MULTI-DIMENSIONAL SUBLINEAR SPARSE FOURIER ALGORITHM

Bosu Choi[*]

Andrew Christlieb[†]

Yang Wang[‡]

Abstract

In this paper, we discuss the development of a sublinear sparse Fourier algorithm for high-dimensional data. In "Adaptive Sublinear Time Fourier Algorithm" by D. Lawlor, Y. Wang and A. Christlieb (2013) [9], an efficient algorithm with empirically $O(k \log k)$ runtime and $O(k)$ sampling complexity for the one-dimensional sparse FFT was developed for signals of bandwidth N, where k is the number of significant modes such that $k \ll N$.

In this work we develop an efficient algorithm for sparse FFT for higher dimensional signals, extending some of the ideas in [9]. Note a higher dimensional signal can always be unwrapped into a one dimensional signal, but when the dimension gets large, unwrapping a higher dimensional signal into a one dimensional array is far too expensive to be realistic. Our approach here introduces two new concepts: "partial unwrapping" and "tilting". These two ideas allow us to efficiently compute the sparse FFT of higher dimensional signals.

Keywords Higher dimensional sparse FFT · Partial unwrapping · Fast Fourier algorithms · Fourier analysis

Mathematics subject classification 65T50 · 68W25

[*]Department of Mathematics, Michigan State University, 619 Red Cedar Road, East Lansing, MI 48824, USA choibosu@math.msu.edu

[†]Department of CMSE and Department of Mathematics, Michigan State University, East Lansing, MI 48824, USA andrewch@math.msu.edu

[‡]Department of Mathematics, The Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong yangwang@ust.hk

1 INTRODUCTION

As the size and dimensionality of data sets in science and engineering grow larger and larger, it is necessary to develop efficient tools to analyze them [5, 10]. One of the best known and most frequently-used tools is the Fast Fourier Transform (FFT). However, in the case that the bandwidth N of frequencies is large, the sampling size becomes large, as dictated by the Shannon-Nyquist sampling theorem. Specifically, the runtime complexity is $O(N\log N)$ and the number of samples is $O(N)$. This issue is only exacerbated in the d-dimensional setting, where the runtime complexity is $O(N^d\log N^d)$ and the number of samples is $O(N^d)$ if we assume the dimension is d and the bandwidth in each dimension is N. Due to this "curse of dimensionality", many higher dimensional problems of interest are beyond current computational capabilities of the traditional FFT. Moreover, in the sparse setting where the number of significant frequencies k is small, it is computationally wasteful to compute all N^d coefficients. In such a setting we refer to the problem as being "sparse". For sparse problems, the idea of sublinear sparse Fourier transforms was introduced [3, 4, 6, 7, 8, 9, 1]. These methods greatly reduce the runtime and sampling complexity of the FFT in the sparse setting. The methods were primarily designed for the one dimensional setting.

The first sparse Fourier algorithm was proposed in [3]. It introduced a randomized algorithm with $O(k^2\log^c N)$ runtime and $O(k^2\log^c N)$ samples where c is a positive number that varies depending on the trade-off between efficiency and accuracy. An algorithm with improved runtime $O(k\log^c N)$ and samples $O(k\log^c N)$ was given in [4]. The algorithms given in [6] and [7] achieved $O(k\log N\log N/k)$ runtime and gave empirical results. The algorithms in [3, 4, 6, 7] are all randomized. The first deterministic algorithm using a combinatorial approach was introduced in [8]. In [9], another deterministic algorithm was given whose procedure recognizes frequencies in a similar manner to [6]. The two methods in [9, 6] were published at the same time and both use the idea of working with two sets of samples, one at $O(k)$ points and the second at the same $O(k)$ points plus a small shift. The ratio of the FFT of the two sets of points, plus extra machinery, lead to fast deterministic algorithms. The first deterministic algorithm [8] has $O(k^2\log^4 N)$ runtime and sampling complexity, and the second one [9] has $O(k\log k)$ runtime and $O(k)$ sampling complexity. Later, [1] introduced modified methods for noisy data with $O(k\log k\log N/k)$ runtime. Our method, discussed throughout this paper builds on the method presented in [9].

The methods introduced in the previous paragraph are for one-dimensional data. In [2], practical algorithms for data in two dimensions were given for the first time. In this paper, we develop algorithms designed for higher dimensional data, which is effective even for dimensions in the hundreds and thousands. To achieve our goal, our approach must address the worst case scenario presented in [2]. We can find a variety of data sets in multiple dimensions that we want to analyze. A relatively low-dimensional example is MRI data, which is three dimensional. However, when we designed the method in this paper, we had much higher dimensional problems in mind, such as some astrophysical data, e.g., the Sloan Digital Sky Survey and Large Synoptic Survey Telescope [11, 10]. They produce tera- or peta-bytes of imaging and spectroscopic data in very high dimensions. Due to the computational effort of a multi-dimensional FFT, spectral analysis of this high dimensional data necessitates a multidimensional sparse fast Fourier transform. Further, given the massive size of data sets in some current and future problems in science and engineering, it is anticipated that the development of such an efficient algorithm will play an important role in the analysis of these types of data.

It is not straightforward to extend one dimensional sparse Fourier transform algorithms to multiple dimensions. We face several obstacles. First, we do not have an efficient FFT for multidimensional problems much higher than three. Using projections onto lower-dimensional spaces solves this problem. However, like all projection methods for sparse FFT, one needs to match frequencies from one projection with those from another projection. This *registration* problem is one of the big challenges in the one dimensional sparse FFT. An equally difficult challenge is that different frequencies may be projected into the same frequency (*the collision problem*). All projection methods for sparse FFT primarily aim to overcome these two challenges. In higher dimensional sparse FFT, these problems become even more challenging as now we are dealing with frequency vectors, not just scalar frequencies.

As a first step to our goal of a high dimensional sparse FFT, this paper addresses the case for continuous data without noise in a high dimensional setting. In a later paper we shall present an adaptation of the algorithm for noisy data. We introduce effective methods to address the registration and the collision problems. In particular, we introduce a novel *partial unwrapping* technique that is shown to be highly effective in reducing the registration and collision complexity while maintains the sublinear runtime efficiency. We shall show that empirically we can achieve $O(dk\log k)$ computational complexity and $O(dk)$ sampling size for randomly generated test data. In Section 5, we present as examples computational results for sparse FFT where the dimensions are 100 and 1000 respectively. For comparison, the traditional d-dimensional FFT requires $O(N^d\log N^d)$ time complexity and $O(N^d)$ sampling complexity, which is impossible to implement on any computers today.

2 PRELIMINARIES

2.1 Review of the One-Dimensional Sublinear Sparse Fourier Algorithms

The one-dimensional sublinear sparse Fourier algorithm inspiring our method was developed in [9]. We briefly introduce the idea and notation of the algorithm before developing the multidimensional ones throughout this paper. We assume a function $f : [0, 1) \to \mathbb{C}$ with sparsity k as the following,

$$f(t) \;=\; \sum_{j=1}^{k} a_j e^{2\pi i w_j t} \tag{2.1}$$

with bandwidth N, i.e., frequency w_j belongs to $[-N/2, N/2) \cap \mathbb{Z}$ and corresponding nonzero coefficient a_j is in $\mathbb{C}$ for all j. We can consider it as a periodic function over $\mathbb{R}$ instead of $[0, 1)$. The goal of the algorithm is to recover all coefficients a_j and frequencies w_j so that we can reconstruct the function f. This algorithm is called the "phase-shift" method since it uses equi-spaced samples from the function and those at positions shifted by a small positive number ϵ. To verify that the algorithm correctly finds the frequencies in the bandwidth N, ϵ should be strictly no bigger than $1/N$. We denote a sequence of samples shifted by ϵ with sampling rate $1/p$, where p is a prime number, as

$$\mathbf{f}_{p,\epsilon} \;=\; \Big(f(0+\epsilon), f(\frac{1}{p}+\epsilon), f(\frac{2}{p}+\epsilon), f(\frac{3}{p}+\epsilon), \cdots, f(\frac{p-1}{p}+\epsilon)\Big). \tag{2.2}$$

We skip much of the details here. In a nutshell, by choosing p slightly larger than k is enough to make the algorithm work. In [9] p is set to be roughly $5k$, which is much smaller than the Nyquist rate N. Discrete Fourier transform (DFT) is then applied to the sample sequence $\mathbf{f}_{p,\epsilon}$, and the h-th element of its result is the following

$$\mathcal{F}(\mathbf{f}_{p,\epsilon})[h] \;=\; p \sum_{w_j = h(\bmod p)} a_j e^{2\pi i \epsilon w_j} \tag{2.3}$$

where $h = 0, 1, 2, \ldots, p-1$. If there is only one frequency w_j congruent to h modulo p,

$$\mathcal{F}(\mathbf{f}_{p,\epsilon})[h] \;=\; p a_j e^{2\pi i \epsilon w_j}. \tag{2.4}$$

By putting 0 instead of ϵ, we can get unshifted samples $\mathbf{f}_{p,0}$ and applying the DFT gives

$$\mathcal{F}(\mathbf{f}_{p,0})[h] \;=\; p a_j. \tag{2.5}$$

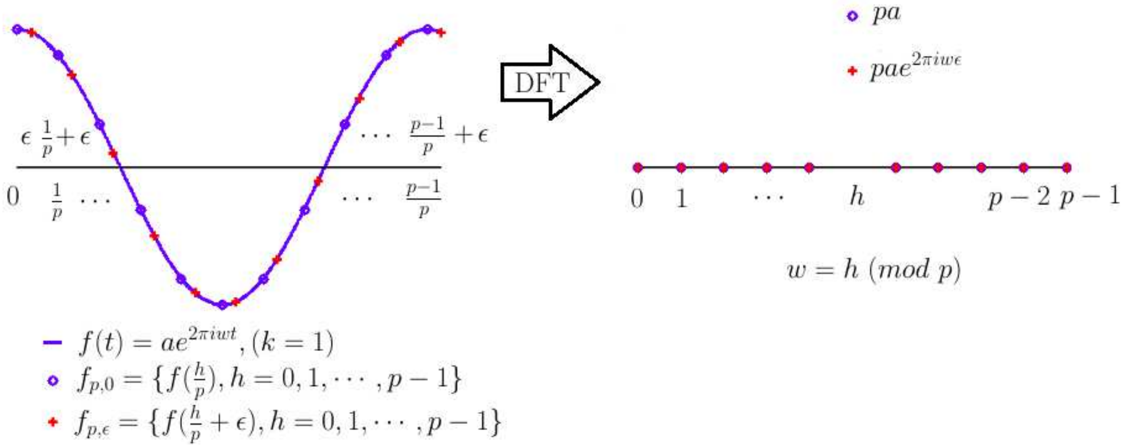
This process so far is visualized in the Figure 1. As long as there is no collision of frequencies with modulo p, we can find frequencies and their corresponding coefficients by the following computation

$$
\begin{aligned}
w_j &= \frac{1}{2\pi\epsilon}\mathrm{Arg}\Big(\frac{\mathcal{F}(\mathbf{f}_{p,\epsilon})[h]}{\mathcal{F}(\mathbf{f}_{p,0})[h]}\Big), \\
a_j &= \frac{1}{p}\mathcal{F}(\mathbf{f}_{p,0})[h], \qquad (2.6)
\end{aligned}
$$

where the function "Arg" gives us the argument falling into $[-\pi, \pi)$. Note that w_j should be the only frequency congruent to h modulo p, i.e., w_j has no collision with other frequencies modulo p. The test to determine whether collision occurs or not is

$$
\frac{|\mathcal{F}(\mathbf{f}_{p,\epsilon})[h]|}{|\mathcal{F}(\mathbf{f}_{p,0})[h]|} = 1. \qquad (2.7)
$$

The equality above holds when there is no collision. If there is a collision, the equality does not hold for almost all ϵ, i.e., the test fails to predict a collision for finite number of ϵ [9]. Further, it is also shown in the same paper that for any $\epsilon = \frac{a}{b}$ with a, b coprime and $b \geq N$, equality (2.7) does not hold unless there is no collision. In practical implementations, we choose ϵ to be $1/cN$ for some positive integer $c \geq 1$ and allow some small difference τ between the left and right sides of (2.7) where τ is very small positive number.



ϵ $\frac{1}{p}+\epsilon$ $\cdots$ $\frac{p-1}{p}+\epsilon$
0 $\frac{1}{p}$ $\cdots$ $\cdots$ $\frac{p-1}{p}$
DFT
pa
$pae^{2\pi i w\epsilon}$
0 1 $\cdots$ h $p-2$ $p-1$
$w = h\ (mod\ p)$
$f(t) = ae^{2\pi iwt}, (k=1)$
$f_{p,0} = \{f(\frac{h}{p}), h = 0, 1, \cdots, p-1\}$
$f_{p,\epsilon} = \{f(\frac{h}{p}+\epsilon), h = 0, 1, \cdots, p-1\}$

Figure 1: Process of 1D sublinear sparse Fourier algorithm

The above process is one loop of the algorithm with a prime number p. To explain it from a different view, we can imagine that there are p bins. Then we sort all frequencies into these bins according to their remainder modulo p. If there are more than one frequencies in one bin, then a collision happened. If there is only one frequency, then there is no collision. To determine whether a collision occurs, we use the above test. In the case where the test fails, i.e., the ratio is not 1, we need to use another prime number p'.

Thus we re-sort the frequencies into p' bins by their remainder modulo p'. Even if two frequencies collide modulo p, it is likely that they do not collide modulo p'. Particularly, the Chinese Remainder Theorem guarantees that with a finite set of prime numbers, $\{p_\ell\}$, any frequency within the bandwidth N can be uniquely identified, given $\prod_\ell p_\ell \geq N$. Algorithmically, for each loop, we choose a different prime number p' and repeat equations (2.2)-(2.7) with p replaced by p'. In this way we can recover all a_j and w_j in sublinear time $O(k\text{log}k)$ using $O(k)$ samples. The overall code is shown in Algorithm 1 referred from [9].

Algorithm 1 Phaseshift

```
1: procedure Phaseshift
2: Input: $f, c, k, N, \epsilon$
3: Output: $R$
4:   $R \leftarrow \emptyset$
5:   $i \leftarrow 1$
6:   while $|R| < k$ do
7:     $k^* \leftarrow k - |R|$
8:     $p \leftarrow ith$ prime number $\geq ck^*$
9:     $g(t) = \sum_{(w,a_w)\in R} a_w e^{2\pi iwt}$
10:    for $h = 1 \rightarrow p$ do
11:      $f_{p,\epsilon}[h] = f(\frac{h}{p}) - g(\frac{h}{p})$
12:      $f_{p,0}[h] = f(\frac{h}{p} + \epsilon) - g(\frac{h}{p} + \epsilon)$
13:    end for
14:    $\mathcal{F}(f_{p,\epsilon}) = FFT(f_{p,\epsilon})$
15:    $\mathcal{F}(f_{p,0}) = FFT(f_{p,0})$
16:    $\mathcal{F}^{sort}(f_{p,0}) = SORT(\mathcal{F}(f_{p,0}))$
17:    for $h = 1 \rightarrow k^*$ do
18:      if $\left|\frac{|\mathcal{F}^{sort}(f_{p,0})[h]|}{|\mathcal{F}^{sort}(f_{p,\epsilon})[h]|} - 1\right| < \epsilon$ then
19:        $\tilde{w} = \frac{1}{2\pi\epsilon}\text{Arg}\left(\frac{\mathcal{F}^{\text{sort}}(\text{f}_{\text{p},\epsilon})[\text{h}]}{\mathcal{F}^{\text{sort}}(\text{f}_{\text{p},0})[\text{h}]}\right)$
20:        $a = \frac{1}{p}\mathcal{F}^{sort}(f_{p,0})[h]$
21:        $R \leftarrow R \cup (\tilde{w}, a)$
22:      end if
23:    end for
24:    prune small coefficients from $R$
25:    $i \leftarrow i + 1$
26:  end while
27: end procedure
```

2.2 Multidimensioanl Problem Setting and Worst Case Scenario

In this section, the multidimensional problem is introduced. Let us consider a function $f : \mathbb{R}^d \to \mathbb{C}$ such that

$$f(\mathbf{t}) \;=\; \sum_{j=1}^{k} a_j e^{2\pi i \mathbf{w}_j \cdot \mathbf{t}}, \tag{2.8}$$

where $\mathbf{w}_j \in ([-N/2, N/2) \cap \mathbb{Z})^d$ and $a_j \in \mathbb{C}$. That is, from (2.1), t is replaced by the d-dimensional phase or time vector $\mathbf{t}$, frequency w_j is replaced by the frequency vector $\mathbf{w}_j$ and thus the operator between $\mathbf{w}_j$ and $\mathbf{t}$ is a dot product instead of simple scalar multiplication. We can see that this is a natural extension of the one-dimensional sparse problem. As in the 1D setting, if we find a_j and $\mathbf{w}_j$, we recover the function f.

However, since our time and frequency domain have changed, we cannot apply the previous algorithm directly. If we project the frequencies onto a line, then we can apply the former algorithm so that we can retain sublinear time complexity. Since the operator between frequency and time vectors is a dot product, we can convert projection of frequencies to that of time. For example, we consider the projection onto the first axis, that is, we put the last $d-1$ elements of time vectors as 0. If the projection is one-to-one, i.e., there is no collision, then we can apply the algorithm in Section 2.1 to this projected function to recover the first element of frequency vectors. If there is a collision on the first axis, then we can try another projection onto ith axis, $i = 2, 3, \cdots, d$, until there are no collisions. We introduce in latter sections how to recover the corresponding remaining $d-1$ elements by extending the test to determine the occurrence of a collision in Section 2.1. Furthermore, to reduce the chance of a collision through projections, we use an "unwrapping method" which unwraps frequencies onto a lower dimension guaranteeing a one-to-one projection. There is both a "full unwrapping" and a "partial unwrapping" method, which are explained in later sections.

We shall call projections onto any one of the coordinate axes a *parallel projection*. The worst case is where there is a collision for every parallel projection. This obviously happens when a subset of frequency vectors form the vertices of a d-dimensional hypercube, but it can happen also with other configurations that require fewer vertices. Then our method cannot recover any of these frequency vectors via parallel projections. To resolve this problem, we introduce *tilted projections*: instead of simple projection onto axes we project frequency vectors onto tilted lines or planes so that there is no collision after the projection. We shall call this the *tilting method* and provide the details in the next section. After introducing these projection methods, we explore which combination of these methods is likely to be optimal.

3 TWO DIMENSIONAL SUBLINEAR SPARSE FOURIER ALGORITHM

As means of explanation, we introduce the two-dimensional case in this section and extend this to higher dimensions in Section 4. The basic two-dimensional algorithm using a parallel projection is introduced in Section 3.1, the full unwrapping method is introduced in Section 3.2 and the tilting method for the worst case is discussed in Section 3.3.

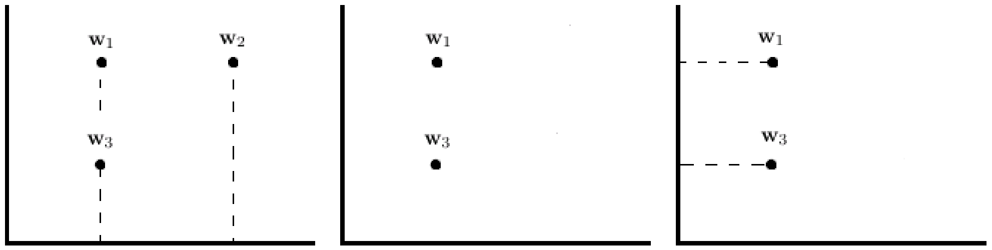


Figure 2: Process of the basic algorithm in 2D

3.1 Basic Algorithm Using Parallel Projection

Our basic two-dimensional sublinear algorithm excludes certain worst case scenarios. In most cases, we can recover frequencies in the 2-D plane by projecting them onto each horizontal axis or vertical axis. Figure 2 is a simple illustration. Here we have three frequency vectors where $\mathbf{w}_1$ and $\mathbf{w}_3$ are colliding with each other when they are projected onto the horizontal axis and $\mathbf{w}_1$ and $\mathbf{w}_2$ are when they are projected onto the vertical axis. The first step is to project the frequency vectors onto the horizontal axis and recover $\mathbf{w}_2$ and its corresponding coefficient a_2 only, since it is not colliding. After subtracting $\mathbf{w}_2$ from the data, we project the remaining frequency vectors onto the vertical axis and then find both $\mathbf{w}_1$ and $\mathbf{w}_3$.

Now let us consider the generalized two-dimensional basic algorithm. Assume that we have a two-dimensional function f with sparsity k :

$$f(\mathbf{t}) \;=\; \sum_{j=1}^{k} a_j e^{2\pi i \mathbf{w}_j \cdot \mathbf{t}}, \quad a_j \in \mathbb{C}, \quad \mathbf{w}_j \in \Big(\Big[-\frac{N}{2}, \frac{N}{2}\Big) \cap \mathbb{Z}\Big)^2. \tag{3.1}$$

For now, let us focus on one frequency vector with index j' which is not collided with any other pairs when they are projected onto the horizontal axis. To clarify put $\mathbf{t} = (t_1, 0)$

with $\mathbf{w}_j = (w_{j1}, w_{j2})$ into (3.1),

$$f^1(t_1) \ := \ f(t_1, 0) \ = \ \sum_{j=1}^{k} a_j e^{2\pi i w_{j1} t_1}, \tag{3.2}$$

which gives the same effect of parallel projection of frequency vectors. Now, we can consider this function as a one-dimensional function f^1 so that we can use the original one dimensional sparse Fourier algorithm to find the first component of $\mathbf{w}_{j'}$. We get the samples $\mathbf{f}^1_{p,0}$ and $\mathbf{f}^1_{p,\epsilon}$ with and without shift by ϵ. We can find these in the form of sequences in (2.2), apply the DFT to them, and then recover the first component of the frequency pair and its coefficient as follows,

$$\begin{aligned} w_{j'1} &= \frac{1}{2\pi\epsilon}\mathrm{Arg}\Big(\frac{\mathcal{F}(\mathbf{f}^1_{\mathrm{p},\epsilon})[\mathrm{h}]}{\mathcal{F}(\mathbf{f}^1_{\mathrm{p},0})[\mathrm{h}]}\Big), \\ a_{j'} &= \frac{1}{p}\mathcal{F}(\mathbf{f}^1_{p,0})[h]. \end{aligned} \tag{3.3}$$

At the same time, we need to find the second component. In (3.2), we replace 0 by ϵ. Then

$$\begin{aligned} f^2(t_1) \ &:= \ f(t_1, \epsilon) \ = \ \sum_{j=1}^{k} a_j e^{2\pi i (w_{j1} t_1 + w_{j2}\epsilon)}, \\ \mathcal{F}(\mathbf{f}^2_{p,\epsilon})[h] \ &= \ p a_{j'} e^{2\pi i w_{j'2}\epsilon}, \\ w_{j'2} \ &= \ \frac{1}{2\pi\epsilon}\mathrm{Arg}\Big(\frac{\mathcal{F}(\mathbf{f}^2_{p,\epsilon})[h]}{\mathcal{F}(\mathbf{f}^1_{p,0})[h]}\Big), \end{aligned} \tag{3.4}$$

where $\mathbf{f}^2_{p,\epsilon}$ are samples shifted by ϵ in the vertical sense with rate $1/p$ from the function f^2. (3.3) holds only when $w_{j'1}$ is the only one congruent to h modulo p among every first component of k frequency pairs and (3.4) holds only when the previous condition is satisfied and $\mathbf{w}_{j'} = (w_{j'1}, w_{j'2})$ does not collide with other frequency pairs from the parallel projection.

Now we have two kinds of collisions. The first one is from taking modulo p after the parallel projection and the second one is from the projection. Thus we need two tests. To determine whether there are both kinds of collisions, we use similar tests as (2.7). If there are at least two different w_{j1} congruent to h modulo p, then the second equality in the following is not satisfied for almost all ϵ, just as (2.7),

$$\frac{|\mathcal{F}(\mathbf{f}^1_{p,\epsilon})[h]|}{|\mathcal{F}(\mathbf{f}^1_{p,0})[h]|} \ = \ \frac{|p\sum_{w_{j1}=h(\bmod p)} a_j e^{2\pi i \epsilon w_{j1}}|}{|p\sum_{w_{j1}=h(\bmod p)} a_j|} \ = \ 1. \tag{3.5}$$

Likewise, if there is a collision from the projection, i.e., the first components w_{j1}'s of at least two frequency vectors are identical and the corresponding w_{j2}'s are different, the following second equality does not hold for almost all ϵ,

$$\frac{|\mathcal{F}(\mathbf{f}^2_{p,\epsilon})[h]|}{|\mathcal{F}(\mathbf{f}^1_{p,0})[h]|} \;=\; \frac{|p \sum_{w_{j1}=h(\bmod p)} a_j e^{2\pi i \epsilon w_{j2}}|}{|p \sum_{w_{j1}=h(\bmod p)} a_j|} \;=\; 1. \tag{3.6}$$

The two tests above are both satisfied only when there is no collision both from taking modulo p and the projection. We use these for the complete recovery of the objective frequencies.

So far we project the frequencies onto the horizontal axis. After we find the non-collided frequencies from the first projection, we subtract a function consisting of found frequencies and their coefficients from the original function f to get a new function. Next we project this new function onto the vertical axis and do a similar process. The difference is to exchange 1 and 2 in the super-indices and sub-indices respectively in (3.2) through (3.6). Again, find the remaining non-collided frequencies, change the axis again and keep doing this until we recover all of the frequencies.

3.2 Full Unwrapping Method

We introduce another kind of projection which is one-to-one. The full unwrapping method uses one-to-one projections onto one-dimensional lines instead of the parallel projection onto axes from the previous method. We consider the k pairs of frequencies (w_{j1}, w_{j2}), $j = 1, 2, \cdots, k$ and transform them as follows

$$(w_{j1}, w_{j2}) \;\to\; w_{j1} + N w_{j2}. \tag{3.7}$$

This transformation in frequency space can be considered as the transformation in phase or time space. That is, from the function in (3.1)

$$g(t) \;:=\; f(t, Nt) \;=\; \sum_{j=1}^{k} a_j e^{2\pi i (w_{j1} + N w_{j2}) t}. \tag{3.8}$$

The function $g(t)$ is a one-dimensional function with sparsity k and bandwidth bounded by N^2. We can apply the algorithm in Section 2.1 on g so that we recover k frequencies of the form on the right side of the arrow in (3.7). Whether unwrapped or not, the coefficients are the same, so we can find them easily. In the end we need to wrap the unwrapped frequencies to get the original pairs. Remember that unwrapping transformation is one-to-one. Thus we can wrap them without any collisions.

Since the pairs of the frequencies are projected onto the one-dimensional line directly, we call this method the "full unwrapping method". Problem with this method occurs when the dimension d gets large. From the above description, we see that after the one-to-one unwrapping the total bandwidth of the two dimensional signal increases from N in each dimension to N^2. If the full unwrapping method is applied to a function in d-dimensions, then to guarantee the one-to-one transformation the bandwidth will be N^d. Theoretically this does not matter. However, since ϵ is dependent on the bandwidth, in the case where d is large, we need to consider the limit of machine precision for practical implementations. As a result, we need to introduce the partial unwrapping method to prevent the bandwidth from becoming too large. The partial unwrapping method is discussed in Section 4.

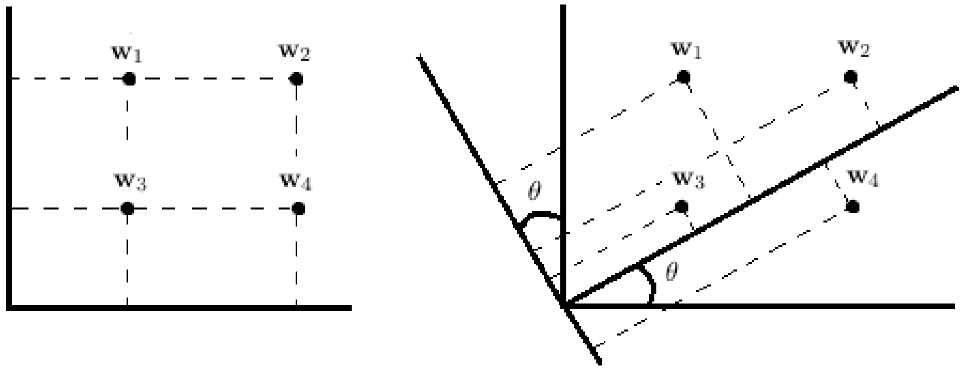


Figure 3: Worst case scenario in $2D$ and solving it through tilting

3.3 Tilting Method for the Worst Case

Up till now, we have assumed that we do not encounter the worst case, i.e., that we do not encounter the case where any frequency pair has collisions from the parallel projection for all coordinate axes. This makes the algorithm break down. The following method is for finding those frequency pairs. Basically, we rotate axes of the frequency plane and thus use a projection onto a one-dimension system which is a tilted line with the tilt chosen so that there are no collisions. If the horizontal and vertical axes are rotated with angle θ then the frequency pair $\mathbf{w}_j = (w_{j1}, w_{j2})$ can be relabeled with new coordinates as the right side of the following

$$(w_{j1}, w_{j2}) \rightarrow (\cos\theta w_{j1} - \sin\theta w_{j2}, \sin\theta w_{j1} + \cos\theta w_{j2}). \tag{3.9}$$

In phase-sense, this rotation can be written as

$$\begin{aligned} g(\tilde{t_1}, \tilde{t_2}) &:= f(\cos\theta\tilde{t_1} + \sin\theta\tilde{t_2}, -\sin\theta\tilde{t_1} + \cos\theta\tilde{t_2}) \\ &= \sum_{j=1}^{k} a_j e^{2\pi i\{w_{j1}(\cos\theta\tilde{t_1}+\sin\theta\tilde{t_2})+w_{j2}(-\sin\theta\tilde{t_1}+\cos\theta\tilde{t_2})\}}, \\ &= \sum_{j=1}^{k} a_j e^{2\pi i\{(\cos\theta w_{j1}-\sin\theta w_{j2})\tilde{t}_1+(\sin\theta w_{j1}+\cos\theta w_{j2})\tilde{t}_2\}}. \end{aligned} \tag{3.10}$$

We can apply the basic algorithm in Section 3.1 to the function g to get the frequency pairs in the form of the right side of the arrow in (3.9).

One problem we face is that the components of the projected frequency pairs should be integers to apply the method, since we assume integer frequencies in the first place. To guarantee injectivity for both projections, $\tan\theta$ should be irrational, however, the projected frequencies become irrational. Thus, we should try rational $\tan\theta$, and to make them integer it is inevitable to increase the bandwidth by multiplying the least common multiple of the denominators of $\sin\theta$ and $\cos\theta$. We assume the following with integers a, b and c

$$\sin\theta = \frac{a}{c}, \quad \cos\theta = \frac{b}{c}, \quad \gcd(a,c) = \gcd(b,c) = 1. \tag{3.11}$$

Multiplying c to both inputs in the right-hand side of (3.10) we obtain

$$\begin{aligned} \hat{g}(\tilde{t_1}, \tilde{t_2}) &:= f(c(\cos\theta\tilde{t_1} + \sin\theta\tilde{t_2}), c(-\sin\theta\tilde{t_1} + \cos\theta\tilde{t_2})) \\ &= \sum_{j=1}^{k} a_j e^{2\pi i\{(c\cos\theta w_{j1}-c\sin\theta w_{j2})\tilde{t}_1+(c\sin\theta w_{j1}+c\cos\theta w_{j2})\tilde{t}_2\}}. \end{aligned} \tag{3.12}$$

As long as there is no collision for at least one projection, the frequency pairs, $(c\cos\theta w_{j1} - c\sin\theta w_{j2}, c\sin\theta w_{j1} + c\cos\theta w_{j2})$, can be found by applying the basic algorithm in Section 3.2 on $\tilde{g}$. Due to machine precision the integer c should not be too large, or the bandwidth gets too large resulting in failure of the algorithm. If four pairs of frequencies are at vertices of a rectangle aligned with coordinate axes before the rotation, then they are not aligned after the rotation with $0 < \theta < \pi/2$. Thus we can assure finding whole frequencies whether they are in the worst case or not.

The pseudo code of the 2D tilting method is shown in Algorithm 2. The lines 14 and 15 mean that each frequency pair (w_{j1}, w_{j2}) is rotated by a matrix [cos $-$sin; sin cos] and scaled to make the rotated components integers. Thus we first find the frequency pairs in the form of $\tilde{\mathbf{w}} = (\cos w_{j1} - \sin w_{j2}, \sin w_{j1} + \cos w_{j2})$ and after finding all of them, we rotate them back into the original pairs with the matrix [cos sin; $-$sin cos] in line 39.

This tilting method is a straight forward way to resolve the worst case problem. We only introduced the tilting method in the two-dimensional case, but the idea of rotating the axes can be extended to the general d-dimensional case with some effort. On the other hand, we may notice that the probability of this worst case is very low, especially when the number of dimensions d is large. Its details are shown in Section 4. Thus, as we recover the frequencies as much as possible from the basic algorithm. If we cannot get any frequency pairs for several projection switching among each axis then, assuming that the worst case happens, we apply the tilting method with several angles until all k frequency pairs are found.

4 PARTIAL UNWRAPPING METHOD FOR HIGH DIMENSIONAL ALGORITHM

In this section we present the *partial unwrapping* method for a sublinear sparse Fourier algorithm for very high dimensional data. As we have already mentioned, while full unwrapping converts a multi-dimensional problem into a single dimensional problem, it is severely limited in its viability when the dimension is large or when the bandwidth is already high because of the increased bandwidth. Partial unwrapping is introduced here to overcome this problem and other problems. In Section 4.1 we give a four dimensional version of the algorithm using the partial unwrapping method as well as a generalize it to d dimension. In Section 4.2, the probability of the worst case in d dimension is analyzed.

4.1 Partial Unwrapping Method

To see the benefit of partial unwrapping we need to examine the main difficulties we may encounter in developing sublinear sparse Fourier algorithms. For this let us consider a hypothetical case of sparse FFT where we have $k = 100$ frequencies in a 20-dimensional Fourier series distributed in $[-10, 10)^{20}$. When we perform the parallel projection method, because the bandwidth is small, there will be a lot of collisions after the projections. It is often impossible to separate any frequency after each projection, and the task could thus not be completed. This, ironically, is a *curse of small bandwidth* for sparse Fourier algorithm. On the other hand, if we do the full unwrapping we would have increased the bandwidth to $N = 20^{20}$, which is impossible to do within reasonable accuracy because N is too large.

However, a partial unwrapping would reap the benefit of both worlds. Let us now break down the 20 dimensions into 5 lower 4-dimensional subspaces, namely we write

$$[-10,10)^{20} = \left([-10,10)^4\right)^5.$$

In each subspace we perform the full unwrapping, which yields bandwidth $N = 20^4 = 160,000$ in the subspace. This bandwidth N is large enough compared with k, so when projection method is used there is a very good probability that collision will occur only for a small percentage of the frequencies, allowing them to be reconstructed. On the other hand, N is not so large that the phase-shift method will incur significant error.

One of the greatest advantage of partial unwrapping is to turn the curse of dimensionality into the *blessing* of dimensionality.

Note that in the above example, the 4 dimensions that for any of the subspaces do not have to follow the natural order. By randomizing (if necessary) the order of the dimensions it may achieve the same goal as the tilting method would. Also note that the dimension for each subspace needs not be uniform. For example, we can break down the above 20-dimensional example into four 3-dimensional subspaces and two 4-dimensional subspaces, i.e.

$$[-10,10)^{20} = \left([-10,10)^3\right)^4 \times \left([-10,10)^4\right)^2.$$

This will lead to further flexibility.

4.1.1 Example of 4-D Case

Before introducing the generalized partial unwrapping algorithm for dimension d, let us think about the simple case of 4 dimensions. We assume that k frequency vectors are in 4-dimensional space ($d = 4$). Then, a function f constructed from these frequency vectors is as follows,

$$f(\mathbf{t}) = \sum_{j=1}^{k} a_j e^{2\pi i \mathbf{w}_j \cdot \mathbf{t}},\ a_j \in \mathbb{C},\ \mathbf{w}_j \in \left(\Big[-\frac{N}{2}, \frac{N}{2}\Big) \cap \mathbb{Z}\right)^4. \tag{4.1}$$

Since $4 = 2 \times 2$, the frequency pairs of the two-two dimensional spaces are both unwrapped onto one-dimensional spaces. Here, 4 dimensions is projected onto 2 dimensions as follows

$$\begin{aligned} g(t_1, t_2) &:= f(t_1, Nt_1, t_2, Nt_2) \\ &= \sum_{j=1}^{k} a_j e^{2\pi i \{(w_{j1}+Nw_{j2})t_1 + (w_{j3}+Nw_{j4})t_2\}} \\ &= \sum_{j=1}^{k} a_j e^{2\pi i (\tilde{w}_{j1} t_1 + \tilde{w}_{j2} t_2)}, \end{aligned} \tag{4.2}$$

where $\tilde{w}_{j1} = w_{j1} + Nw_{j2}$ and $\tilde{w}_{j2} = w_{j3} + Nw_{j4}$. Note that this projection is one-to-one so as to guarantee the inverse transformation.

Now we can apply the basic projection method in Section 3.1 to this function g redefined as the 2-dimensional one. To make this algorithm work, $\tilde{\mathbf{w}}_j = (\tilde{w}_{j1}, \tilde{w}_{j2})$ should not collide with any other frequency pair after the projection onto either the horizontal or vertical axes. If not, we can consider using the tilting method. After finding all the frequencies in the form of $(\tilde{w}_{j1}, \tilde{w}_{j2})$, it can be transformed to $(w_{j1}, w_{j2}, w_{j3}, w_{j4})$.

4.1.2 Generalization

We introduce the final version of the multidimensional algorithm in this section. Its pseudo code and detailed explanation are given in Algorithm 3 and Section 5.1, respectively. We start with a d-dimensional function f,

$$f(\mathbf{t}) = \sum_{j=1}^{k} a_j e^{2\pi i \mathbf{w}_j \cdot \mathbf{t}}, \quad a_j \in \mathbb{C}, \quad \mathbf{w}_j \in \Big(\Big[-\frac{N}{2}, \frac{N}{2}\Big) \cap \mathbb{Z}\Big)^d. \tag{4.3}$$

Let us assume that d can be divided into d_1 and d_2 - the case of d being a prime number will be mentioned at the end of this section. The domain of frequencies can be considered as $([-N/2, N/2) \cap \mathbb{Z})^d = (([-N/2, N/2) \cap \mathbb{Z})^{d_1})^{d_2}$ and $([-N/2, N/2) \cap \mathbb{Z})^{d_1}$ will be reduced to one dimension, as d_1 is in the 4 dimensional case. Each of the d_1 elements of a frequency vector, $\mathbf{w}_j = (w_{j1}, w_{j2}, \cdots, w_{jd})$, is unwrapped as

$$\begin{aligned} &\big(w_{j(d_1q+1)}, w_{j(d_1q+2)}, w_{j(d_1q+3)}, \cdots, w_{j(d_1q+d_1)}\big) \\ &\to\ w_{j(d_1q+1)} + Nw_{j(d_1q+2)} + N^2 w_{j(d_1q+3)} + \cdots + N^{d_1-1} w_{j(d_1q+d_1)} \\ &=:\ \tilde{w}_{j(q+1)} \end{aligned} \tag{4.4}$$

with $q = 0, 1, 2, \cdots, d_2-1$, increasing the respective bandwidth from N to N^{d_1} and having injectivity. We rewrite this transformation in terms of the phase. With $\mathbf{t} = (t_1, t_2, \cdots, t_d)$ and put the following into t_ℓ

$$N^{R(\ell-1,d_1)} \tilde{t}_{Q(\ell,d_1)} \tag{4.5}$$

for all $\ell = 1, 2, \cdots, d$, where $R(\ell-1, d_1)$ and $Q(\ell, d_1)$ are the remainder from dividing $\ell-1$ by d_1 and quotient from dividing ℓ by d_1 respectively, and $\tilde{\mathbf{t}} = (\tilde{t}_1, \tilde{t}_2, \cdots, \tilde{t}_{d_2})$ is a phase vector in d_2 dimensions after projection. Define a function g on d_2 dimension as

$$\begin{aligned} g(\tilde{\mathbf{t}}) &:= f(\cdots, N^{R(\ell-1,d_1)}\tilde{t}_{Q(\ell,d_1)}, \cdots) \\ &= \sum_{j=1}^{k} a_j e^{2\pi i \sum_{q=0}^{d_2-1} \left(\sum_{r=0}^{d_1-1} w_{j(d_1 q+r+1)} N^r\right)\tilde{t}_{q+1}}, \end{aligned} \tag{4.6}$$

where $N^{R(\ell-1,d_1)}\tilde{t}_{Q(\ell,d_1)}$ is the ℓth element of the input of f. If we project frequency vectors of g onto the mth axis, then the nth element of a frequency vector $\tilde{\mathbf{w}}_j$ can be found in the following computation,

$$\begin{aligned} \mathbf{g}_{p,\epsilon}^{m,n} &= \Big(g(0\mathbf{e}_m + \epsilon\mathbf{e}_n), g(\frac{1}{p}\mathbf{e}_m + \epsilon\mathbf{e}_n), \cdots, g(\frac{p-1}{p}\mathbf{e}_m + \epsilon\mathbf{e}_n)\Big) \\ \tilde{w}_{jn} &= \frac{1}{2\pi\epsilon}\mathrm{Arg}\Big(\frac{\mathcal{F}(\mathbf{g}_{p,\epsilon}^{m,n})[h]}{\mathcal{F}(\mathbf{g}_{p,0}^{m,n})[h]}\Big) \\ a_j &= \frac{1}{p}\mathcal{F}(\mathbf{g}_{p,0}^{m,n})[h], \end{aligned} \tag{4.7}$$

where $\mathbf{e}_m$ is the m-th unit vector with length d_2, i.e., all elements are zero except the m-th one with entry 1. (4.7) holds as long as $\tilde{w}_{jn}$ is the only one congruent to h modulo p among all n-th elements of the frequency vectors and $\tilde{\mathbf{w}}_j$ does not collide with any other frequency vector due to the projection onto the m-th axis. The test for checking whether these conditions are satisfied is

$$\frac{|\mathcal{F}(\mathbf{g}_{p,\epsilon}^{m,n})[h]|}{|\mathcal{F}(\mathbf{g}_{p,0}^{m,n})[h]|} = 1 \tag{4.8}$$

for all $1 \leq n \leq d_2$. The projections onto the m-th axis, where $m = 1, \cdots, d_2$, take turns until we recover all frequency vectors and their coefficients. After that we wrap the unwrapped frequency vectors up from d_2 to d dimension. Since the unwrapping transformation is one-to-one, this inverse transformation is well-defined.

So far, we assumed that dimension d can be divided into two integers, d_1 and d_2. For the case that d is a prime number or both d_1 and d_2 are so large that the unwrapped data has a bandwidth such that ϵ is below the machine precision, a strategy of divide and conquer can be applied. In that case we can think about applying partial unwrapping method in a way that each unwrapped component has a different size of bandwidth. If d is 3, for example, then we can unwrap the first two components of the frequency vector onto one dimension and the last one lies in the same dimension. In that case, the unwrapped data is in two dimensions, and the bandwidth of the first component is bounded by N^2

and that of second component is bounded by N. In this case we can choose a shift $\epsilon < 1/N^2$ where N^2 is the largest bandwidth. We can extend this to the general case, so the partial unwrapping method has a variety of choices balancing the bandwidth and machine precision.

4.2 Probability of Worst Case Scenario

In this section, we give an upper bound of the probability of the worst case assuming that we randomly choose a partial unwrapping method. As addressed in the Section 4.1, there is flexibility in choosing certain partial unwrapping method. Assuming a certain partial unwrapping method and considering a stronger condition to avoid its failure, we can find the upper bound of the probability of the worst case where there is a collision for each parallel projection.

For simple explanation, consider a two dimensional problem. Choosing the first frequency vector (w_{11}, w_{12}) on a two dimensional plane, if the second frequency vector, (w_{21}, w_{22}), is not on the vertical line crossing $(w_{11}, 0)$ and the horizontal line crossing $(0, w_{12})$, then the projection method works. Then if the third frequency vector is not on four lines, those two lines mentioned before, the vertical line crossing $(w_{21}, 0)$ and the horizontal line crossing $(0, w_{22})$, then again the projection method works. We keep choosing next frequency vector in this way, excluding the lines containing previous frequencies. Thus, letting such event A, the probability that the projection method fails is bounded above by $1 - \mathbb{P}(A)$.

Generally, let us assume that we randomly choose a partial unwrapping, without loss of generality, the total dimension is $d = d_1 + d_2 + \cdots + d_r$ where r is the number of subspaces and $d_1, d_2, \cdots, d_r$ are the dimensions of each subspace. That is, partially unwrapped frequency vectors are in $r < d$ dimension and each bandwidth is $N^{d_1}, N^{d_2}, \cdots, N^{d_r}$, respectively, which is integer strictly larger than 1. Then, the failure probability of projection method is bounded above by

$$\begin{aligned}
1-\prod_{j=1}^{k}\mathbb{P}(A_j) &\le 1-\prod_{j=1}^{k}\frac{N^d-(i-1)(N^{d_1}+N^{d_2}+\cdots+N^{d_r})}{N^d}\\
&= 1-\frac{1}{\tau^{s-1}}\frac{(\tau-1)!}{(\tau-(s-2))!} \qquad \left(\tau := \frac{N^d}{N^{d_1}+N^{d_2}+\cdots+N^{d_r}}\right)\\
&\sim 1-\frac{1}{\tau^{s-1}}\sqrt{\frac{\tau-1}{\tau-(s-2)}}\frac{\left(\frac{\tau-1}{e}\right)^{\tau-1}}{\left(\frac{\tau-(s-2)}{e}\right)^{\tau-(s-2)}} \qquad \text{(Sterling's formula)}\\
&= 1-\frac{1}{e^{s-3}}\frac{(\tau-1)^{\tau-1/2}}{\tau^{s-1}(\tau-s+2))^{\tau-s+5/2}} \qquad (4.9)
\end{aligned}$$

where A_i is the event that we choose ith frequency not on the lines, crossing formerly chosen frequency vectors and parallel to each coordinate axis. Noting $N^d = N^{d_1}\times N^{d_2}\times\cdots\times N^{d_r}$, sparsity k is relatively small compared to N^d, and τ is large, we can see that the upper bound above gets closer to 0 as d or N grows to infinity.

5 EMPIRICAL RESULT

The partial unwrapping method is implemented in the C language. The pseudo code of this algorithm is shown in Algorithm 3. It is explained in detail in Section 5.1. In our experiment, dimension d is set to 100 and 1000, d_1 is 5 and d_2 is 20 and 200, accordingly. Frequency bandwidth N in each dimension is 20 and sparsity k varies as $1, 2, 2^2, \cdots, 2^{10}$. The value of ϵ for shifting is set to $1/2N^{d_1}$ and the constant number c determining the prime number p is set to 5.

We randomly choose k frequency vectors $\mathbf{w}_j \in \left(\left[-\frac{N}{2},\frac{N}{2}\right)\cap\mathbb{Z}\right)^d$ and corresponding coefficients $a_j = e^{2\pi i\theta_j}\in\mathbb{C}$ from randomly chosen angles $\theta_j\in[0,1)$ so that the magnitude of each a_j is 1. For each d and k we have 100 trials. We get the result by averaging ℓ^2 errors, the number of samples used and CPU TICKS out of 100 trials.

Since it is difficult to implement high dimensional FFT and there is no practical high dimensional sparse Fourier transform it is hard to compare the result of ours with others, as so far no one else was able to do FFT on this large data set. Thus we cannot help but show ours only. From Figure 4 we can see that the average ℓ^2 errors are below 2^{-52}. Those errors are from all differences of frequency vectors and coefficients of the original and recovered values. Since all frequency components are integers and thus the least difference is 1, we can conclude that our algorithm recover the frequency vectors perfectly. Those errors are only from the coefficients. In Figure 5 the average sampling complexity is shown. We can see that the logarithm of the number of samples is almost proportional to that of sparsity. Note that the traditional FFT would show the same sampling complexity even though sparsity k varies since it only depends on the bandwidth N and dimension d. In Figure 6 the average CPU TICKS are shown. We can see the the logarithm of CPU TICKS is also almost proportional to that of sparsity. Note that the traditional FFT might show the same CPU TICKS even though sparsity k varies since it also depends on the bandwidth N and dimension d only.

5.1 Algorithm

In this section, the explanation of Algorithm 3 is given. In [9] several versions of 1D algorithms are shown. Among them, non-adaptive and adaptive algorithms are introduced where the input function f is not modified throughout the whole iteration, and is modified by subtracting the function constructed from the data in registry R, respectively. In our multidimensional algorithm, however, the adaptive version is mandatory since excluding the contribution of the currently recovered data is the key of our algorithm to avoid the collision of frequencies through projections, whose simple pictorial description is given in Figure 2. In Algorithm 3, the function g is the one constructed from the data in the registry R.

Our algorithm begins with entering inputs, a function f, a constant number c determining p, a sparsity k, a bandwidth N of each dimension, a dimension d, factors d_1 and d_2 of d and a shifting number $\epsilon < 1/N$. For each iteration of the algorithm, the number of frequencies to find is updated as $k^* = k - |R|$. It stops when $|R|$ becomes equal to the sparsity k. The prime number p is determined depending on this new k^* as $p \geq ck^*$ and is chosen as the next larger prime number. The lines 13 and 14 of Algorithm 3 represent the partial unwrapping and sampling with and without shifting from the function where the contribution of former data is excluded. After applying the FFT on each sequence, sorting them according to the magnitude of $\mathcal{F}(f_{p,0}^{m,n})$, we check the ratio between the FFT's of the unshifted and shifted sequences to determined whether there is a collision, either from modulo p or a parallel projection. If all tests are passed, then we find each frequency component and corresponding coefficient for the data that passed and store them in R. After several iterations, we find all the data and the final wrapping process gives the original frequency vectors in d dimensions.

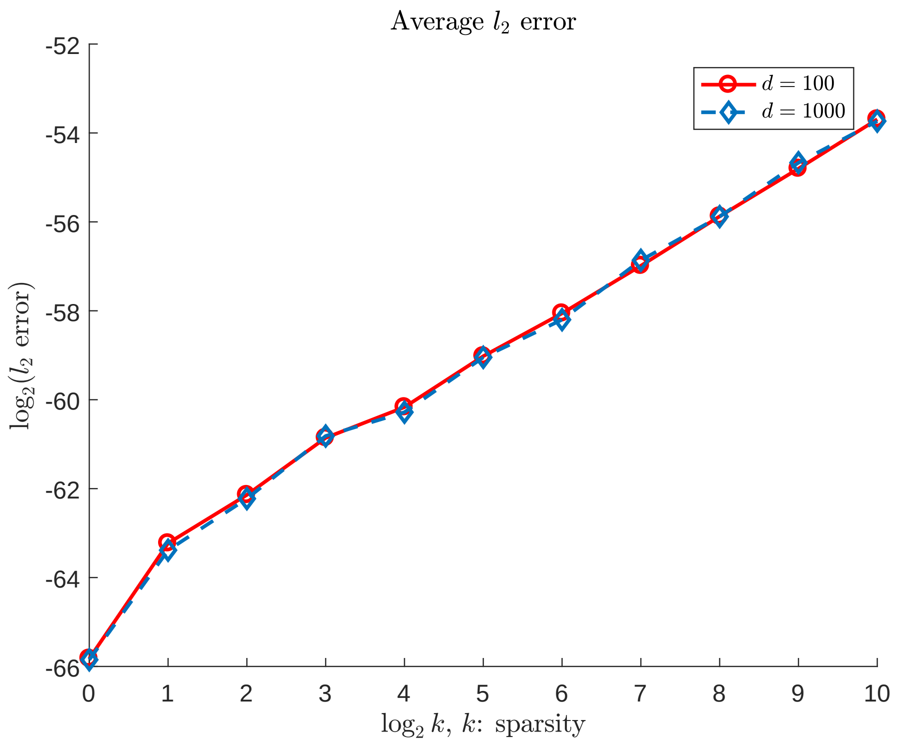


Figure 4: Average ℓ^2 error

5.2 Accuracy

We assume that there is no noise on the data that we want to recover. Figure 4 shows that we can find frequencies perfectly and the ℓ^2 error from coefficients are significantly small. This error is what we average out over 100 trials for each $d = 100, 1000$ and $k = 1, 2^1, 2^2, \cdots, 2^{10}$ when N is fixed to 20. The horizontal axis represents the logarithm with base 2 of k and the vertical axis represents the logarithm with base 2 of the ℓ^2 error. It is increasing as the sparsity k is increasing since the number of nonzero coefficients increases. The red graph in the Figure 4 shows the error when the number of dimensions is 100 and the blue one shows the error when the number of dimensions is 1000. Thus, we see that the errors are not substantially impacted by the dimensions.

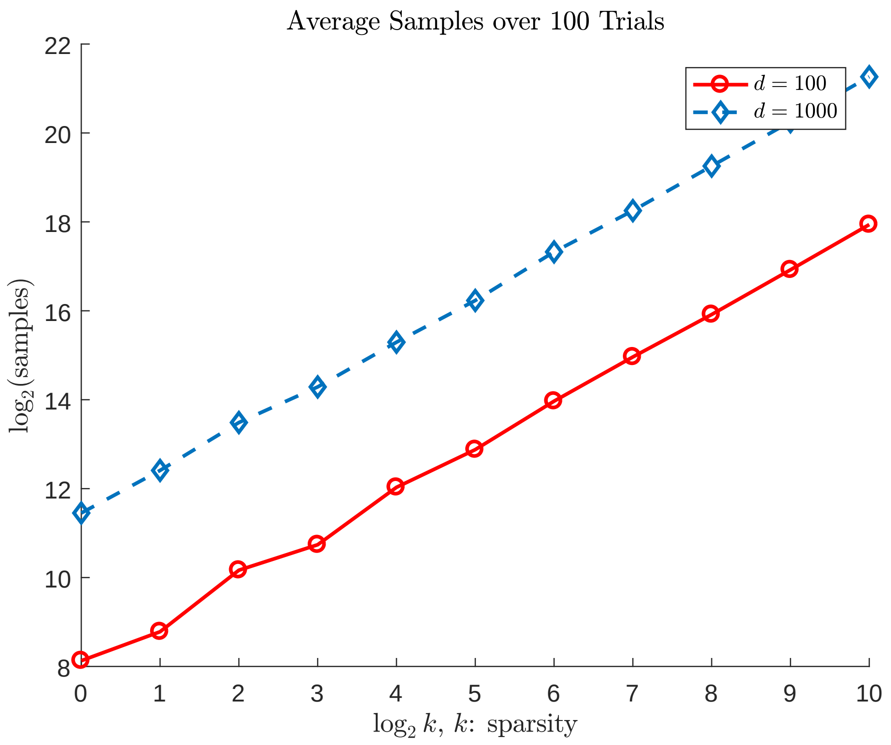


Figure 5: Average sampling complexity

5.3 Sampling Complexity

Figure 5 shows the sampling complexity of our algorithm averaged out from 100 tests for each dimension and sparsity. The horizontal axis means the logarithm with base 2 of k and the vertical axis represents the logarithm with base 2 of the total number of samples from the randomly constructed function which are used to find all frequencies and coefficients. The red graph in the Figure 5 shows the sampling complexity when the number of dimensions is 100 and the blue one shows the one when the number of dimensions is 1000. Both graphs increase as k increases. When d is large, we see that it requires more samples since there are more frequency components to find. From the graphs, we see that the scaling seems to be proportional to d.

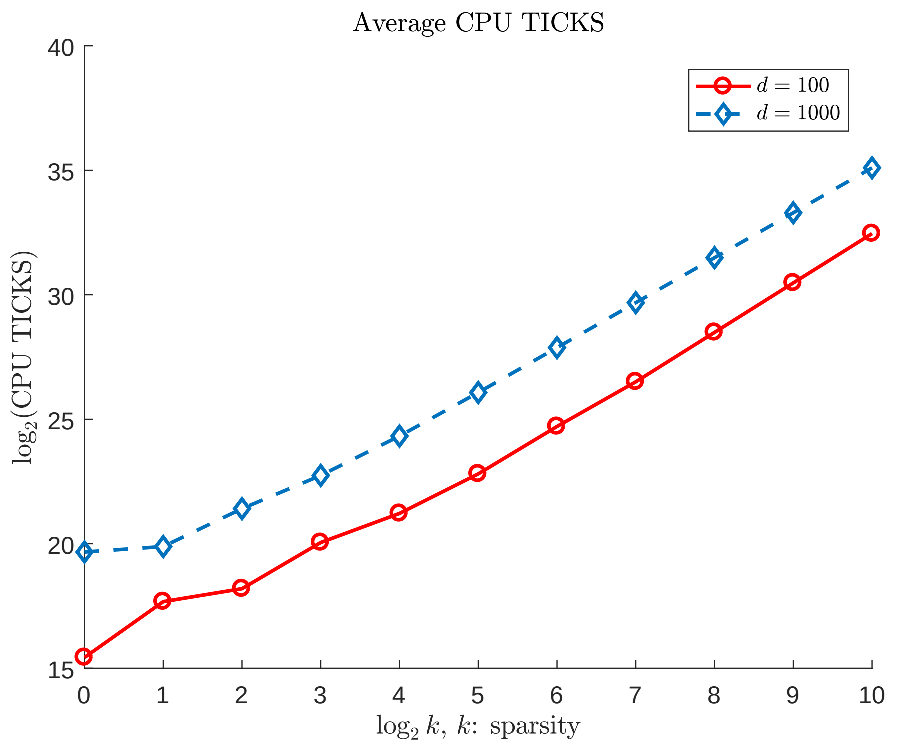


Figure 6: Average CPU TICKS

5.4 Runtime Complexity

In Figure 6, we plot the runtime complexity of the main part of our algorithm averaged over 100 tests for each dimension and sparsity. "Main part" means that we have excluded the time for constructing a function consisting of frequencies and coefficients and the time associated with getting samples from it. The horizontal axis is the logarithm, base 2, of k and the vertical axis is the logarithm, base 2, of CPU TICKS. The red curve shows the runtime when we set the number of dimensions to 100 and the blue one shows the same thing when the number of dimensions to 1000. Both plots increase as k increases. When d is larger, the plots show that it takes more time to run the algorithm. From the graphs we see that the runtime looks proportional to d.

Unfortunately, the sampling process of getting the samples from continuous functions dominates the runtime of the whole algorithm instead of the main algorithm. To show the runtime of our main algorithm, however, we showed CPU TICKS without sampling process. Reducing the time for sampling is still a problem. In [8] the fully discrete Fourier transform is introduced that we expect to use to reduce it. Exploring how to use this will be one part of our future work.

6 CONCLUSION

In this paper we show how to extend our deterministic $1D$ sublinear sparse Fourier algorithm to the general d dimensional case. The method projects d dimensional frequency vectors onto lower dimensions. In this process we encounter several obstacles. Thus we introduced "tilting method" for the worst case problems and the "partial unwrapping method" to reduce the chance of collisions and to increase the frequency bandwidth within the limit of computation. In this way we can overcome the obstacles as well as maintain the advantage of the $1D$ algorithm. In [9] the sampling complexity is $O(k)$ and the runtime complexity is $O(k\log k)$. Extended this estimation from our $1D$ algorithm, we have $O(dk)$ sampling complexity and a runtime complexity of $O(dk\log k)$.

Multidimensional sparse Fourier algorithms have not been discussed much so far, so there is a lot of room for future work. The algorithms in this paper are for recovering data from a noiseless environment only. However most of the actual data contains noise. Thus, the next step will be developing an algorithm for noisy multidimensional data. As mentioned in the previous section, reducing sampling time is another problem to consider. Furthermore, algorithms for fully discrete or nonuniform data will be explored. In the end, it is expected that we apply them to real problems like astrophysical data or MRI data.

ACKNOWLEDGEMENTS We would like to thank Mark Iwen for his valuable advice. This research is supported in part by AFOSR grants FA9550-11-1-0281, FA9550-12-1-0343 and FA9550-12-1-0455, NSF grant DMS-1115709, and MSU Foundation grant SPG-RG100059, as well as Hong Kong Research Grant Council grants 16306415 and 16317416.

Algorithm 2 2D Sparse Fourier Algorithm with Tilting Method Pseudo Code

```
1: procedure 2DTiltedPhaseshift
2: Input: $f, c, k, N, d, \epsilon$, integers $base$, $height$, $hypo$
3: Output: $R$
4:    $R \leftarrow \emptyset$
5:    $i \leftarrow 1$
6:    $\cos \leftarrow base$, $\sin \leftarrow height$
7:    while $|R| < k$ do
8:       $k^* \leftarrow k - |R|$
9:       $p \leftarrow ith$ prime number $\geq ck^*$
10:      $m \leftarrow (i \bmod 2)+1$
11:      $g(\mathbf{t}) = \sum_{(\mathbf{w},a_{\mathbf{w}})\in R} a_{\mathbf{w}} e^{2\pi i \mathbf{w}\cdot\mathbf{t}}$
12:      for $n = 1 \to 2$ do
13:         for $h = 1 \to p$ do
14:            $m' \leftarrow m \bmod 2$, $m'' \leftarrow m+1 \bmod 2$, $n' \leftarrow n \bmod 2$, $n'' \leftarrow n+1 \bmod 2$
15:            $f^{m,n}_{p,\epsilon}[h] =$
                 $f((\frac{h-1}{p}m' + \epsilon n')\cos + (\frac{h-1}{p}m'' + \epsilon n'')\sin, -(\frac{h-1}{p}m' + \epsilon n')\sin + (\frac{h-1}{p}m'' + \epsilon n'')\cos)$
                 $-g(\frac{h-1}{p}\mathbf{e}_m + \epsilon\mathbf{e}_n)$
16:            $f^{m,n}_{p,0}[h] = f(\frac{h-1}{p}m'\cos + \frac{h-1}{p}m''\sin, -\frac{h-1}{p}m'\sin + \frac{h-1}{p}m''\cos) - g(\frac{h-1}{p}\mathbf{e}_m)$
17:         end for
18:         $\mathcal{F}(f^{m,n}_{p,\epsilon}) = FFT(f^{m,n}_{p,\epsilon})$
19:         $\mathcal{F}(f^{m,n}_{p,0}) = FFT(f^{m,n}_{p,0})$
20:         $\mathcal{F}^{sort}(f^{m,n}_{p,0}) = SORT(\mathcal{F}(f^{m,n}_{p,0}))$
21:      end for
22:      for $h = 1 \to k$ do
23:         $\ell \leftarrow 0$
24:         for $n = 1 \to 2$ do
25:            if $\left|\frac{|\mathcal{F}^{sort}(f^{m,n}_{p,0})[h]|}{|\mathcal{F}^{sort}(f^{m,n}_{p,\epsilon})[h]|} - 1\right| < \epsilon$ then
26:               $\ell \leftarrow \ell + 1$
27:            end if
28:            $\tilde{w}_n = \frac{1}{2\pi\epsilon}\mathrm{Arg}\left(\frac{\mathcal{F}^{\mathrm{sort}}(\mathrm{f}^{\mathrm{m,n}}_{\mathrm{p},\epsilon})[\mathrm{h}]}{\mathcal{F}^{\mathrm{sort}}(\mathrm{f}^{\mathrm{m,n}}_{\mathrm{p},0})[\mathrm{h}]}\right)$
29:            $a = \frac{1}{p}\mathcal{F}^{sort}(f^{m,n}_{p,0})[h]$
30:         end for
31:         if $\ell == 2$ then
32:            $R \leftarrow R \cup (\tilde{\mathbf{w}}, a)$
33:         end if
34:      end for
35:      prune small coefficients from $R$
36:      $i \leftarrow i + 1$
37:   end while
38:   $\cos \leftarrow \frac{base}{hypo}$, $\sin \leftarrow \frac{height}{hypo}$
39:   rotate each $\tilde{\mathbf{w}}$ back to $\mathbf{w}$ using a matrix [cos sin;$-$sin cos] and restore it in $R$
40: end procedure
```

Algorithm 3 Multidimensional Sparse Fourier Algorithm Pseudo Code

```
procedure MultiPhaseshift
Input:$f, c, k, N, d, d_1, d_2, \epsilon$
Output:$R$
    $R \leftarrow \emptyset$
    $i \leftarrow 1$
    while $|R| < k$ do
        $k^* \leftarrow k - |R|$
        $p \leftarrow ith$ prime number $\geq ck^*$
        $m \leftarrow (i \bmod d_2) + 1$
        $g(\mathbf{t}) = \sum_{(\mathbf{w}, a_{\mathbf{w}}) \in R} a_{\mathbf{w}} e^{2\pi i \mathbf{w} \cdot \mathbf{t}}$
        for $n = 1 \to d_2$ do
            for $h = 1 \to p$ do
                $f_{p,\epsilon}^{m,n}[h] = f(\sum_{\ell=1}^{d_1} N^\ell \frac{h-1}{p} \mathbf{e}_{d_1(m-1)+\ell} + \epsilon \sum_{\ell=1}^{d_1} N^\ell \mathbf{e}_{d_1(n-1)+\ell}) - g(\frac{h-1}{p} \mathbf{e}_m + \epsilon \mathbf{e}_n)$
                $f_{p,0}^{m,n}[h] = f(\sum_{\ell=1}^{d_1} N^\ell \frac{h-1}{p} \mathbf{e}_{d_1(m-1)+\ell}) - g(\frac{h-1}{p} \mathbf{e}_m)$
            end for
            $\mathcal{F}(f_{p,\epsilon}^{m,n}) = FFT(f_{p,\epsilon}^{m,n})$
            $\mathcal{F}(f_{p,0}^{m,n}) = FFT(f_{p,0}^{m,n})$
            $\mathcal{F}^{sort}(f_{p,0}^{m,n}) = SORT(\mathcal{F}(f_{p,0}^{m,n}))$
        end for
        for $h = 1 \to k$ do
            $\ell \leftarrow 0$
            for $n = 1 \to d_2$ do
                if $\left| \frac{|\mathcal{F}^{sort}(f_{p,0}^{m,n})[h]|}{|\mathcal{F}^{sort}(f_{p,\epsilon}^{m,n})[h]|} - 1 \right| < \epsilon$ then
                    $\ell \leftarrow \ell + 1$
                end if
                $\tilde{w}_n = \frac{1}{2\pi\epsilon} \mathrm{Arg}\left( \frac{\mathcal{F}^{\mathrm{sort}}(\mathrm{f}_{\mathrm{p},\epsilon}^{\mathrm{m,n}})[\mathrm{h}]}{\mathcal{F}^{\mathrm{sort}}(\mathrm{f}_{\mathrm{p},0}^{\mathrm{m,n}})[\mathrm{h}]} \right)$
                $a = \frac{1}{p} \mathcal{F}^{sort}(f_{p,0}^{m,n})[h]$
            end for
            if $\ell == d2$ then
                $R \leftarrow R \cup (\tilde{\mathbf{w}}, a)$
            end if
        end for
        prune small coefficients from $R$
        $i \leftarrow i + 1$
    end while
    inverse-transform each $\tilde{\mathbf{w}}$ in $d_2$-D to $\mathbf{w}$ $d$-D and restore it in $R$
end procedure
```

References

[1] Christlieb, A., Lawlor, D., and Wang, Y. A multiscale sub-linear time fourier algorithm for noisy data. *Applied and Computational Harmonic Analysis 40*, 3 (2016), 553–574.

[2] Ghazi, B., Hassanieh, H., Indyk, P., Katabi, D., Price, E., and Shi, L. Sample-optimal average-case sparse fourier transform in two dimensions. In *Communication, Control, and Computing (Allerton), 2013 51st Annual Allerton Conference on* (2013), IEEE, pp. 1258–1265.

[3] Gilbert, A. C., Guha, S., Indyk, P., Muthukrishnan, S., and Strauss, M. Near-optimal sparse fourier representations via sampling. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing* (2002), ACM, pp. 152–161.

[4] Gilbert, A. C., Muthukrishnan, S., and Strauss, M. Improved time bounds for near-optimal sparse fourier representations. In *Optics & Photonics 2005* (2005), International Society for Optics and Photonics, pp. 59141A–59141A.

[5] Greene, C. S., Krishnan, A., Wong, A. K., Ricciotti, E., Zelaya, R. A., Himmelstein, D. S., Zhang, R., Hartmann, B. M., Zaslavsky, E., Sealfon, S. C., et al. Understanding multicellular function and disease with human tissue-specific networks. *Nature genetics 47*, 6 (2015), 569–576.

[6] Hassanieh, H., Indyk, P., Katabi, D., and Price, E. Nearly optimal sparse fourier transform. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing* (2012), ACM, pp. 563–578.

[7] Hassanieh, H., Indyk, P., Katabi, D., and Price, E. Simple and practical algorithm for sparse fourier transform. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms* (2012), Society for Industrial and Applied Mathematics, pp. 1183–1194.

[8] Iwen, M. A. Combinatorial sublinear-time fourier algorithms. *Foundations of Computational Mathematics 10*, 3 (2010), 303–338.

[9] Lawlor, D., Wang, Y., and Christlieb, A. Adaptive sub-linear time fourier algorithms. *Advances in Adaptive Data Analysis 5*, 01 (2013), 1350003.

[10] LSST. Large synoptic survey telescope. `http://www.lsst.org/`.

[11] SDSS. Sloan digital sky survey. `http://www.sdss.org/`.