
- complex one_over_square (double x, int der, const double *param)
- complex one_over_modulus (double x, int der, const double *param)
- complex one_over_x (double x, int der, const double *param)
- complex inverse_multiquadric3 (double x, int der, const double *param)
- complex sinc_kernel (double x, int der, const double *param)
- complex cosc (double x, int der, const double *param)
- complex cot (double x, int der, const double *param)

9.10.1 Detailed Description

Header file with predefined kernels for the fast summation algorithm.

Definition in file kernels.h.

9.11 linogram_fft_test.c File Reference

NFFT-based pseudo-polar FFT and inverse.

```
#include <math.h>
#include <stdlib.h>
#include "util.h"
#include "nfft3.h"
```

Functions

- int linogram_grid (int T, int R, double *x, double *w)
  Generates the points x with weights w for the linogram grid with T slopes and R offsets.
- int linogram_dft (fftw_complex *f_hat, int NN, fftw_complex *f, int T, int R, int m)
  discrete pseudo-polar FFT
- int linogram_fft (fftw_complex *f_hat, int NN, fftw_complex *f, int T, int R, int m)
  NFFT-based pseudo-polar FFT.
- int inverse_linogram_fft (fftw_complex *f, int T, int R, fftw_complex *f_hat, int NN, int max_i, int m)
  NFFT-based inverse pseudo-polar FFT.
- int comparison_fft (FILE *fp, int N, int T, int R)
  Comparison of the FFTW, linogram FFT, and inverse linogram FFT.
- int main (int argc, char **argv)
  test program for various parameters

Variables

- double GLOBAL_elapsed_time

9.11.1 Detailed Description

NFFT-based pseudo-polar FFT and inverse.

Computes the NFFT-based pseudo-polar FFT and its inverse.

Author:
Markus Fenn

Date:
2006

Definition in file linogram_fft_test.c.

9.12 mpolar_fft_test.c File Reference

NFFT-based polar FFT and inverse on modified polar grid.

```
#include <math.h>
#include <stdlib.h>
#include "util.h"
#include "nfft3.h"
```

Functions

- int mpolar_grid (int T, int R, double *x, double *w)
  Generates the points $x_{t,j}$ with weights $w_{t,j}$ for the modified polar grid with T angles and R offsets.
- int mpolar_dft (fftw_complex *f_hat, int NN, fftw_complex *f, int T, int R, int m)
  discrete mpolar FFT
- int mpolar_fft (fftw_complex *f_hat, int NN, fftw_complex *f, int T, int R, int m)
  NFFT-based mpolar FFT.
- int inverse_mpolar_fft (fftw_complex *f, int T, int R, fftw_complex *f_hat, int NN, int max_i, int m)
  inverse NFFT-based mpolar FFT
- int comparison_fft (FILE *fp, int N, int T, int R)
  Comparison of the FFTW, mpolar FFT, and inverse mpolar FFT.
- int main (int argc, char **argv)
  test program for various parameters

Variables

- double GLOBAL_elapsed_time

9.12.1 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

Definition in file mpolar_fft_test.c.

9.13 ndft_fast.c File Reference

Testing ndft, Horner-like ndft, and fully precomputed ndft.

```
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <stdlib.h>
#include <complex.h>
#include "util.h"
#include "nfft3.h"
```

Functions

- void ndft_horner_trafo (nfft_plan *ths)
- void ndft_pre_full_trafo (nfft_plan *ths, double complex *A)
- void ndft_pre_full_init (nfft_plan *ths, double complex *A)
- void ndft_time (int N, int M, unsigned test_ndft, unsigned test_pre_full)
- int main (int argc, char **argv)

9.13.1 Detailed Description

Testing ndft, Horner-like ndft, and fully precomputed ndft.

Author:
Stefan Kunis

Definition in file ndft_fast.c.

9.13.2 Function Documentation

9.13.2.1 void ndft_time (int *N*, int *M*, unsigned *test_ndft*, unsigned *test_pre_full*)

init pseudo random nodes

init pseudo random Fourier coefficients

NDFT

Horner NDFT

Fully precomputed NDFT

Definition at line 63 of file ndft_fast.c.

References nfft_plan::f_hat, nfft_plan::M_total, nfft_plan::N_total, ndft_horner_trafo(), ndft_pre_full_init(), ndft_pre_full_trafo(), ndft_trafo(), nfft_finalize(), nfft_init_1d(), nfft_second(), nfft_trafo(), nfft_vrand_shifted_unit_double(), nfft_vrand_unit_complex(), and nfft_plan::x.

Referenced by main().

9.14 nfft3.h File Reference

Header file for the nfft3 library.

```
#include <complex.h>
#include <fftw3.h>
```

Data Structures

- struct nfft_plan

  Structure for a NFFT plan.

- struct nfct_plan

  Structure for a transform plan.

- struct nfst_plan

  Structure for a transform plan.

- struct nnfft_plan

  Structure for a transform plan.

- struct nsfft_plan

  Structure for a NFFT plan.

- struct mri_inh_2d1d_plan

  The structure for the transform plan.

- struct mri_inh_3d_plan

  The structure for the transform plan.

- struct nfsft_plan

  Structure for a NFSFT transform plan.

- struct infft_plan

  Structure for an inverse transform plan.

- struct infct_plan

  Structure for an inverse transform plan.

- struct infst_plan

  Structure for an inverse transform plan.

- struct innfft_plan

  Structure for an inverse transform plan.

- struct imri_inh_2d1d_plan

  Structure for an inverse transform plan.

- struct imri_inh_3d_plan

Structure for an inverse transform plan.

- struct infsft_plan

  Structure for an inverse transform plan.

Defines

- #define MACRO_MV_PLAN(float_type)

  Macros for public members inherited by all plan structures. Vector of samples, \ size is M_total float types.

- #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.

- #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.

- #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.

- #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).

- #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.

- #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.

- #define MALLOC_X (1U<< 6)

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

- #define MALLOC_F_HAT (1U<< 7)

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

- #define MALLOC_F (1U<< 8)

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

- #define FFT_OUT_OF_PLACE (1U<< 9)

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

- #define FFTW_INIT (1U<< 10)

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

- #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 $\mathbf{B}$ *).*

- #define MALLOC_V (1U<< 11)

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

- #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 nsdft_trafo and nsdft_adjoint.

- #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*

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

  .

- #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.

- #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.

- #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.*

- #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.*

- #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.*

- #define NFSFT_PRESERVE_F_HAT (1U << 7)

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

- #define NFSFT_PRESERVE_X (1U << 8)

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

- #define NFSFT_PRESERVE_F (1U << 9)

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

- #define NFSFT_DESTROY_F_HAT (1U << 10)

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

- #define NFSFT_DESTROY_X (1U << 11)

  If this flag is set, it is explicitely allowed that during an execution of ndsft_trafo, nfsft_trafo or ndsft_adjoint, nfsft_adjoint the content of `x` *may be changed.*

- #define NFSFT_DESTROY_F (1U << 12)

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

- #define NFSFT_NO_DIRECT_ALGORITHM (1U << 13)

  If this flag is set, the transforms ndsft_trafo and ndsft_adjoint do not work.

- #define NFSFT_NO_FAST_ALGORITHM (1U << 14)

  If this flag is set, the transforms nfsft_trafo and nfsft_adjoint do not work.

- #define NFSFT_ZERO_F_HAT (1U << 16)

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

- #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

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

  .

- #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.

- #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)

TODO Don't use this flag!

- #define LANDWEBER (1U<< 0)

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

- #define STEEPEST_DESCENT (1U<< 1)

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

- #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.

- #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.

- #define NORMS_FOR_LANDWEBER (1U<< 4)

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

- #define PRECOMPUTE_WEIGHT (1U<< 5)

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

- #define PRECOMPUTE_DAMP (1U<< 6)

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

- #define MACRO_SOLVER_PLAN(MV, FLT, FLT_TYPE)

  Complete macro for mangling an inverse transform.

Typedefs

- typedef fpt_set_s_ * fpt_set

  A set of precomputed data for a set of DPT transforms of equal maximum length.

Functions

- void ndft_trafo (nfft_plan *ths)

  Computes a NDFT, see the definition.

- void ndft_adjoint (nfft_plan *ths)

  Computes an adjoint NDFT, see the definition.

- void nfft_trafo (nfft_plan *ths)

  Computes a NFFT, see the definition.

- void nfft_adjoint (nfft_plan *ths)

  Computes an adjoint NFFT, see the definition.

- void nfft_init_1d (nfft_plan *ths, int N1, int M)

  Initialisation of a transform plan, wrapper d=1.

- void nfft_init_2d (nfft_plan *ths, int N1, int N2, int M)

  Initialisation of a transform plan, wrapper d=2.

- void nfft_init_3d (nfft_plan *ths, int N1, int N2, int N3, int M)

  Initialisation of a transform plan, wrapper d=3.

- void nfft_init (nfft_plan *ths, int d, int *N, int M)

  Initialisation of a transform plan, simple.

- void nfft_init_advanced (nfft_plan *ths, int d, int *N, int M, unsigned nfft_flags_on, unsigned nfft_flags_off)

  Initialisation of a transform plan, advanced.

- void nfft_init_guru (nfft_plan *ths, int d, int *N, int M, int *n, int m, unsigned nfft_flags, unsigned fftw_flags)

  Initialisation of a transform plan, guru.

- void nfft_precompute_one_psi (nfft_plan *ths)

  Precomputation for a transform plan.

- void nfft_precompute_full_psi (nfft_plan *ths)

  Superceded by nfft_precompute_one_psi.

- void nfft_precompute_psi (nfft_plan *ths)

  Superceded by nfft_precompute_one_psi.

- void nfft_precompute_lin_psi (nfft_plan *ths)

  Superceded by nfft_precompute_one_psi.

- void nfft_check (nfft_plan *ths)

  Checks a transform plan for frequently used bad parameter.

- void nfft_finalize (nfft_plan *ths)

  Destroys a transform plan.

- void nfct_init_1d (nfct_plan *ths_plan, int N0, int M_total)

  Creates a 1-dimensional transform plan.

- void nfct_init_2d (nfct_plan *ths_plan, int N0, int N1, int M_total)

  Creates a 3-dimensional transform plan.

- void nfct_init_3d (nfct_plan *ths_plan, int N0, int N1, int N2, int M_total)

  Creates a 3-dimensional transform plan.

- void nfct_init (nfct_plan *ths_plan, int d, int *N, int M_total)

  Creates a d-dimensional transform plan.

- void nfct_init_guru (nfct_plan *ths_plan, int d, int *N, int M_total, int *n, int m, unsigned nfct_flags, unsigned fftw_flags)

  Creates a d-dimensional transform plan.

- void nfct_precompute_psi (nfct_plan *ths_plan)

  precomputes the values psi if the PRE_PSI is set the application program has to call this routine after setting the nodes this_plan->x

- void nfct_trafo (nfct_plan *ths_plan)

  executes a NFCT (approximate,fast), computes for $j = 0, ..., M\_total - 1$ $f_j^C(x_j) = \sum_{k \in I_0^{N,d}} \hat{f}_k^C * cos(2\pi k x_j)$

- void ndct_trafo (nfct_plan *ths_plan)

  executes a NDCT (exact,slow), computes for $j = 0, ..., M\_total - 1$ $f_j^C(x_j) = \sum_{k \in I_0^{N,d}} \hat{f}_k^C * cos(2\pi k x_j)$

- void nfct_adjoint (nfct_plan *ths_plan)

  executes a transposed NFCT (approximate,fast), computes for $k \in I_0^{N,d}$ $h^C(k) = \sum_{j \in I_0^{(M\_total,1)}} f_j^C * cos(2\pi k x_j)$

- void ndct_adjoint (nfct_plan *ths_plan)

  executes a direct transposed NDCT (exact,slow), computes for $k \in I_0^{N,d}$ $h^C(k) = \sum_{j \in I_0^{(M\_total,1)}} f_j^C * cos(2\pi k x_j)$

- void nfct_finalize (nfct_plan *ths_plan)

  Destroys a plan.

- double nfct_phi_hut (nfct_plan *ths_plan, int k, int d)

  do some adjustments (N,n) then compute PHI_HUT

- double nfct_phi (nfct_plan *ths_plan, double x, int d)

  do some adjustments (N,n) then compute PHI

- int nfct_fftw_2N (int n)

  returns 2(n-1), fftw related issue

- int nfct_fftw_2N_rev (int n)

  returns 0.5n+1, fftw related issue

- void nfst_init_1d (nfst_plan *ths_plan, int N0, int M_total)

  Creates a 1-dimensional transform plan.

- void nfst_init_2d (nfst_plan *ths_plan, int N0, int N1, int M_total)

  Creates a 3-dimensional transform plan.

- void nfst_init_3d (nfst_plan *ths_plan, int N0, int N1, int N2, int M_total)

  Creates a 3-dimensional transform plan.

- void nfst_init (nfst_plan *ths_plan, int d, int *N, int M_total)

  Creates a d-dimensional transform plan.

- void nfst_init_m (nfst_plan *ths_plan, int d, int *N, int M_total, int m)

  Creates a d-dimensional transform plan with pcific m.

- void nfst_init_guru (nfst_plan *ths_plan, int d, int *N, int M_total, int *n, int m, unsigned nfst_flags, unsigned fftw_flags)

  Creates a d-dimensional transform plan.

- void nfst_precompute_psi (nfst_plan *ths_plan)

  precomputes the values psi if the PRE_PSI is set the application program has to call this routine after setting the nodes this_plan->x

- void nfst_trafo (nfst_plan *ths_plan)

  executes a NFST (approximate,fast), computes for $j = 0, ..., M\_total - 1$ $f_j^S(x_j) = \sum_{k \in I_1^{N,d}} \hat{f}_k^S * sin(2\pi k x_j)$

- void ndst_trafo (nfst_plan *ths_plan)

  executes a NDST (exact,slow), computes for $j = 0, ..., M\_total - 1$ $f_j^S(x_j) = \sum_{k \in I_1^{N,d}} \hat{f}_k^S * sin(2\pi k x_j)$

- void nfst_adjoint (nfst_plan *ths_plan)

  executes a transposed NFST (approximate,fast), computes for $k \in I_1^{N,d}$ $h^S(k) = \sum_{j \in I_0^{M\_total,1}} f_j^S * cos(2\pi k x_j)$

- void ndst_adjoint (nfst_plan *ths_plan)

  executes a direct transposed NDST (exact,slow), computes for $k \in I_1^{N,d}$ $h^S(k) = \sum_{j \in I_0^{M\_total,1}} f_j^S * cos(2\pi k x_j)$

- void nfst_finalize (nfst_plan *ths_plan)

  Destroys a plan.

- void nfst_full_psi (nfst_plan *ths_plan, double eps)

  more memory usage, a bit faster

- double nfst_phi_hut (nfst_plan *ths_plan, int k, int d)

  do some adjustments (N,n) then compute PHI_HUT

- double nfst_phi (nfst_plan *ths_plan, double x, int d)

  do some adjustments (N,n) then compute PHI

- int nfst_fftw_2N (int n)

  returns 2(n+1), fftw related issue

- int nfst_fftw_2N_rev (int n)

  returns 0.5n-1, fftw related issue

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

  Creates a transform plan.

- 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.

- void nndft_trafo (nnfft_plan *ths_plan)

  Executes a direct NNDFT, i.e.

- void nndft_adjoint (nnfft_plan *ths_plan)

  Executes a direct adjoint NNDFT, i.e.

- void nnfft_trafo (nnfft_plan *ths_plan)

  Executes a NNFFT, i.e.

- void nnfft_adjoint (nnfft_plan *ths_plan)

  Executes a adjoint NNFFT, i.e.

- void nnfft_precompute_lin_psi (nnfft_plan *ths_plan)

  Precomputation for a transform plan.

- void nnfft_precompute_psi (nnfft_plan *ths_plan)

  Precomputation for a transform plan.

- void nnfft_precompute_full_psi (nnfft_plan *ths_plan)

  Precomputation for a transform plan.

- void nnfft_precompute_phi_hut (nnfft_plan *ths_plan)

  Precomputation for a transform plan.

- void nnfft_finalize (nnfft_plan *ths_plan)

  Destroys a plan.

- void nsdft_trafo (nsfft_plan *ths)

  Executes an NSDFT, computes for $j = 0, \ldots, M-1$ *:*

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

  .

- void nsdft_adjoint (nsfft_plan *ths)

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

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

  .

- void nsfft_trafo (nsfft_plan *ths)

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

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

  .

- void nsfft_adjoint (nsfft_plan *ths)

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

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

.

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

  Copy coefficients from nsfft plan to a nfft plan.

- void nsfft_init_random_nodes_coeffs (nsfft_plan *ths)

  Initialisation of pseudo random nodes and coefficients.

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

  Initialisation of a transform plan.

- void nsfft_finalize (nsfft_plan *ths)

  Destroys a transform plan.

- void mri_inh_2d1d_trafo (mri_inh_2d1d_plan *ths)

  Executes a mri transformation considering the field inhomogeneity with the 2d1d method, i.e.

- void mri_inh_2d1d_adjoint (mri_inh_2d1d_plan *ths)

  Executes an adjoint mri transformation considering the field inhomogeneity with the 2d1d method, i.e.

- 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.

- void mri_inh_2d1d_finalize (mri_inh_2d1d_plan *ths)

  Destroys a plan.

- void mri_inh_3d_trafo (mri_inh_3d_plan *ths)

  Executes a mri transformation considering the field inhomogeneity with the 3d method, i.e.

- void mri_inh_3d_adjoint (mri_inh_3d_plan *ths)

  Executes an adjoint mri transformation considering the field inhomogeneity with the 3d method, i.e.

- void mri_inh_3d_init_guru (mri_inh_3d_plan *ths, int *N, int M, int *n, int m, double sigma, unsigned nfft_flags, unsigned fftw_flags)
- void mri_inh_3d_finalize (mri_inh_3d_plan *ths)

  Destroys a plan.

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

  Creates a transform plan.

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

  Creates a transform plan.

- void nfsft_init_guru (nfsft_plan *plan, int N, int M, unsigned int nfsft_flags, int nfft_flags, int nfft_cutoff)

  Creates a transform plan.

- 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 $N \in \mathbb{N}_2$.

- void nfsft_forget ()

  Forgets all precomputed data.

- void ndsft_trafo (nfsft_plan *plan)

  Executes a direct NDSFT, i.e.

- void ndsft_adjoint (nfsft_plan *plan)

  Executes a direct adjoint NDSFT, i.e.

- void nfsft_trafo (nfsft_plan *plan)

  Executes a NFSFT, i.e.

- void nfsft_adjoint (nfsft_plan *plan)

  Executes an adjoint NFSFT, i.e.

- void nfsft_finalize (nfsft_plan *plan)

  Destroys a plan.

- void nfsft_precompute_x (nfsft_plan *plan)
- 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.

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

  Computes the data required for a single DPT transform.

- void dpt_trafo (fpt_set set, const int m, const double complex *x, double complex *y, const int k_end, const unsigned int flags)

  Computes a single DPT transform.

- void fpt_trafo (fpt_set set, const int m, const double complex *x, double complex *y, const int k_end, const unsigned int flags)

  Computes a single DPT transform.

- void dpt_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.

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

  Computes a single DPT transform.

- void fpt_finalize (fpt_set set)
- void infft_init (infft_plan *ths, nfft_plan *mv)

  Simple initialisation.

- void infft_init_advanced (infft_plan *ths, nfft_plan *mv, unsigned infft_flags)
  Advanced initialisation.
- void infft_before_loop (infft_plan *ths)
  Setting up residuals before the actual iteration.
- void infft_loop_one_step (infft_plan *ths)
  Doing one step in the iteration.
- void infft_finalize (infft_plan *ths)
  Destroys the plan for the inverse transform.
- void infct_init (infct_plan *ths, nfct_plan *mv)
  Simple initialisation.
- void infct_init_advanced (infct_plan *ths, nfct_plan *mv, unsigned infct_flags)
  Advanced initialisation.
- void infct_before_loop (infct_plan *ths)
  Setting up residuals before the actual iteration.
- void infct_loop_one_step (infct_plan *ths)
  Doing one step in the iteration.
- void infct_finalize (infct_plan *ths)
  Destroys the plan for the inverse transform.
- void infst_init (infst_plan *ths, nfst_plan *mv)
  Simple initialisation.
- void infst_init_advanced (infst_plan *ths, nfst_plan *mv, unsigned infst_flags)
  Advanced initialisation.
- void infst_before_loop (infst_plan *ths)
  Setting up residuals before the actual iteration.
- void infst_loop_one_step (infst_plan *ths)
  Doing one step in the iteration.
- void infst_finalize (infst_plan *ths)
  Destroys the plan for the inverse transform.
- void innfft_init (innfft_plan *ths, nnfft_plan *mv)
  Simple initialisation.
- void innfft_init_advanced (innfft_plan *ths, nnfft_plan *mv, unsigned innfft_flags)
  Advanced initialisation.
- void innfft_before_loop (innfft_plan *ths)
  Setting up residuals before the actual iteration.

- void innfft_loop_one_step (innfft_plan *ths)

  Doing one step in the iteration.

- void innfft_finalize (innfft_plan *ths)

  Destroys the plan for the inverse transform.

- void imri_inh_2d1d_init (imri_inh_2d1d_plan *ths, mri_inh_2d1d_plan *mv)

  Simple initialisation.

- void imri_inh_2d1d_init_advanced (imri_inh_2d1d_plan *ths, mri_inh_2d1d_plan *mv, unsigned imri_inh_2d1d_flags)

  Advanced initialisation.

- void imri_inh_2d1d_before_loop (imri_inh_2d1d_plan *ths)

  Setting up residuals before the actual iteration.

- void imri_inh_2d1d_loop_one_step (imri_inh_2d1d_plan *ths)

  Doing one step in the iteration.

- void imri_inh_2d1d_finalize (imri_inh_2d1d_plan *ths)

  Destroys the plan for the inverse transform.

- void imri_inh_3d_init (imri_inh_3d_plan *ths, mri_inh_3d_plan *mv)

  Simple initialisation.

- void imri_inh_3d_init_advanced (imri_inh_3d_plan *ths, mri_inh_3d_plan *mv, unsigned imri_inh_3d_flags)

  Advanced initialisation.

- void imri_inh_3d_before_loop (imri_inh_3d_plan *ths)

  Setting up residuals before the actual iteration.

- void imri_inh_3d_loop_one_step (imri_inh_3d_plan *ths)

  Doing one step in the iteration.

- void imri_inh_3d_finalize (imri_inh_3d_plan *ths)

  Destroys the plan for the inverse transform.

- void infsft_init (infsft_plan *ths, nfsft_plan *mv)

  Simple initialisation.

- void infsft_init_advanced (infsft_plan *ths, nfsft_plan *mv, unsigned infsft_flags)

  Advanced initialisation.

- void infsft_before_loop (infsft_plan *ths)

  Setting up residuals before the actual iteration.

- void infsft_loop_one_step (infsft_plan *ths)

  Doing one step in the iteration.

- void infsft_finalize (infsft_plan *ths)

  Destroys the plan for the inverse transform.

9.14.1 Detailed Description

Header file for the nfft3 library.

Definition in file nfft3.h.

9.14.2 Define Documentation

9.14.2.1 #define MACRO_MV_PLAN(float_type)

Value:

```
int N_total;                          \
  int M_total;                          \
                                                                  \
  float_type *f_hat;                    \
  float_type *f;
```

Macros for public members inherited by all plan structures. Vector of samples, \ size is M_total float types.

Definition at line 14 of file nfft3.h.

9.15 nfsft.c File Reference

Implementation file for the NFSFT module.

```
#include <math.h>
#include <stdlib.h>
#include <string.h>
#include "nfft3.h"
#include "legendre.h"
#include "api.h"
#include "util.h"
```

Defines

- #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

- 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

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

  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 $n = -M, \ldots, M$.

- void c2e_transposed (nfsft_plan *plan)

  Transposed version of the function c2e.

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

  Creates a transform plan.

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

  Creates a transform plan.

- void nfsft_init_guru (nfsft_plan *plan, int N, int M, unsigned int nfsft_flags, int nfft_flags, int nfft_-cutoff)

  Creates a transform plan.

- 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 $N \in \mathbb{N}_2$.

- void nfsft_forget ()

  Forgets all precomputed data.

- void nfsft_finalize (nfsft_plan *plan)

  Destroys a plan.

- void ndsft_trafo (nfsft_plan *plan)

  Executes a direct NDSFT, i.e.

- void ndsft_adjoint (nfsft_plan *plan)

  Executes a direct adjoint NDSFT, i.e.

- void nfsft_trafo (nfsft_plan *plan)

  Executes a NFSFT, i.e.

- void nfsft_adjoint (nfsft_plan *plan)

  Executes an adjoint NFSFT, i.e.

- void nfsft_precompute_x (nfsft_plan *plan)

Variables

- static struct nfsft_wisdom wisdom = {false,0U}

  The global wisdom structure for precomputed data.

9.15.1 Detailed Description

Implementation file for the NFSFT module.

Author:
Jens Keiner

Definition in file nfsft.c.

9.16 polar_fft_test.c File Reference

NFFT-based polar FFT and inverse.

```
#include <math.h>
#include <stdlib.h>
#include "util.h"
#include "nfft3.h"
```

Functions

- int polar_grid (int T, int R, double *x, double *w)
  Generates the points $x_{t,j}$ with weights $w_{t,j}$ for the polar grid with T angles and R offsets.
- int polar_dft (fftw_complex *f_hat, int NN, fftw_complex *f, int T, int R, int m)
  discrete polar FFT
- int polar_fft (fftw_complex *f_hat, int NN, fftw_complex *f, int T, int R, int m)
  NFFT-based polar FFT.
- int inverse_polar_fft (fftw_complex *f, int T, int R, fftw_complex *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

9.16.1 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

Definition in file polar_fft_test.c.

9.17 radon.c File Reference

NFFT-based discrete Radon transform.

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>
#include "util.h"
#include "nfft3.h"
```

Defines

- #define KERNEL(r) (1.0-fabs((double)(r))/((double)R/2))
  define weights of kernel function for discrete Radon transform

Functions

- int polar_grid (int T, int R, double *x, double *w)
  generates the points x with weights w for the polar grid with T angles and R offsets
- int linogram_grid (int T, int R, double *x, double *w)
  generates the points x with weights w for the linogram grid with T slopes and R offsets
- int Radon_trafo (int(*gridfcn)(), int T, int R, double *f, int NN, double *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

9.17.1 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

Definition in file radon.c.

9.17.2 Function Documentation

9.17.2.1 int Radon_trafo (int(∗)() *gridfcn*, int *T*, int *R*, double ∗ *f*, int *NN*, double ∗ *Rf*)

computes the NFFT-based discrete Radon transform of f on the grid given by gridfcn() with T angles and R offsets

< index for nodes and freqencies

< plan for the nfft-2D

< variable for the fftw-1Ds

< plan for the fftw-1Ds

< index for directions and offsets

< knots and associated weights

init two dimensional NFFT plan

init nodes from grid

precompute psi, the entries of the matrix B

init Fourier coefficients from given image

NFFT-2D

FFTW-1Ds

finalise the plans and free the variables

Definition at line 91 of file radon.c.

References nfft_plan::f, nfft_plan::f_hat, fft(), FFT_OUT_OF_PLACE, FFTW_INIT, KERNEL, nfft_plan::M_total, MALLOC_F, MALLOC_F_HAT, MALLOC_X, nfft_plan::N_total, nfft_fftshift_complex(), nfft_finalize(), nfft_plan::nfft_flags, nfft_init_guru(), nfft_precompute_full_psi(), nfft_precompute_lin_psi(), nfft_precompute_psi(), nfft_trafo(), PRE_FULL_PSI, PRE_LIN_PSI, PRE_PHI_HUT, PRE_PSI, and nfft_plan::x.

Referenced by main().

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

simple test program for the discrete Radon transform

< grid generating function

< number of directions/offsets

< image size

load data

Radon transform

write result

free the variables

Definition at line 182 of file radon.c.

References linogram_grid(), polar_grid(), and Radon_trafo().

9.18 taylor_nfft.c File Reference

Testing the nfft againt a Taylor expansion based version.

```
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <stdlib.h>
#include "util.h"
#include "nfft3.h"
```

Data Structures

- struct taylor_plan

Functions

- void taylor_init (taylor_plan *ths, int N, int M, int n, int m)

  Initialisation of a transform plan.

- void taylor_precompute (taylor_plan *ths)

  Precomputation of weights and indices in Taylor expansion.

- void taylor_finalize (taylor_plan *ths)

  Destroys a transform plan.

- 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,.

- 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)

9.18.1 Detailed Description

Testing the nfft againt a Taylor expansion based version.

Author:
Stefan Kunis

References: Time and memory requirements of the Nonequispaced FFT

Definition in file taylor_nfft.c.

9.18.2 Function Documentation

9.18.2.1 void taylor_init (taylor_plan * *ths*, int *N*, int *M*, int *n*, int *m*)

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 37 of file taylor_nfft.c.

References FFT_OUT_OF_PLACE, FFTW_INIT, MALLOC_F, MALLOC_F_HAT, MALLOC_X, and nfft_init_guru().

Referenced by taylor_time_accuracy().

9.18.2.2 void taylor_precompute (taylor_plan * *ths*)

Precomputation of weights and indices in Taylor expansion.

- ths The pointer to a taylor plan

Author:
Stefan Kunis

Definition at line 56 of file taylor_nfft.c.

References taylor_plan::deltax0, taylor_plan::idx0, nfft_plan::M_total, nfft_plan::n, and nfft_plan::x.

Referenced by taylor_time_accuracy().

9.18.2.3 void taylor_finalize (taylor_plan * *ths*)

Destroys a transform plan.

- ths The pointer to a taylor plan

Author:
Stefan Kunis, Daniel Potts

Definition at line 78 of file taylor_nfft.c.

References taylor_plan::deltax0, taylor_plan::idx0, and nfft_finalize().

Referenced by taylor_time_accuracy().

9.18.2.4 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])

- ths The pointer to a taylor plan

Author:
Stefan Kunis

Definition at line 96 of file taylor_nfft.c.

References taylor_plan::deltax0, nfft_plan::f, nfft_plan::f_hat, nfft_plan::g1, nfft_plan::g2, taylor_plan::idx0, nfft_plan::m, nfft_plan::M_total, nfft_plan::my_fftw_plan1, nfft_plan::N_total, nfft_plan::n_total, and PI.

Referenced by taylor_time_accuracy().

9.18.2.5 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.

- 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

share nodes, input, and output vectors

output vector ndft

init pseudo random nodes

nfft precomputation

nfft precomputation

init pseudo random Fourier coefficients

NDFT

NFFT

TAYLOR NFFT

finalise

Definition at line 152 of file taylor_nfft.c.

References nfft_plan::f, nfft_plan::f_hat, FFT_OUT_OF_PLACE, FFTW_INIT, nfft_plan::M_total, nfft_plan::N_total, ndft_trafo(), nfft_error_l_infty_complex(), nfft_finalize(), nfft_plan::nfft_flags, nfft_init_guru(), nfft_precompute_one_psi(), nfft_second(), NFFT_SWAP_complex, nfft_trafo(), nfft_vrand_shifted_unit_double(), nfft_vrand_unit_complex(), taylor_plan::p, PRE_FG_PSI, PRE_ONE_PSI, PRE_PHI_HUT, taylor_finalize(), taylor_init(), taylor_precompute(), taylor_trafo(), and nfft_plan::x.

Referenced by main().

9.19 util.h File Reference

Header file for utility functions used by the nfft3 library.

`#include <complex.h>`

Defines

- #define NFFT_SWAP_complex(x, y) {double complex∗ temp; temp=(x); (x)=(y); (y)=temp;}

  Swapping of two vectors.

- #define NFFT_SWAP_double(x, y) {double∗ temp; temp=(x); (x)=(y); (y)=temp;}

  Swapping of two vectors.

- #define PI 3.1415926535897932384

  Formerly known to be an irrational number.

- #define NFFT_MAX(a, b) ((a)>(b)? (a) : (b))

  Maximum of its two arguments.

- #define NFFT_MIN(a, b) ((a)<(b)? (a) : (b))

  Mimimum of its two arguments.

Functions

- double nfft_second ()

  Actual used CPU time in seconds; calls getrusage, limited accuracy.

- int nfft_total_used_memory ()

  Actual used memory in bytes; calls mallinfo if define HAVE_MALLOC_H.

- int nfft_ld (int m)

  Integer logarithm of 2.

- int nfft_int_2_pow (int a)

  Integer power of 2.

- int nfft_next_power_of_2 (int N)

  Computes $n \geq N$ *such that* $n = 2^j$, $j \in \mathbb{N}_0$.

- void nfft_next_power_of_2_exp (int N, int ∗N2, int ∗t)

  Computes ?

- double nfft_sinc (double x)

  Computes the sinus cardinalis $\frac{sin(x)}{x}$.

- double nfft_bspline_old (int k, double x, double ∗A)

  To test the new one.

- double nfft_bspline (int k, double x, double *scratch)

  Computes the B-spline $M_{k,0}(x)$, scratch is used for de Boor's scheme.

- double nfft_i0 (double x)

  Modified Bessel function of order zero; adapted from Stephen Moshier's Cephes Math Library Release 2.8.

- int nfft_prod_int (int *vec, int d)

  Computes integer $\prod_{t=0}^{d-1} v_t$.

- int nfct_prod_int (int *vec, int d)

  Computes integer $\prod_{t=0}^{d-1} v_t$.

- int nfst_prod_minus_a_int (int *vec, int a, int d)

  Computes integer $\prod_{t=0}^{d-1} v_t - a$.

- int nfft_plain_loop (int *idx, int *N, int d)

  Computes $\sum_{t=0}^{d-1} i_t \prod_{t'=t+1}^{d-1} N_{t'}$.

- double nfft_prod_real (double *vec, int d)

  Computes double $\prod_{t=0}^{d-1} v_t$.

- double nfft_dot_complex (double complex *x, int n)

  Computes the inner/dot product $x^H x$.

- double nfft_dot_double (double *x, int n)

  Computes the inner/dot product $x^H x$.

- double nfft_dot_w_complex (double complex *x, double *w, int n)

  Computes the weighted inner/dot product $x^H (w \odot x)$.

- double nfft_dot_w_double (double *x, double *w, int n)

  Computes the weighted inner/dot product $x^H (w \odot x)$.

- double nfft_dot_w_w2_complex (double complex *x, double *w, double *w2, int n)

  Computes the weighted inner/dot product $x^H (w1 \odot w2 \odot w2 \odot x)$.

- double nfft_dot_w2_complex (double complex *x, double *w2, int n)

  Computes the weighted inner/dot product $x^H (w2 \odot w2 \odot x)$.

- void nfft_cp_complex (double complex *x, double complex *y, int n)

  Copies $x \leftarrow y$.

- void nfft_cp_double (double *x, double *y, int n)

  Copies $x \leftarrow y$.

- void nfft_cp_a_complex (double complex *x, double a, double complex *y, int n)

  Copies $x \leftarrow ay$.

- void nfft_cp_w_complex (double complex *x, double *w, double complex *y, int n)

Copies $x \leftarrow w \odot y$.

- void nfft_cp_w_double (double *x, double *w, double *y, int n)
  Copies $x \leftarrow w \odot y$.
- void nfft_upd_axpy_complex (double complex *x, double a, double complex *y, int n)
  Updates $x \leftarrow ax + y$.
- void nfft_upd_axpy_double (double *x, double a, double *y, int n)
  Updates $x \leftarrow ax + y$.
- void nfft_upd_xpay_complex (double complex *x, double a, double complex *y, int n)
  Updates $x \leftarrow x + ay$.
- void nfft_upd_xpay_double (double *x, double a, double *y, int n)
  Updates $x \leftarrow x + ay$.
- void nfft_upd_axpby_complex (double complex *x, double a, double complex *y, double b, int n)
  Updates $x \leftarrow ax + by$.
- void nfft_upd_axpby_double (double *x, double a, double *y, double b, int n)
  Updates $x \leftarrow ax + by$.
- void nfft_upd_xpawy_complex (double complex *x, double a, double *w, double complex *y, int n)
  Updates $x \leftarrow x + aw \odot y$.
- void nfft_upd_xpawy_double (double *x, double a, double *w, double *y, int n)
  Updates $x \leftarrow x + aw \odot y$.
- void nfft_upd_axpwy_complex (double complex *x, double a, double *w, double complex *y, int n)
  Updates $x \leftarrow ax + w \odot y$.
- void nfft_upd_axpwy_double (double *x, double a, double *w, double *y, int n)
  Updates $x \leftarrow ax + w \odot y$.
- void nfft_fftshift_complex (double complex *x, int d, int *N)
  Swaps each half over N[d]/2.
- double nfft_error_l_infty_complex (double complex *x, double complex *y, int n)
  Computes $\frac{\|x-y\|_\infty}{\|x\|_\infty}$.
- double nfft_error_l_infty_double (double *x, double *y, int n)
  Computes $\frac{\|x-y\|_\infty}{\|x\|_\infty}$.
- double nfft_error_l_infty_1_complex (double complex *x, double complex *y, int n, double complex *z, int m)
  Computes $\frac{\|x-y\|_\infty}{\|z\|_1}$.

- double nfft_error_l_infty_1_double (double *x, double *y, int n, double *z, int m)
  Computes $\frac{\|x-y\|_\infty}{\|z\|_1}$.
- double nfft_error_l_2_complex (double complex *x, double complex *y, int n)
  Computes $\frac{\|x-y\|_2}{\|x\|_2}$.
- double nfft_error_l_2_double (double *x, double *y, int n)
  Computes $\frac{\|x-y\|_2}{\|x\|_2}$.
- void nfft_vpr_int (int *x, int n, char *text)
  Prints a vector of integer numbers.
- void nfft_vpr_double (double *x, int n, char *text)
  Prints a vector of doubles numbers.
- void nfft_vpr_complex (double complex *x, int n, char *text)
  Prints a vector of complex numbers.
- void nfft_vrand_unit_complex (double complex *x, int n)
  Inits a vector of random complex numbers in $[0,1] \times [0,1]\mathrm{i}$.
- void nfft_vrand_shifted_unit_double (double *x, int n)
  Inits a vector of random double numbers in $[-1/2, 1/2]$.
- void nfft_voronoi_weights_1d (double *w, double *x, int M)
  Computes non periodic voronoi weights, assumes ordered nodes x_j.
- double nfft_modified_fejer (int N, int kk)
  Computes the damping factor for the modified Fejer kernel, ie $\frac{2}{N}\left(1-\frac{|2k+1|}{N}\right)$.
- double nfft_modified_jackson2 (int N, int kk)
  Computes the damping factor for the modified Jackson kernel.
- double nfft_modified_jackson4 (int N, int kk)
  Computes the damping factor for the modified generalised Jackson kernel.
- double nfft_modified_sobolev (double mu, int kk)
  Computes the damping factor for the modified Sobolev kernel.
- double nfft_modified_multiquadric (double mu, double c, int kk)
  Computes the damping factor for the modified multiquadric kernel.

9.19.1 Detailed Description

Header file for utility functions used by the nfft3 library.

Definition in file util.h.