HCFFT

2.0

Generated by Doxygen 1.7.6.1

Thu Jul 14 2016 11:27:29

For questions, errors and updates please contact:
Jan Hamaekers
jan.hamaekers@scai.fraunhofer.de

Developers:
Jan Hamaekers
Henning Pöttker
Kevin Matuschke
Tobias Olbrich
Vasil Velikov

Fraunhofer SCAI

Acknowledgements:
This work was funded in parts by the Human Brain Project (HBP)
and the Collaborative Research Centre (CRC) 1060.

Contents

1 Main Page 1

1.1 Introduction to HCFFT . . . 1

1.1.1 Supported transforms . . . 1

1.1.2 Generalized grids . . . 2

1.1.3 Adaptive grids . . . 2

1.1.4 Basic concepts . . . 3

1.1.5 Examples . . . 5

2 Dyadic Fourier sparse grid 7

3 Mixed dyadic Fourier-Chebyshev grid 11

4 Anisotropic Chebyshev grid 15

5 Fine control over the number of grid points 19

6 Using a user-defined transform 23

7 Basic adaptive grid 27

8 User-defined adaptive grid 31

9 Class Index 37

9.1 Class List . . . 37

10 File Index 39

10.1 File List . . . 39

11 Class Documentation 41

11.1 hcfft_AdaptiveIndexSet Struct Reference 41
11.1.1 Detailed Description 41
11.2 hcfft_Block Struct Reference 42
11.2.1 Detailed Description 42
11.3 hcfft_Box Struct Reference 43
11.3.1 Detailed Description 43
11.4 hcfft_Grid Struct Reference 43
11.4.1 Detailed Description 44
11.4.2 Member Data Documentation 45
11.4.2.1 gridBlockArraySize 45
11.5 hcfft_Index Struct Reference 45
11.5.1 Detailed Description 45
11.6 hcfft_Interval Struct Reference 45
11.6.1 Detailed Description 46
11.7 hcfft_StaticIndexSet Struct Reference 46
11.7.1 Detailed Description 46
11.8 hcfft_TransformData Struct Reference 46
11.8.1 Detailed Description 48
11.9 hcfft_TransformParams Struct Reference 48
11.9.1 Detailed Description 48
11.10hcfft_Vector Struct Reference 49
11.10.1 Detailed Description 49

12 File Documentation **51**

12.1 /home/hamaeker/codes/hcfft/src/avl_tree.h File Reference 51
12.1.1 Detailed Description 51
12.2 /home/hamaeker/codes/hcfft/src/block.h File Reference 51
12.2.1 Detailed Description 52
12.2.2 Typedef Documentation 52
12.2.2.1 hcfft_ComputeBlockWeight 52
12.2.2.2 hcfft_TestBlock 52
12.2.3 Function Documentation 52
12.2.3.1 hcfft_GetBlockGrid 53
12.2.3.2 hcfft_GetBlockIndex 53

12.2.3.3 hcfft_GetBlockLength . . . 53
12.2.3.4 hcfft_GetBlockStart . . . 53
12.2.3.5 hcfft_GetBlockWeight . . . 54
12.3 /home/hamaeker/codes/hcfft/src/defines.h File Reference . . . 54
12.3.1 Detailed Description . . . 55
12.3.2 Typedef Documentation . . . 55
12.3.2.1 hcfft_NodeFunction . . . 55
12.4 /home/hamaeker/codes/hcfft/src/grid.h File Reference . . . 55
12.4.1 Detailed Description . . . 56
12.4.2 Function Documentation . . . 56
12.4.2.1 hcfft_GetBlock . . . 56
12.4.2.2 hcfft_GetBoundary . . . 56
12.4.2.3 hcfft_GetDimension . . . 57
12.4.2.4 hcfft_GetDOF . . . 57
12.4.2.5 hcfft_GetDomainLength . . . 57
12.4.2.6 hcfft_GetElementSize . . . 57
12.4.2.7 hcfft_GetMaxLevel . . . 58
12.4.2.8 hcfft_GetNumberOfBlocks . . . 58
12.4.2.9 hcfft_GetValueType . . . 58
12.5 /home/hamaeker/codes/hcfft/src/hcfft.h File Reference . . . 59
12.5.1 Detailed Description . . . 61
12.5.2 Typedef Documentation . . . 61
12.5.2.1 hcfft_ArrayBaseFunction . . . 61
12.5.2.2 hcfft_DestroyParameters . . . 61
12.5.2.3 hcfft_SetGppl . . . 61
12.5.2.4 hcfft_SetPoints . . . 62
12.5.2.5 hcfft_UpdateParameters . . . 62
12.5.3 Enumeration Type Documentation . . . 62
12.5.3.1 hcfft_BlockWeightType . . . 62
12.5.3.2 hcfft_TransformType . . . 62
12.5.4 Function Documentation . . . 63
12.5.4.1 hcfft_CalculateDerivativeCoefficients . . . 63
12.5.4.2 hcfft_CalculateLaplaceCoefficients . . . 63
12.5.4.3 hcfft_ComputeInterpolatedValues . . . 63

12.5.4.4 hcfft_CreateAdaptiveGrid . . . 64
12.5.4.5 hcfft_CreateDefaultAdaptiveIndexSet . . . 64
12.5.4.6 hcfft_CreateDefaultStaticIndexSet . . . 65
12.5.4.7 hcfft_CreateGrid . . . 65
12.5.4.8 hcfft_Dehierarchize . . . 65
12.5.4.9 hcfft_DestroyGrid . . . 66
12.5.4.10 hcfft_Hierarchize . . . 66
12.5.4.11 hcfft_InitAdaptiveIndexSet . . . 66
12.5.4.12 hcfft_InitStaticIndexSet . . . 66
12.5.4.13 hcfft_InitTransformParams . . . 66
12.5.4.14 hcfft_InverseTransform . . . 67
12.5.4.15 hcfft_IterateGridNodes . . . 67
12.5.4.16 hcfft_SetGpplPlus1 . . . 67
12.5.4.17 hcfft_SparseInverseTransform . . . 67
12.5.4.18 hcfft_SparseTransform . . . 68
12.5.4.19 hcfft_Transform . . . 68
12.6 /home/hamaeker/codes/hcfft/src/hcfft_internal.h File Reference . . . 68
12.6.1 Detailed Description . . . 69
12.6.2 Function Documentation . . . 69
12.6.2.1 hcfft_chooseValueType . . . 69
12.6.2.2 hcfft_GetGppl . . . 70
12.6.2.3 hcfft_GetkFactor . . . 70
12.6.2.4 hcfft_GetPoints . . . 70
12.6.2.5 hcfft_GetStandardInterval . . . 71
12.7 /home/hamaeker/codes/hcfft/src/index.h File Reference . . . 71
12.7.1 Detailed Description . . . 72
12.7.2 Typedef Documentation . . . 72
12.7.2.1 hcfft_TestIndex . . . 72
12.7.3 Function Documentation . . . 72
12.7.3.1 hcfft_CopyIndex . . . 72
12.7.3.2 hcfft_CreateEmptyIndex . . . 73
12.7.3.3 hcfft_DecrementIndex . . . 73
12.7.3.4 hcfft_DestroyIndex . . . 73
12.7.3.5 hcfft_GetIndexDim . . . 73

12.7.3.6 hcfft_GetIndexElement . . . 74
12.7.3.7 hcfft_GetIndexMax . . . 74
12.7.3.8 hcfft_GetIndexOrder . . . 74
12.7.3.9 hcfft_GetIndexSum . . . 75
12.7.3.10 hcfft_IncrementIndex . . . 75
12.7.3.11 hcfft_SetIndexElement . . . 75
12.8 /home/hamaeker/codes/hcfft/src/lattice_rules.h File Reference . . . 75
12.8.1 Detailed Description . . . 75
12.9 /home/hamaeker/codes/hcfft/src/leja/classic_leja.h File Reference . . . 76
12.9.1 Detailed Description . . . 76
12.10/home/hamaeker/codes/hcfft/src/leja/hermite_leja.h File Reference . . . 76
12.10.1 Detailed Description . . . 76
12.11/home/hamaeker/codes/hcfft/src/leja/laguerre_leja.h File Reference . . . 76
12.11.1 Detailed Description . . . 76
12.12/home/hamaeker/codes/hcfft/src/leja/leja.h File Reference . . . 76
12.12.1 Detailed Description . . . 77
12.12.2 Function Documentation . . . 77
12.12.2.1 hcfft_GenerateLejaPoints . . . 77
12.13/home/hamaeker/codes/hcfft/src/priority_queue.h File Reference . . . 77
12.13.1 Detailed Description . . . 77
12.14/home/hamaeker/codes/hcfft/src/transform_data.h File Reference . . . 77
12.14.1 Detailed Description . . . 78
12.14.2 Function Documentation . . . 78
12.14.2.1 hcfft_DestroyTransformationStuff . . . 78
12.14.2.2 hcfft_GetStdTSInterval . . . 78
12.14.2.3 hcfft_InitTransformationStuff . . . 79
12.15/home/hamaeker/codes/hcfft/src/transforms/general_transform.h File Reference . . . 79
12.15.1 Detailed Description . . . 79
12.16/home/hamaeker/codes/hcfft/src/transforms/legendre.h File Reference . . . 79
12.16.1 Detailed Description . . . 79
12.17/home/hamaeker/codes/hcfft/src/utils.h File Reference . . . 80
12.17.1 Detailed Description . . . 80
12.17.2 Function Documentation . . . 81

12.17.2.1 hcfft_ComputeL2Error 81
12.17.2.2 hcfft_ComputeL2Norm 81
12.17.2.3 hcfft_EvalErrorAtGridPoints 81
12.17.2.4 hcfft_EvalErrorAtRandomPoints 82
12.17.2.5 hcfft_GetGridIndices 82
12.17.2.6 hcfft_GetGridPoints 83
12.17.2.7 hcfft_PrintCoeffVector 83
12.17.2.8 hcfft_PrintGridBasisIndices 83
12.17.2.9 hcfft_PrintGridLevels 83
12.17.2.10hcfft_PrintGridPoints 84
12.18/home/hamaeker/codes/hcfft/src/vector.h File Reference 84
12.18.1 Detailed Description 84
12.18.2 Function Documentation 85
12.18.2.1 hcfft_AddVector 85
12.18.2.2 hcfft_ComputeSquareL2Norm 85
12.18.2.3 hcfft_CopyVector 85
12.18.2.4 hcfft_CreateVector 85
12.18.2.5 hcfft_DestroyVector 86
12.18.2.6 hcfft_GetComplexData 86
12.18.2.7 hcfft_GetRealData 86
12.18.2.8 hcfft_SetVectorToZero 87

Chapter 1

Main Page

1.1 Introduction to HCFFT

HCFFT is a flexible C library for interpolation on sparse grids. Despite its name, it is neither limited to the Fourier transform, nor to hyperbolic cross sparse grids. Since its first version it has evolved into a generic framework for grid interpolation. It supports many different types of transforms, grids and grid generation methods. It is easily extensible with user-defined transforms, index sets, etc.

1.1.1 Supported transforms

The following transform types are currently supported:

- Fourier Transform
- Real Fourier Transform
- Cosine Transform
- Sine Transform
- Chebyshev Transform
- Legendre Transform
- Generalized Hermite transform
- Jacobi transform
- Laguerre transform

Whenever possible, the fast versions of the transforms are used for the one-dimensional operations.

1.1.2 Generalized grids

Each of the transforms has some default behaviour, which is guided by the default interpolation points, number of grid points per level and interpolation interval. In many applications the default values provide optimal convergence. In some cases, however, they do not suffice. Each type of transform is completely customizable. The user can provide custom:

- interpolation points
- number of points per level
- interpolation interval
- basis function parameters α and β, whenever applicable.

The user can also define a completely custom transform. To achieve this, apart from the list above, the user should also provide:

- the basis functions
- the type of data on which the functions acts - real or complex.

Typical dyadic sparse grids work with generalized hyperbolic cross index sets $\mathscr{I}_L^T$, where

$$\mathscr{I}_L^T = \{\nu : |\nu|_1 - T|\nu|_\infty \leq (1-T)L\}.$$

Sometimes the form of these index sets is not flexible enough for the problem at hand. The most basic example is a function in which one of the dimensions is much more important than the others. In these cases it is best to use an anisotropic grid i.e. a grid that generates more levels in the important dimension. The sets $\mathscr{I}_L^T$ cannot handle this case, as well as many other cases. Therefore, HCFFT allows the user to provide a custom index set. The only requirement on the user-defined set is that it must be admissible.

Lastly, all the described techniques can be combined together. HCFFT can generate grids with arbitrary index sets, modified and user-defined transforms. A different transform can be used in every direction. Transforms that work on different types of data(real or complex) can be mixed. Thus, it is possible to generate as customized sparse grids as desired.

1.1.3 Adaptive grids

HCFFT allows the user to create adaptive sparse grids. These are grids which are constructed based on the target function. The adaptive algorithm constructs the grid iteratively. On each step it inspects a level index and tries to determine its importance. If it is considered important then it is inserted into the grid. Otherwise it is discarded. The importance of a level index is measure by the so-called index weight. The higher the weight, the higher the importance of the index. Calculating the weight of the index usually involves function computation at the grid nodes that correspond to this level.

HCFFT implements a default version of the adaptive algorithm that can be used by calling a single function. However, each step of the algorithm can be customized by providing a custom index rejection functions, weighing function and termination criteria.

1.1.4 Basic concepts

Before moving on to the examples, it is useful to define the basic concepts that the library uses. Roughly speaking, each concept is wrapped in a corresponding C struct. The most important once are summarized in the table below.

C struct	Description
hcfft_Grid	This is the main object in HCFFT. It encapsulates everything related to the sparse grid: • The dimension • The transforms and their parameters • The type of data(real/complex) • The type of grid(dyadic, general, adaptive) • The index set It is created with hcfft_CreateGrid or with hcfft_CreateAdaptiveGrid.
hcfft_Vector	This structure represents a vector of values, one for each node in the sparse grids. It can be used to store function values, regular coefficients or hierarchical coefficients. Each vector is bound to the grid for which it is created. This is why all functions that are executed on a vector also take a grid as a second argument.
hcfft_Index	Represents an index in the index set. Just as hcfft_Vector, an index only exists in the context of the grid for which it is created. This is why most functions that act on it also also take the grid as a second argument.
hcfft_Block	A block in the sparse grid corresponds to a level index in the index set. Alternatively, a block is constructed for each hierarchical subspace W_v, for every $v \in \mathcal{I}$. Consequently, the corresponding subgrid has a product structure. Direct interaction with grid blocks is usualy needed only for more advanced usages of HCFFT.
hcfft_TransformParams	This struct is used to specify the transform for a certain dimension of the grid. One such structure must be provided for each dimension. It is passed as a parameter to hcfft_CreateGrid and hcfft_CreateAdaptiveGrid.
hcfft_StaticIndexSet	Parameters that guide the creation of the index set of the non-adaptive part of the grid. If the user wants a hyperbolic cross index set then hcfft_CreateDefaultStaticIndexSet can be used. For more complex applications

These are the structures in HCFFT with which the user will usually interact. Simple applications require the usage of hcfft_Grid, hcfft_Vector, hcfft_TransformParams and hcfft_StaticIndexSet. More complex applications might require all of them.

1.1.5 Examples

The main features of the library are demonstrated in the examples below.

- Dyadic Fourier sparse grid
- Mixed dyadic Fourier-Chebyshev grid
- Anisotropic Chebyshev grid
- Fine control over the number of grid points
- Using a user-defined transform
- Basic adaptive grid
- User-defined adaptive grid

Chapter 2

Dyadic Fourier sparse grid

This example perfoms a Fourier transform on a regular dyadic sparse grid. The interpolated function is 2-dimensional:

$$f(x_1,x_2) = \prod_{i=1}^{2}(2 + sign(x_i - \pi)sin(x_i)^2)$$

The main steps in the example are:

- The first step of this example is to allocate one hcfft_TransformParams instance for each dimension of the grid. This has to be done even if all dimension are equivalent. Then each of these parameters is initialized and set to use the default Fourier transform.

- The next step is to create the parameters for the index set generation. For this purpose we have to create an instance of hcfft_StaticIndexSet. The function hcfft_CreateDefaultStaticIndexSet creates a set that corresponds to a parameterized hyperbolic cross.

- A sparse grid is created from the provided transform parameters and index set. This is done with the function hcfft_CreateGrid. Note that this function does not perform any heavy computations. It simply initializes the internal structure of the grid.

- The function hcfft_CreateVector is used to create a data vector that stores contains one value for each grid node. The created vector is empty.

- hcfft_IterateGridNodes visits every grid node and executes the given function on it. Every function evaluation is recorded in the appropriate place in the *coeff* vector.

- At this point *coeffs* contains the function values. They have to be transformed to the Fourier coefficients. This is achieved with a call to hcfft_Transform.

- The utility function hcfft_ComputeL2Error computes the error between the target function and the interpolant. The interpolant is defined by the pair (grid, coeffs).

- Finally, all allocated resources are freed.

```
#include <stdio.h>
#include <stdlib.h>
#include "hcfft.h"

// The interpolated function.
void testFunction(const struct hcfft_Grid *grid, const int *nodeIndex, const
      hcfft_double *nodePoint, void *nodeValue, void *params)
{
  (void)nodeIndex;
  (void)params;

  // Get the grid dimensions.
  int D = hcfft_GetDimension(grid);

  // The Fourier transform operates on complex coefficients.
  hcfft_complex* result = (hcfft_complex*)nodeValue;

  // Calculate the function value at this point.
  (*result)[0] = 1.0;
  (*result)[1] = 0.0;
  for(int d = 0; d < D; ++d)
  {
    hcfft_double x = nodePoint[d];
    int sign = ((x - hcfft_pi) < 0)? -1 : ((x - hcfft_pi) > 0)? 1 : 0;
    (*result)[0] *= (2 + sign * sin(x) * sin(x));
  }
}

int main()
{
  int dim = 2;              // The grid dimension.
  int level = 10;           // The grid level.
  hcfft_double T = 0.0;     // The hyperbolic cross index set parameter.

  // Allocate hcfft_TransformParams struct for every dimension.
  struct hcfft_TransformParams* transforms = malloc(dim * sizeof(*transforms));

  for(int d = 0; d < dim; ++d)
  {
    // Initialize the structure.
    hcfft_InitTransformParams(&transforms[d]);

    // Set the transform type to Fourier. We will use the default transform
       parameters so
    // leave everything else as it is.
    transforms[d].type = hcfft_Fourier;
  }

  // Create default parameters for the index set generation.
  struct hcfft_StaticIndexSet staticIndexSet = hcfft_CreateDefaultStaticIndexSet
      (level, T, dim);

  // Generate a regular dyadic sparse grid.
  struct hcfft_Grid *grid = hcfft_CreateGrid(dim, level, transforms,
      staticIndexSet);
```

```
  // Print the number of nodes in the grid.
  int DOF = hcfft_GetDOF(grid);
  printf("The grid has %d nodes.\n", DOF);

  // Allocate a vector for the Fourier coefficients.
  struct hcfft_Vector *coeffs = hcfft_CreateVector(grid);

  // Compute the function values at the grid points and store them in the
       vector.
  hcfft_IterateGridNodes(grid, testFunction, NULL, coeffs);

  // Perform a Fourier transform on the coefficient vector.
  hcfft_Transform(grid, coeffs);

  // Compute the L2 error of the interpolant.
  hcfft_double L2er = hcfft_ComputeL2Error(grid, coeffs, testFunction, NULL, 10
      , NULL);
  printf("L2 Error: %e\n", L2er);

  // Free the allocated coefficients vector.
  hcfft_DestroyVector(coeffs);

  // Free the sparse grid.
  hcfft_DestroyGrid(grid);

  // Free the transform parameters.
  free(transforms);

  return 0;
}
```

Chapter 3

Mixed dyadic Fourier-Chebyshev grid

This example demonstrates the usage of mixed grids. The goal is to intepolate a four dimensional function on a mixed grid that uses the Fourier transform for the first two dimensions and the Chebyshev transform for the other two. The test function is:

$$f(x) = \prod_{i=1}^{2}(2 + sign(x_i - \pi)sin(x_i)^2) \prod_{i=3}^{4} \frac{1}{1+x_i^2}$$

The steps are almost identical to Dyadic Fourier sparse grid. The new parts are listed below:

- Care must be taken about the data type on which the mixed grid works. If all used transforms use real data then the grid also works on real data. If even a single transform uses complex numbers then the grid also uses complex numbers. Consequently, real transforms executed on complex data must be executed once for the real part and once for the complex part of the data. This is handled internally by HCFFT.

- Using a mixed grid is simply a matter of setting different type values in hcfft_TransformParams. Everything else happens transparently to the user.

- The function hcfft_EvalErrorAtGridPoints computes the maximum difference between the interpolant and the original function at the grid points. If the interpolation is correct then this error should be equal to zero(i.e. less than 1e-12). The function can be used for testing purposes.

```
#include <stdio.h>
#include <stdlib.h>
#include "hcfft.h"

// The interpolated function.
void testFunction(const struct hcfft_Grid *grid, const int *nodeIndex, const
      hcfft_double *nodePoint, void *nodeValue, void *params)
{
  (void)nodeIndex;
```

```
  (void)params;

  // Get the grid dimensions.
  int D = hcfft_GetDimension(grid);

  // A mixed Fourier-Chebyshev transform operates on complex coefficients.
  hcfft_complex* result = (hcfft_complex*)nodeValue;

  // Calculate the function value at this point.
  (*result)[0] = 1.0;
  (*result)[1] = 0.0;
  for(int d = 0; d < D; ++d)
  {
    hcfft_double x = nodePoint[d];
    if(d < D / 2)
    {
      int sign = ((x - hcfft_pi) < 0)? -1 : ((x - hcfft_pi) > 0)? 1 : 0;
      (*result)[0] *= (2 + sign * sin(x) * sin(x));
    }
    else
    {
      (*result)[0] *= 1.0 / (1 + nodePoint[d] * nodePoint[d]);
    }
  }
}

int main()
{
  int dim = 4;             // The grid dimension.
  int level = 7;           // The grid level.
  hcfft_double T = 0.0;    // The hyperbolic cross index set parameter.

  // Allocate hcfft_TransformParams struct for every dimension.
  struct hcfft_TransformParams* transforms = malloc(dim * sizeof(*transforms));

  for(int d = 0; d < dim; ++d)
  {
    // Initialize the structure.
    hcfft_InitTransformParams(&transforms[d]);

    if(d < dim / 2)
    {
      // Set the transform type to Fourier for the first half of dimensions.
      transforms[d].type = hcfft_Fourier;
    }
    else
    {
      // Set the transform type to Chebyshev for the second half of dimensions.
      transforms[d].type = hcfft_Chebyshev;
    }
  }

  // Create default parameters for the index set generation.
  struct hcfft_StaticIndexSet staticIndexSet = hcfft_CreateDefaultStaticIndexSet
      (level, T, dim);

  // Generate a regular dyadic sparse grid.
  struct hcfft_Grid *grid = hcfft_CreateGrid(dim, level, transforms,
      staticIndexSet);

  // Print the number of nodes in the grid.
  int DOF = hcfft_GetDOF(grid);
```

```
  printf("The grid has %d nodes.\n", DOF);

  // Allocate a vector for the coefficients.
  struct hcfft_Vector *coeffs = hcfft_CreateVector(grid);

  // Compute the function values at the grid points and store them in the
       vector.
  hcfft_IterateGridNodes(grid, testFunction, NULL, coeffs);

  // Perform a mixed transform on the coefficient vector.
  hcfft_Transform(grid, coeffs);

  // Compute the L2 error of the interpolant.
  hcfft_double L2er = hcfft_ComputeL2Error(grid, coeffs, testFunction, NULL, 10
      , NULL);
  printf("L2 Error: %e\n", L2er);

  // Calculate the interpolation error at the grid points.
  hcfft_double gridError = hcfft_EvalErrorAtGridPoints(grid, coeffs,
      testFunction, NULL);
  printf("Grid nodes error: %e\n", gridError);

  // Free the allocated coefficients vector.
  hcfft_DestroyVector(coeffs);

  // Free the sparse grid.
  hcfft_DestroyGrid(grid);

  // Free the transform parameters.
  free(transforms);

  return 0;
}
```

Chapter 4

Anisotropic Chebyshev grid

This example demonstrates how to construct a grid with a custom index set. The example constructs an anisotropic 2D Chebysev grid with level indices which satisfy $v_0 + 1.5v_1 \leq L$ for some value L. The approximated function is:

$$f(x) = \frac{1}{(1+10x_0^2)(1+0.01x_1^2)}$$

The most important parts of this example are:

- To create a custom index set the user must create custom hcfft_StaticIndexSet parameters. All fields must be set to some valid values. The most important one is hcfft_StaticIndexSet.insertBlock. This is a predicate function which answers the question: "Should I insert the given index in the index set?".

- The predicate function can be passed user-defined parameters. This is the meaning of the variable hcfft_StaticIndexSet.params. These parameters are not modified in any way by HCFFT.

- Note the usage of hcfft_Index in insertBlock(). The access to the index elements is done with accessor functions. Direct access is impossible because the implementation of hcfft_Index is hidden from the user.

- Note that if the values of w0 and w1 are swapped the error increases significantly.

- The grid construction function hcfft_CreateGrid accepts a maxLevel parameter. This parameter has nothing to do with the generated index set. It denotes the maximum level of refinement in every dimension. Indices which have an element greater than this value are not inserted in the index set, even if insertBlock permits it.

```
#include <stdio.h>
#include <stdlib.h>
```

```
#include "hcfft.h"

// The interpolated function.
void testFunction(const struct hcfft_Grid *grid, const int *nodeIndex, const
      hcfft_double *nodePoint, void *nodeValue, void *params)
{
  (void)nodeIndex;
  (void)params;
  (void)grid;

  // A Chebyshev transform operates on real coefficients.
  hcfft_double* result = (hcfft_double*)nodeValue;

  // Calculate the function value at this point.
  *result = 1.0 / (1 + 10 * nodePoint[0] * nodePoint[0]);
  *result *= 1.0 / (1 + 0.01 * nodePoint[1] * nodePoint[1]);
}

// Custom function for index insertion into the index set.
int insertBlock(struct hcfft_Grid* grid, struct hcfft_Index* index, void*
      params)
{
  (void)grid;
  (void)index;
  (void)params;

  const int L = 10;

  // Get the index elements.
  int nu0 = hcfft_GetIndexElement(grid, index, 0);
  int nu1 = hcfft_GetIndexElement(grid, index, 1);

  // Element weights.
  hcfft_double w0 = 1.0;
  hcfft_double w1 = 1.5;

  // Use weights on the index elements.
  // The higher the weight, the less blocks will be allowed in this direction.
  return (w0 * nu0 + w1 * nu1 <= L)? 1 : 0;
}

int main()
{
  int dim = 2;             // The grid dimension.
  int maxLevel = 10;        // Maximum allowed refinement level in each
       dimension.

  // Allocate hcfft_TransformParams struct for every dimension.
  struct hcfft_TransformParams* transforms = malloc(dim * sizeof(*transforms));

  for(int d = 0; d < dim; ++d)
  {
    // Initialize the structure.
    hcfft_InitTransformParams(&transforms[d]);

    // Set the transform type to Chebyshev for the second half of dimensions.
    transforms[d].type = hcfft_Chebyshev;
  }

  // Create custom parameters for the index set generation.
  struct hcfft_StaticIndexSet staticIndexSet;
  staticIndexSet.insertBlock = insertBlock;
```

```
  staticIndexSet.params = NULL;
  staticIndexSet.destroyParams = NULL;

  // Generate a custom dyadic grid.
  struct hcfft_Grid *grid = hcfft_CreateGrid(dim, maxLevel, transforms,
      staticIndexSet);

  // Print the number of nodes in the grid.
  int DOF = hcfft_GetDOF(grid);
  printf("The grid has %d nodes.\n", DOF);

  // Allocate a vector for the coefficients.
  struct hcfft_Vector *coeffs = hcfft_CreateVector(grid);

  // Compute the function values at the grid points and store them in the
       vector.
  hcfft_IterateGridNodes(grid, testFunction, NULL, coeffs);

  // Perform a transform on the coefficient vector.
  hcfft_Transform(grid, coeffs);

  // Compute the L2 error of the interpolant.
  hcfft_double L2norm = hcfft_ComputeL2Norm(grid, testFunction, NULL, 10, NULL)
      ;
  hcfft_double L2er = hcfft_ComputeL2Error(grid, coeffs, testFunction, NULL, 10
      , NULL);
  printf("Relative L2 Error: %e\n", L2er / L2norm);

  // Free the allocated coefficients vector.
  hcfft_DestroyVector(coeffs);

  // Free the sparse grid.
  hcfft_DestroyGrid(grid);

  // Free the transform parameters.
  free(transforms);

  return 0;
}
```

Chapter 5

Fine control over the number of grid points

All previous examples were using dyadic grids. In such grids, performing one refinement step in a fixed dimension increases the number of points in this direction roughly twice. This is sometimes undersirable because it does not allow fine control over the number of points in the grid when the dimension is high. The solution to this problem is to perform smaller refinement steps. HCFFT allows the user to provide the desired number of grid points per level for each dimension. This example demonstrates how to add only one new point to each new level. The constructed grid is a Chebyshev one and the approximated function is

$$f(x) = \prod_{d=1}^{n} \frac{1}{(1+x_d^2)}$$

The only new idea in this example is the function *setGppl()*. This is a user defined function that is used by the grid to set the size of the refinement steps. Every dimension can have its own function. It is set through the hcfft_TransformParams parameters by simply assigning it to hcfft_TransformParams.gpplFunc. The default behaviour of this function for most transforms is to set $g_n = 2^n$. The only requirement on the user-defined values g_n is that they form an increasing sequence.

Note also that because of the small refinement steps we need to build much bigger index sets if we want to get the same number of grid nodes. In this case the level of the grid is equal to 20. This is clearly not possible for dyadic grids.

```
#include <stdio.h>
#include <stdlib.h>
#include "hcfft.h"

// The interpolated function.
void testFunction(const struct hcfft_Grid *grid, const int *nodeIndex, const
      hcfft_double *nodePoint, void *nodeValue, void *params)
{
  (void)nodeIndex;
  (void)params;
```

```
  (void)grid;

  // Get the grid dimensions.
  int D = hcfft_GetDimension(grid);

  // A Chebyshev transform operates on real coefficients.
  hcfft_double* result = (hcfft_double*)nodeValue;

  // Calculate the function value at this point.
  *result = 1.0;
  for(int d = 0; d < D; ++d)
  {
    *result *= 1.0 / (1 + nodePoint[d] * nodePoint[d]);
  }
}

// A function that sets the elements of the array gppl to 1, 2, 3, 4...
void setGppl(int *gppl, int L)
{
  for(int i = 0; i <= L; ++i)
    gppl[i] = i + 1;
}

int main()
{
  int dim = 5;              // The grid dimension.
  int level = 20;           // Maximum allowed refinement level in each
       dimension.
  hcfft_double T = 0.0;     // The hyperbolic cross index set parameter.

  // Allocate hcfft_TransformParams struct for every dimension.
  struct hcfft_TransformParams* transforms = malloc(dim * sizeof(*transforms));

  for(int d = 0; d < dim; ++d)
  {
    // Initialize the structure.
    hcfft_InitTransformParams(&transforms[d]);

    // Set the transform type to Chebyshev for the second half of dimensions.
    transforms[d].type = hcfft_Chebyshev;

    // Set the function that provides the number of grid points per level.
    transforms[d].gpplFunc = setGppl;
  }

  // Create default parameters for the index set generation.
  struct hcfft_StaticIndexSet staticIndexSet = hcfft_CreateDefaultStaticIndexSet
      (level, T, dim);

  // Generate the slowly-increasing grid.
  struct hcfft_Grid *grid = hcfft_CreateGrid(dim, level, transforms,
      staticIndexSet);

  // Print the number of nodes in the grid.
  int DOF = hcfft_GetDOF(grid);
  printf("The grid has %d nodes.\n", DOF);

  // Allocate a vector for the coefficients.
  struct hcfft_Vector *coeffs = hcfft_CreateVector(grid);

  // Compute the function values at the grid points and store them in the
       vector.
```

```
  hcfft_IterateGridNodes(grid, testFunction, NULL, coeffs);

  // Perform a transform on the coefficient vector.
  hcfft_Transform(grid, coeffs);

  // Compute the L2 error of the interpolant.
  hcfft_double L2er = hcfft_ComputeL2Error(grid, coeffs, testFunction, NULL, 10
      , NULL);
  printf("L2 Error: %e\n", L2er);

  // Free the allocated coefficients vector.
  hcfft_DestroyVector(coeffs);

  // Free the sparse grid.
  hcfft_DestroyGrid(grid);

  // Free the transform parameters.
  free(transforms);

  return 0;
}
```

Chapter 6

Using a user-defined transform

The previous example demonstrated how to set the number of grid points per level manually. HCFFT is much more flexible than that and allows the user to define a completely custom transform. This way the library can be extended very easily to support even more types of sparse grids.

To define a custom transform the user has to fill in all the fields of hcfft_TransformParams with user-defined functions. The code below demonstrates how to do this. The approximated function is

$$f(x) = \prod_{d=1}^{n} \frac{1}{(1+x_d^2)}$$

The user-defined grid has the following properties:

- The constructed grid has a slowly increasing sequence of grid points per level.
- The basis functions are the monomials. These basis functions are generally not a good choice for interpolation. However, they are convenient for simple examples.
- The interpolation points are $\mathscr{R}$-Leja points. The point generation function is copied from the internal implementation in HCFFT. You don't have to understand how it works, you only need to understand what its result is.
- The interpolation domain is [-1,1]
- The data on which the monomials operate is real.

```
#include <stdio.h>
#include <stdlib.h>
#include "hcfft.h"

// The interpolated function.
void testFunction(const struct hcfft_Grid *grid, const int *nodeIndex, const
      hcfft_double *nodePoint, void *nodeValue, void *params)
{
  (void)nodeIndex;
  (void)params;
```

```
  (void)grid;

  // Get the grid dimensions.
  int D = hcfft_GetDimension(grid);

  // A Chebyshev transform operates on real coefficients.
  hcfft_double* result = (hcfft_double*)nodeValue;

  // Calculate the function value at this point.
  *result = 1.0;
  for(int d = 0; d < D; ++d)
  {
    *result *= 1.0 / (1 + nodePoint[d] * nodePoint[d]);
  }
}

// A function that sets the elements of the array gppl to 1, 3, 5, 7...
void setGppl(int *gppl, int L)
{
  for(int i = 0; i <= L; ++i)
    gppl[i] = 2 * i + 1;
}

// A functions that sets the interpolation points. The points are stored in the
// points array. The first n interpolation points should be provided. The
      parameters alpha
// and beta are used only for transforms parameterizable basis functions. Thus,
      here they are
// ignored.
// This function computes the first n points from the R-Leja. It can be used as
      a
// black box function.
void setPoints(hcfft_double *points, int n, hcfft_double alpha, hcfft_double
     beta)
{
  (void)alpha;
  (void)beta;

  int max_dlev = (int) ceil(log2((double)n));
  int act_point = 3;
  double act_length = 0.0;

  if(n >= 1) points[0] = 0;
  if(n >= 2) points[1] = hcfft_pi;
  if(n >= 3) points[2] = 0.5 * hcfft_pi;

  for(int i = 1; i <= max_dlev && act_point < n; ++i)
  {
    act_length = points[(1 << (i-1)) + 1];

    for(int j = (1 << (i-1)); j < (1 << i); ++j)
      if(act_point < n)
        points[act_point++] = points[j + 1] - 0.5 * act_length;

    for(int j = (1 << (i-1)); j < (1 << i); ++j)
      if(act_point < n)
        points[act_point++] = points[j + 1] + 0.5 * act_length;
  }

  if(n >= 1) points[0] = 0.5 * hcfft_pi;
  if(n >= 2) points[1] = 0.0;
  if(n >= 3) points[2] = hcfft_pi;
```

```
  for(int i = 0; i < n; ++i)
    points[i] = hcfft_cos(points[i]);
}

// This is the function that computes the basis functions at a given point. The
      function computes
// the first N basis functions. The computations in HCFFT are organized in such
      a way that
// a basis function is never computed on its own. Instead, basis functions are
      computed in bunches.
// The parameters alpha and beta can be ignored because our basis functions do
      not need any extra
// parameters.
void arrayBaseFunction(int n, hcfft_double x, void* result, hcfft_double alpha,
      hcfft_double beta)
{
  (void)alpha;
  (void)beta;

  hcfft_double * const res = (hcfft_double*)result;

  res[0] = 1.0;
  for(int i = 1; i < n; ++i)
    res[i] = res[i-1] * x;
}

int main()
{
  int dim = 3;             // The grid dimension.
  int level = 15;          // Maximum allowed refinement level in each
      dimension.
  hcfft_double T = 0.0;    // The hyperbolic cross index set parameter.

  // Allocate hcfft_TransformParams struct for every dimension.
  struct hcfft_TransformParams* transforms = malloc(dim * sizeof(*transforms));

  struct hcfft_Interval boundary = { -1, 1 };
  for(int d = 0; d < dim; ++d)
  {
    // Initialize the structure.
    hcfft_InitTransformParams(&transforms[d]);

    // All fields of transforms[d] must be set manually
    transforms[d].type = hcfft_CustomTransform;
    transforms[d].boundary = &boundary;
    transforms[d].gpplFunc = setGppl;
    transforms[d].pointsFunc = setPoints;
    transforms[d].arrayBaseFunction = arrayBaseFunction;
    transforms[d].valueType = hcfft_RealValue;
  }

  // Create default parameters for the index set generation.
  struct hcfft_StaticIndexSet staticIndexSet = hcfft_CreateDefaultStaticIndexSet
      (level, T, dim);

  // Generate the slowly-increasing grid.
  struct hcfft_Grid *grid = hcfft_CreateGrid(dim, level, transforms,
      staticIndexSet);

  // Print the number of nodes in the grid.
  int DOF = hcfft_GetDOF(grid);
```

```
  printf("The grid has %d nodes.\n", DOF);

  // Allocate a vector for the coefficients.
  struct hcfft_Vector *coeffs = hcfft_CreateVector(grid);

  // Compute the function values at the grid points and store them in the
       vector.
  hcfft_IterateGridNodes(grid, testFunction, NULL, coeffs);

  // Perform a transform on the coefficient vector.
  hcfft_Transform(grid, coeffs);

  // Compute the L2 error of the interpolant.
  hcfft_double L2er = hcfft_ComputeL2Error(grid, coeffs, testFunction, NULL, 10
      , NULL);
  printf("L2 Error: %e\n", L2er);

  // Free the allocated coefficients vector.
  hcfft_DestroyVector(coeffs);

  // Free the sparse grid.
  hcfft_DestroyGrid(grid);

  // Free the transform parameters.
  free(transforms);

  return 0;
}
```

Chapter 7

Basic adaptive grid

This example demonstrates how to construct an adaptive grid based on a target function. The approximated function is:

$$f(x) = \frac{1}{(1+10x_0^2)(1+0.1x_1^2)(1+0.1x_2^2)(1+0.1x_3^2)}$$

This function has one very important dimension and three less important ones. - Constructing an isotropic grid for it would not be optimal. One way to achieve better approximation is to use an anisotropic grid which has more points in the first dimension. However, it is difficult to choose how many more samples to add compared to the other dimensions. Thus, it is best to do this automatically. This is the purpose of the adaptive algorithm - to construct a proper index set that takes into account the concrete function properties.

The default adaptive algorithm starts with some initial index set and tries to refine it. The refinement process is iterative. On each step, the algorithm choose the best block and inserts it in the grid. If the block has a weight under a certain threshold then its neighbours are queued for later inspection. The algorithm terminates when there are no more blocks with the desired weight or when the maximum number of allowed grid points has been exceeded.

The most important parts of the code below are:

- The user is allowed to provide an initial index set. This is the role of the staticIndexSet in this example. The adaptive algorithm is executed after the initial grid is constructed. In our case we insert just a single block in the initial set.

- We can control the maximum order of the indices that enter the grid. This way we can limit the interaction between the dimensions. The static and adaptive parts of the grid are controlled separately.

- We can also set a limit to the number of nodes that the adaptive grid constructs. The algorithm terminates once the limit is exceeded.

- The most important parameter of the algorithm is the threshold ε., as well as the block weighing function. Blocks whose weight is measured to be more than the threshold are expanded from the adaptive algorithm. Thus, the right choice of a weighing function is extremely important. By default, HCFFT offers four options:

1. hcfft_HierarchicalSquareSum - Squared sum of the hierarchical coefficients in the block.
2. hcfft_CoefficientsSquareSum - Squared sum of the regular coefficients in the block.
3. hcfft_AverageHierarchicalSquareSum - Averaged squared sum of the hierarchical coefficients in the block.
4. hcfft_AverageCoefficientsSquareSum - Averaged squared sum of the regular coefficients in the block.

```
#include <stdio.h>
#include <stdlib.h>
#include "hcfft.h"

// The interpolated function.
void testFunction(const struct hcfft_Grid *grid, const int *nodeIndex, const
      hcfft_double *nodePoint, void *nodeValue, void *params)
{
  (void)nodeIndex;
  (void)params;
  (void)grid;

  // A Chebyshev transform operates on real coefficients.
  hcfft_double* result = (hcfft_double*)nodeValue;

  // Calculate the function value at this point.
  *result = 1.0 / (1 + 10 * nodePoint[0] * nodePoint[0]);
  *result *= 1.0 / (1 + 0.1 * nodePoint[1] * nodePoint[1]);
  *result *= 1.0 / (1 + 0.1 * nodePoint[2] * nodePoint[2]);
  *result *= 1.0 / (1 + 0.1 * nodePoint[3] * nodePoint[3]);
}

int main()
{
  int dim = 4;              // The grid dimension.
  int initLevel = 0;        // The initial non-adaptive level.
  int maxLevel = 10;        // Maximum allowed refinement level in each
       dimension.
  hcfft_double T = 0.0;     // The hyperbolic cross parameter for the initial
       index set.
  hcfft_double eps = 1e-5;  // Termination criteria.
  int maxOrder = dim;       // Indices with great order than this will not be
       allowed in the grid.
  int maxNodes = (1 << 30); // The algorithm terminates when the number of grid
       nodes exceeds this number.

  // Allocate hcfft_TransformParams struct for every dimension.
  struct hcfft_TransformParams* transforms = malloc(dim * sizeof(*transforms));

  for(int d = 0; d < dim; ++d)
  {
    // Initialize the structure.
```

```
    hcfft_InitTransformParams(&transforms[d]);

    // Set the transform type to Chebyshev for the second half of dimensions.
    transforms[d].type = hcfft_Chebyshev;
  }

  // Create default parameters for the initial index set generation.
  struct hcfft_StaticIndexSet staticIndexSet = hcfft_CreateDefaultStaticIndexSet
      (initLevel, T, maxOrder);

  // Create default parameters for the adaptive index set generation.
  struct hcfft_AdaptiveIndexSet adaptiveIndexSet =
      hcfft_CreateDefaultAdaptiveIndexSet(
    testFunction, NULL, eps, hcfft_AverageHierarchicalSquareSum, maxNodes,
      maxOrder);

  // Generate an adaptive dyadic grid.
  struct hcfft_Grid *grid = hcfft_CreateAdaptiveGrid(dim, dim, maxLevel,
      transforms, staticIndexSet, adaptiveIndexSet);

  // Print the number of nodes in the grid.
  int DOF = hcfft_GetDOF(grid);
  printf("The grid has %d nodes.\n", DOF);

  // Allocate a vector for the coefficients.
  struct hcfft_Vector *coeffs = hcfft_CreateVector(grid);

  // Compute the function values at the grid points and store them in the
       vector.
  hcfft_IterateGridNodes(grid, testFunction, NULL, coeffs);

  // Perform a transform on the coefficient vector.
  hcfft_Transform(grid, coeffs);

  // Compute the relative L2 error of the interpolant.
  hcfft_double L2norm = hcfft_ComputeL2Norm(grid, testFunction, NULL, 10, NULL)
      ;
  hcfft_double L2er = hcfft_ComputeL2Error(grid, coeffs, testFunction, NULL, 10
      , NULL);
  printf("Relative L2 Error: %e\n", L2er / L2norm);

  // Free the allocated coefficients vector.
  hcfft_DestroyVector(coeffs);

  // Free the sparse grid.
  hcfft_DestroyGrid(grid);

  // Free the transform parameters.
  free(transforms);

  return 0;
}
```

Chapter 8

User-defined adaptive grid

This example demonstrates how to construct a custom adaptive grid based on a target function. The approximated function is:

$$f(x) = \frac{1}{(1+10x_0^2)(1+0.1x_1^2)(1+0.1x_2^2)(1+0.1x_3^2)}$$

This function has one very important dimension and three less important ones. - Constructing an isotropic grid for it would not be optimal. One way to achieve better approximation is to use an anisotropic grid which has more points in the first dimension. However, it is difficult to choose how many more samples to add compared to the other dimensions. Thus, it is best to do this automatically. This is the purpose of the adaptive algorithm - to construct a proper index set that takes into account the concrete function properties.

The previous example used the default adaptive algorithm in HCFFT. Here, we will customize it. The goal is to refine the grid iteratively until the L_2-error decreases under a certain threshold. Note that this way we apply a global termination criteria. The basic adaptive algorithm was using a local one.

The implementation is a bit more involved. The basic idea is that we need to specify four functions:

- weighBlock - computes the weight of a grid block. This is set to the averaged squared sum of the hierarchical coefficients. This is the same as in the previous example.
- updateBlock - this is called every time the grid changes. We use it to update the current L_2-error.
- insertBlock - a block is inserted for inspection only if the error is greater than the threshold.
- expandBlock - an inserted block is always expanded. We could also check the termination criteria here but it would be redundant, as it is done in every insertBlock call.

```
#include <stdio.h>
#include <stdlib.h>
#include "hcfft.h"

// The interpolated function.
void testFunction(const struct hcfft_Grid *grid, const int *nodeIndex, const
      hcfft_double *nodePoint, void *nodeValue, void *params)
{
  (void)nodeIndex;
  (void)params;
  (void)grid;

  // A Chebyshev transform operates on real coefficients.
  hcfft_double* result = (hcfft_double*)nodeValue;

  // Calculate the function value at this point.
  *result = 1.0 / (1 + 10 * nodePoint[0] * nodePoint[0]);
  *result *= 1.0 / (1 + 0.1 * nodePoint[1] * nodePoint[1]);
  *result *= 1.0 / (1 + 0.1 * nodePoint[2] * nodePoint[2]);
  *result *= 1.0 / (1 + 0.1 * nodePoint[3] * nodePoint[3]);
}

// Parameters that are passed to the functions expandBlock, insertBlock,
       updateBlock and weighBlock
struct AdaptiveParams
{
  struct hcfft_Vector* hcoeffs;    // Hierarchical coefficient vector for the
       current grid.
  hcfft_double eps;                // Threshold for the L2 error.
  hcfft_double L2error;            // The current L2 error.
  hcfft_NodeFunction f;            // The target function.
  void* fParams;                   // The target function's parameters.
};

// This function is called at the end of the adaptive algorithm. It is used  to
       clean up
// the parameters of the adaptive function.
void destroyParams(void* params)
{
  struct AdaptiveParams* adaptiveParams = params;
  hcfft_DestroyVector(adaptiveParams->hcoeffs);
}

// This function is called after every grid modification. We need to update the
       internal
// parameters every time the grid changes.
void updateParams(struct hcfft_Grid * grid, void* params)
{
  struct AdaptiveParams* adaptiveParams = params;

  // Reallocate the coefficient vector because the grid has grown.
  hcfft_DestroyVector(adaptiveParams->hcoeffs);
  adaptiveParams->hcoeffs = hcfft_CreateVector(grid);

  // Calculate the function at the grid nodes.
  hcfft_IterateGridNodes(grid, adaptiveParams->f, adaptiveParams->fParams,
      adaptiveParams->hcoeffs);

  // Compute the regular coefficients.
  hcfft_Transform(grid, adaptiveParams->hcoeffs);

  // Compute the current L2 error.
```

```
  adaptiveParams->L2error = hcfft_ComputeL2Error(grid, adaptiveParams->hcoeffs,
       adaptiveParams->f, adaptiveParams->fParams, 10, NULL);
  printf("Current L2 error: %e\n", adaptiveParams->L2error);

  // Transform to the hierarchical coefficients.
  hcfft_Hierarchize(grid, adaptiveParams->hcoeffs);
}

// Blocks are always expanded, the termination criteria will be implemented in
       the insertion function.
int expandBlock(struct hcfft_Grid* grid, struct hcfft_Block *block, void *
      params)
{
  (void)grid;
  (void)block;
  (void)params;

  return 1;
}

// Insert a block only if the current L2 error is higher than the threshold.
int insertBlock(struct hcfft_Grid* grid, struct hcfft_Index* index, void *
      params)
{
  (void)grid;
  (void)params;
  (void)index;

  struct AdaptiveParams * adaptiveParams = params;
  return (adaptiveParams->L2error > adaptiveParams->eps)? 1 : 0;
}

// The block weight is euqal to the averaged squared sum of hierarchical
       coefficients.
hcfft_double weighBlock(struct hcfft_Block *block, void *params)
{
  (void)params;

  struct AdaptiveParams * adaptiveParams = params;

  struct hcfft_Grid* grid = hcfft_GetBlockGrid(block);
  int start = hcfft_GetBlockStart(block);
  int length = hcfft_GetBlockLength(block);
  int k = length * length;
  return hcfft_ComputeSquareL2Norm(grid, adaptiveParams->hcoeffs, start, length
      ) / k;
}

int main()
{
  int dim = 4;              // The grid dimension.
  int initLevel = 0;        // The initial non-adaptive level.
  int maxLevel = 10;        // Maximum allowed refinement level in each
       dimension.
  hcfft_double T = 0.0;     // The hyperbolic cross parameter for the initial
       index set.
  int maxOrder = dim;       // Indices with great order than this will not be
       allowed in the grid.

  // Allocate hcfft_TransformParams struct for every dimension.
  struct hcfft_TransformParams* transforms = malloc(dim * sizeof(*transforms));
```

```
for(int d = 0; d < dim; ++d)
{
  // Initialize the structure.
  hcfft_InitTransformParams(&transforms[d]);

  // Set the transform type to Chebyshev for the second half of dimensions.
  transforms[d].type = hcfft_Chebyshev;
}

// Create default parameters for the initial index set generation.
struct hcfft_StaticIndexSet staticIndexSet = hcfft_CreateDefaultStaticIndexSet
    (initLevel, T, maxOrder);

// Create custom parameters for the adaptive index set generation functions.
struct AdaptiveParams params;
params.hcoeffs = NULL;
params.eps = 1e-4;
params.f = testFunction;
params.fParams = NULL;

// Set the custom functions for the adaptive algorithm.
struct hcfft_AdaptiveIndexSet adaptiveIndexSet;
adaptiveIndexSet.params = &params;
adaptiveIndexSet.expandBlock = expandBlock;
adaptiveIndexSet.updateParams = updateParams;
adaptiveIndexSet.weighBlock = weighBlock;
adaptiveIndexSet.insertBlock = insertBlock;
adaptiveIndexSet.destroyParams = destroyParams;

// Generate a custom dyadic grid.
struct hcfft_Grid *grid = hcfft_CreateAdaptiveGrid(dim, dim, maxLevel,
    transforms, staticIndexSet, adaptiveIndexSet);

// Print the number of nodes in the grid.
int DOF = hcfft_GetDOF(grid);
printf("The grid has %d nodes.\n", DOF);

// Allocate a vector for the coefficients.
struct hcfft_Vector *coeffs = hcfft_CreateVector(grid);

// Compute the function values at the grid points and store them in the
     vector.
hcfft_IterateGridNodes(grid, testFunction, NULL, coeffs);

// Perform a transform on the coefficient vector.
hcfft_Transform(grid, coeffs);

// Compute the relative L2 error of the interpolant.
hcfft_double L2error = hcfft_ComputeL2Error(grid, coeffs, testFunction, NULL,
     10, NULL);
printf("L2 Error: %e\n", L2error);

// Free the allocated coefficients vector.
hcfft_DestroyVector(coeffs);

// Free the sparse grid.
hcfft_DestroyGrid(grid);

// Free the transform parameters.
free(transforms);

return 0;
```

```
}
```

Chapter 9

Class Index

9.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

hcfft_AdaptiveIndexSet
: Parameters used for the generation of the grid's adaptive index set . 41

hcfft_Block
: A structure that describes a block of the sparse grid 42

hcfft_Box
: A structure that represents the domain of interpolation. The domain is simply a product of one-dimensional intervals 43

hcfft_Grid
: The main structure that controls the generation and computation on a sparse grid 43

hcfft_Index
: Represents a level index in the sparse grid 45

hcfft_Interval
: Represents an interval of real numbers 45

hcfft_StaticIndexSet
: Parameters used for the generation of the grid's static index set . . . 46

hcfft_TransformData
: Represents a 1-dimensional transform 46

hcfft_TransformParams
: Describes a transform used in a single dimension of the grid 48

hcfft_Vector
: A vector of coefficients, one per grid node 49

Chapter 10

File Index

10.1 File List

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

/home/hamaeker/codes/hcfft/src/avl_tree.h
Contains an implementation of a AVL-tree used to track the level indices of the grid . . . 51
/home/hamaeker/codes/hcfft/src/block.h
Contains various sparse block accessor function . . . 51
/home/hamaeker/codes/hcfft/src/defines.h
Basis defines and data types used thoughout the library . . . 54
/home/hamaeker/codes/hcfft/src/grid.h
Contains various sparse grid accessor function . . . 55
/home/hamaeker/codes/hcfft/src/hcfft.h
The main header file of the HCFFT libary . . . 59
/home/hamaeker/codes/hcfft/src/hcfft_internal.h
Contains essential HCFFT structures and functions that are not a part of the library's public interface . . . 68
/home/hamaeker/codes/hcfft/src/index.h
Contains various functions related to the sparse grid's level indices . 71
/home/hamaeker/codes/hcfft/src/lattice_rules.h
Rank-1 lattice rules for multidimensional quadrature . . . 75
/home/hamaeker/codes/hcfft/src/priority_queue.h
Contains an implementation of a priority queue used to track the active set of the grid . . . 77
/home/hamaeker/codes/hcfft/src/transform_data.h
Contains the definition of hcfft_TransformData, as well as related functions . . . 77
/home/hamaeker/codes/hcfft/src/utils.h
Contains various utility functions . . . 80
/home/hamaeker/codes/hcfft/src/vector.h
Contains various functions related to coefficient vectors . . . 84

/home/hamaeker/codes/hcfft/src/leja/classic_leja.h
Precomputed classic Leja points 76
/home/hamaeker/codes/hcfft/src/leja/hermite_leja.h
Precomputed Hermite Leja points for alpha = 0 and beta = 0 76
/home/hamaeker/codes/hcfft/src/leja/laguerre_leja.h
Precomputed Laguerre Leja points for alpha = 0 76
/home/hamaeker/codes/hcfft/src/leja/leja.h
Contains functions for the computation of Leja sequences 76
/home/hamaeker/codes/hcfft/src/transforms/general_transform.h
Functions related to general transforms(non-dyadic ones) 79
/home/hamaeker/codes/hcfft/src/transforms/legendre.h
Functions related to the Legendre transform 79

Chapter 11

Class Documentation

11.1 hcfft_AdaptiveIndexSet Struct Reference

Parameters used for the generation of the grid's adaptive index set.

`#include <hcfft.h>`

Public Attributes

- hcfft_TestBlock expandBlock

  A function that decides if a given level index will be expaned.
- hcfft_TestIndex insertBlock

  A function that decides if a given level index will be inserted in the grid.
- hcfft_ComputeBlockWeight weighBlock

  A function that computes the weight of a grid block.
- void * params

  Extra parameters that are passed to expandBlock, insertBlock and weighBlock.
- hcfft_UpdateParameters updateParams

  A function that updates params on every iteration of the adaptive algorithm.
- hcfft_DestroyParameters destroyParams

  A function that destroys the extra parameters params.

11.1.1 Detailed Description

Parameters used for the generation of the grid's adaptive index set.

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

- /home/hamaeker/codes/hcfft/src/hcfft.h

11.2 hcfft_Block Struct Reference

A structure that describes a block of the sparse grid.

`#include <hcfft_internal.h>`

Public Attributes

- struct hcfft_Grid * grid

  The grid to which the block belongs.
- int start

  The start of the block data in the coefficient vector.
- int length

  The number of nodes in this block.
- struct hcfft_Index * index

  The level index of the block.
- hcfft_double weight

  The block's weight.
- int * block

  The number of nodes of the block in every dimension.
- struct hcfft_Block ** forwardGridBlock

  List of all forward neighbours.
- struct hcfft_Block ** backwardGridBlock

  List of all backward neighbours.
- struct hcfft_Block ** nextPencilGridBlock

  The list of neighbour blocks which must be traversed when collecting the pencil values.
- int insertionIndex

  When was the block inserted in the grid? Only valid for adaptive grids.
- int BSBd

  The dimension along which this block was added to the grid.
- int * **pencilLength**

11.2.1 Detailed Description

A structure that describes a block of the sparse grid.

A block of the sparse grid represents the sparse grid data which correspond to a given index level.

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

- /home/hamaeker/codes/hcfft/src/hcfft_internal.h

11.3 hcfft_Box Struct Reference

A structure that represents the domain of interpolation. The domain is simply a product of one-dimensional intervals.

`#include <hcfft_internal.h>`

Public Attributes

- struct hcfft_Interval * boundary

  Defines an interval for each dimension.
- hcfft_double * kFactor

  Contains the ratio of the length of the standard interpolation interval and the actual interval that is used. One value for each dimension.
- hcfft_double * length

  Contains the length of the interval in each dimension.

11.3.1 Detailed Description

A structure that represents the domain of interpolation. The domain is simply a product of one-dimensional intervals.

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

- /home/hamaeker/codes/hcfft/src/hcfft_internal.h

11.4 hcfft_Grid Struct Reference

The main structure that controls the generation and computation on a sparse grid.

`#include <hcfft_internal.h>`

Public Attributes

- struct hcfft_TransformData * TS

  Data about the type of transformation(Fourier, Chebyshev, etc) for which the sparse grid is being created.
- struct hcfft_TransformParams * transforms

  The grid parameters.
- enum hcfft_ValueType value

  The type of data with which the grid works - real or complex.
- size_t elementSize

  The size of a single coefficient, in bytes.
- int dim

The actual dimension of the sparse grid.

- int maxDim

  The maximum allowed dimension of the sparse grid. Valid only for adaptive grids.

- int maxL

  The maximum allowed transform level in each dimension.

- int maxIndex

  The number of nodes in the grid.

- int maxGridBlocks

  The number of blocks in the grid.

- int gridBlockArraySize

  The number of blocks in the array of sparse blocks.

- struct hcfft_Block * rootBlock

  The root block i.e. the block in level index (0, 0,... 0).

- struct hcfft_Block ** gridBlockArray

  An array containing all blocks of the sparse grid.

- struct hcfft_AVLTree * nuAVLTree

  The stucture that keeps track of the level indices in the sparse grid.

- void * pencil

  Preallocated array of values used for the temporary storage of pencils during the pencil calculations.

- hcfft_double * realPencil

  Preallocated array of values, used when the grid uses both real and complex transforms.

- int pencilSize

  Number of elements in the arrays pencil and realPencil.

- hcfft_PencilOperation * operations

  Preallocated array of pointers for the temporary storage of pencil operations.

- struct hcfft_Box box

  The domain of interpolation.

- struct hcfft_StaticIndexSet staticIndexSet

  Parameters used for the creation of the static grid's index set.

- struct hcfft_AdaptiveIndexSet adaptiveIndexSet

  Parameters used for the creation of the adaptive grid's index set.

11.4.1 Detailed Description

The main structure that controls the generation and computation on a sparse grid.

11.4.2 Member Data Documentation

11.4.2.1 int hcfft_Grid::gridBlockArraySize

The number of blocks in the array of sparse blocks.

Note that this value might be greater than maxGridBlocks since some of the blocks in the array might still be unused.

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

- /home/hamaeker/codes/hcfft/src/hcfft_internal.h

11.5 hcfft_Index Struct Reference

Represents a level index in the sparse grid.

`#include <hcfft_internal.h>`

Public Attributes

- int * nu

  The elements of the index.
- int sum

  The sum of all elements.
- int max

  The maximum element.
- int order

  The number of non-zero elements.

11.5.1 Detailed Description

Represents a level index in the sparse grid.

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

- /home/hamaeker/codes/hcfft/src/hcfft_internal.h

11.6 hcfft_Interval Struct Reference

Represents an interval of real numbers.

`#include <hcfft.h>`

Public Attributes

- hcfft_double left

  The left end of the interval.
- hcfft_double right

  The right end of the interval.

11.6.1 Detailed Description

Represents an interval of real numbers.

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

- /home/hamaeker/codes/hcfft/src/hcfft.h

11.7 hcfft_StaticIndexSet Struct Reference

Parameters used for the generation of the grid's static index set.

`#include <hcfft.h>`

Public Attributes

- hcfft_TestIndex insertBlock

  A function that decides if a given level index will enter the index set.
- void * params

  Extra parameters that are passed to insertBlock.
- hcfft_DestroyParameters destroyParams

  A function that destroys the extra parameters for insertBlock.

11.7.1 Detailed Description

Parameters used for the generation of the grid's static index set.

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

- /home/hamaeker/codes/hcfft/src/hcfft.h

11.8 hcfft_TransformData Struct Reference

Represents a 1-dimensional transform.

`#include <transform_data.h>`

Public Attributes

- enum hcfft_TransformType type

  The type of transform.
- hcfft_double alpha

  An optional parameter for the basis functions.
- hcfft_double beta

  An optional parameter for the basis functions.
- void * TD

  Additional data required by the transform pencil operations.
- struct hcfft_Interval stdInterval

  The standard transform interval.
- enum hcfft_ValueType value

  The required coefficient type.
- hcfft_SetPoints setPoints

  The function which calculates the interpolation points.
- hcfft_SetGppl setGppl

  The function which calculates the grid points per level.
- int isGeneral

  Equal to 0 if this is a standard dyadic transform. Equal to 1 in all other cases.
- int maxL

  The maximum level that will be used.
- hcfft_double * points

  The interpolation points.
- int * gppl

  Number of grid points per level.
- void(* DestroyTD)(void *TD)

  Frees the allocated resources needed by TD.
- void *(* InitTD)(const struct hcfft_TransformData *TS)

  Initializes the transform data.
- hcfft_PencilOperation PerformPencilTransformation

  Pencil operation that performs a transform.
- hcfft_PencilOperation PerformPencilInverseTransformation

  Pencil operation that performs an inverse transform.
- hcfft_PencilOperation PerformPencilDehierar

  Pencil operation that performs dehierarchization.
- hcfft_PencilOperation PerformPencilHierar

  Pencil operation that performs hierarchization.
- hcfft_PencilOperation DerivativeCoefficients

  Pencil operation that converts the coefficients of the function to the coefficients of the derivative of the function.
- hcfft_PencilOperation LaplaceCoefficients

  Pencil operation that converts the coefficients of the function to the coefficients of the Laplacian of the function.
- hcfft_ArrayBaseFunction EvalArrayBaseFunction

  A function which performs multiple basis function evaluations.

11.8.1 Detailed Description

Represents a 1-dimensional transform.

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

- /home/hamaeker/codes/hcfft/src/transform_data.h

11.9 hcfft_TransformParams Struct Reference

Describes a transform used in a single dimension of the grid.

`#include <hcfft.h>`

Public Attributes

- enum hcfft_TransformType type

  The type of transform.
- struct hcfft_Interval * boundary

  The domain on which the transform is executed. Set to NULL to use the default domain.
- hcfft_SetGppl gpplFunc

  Computes the number of grid points per level for this transform. Set to NULL to use the default number of grid points per level.
- hcfft_SetPoints pointsFunc

  Computes the interpolation points for this dimension. Set to NULL to use the default interpolation points.
- hcfft_ArrayBaseFunction arrayBaseFunction

  Computes several basis functions at the same point.
- enum hcfft_ValueType valueType

  The standard type of data on which the tranform works(real or complex).
- hcfft_double alpha

  Additional parameter for the basis functions.
- hcfft_double beta

  Additional parameter for the basis functions.

11.9.1 Detailed Description

Describes a transform used in a single dimension of the grid.

Warning

Every instance of this structure should always be initialzied with hcfft_InitTransform-Params before usage.

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

- /home/hamaeker/codes/hcfft/src/hcfft.h

11.10 hcfft_Vector Struct Reference

A vector of coefficients, one per grid node.

`#include <hcfft_internal.h>`

Public Attributes

- void * data

  The chunk of memory in which the coefficients are saved.

11.10.1 Detailed Description

A vector of coefficients, one per grid node.

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

- /home/hamaeker/codes/hcfft/src/hcfft_internal.h

Chapter 12

File Documentation

12.1 /home/hamaeker/codes/hcfft/src/avl_tree.h File Reference

Contains an implementation of a AVL-tree used to track the level indices of the grid.

12.1.1 Detailed Description

Contains an implementation of a AVL-tree used to track the level indices of the grid.

12.2 /home/hamaeker/codes/hcfft/src/block.h File Reference

Contains various sparse block accessor function.

Typedefs

- typedef int(∗ hcfft_TestBlock)(struct hcfft_Grid ∗grid, struct hcfft_Block ∗block, void ∗params)

  A predicate function executed on a grid block.
- typedef hcfft_double(∗ hcfft_ComputeBlockWeight)(struct hcfft_Block ∗block, void ∗params)

  Computes the weight of a sparse block.

Functions

- struct hcfft_Grid ∗ hcfft_GetBlockGrid (const struct hcfft_Block ∗block)

  Get the grid to which the block belongs.
- int hcfft_GetBlockStart (const struct hcfft_Block ∗block)

  Get the index of the first coefficient that belongs to this block.

- int hcfft_GetBlockLength (const struct hcfft_Block *block)

  Get the number of grid points in the block.
- struct hcfft_Index * hcfft_GetBlockIndex (const struct hcfft_Block *block)

  Get the level index of the block.
- hcfft_double hcfft_GetBlockWeight (const struct hcfft_Block *block)

  Get the weight of the block.

12.2.1 Detailed Description

Contains various sparse block accessor function.

12.2.2 Typedef Documentation

12.2.2.1 typedef hcfft_double(* hcfft_ComputeBlockWeight)(struct hcfft_Block *block, void *params)

Computes the weight of a sparse block.

Parameters

block	The grid block.
params	User-defined parameters for the function.

Returns

The computed block weight.

12.2.2.2 typedef int(* hcfft_TestBlock)(struct hcfft_Grid *grid, struct hcfft_Block *block, void *params)

A predicate function executed on a grid block.

Parameters

grid	The grid.
block	The grid block.
params	User-defined parameters for the function.

Returns

1 if the block passes the test and 0 otherwise.

12.2.3 Function Documentation

12.2.3.1 struct hcfft_Grid* hcfft_GetBlockGrid (const struct hcfft_Block * *block*) [read]

Get the grid to which the block belongs.

Parameters

block	The grid block.

Returns

The grid to which the block belongs.

12.2.3.2 struct hcfft_Index* hcfft_GetBlockIndex (const struct hcfft_Block * *block*) [read]

Get the level index of the block.

Parameters

block	The grid block.

Returns

A pointer to the interval level index of the block.

12.2.3.3 int hcfft_GetBlockLength (const struct hcfft_Block * *block*)

Get the number of grid points in the block.

Parameters

block	The grid block.

Returns

The number of grid points in the block.

12.2.3.4 int hcfft_GetBlockStart (const struct hcfft_Block * *block*)

Get the index of the first coefficient that belongs to this block.

A sparse grid works on coefficient vectors which have one entry for every grid node. The grid, and consequently the coefficient vectors, are partitioned into the grid blocks. The coefficients that correspond to a block are stored linearly in the entire vector. This function returns the index of the first such coefficient.

Parameters

block	The grid block.

Returns

The index of the first coefficient that belongs to this block.

12.2.3.5 hcfft_double hcfft_GetBlockWeight (const struct hcfft_Block * *block*)

Get the weight of the block.

Parameters

block	The block.

Returns

The block's weight.

12.3 /home/hamaeker/codes/hcfft/src/defines.h File Reference

Basis defines and data types used thoughout the library.

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

Defines

- #define **hcfft_double** double
- #define **hcfft_complex** fftw_complex
- #define **hcfft_sin** sin
- #define **hcfft_cos** cos
- #define **hcfft_acos** acos
- #define **hcfft_exp** exp
- #define **hcfft_pow** pow
- #define **hcfft_sqrt** sqrt
- #define **hcfft_fabs** fabs
- #define **hcfft_plan_dft_1d** fftw_plan_dft_1d
- #define **hcfft_plan_r2r_1d** fftw_plan_r2r_1d
- #define **hcfftw_plan** fftw_plan
- #define **hcfft_execute** fftw_execute
- #define **hcfft_destroy_plan** fftw_destroy_plan
- #define **hcfft_malloc** fftw_malloc
- #define **hcfft_free** fftw_free
- #define **hcfftw_cleanup** fftw_cleanup
- #define **hcfft_pi** 3.14159265358979323846
- #define **hcfft_infinity** DBL_MAX

Typedefs

- typedef void(∗ hcfft_NodeFunction)(const struct hcfft_Grid ∗grid, const int ∗nodeIndex, const hcfft_double ∗nodePoint, void ∗nodeValue, void ∗params)

  A function that is executed on each grid node.

Enumerations

- enum hcfft_ValueType { **hcfft_UndefinedValue**, **hcfft_RealValue**, **hcfft_ComplexValue** }

  The type of data stored in the coefficients - real or complex.

12.3.1 Detailed Description

Basis defines and data types used thoughout the library.

12.3.2 Typedef Documentation

12.3.2.1 typedef void(∗ hcfft_NodeFunction)(const struct hcfft_Grid ∗grid, const int ∗nodeIndex, const hcfft_double ∗nodePoint, void ∗nodeValue, void ∗params)

A function that is executed on each grid node.

Parameters

grid	The grid.
nodeIndex	The index of the current node.
nodePoint	The position of the current node.
nodeValue	The result of the function at the current node.
params	Additional user-defined parameters.

12.4 /home/hamaeker/codes/hcfft/src/grid.h File Reference

Contains various sparse grid accessor function.

Functions

- int hcfft_GetDimension (const struct hcfft_Grid ∗grid)

  Get the dimension of the sparse grid.
- int hcfft_GetNumberOfBlocks (const struct hcfft_Grid ∗grid)

  Get the number of blocks in the sparse grid.
- int hcfft_GetDOF (const struct hcfft_Grid ∗grid)

  Get the degrees of freedom in the sparse grid(i.e. the number of nodes).

- hcfft_double hcfft_GetDomainLength (const struct hcfft_Grid *grid, int d)

  Get the domain length in the given direction.
- struct hcfft_Interval hcfft_GetBoundary (const struct hcfft_Grid *grid, int d)

  Get the domain interval in the given direction.
- enum hcfft_ValueType hcfft_GetValueType (const struct hcfft_Grid *grid)

  Get the type of data used by the sparse grid(real or complex).
- size_t hcfft_GetElementSize (struct hcfft_Grid *grid)

  Get the size in bytes of a single coefficient used by the sparse grid.
- int hcfft_GetMaxLevel (const struct hcfft_Grid *grid)

  Get the maximum allowed block level in each grid dimension. The elements of all indices in the grid are limited by this number.
- struct hcfft_Block * hcfft_GetBlock (const struct hcfft_Grid *grid, int idx)

  Get the block with the given local index.

12.4.1 Detailed Description

Contains various sparse grid accessor function.

12.4.2 Function Documentation

12.4.2.1 struct hcfft_Block* hcfft_GetBlock (const struct hcfft_Grid * *grid,* int *idx*) [read]

Get the block with the given local index.

Parameters

grid	The grid.
idx	The block's local index.

Returns

A pointer to the block with the given local index.

12.4.2.2 struct hcfft_Interval hcfft_GetBoundary (const struct hcfft_Grid * *grid,* int *d*) [read]

Get the domain interval in the given direction.

Parameters

grid	The grid.
d	The requested dimension.

Returns

The domain interval in direction *d*.

12.4.2.3 int hcfft_GetDimension (const struct hcfft_Grid * *grid*)

Get the dimension of the sparse grid.

Parameters

grid	The grid.

Returns

The dimension of the sparse grid.

12.4.2.4 int hcfft_GetDOF (const struct hcfft_Grid * *grid*)

Get the degrees of freedom in the sparse grid(i.e. the number of nodes).

Parameters

grid	The grid.

Returns

The degrees of freedom in the sparse grid.

12.4.2.5 hcfft_double hcfft_GetDomainLength (const struct hcfft_Grid * *grid,* int *d*)

Get the domain length in the given direction.

Parameters

grid	The grid.
d	The requested dimension.

Returns

The length of the interpolation domain in direction *d*.

12.4.2.6 size_t hcfft_GetElementSize (struct hcfft_Grid * *grid*)

Get the size in bytes of a single coefficient used by the sparse grid.

Parameters

grid	The grid.

Returns

Returns sizeof(hcfft_double) if the grid works on real numbers, and sizeof(hcfft_complex) if the grid works on complex numbers.

12.4.2.7 int hcfft_GetMaxLevel (const struct hcfft_Grid * *grid*)

Get the maximum allowed block level in each grid dimension. The elements of all indices in the grid are limited by this number.

Parameters

grid	The grid.

Returns

The maximum allowed block level.

12.4.2.8 int hcfft_GetNumberOfBlocks (const struct hcfft_Grid * *grid*)

Get the number of blocks in the sparse grid.

Parameters

grid	The grid.

Returns

The number of blocks in the sparse grid.

12.4.2.9 enum hcfft_ValueType hcfft_GetValueType (const struct hcfft_Grid * *grid*)

Get the type of data used by the sparse grid(real or complex).

Parameters

grid	The grid.

Returns

The type of data used by the sparse grid.

12.5 /home/hamaeker/codes/hcfft/src/hcfft.h File Reference

The main header file of the HCFFT libary.

`#include <stdio.h> #include "config.h" #include "defines.h" #include "index.h" #include "vector.h" #include "block.h" #include "grid.h" #include "utils.h"`

Classes

- struct hcfft_Interval

  Represents an interval of real numbers.
- struct hcfft_StaticIndexSet

  Parameters used for the generation of the grid's static index set.
- struct hcfft_AdaptiveIndexSet

  Parameters used for the generation of the grid's adaptive index set.
- struct hcfft_TransformParams

  Describes a transform used in a single dimension of the grid.

Typedefs

- typedef void(∗ hcfft_ArrayBaseFunction)(int N, hcfft_double x, void ∗result, hcfft_double alpha, hcfft_double beta)

  Computes multiple basis functions at the same point.
- typedef void(∗ hcfft_SetGppl)(int ∗gppl, int L)

  Sets the number of grid points per level.
- typedef void(∗ hcfft_SetPoints)(hcfft_double ∗points, int n, hcfft_double alpha, hcfft_double beta)

  Sets the first (n + 1) grid points.
- typedef void(∗ hcfft_DestroyParameters)(void ∗params)

  Destroys the given user-defined parameters.
- typedef void(∗ hcfft_UpdateParameters)(struct hcfft_Grid ∗grid, void ∗params)

  Initializes or updates the user-defined parameters used for adaptive grid generation.

Enumerations

- enum hcfft_TransformType { hcfft_UndefinedTransform, hcfft_CustomTransform, hcfft_Fourier, hcfft_RealFourier, hcfft_DCT, hcfft_DST, hcfft_Chebyshev, hcfft_Legendre, hcfft_Hermite, hcfft_Laguerre, hcfft_Jacobi }

  The types of available transforms in the library.
- enum hcfft_BlockWeightType { hcfft_HierarchicalSquareSum, hcfft_CoefficientsSquareSum, hcfft_AverageHierarchicalSquareSum, hcfft_AverageCoefficientsSquareSum }

  The type of default block weight calcultion.

Functions

- struct hcfft_Grid * hcfft_CreateGrid (int dimension, int maxLevel, struct hcfft_TransformParams *transforms, struct hcfft_StaticIndexSet staticIndexSet)

  Create a general static grid.
- struct hcfft_Grid * hcfft_CreateAdaptiveGrid (int dimension, int maxDimension, int maxLevel, struct hcfft_TransformParams *transforms, struct hcfft_StaticIndexSet staticIndexSet, struct hcfft_AdaptiveIndexSet adaptiveIndexSet)

  Create a general adaptive grid.
- void hcfft_DestroyGrid (struct hcfft_Grid *grid)

  Destroy the grid and free all allocated resources related to it.
- void hcfft_CalculateDerivativeCoefficients (struct hcfft_Grid *grid, struct hcfft_Vector *coeffs, int d)

  Calculate an approximation of the coefficients of the derivative of the real space function.
- void hcfft_CalculateLaplaceCoefficients (struct hcfft_Grid *grid, struct hcfft_Vector *coeffs)

  Calculate an approximation of the regular coefficients of the Laplacian of the function.
- void hcfft_IterateGridNodes (const struct hcfft_Grid *grid, hcfft_NodeFunction f, void *fParams, struct hcfft_Vector *coeffs)

  Execute the function f on every node of the grid.
- void hcfft_Transform (struct hcfft_Grid *grid, struct hcfft_Vector *coeffs)

  Perform a transformation on the given vector.
- void hcfft_InverseTransform (struct hcfft_Grid *grid, struct hcfft_Vector *coeffs)

  Perform an inverse transformation on the given vector.
- void hcfft_SparseTransform (struct hcfft_Grid *grid, struct hcfft_Vector *coeffs)

  Perform a hierarchical transformation on the given vector.
- void hcfft_SparseInverseTransform (struct hcfft_Grid *grid, struct hcfft_Vector *coeffs)

  Perform an inverse hierarchical transformation on the given vector.
- void hcfft_Dehierarchize (struct hcfft_Grid *grid, struct hcfft_Vector *coeffs)

  Perform a dehierarchization on the values in the given vector.
- void hcfft_Hierarchize (struct hcfft_Grid *grid, struct hcfft_Vector *coeffs)

  Perform a hierarchization on the values in the given vector.
- void hcfft_ComputeInterpolatedValues (struct hcfft_Grid *grid, struct hcfft_Vector *coeffs, int numPoints, const hcfft_double *points, void *values)

  Compute the interpolant at the given points.
- struct hcfft_StaticIndexSet hcfft_CreateDefaultStaticIndexSet (int L, double T, int maxOrder)

  Create the parameters for default static index set generation.
- struct hcfft_AdaptiveIndexSet hcfft_CreateDefaultAdaptiveIndexSet (hcfft_NodeFunction f, void *fParams, double epsilon, enum hcfft_BlockWeightType weightType, int maxNodes, int maxOrder)

  Create the parameters for default adaptive index set generation.
- void hcfft_InitStaticIndexSet (struct hcfft_StaticIndexSet *data)

Initialize an empty static index set.

- void hcfft_InitAdaptiveIndexSet (struct hcfft_AdaptiveIndexSet *data)

  Initialize an empty adaptive index set.

- void hcfft_InitTransformParams (struct hcfft_TransformParams *params)

  Initialize an empty transformation params.

- void hcfft_SetGpplPlus1 (int *gppl, int L)

  Sets the number of grid points per level to the index of the level plus 1. After the execution of the function gppl contains the numbers 1, 2, 3... L + 1.

12.5.1 Detailed Description

The main header file of the HCFFT libary.

12.5.2 Typedef Documentation

12.5.2.1 typedef void(* hcfft_ArrayBaseFunction)(int N, hcfft_double x, void *result, hcfft_double alpha, hcfft_double beta)

Computes multiple basis functions at the same point.

Computes the basis functions with indices 0, 1 ... N-1.

Parameters

N	The number of basis functions that must be computed.
x	The point at which the basis functions should be computed.
result	The array where the computation results will be written.
alpha	Additional parameter for the basis function.
beta	Additional parameter for the basis function.

12.5.2.2 typedef void(* hcfft_DestroyParameters)(void *params)

Destroys the given user-defined parameters.

Parameters

params	The user-defined parameters.

12.5.2.3 typedef void(* hcfft_SetGppl)(int *gppl, int L)

Sets the number of grid points per level.

Parameters

gppl	An array of size (L + 1) holding the number of grid points per level for levels 0, 1... L.
L	The maximum level stored in *gppl*.

12.5.2.4 typedef void(∗ hcfft_SetPoints)(hcfft_double ∗points, int n, hcfft_double alpha, hcfft_double beta)

Sets the first (n + 1) grid points.

Parameters

points	An array of size (n + 1) that will contain the grid points.
n	The maximum grid point index.
alpha	Additional parameter for the transform.
beta	Additional parameter for the transform.

12.5.2.5 typedef void(∗ hcfft_UpdateParameters)(struct hcfft_Grid ∗grid, void ∗params)

Initializes or updates the user-defined parameters used for adaptive grid generation.

Parameters

grid	The grid.
params	The user-defined parameters.

12.5.3 Enumeration Type Documentation

12.5.3.1 enum hcfft_BlockWeightType

The type of default block weight calcultion.

Enumerator:

hcfft_HierarchicalSquareSum Squared sum of hierarchical coefficients in the block.

hcfft_CoefficientsSquareSum Squared sum of regular coefficients in the block.

hcfft_AverageHierarchicalSquareSum Averaged squared sum of hierarchical coefficients in the block.

hcfft_AverageCoefficientsSquareSum Averaged squared sum of regular coefficients in the block.

12.5.3.2 enum hcfft_TransformType

The types of available transforms in the library.

Enumerator:

hcfft_UndefinedTransform Undefined transorm type. For internal usage only.

hcfft_CustomTransform User-defined transform.

hcfft_Fourier Complex Fourier transform.

hcfft_RealFourier Real Fourier transform.

hcfft_DCT Discrete cosine transform.

hcfft_DST Discrete sine transform.

hcfft_Chebyshev Chebyshev transform.

hcfft_Legendre Legendre transform.

hcfft_Hermite Generalized Hermite transform.

hcfft_Laguerre Laguerre transform.

hcfft_Jacobi Generalized Jacobi transform.

12.5.4 Function Documentation

12.5.4.1 void hcfft_CalculateDerivativeCoefficients (struct hcfft_Grid * *grid,* struct hcfft_Vector * *coeffs,* int *d*)

Calculate an approximation of the coefficients of the derivative of the real space function.

Parameters

grid	The grid.
coeffs	Initially contains the coefficients of the real space values in the space of basis functions. At the end, it contains the coefficients of the derivative of the real space values in the space of basis functions.
d	Direction along which the derivative is taken.

12.5.4.2 void hcfft_CalculateLaplaceCoefficients (struct hcfft_Grid * *grid,* struct hcfft_Vector * *coeffs*)

Calculate an approximation of the regular coefficients of the Laplacian of the function.

Parameters

grid	The grid.
coeffs	Initially contains the regular coefficients of the function. At the end it contains the regular coefficients of the Laplacian of the function.

12.5.4.3 void hcfft_ComputeInterpolatedValues (struct hcfft_Grid * *grid,* struct hcfft_Vector * *coeffs,* int *numPoints,* const hcfft_double * *points,* void * *values*)

Compute the interpolant at the given points.

Parameters

grid	The grid.
coeffs	The regular coefficients.
numPoints	The number of points.
points	The points at which the interpolation will be computed. The first point is at indices 0..D-1, the second is at D..2∗D-1 and so on.
values	Contains the computed interpolated values.

12.5.4.4 struct hcfft_Grid∗ hcfft_CreateAdaptiveGrid (int *dimension,* int *maxDimension,* int *maxLevel,* struct hcfft_TransformParams ∗ *transforms,* struct hcfft_StaticIndexSet *staticIndexSet,* struct hcfft_AdaptiveIndexSet *adaptiveIndexSet*) [read]

Create a general adaptive grid.

Parameters

dimension	The initial dimension of the grid.
maxDimension	The maximum possible dimension of the grid. No more dimensions will be added, even if the adaptive algorithm decides that it should.
maxLevel	The maximum allowed index level in the index set. No indices from higher levels will be inserted, even if the static index parameters allow it.
transforms	The transform descriptors for each dimension.
staticIndexSet	The parameters for the initial index set generation.
adaptiveIndexSet	The parameters for the adaptive index set generation.

Returns

The generated grid.

12.5.4.5 struct hcfft_AdaptiveIndexSet hcfft_CreateDefaultAdaptiveIndexSet (hcfft_NodeFunction *f,* void ∗ *fParams,* double *epsilon,* enum hcfft_BlockWeightType *weightType,* int *maxNodes,* int *maxOrder*) [read]

Create the parameters for default adaptive index set generation.

Parameters

f	The function for which the grid is begin generated.
fParams	Parameters passed to *f*.
epsilon	Only blocks with weigth $>$ *epsilon* are expanded.
weightType	Default block weight type.
maxNodes	The grid generation will terminate when this number is exceeded.
maxOrder	Only indices with at most *maxOrder* non-zero elements will be inserted in the grid.

12.5.4.6 struct hcfft_StaticIndexSet hcfft_CreateDefaultStaticIndexSet (int *L*, double *T*, int *maxOrder*) [read]

Create the parameters for default static index set generation.

Parameters

L	Only indices l that satisfy $\|l\|_1 - T\|l\|_infty <= (1-T)L$ will be inserted in the grid.
T	Only indices l that satisfy $\|l\|_1 - T\|l\|_infty <= (1-T)L$ will be inserted in the grid.
maxOrder	Only indices with at most *maxOrder* non-zero elements will be inserted in the grid.

12.5.4.7 struct hcfft_Grid* hcfft_CreateGrid (int *dimension*, int *maxLevel*, struct hcfft_TransformParams * *transforms*, struct hcfft_StaticIndexSet *staticIndexSet*) [read]

Create a general static grid.

Parameters

dimension	The dimension of the grid.
maxLevel	The maximum allowed index level in the index set. No indices from higher levels will be inserted, even if the static index parameters allow it.
transforms	The transform descriptors for each dimension.
staticIndexSet	The parameters for the index set generation.

Returns

The generated grid.

12.5.4.8 void hcfft_Dehierarchize (struct hcfft_Grid * *grid*, struct hcfft_Vector * *coeffs*)

Perform a dehierarchization on the values in the given vector.

Parameters

grid	The grid.
coeffs	Initially contains the hierachical coefficients. At the end it contains the regular coefficients.

12.5.4.9 void hcfft_DestroyGrid (struct hcfft_Grid * *grid*)

Destroy the grid and free all allocated resources related to it.

Parameters

grid	The grid that must be destroyed.

12.5.4.10 void hcfft_Hierarchize (struct hcfft_Grid * *grid,* struct hcfft_Vector * *coeffs*)

Perform a hierarchization on the values in the given vector.

Parameters

grid	The grid.
coeffs	Initially contains the regular coefficients. At the end it contains the hierarchical coefficients.

12.5.4.11 void hcfft_InitAdaptiveIndexSet (struct hcfft_AdaptiveIndexSet * *data*)

Initialize an empty adaptive index set.

Parameters

data	The index set that is being initialized.

12.5.4.12 void hcfft_InitStaticIndexSet (struct hcfft_StaticIndexSet * *data*)

Initialize an empty static index set.

Parameters

data	The index set that is being initialized.

12.5.4.13 void hcfft_InitTransformParams (struct hcfft_TransformParams * *params*)

Initialize an empty transformation params.

Parameters

params	The params that are being initialized.

12.5.4.14 void hcfft_InverseTransform (struct hcfft_Grid * *grid*, struct hcfft_Vector * *coeffs*)

Perform an inverse transformation on the given vector.

Converts the values in the vector from regular coefficient space to real space.

Parameters

grid	The grid.
coeffs	Initially contains the regular coefficients. At the end it contains the real space values.

12.5.4.15 void hcfft_IterateGridNodes (const struct hcfft_Grid * *grid*, hcfft_NodeFunction *f*, void * *fParams*, struct hcfft_Vector * *coeffs*)

Execute the function *f* on every node of the grid.

Parameters

grid	The grid.
f	The function that is being executed on every grid node
fparams	Parameters passed to *f*.
coeffs	Contains one value for each node of the grid. This value is being passed to the executed function and can be used for both input and output operations.

12.5.4.16 void hcfft_SetGpplPlus1 (int * *gppl*, int *L*)

Sets the number of grid points per level to the index of the level plus 1. After the execution of the function *gppl* contains the numbers 1, 2, 3... L + 1.

Parameters

gppl	An array of size (L + 1) holding the number of grid points per level for levels 0, 1... L.
L	The maximum level stored in *gppl*.

12.5.4.17 void hcfft_SparseInverseTransform (struct hcfft_Grid * *grid*, struct hcfft_Vector * *coeffs*)

Perform an inverse hierarchical transformation on the given vector.

Converts the values in the vector from hierarchical coefficient space to real space.

Parameters

grid	The grid.
coeffs	Initially contains the hierarchical coefficients. At the end it contains the real space values.

12.5.4.18 void hcfft_SparseTransform (struct hcfft_Grid * *grid*, struct hcfft_Vector * *coeffs*)

Perform a hierarchical transformation on the given vector.

Converts the values in the given vector from real space to hierarchical coefficient space.

Parameters

grid	The grid.
coeffs	Initially contains the real space values. At the end it contains the hierarchical coefficients.

12.5.4.19 void hcfft_Transform (struct hcfft_Grid * *grid*, struct hcfft_Vector * *coeffs*)

Perform a transformation on the given vector.

Converts the values in the given vector from real space to regular coefficient space.

Parameters

grid	The grid.
coeffs	Initially contains the real space values. At the end it contains the regular coefficients.

12.6 /home/hamaeker/codes/hcfft/src/hcfft_internal.h File Reference

Contains essential HCFFT structures and functions that are not a part of the library's public interface.

```
#include "hcfft.h"
```

Classes

- struct hcfft_Vector

  A vector of coefficients, one per grid node.
- struct hcfft_Index

  Represents a level index in the sparse grid.
- struct hcfft_Block

  A structure that describes a block of the sparse grid.

- struct hcfft_Box

  A structure that represents the domain of interpolation. The domain is simply a product of one-dimensional intervals.
- struct hcfft_Grid

  The main structure that controls the generation and computation on a sparse grid.

Typedefs

- typedef void(∗ **hcfft_PencilOperation**)(void ∗TD, const int ∗gppl, const int D, void ∗Pencil, const int dir, const int ∗const PencilSize, const int l)

Functions

- const hcfft_double ∗ hcfft_GetkFactor (const struct hcfft_Grid ∗grid)

  Get the domain scaling factors in each dimension.
- hcfft_double ∗ hcfft_GetPoints (const struct hcfft_Grid ∗grid, int d)

  Get the grid points in the given dimension.
- int ∗ hcfft_GetGppl (const struct hcfft_Grid ∗grid, int d)

  Get the grid points per level for the given dimension.
- struct hcfft_Interval hcfft_GetStandardInterval (const struct hcfft_Grid ∗grid, int d)

  Get the standard domain for the given dimension.
- enum hcfft_ValueType hcfft_chooseValueType (const struct hcfft_TransformParams ∗transforms, int D)

  Determine the type of data that is required by the D-dimensional mixed transform.

12.6.1 Detailed Description

Contains essential HCFFT structures and functions that are not a part of the library's public interface.

12.6.2 Function Documentation

12.6.2.1 enum hcfft_ValueType hcfft_chooseValueType (const struct hcfft_TransformParams ∗ *transforms*, int *D*)

Determine the type of data that is required by the D-dimensional mixed transform.

If at least one of the transforms works standardly on complex numbers then the entire D-dimensional transform will also work on complex numbers. Otherwise, the data type will be real.

Parameters

transforms	The list of transform parameters.
D	The dimension.

Returns

The data type.

12.6.2.2 int* hcfft_GetGppl (const struct hcfft_Grid * *grid,* int *d*)

Get the grid points per level for the given dimension.

Parameters

grid	The grid.
d	The dimension.

Returns

An array that contains the grid poins per level in the given dimension.

12.6.2.3 const hcfft_double* hcfft_GetkFactor (const struct hcfft_Grid * *grid*)

Get the domain scaling factors in each dimension.

Parameters

grid	The grid.

Returns

The array of domain scaling factors in each dimension.

12.6.2.4 hcfft_double* hcfft_GetPoints (const struct hcfft_Grid * *grid,* int *d*)

Get the grid points in the given dimension.

Parameters

grid	The grid.
d	The dimension.

Returns

An array that contains the grid points in the given dimension.

12.6.2.5 struct hcfft_Interval hcfft_GetStandardInterval (const struct hcfft_Grid * *grid,* int *d*) [read]

Get the standard domain for the given dimension.

Parameters

grid	The grid.
d	The dimension.

Returns

The two endpoints of the standard domain in the given dimension.

12.7 /home/hamaeker/codes/hcfft/src/index.h File Reference

Contains various functions related to the sparse grid's level indices.

Typedefs

- typedef int(* hcfft_TestIndex)(struct hcfft_Grid *grid, struct hcfft_Index *index, void *params)

  A predicate function executed on a level index.

Functions

- struct hcfft_Index * hcfft_CreateEmptyIndex (struct hcfft_Grid *grid)

  Create empty level index. The result should later be destroyed with hcfft_DestroyIndex.
- void hcfft_DestroyIndex (struct hcfft_Index *index)

  Destroy a level index that was created with hcfft_CreateEmptyIndex.
- void hcfft_CopyIndex (struct hcfft_Grid *grid, struct hcfft_Index *dest, struct hcfft_Index *src)

  Copy the level index src to dest.
- void hcfft_IncrementIndex (struct hcfft_Grid *grid, struct hcfft_Index *index, int d)

  Increment the given index in the given direction.
- void hcfft_DecrementIndex (struct hcfft_Grid *grid, struct hcfft_Index *index, int d)

  Decrement the given index in the given direction.
- int hcfft_GetIndexSum (struct hcfft_Grid *grid, struct hcfft_Index *index)

Get the sum of the index elements, i.e. the l-1 norm.

- int hcfft_GetIndexOrder (struct hcfft_Grid *grid, struct hcfft_Index *index)

  Get the number of non-zero index elements.
- int hcfft_GetIndexMax (struct hcfft_Grid *grid, struct hcfft_Index *index)

  Get the maximum index element, i.e. the l-infinity norm.
- int hcfft_GetIndexElement (struct hcfft_Grid *grid, struct hcfft_Index *index, int d)

  Get the index element in the given direction.
- void hcfft_SetIndexElement (struct hcfft_Grid *grid, struct hcfft_Index *index, int d, int v)

  Set the index element in the given direction.
- int hcfft_GetIndexDim (struct hcfft_Grid *grid, struct hcfft_Index *index)

  Get the effective dimension of the index.

12.7.1 Detailed Description

Contains various functions related to the sparse grid's level indices.

12.7.2 Typedef Documentation

12.7.2.1 typedef int(* hcfft_TestIndex)(struct hcfft_Grid *grid, struct hcfft_Index *index, void *params)

A predicate function executed on a level index.

Parameters

grid	The grid.
index	The level index.
params	User-defined parameters for the function.

Returns

1 if the index passes the test and 0 otherwise.

12.7.3 Function Documentation

12.7.3.1 void hcfft_CopyIndex (struct hcfft_Grid * *grid*, struct hcfft_Index * *dest*, struct hcfft_Index * *src*)

Copy the level index *src* to *dest*.

Parameters

grid	The sparse grid.
dest	The level index that is being written.
src	The level index that is being copied.

12.7.3.2 struct hcfft_Index* hcfft_CreateEmptyIndex (struct hcfft_Grid * *grid*) [read]

Create empty level index. The result should later be destroyed with hcfft_DestroyIndex.

Parameters

grid	The grid.

Returns

The newly created empty level index.

12.7.3.3 void hcfft_DecrementIndex (struct hcfft_Grid * *grid,* struct hcfft_Index * *index,* int *d*)

Decrement the given index in the given direction.

Parameters

grid	The sparse grid.
index	The level index that is being decremented.
d	The direction in which the index is decremented.

12.7.3.4 void hcfft_DestroyIndex (struct hcfft_Index * *index*)

Destroy a level index that was created with hcfft_CreateEmptyIndex.

Parameters

index	The level index to destroy.

Returns

The newly created empty level index.

12.7.3.5 int hcfft_GetIndexDim (struct hcfft_Grid * *grid,* struct hcfft_Index * *index*)

Get the effective dimension of the index.

Parameters

grid	The sparse grid.
index	The level index.

Returns

The dimension of the index.

12.7.3.6 int hcfft_GetIndexElement (struct hcfft_Grid * *grid*, struct hcfft_Index * *index*, int *d*)

Get the index element in the given direction.

Parameters

grid	The sparse grid.
index	The level index.
d	The direction.

Returns

The index element in the given direction.

12.7.3.7 int hcfft_GetIndexMax (struct hcfft_Grid * *grid*, struct hcfft_Index * *index*)

Get the maximum index element, i.e. the l-infinity norm.

Parameters

grid	The sparse grid.
index	The level index.

Returns

The maximum index element.

12.7.3.8 int hcfft_GetIndexOrder (struct hcfft_Grid * *grid*, struct hcfft_Index * *index*)

Get the number of non-zero index elements.

Parameters

grid	The sparse grid.
index	The level index.

Returns

The number of non-zero index elements.

12.7.3.9 int hcfft_GetIndexSum (struct hcfft_Grid * grid, struct hcfft_Index * index)

Get the sum of the index elements, i.e. the l-1 norm.

Parameters

grid	The sparse grid.
index	The level index.

Returns

The sum of the index elements.

12.7.3.10 void hcfft_IncrementIndex (struct hcfft_Grid * grid, struct hcfft_Index * index, int d)

Increment the given index in the given direction.

Parameters

grid	The sparse grid.
index	The level index that is being incremented.
d	The direction in which the index is incremented.

12.7.3.11 void hcfft_SetIndexElement (struct hcfft_Grid * grid, struct hcfft_Index * index, int d, int v)

Set the index element in the given direction.

Parameters

grid	The sparse grid.
index	The level index.
d	The direction.
v	The new value.

12.8 /home/hamaeker/codes/hcfft/src/lattice_rules.h File Reference

Rank-1 lattice rules for multidimensional quadrature.

12.8.1 Detailed Description

Rank-1 lattice rules for multidimensional quadrature.

12.9 /home/hamaeker/codes/hcfft/src/leja/classic_leja.h File - Reference

Precomputed classic Leja points.

12.9.1 Detailed Description

Precomputed classic Leja points.

12.10 /home/hamaeker/codes/hcfft/src/leja/hermite_leja.h File - Reference

Precomputed Hermite Leja points for alpha = 0 and beta = 0.

12.10.1 Detailed Description

Precomputed Hermite Leja points for alpha = 0 and beta = 0.

12.11 /home/hamaeker/codes/hcfft/src/leja/laguerre_leja.h File - Reference

Precomputed Laguerre Leja points for alpha = 0.

12.11.1 Detailed Description

Precomputed Laguerre Leja points for alpha = 0.

12.12 /home/hamaeker/codes/hcfft/src/leja/leja.h File Reference

Contains functions for the computation of Leja sequences.

```
#include "hcfft.h"
```

Functions

- void hcfft_GenerateLejaPoints (enum hcfft_TransformType transformType, hcfft_-double *points, int n, double alpha, double beta)

  Computes a Leja sequence for the given transform type.

12.12.1 Detailed Description

Contains functions for the computation of Leja sequences.

12.12.2 Function Documentation

12.12.2.1 void hcfft_GenerateLejaPoints (enum hcfft_TransformType *transformType*, hcfft_double * *points*, int *n*, double *alpha*, double *beta*)

Computes a Leja sequence for the given transform type.

Parameters

transform-Type	The type of transform.
points	The Leja points are stored here after the computation.
n	The requested number of points.
alpha	First basis function parameter.
beta	Second basis function parameter.

12.13 /home/hamaeker/codes/hcfft/src/priority_queue.h File Reference

Contains an implementation of a priority queue used to track the active set of the grid.

12.13.1 Detailed Description

Contains an implementation of a priority queue used to track the active set of the grid.

12.14 /home/hamaeker/codes/hcfft/src/transform_data.h File Reference

Contains the definition of hcfft_TransformData, as well as related functions.

```
#include "hcfft_internal.h"
```

Classes

- struct hcfft_TransformData

  Represents a 1-dimensional transform.

Functions

- void hcfft_InitTransformationStuff (struct hcfft_TransformData *TS, struct hcfft_TransformParams transformData, const int maxL, struct hcfft_TransformData *prevTS, int numPrevTS)

  Initialize the data for the given transformation.
- void hcfft_DestroyTransformationStuff (struct hcfft_TransformData *transformData)

  Destroy the given transformation data.
- struct hcfft_Interval hcfft_GetStdTSInterval (enum hcfft_TransformType type)

  Get the default interval for this transformation type.

12.14.1 Detailed Description

Contains the definition of hcfft_TransformData, as well as related functions.

12.14.2 Function Documentation

12.14.2.1 void hcfft_DestroyTransformationStuff (struct hcfft_TransformData * *transformData*)

Destroy the given transformation data.

Parameters

transformData	The transformation data that is being destroyed.

12.14.2.2 struct hcfft_Interval hcfft_GetStdTSInterval (enum hcfft_TransformType *type*) [read]

Get the default interval for this transformation type.

Parameters

type	The type of transform.

Returns

The default interval for this transform type.

12.14.2.3 void hcfft_InitTransformationStuff (struct hcfft_TransformData * *TS*, struct hcfft_TransformParams *transformData*, const int *maxL*, struct hcfft_TransformData * *prevTS*, int *numPrevTS*)

Initialize the data for the given transformation.

Parameters

TS	The transformation data will be initialized.
transform-Data	The transform parameters.
maxL	Maximum level.
prevTS	An array of transformations which have already been computed.
numPrevTS	The number of elements in the *prevTS* array.

12.15 /home/hamaeker/codes/hcfft/src/transforms/general_transform.h File Reference

Functions related to general transforms(non-dyadic ones).

```
#include <stdlib.h> #include "hcfft_internal.h" #include
<lapacke.h>
```

12.15.1 Detailed Description

Functions related to general transforms(non-dyadic ones).

12.16 /home/hamaeker/codes/hcfft/src/transforms/legendre.h File - Reference

Functions related to the Legendre transform.

```
#include "defines.h"
```

12.16.1 Detailed Description

Functions related to the Legendre transform.

12.17 /home/hamaeker/codes/hcfft/src/utils.h File Reference

Contains various utility functions.

Functions

- hcfft_double hcfft_EvalErrorAtGridPoints (struct hcfft_Grid *grid, struct hcfft_Vector *coeffs, hcfft_NodeFunction f, void *params)

  Evaluate the interpolation error at the grid points.

- hcfft_double hcfft_EvalErrorAtRandomPoints (struct hcfft_Grid *grid, struct hcfft_Vector *coeffs, hcfft_NodeFunction f, void *params, int numSamples, const struct hcfft_Interval *box)

  Evaluate the interpolation error at random points inside the interpolation domain.

- hcfft_double hcfft_ComputeL2Error (struct hcfft_Grid *grid, struct hcfft_Vector *coeffs, hcfft_NodeFunction f, void *params, int acc, const struct hcfft_Interval *box)

  Compute the L2 error of the approximation.

- hcfft_double hcfft_ComputeL2Norm (struct hcfft_Grid *grid, hcfft_NodeFunction f, void *params, int acc, const struct hcfft_Interval *box)

  Compute the L2 norm of the given function.

- void hcfft_PrintGridLevels (struct hcfft_Grid *grid, FILE *output, struct hcfft_Vector *v)

  For each sparse grid block print its index, the number of points in the block, the sum of the squared coefficients, and the average sum of the squared coefficients .

- void hcfft_PrintCoeffVector (struct hcfft_Grid *grid, FILE *output, struct hcfft_Vector *v)

  Print the values stored in the coefficients vector. For each grid point print its coordinates, the multiindex of the basis function and the corresponding coefficient.

- void hcfft_PrintGridPoints (struct hcfft_Grid *grid, FILE *output)

  Print the points of the sparse grid.

- void hcfft_PrintGridBasisIndices (struct hcfft_Grid *grid, FILE *output)

  Print the indices of all basis functions.

- void hcfft_GetGridPoints (struct hcfft_Grid *grid, hcfft_double *p)

  Get the points of the sparse grids.

- void hcfft_GetGridIndices (struct hcfft_Grid *grid, int *k)

  Get the multi-indices of the basis functions of the sparse grids.

12.17.1 Detailed Description

Contains various utility functions.

12.17.2 Function Documentation

12.17.2.1 hcfft_double hcfft_ComputeL2Error (struct hcfft_Grid * *grid*, struct hcfft_Vector * *coeffs*, hcfft_NodeFunction *f*, void * *params*, int *acc*, const struct hcfft_Interval * *box*)

Compute the L2 error of the approximation.

Parameters

grid	The sparse grid.
coeffs	The regular coefficients.
f	The target function.
params	Parameters of the target function.
acc	The desired accuracy. The accuracy error is roughly 1e-Acc.
box	If not NULL then the integration is performed in this domain.

Returns

The approximate L2 error.

12.17.2.2 hcfft_double hcfft_ComputeL2Norm (struct hcfft_Grid * *grid*, hcfft_NodeFunction *f*, void * *params*, int *acc*, const struct hcfft_Interval * *box*)

Compute the L2 norm of the given function.

Parameters

grid	The sparse grid.
f	The target function.
params	Parameters of the target function.
acc	The desired accuracy. The accuracy error is roughly 1e-Acc.
box	If not NULL then the integration is performed in this domain.

Returns

The approximate L2 norm.

12.17.2.3 hcfft_double hcfft_EvalErrorAtGridPoints (struct hcfft_Grid * *grid*, struct hcfft_Vector * *coeffs*, hcfft_NodeFunction *f*, void * *params*)

Evaluate the interpolation error at the grid points.

This error should be equal to zero if the interpolation was successful. Thus, this function is useful ofr testing purposes.

Parameters

grid	The sparse grid.
coeffs	The regular coefficients.
f	The target function.
params	Parameters of the target function.

Returns

The maximum difference between the function value and the interpolated value among all grid points.

12.17.2.4 hcfft_double hcfft_EvalErrorAtRandomPoints (struct hcfft_Grid * *grid,* struct hcfft_Vector * *coeffs,* hcfft_NodeFunction *f,* void * *params,* int *numSamples,* const struct hcfft_Interval * *box*)

Evaluate the interpolation error at random points inside the interpolation domain.

Parameters

grid	The sparse grid.
coeffs	The regular coefficients.
f	The target function.
params	Parameters of the target function.
num-Samples	Number of random points.
box	If not NULL then the samples are taken in this domain.

Returns

The maximum difference between the function value and the interpolated value among all random points.

12.17.2.5 void hcfft_GetGridIndices (struct hcfft_Grid * *grid,* int * *k*)

Get the multi-indices of the basis functions of the sparse grids.

The first D numbers contain the multi-index of the first basis function, the next D numbers are the multi-index of the second basis function and so on.

Parameters

grid	The sparse grid plan.
k	An array of DOF * D integers containing the multi-indices.

12.17.2.6 void hcfft_GetGridPoints (struct hcfft_Grid * *grid,* hcfft_double * *p*)

Get the points of the sparse grids.

The grid points are recorded as follows(in the case D = 3): x1, y1, z1, x2, y2, z2, x3, y3, z3, ...

Parameters

grid	The sparse grid.
p	An array of DOF * D numbers containing the point coordinates.

12.17.2.7 void hcfft_PrintCoeffVector (struct hcfft_Grid * *grid,* FILE * *output,* struct hcfft_Vector * *v*)

Print the values stored in the coefficients vector. For each grid point print its coordinates, the multiindex of the basis function and the corresponding coefficient.

Parameters

grid	The sparse grid.
output	The output will be stored in this file.
v	The coefficients vector.

12.17.2.8 void hcfft_PrintGridBasisIndices (struct hcfft_Grid * *grid,* FILE * *output*)

Print the indices of all basis functions.

Parameters

grid	The sparse grid.
output	The output will be stored in this file.

12.17.2.9 void hcfft_PrintGridLevels (struct hcfft_Grid * *grid,* FILE * *output,* struct hcfft_Vector * *v*)

For each sparse grid block print its index, the number of points in the block, the sum of the squared coefficients, and the average sum of the squared coefficients .

Parameters

grid	The sparse grid.
output	The output will be stored in this file.
v	The coefficients vector.

12.17.2.10 void hcfft_PrintGridPoints (struct hcfft_Grid * *grid*, FILE * *output*)

Print the points of the sparse grid.

Parameters

grid	The sparse grid.
output	The output will be stored in this file.

12.18 /home/hamaeker/codes/hcfft/src/vector.h File Reference

Contains various functions related to coefficient vectors.

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

Functions

- struct hcfft_Vector * hcfft_CreateVector (struct hcfft_Grid *grid)

  Create a dynamically allocated array of coefficients, one value for each node of the sparse grid.
- void hcfft_DestroyVector (struct hcfft_Vector *v)

  Free the resources allocated for the given vector.
- const hcfft_double * hcfft_GetRealData (const struct hcfft_Grid *grid, const struct hcfft_Vector *v)

  Get an hcfft_double pointer to the coefficient vector, if possible. If the grid works on complex values then the returned pointer is NULL.*
- const hcfft_complex * hcfft_GetComplexData (const struct hcfft_Grid *grid, const struct hcfft_Vector *v)

  Get an hcfft_complex pointer to the coefficient vector, if possible. If the grid works on real values then the returned pointer is NULL.*
- void hcfft_CopyVector (struct hcfft_Grid *grid, struct hcfft_Vector *v1, const struct hcfft_Vector *v2)

  Copy the contents of one coefficients vector to another one.
- void hcfft_AddVector (struct hcfft_Grid *grid, struct hcfft_Vector *dest, const struct hcfft_Vector *src)

  Add the values of one coefficients vector to the values of another one.
- void hcfft_SetVectorToZero (struct hcfft_Grid *grid, struct hcfft_Vector *v)

  Set all elements of the given vector to zero.
- hcfft_double hcfft_ComputeSquareL2Norm (struct hcfft_Grid *grid, const struct hcfft_Vector *v, int start, int len)

  Compute the squared L2 norm of a part of a coefficient vector.

12.18.1 Detailed Description

Contains various functions related to coefficient vectors.

12.18.2 Function Documentation

12.18.2.1 void hcfft_AddVector (struct hcfft_Grid * grid, struct hcfft_Vector * dest, const struct hcfft_Vector * src)

Add the values of one coefficients vector to the values of another one.

Parameters

grid	The sparse grid plan.
dest	The first vector. It will contain the final result.
src	The second vector.

12.18.2.2 hcfft_double hcfft_ComputeSquareL2Norm (struct hcfft_Grid * grid, const struct hcfft_Vector * v, int start, int len)

Compute the squared L2 norm of a part of a coefficient vector.

Parameters

grid	The grid.
v	The coefficient vector.
start	The beginning of the vector chunk that is being considered.
len	The length of the vector chunk that is being considered.

Returns

The squared L2 norm of a part of the given vector.

12.18.2.3 void hcfft_CopyVector (struct hcfft_Grid * grid, struct hcfft_Vector * v1, const struct hcfft_Vector * v2)

Copy the contents of one coefficients vector to another one.

Parameters

grid	The sparse grid plan.
v1	The vector to which the contents are being copied.
v2	The vector from which the contents are being taken.

12.18.2.4 struct hcfft_Vector* hcfft_CreateVector (struct hcfft_Grid * grid) [read]

Create a dynamically allocated array of coefficients, one value for each node of the sparse grid.

The type of the values depends on the type of the sparse grid i.e. on the transform that is used on the sparse grid. For example, if we are using Fourier transform then the values are of type hcfft_complex. If we are using Chebyshev transform then the values are of type hcfft_double.

Parameters

grid	The grid.

Returns

A pointer to the first element of the array.

12.18.2.5 void hcfft_DestroyVector (struct hcfft_Vector * *v*)

Free the resources allocated for the given vector.

Parameters

v	The vector which will be freed.

12.18.2.6 const hcfft_complex* hcfft_GetComplexData (const struct hcfft_Grid * *grid*, const struct hcfft_Vector * *v*)

Get an hcfft_complex* pointer to the coefficient vector, if possible. If the grid works on real values then the returned pointer is NULL.

Parameters

grid	The sparse grid.
v	The vector.

Returns

A pointer to the complex coefficients, or NULL.

12.18.2.7 const hcfft_double* hcfft_GetRealData (const struct hcfft_Grid * *grid*, const struct hcfft_Vector * *v*)

Get an hcfft_double* pointer to the coefficient vector, if possible. If the grid works on complex values then the returned pointer is NULL.

Parameters

grid	The sparse grid.
v	The vector.

Returns

A pointer to the real coefficients, or NULL.

12.18.2.8 void hcfft_SetVectorToZero (struct hcfft_Grid * *grid,* struct hcfft_Vector * *v*)

Set all elements of the given vector to zero.

Parameters

grid	The sparse grid plan.
v	The vector that is being set to zero.