CUNFFT 1.0 - Tutorial

Susanne Kunis[*] Stefan Kunis[†]

[*]sukunis@uos.de, University Osnabrück, Department of Mathematics, 49076 Osnabrück, Germany
[†]skunis@uos.de, University Osnabrück, Department of Mathematics, 49076 Osnabrück, Germany

Contents

1 Introduction

The CUNDFT and CUNFFT implements the NDFT and NFFT algorithm on a cuda-able graphic device. See the NFFT documentation for mathematical theory and ideas behind the NFFT.

2 Notation, the CUNDFT, and the CUNFFT

The algorithm

In summary, the following Algorithm 1 is a parallel implementation of the NFFT .

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

1: For $\boldsymbol{k} \in I_{\boldsymbol{N}}$ compute
       $$\hat{g}_{\boldsymbol{k}} := \frac{\hat{f}_{\boldsymbol{k}}}{|I_{\boldsymbol{n}}|\, c_{\boldsymbol{k}}(\tilde{\varphi})}.$$
2: For $\boldsymbol{l} \in I_{\boldsymbol{n}}$ compute by $d$-variate FFT
       $$g_{\boldsymbol{l}} := \sum_{\boldsymbol{k} \in I_{\boldsymbol{N}}} \hat{g}_{\boldsymbol{k}}\, \mathrm{e}^{-2\pi\mathrm{i}\boldsymbol{k}\left(\boldsymbol{n}^{-1} \odot \boldsymbol{l}\right)}.$$
3: For $j = 0, \dots, M-1$ compute
       $$f_j := \sum_{\boldsymbol{l} \in I_{\boldsymbol{n},m}(\boldsymbol{x}_j)} g_{\boldsymbol{l}}\, \tilde{\psi}\left(\boldsymbol{x}_j - \boldsymbol{n}^{-1} \odot \boldsymbol{l}\right).$$

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

Algorithm 1: `CUNFFT`

Algorithm 1 reads in matrix vector notation as

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

where $\boldsymbol{B}$ denotes the real $M \times |I_{\boldsymbol{n}}|$ sparse matrix

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

where $\boldsymbol{B}$ is the Fourier matrix of size $|I_{\boldsymbol{n}}| \times |I_{\boldsymbol{n}}|$, and where $\boldsymbol{B}$ is the real $|I_{\boldsymbol{n}}| \times |I_{\boldsymbol{N}}|$ 'diagonal' matrix

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

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

The corresponding computation of the adjoint matrix vector product reads as

$$\boldsymbol{A}^{\mathsf{H}}\hat{\boldsymbol{f}} \approx \boldsymbol{B}^{\top}\boldsymbol{F}^{\mathsf{H}}\boldsymbol{D}^{\top}\hat{\boldsymbol{f}}\,.$$

With the help of the transposed index set

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

one obtains Algorithm 2 for the adjoint CUNFFT. Due to the characterisation of the nonzero

```
Input: $d, M \in \mathbb{N}$, $\boldsymbol{N} \in 2\mathbb{N}^d$
        $\boldsymbol{x}_j \in [-\frac{1}{2}, \frac{1}{2}]^d$, $j = 0, \ldots, M-1$, and $f_j \in \mathbb{C}$, $j = 0, \ldots, M-1$,

1: For $\boldsymbol{l} \in I_{\boldsymbol{n}}$ compute
        $g_{\boldsymbol{l}} := \sum_{j \in I_{\boldsymbol{n},m}^{\top}(\boldsymbol{l})} f_j \, \tilde{\psi}\left(\boldsymbol{x}_j - \boldsymbol{n}^{-1} \odot \boldsymbol{l}\right)$ .
2: For $\boldsymbol{k} \in I_{\boldsymbol{N}}$ compute by $d$-variate (backward) FFT
        $\hat{g}_{\boldsymbol{k}} := \sum_{\boldsymbol{l} \in I_{\boldsymbol{n}}} g_{\boldsymbol{l}} \, \mathrm{e}^{+2\pi\mathrm{i}\boldsymbol{k}\left(\boldsymbol{n}^{-1} \odot \boldsymbol{l}\right)}$.
3: For $\boldsymbol{k} \in I_{\boldsymbol{N}}$ compute
        $\hat{h}_{\boldsymbol{k}} := \frac{\hat{g}_{\boldsymbol{k}}}{|I_{\boldsymbol{n}}| \, c_{\boldsymbol{k}}(\tilde{\varphi})}$.
Output: approximate values $\hat{h}_{\boldsymbol{k}}$, $\boldsymbol{k} \in I_{\boldsymbol{N}}$.
```

Algorithm 2: `CUNFFT`$^{\mathsf{H}}$

elements of the matrix $\boldsymbol{B}$, i.e.,

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

the multiplication with the sparse matrix $\boldsymbol{B}^{\top}$ is implemented in a 'transposed' way in the library, summation as outer loop and only using the multi-index sets $I_{\boldsymbol{n},m}(\boldsymbol{x}_j)$.

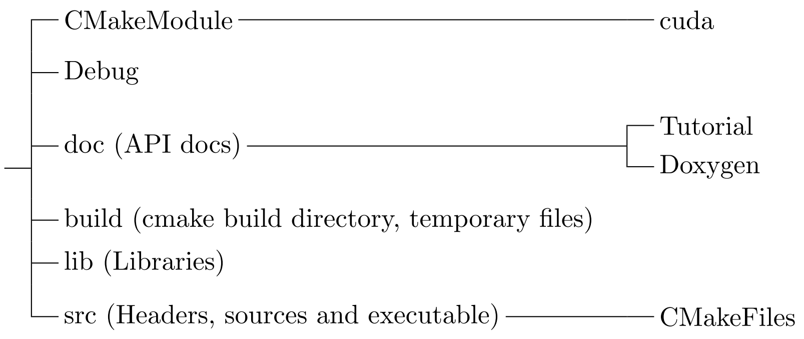
CMakeModule
cuda
Debug
doc (API docs)
Tutorial
Doxygen
build (cmake build directory, temporary files)
lib (Libraries)
src (Headers, sources and executable)
CMakeFiles

Figure 1.1: Directory structure of the CUNFFT package

3 Library

The library is completely written in C and CUDA 5.5 and uses the CUFFT library which has to be installed on your system. The library has several options (determined at compile time) and parameters (determined at run time).

3.1 Installation

Download from `http://www.analysis.uni-osnabrueck.de/index.php?n=Software.Software` the most recent version `cunfft.2.0.tar.gz`. In the following, we assume that you use a `bash` compatible shell. Uncompress the archive.

- `tar xfvz cunfft.2.0.tar.gz`

Change to the newly created directory.

- `cd cunfft.2.0/build/`

Run the cmake script.

- `cmake ..`

Compile the sources.

- `make`

Now you find the documentation in `cunfft.2.0/doc` and an examples executable in `cunfft.2.0/src`.
You can run `make install` to install the library on your system to `cunfft.2.0/lib`.

3.2 Options

The following options can also be used in conjunction with the cmake script: The CUNFFT routines use the Gaussian window function (see NFFT documentation). You can change cutoff parameter of the window function with the option `-DCUT_OFF=<val>`. For Fast Gaussian Gridding set `-DCOM_FG_PSI=ON`.

The CUNFFT transforms can be configured to measure elapsed time for each step of Algorithm 1 and 2. This should help customise the library to one's needs. You can enable this behaviour with the options `-DMEASURED_TIMES=ON`. The time is measured in seconds. If you wish an output in milliseconds set `-DMILLI_SEC=ON`.

Enable double precision computation with `-DCUNFFT_DOUBLE_PRECISION=ON`. For large data size use `-DLARGE_INPUT=ON` to use `ptrdiff_t` for int.

Specify your computation grid with `-DTHREAD_DIM_X=<val>` and `-DTHREAD_DIM_Y=<val>` for restriction for maximum numbers of threads per block in x and y direction. The restrictions for maximum number of threads inside a block is set by `-DMAX_BLOCK_DIM_X=<val>`, `-DMAX_BLOCK_DIM_Y=<val>` and `-DMAX_BLOCK_DIM_Z=<val>`. The restriction for maximum numbers of blocks per grid is set by `-DMAX_GRID_DIM_X=<val>`,`-DMAX_GRID_DIM_Y=<val>` and `-DMAX_GRID_DIM_Z=<val>`. For choose the best value check also the technical data of your grafic card. For the control of the kernel launch set `-DPRINT_CONFIG=ON` to see the launch configuration of the kernel. To verify CUDA function calls and get information about errors enable the debug output with `-DDEBUG=ON`.

3.3 Procedure for computing an CUNFFT

One has to follow certain steps to write a simple program using the CUNFFT library. A complete example is shown in Section 4.1. The first argument of each function is a pointer to a application-owned variable of type `cunfft_plan`. The aim of this structure is to keep interfaces small, it contains all parameters and data.

Initialisation

Initialisation of a plan is done by one of the `cunfft_init`-functions. The simplest version for the univariate case $d = 1$ just specifies the number of Fourier coefficients N_0 at the array N and the number of nonequispaced nodes M. For an application-owned variable `cunfft_plan my_plan` the function call is

```
cunfft_init(&my_plan,d,N,M);
```

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

Setting nodes

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

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

Doing the transform

Before the device can compute the transformation, you has to copy the data from host to the device with

```
copyDataToDevice(&my_plan);
```

Algorithm 1 is implemented as

```
cunfft_transform(&my_plan);
```

takes `my_plan.f_hat` as its input and overwrites `my_plan.f`. One only needs one plan for several transforms of the same kind, i.e. transforms with equal initialisation parameters. For comparison, the direct calculation of CUNDFT is done by `cundft_transform`. The adjoint transforms are given by `cunfft_adjoint` and `cundft_adjoint`, respectively. Use `copyDataToDeviceAd` for data transfer to device.

All data with multi-indices is stored plain, in particular the Fourier coefficient $\hat{f}_{\boldsymbol{k}}$ is stored in `my_plan.f_hat[k]` with $\mathtt{k} := \sum_{t=0}^{d-1}(k_t + \frac{N_t}{2}) \prod_{t'=t+1}^{d-1} N_{t'}$. To tranfer the data back to host use `copyDataToHost` and for adjoint transformation `copyDataToHostAd`.

Finalisation

All memory allocated by the init routine is deallocated by

```
cunfft_finalize(&my_plan);
```

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

Data structure and functions

The library defines the structure `cunfft_plan`, the most important members are listed in Table 3.1. Moreover, the user functions for the CUNFFT are collected in Table 3.2. They all have return type `void` and their first argument is of type `cunfft_plan*`. Interesting device functions are listed in Table 3.3.

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

Table 3.1: Interesting members of `cunfft_plan`.

Name	Additional arguments
`copyDataToDevice`	
`copyDataToHost`	
`copyDataToDeviceAd`	
`copyDataToHostAd`	
`cundft_transform`	
`cundft_adjoint`	
`cunfft_transform`	
`cunfft_adjoint`	
`cunfft_init`	`int d, int* N, int M`
`cunfft_init_guru`	`int d, int* N, int M, int* n, unsigned cunfft_flags`
`cunfft_finalize`	

Table 3.2: User functions of the CUNFFT.

Name	description
`resetDevice`	reset device
`findDevice`	find cuda able devices
`getGPUMemProps_ToStdout`	show available memory on device and input restrictions for CUNFFT

Table 3.3: Device functions of the CUNFFT.

4 Examples

The library was tested on a Intel®Core$^{\text{TM}}$ i5 CPU 760 @ 2.80GHz x 4 , 3,9GB memory, Ubuntu 14.04 LTS, kernel 3.13.0-62-generic, gcc version 4.8.2(Ubuntu 4.8.2-19ubuntu1) and the graphic device GeForce GTX 460 with compute capability 2.1 with the NVIDIA driver NVIDIA UNIX x86_64 Kernel Module 331.113. In all tests with random input the nodes $\boldsymbol{x}_j$ and the Fourier coefficients $\hat{f}_{\boldsymbol{k}}$ are chosen pseudo randomly with $\boldsymbol{x}_j \in [-0.5, 0.5]^d$ and $\hat{f}_{\boldsymbol{k}} \in [0,1] \times [0,1]\mathrm{i}$.

4.1 Computing your first transform

The following code summarises the steps of Section 3.3 and computes a univariate CUNFFT from 32 Fourier coefficients and 64 nodes. Note that this routine is part of `simpleTest.cpp` in `src/` and uses additional routines as defined `cunfft_util.h` to set up and show vectors.

```
void simple_test_cunfft_1d()
{
  resetDevice();
  uint_t N[3],M;
  N[0]=32; M=64;
  cunfft_plan p;
  cunfft_init(&p,1,N,M);
  getExampleDate_uniDistr(&p);

  copyDataToDevice(&p);
  cundft_transform(&p);
  copyDataToHost(&p);
  showCoeff_cuComplex(p.f,32,"cundft , vector f (first few entries)");

  cunfft_reinit(&p);
  copyDataToDevice(&p);
  cunfft_transform(&p);
  copyDataToHost(&p);
  showCoeff_cuComplex(p.f,32,"cunfft , vector f (first few entries)");

  cunfft_finalize(&p);
}
```

4.2 Computation time vs. problem size

Tested the straightforward evaluation of CUNFFT, denoted by CUNDFT, and the NFFT for increasing total problem sizes $|I_{\boldsymbol{N}}|$ and space dimensions $d = 1, 2, 3$, where $\boldsymbol{N} = (N, \ldots, N)^\top$, $N \in \mathbb{N}$. We choose $M = N^d$ random nodes for the CUNDFT, the NFFT and CUNFFT. Within the latter, we use the oversampling factor $\sigma = 2$, the cut-off $m = 6$ ($m = 12$) and the Fast Gaussian window function (`COM_FG_PSI`). This results in a fixed accuracy of $E_\infty := \|\boldsymbol{f} - \boldsymbol{s}\|_\infty / \|\hat{\boldsymbol{f}}\|_1 \approx 10^{-7} (\approx 10^{-13})$ for $d = 1, 2, 3$.

		$m = 4, E_\infty \approx 10^{-6}$		$m = 2, E_\infty \approx 10^{-4}$	
l_N	CUNDFT	NFFT	CUNFFT (pFac)	NFFT	CUNFFT (pFac)
$d = 1$					
4	4.37E-05	8.92E-06	1.22E-04 (0,07)	8.22E-06	1.21E-04 (0,07)
5	6.55E-05	2.53E-05	1.90E-04 (0,13)	1.26E-05	1.24E-04 (0,10)
6	1.10E-04	2.02E-05	1.29E-04 (0,16)	2.01E-05	1.28E-04 (0,16)
7	2.13E-04	3.30E-05	1.32E-04 (0,25)	7.27E-05	2.27E-04 (0,32)
8	5.66E-04	6.24E-05	1.40E-04 (0,45)	5.49E-05	1.32E-04 (0,41)
9	1.11E-03	1.16E-04	1.46E-04 (0,79)	1.02E-04	1.39E-04 (0,74)
10	2.20E-03	5.53E-04	2.45E-04 (2,26)	1.96E-04	1.47E-04 (1,33)
11	8.70E-03	4.22E-04	1.47E-04 (2,87)	3.98E-04	1.44E-04 (2,75)
12	2.60E-02	8.53E-04	1.61E-04 (5,30)	8.08E-04	1.57E-04 (5,16)
13	8.66E-02	1.87E-03	4.49E-04 (4,16)	1.76E-03	4.30E-04 (4,10)
14	3.46E-01	3.81E-03	2.78E-04 (13,70)	3.61E-03	2.66E-04 (13,56)
15	1.32E+00	7.84E-03	7.70E-04 (10,19)	7.38E-03	7.21E-04 (10,23)
16	5.12E+00	1.63E-02	1.19E-03 (13,64)	1.50E-02	1.03E-03 (14,53)
17	2.05E+01	3.39E-02	2.09E-03 (16,21)	3.29E-02	1.78E-03 (18,53)
18	*	7.54E-02	3.96E-03 (19,07)	7.14E-02	3.28E-03 (21,76)
19	*	1.87E-01	7.73E-03 (24,14)	1.82E-01	6.33E-03 (28,77)
20	*	4.01E-01	1.56E-02 (25,75)	3.84E-01	1.28E-02 (30,13)
21	*	8.63E-01	3.20E-02 (26,99)	8.00E-01	2.61E-02 (30,61)
22	*	1.90E+00	6.42E-02 (29,54)	1.78E+00	5.25E-02 (33,96)
$d = 2$					
6	1.29E-04	4.08E-05	1.56E-04 (0,26)	3.26E-05	1.50E-04 (0,22)
8	6.09E-04	2.52E-04	2.44E-04 (1,03)	1.04E-04	1.60E-04 (0,65)
10	2.36E-03	5.38E-04	1.90E-04 (2,83)	3.98E-04	1.70E-04 (2,34)
12	2.74E-02	2.36E-03	2.80E-04 (8,42)	3.20E-03	2.16E-04 (14,80)
14	3.63E-01	1.12E-02	1.28E-03 (8,73)	7.92E-03	8.16E-04 (9,71)
16	5.37E+00	4.81E-02	4.90E-03 (9,81)	3.57E-02	2.52E-03 (14,16)
18	8.54E+01	3.40E-01	2.02E-02 (16,84)	2.27E-01	9.39E-03 (24,20)
20	*	1.62E+00	8.34E-02 (19,37)	1.18E+00	3.88E-02 (30,36)
$d = 3$					
9	1.26E-03	1.43E-03	5.05E-04 (2,83)	5.13E-04	2.61E-04 (1,96)
12	2.79E-02	1.31E-02	2.67E-03 (4,91)	4.73E-03	6.18E-04 (7,66)
15	1.39E+00	1.19E-01	2.34E-02 (5,09)	4.80E-02	5.75E-03 (8,35)
18	*	2.03E+00	2.06E-01 (9,87)	7.60E-01	4.72E-02 (16,11)

Table 4.1: Transformation:Computation time in seconds with respect to $l_N = \log_2 |I_{\boldsymbol{N}}|$. Configuration: Blocksize =128x128. Numbers in parentheses give the penalty factor, i.e., the qoutient of computation times for NFFT and CUNFFT. Note that we used accumulated measurements in case of small times and the times (*) are not displayed due to the large response time in comparison to the CUNDFT time.

		$m=4, E_\infty \approx 10^{-7}$		$m=2, E_\infty \approx 10^{-5}$	
l_N	CUNDFT	NFFT	CUNFFT (pFac)	NFFT	CUNFFT (pFac)
$d=1$					
4	4.35E-05	8.88E-06	1.30E-04 (0,07)	8.68E-06	1.15E-04 (0,08)
5	6.60E-05	1.23E-05	1.47E-04 (0,08)	1.19E-05	1.29E-04 (0,09)
6	1.09E-04	1.90E-05	1.56E-04 (0,12)	2.15E-05	1.38E-04 (0,16)
7	2.09E-04	3.24E-05	1.75E-04 (0,19)	2.97E-05	1.44E-04 (0,21)
8	5.51E-04	5.76E-05	1.73E-04 (0,33)	5.56E-05	1.49E-04 (0,37)
9	1.09E-03	1.06E-04	2.01E-04 (0,53)	9.87E-05	1.64E-04 (0,60)
10	2.14E-03	2.07E-04	2.57E-04 (0,81)	1.94E-04	2.03E-04 (0,95)
11	8.43E-03	4.35E-04	3.22E-04 (1,35)	4.12E-04	2.42E-04 (1,70)
12	2.52E-02	2.33E-03	5.16E-04 (4,51)	8.21E-04	3.32E-04 (2,47)
13	8.39E-02	2.23E-03	6.41E-04 (3,47)	1.79E-03	4.36E-04 (4,11)
14	3.35E-01	3.92E-03	9.76E-04 (4,01)	3.65E-03	6.69E-04 (5,45)
15	1.27E+00	8.07E-03	2.24E-03 (3,61)	7.46E-03	1.47E-03 (5,08)
16	4.96E+00	1.61E-02	4.80E-03 (3,36)	1.51E-02	3.09E-03 (4,88)
17	1.99E+01	3.44E-02	9.98E-03 (3,44)	3.22E-02	6.36E-03 (5,06)
18	7.89E+01	7.59E-02	2.05E-02 (3,71)	7.20E-02	1.30E-02 (5,53)
19	*	1.87E-01	4.14E-02 (4,52)	1.83E-01	2.63E-02 (6,94)
20	*	4.05E-01	8.37E-02 (4,84)	3.81E-01	5.33E-02 (7,15)
21	*	8.51E-01	1.69E-01 (5,04)	8.21E-01	1.08E-01 (7,61)
22	*	1.74E+00	3.39E-01 (5,13)	1.73E+00	2.16E-01 (7,98)
$d=2$					
6	1.21E-04	5.18E-05	6.74E-04 (0,08)	3.64E-05	3.37E-04 (0,11)
8	5.72E-04	1.65E-04	7.68E-04 (0,21)	1.14E-04	3.80E-04 (0,30)
10	2.21E-03	6.35E-04	1.26E-03 (0,50)	4.53E-04	5.74E-04 (0,79)
12	2.57E-02	2.75E-03	2.77E-03 (0,99)	1.97E-03	1.18E-03 (1,66)
14	3.41E-01	1.21E-02	7.82E-03 (1,55)	8.18E-03	3.29E-03 (2,49)
16	5.05E+00	5.01E-02	3.65E-02 (1,37)	3.56E-02	1.49E-02 (2,39)
18	8.03E+01	3.27E-01	1.52E-01 (2,16)	1.99E-01	6.07E-02 (3,28)
20	*	1.61E+00	6.16E-01 (2,62)	9.34E-01	2.46E-01 (3,79)
$d=3$					
9	1.23E-03	2.29E-03	6.25E-03 (0,37)	7.67E-04	1.53E-03 (0,50)
12	2.71E-02	2.09E-02	1.96E-02 (1,07)	6.15E-03	4.57E-03 (1,35)
15	1.36E+00	1.55E-01	1.72E-01 (0,90)	5.44E-02	3.92E-02 (1,39)
18	8.40E+01	2.63E+00	1.46E+00 (1,80)	8.87E-01	3.28E-01 (2,70)

Table 4.2: Adjoint Transformation: Computation time in seconds with respect to $l_N = \log_2 |I_{\boldsymbol{N}}|$. Configuration: Blocksize =128x128. Numbers in parentheses give the penalty factor, i.e., the qoutient of computation times for NFFT and CUNFFT. Note that we used accumulated measurements in case of small times and the times (*) are not displayed due to the large response time in comparison to the CUNDFT time.

4.3 Accuracy vs. window function and cut-off parameter m

The accuracy of the Algorithm 1, measured by

$$E_\infty = \frac{\|\boldsymbol{f} - \boldsymbol{s}\|_\infty}{\|\hat{\boldsymbol{f}}\|_1} = \max_{0 \le j < M} |f_j - s(\boldsymbol{x}_j)| / \sum_{\boldsymbol{k} \in I_{\boldsymbol{N}}} |\hat{f}_{\boldsymbol{k}}|$$

is shown in Figure 4.1.

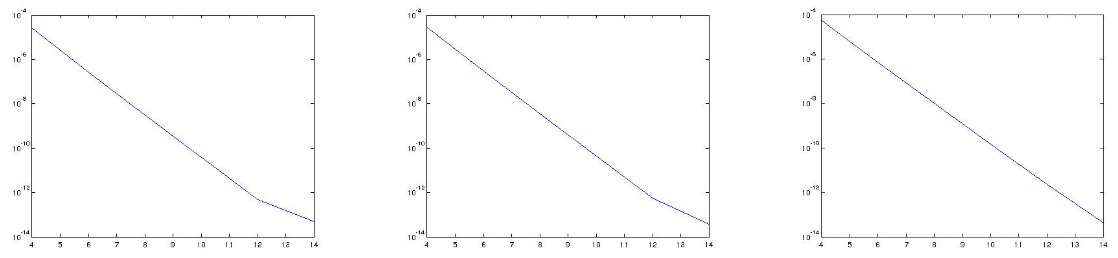

Figure 4.1: The error E_∞ with respect to m, from left to right $d = 1, 2, 3$ ($N = 2^{12}, 2^6, 2^4$, $\sigma = 2$, $M = 10000$) for the Gaussian window.