Sparse Fourier Transforms

Jingxing Wang

September 20, 2024

Contents

1 An Overview on the Application Side

The original development of q-SFT was for genetics engineering purposes, but to understand it we must first discuss the general pipeline of viewing problems through q-SFT. There are many places where Pseudo-q-ary functions play an important role. For instance, a sequence of genes has certain deterministic structure in its protein folding, and the function that decides what gene sequence gets mapped to what protein structure can be viewed as a Pseudo-4-ary function if the folding is represented with a real number.

The Fourier basis, or the input-coefficient pairs, gives us full information about these functions. In many cases, these basis are sparse. Q-SFT bridges the gap between the realization of having a sparse basis and the recovery of this sparse basis. Here, a couple interesting directions can be taken. First, it might be case that the recovery of the original function is of direct interest. This would apply to the gene-protein setting. Another interesting function is related to decision making: let's say you're given a sequence of decisions to make and you accept the first one, reject the second one, reject the third one, and so on. This sequence of "do", "not do", "not do", ... can be viewed as a binary input signal, and the consequence of this sequence of actions can be viewed as the output. In the case where the action-consequence matching are sparse in the Fourier basis, q-SFT can be used to recover the knowing of the consequences of all possible sequences of actions. This goal is more aligned with a traditional machine learning setting. Another direction that can be taken is the attempt to understand why the basis is sparse. This includes studying the relationships between the input signals that have non-zero Fourier coefficients. For instance, maybe they're all low degree. Maybe they represent some structure in the domain of the inputs. Maybe they represent interaction between things. A concrete example is given on the topic of sentiment analysis through masking. Given a sentence, certain language models can successfully predict the sentiment of the sentence. One can then perform masking on the original sentence to obtain a Fourier representation. This representation demonstrates which different words, word-pairs, or word-trios contribute the most to the final sentiment score.

2 A Brief Summary of the Theory

This section reviews the theory of q-SFT.

The major objects of interest are sparse Pseudo-q-ary functions $f : \mathbb{Z}_q^n \to \mathbb{R}$. We must also specify their relationships with their Fourier transforms, and it is represented by the following formula

$$F[k] = \frac{1}{2^n} \sum_{m \in \mathbb{Z}_q^n} f[m] e^{-i2\pi \langle m,k \rangle / q}, \ k \in \mathbb{Z}_q^n. \tag{1}$$

Sparsity S means that F maps the majority $(q^n - S)$ of its finite support of size q^n to 0. This property of F makes it useful because by the Fourier inversion formula as given in 2, one can reconstruct f if the the sparse support of F is learned, which is a much easier task than learning the function f itself.

$$f[m] = \sum_{k \in \mathbb{Z}_q^n} F[k](-1)^{\langle m,k \rangle}, \ m \in \mathbb{Z}_q^n. \tag{2}$$

On a high level, F acts as a valid basis by which one can represent f. Q-SFT is a fast algorithm that learns F.

2.1 Notations

Notations are crucial in both section 2 and 4. In the prior, they're needed to understand the algorithm. In the latter, they represent the parameters that we're selecting. This is why the notation section is introduced in such an early place. In the case when a notation is not yet defined, the place where it is defined will be specified.

- n, q jointly define the input signal size.
- $f : \mathbb{Z}_q^n \to \mathbb{R}$ is the pseudo-q-ary function.
- $F : \mathbb{Z}_q^n \to \mathbb{R}$ is the Fourier transform of f.
- S is the sparsity of f, or the number of $k \in \mathbb{Z}_b^n$ such that $F[k] \neq 0$.
- b defines the number of bins. q^b is the total number of bins, introduced in section 2.2.
- M or $M_c \in \mathbb{Z}_q^{b \times n}$ is the aliasing matrix. $v_k = M_c k \in \mathbb{Z}_q^b$ is the bin that k belongs to for $k \in \mathbb{Z}_q^n$. Introduced in section 2.2.
- C is the refereed to as the number of sub-samples in section 4. It is the number of times aliasing is done, or the number of M matrices. Aliasing is done multiple times to ensure that multi-tons are reduced to singletons during the peeling phase.
- P is the number of times sub-sampling is done for a given bin $j \in \mathbb{Z}_q^n$. It is referred to as num-repeat in section 4. As P increasing, the dimension of the linear equations that solve for k is reduced and it is easier for k to be recovered.
- D or $D_c \in \mathbb{Z}_q^{P \times n}$ is the delay block. Its rows are the delay vectors we use during the sub-sampling phase. The delay blocks are introduced in section 2.2.
- $U_{c,p}[j]$ denotes the samples we obtain for bin $j \in \mathbb{Z}_q^b$. This is done in total $C \times P$ times, as mentioned in section 2.2.

2.2 Algorithm Details in Noiseless Setting

Q-SFT tries to recover the S pairs of $(k, F[k])$ where $F[k]$ is non-zero. On the very high level, the algorithm involves three parts: sub-sampling and aliasing, bin detection, and peeling. We proceed to describe them one by one.

In sub-sampling and aliasing, we view the input support, $\mathbb{Z}_q^n$, as q^n different elements. We want to put these elements into a much lesser amount of bins, the size of which is q^b. This is achieved by the sub-sampling matrix $M \in \mathbb{Z}_q^{b \times n}$. Moreover, we want to repeat the process C different times, so we need to come up with C different matrices, denotes as $M_c,\ c \in \{1, ..., C\}$. The hope is that each of these M_cs would divide the elements equally among the bins with respect to a prior distribution. This means: suppose k comes from a discrete probability distribution supported on $\mathbb{Z}_b^n$, we sample a large number of ks and map each to a corresponding bin by computing $M_c k$ for any c, then the number of elements in each bin should be equal.

We can take certain assumptions on the prior distribution. For instance, the easiest assumption would be a uniform distribution among the support. Another would be one that favors low degree inputs, or inputs that have a low number of ones, for instance a geometric prior for the degree on a uniform distribution among all input of that degree. [1] takes on the uniform assumption. The matrix being constructed is an identity joined by zeroes. Notice that $n > b$. M_c is constructed to be first adding c columns of zeros, then add in an identity matrix of size $b \times b$, then add in $n - c - b$ columns on zeros. In generally, the construction of matrix M_c is one of the more important parts of the algorithm. This process is called Aliasing, which we study more closely in the next section (3.1).

The one-to-one correspondence between a fixed $k \in \mathbb{Z}_q^n$ and its Fourier coefficient $F[k]$ allows us to view the bins as not containing the ks but the $F[k]$s. Hypothetically, if a bin contains only one k for which the $F[k]$ is non-zero, then we can recover the pair $(k, F[k])$. In reality, the recovery process described above is non-trivial, and it relies on the what's called the delay blocks: let P be the number of delay blocks, and let $D_c \in \mathbb{Z}_q^{P \times n}$ be the delay block matrix, where $d_{c,p}$ is the pth column if D_c for $p \in \{1, ..., P\}$. We have

$$U_{c,p}[j] = \frac{1}{q^b} \sum_{l \in \mathbb{Z}_q^b} f[M_c^\top l + d_{c,p}] e^{i2\pi \langle j,l \rangle / q}. \tag{3}$$

$$U_{c,p}[j] = \sum_{k: M_c k = j} F[k] e^{-i2\pi \langle d_{c,p}, k \rangle / q}. \tag{4}$$

[1] has demonstrated that equation 3 implies equation 4. The prior is needed during the sub-sampling part: for any $j \in \mathbb{Z}_q^b$, $U_{c,p}[j]$ can be calculated through 3, by first multiplying the transpose of the aliasing matrix and then adding in a corresponding delay vector. Because M_c and D_c are designed beforehand, the sub-samples one take from the f function is deterministic. The latter interprets $U_{c,p}[j]$ as a complex linear combination of the Fourier coefficients of elements in a given bin. Intuitively, the number of non-zero $F[k]$s in each bin plays an important role.

In bin detection, we formalize the above intuition. Bins are the sets $\{U_{c,p}[j] : p \in \{1, ..., P\}\}$. In other words, bins are still indexed by c and j. We sub-sample C times in total, and each time we split q^n elements into q^b bins. A bin that does not contain any non-zero $F[k]$ is called a zero-ton, or $\mathcal{H}_Z$. $U_{c,p}[j]$ is 0 for any c, p, j if and only if the case of a zero-ton. A bin that contains only one non-zero $F[k]$ is called a singleton, which we will denote as $\mathcal{H}_s$. Otherwise, a bin is called a multi-ton, or $\mathcal{H}_M$. We've already specified how a zero-ton can be detected. Now, suppose we have a singleton where the non-zero $F[k]$ corresponds to k, and suppose $d_{c,1}$ is the zero vector. This would mean that its inner product with k is 0, so $U_{c,1}[j] = F[k]$. Consider the ratio between $U_{c,p}[j]$ and $U_{c,1}[j]$ for any p. According to equation 4, we must have

$$\left|\frac{U_{c,p}[j]}{U_{c,1}[j]}\right| = |e^{-i2\pi \langle d_{c,p}, k \rangle / q}| = 1. \tag{5}$$

Equation 5 is true because the expression in the middle is a rotation of the unit vector on the unit circle, which must have length 1. Thus, if equation 5 holds for all $p \neq 1$, we declare the corresponding bin as a singleton. Otherwise, the bin is declared to be a multi-ton.

In peeling, we focus on the singletons. Since we've already constructed $d_{c,1}$ to be all zeros, we can recover $F[k]$ for the singletons. It is the k that we try to find now, and this is done through solving a set of linear equations. Specifically, equation 5 establishes the relationship between the ratio and the inner product of $d_{c,p}$ and k. If D_c is chosen to be a column of zeros and the identity matrix, we would automatically recover k. As we decrease the number of rows of D, in other words P, the problem becomes increasingly more difficult. The peeling phase begins with recovering all the $(k, F[k])$ pairs for the detected singletons. Recall that there are C different M matrices. Fix a non-zero $F[k]$, we have placed it a bin (can be the same or different) using M_c for C different times. If some of these bins are singletons, then we can successfully recover the pair $(k, F[k])$. Now, for the multi-tons among these bins, we take $F[k]$ out in hope of getting a new singleton. The repeating of the above process until full recovery or failure is the entirety of the peeling phase.

2.3 A separate discussion for the noisy setting

A separate discussion for the noisy setting is also necessary since it is the primary object of study in section 4.2. Suppose that when taking samples of function f, a zero-mean Gaussian with variance σ^2 is incurred. This would primarily affect bin detection as well as peeling. If we replace $f[M_c^\top l + d_{c,p}]$ in equation 3 with $f[M_c^\top l + d_{c,p}] + \epsilon$ for $\epsilon \sim N(0, \sigma^2)$, $U_{c,p}[j]$ would be non-zero even for what would have been a zero-ton. Similarly, the ratio established in equation 5 would never be exactly 1, but we want to declare singletons for ratios that are approximately 1. This is done by defining intervals around the ratios mentioned above.

An alternative scheme is developed and can be described as the following: suppose σ is given, and suppose the ratios in equation 5 are calculated to be $r_2, ..., r_P$. Declare a bin to be a zero-ton if $r_i \in (0, g(\epsilon)]$ for all $i \in \{2, ..., P\}$ for some monotonely increasing function g. If the bin is not a zero-ton, declare it to be a singleton if $r_i \in [1 - f(\epsilon), 1 + f(\epsilon)]$ for all i. Otherwise, declare it as a multi-ton. A crucial problem arises since σ is often not given for a real world application. The search for ϵ used during the peeling phase is necessary, and it is the primary object of study in section 4.2.

Because of how the cutoffs are calculated, an incorrectly selected peeling noise can lead to failure in peeling. Suppose that we have, for a fixed p and c, a zero-ton j. This means that in the noiseless setting, $U_{c,p}[j]$ would equal zero. Recall that in calculating this value, we iterate over the bins to be inversely mapped by the Aliasing matrix, and then add in a delay vector to obtain a signal that lies in the original support of f. The fact that we have a zero-ton would mean each Fourier coefficient corresponding to a resulting vector from the previous step is zero. There are q^b of them in total. $U_{c,p}[j]$ equals to a linear combination of zeros in the noiseless setting, but in the noisy setting it would equal to a linear combination of small Gaussian noises, which wouldn't be exactly zero. Instead, its distance to the origin would be a random variable that depends on the actual noise. If the peeling noise is zero or significantly smaller than the true noise, then the region, defined by the cutoff, in which we declare zero-tons captures true zero-tons with small probability. This means that we'll declare almost no singleton. On the other hand, if the peeling noise is much larger than the actual noise, following a similar argument, we will declare everything as a zero-ton.

It is also crucial to list all the outputs of the algorithm: apart from the peeled coefficients and their locations, which are the $(k, \hat{F}(k))$ pairs, there are several other numbers being provided. Some are related to runtime: the runtime and the number of samples taken. Others are related to the degree properties of the input signals: the max and average hamming weight of the ks with non-zero coefficients. Finally, we also know the count of multi-tons and singletons during each peeling iteration.

Now we must discuss a more applied aspect of the algorithm: in section 4, when dealing with the problem of parameter selection, it is important to be able to identify when the algorithm is failing. One of the most direct indicator is the Normalized Mean Squared Error, or NMSE, which is the normalized Euclidean distance between the true Fourier coefficients of the signal and the coefficients obtained by q-SFT.

The NMSE can be calculated using the following formula:

$$\text{NMSE} = \frac{\sum_{m \in \mathbb{Z}_q^n} (f(m) - \hat{f}(m))^2}{\sum_{m \in \mathbb{Z}_q^n} f(m)^2}. \tag{6}$$

where $\hat{f}$ is the estimated function based on the set of coefficients recovered. Define P to be the set of non-zero coefficient pairs and P' to be the set of recovered pairs. This means $|P| = S$ and $|P'| \leq S$. We have

$$\hat{f}[m] = \sum_{k \in P'} \hat{F}[k] \omega^{\langle m,k \rangle}. \tag{7}$$

The quantity in equation 6 cannot be directly calculation since it requires full knowledge of f. However, it is possible to be estimated by sampling from the support of f. Let $X = \{X_1, ..., X_n\}$ be sampled independently and uniformly from $\mathbb{Z}_q^n$, we have

$$\hat{\text{NMSE}}_n = \frac{\sum_{m \in X} (f(m) - \hat{f}(m))^2}{\sum_{m \in X} f(m)^2}. \tag{8}$$

3 Collected Proofs

This section consists of proofs I've attempted on that would complement the original paper. It's still under development.

3.1 Aliasing Pattern

The objective of aliasing can be described independently from the algorithm as a whole. For simplicity purposes, consider $q = 2$. Additions are done in module 2. Given that input signals are sampled from a prior distribution μ supported on $\mathbb{Z}_2^n$, we want the construct M so the output bins are uniformly chosen with equal probability. Let's first take the example of sampling a single k from μ and then aliasing with a random M matrix of identical and independent Bernoulli 0.5. We proceed to show that in this case, k is mapped to every bin with equal probability.

Theorem 1 *Let $v \in \mathbb{Z}_2^b$, $M \in \{0,1\}^{(b\times n)}$ where $M_{i,j} \sim$ Bernoulli$(\frac{1}{2})$ be a random matrix, and k be sampled from any prior distribution μ that gives the all-zeros vector zero weight. Then, $P(Mk = v) = (\frac{1}{2})^b$.*

Proof:

$$P(Mk = v) = \sum_{k' \in \mathbb{Z}_2^n} P(Mk = v | k = k')\mu(k') = \sum_{k' \in \mathbb{Z}_2^n} P(Mk' = v)\mu(k'). \tag{9}$$

In equation 9, by conditioning on k', we reduce the randomness in each probability from M and k being both random to only M being random. Further more, knowing that each row of M is independent from each other and let v_i be the i-th entry of v, M_i be the i-th row of M, we have

$$\sum_{k' \in \mathbb{Z}_2^n} P(Mk' = v)\mu(k') = \sum_{k' \in \mathbb{Z}_2^n} \prod_{i=1}^{b} P(M_i k' = v_i)\mu(k'). \tag{10}$$

Observe that $P(M_i k' = v_i) = \frac{1}{2}$ as long as k' is not the zero vector. Let m be the number of non-zero entries in k'. Since the positions of the M_i random row vector corresponding to these m non-zero elements in k' are i.i.d. Bernoulli's, the probability that their sum equals an even number or an odd number must be the same. Since one of these two events has to happen, each happens with probability $\frac{1}{2}$. Thus, we have

$$P(Mk = v) = (\frac{1}{2})^b \sum_{k' \in \mathbb{Z}_2^n} \mu(k') = (\frac{1}{2})^b. \tag{11}$$

Since the expression in equation 11 holds regardless of v, our claim holds true.

3.2 Aliasing with Geometric Prior

We can certainly do better than casting M as a random matrix filled with 0s and 1s. In most real world settings, input signals are of mostly low degrees. This allows us to make assumptions on μ: suppose k' is sampled from a geometric p distribution. For now, we can even take $p = \frac{1}{2}$. An input signal k is then sampled from all permutations of signals with degree k'.

In this setting,

3.3 Attempt to Bound Error Estimation

NMSE and $\hat{\text{NMSE}}_t$ are introduced in section 2.3. This subsection attempts to understand how fast the latter converges to the prior as a function of t. We proceed by first decomposing the numerator and denominator of NMSE, then bounding each individually. Let the notations in section 2.3 prevail.

Theorem 2 *Let $g, h : (\mathbb{Z}_q^n)^t \to \mathbb{R}$ be defined as $g(x_1, ..., x_t) = \sum_{x_i}(f(x_i) - \hat{f}(x_i))^2$ and $h(x_1, ..., x_t) = 1/(\sum_{x_i} f(x_i)^2)$, then $\hat{\text{NMSE}}_t = g(X_1, ..., X_t)h(X_1, ..., X_t)$ and NMSE $= E[g(X_1, ..., X_t)]E[h(X_1, ..., X_t)]$.*

Proof: The first result follows from definition. To show the second result, we have

$$E[h(X_1,...,X_t)] = \frac{1}{\sum_{x_i} E[f(x_i)^2]} = \frac{q^n}{t\sum_{m\in\mathbb{Z}_q^n} f(m)^2}. \tag{12}$$

and

$$E[g(X_1,...,X_t)] = \sum_{x_i} E[(f_{x_i} - \hat{f}(x_i))^2] = \frac{t}{q^n} \sum_{m\in\mathbb{Z}_q^n} (f_m - \hat{f}(m))^2. \tag{13}$$

Taking the product of the above two expressions would yield the desired result.

The next claim is we can bound gh by bounding g and h individually.

Theorem 3 *Consider $P(|N\hat{MSE}_t - NMSE| \geq u)$. This can be calculated by properly selecting a and b and bounding $P(|g - E[g]| \geq a)$ and $P(|h - E[h]| \geq b)$.*

Proof: Suppose events $\{|g - E[g]| \leq a\}$ and $\{|h - E[h]| \leq b\}$ hold. This implies

$$gh - E[g]E[h] \leq (a + E[g])(b + E[h]) - E[g]E[h] = ab + bE[g] + aE[h]. \tag{14}$$

and

$$gh - E[g]E[h] \geq (E[g] - a)(E[h] - b) - E[g]E[h] = ab - bE[g] - aE[h]. \tag{15}$$

so

$$|gh - E[g]E[h]| \leq ab + bE[g] + aE[h]. \tag{16}$$

If a, b are chosen so that $ab + bE[g] + aE[h] \leq u$, then $\{|gh - E[g]E[h]| \geq u\}$ implies either $\{|g - E[g]| \geq a\}$ or $\{|h - E[h]| \geq b\}$, so by union bound

$$P(|\text{N}\hat{\text{M}}\text{SE}_t - \text{NMSE}| \geq u) = P(|gh - E[g]E[h]| \geq u) \leq P(|g - E[g]| \geq a) + P(|h - E[h]| \geq b). \tag{17}$$

Notice that the choice of a and b do depend on $E[g]$ and $E[h]$. This problem, however, can be overcome with certain assumptions on the magnitude of g and h.

4 Major Results in Parameter Selection

This section consists of the results I obtained after running the algorithm on different setting with different parameters in hope of recognizing some pattern. It also contains an algorithm I developed to search for peeling noise.

4.1 Parameter Selection

When applying q-SFT, setting the parameters of the algorithm greatly affects its performance. These parameters include num-repeat P, or the number of rows of the decoder block, num-subsample C, or the number of times aliasing is done, b, or the constant that relates to the number of bins, and peeling noise. While setting them too high results in drastically increasing the run time, setting them too low might result in failure of peeling. For example, taking the delay block to be a square identity guarantees the recovery of k but takes too long to run, and taking P to be too small results in failure to solve k. Similarly, if C is too small, then we might run out of singletons to peel before full recovery.

The pipeline of parameter selection can be described as the following:

- 1, Get sparsity S and an initial noise estimate.
- 2, Choose b, C, and P based on S, n, q, and the initial noise estimate.
- 3, Sample the function at specified coordinates.
- 4, Run q-SFT peeling with the chosen noise and observe the mode of failure while peeling.
- 5, Update the chosen noise, run q-SFT peeling again, and observe the mode of failure again.
- 6, Repeat until we approach the true noise.

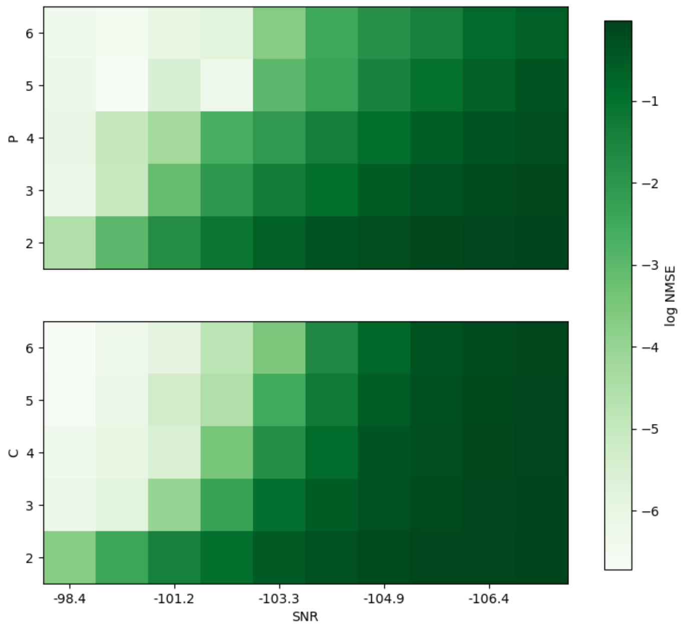
The iterative algorithm in steps 3, 4, 5, and 6 is described in details in the next section 4.2. This section studies the problem in step 2: that is, given that we know n, q, and S, and given that our noise estimation is somewhat accurate, how we should choose b, C, and P accordingly.

First, it is worth noticing that if all other settings remain the same, increasing b has a greater impact than increasing P, and increasing P has a greater impact than increasing C. These two results are illustrated in figures 1, 2.

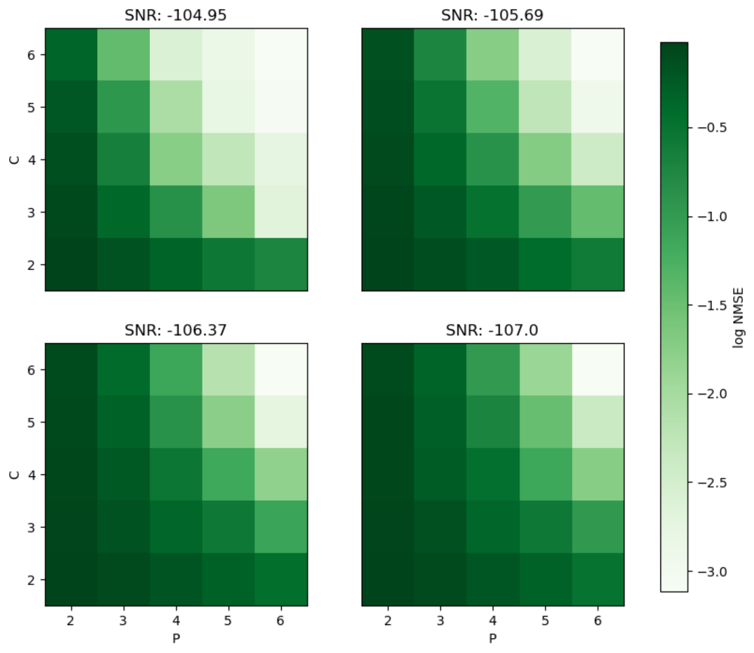
In figure 1, the impact of increasing P versus C on the resulting NMSE is studied on a wide range of peeling noises. Figure 1a shows that increasing P by 1 has almost the same impact as increasing C by 1. The interval of noises in which changing P can have an impact on the performance of the algorithm is comparable to that of C. Figure 1b takes a further step and plots P versus C for four different noises. From these four plots we conclude that P causes slightly more impact C on reducing the noise. Consider, for example, the first column of all of these four plots: increasing C by an arbitrary amount has almost no impact on the NMSE. Consider, now, the last row of each plot: the NMSE is slowly decreasing and the change of color from dark to light is more obvious than for the columns.

In figure 2, a similar experiment is conducted except on P versus b. Contrary to before, b is shown to have a significantly higher impact on the performance of the algorithm than P. As shown in figure 2a, the noise interval on which changing P can improve the performance is much shorter than that of b. This interval is around three blocks wide for P: for any noise less than that represented by the first column, the algorithm works with all Ps, and for any noise more than that represented by the forth column, the algorithm fails with all Ps. However, this interval is more than ten blocks for b. In figure 2b, increasing P almost never makes the NMSE decrease from near 1 to near 0, but increasing b always makes the NMSE decrease from near 1 to near 0.

During parameter selection, we want to properly select the most impactful parameter first. When choosing b, we want to ensure that $q^b > S$. However, if this condition is satisfied, our decision of b should depend on only the guessed noise but not q, n, or S. This fact is shown in figure 3: on a very wide range of sparsities, the NMSE is the same for a given b. Thus, we should decide b based on the estimated peeling noise only. A good strategy would be to initialize b as the minimal b possible. Then, as we increase the peeling noise, we increase b by 1 each time we increase log noise by 1. Lastly, for P and C, since their impact on the performance is much less than b, we initialize them to be a reasonable value, for example 4 for both P and C, and leave them unchanged until we've found a reasonable b and peeling noise. If we want to decrease them to optimize running time or increase them to minimize NMSE, we decrease or increase both by 1 at a time.

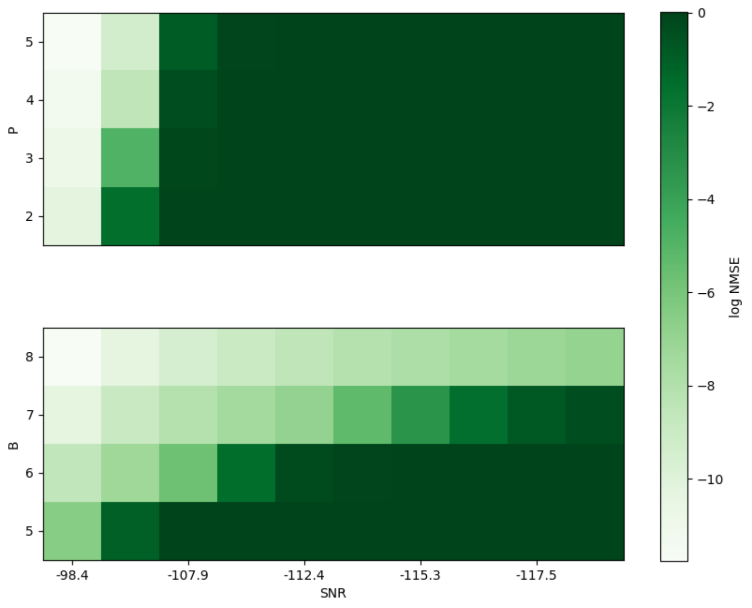


(a) P and C plotted separately against 10 noises. The noises are equally spaced from 10 to 27. The corresponding SNR is from -98 to -107.

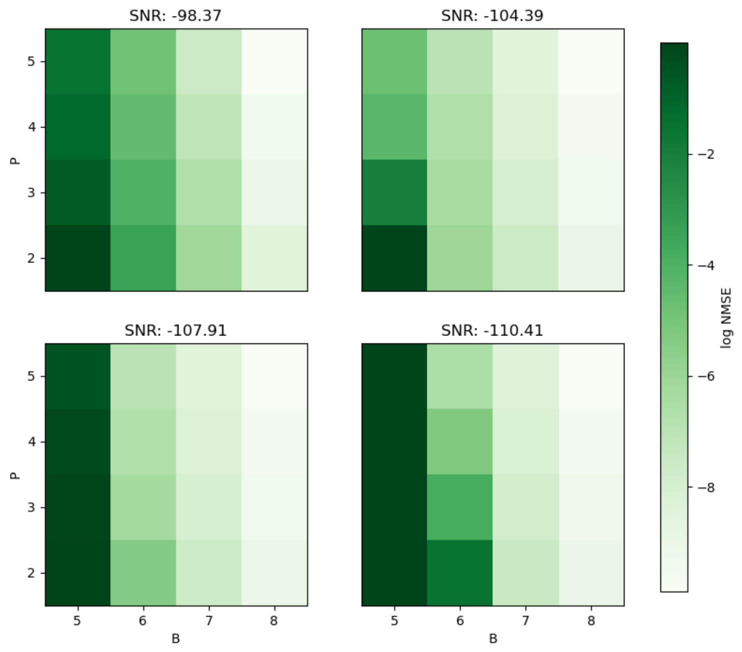


(b) C plotted against P for 4 noises. The chosen SNRs are -104.95, -105.69, -106.37, -107.00.

Figure 1: Both plots use a random signal generated. $n = 18$, $q = 4$, $a = 1$, sparsity$= 1000$, and b is chosen to be 5. The ranges for P and C are both from 2 to 6. NMSE is represented with log scale and base 10.

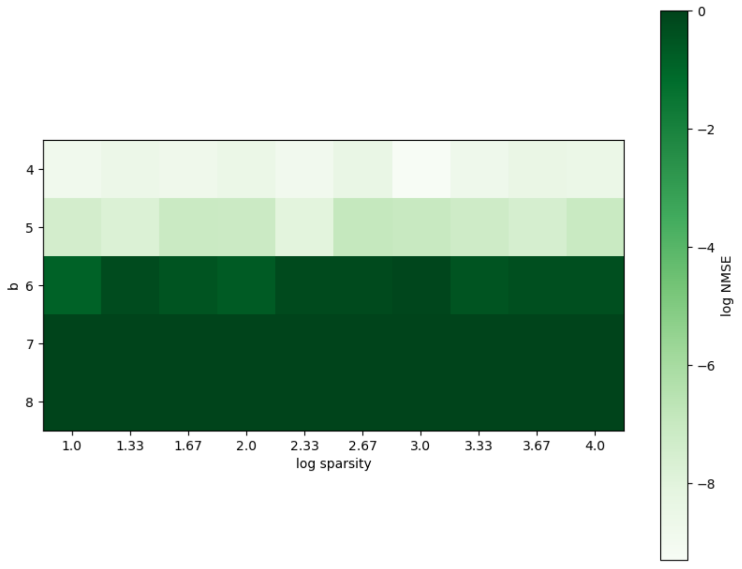


(a) P and b plotted against 10 noises. The noises are equally spaced between 10 and 100. The corresponding SNR is from -98 to -118.



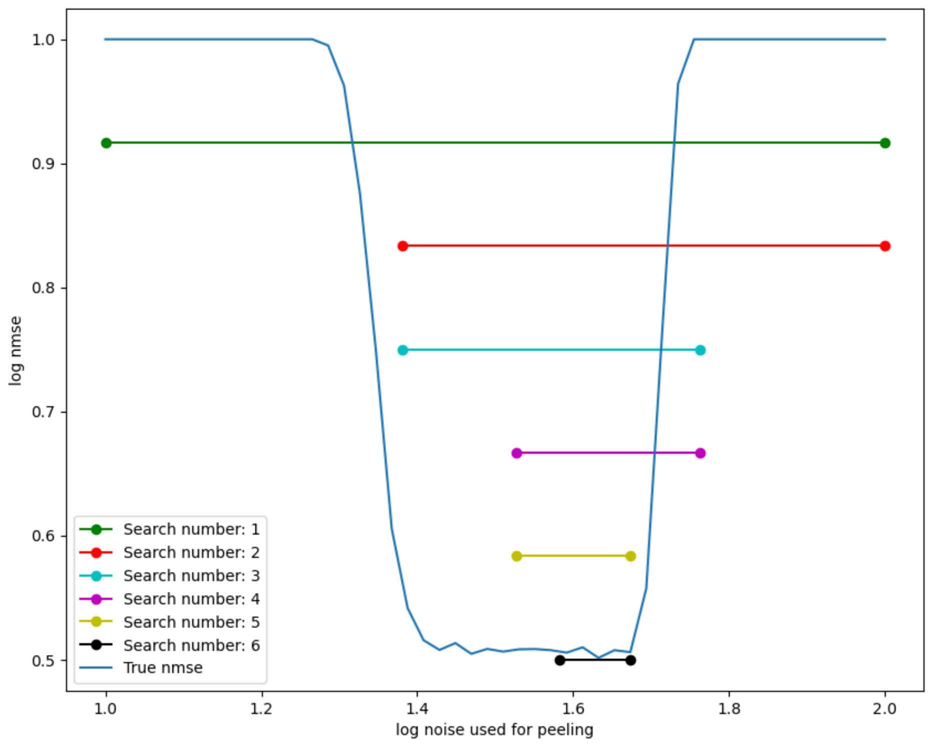
(b) P plotted against b for four different noises. The SNRs are -98.37, -104.39, -107.91, -110.41.

Figure 2: Both plots use a random signal generated. $n = 18$, $q = 4$, $a = 1$, sparsity$= 1000$, and C is chosen to be 5. The ranges for P is again from 2 to 5. The range for b is from 5 to 8. Notice that we must have $q^b >$ sparsity for the algorithm to work, so the minimum b is 5. NMSE is represented with log scale and base 10.



(a) b plotted against log sparsity.

Figure 3: The signal is as following: $n = 18$, $q = 4$, $a = 1$, $P = 5$ and $C = 5$. The range for b is from 4 to 8. Both NMSE and sparsity are represented in log scale with base 10. Noise used for this plot is 50.



(a) Errors plotted against peeling noises

Figure 4: The resulting NMSEs using different peeling noises. Both axes use log scale. The true noise of the signal in this example is 1.4. The searching intervals converge so values slightly larger than the true peeling noise because peeling with a larger noise generate lower errors.

4.2 Choosing Peeling Noise

Since it is often impossible to know the true noise of the signal, the following algorithm has been developed to search for an appropriate noise for peeling.

Independent of other parameters, as our peeling noise diverges from the true noise, the peeling NMSE increases exponentially. As shown in figure 4, within a range of peeling noises the algorithm performs well, but outside of this range the resulting NMSE increases to 1 immediately.

The search for the noise is divided into two phases. First, we search for an interval that captures the true noise, or the noise region in figure 4 that has low NMSE values when used for peeling. It is worth noticing that the endpoints of the region must have peeling NMSE of 1. To do so, we use the following property: if the peeling noise is significantly greater than the true noise, then the number of multi-ton declared by the peeling phase is zero. Otherwise it is nonzero. This is because the peeling noise determines the cutoff in bin detection, and a high cutoff implies that everything gets declared as a zero-ton. To search for the interval, we initialize noise value, l, and a step size for searching, s. Define the error and multi-ton count when peeling with noise l to be $e(l), mc(l)$. The search for the interval (a, b) is then done using the following algorithm 1.

Once this part of the algorithm is done and we've successfully found both ends of the interval, we then use golden search to find the true noise. We define a threshold to determine when to terminate the algorithm. Moreover, the gold ratio is: $gr = \frac{\sqrt{5}+1}{2}$. In each recursive call, we begin with left and right ends of the interval, knowing that the true noise is in between. Two middle points, m_1, m_2, are calculated using the goldren ratio. Then, we run test to obtain multi-ton count and NMSE when peeling with noises left, m_1, m_2, and right. Again, we use $mc(.)$ to denote the multi-ton count when peeling at a corresponding noise, $e(.)$ to denote the corresponding NMSE. Lastly, by comparing them, we decide the next left and right end points to search on. Define t to be the threshold of termination, where if the length of the interval lies below t, then algorithm is terminated. The detailed logic is described in 2.

Overall, as one can see in figure 4, the searches converge to the true noise, or at least the region around the true

Algorithm 1 Find interval

```
Input: $l, s$
Done $\leftarrow$ False, $a =$ None, $b =$ None
while not Done do
    Compute $e(l), mc(l)$
    if $e(l) = 1$ then
        if $mc(l) = 0$ then
            $b \leftarrow l$
            if $a$ is not None then
                Done $\leftarrow$ True
            else
                $l \leftarrow l - s$
            end if
        else
            $a \leftarrow l$
            if $b$ is not None then
                Done $\leftarrow$ True
            else
                $l \leftarrow l + s$
            end if
        end if
    else
        if $a$ is not None then
            $l \leftarrow l + s$
        else
            $l \leftarrow l - s$
        end if
    end if
end while
Output: the interval $(a, b)$
```

Algorithm 2 Golden search

```
Input: the interval $(a, b), t$
left $\leftarrow a$, right $\leftarrow b$
while right - left $< t$ do
    $m_1 \leftarrow$ right $-$ (right $-$ left)$/gr$
    $m_2 \leftarrow$ left $+$ (right $-$ left)$/gr$
    Compute $mc$(left), $mc(m_1)$, $mc(m_2)$, $mc$(right), $e$(left), $e(m_1)$, $e(m_2)$, $e$(right)
    if $e$(left) $= e(m_1) = e(m_2) = e$(right) $= 0$ then
        if $mc(m_1) = mc(m_2) = 0$ then
            right $\leftarrow m_1$
        else if $mc(m_1) \neq 0$ and $mc(m_2) \neq 0$ then
            left $\leftarrow m_2$
        else
            left $\leftarrow m_1$, right $\leftarrow m_2$
        end if
    else
        if $e(m_1) < e(m_2)$ then
            right $\leftarrow m_2$
        else
            left $\leftarrow m_1$
        end if
    end if
end while
Output: estimated peeling noise $\frac{\text{left}+\text{right}}{2}$
```

noise where the peeling error is low.

References

[1] Yigit Efe Erginbas et al. "Efficiently Computing Sparse Fourier Transforms of q-ary Functions". In: (2023). arXiv: 2301.06200 [eess.SP].