Instruction to Fourier Algorithm for Structured Sparsity

Mark Iwen (markiwen@math.msu.edu)

Sina Bittens (sina.bittens@mathematik.uni-goettingen.de)

Ruochuan Zhang (zhangr12@math.msu.edu)

How to compile and run the code:

(*>> … : command line)

1. Put ‘FAST_block_sparse_Cpp.zip’ in your home directory.

2. Unzip ‘FAST_block_sparse_Cpp.zip’.

3. Open ‘Makefile’ and see ‘Instructions’ in it.

```
#############################################################################################
#Instructions
#############################################################################################
#1. You have to edit line 23 (the line starting with LIBS) to indicate where your 'fftw3 library' is located.
#     For example, the LIBS line can be:
#     LIBS = -L/Users/username/usr/lib -lfftw3 -lm
#     You may need to edit the directory appearing after -L to give the correct path of LIBS.
#2. If you don't have it installed yet, then see the following:
#     1) visit : http://micro.stanford.edu/wiki/Install_FFTW3    to see how to install the library.
#     2) In the command line:
#          >>mkdir $HOME/usr
#          >>mkdir $HOME/soft
#          >>cd ~/soft
#          >>wget http://www.fftw.org/fftw-3.3.4.tar.gz (this is where the 'latest' fftw3 library is located)
#          >>tar -zxvf fftw-3.3.4.tar.gz
#          >>cd fftw-3.3.4
#          >>./configure --prefix=$HOME/usr --enable-shared=yes
#          >>make --jobs=8
#          >>make install
#############################################################################################
```

4. Edit ‘Makefile’ according to the instructions. Mainly, the edition is changing the directory of LIBS.

5. Open ‘StructuredSFT.cpp’ and read the instructions.

```
//*************************************************************************************************************//
//Instructions
//This code runs numerical experiments for the Fourier Algorithm for Structured SparsiTy (FAST).

//Parameter setting starts at line 32 (after the include and using namespace commands).

//Set up the N, num_block and B correctly.
//Set Randomized to be true if you want to run the randomized version, i.e., FASTR.
//When running the randomized version, only increase the value of randomize_int if you want to increase the probability of correct recovery.
//In the ReadMe file we provide exemplary test runs that demonstrate how we set the parameters.
//Set fftwMeasure to be true if you want FFTW to run all the DFTs using FFTW_MEASURE plan.
//This can make the algorithm have a better running time performance, but it can take relatively long to generate fftw_plan.
//One can also include the directory location of fftw3.h in the include command.
//For example, one can change #include <fftw3.h> to #include </User/username/usr/include/fftw3.h> to let the compiler know the correct
path of fftw3.h.

//We have to compile this file (StructuredSFT.cpp) with primes.h using Makefile.
//*************************************************************************************************************
```

6. Edit the parameters in ‘StructuredSFT.cpp’ based on the description of each parameter.

```
//*************************************************************************************************************//
//Below is the part of the code for setting the parameters.
//*************************************************************************************************************//
//Number of repeated numerical tests.
int num_test = 1;
//Number of different numerical tests.
int rounds = 100;
//N is the bandwidth; later we will take the value of N from outside (N is the smallest power of two that is greater than the sample size),
//e.g., if the sample size is 1000, then N is 2^10 = 1024.
const int N = pow(2, 22);
//Number of significant frequencies in each block.
const int B = pow(2,4);
//num_block is n in the pseudo-code of the (n,B)-block Fourier Algorithm for Structured SparsiTy.
const int num_block = 3;
//Set Randomized to true if you want to use the randomized version, i.e. FASTR.
//The deterministic version is FAST.
bool Randomized = true;
```

```
//Set randomize_int to be a larger odd number if you want a higher probability of correct recovery.
int randomize_int = 3;
//Set fftwMeasure to true if you want to use FFTW_MEASURE plan (better performance, but it takes a really long time to generate the
//fftw_plans).
//Set it to false if you want to use FFTW_ESTIMATE plan (sub-optimal performance, but it takes much less time to generate the
//fftw_plans).
bool fftwMeasure = false;
//Set noiseless to true if you want to run the algorithm in the noiseless case, i.e. FAST.
//Otherwise set it to true.
bool noiseless = false;
//Signal to Noise Ratio (SNR).
double SNR=30;
//*****************************************************************************************************************//
```

7. Generate ‘StructuredSFT.exe’.

   >>make

   To execute ‘StructuredSFT.exe’.

   >> ./StructuredSFT.exe

8. Output example:

```
[zhangr12@dev-intel16 StructuredSFT]$ ./StructuredSFT.exe
Using FFTW_ESTIMATE, not FFTW_MEASURE!
The bandwidth, number of blocks and frequencies in each block are: 4194304, 3, 16
u: 16
The number of blocks is: 3
t_l
1 9 5 7 11 13 17
s_k
19 23 29 31 37
The ratio between the total number of samples and the bandwidth: 0.0334053
Initialize samples...
Create fftw plans for forward and backward FFT
Create fftw_plans uses: (in second) 0.04
Start numerical experiment...
Total number of different tests is: 10.
Each test is repeated 10 times.
The noise level (SNR) is: 30
Among 10 numerical tests the number of correct recovery is 10
The maximum error among the numerical tests is : 0.00381386
The error in average L1 norm is : 0.00121466
The running time is: 0.017
[zhangr12@dev-intel16 StructuredSFT]$
```

9. Guidance of parameter settings

In this part, we will provide guidance to the parameter settings for the randomized version of the algorithm. In the randomized version of the algorithm, FASTR, the only parameter that needs to be tuned is ‘*randomize_int*’. This parameter determines how many prime numbers will be used in ‘*s_k*’. Generally speaking, a small ‘*randomize_int*’ implies faster runtime and lower accuracy. In our numerical experiment we test FASTR for signals with different numbers of blocks and different block lengths. Let *n* denote the number of blocks and *B* the length of each block. Then we obtain the table below:

n	*B*	*randomize_int*
2	2^2 ~ 2^10	1
2	2^11	3

3	2^2 ~ 2^4	1
3	2^5 ~ 2^11	3

By using the above setting, the probability of correct recovery is at least 0.9.