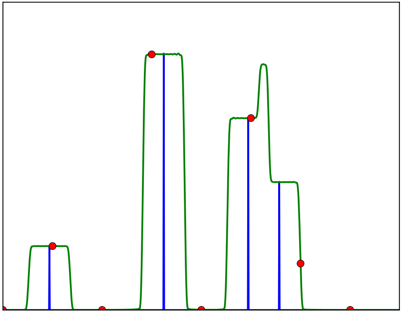
Sparse Fourier Transform Algorithms

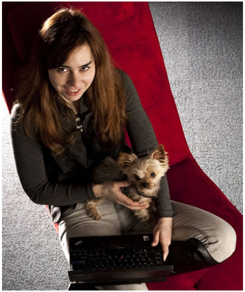
Eric Price

MIT

2013-2-17



Collaboration with Rest of Session Speakers



Outline

1. Introduction

Outline

1. Introduction

2. Algorithm (exactly sparse case)

Outline

Outline

1. Introduction

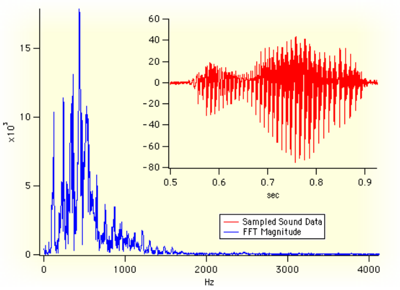
2. Algorithm (exactly sparse case)

3. Algorithm (noisy case)

Discrete Fourier Transform (DFT)

- Given $x \in \mathbb{C}^n$, compute Fourier transform $\widehat{x}$:

$$\widehat{x}_i = \sum_j \omega^{ij} x_j \quad \text{for} \quad \omega = e^{2\pi \mathbf{i}/n}$$

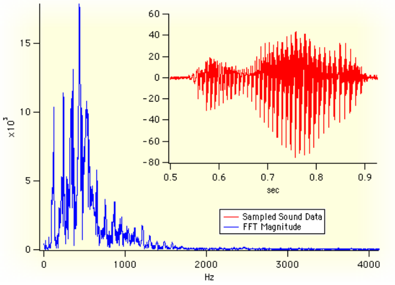


Discrete Fourier Transform (DFT)

- Given $x \in \mathbb{C}^n$, compute Fourier transform $\widehat{x}$:

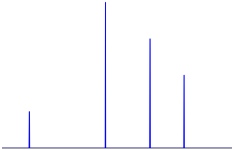
$$\widehat{x}_i = \sum_j \omega^{ij} x_j \quad \text{for} \quad \omega = e^{2\pi\mathbf{i}/n}$$

$$\widehat{x} = Fx \quad \text{for} \quad F_{ij} = \omega^{ij}$$



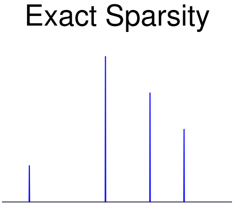
Sparse Fourier Transform

Exact Sparsity



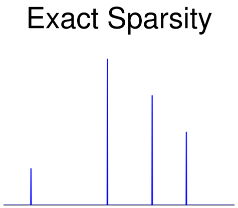
- Suppose $\widehat{x}$ is k-sparse: only k non-zero coefficients.

Sparse Fourier Transform



- Suppose $\widehat{x}$ is k-sparse: only k non-zero coefficients.
- How fast can we hope to compute it?

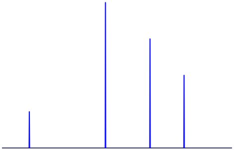
Sparse Fourier Transform



- Suppose $\widehat{x}$ is k-sparse: only k non-zero coefficients.
- How fast can we hope to compute it?
  - $\Omega(k \log k)$: dense "lower bound."

Sparse Fourier Transform

Exact Sparsity

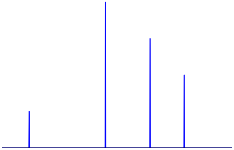


Our Result: $O(k \log n)$

- Suppose $\widehat{x}$ is k-sparse: only k non-zero coefficients.
- How fast can we hope to compute it?
  - $\Omega(k \log k)$: dense "lower bound."

Sparse Fourier Transform

Exact Sparsity

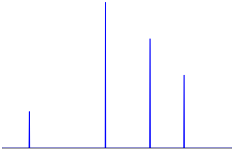


Our Result: $O(k \log n)$

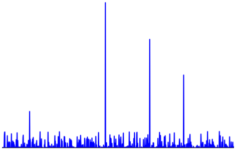
- Suppose $\widehat{x}$ is k-sparse: only k non-zero coefficients.
- How fast can we hope to compute it?
  - $\Omega(k \log k)$: dense "lower bound."
- More generally, $\widehat{x}$ is "close" to k-sparse vector.

Sparse Fourier Transform

Exact Sparsity



Approximate Sparsity

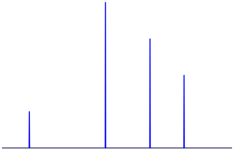


Our Result: $O(k \log n)$

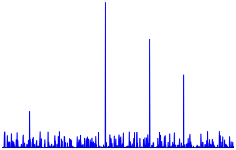
- Suppose $\widehat{x}$ is k-sparse: only k non-zero coefficients.
- How fast can we hope to compute it?
  - $\Omega(k \log k)$: dense "lower bound."
- More generally, $\widehat{x}$ is "close" to k-sparse vector.
  - Sublinear time can't possibly output $\widehat{x}$ exactly.

Sparse Fourier Transform

Exact Sparsity



Approximate Sparsity

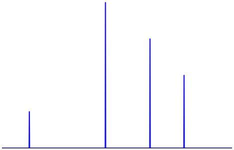


Our Result: $O(k \log n)$

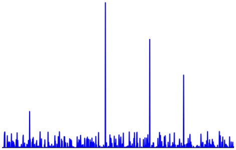
- Suppose $\widehat{x}$ is k-sparse: only k non-zero coefficients.
- How fast can we hope to compute it?
  - $\Omega(k \log k)$: dense "lower bound."
- More generally, $\widehat{x}$ is "close" to k-sparse vector.
  - Sublinear time can't possibly output $\widehat{x}$ exactly.
  - Best k-sparse approximation has error E.

Sparse Fourier Transform

Exact Sparsity



Approximate Sparsity

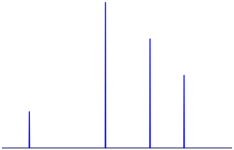


Our Result: $O(k \log n)$

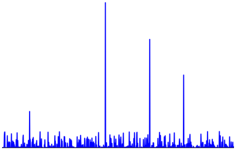
- Suppose $\widehat{x}$ is k-sparse: only k non-zero coefficients.
- How fast can we hope to compute it?
  - $\Omega(k \log k)$: dense “lower bound.”
- More generally, $\widehat{x}$ is “close” to k-sparse vector.
  - Sublinear time can't possibly output $\widehat{x}$ exactly.
  - Best k-sparse approximation has error E.
  - Goal: error $(1 + \epsilon)E$.

Sparse Fourier Transform

Exact Sparsity



Approximate Sparsity

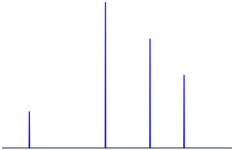


Our Result: $O(k \log n)$ $O(k \log(n/k) \log n)$

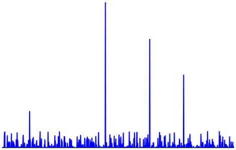
- Suppose $\widehat{x}$ is k-sparse: only k non-zero coefficients.
- How fast can we hope to compute it?
  - $\Omega(k \log k)$: dense "lower bound."
- More generally, $\widehat{x}$ is "close" to k-sparse vector.
  - Sublinear time can't possibly output $\widehat{x}$ exactly.
  - Best k-sparse approximation has error E.
  - Goal: error $(1 + \epsilon)E$.

Sparse Fourier Transform

Exact Sparsity



Approximate Sparsity



Our Result: $O(k \log n)$ $O(k \log(n/k) \log n)$

- Suppose $\widehat{x}$ is k-sparse: only k non-zero coefficients.
- How fast can we hope to compute it?
  - $\Omega(k \log k)$: dense "lower bound."
- More generally, $\widehat{x}$ is "close" to k-sparse vector.
  - Sublinear time can't possibly output $\widehat{x}$ exactly.
  - Best k-sparse approximation has error E.
  - Goal: error $(1 + \epsilon)E$.
- Faster than FFT for any $k \ll n$.

Previous work on Sparse Fourier Transforms

- Boolean cube: [KM92], [GL89]. $\mathbb{C}^n$: [Mansour '92] $k^c \log^c n$.
- Long line of additional work [GGIMS02, AGS03, GMS05, Iwen '10, Akavia '10]
- Mark Iwen's tutorial: $k \log^c n$ time.

Previous work on Sparse Fourier Transforms

- Boolean cube: [KM92], [GL89]. $\mathbb{C}^n$: [Mansour '92] $k^c \log^c n$.
- Long line of additional work [GGIMS02, AGS03, GMS05, Iwen '10, Akavia '10]
- Mark Iwen's tutorial: $k \log^c n$ time.
  - But c matters; faster than FFT if $k/n \ll 1/\log^{c-1} n$.

Previous work on Sparse Fourier Transforms

- Boolean cube: [KM92], [GL89]. $\mathbb{C}^n$: [Mansour '92] $k^c \log^c n$.
- Long line of additional work [GGIMS02, AGS03, GMS05, Iwen '10, Akavia '10]
- Mark Iwen's tutorial: $k \log^c n$ time.
  - But c matters; faster than FFT if $k/n \ll 1/\log^{c-1} n$.
  - Also yesterday: 4D light fields have $k/n \approx 0.1\%$.

Previous work on Sparse Fourier Transforms

- Boolean cube: [KM92], [GL89]. $\mathbb{C}^n$: [Mansour '92] $k^c \log^c n$.
- Long line of additional work [GGIMS02, AGS03, GMS05, Iwen '10, Akavia '10]
- Mark Iwen's tutorial: $k \log^c n$ time.
  - But c matters; faster than FFT if $k/n \ll 1/\log^{c-1} n$.
  - Also yesterday: 4D light fields have $k/n \approx 0.1\%$.
  - If $c = 4$: *slower* than FFT.

Previous work on Sparse Fourier Transforms

- Boolean cube: [KM92], [GL89]. $\mathbb{C}^n$: [Mansour '92] $k^c \log^c n$.
- Long line of additional work [GGIMS02, AGS03, GMS05, Iwen '10, Akavia '10]
- Mark Iwen's tutorial: $k \log^c n$ time.
  - But c matters; faster than FFT if $k/n \ll 1/\log^{c-1} n$.
  - Also yesterday: 4D light fields have $k/n \approx 0.1\%$.
  - If $c = 4$: *slower* than FFT.
  - Our result: faster than FFT for *any* $k/n \ll 1$.

Previous work on Sparse Fourier Transforms

- Boolean cube: [KM92], [GL89]. $\mathbb{C}^n$: [Mansour '92] $k^c \log^c n$.
- Long line of additional work [GGIMS02, AGS03, GMS05, Iwen '10, Akavia '10]
- Mark Iwen's tutorial: $k \log^c n$ time.
  - But c matters; faster than FFT if $k/n \ll 1/\log^{c-1} n$.
  - Also yesterday: 4D light fields have $k/n \approx 0.1\%$.
  - If $c = 4$: *slower* than FFT.
  - Our result: faster than FFT for *any* $k/n \ll 1$.
  - If no hidden constants, would be 100x faster at 0.1%.

Formal results

- k-sparse Fourier transform in $O(k \log n)$ time.

Formal results

- k-sparse Fourier transform in $O(k \log n)$ time.
- Approximate sparse Fourier transform in $O(\frac{1}{\epsilon} k \log(n/k) \log n)$ time:

$$\|\text{result} - \widehat{x}\|_2 \leqslant (1+\epsilon) \min_{k\text{-sparse } \widehat{x}_{(k)}} \|\widehat{x}_{(k)} - \widehat{x}\|_2$$

  for $\epsilon > 0$.

Formal results

- k-sparse Fourier transform in $O(k \log n)$ time.
- Approximate sparse Fourier transform in $O(\frac{1}{\epsilon} k \log(n/k) \log n)$ time:
  $$\|\text{result} - \widehat{x}\|_2 \leqslant (1+\epsilon) \min_{k\text{-sparse } \widehat{x}_{(k)}} \|\widehat{x}_{(k)} - \widehat{x}\|_2$$
  for $\epsilon > 0$.
- Faster than FFT whenever $k/n < C$ for fixed constant C.

Formal results

- k-sparse Fourier transform in $O(k \log n)$ time.
- Approximate sparse Fourier transform in $O(\frac{1}{\epsilon} k \log(n/k) \log n)$ time:

$$\|\text{result} - \widehat{x}\|_2 \leqslant (1+\epsilon) \min_{k\text{-sparse } \widehat{x}_{(k)}} \|\widehat{x}_{(k)} - \widehat{x}\|_2$$

  for $\epsilon > 0$.
- Faster than FFT whenever $k/n < C$ for fixed constant C.
  - Exact case: $C \approx 1\%$.

Formal results

- k-sparse Fourier transform in $O(k \log n)$ time.
- Approximate sparse Fourier transform in $O(\frac{1}{\epsilon} k \log(n/k) \log n)$ time:
$$\|\text{result} - \widehat{x}\|_2 \leqslant (1+\epsilon) \min_{k\text{-sparse } \widehat{x}_{(k)}} \|\widehat{x}_{(k)} - \widehat{x}\|_2$$
for $\epsilon > 0$.
- Faster than FFT whenever $k/n < C$ for fixed constant C.
  - Exact case: $C \approx 1\%$.
- Caveats:
  - Output $\widehat{x}$ has $\log n$ bit precision.
  - n must be a power of 2 (more generally, n must be smooth).
  - Succeed with 90% probability.

Outline

Algorithm

Suppose $\widehat{x}$ is k-sparse.

Theorem

We can compute $\widehat{x}$ in $O(k \log n)$ time with 90% probability.

Algorithm

Suppose $\widehat{x}$ is k-sparse.

Theorem

We can compute $\widehat{x}$ in $O(k \log n)$ time with 90% probability.

Suffices to recover *most* of $\widehat{x}$, so residual is $k/2$ sparse:

Algorithm

Suppose $\widehat{x}$ is k-sparse.

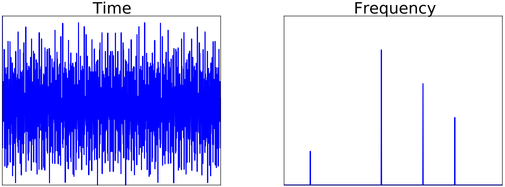
Theorem

We can compute $\widehat{x}$ in $O(k \log n)$ time with 90% probability.

Suffices to recover *most* of $\widehat{x}$, so residual is $k/2$ sparse:

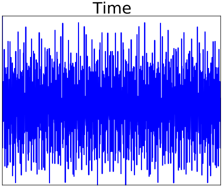
- Then: repeat on residual, with $k \to k/2$ and decreasing the error probability.

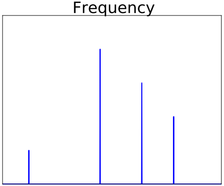
What can you do with Fourier measurements?



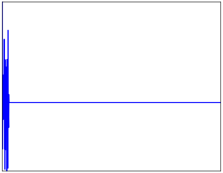
n-dimensional DFT:
$O(n \log n)$
$x \to \widehat{x}$

What can you do with Fourier measurements?

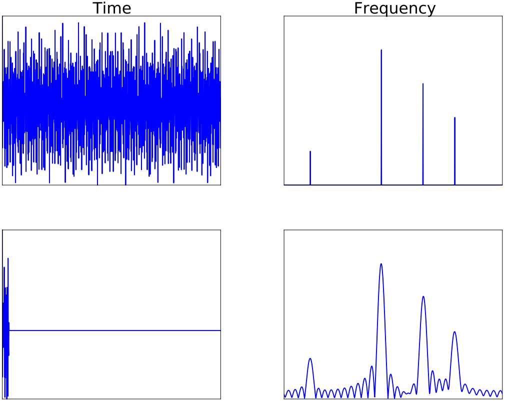




n-dimensional DFT:
$O(n \log n)$
$x \to \widehat{x}$



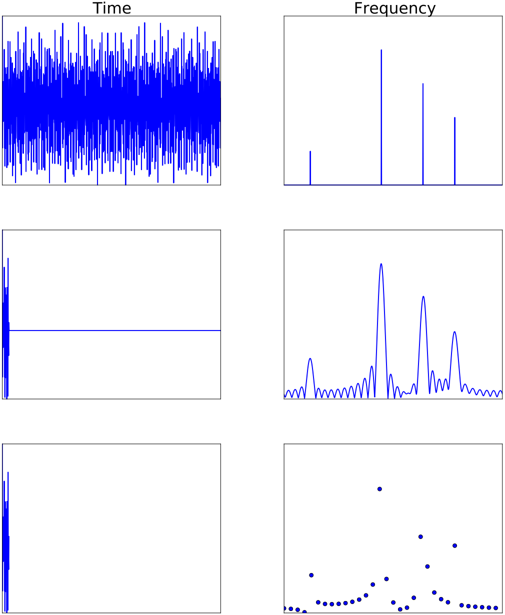
What can you do with Fourier measurements?



n-dimensional DFT:
$O(n \log n)$
$x \to \widehat{x}$

n-dimensional DFT of first B terms: $O(n \log n)$
$x \cdot \text{boxcar} \to \widehat{x} * \text{sinc}$.

What can you do with Fourier measurements?

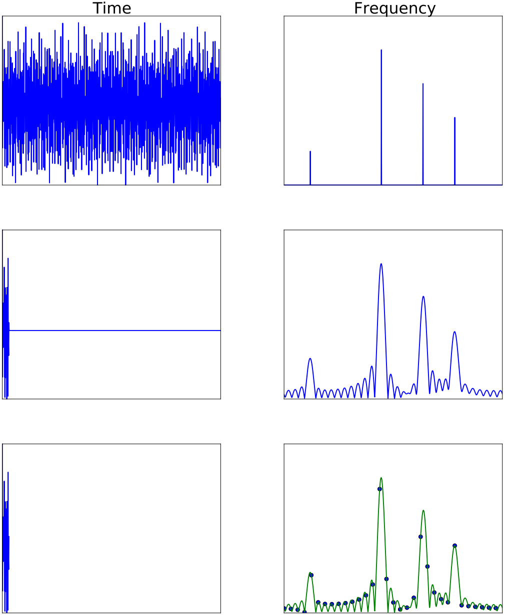


n-dimensional DFT: $O(n \log n)$
$x \to \widehat{x}$

n-dimensional DFT of first B terms: $O(n \log n)$
$x \cdot \text{boxcar} \to \widehat{x} * \text{sinc}$.

B-dimensional DFT of first B terms: $O(B \log B)$
$\text{alias}(x \cdot \text{boxcar}) \to \text{subsample}(\widehat{x} * \text{sinc})$.

What can you do with Fourier measurements?

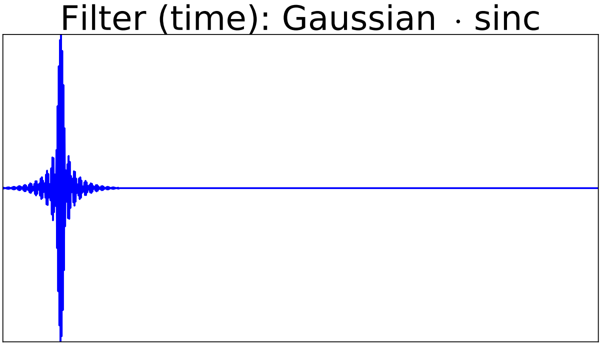


n-dimensional DFT: $O(n \log n)$
$x \to \widehat{x}$

n-dimensional DFT of first B terms: $O(n \log n)$
$x \cdot \text{boxcar} \to \widehat{x} * \text{sinc}$.

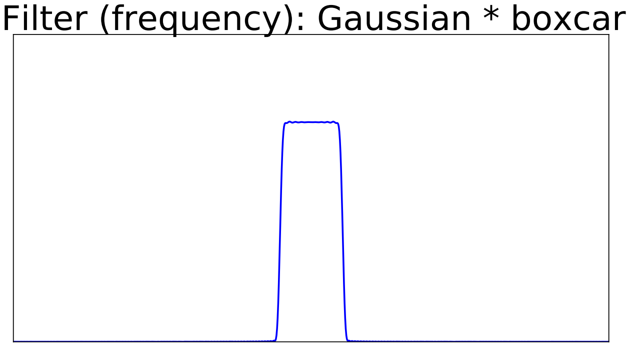
B-dimensional DFT of first B terms: $O(B \log B)$
$\text{alias}(x \cdot \text{boxcar}) \to \text{subsample}(\widehat{x} * \text{sinc})$.

A Better Filter



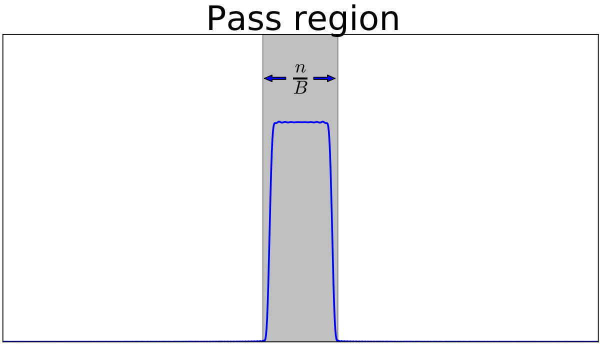
- Time domain is $O(B \log n)$ sparse.

A Better Filter



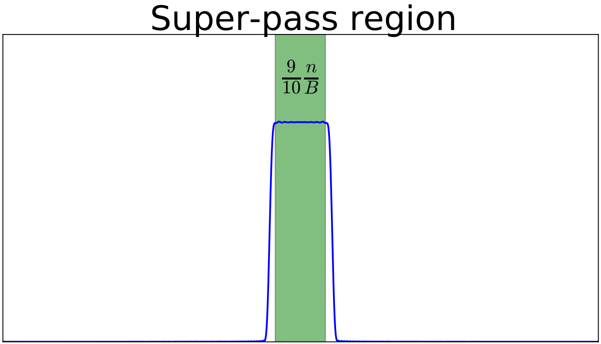
- Time domain is $O(B \log n)$ sparse.

A Better Filter



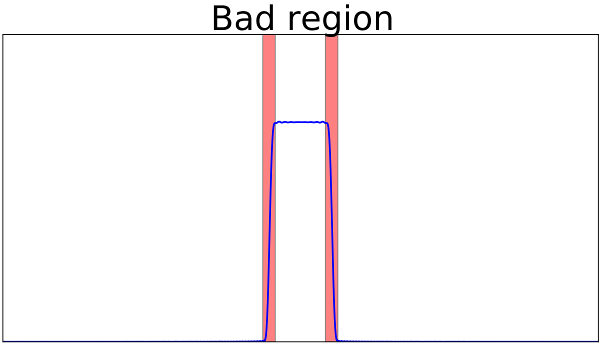
- Time domain is $O(B \log n)$ sparse.
- "Pass region" of size n/B, outside which filter is negligible δ.

A Better Filter



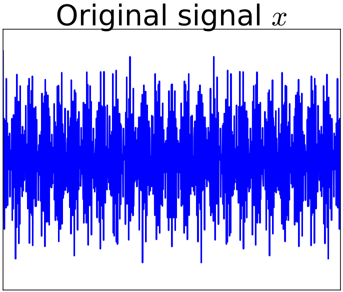
- Time domain is $O(B \log n)$ sparse.
- "Pass region" of size n/B, outside which filter is negligible δ.
- "Super-pass region", where filter ≈ 1.

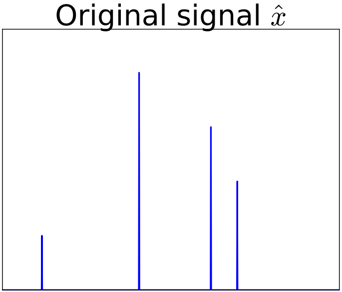
A Better Filter



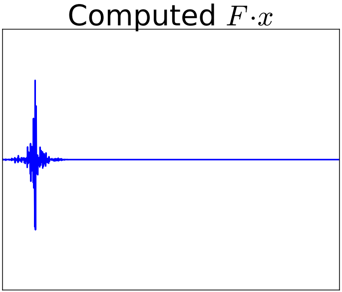
- Time domain is $O(B \log n)$ sparse.
- "Pass region" of size n/B, outside which filter is negligible δ.
- "Super-pass region", where filter ≈ 1.
- Small fraction (say 10%) is "bad region" with intermediate value.

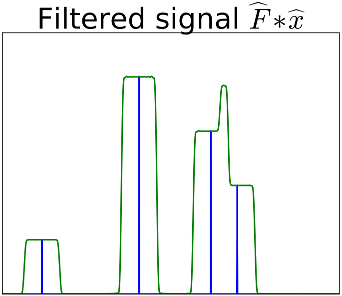
Algorithm for *exactly sparse* signals



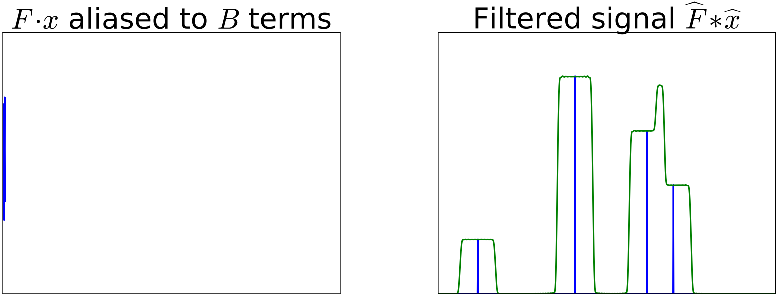


Algorithm for *exactly sparse* signals



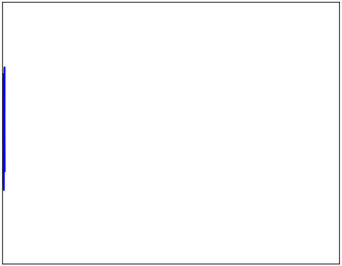


Algorithm for *exactly sparse* signals

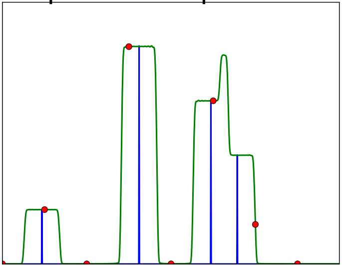


Algorithm for *exactly sparse* signals

$F \cdot x$ aliased to B terms

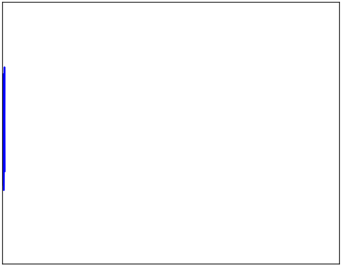


Computed samples of $\widehat{F} * \widehat{x}$

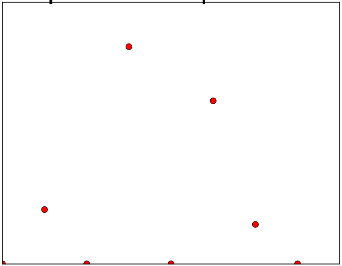


Algorithm for *exactly sparse* signals

$F \cdot x$ aliased to B terms

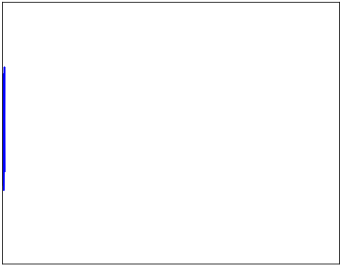


Computed samples of $\widehat{F} * \widehat{x}$

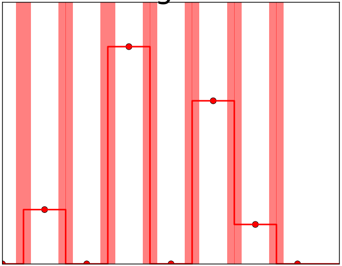


Algorithm for *exactly sparse* signals

$F \cdot x$ aliased to B terms

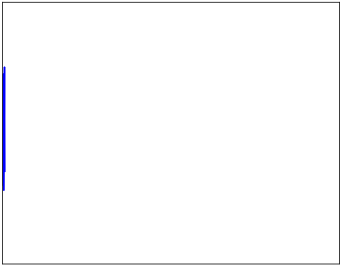


Knowledge about $\hat{x}$

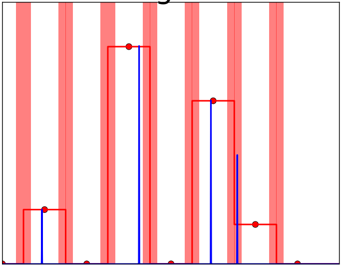


Algorithm for *exactly sparse* signals

$F \cdot x$ aliased to B terms

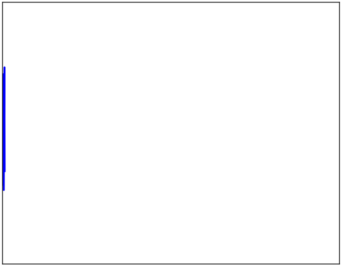


Knowledge about $\hat{x}$

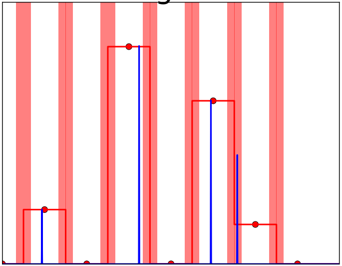


Algorithm for *exactly sparse* signals

$F \cdot x$ aliased to B terms



Knowledge about $\hat{x}$



Lemma

If i is isolated in its bucket and in the "super-pass" region, the value b we compute for its bucket satisfies

$$b = \widehat{x}_i.$$

Computing the b for all B buckets takes $O(B \log n)$ time.

Algorithm for *exactly sparse* signals

Lemma

If i is isolated in its bucket and in the "super-pass" region, the value b we compute for its bucket satisfies

$$b = \widehat{x}_i.$$

Computing the b for all B buckets takes $O(B \log n)$ time.

Algorithm for *exactly sparse* signals

Lemma

If i is isolated in its bucket and in the "super-pass" region, the value b we compute for its bucket satisfies

$$b = \widehat{x}_i.$$

Computing the b for all B buckets takes $O(B \log n)$ time.

- Time-shift x by one and repeat: $b' = \widehat{x}_i \omega^i$.
- Divide to get $b'/b = \omega^i$

Algorithm for *exactly sparse* signals

Lemma

If i is isolated in its bucket and in the "super-pass" region, the value b we compute for its bucket satisfies

$$b = \widehat{x}_i.$$

Computing the b for all B buckets takes $O(B \log n)$ time.

- Time-shift x by one and repeat: $b' = \widehat{x}_i \omega^i$.
- Divide to get $b'/b = \omega^i \implies$ can compute i if it is isolated.

Algorithm for *exactly sparse* signals

Lemma

If i is isolated in its bucket and in the "super-pass" region, the value b we compute for its bucket satisfies

$$b = \widehat{x}_i.$$

Computing the b for all B buckets takes $O(B \log n)$ time.

- Time-shift x by one and repeat: $b' = \widehat{x}_i \omega^i$.
- Divide to get $b'/b = \omega^i \implies$ can compute i if it is isolated.
- Dilation in time $\implies$ dilation in frequency; randomizes isolation.

Algorithm for *exactly sparse* signals

Lemma

If i is isolated in its bucket and in the "super-pass" region, the value b we compute for its bucket satisfies

$$b = \widehat{x}_i.$$

Computing the b for all B buckets takes $O(B \log n)$ time.

- Time-shift x by one and repeat: $b' = \widehat{x}_i \omega^i$.
- Divide to get $b'/b = \omega^i \implies$ can compute i if it is isolated.
- Dilation in time $\implies$ dilation in frequency; randomizes isolation.
- Gives weak sparse recovery $\widehat{x}'$ such that $\widehat{x} - \widehat{x}'$ is $k/2$-sparse.

Algorithm for *exactly sparse* signals

Lemma

If i is isolated in its bucket and in the "super-pass" region, the value b we compute for its bucket satisfies

$$b = \widehat{x}_i.$$

Computing the b for all B buckets takes $O(B \log n)$ time.

- Time-shift x by one and repeat: $b' = \widehat{x}_i \omega^i$.
- Divide to get $b'/b = \omega^i \implies$ can compute i if it is isolated.
- Dilation in time $\implies$ dilation in frequency; randomizes isolation.
- Gives weak sparse recovery $\widehat{x}'$ such that $\widehat{x} - \widehat{x}'$ is $k/2$-sparse.
  - Repeat on residual to recover $\widehat{x}$ exactly.

Algorithm for *exactly sparse* signals

Lemma

If i is isolated in its bucket and in the "super-pass" region, the value b we compute for its bucket satisfies

$$b = \widehat{x}_i.$$

Computing the b for all B buckets takes $O(B \log n)$ time.

- Time-shift x by one and repeat: $b' = \widehat{x}_i \omega^i$.
- Divide to get $b'/b = \omega^i \implies$ can compute i if it is isolated.
- Dilation in time $\implies$ dilation in frequency; randomizes isolation.
- Gives weak sparse recovery $\widehat{x}'$ such that $\widehat{x} - \widehat{x}'$ is $k/2$-sparse.
  - Repeat on residual to recover $\widehat{x}$ exactly.
  - Can't evaluate x' in time domain, but can hash $\widehat{x'}$ directly.

Algorithm for *exactly sparse* signals

Lemma

If i is isolated in its bucket and in the "super-pass" region, the value b we compute for its bucket satisfies

$$b = \widehat{x}_i.$$

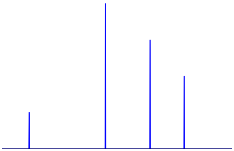
Computing the b for all B buckets takes $O(B \log n)$ time.

- Time-shift x by one and repeat: $b' = \widehat{x}_i \omega^i$.
- Divide to get $b'/b = \omega^i \implies$ can compute i if it is isolated.
- Dilation in time $\implies$ dilation in frequency; randomizes isolation.
- Gives weak sparse recovery $\widehat{x}'$ such that $\widehat{x} - \widehat{x}'$ is $k/2$-sparse.
  - Repeat on residual to recover $\widehat{x}$ exactly.
  - Can't evaluate x' in time domain, but can hash $\widehat{x'}$ directly.
- Gives $O(k \log n)$ time sparse Fourier transform. $\square$

Outline

Approximate sparsity

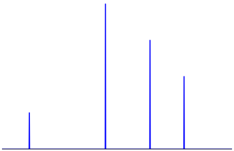
Exact Sparsity



Our Result: $O(k \log n)$

Approximate sparsity

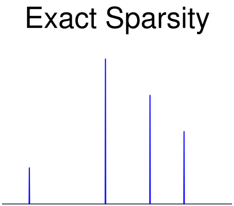
Exact Sparsity

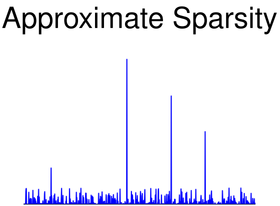


Our Result: $O(k \log n)$

- What if there's mass outside top k elements x_k?

Approximate sparsity



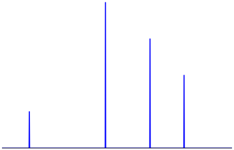


Our Result: $O(k \log n)$ $O(k \log(n/k) \log n)$

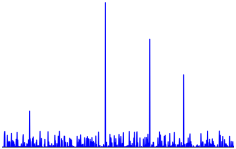
- What if there's mass outside top k elements x_k?

Approximate sparsity

Exact Sparsity



Approximate Sparsity

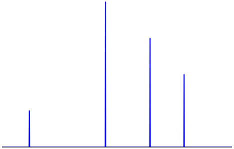


Our Result: $O(k \log n)$ $O(k \log(n/k) \log n)$

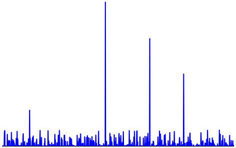
- What if there's mass outside top k elements x_k?
- Error proportional to noise $\|x - x_k\|_2^2$.

Approximate sparsity

Exact Sparsity



Approximate Sparsity

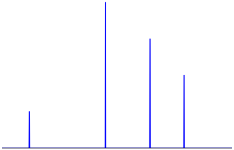


Our Result: $O(k \log n)$ $O(k \log(n/k) \log n)$

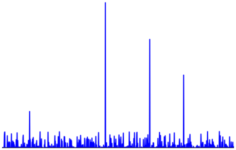
- What if there's mass outside top k elements x_k?
- Error proportional to noise $\|x - x_k\|_2^2$.
- For *all* x, work with 90% probability.

Approximate sparsity

Exact Sparsity



Approximate Sparsity

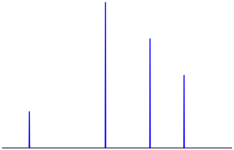


Our Result: $O(k \log n)$ $O(k \log(n/k) \log n)$

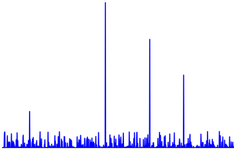
- What if there's mass outside top k elements x_k?
- Error proportional to noise $\|x - x_k\|_2^2$.
- For *all* x, work with 90% probability.
- Bridge between the two results?

Approximate sparsity

Exact Sparsity



Approximate Sparsity

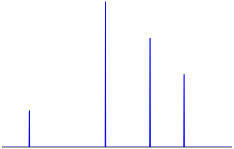


Our Result: $O(k \log n)$ $O^*(k \log_{1+SNR}(n/k) \log n)$

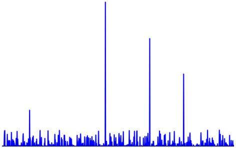
- What if there's mass outside top k elements x_k?
- Error proportional to noise $\|x - x_k\|_2^2$.
- For *all* x, work with 90% probability.
- Bridge between the two results?
    - Signal-to-Noise Ratio $\|x_k\|_2^2 / \|x - x_k\|_2^2$.

Approximate sparsity

Exact Sparsity



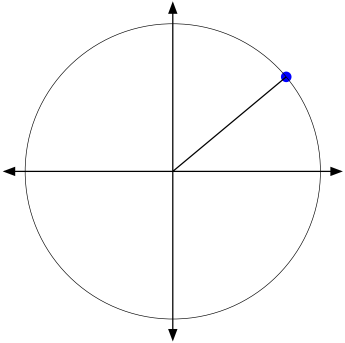
Approximate Sparsity



Our Result: $O(k \log n)$ $O(k \log(n/k) \log n)$

- What if there's mass outside top k elements x_k?
- Error proportional to noise $\|x - x_k\|_2^2$.
- For *all* x, work with 90% probability.
- Bridge between the two results?
  - Signal-to-Noise Ratio $\|x_k\|_2^2 / \|x - x_k\|_2^2$.

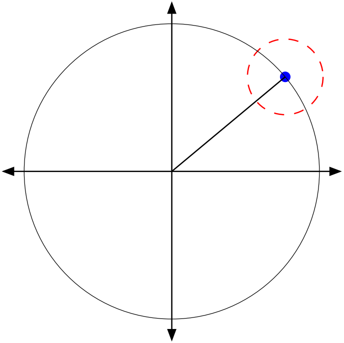
Algorithm for *approximately sparse* signals



$$b'/b = \omega^i$$

- Run same algorithm as for exact sparsity:
  - Compute $O(k)$ buckets in $O(k \log n)$ time, samples.
  - For each bucket with isolated i, get $b'/b \approx \omega^i$.

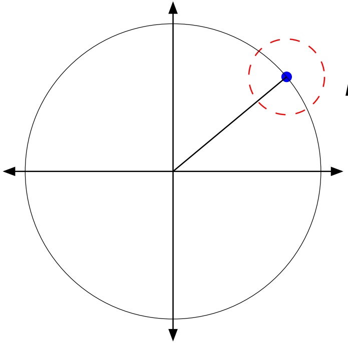
Algorithm for *approximately sparse* signals



$b'/b = \omega^i + \text{noise}$

- Run same algorithm as for exact sparsity:
  - Compute $O(k)$ buckets in $O(k \log n)$ time, samples.
  - For each bucket with isolated i, get $b'/b \approx \omega^i$.
- Approximately sparse: estimates have noise.

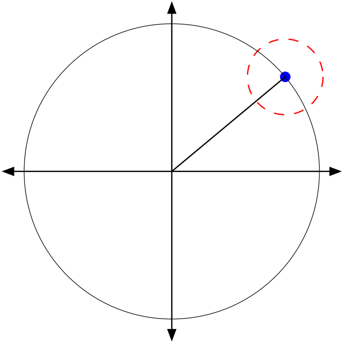
Algorithm for *approximately sparse* signals



$$b'/b = \omega^i + \text{noise}$$

- Run same algorithm as for exact sparsity:
  - Compute $O(k)$ buckets in $O(k \log n)$ time, samples.
  - For each bucket with isolated i, get $b'/b \approx \omega^i$.
- Approximately sparse: estimates have noise.
- Only $\log SNR = O(1)$ useful bits.

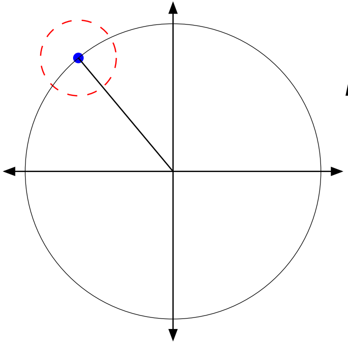
Algorithm for *approximately sparse* signals



$b'/b = \omega^i + \text{noise}$

- Run same algorithm as for exact sparsity:
  - Compute $O(k)$ buckets in $O(k \log n)$ time, samples.
  - For each bucket with isolated i, get $b'/b \approx \omega^i$.
- Approximately sparse: estimates have noise.
- Only $\log SNR = O(1)$ useful bits.
- Goal: choose $\Theta(\log(n/k))$ time shifts c to recover i.

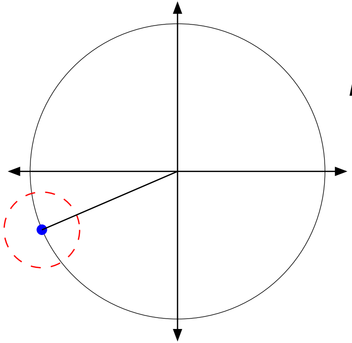
Algorithm for *approximately sparse* signals



$$b'/b = \omega^{c_2 i} + \text{noise}$$

- Run same algorithm as for exact sparsity:
  - Compute $O(k)$ buckets in $O(k \log n)$ time, samples.
  - For each bucket with isolated i, get $b'/b \approx \omega^i$.
- Approximately sparse: estimates have noise.
- Only $\log SNR = O(1)$ useful bits.
- Goal: choose $\Theta(\log(n/k))$ time shifts c to recover i.

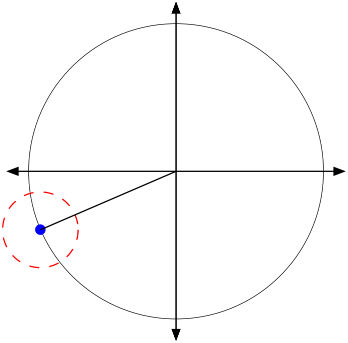
Algorithm for *approximately sparse* signals



$$b'/b = \omega^{c_3 i} + \text{noise}$$

- Run same algorithm as for exact sparsity:
  - Compute $O(k)$ buckets in $O(k \log n)$ time, samples.
  - For each bucket with isolated i, get $b'/b \approx \omega^i$.
- Approximately sparse: estimates have noise.
- Only $\log SNR = O(1)$ useful bits.
- Goal: choose $\Theta(\log(n/k))$ time shifts c to recover i.

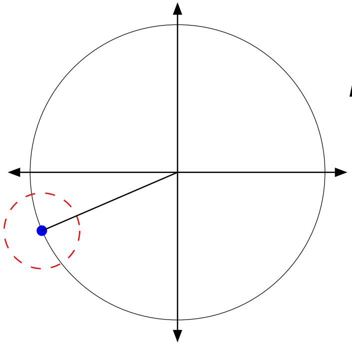
Recovering i from random time shifts c



$$b'/b = \omega^{c_3 i} + \text{noise}$$

- Choose $\Theta(\log(n/k))$ time shifts c, or "measurements".

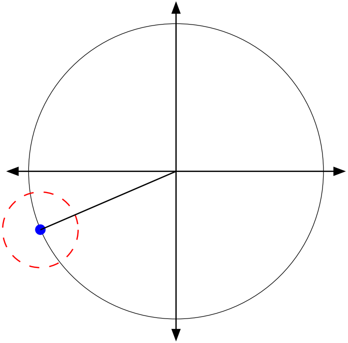
Recovering i from random time shifts c



$$b'/b = \omega^{c_3 i} + \text{noise}$$

- Choose $\Theta(\log(n/k))$ time shifts c, or “measurements”.
- Takes $O(k \log n \log(n/k))$ time to compute all measurements.

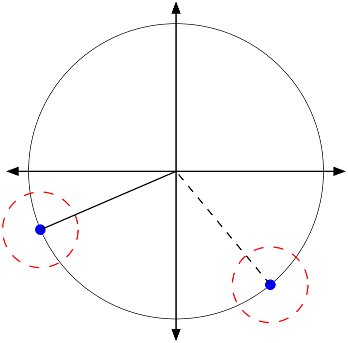
Recovering i from random time shifts c



$$b'/b = \omega^{c_3 i} + \text{noise}$$

- Choose $\Theta(\log(n/k))$ time shifts c, or "measurements".
- Takes $O(k \log n \log(n/k))$ time to compute all measurements.
- Random measurements can identify i among n/k in bucket:

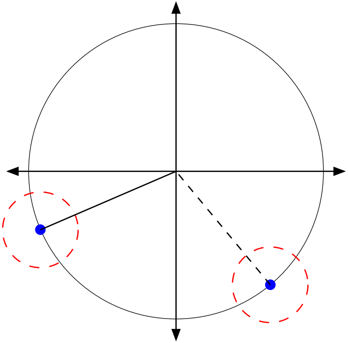
Recovering i from random time shifts c



$$b'/b = \omega^{c_3 i} + \text{noise}$$

- Choose $\Theta(\log(n/k))$ time shifts c, or "measurements".
- Takes $O(k \log n \log(n/k))$ time to compute all measurements.
- Random measurements can identify i among n/k in bucket:
  - If $j \neq i$, $\omega^{ci} - \omega^{cj}$ is usually "large".

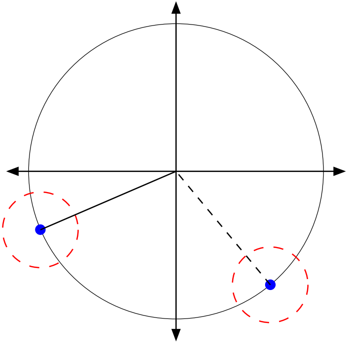
Recovering i from random time shifts c



$$b'/b = \omega^{c_3 i} + \text{noise}$$

- Choose $\Theta(\log(n/k))$ time shifts c, or "measurements".
- Takes $O(k \log n \log(n/k))$ time to compute all measurements.
- Random measurements can identify i among n/k in bucket:
  - If $j \neq i$, $\omega^{ci} - \omega^{cj}$ is usually "large".
  - If so, distinguish $\omega^i + \text{noise}$ from $\omega^j + \text{noise}$.

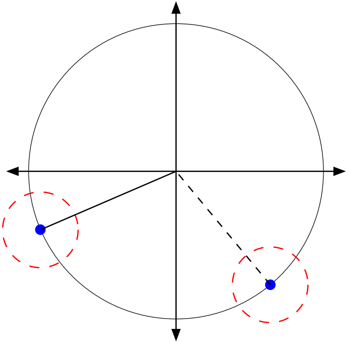
Recovering i from random time shifts c



$$b'/b = \omega^{c_3 i} + \text{noise}$$

- Choose $\Theta(\log(n/k))$ time shifts c, or "measurements".
- Takes $O(k \log n \log(n/k))$ time to compute all measurements.
- Random measurements can identify i among n/k in bucket:
  - If $j \neq i$, $\omega^{ci} - \omega^{cj}$ is usually "large".
  - If so, distinguish $\omega^i + \text{noise}$ from $\omega^j + \text{noise}$.
- But... time to decode i?

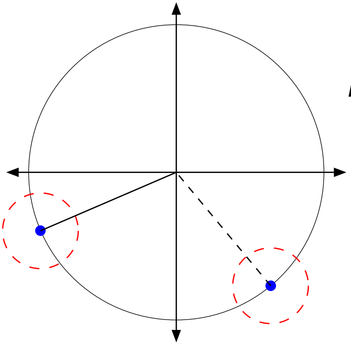
Recovering i from random time shifts c



$$b'/b = \omega^{c_3 i} + \text{noise}$$

- Choose $\Theta(\log(n/k))$ time shifts c, or "measurements".
- Takes $O(k \log n \log(n/k))$ time to compute all measurements.
- Random measurements can identify i among n/k in bucket:
  - If $j \neq i$, $\omega^{ci} - \omega^{cj}$ is usually "large".
  - If so, distinguish ω^i + noise from ω^j + noise.
- But... time to decode i?
- Decoding k buckets, so need an efficiently decodable "code".

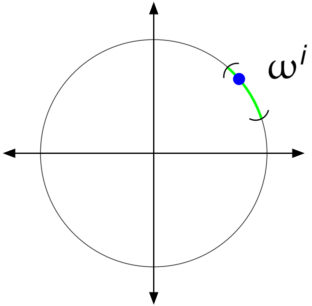
Recovering i from random time shifts c



$$b'/b = \omega^{c_3 i} + \text{noise}$$

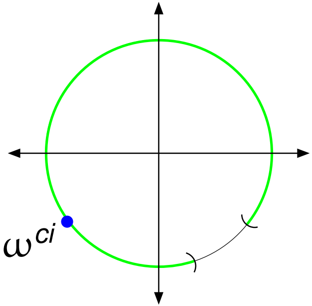
- Choose $\Theta(\log(n/k))$ time shifts c, or "measurements".
- Takes $O(k \log n \log(n/k))$ time to compute all measurements.
- Random measurements can identify i among n/k in bucket:
  - If $j \neq i$, $\omega^{ci} - \omega^{cj}$ is usually "large".
  - If so, distinguish $\omega^i + \text{noise}$ from $\omega^j + \text{noise}$.
- But... time to decode i?
- Decoding k buckets, so need an efficiently decodable "code".
  - Constant rate, quadratic decoding time.

Recover high order bits in sequence



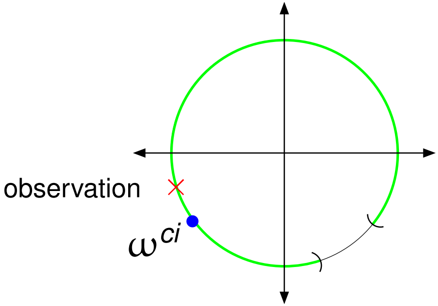
- Want to find i in contiguous region of size $R = n/k$.

Recover high order bits in sequence



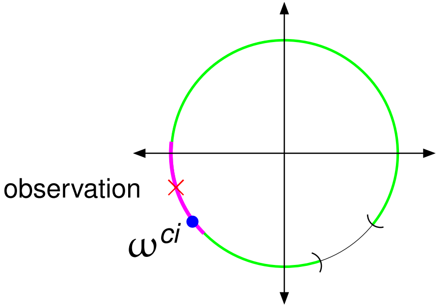
- Want to find i in contiguous region of size $R = n/k$.
- Choose $c \approx n/R$ so possibilities cover most of circle.

Recover high order bits in sequence



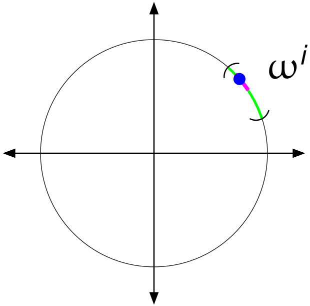
- Want to find i in contiguous region of size $R = n/k$.
- Choose $c \approx n/R$ so possibilities cover most of circle.

Recover high order bits in sequence



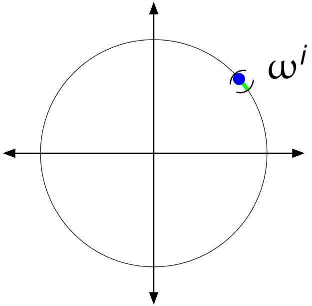
- Want to find i in contiguous region of size $R = n/k$.
- Choose $c \approx n/R$ so possibilities cover most of circle.
- Observation identifies i inside constant fraction of region.

Recover high order bits in sequence



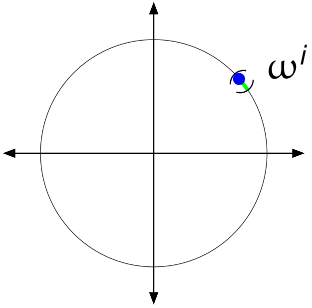
- Want to find i in contiguous region of size $R = n/k$.
- Choose $c \approx n/R$ so possibilities cover most of circle.
- Observation identifies i inside constant fraction of region.

Recover high order bits in sequence



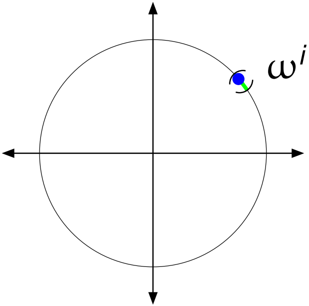
- Want to find i in contiguous region of size $R = n/k$.
- Choose $c \approx n/R$ so possibilities cover most of circle.
- Observation identifies i inside constant fraction of region.
- Restrict and repeat, $O(\log(n/k))$ times.

Recover high order bits in sequence



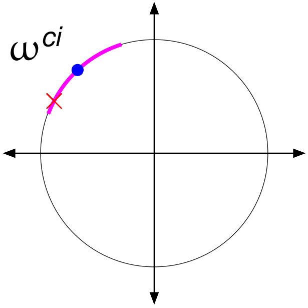
- Want to find i in contiguous region of size $R = n/k$.
- Choose $c \approx n/R$ so possibilities cover most of circle.
- Observation identifies i inside constant fraction of region.
- Restrict and repeat, $O(\log(n/k))$ times.
- Linear decoding time.

Recover high order bits in sequence



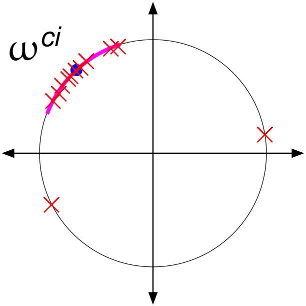
- Want to find i in contiguous region of size $R = n/k$.
- Choose $c \approx n/R$ so possibilities cover most of circle.
- Observation identifies i inside constant fraction of region.
- Restrict and repeat, $O(\log(n/k))$ times.
- Linear decoding time.
  - But rate isn't constant, just $1/\log\log(n/k)$...

Recover high order bits in sequence



- Want to find i in contiguous region of size $R = n/k$.
- Choose $c \approx n/R$ so possibilities cover most of circle.
- Observation identifies i inside constant fraction of region.
- Restrict and repeat, $O(\log(n/k))$ times.
- Linear decoding time.
  - But rate isn't constant, just $1/\log\log(n/k)$...

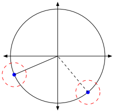
Recover high order bits in sequence



- Want to find i in contiguous region of size $R = n/k$.
- Choose $c \approx n/R$ so possibilities cover most of circle.
- Observation identifies i inside constant fraction of region.
- Restrict and repeat, $O(\log(n/k))$ times.
- Linear decoding time.
  - But rate isn't constant, just $1/\log\log(n/k)$...

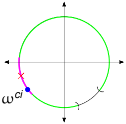
Combining the two approaches

Random



Constant rate
Exponential runtime
Exponentially small failure

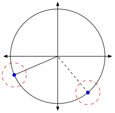
Largest bit, sequentially



Linear runtime
Constant failure per bit

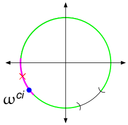
Combining the two approaches

Random



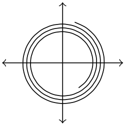
Constant rate
Exponential runtime
Exponentially small failure

Largest bit, sequentially



Linear runtime
Constant failure per bit

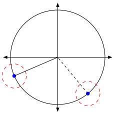
$\log\log(n/k)$ largest bits, sequentially



$R \to R/\log(n/k)$
$O(\log(n/k))$ runtime
$1/\log^c(n/k)$ failure

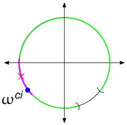
Combining the two approaches

Random



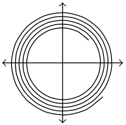
Constant rate
Exponential runtime
Exponentially small failure

Largest bit, sequentially



Linear runtime
Constant failure per bit

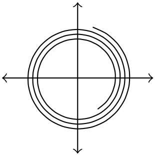
$\log\log(n/k)$ largest bits, sequentially



$R \to R/\log(n/k)$
$O(\log(n/k))$ runtime
$1/\log^c(n/k)$ failure

Our Decoding Procedure

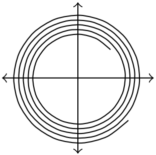
$\log\log(n/k)$ largest bits, sequentially



$R \to R/\log(n/k)$
$O(\log^2(n/k))$ runtime
$1/\log^c(n/k)$ failure

Our Decoding Procedure

$\log\log(n/k)$ largest bits, sequentially

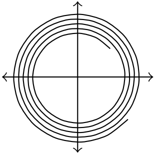


$R \to R/\log(n/k)$
$O(\log^2(n/k))$ runtime
$1/\log^c(n/k)$ failure

- $O(\log^2(n/k))$ time to decode each coordinate.

Our Decoding Procedure

$\log\log(n/k)$ largest bits, sequentially



$R \to R/\log(n/k)$
$O(\log^2(n/k))$ runtime
$1/\log^c(n/k)$ failure

- $O(\log^2(n/k))$ time to decode each coordinate.
- Gives $O(k\log(n/k)\log n)$ time/sample complexity.

Conclusion

- $O(k \log n)$ for exactly sparse $\widehat{x}$
- $O(k \log(n/k) \log n)$ for approximation.

Conclusion

- $O(k \log n)$ for exactly sparse $\widehat{x}$
- $O(k \log(n/k) \log n)$ for approximation.
- Can we do better?

Conclusion

- $O(k \log n)$ for exactly sparse $\widehat{x}$
- $O(k \log(n/k) \log n)$ for approximation.
- Can we do better?
- The $\log n$ sample complexity loss comes from our filters.

Conclusion

- $O(k \log n)$ for exactly sparse $\widehat{x}$
- $O(k \log(n/k) \log n)$ for approximation.
- Can we do better?
- The $\log n$ sample complexity loss comes from our filters.
- Avoidable if domain has enough subgroups.
  - Sparse Hadamard transform: $\log n$-dimensional DFT.
  - (Piotr's talk) 2-dimensional DFT *on average*.
  - Similar results for 1-dimensional DFT if n has enough factors.

Conclusion

- $O(k \log n)$ for exactly sparse $\widehat{x}$
- $O(k \log(n/k) \log n)$ for approximation.
- Can we do better?
- The $\log n$ sample complexity loss comes from our filters.
- Avoidable if domain has enough subgroups.
  - Sparse Hadamard transform: $\log n$-dimensional DFT.
  - (Piotr's talk) 2-dimensional DFT *on average*.
  - Similar results for 1-dimensional DFT if n has enough factors.
- Can we avoid it in general?