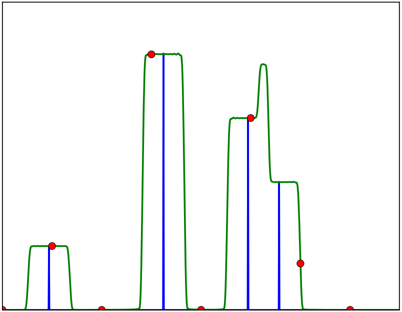
Nearly Optimal Sparse Fourier Transform

Haitham Hassanieh Piotr Indyk Dina Katabi **Eric Price**

MIT

2012-04-27



Outline

1. Introduction

Outline

1. Introduction
2. Special case: exactly sparse signals

Outline

Outline

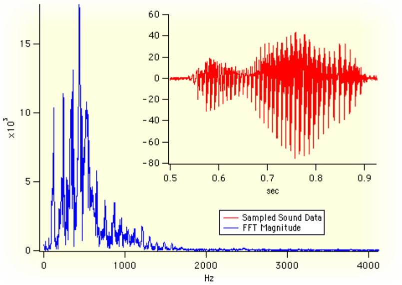
Outline

1. Introduction
2. Special case: exactly sparse signals
3. General case: approximately sparse signals
4. Experiments

The Dicrete Fourier Transform

- Discrete Fourier transform: given $x \in \mathbb{C}^n$, find

$$\widehat{x}_i = \sum x_j \omega^{ij}$$

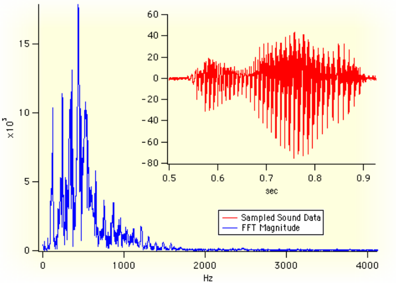


The Dicrete Fourier Transform

- Discrete Fourier transform: given $x \in \mathbb{C}^n$, find

$$\widehat{x}_i = \sum x_j \omega^{ij}$$

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



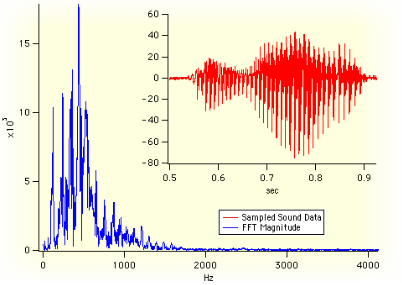
The Dicrete Fourier Transform

- Discrete Fourier transform: given $x \in \mathbb{C}^n$, find

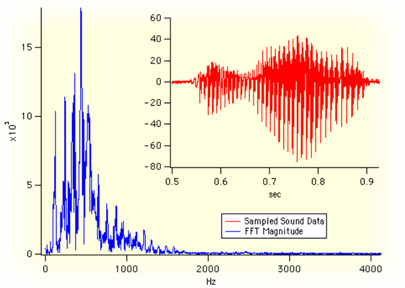
$$\widehat{x}_i = \sum x_j \omega^{ij}$$

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

- Fundamental tool
  - Compression (audio, image, video)
  - Signal processing
  - Data analysis
  - ...
- FFT: $O(n \log n)$ time.

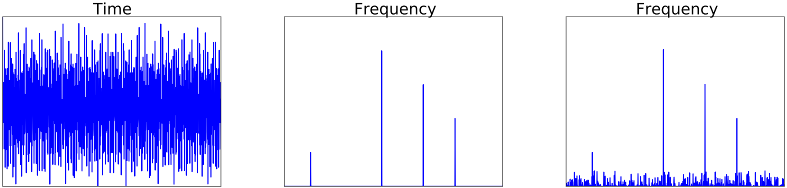


Sparse Fourier Transform



- Often the Fourier transform is dominated by a small number of "peaks"
  - Precisely the reason to use for compression.
- If most of mass in k locations, can we compute FFT faster?

Sparse Fourier Transform



- If at most k non-zero coefficients, then "exactly k-sparse."
- More often well approximated by k largest coefficients: "approximately k-sparse."

Previous work

- Boolean cube: [KM92], [GL89]. $\mathbb{C}^n$: [Mansour92] $k^c \log^c n$.
- Long line of additional work [GGIMS02, AGS03, Iwen10, Aka10]
- Fastest is [Gilbert-Muthukrishnan-Strauss-05]: $k \log^4 n$.

Previous work

- Boolean cube: [KM92], [GL89]. $\mathbb{C}^n$: [Mansour92] $k^c \log^c n$.
- Long line of additional work [GGIMS02, AGS03, Iwen10, Aka10]
- Fastest is [Gilbert-Muthukrishnan-Strauss-05]: $k \log^4 n$.
  - All have poor constants, many logs.

Previous work

- Boolean cube: [KM92], [GL89]. $\mathbb{C}^n$: [Mansour92] $k^c \log^c n$.
- Long line of additional work [GGIMS02, AGS03, Iwen10, Aka10]
- Fastest is [Gilbert-Muthukrishnan-Strauss-05]: $k \log^4 n$.
  - All have poor constants, many logs.
  - Need $n/k > 40{,}000$ or $\omega(\log^3 n)$ to beat FFTW.

Previous work

- Boolean cube: [KM92], [GL89]. $\mathbb{C}^n$: [Mansour92] $k^c \log^c n$.
- Long line of additional work [GGIMS02, AGS03, Iwen10, Aka10]
- Fastest is [Gilbert-Muthukrishnan-Strauss-05]: $k \log^4 n$.
  - All have poor constants, many logs.
  - Need $n/k > 40{,}000$ or $\omega(\log^3 n)$ to beat FFTW.
  - Our goal: faster, beat FFTW for smaller n/k in theory and practice.

Our results

- $O(k \log(n/k) \log n)$ time.
- $O(k \log n)$ for special case: exactly k-sparse.
- Faster than FFT when $n/k = \omega(1)$.
- Lower bounds:
  - $\Omega(k \log k)$ for special case assuming FFT is optimal.
  - For general case, $\Omega(k \log(n/k)/ \log\log(n/k))$ samples even with *adaptive* sampling.

Our results

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

Our results

- Compute the k-sparse Fourier transform in $O(k \log(n/k) \log n)$ time.
- Get $\widehat{x'}$ with approximation error

$$\|\widehat{x'} - \widehat{x}\|_2^2 \leq 2 \min_{k\text{-sparse } \widehat{x}_k} \|\widehat{x} - \widehat{x}_k\|_2^2$$

with $3/4$ probability.

Our results

- Compute the k-sparse Fourier transform in $O(k \log(n/k) \log n)$ time.
- Get $\widehat{x'}$ with approximation error

$$\|\widehat{x'} - \widehat{x}\|_2^2 \leq 2 \min_{k\text{-sparse } \widehat{x}_k} \|\widehat{x} - \widehat{x}_k\|_2^2$$

  with $3/4$ probability.
- If $\widehat{x}$ is sparse, recover it exactly.

Our results

- Compute the k-sparse Fourier transform in $O(k \log(n/k) \log n)$ time.
- Get $\widehat{x'}$ with approximation error

$$\|\widehat{x'} - \widehat{x}\|_2^2 \leq 2 \min_{k\text{-sparse } \widehat{x_k}} \|\widehat{x} - \widehat{x_k}\|_2^2$$

  with $3/4$ probability.
- If $\widehat{x}$ is sparse, recover it exactly.
  - In $O(k \log n)$ time.

Our results

- Compute the k-sparse Fourier transform in $O(k \log(n/k) \log n)$ time.
- Get $\widehat{x'}$ with approximation error

$$\|\widehat{x'} - \widehat{x}\|_2^2 \leq 2 \min_{k\text{-sparse } \widehat{x_k}} \|\widehat{x} - \widehat{x_k}\|_2^2$$

  with $3/4$ probability.
- If $\widehat{x}$ is sparse, recover it exactly.
  - In $O(k \log n)$ time.
- Caveats:
  - Additional $\|x\|_2^2 / n^{\Theta(1)}$ error. Alternatively, $\widehat{x}$ has poly(n) precision.
  - n must be a power of 2.

Outline

Algorithm

Suppose $\widehat{x}$ is k-sparse, with integer coefficients in $\{-n^{\Theta(1)}, \ldots, n^{\Theta(1)}\}$.

Theorem

We can recover $\widehat{x}$ in $O(k \log n)$ time with $3/4$ probability.

Algorithm

Suppose $\widehat{x}$ is k-sparse, with integer coefficients in $\{-n^{\Theta(1)}, \dots, n^{\Theta(1)}\}$.

Theorem

We can recover $\widehat{x}$ *in* $O(k \log n)$ *time with* $3/4$ *probability.*

Lemma (Weak sparse recovery)

We can recover $\widehat{x'}$ *in* $O(k \log n)$ *time with* $3/4$ *probability such that* $\widehat{x} - \widehat{x'}$ *is* $k/2$*-sparse.*

Algorithm

Suppose $\widehat{x}$ is k-sparse, with integer coefficients in $\{-n^{\Theta(1)}, \ldots, n^{\Theta(1)}\}$.

Theorem

We can recover $\widehat{x}$ in $O(k \log n)$ time with $3/4$ probability.

Lemma (Weak sparse recovery)

We can recover $\widehat{x'}$ in $O(k \log n)$ time with $3/4$ probability such that $\widehat{x} - \widehat{x'}$ is $k/2$-sparse.

- Then: repeat on $\widehat{x} - \widehat{x'}$, with $k \to k/2$ and decreasing the error probability. [Eppstein-Goodrich '07]

Inspiration: arbitrary linear measurements

Eppstein-Goodrich '07

- Get linear measurements $x_i = F_i^{-1}\widehat{x}$ of $\widehat{x}$

Inspiration: arbitrary linear measurements

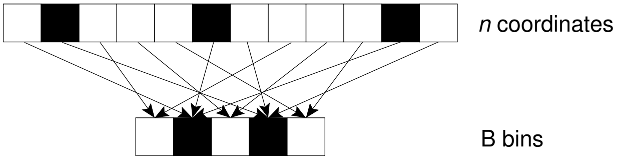
Eppstein-Goodrich '07

- Get linear measurements $x_i = F_i^{-1}\widehat{x}$ of $\widehat{x}$
- What if we could choose *arbitrary* linear measurements?

Inspiration: arbitrary linear measurements

Eppstein-Goodrich '07

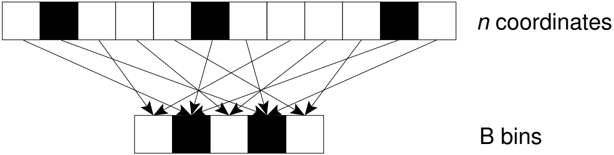
- Get linear measurements $x_i = F_i^{-1}\widehat{x}$ of $\widehat{x}$
- What if we could choose *arbitrary* linear measurements?
- Pairwise independent hash: $h : [n] \to [B]$ for $B = \Theta(k)$.



Inspiration: arbitrary linear measurements

Eppstein-Goodrich '07

- Get linear measurements $x_i = F_i^{-1}\widehat{x}$ of $\widehat{x}$
- What if we could choose *arbitrary* linear measurements?
- Pairwise independent hash: $h : [n] \to [B]$ for $B = \Theta(k)$.



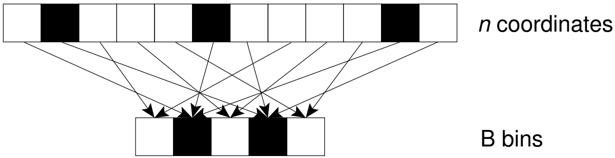
- For $j \in [B]$, observe

$$u_j = \sum_{h(i)=j} \widehat{x}_i \qquad u'_j = \sum_{h(i)=j} i \cdot \widehat{x}_i$$

Inspiration: arbitrary linear measurements

Eppstein-Goodrich '07

- Get linear measurements $x_i = F_i^{-1}\widehat{x}$ of $\widehat{x}$
- What if we could choose *arbitrary* linear measurements?
- Pairwise independent hash: $h : [n] \to [B]$ for $B = \Theta(k)$.



- For $j \in [B]$, observe

$$u_j = \sum_{h(i)=j} \widehat{x}_i \qquad\qquad u'_j = \sum_{h(i)=j} i \cdot \widehat{x}_i$$

- For each j, set $i^* = u'_j / u_j$ and $\widehat{x'}_{i^*} = u_j$.

Inspiration: arbitrary linear measurements

- For $j \in [B]$, observe

$$u_j = \sum_{h(i)=j} \widehat{x}_i \qquad\qquad u'_j = \sum_{h(i)=j} i \cdot \widehat{x}_i$$

- For each j, set $i^* = u'_j / u_j$ and $\widehat{x'}_{i^*} = u_j$.

Inspiration: arbitrary linear measurements

- For $j \in [B]$, observe

$$u_j = \sum_{h(i)=j} \widehat{x}_i \qquad\qquad u'_j = \sum_{h(i)=j} i \cdot \widehat{x}_i$$

- For each j, set $i^* = u'_j / u_j$ and $\widehat{x'}_{i^*} = u_j$.
- Gives weak sparse recovery:

Inspiration: arbitrary linear measurements

- For $j \in [B]$, observe

$$u_j = \sum_{h(i)=j} \widehat{x}_i \qquad\qquad u'_j = \sum_{h(i)=j} i \cdot \widehat{x}_i$$

- For each j, set $i^* = u'_j / u_j$ and $\widehat{x'}_{i^*} = u_j$.
- Gives weak sparse recovery:
  - If i alone in bucket $h(i)$, recovered correctly.

Inspiration: arbitrary linear measurements

- For $j \in [B]$, observe

$$u_j = \sum_{h(i)=j} \widehat{x}_i \qquad\qquad u'_j = \sum_{h(i)=j} i \cdot \widehat{x}_i$$

- For each j, set $i^* = u'_j / u_j$ and $\widehat{x'}_{i^*} = u_j$.
- Gives weak sparse recovery:
  - If i alone in bucket $h(i)$, recovered correctly.
  - Hence i recovered correctly with $1 - k/B \geq 15/16$ probability.

Inspiration: arbitrary linear measurements

- For $j \in [B]$, observe

$$u_j = \sum_{h(i)=j} \widehat{x}_i \qquad\qquad u'_j = \sum_{h(i)=j} i \cdot \widehat{x}_i$$

- For each j, set $i^* = u'_j / u_j$ and $\widehat{x'}_{i^*} = u_j$.
- Gives weak sparse recovery:
  - If i alone in bucket $h(i)$, recovered correctly.
  - Hence i recovered correctly with $1 - k/B \geq 15/16$ probability.
  - If i recovered incorrectly, can add one spurious coordinate.

Inspiration: arbitrary linear measurements

- For $j \in [B]$, observe

$$u_j = \sum_{h(i)=j} \widehat{x}_i \qquad\qquad u'_j = \sum_{h(i)=j} i \cdot \widehat{x}_i$$

- For each j, set $i^* = u'_j / u_j$ and $\widehat{x'}_{i^*} = u_j$.
- Gives weak sparse recovery:
  - If i alone in bucket $h(i)$, recovered correctly.
  - Hence i recovered correctly with $1 - k/B \geq 15/16$ probability.
  - If i recovered incorrectly, can add one spurious coordinate.
  - With $3/4$ probability, less than $k/4$ such mistakes.

Inspiration: arbitrary linear measurements

- For $j \in [B]$, observe

$$u_j = \sum_{h(i)=j} \widehat{x}_i \qquad\qquad u'_j = \sum_{h(i)=j} i \cdot \widehat{x}_i$$

- For each j, set $i^* = u'_j / u_j$ and $\widehat{x'}_{i^*} = u_j$.
- Gives weak sparse recovery:
  - If i alone in bucket $h(i)$, recovered correctly.
  - Hence i recovered correctly with $1 - k/B \geq 15/16$ probability.
  - If i recovered incorrectly, can add one spurious coordinate.
  - With $3/4$ probability, less than $k/4$ such mistakes.
  - Hence $\widehat{x} - \widehat{x'}$ is $k/2$-sparse.

Inspiration: arbitrary linear measurements

- For $j \in [B]$, observe

$$u_j = \sum_{h(i)=j} \widehat{x}_i \qquad u'_j = \sum_{h(i)=j} i \cdot \widehat{x}_i$$

- For each j, set $i^* = u'_j / u_j$ and $\widehat{x'}_{i^*} = u_j$.
- Gives weak sparse recovery:
  - If i alone in bucket $h(i)$, recovered correctly.
  - Hence i recovered correctly with $1 - k/B \geq 15/16$ probability.
  - If i recovered incorrectly, can add one spurious coordinate.
  - With $3/4$ probability, less than $k/4$ such mistakes.
  - Hence $\widehat{x} - \widehat{x'}$ is $k/2$-sparse.
- Goal: construct u, u' from Fourier samples.

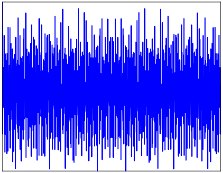
Inspiration: arbitrary linear measurements

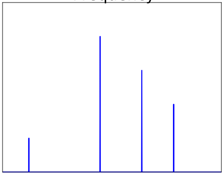
- For $j \in [B]$, observe

$$u_j = \sum_{h(i)=j} \widehat{x}_i \qquad\qquad u'_j = \sum_{h(i)=j} i \cdot \widehat{x}_i$$

- For each j, set $i^* = u'_j / u_j$ and $\widehat{x'}_{i^*} = u_j$.
- Gives weak sparse recovery:
  - If i alone in bucket $h(i)$, recovered correctly.
  - Hence i recovered correctly with $1 - k/B \geq 15/16$ probability.
  - If i recovered incorrectly, can add one spurious coordinate.
  - With $3/4$ probability, less than $k/4$ such mistakes.
  - Hence $\widehat{x} - \widehat{x'}$ is $k/2$-sparse.
- Goal: construct u, u' from Fourier samples.
  - Will be able to do this in $O(B \log n)$ time.

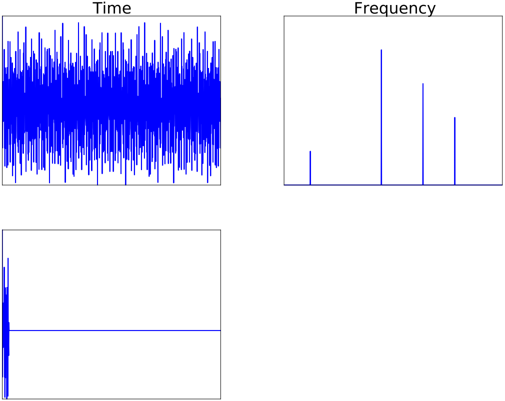
What can you do with Fourier measurements?





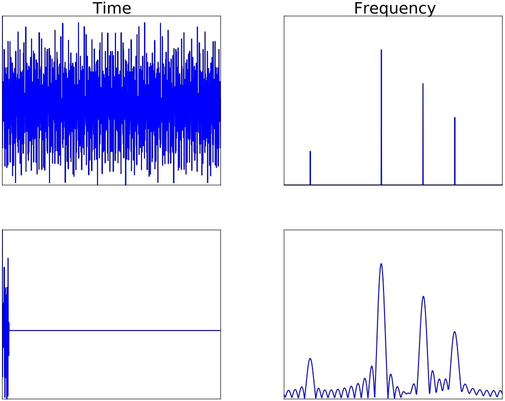
n-dimensional DFT:
$O(n \log n)$

What can you do with Fourier measurements?



n-dimensional DFT:
$O(n \log n)$

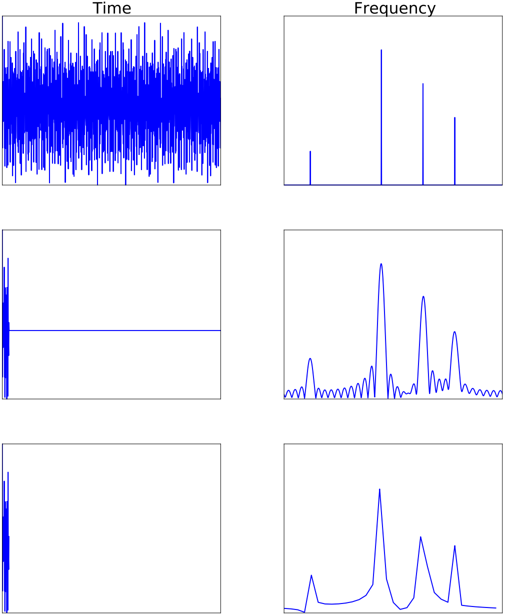
What can you do with Fourier measurements?



n-dimensional DFT: $O(n \log n)$

n-dimensional DFT of first B terms: $O(n \log n)$

What can you do with Fourier measurements?

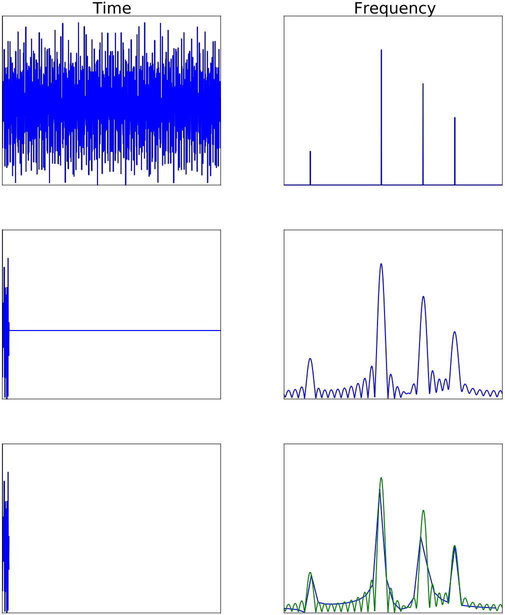


n-dimensional DFT: $O(n \log n)$

n-dimensional DFT of first B terms: $O(n \log n)$

B-dimensional DFT of first B terms: $O(B \log B)$

What can you do with Fourier measurements?

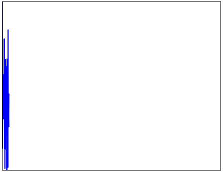


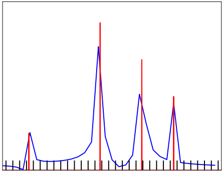
n-dimensional DFT: $O(n \log n)$

n-dimensional DFT of first B terms: $O(n \log n)$

B-dimensional DFT of first B terms: $O(B \log B)$

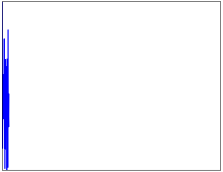
Framework

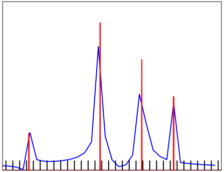




- "Hashes" into B buckets in $B \log B$ time.
- Analogous to $u_j = \sum_{h(i)=j} \widehat{x}_i$.

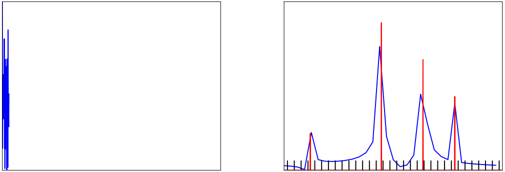
Framework





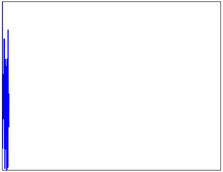
- "Hashes" into B buckets in $B \log B$ time.
- Analogous to $u_j = \sum_{h(i)=j} \widehat{x}_i$.
- Issues:
  - "Hashing" needs a random hash function
  - Leakage
  - Want analog of $u'_j = \sum_{h(i)=j} i \cdot \widehat{x}_i$.

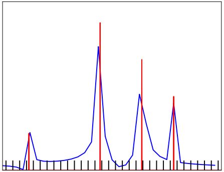
Framework



- "Hashes" into B buckets in $B \log B$ time.
- Analogous to $u_j = \sum_{h(i)=j} \widehat{x}_i$.
- Issues:
  - "Hashing" needs a random hash function
    - ★ Access $x'_t = \omega^{-bt} x_{at}$, so $\widehat{x'}_{at+b} = \widehat{x}_t$ [GMS05]
  - Leakage
  - Want analog of $u'_j = \sum_{h(i)=j} i \cdot \widehat{x}_i$.

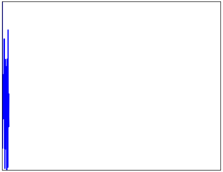
Framework

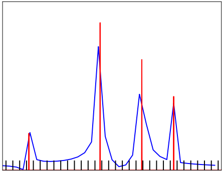




- "Hashes" into B buckets in $B \log B$ time.
- Analogous to $u_j = \sum_{h(i)=j} \widehat{x}_i$.
- Issues:
  - "Hashing" needs a random hash function
    - ★ Access $x'_t = \omega^{-bt} x_{at}$, so $\widehat{x'}_{at+b} = \widehat{x}_t$ [GMS05]
  - Leakage
  - Want analog of $u'_j = \sum_{h(i)=j} i \cdot \widehat{x}_i$.
    - ★ Time shift $x'_t = x_{t-1}$: get phase shift $\widehat{x'}_i = \omega^i \widehat{x}_i$.

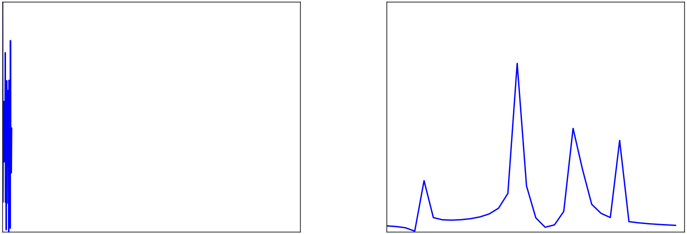
Framework





- "Hashes" into B buckets in $B \log B$ time.
- Analogous to $u_j = \sum_{h(i)=j} \widehat{x}_i$.
- Issues:
  - "Hashing" needs a random hash function
    - ★ Access $x'_t = \omega^{-bt} x_{at}$, so $\widehat{x'}_{at+b} = \widehat{x}_t$ [GMS05]
  - Leakage
  - Want analog of $u'_j = \sum_{h(i)=j} i \cdot \widehat{x}_i$.
    - ★ Time shift $x'_t = x_{t-1}$: get phase shift $\widehat{x'}_i = \omega^i \widehat{x}_i$.

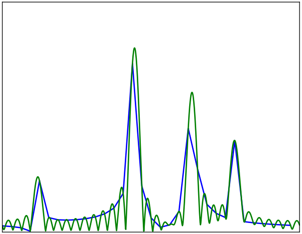
Leakage



- Let $F_i = \begin{cases} 1 & i < B \\ 0 & \text{otherwise} \end{cases}$ be the "boxcar" filter. (Used in [GGIMS02,GMS05])
- Observe

  $$\text{DFT}(F \cdot x, B)$$

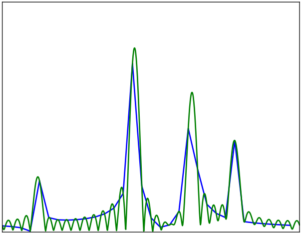
Leakage



- Let $F_i = \begin{cases} 1 & i < B \\ 0 & \text{otherwise} \end{cases}$ be the “boxcar” filter. (Used in [GGIMS02,GMS05])
- Observe

$$\mathrm{DFT}(F \cdot x, B) = \mathrm{subsample}(\mathrm{DFT}(F \cdot x, n), B)$$

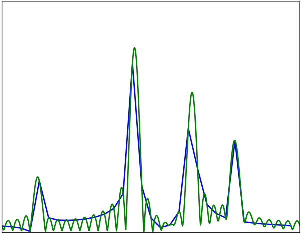
Leakage



- Let $F_i = \begin{cases} 1 & i < B \\ 0 & \text{otherwise} \end{cases}$ be the "boxcar" filter. (Used in [GGIMS02,GMS05])
- Observe

$$\mathrm{DFT}(F \cdot x, B) = \mathrm{subsample}(\mathrm{DFT}(F \cdot x, n), B) = \mathrm{subsample}(\widehat{F} * \widehat{x}, B).$$

Leakage

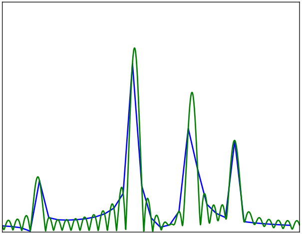


- Let $F_i = \begin{cases} 1 & i < B \\ 0 & \text{otherwise} \end{cases}$ be the “boxcar” filter. (Used in [GGIMS02,GMS05])
- Observe

$$\text{DFT}(F \cdot x, B) = \text{subsample}(\text{DFT}(F \cdot x, n), B) = \text{subsample}(\widehat{F} * \widehat{x}, B).$$

- DFT $\widehat{F}$ of boxcar filter is sinc, decays as $1/i$.

Leakage

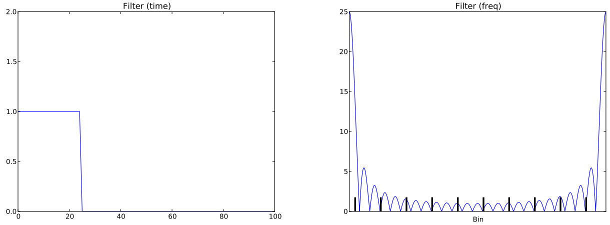


- Let $F_i = \begin{cases} 1 & i < B \\ 0 & \text{otherwise} \end{cases}$ be the "boxcar" filter. (Used in [GGIMS02,GMS05])
- Observe

$$\text{DFT}(F \cdot x, B) = \text{subsample}(\text{DFT}(F \cdot x, n), B) = \text{subsample}(\widehat{F} * \widehat{x}, B).$$

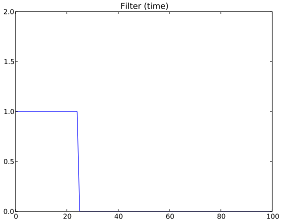
- DFT $\widehat{F}$ of boxcar filter is sinc, decays as $1/i$.
- Need a better filter F!

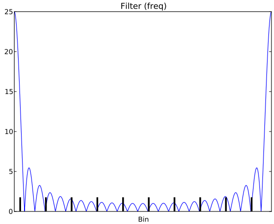
Filters



- Given $|\mathrm{supp}(F)| = B$, concentrate $\widehat{F}$.

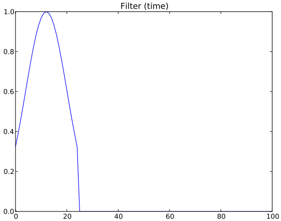
Filters

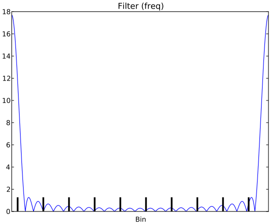




- Given $|\mathrm{supp}(F)| = B$, concentrate $\widehat{F}$.
- Boxcar filter: decays perfectly in time, $1/t$ in frequency.
  - Non-trivial leakage everywhere.

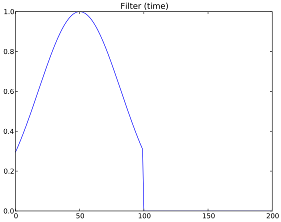
Filters

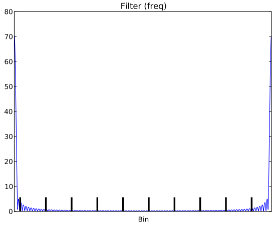




- Given $|\text{supp}(F)| = B$, concentrate $\widehat{F}$.
- Boxcar filter: decays perfectly in time, $1/t$ in frequency.
  - Non-trivial leakage everywhere.
- Gaussians: decay as e^{-t^2} in time and frequency.
  - Non-trivial leakage to $O(\sqrt{\log n} \cdot \sqrt{\log n}) = O(\log n)$ buckets.

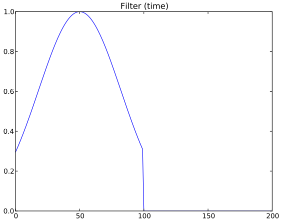
Filters

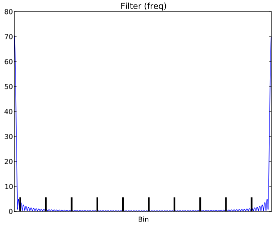




- Given $|\text{supp}(F)| = B \log n$, concentrate $\widehat{F}$.
- Boxcar filter: decays perfectly in time, $1/t$ in frequency.
  - Non-trivial leakage everywhere.
- Gaussians: decay as e^{-t^2} in time and frequency.
  - Non-trivial leakage to $O(\sqrt{\log n} \cdot \sqrt{\log n}) = O(\log n)$ buckets.
- Still $O(B \log n)$ time when $|\text{supp}(\widehat{F})| = B \log n$.
  - Non-trivial leakage to 0 buckets.

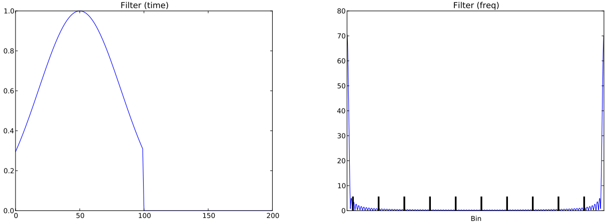
Filters





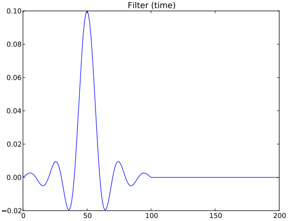
- Given $|\mathrm{supp}(F)| = B \log n$, concentrate $\widehat{F}$.
- Boxcar filter: decays perfectly in time, $1/t$ in frequency.
  - Non-trivial leakage everywhere.
- Gaussians: decay as e^{-t^2} in time and frequency.
  - Non-trivial leakage to $O(\sqrt{\log n} \cdot \sqrt{\log n}) = O(\log n)$ buckets.
- Still $O(B \log n)$ time when $|\mathrm{supp}(\widehat{F})| = B \log n$.
  - Non-trivial leakage to 0 buckets.
  - Trivial contribution to correct bucket.

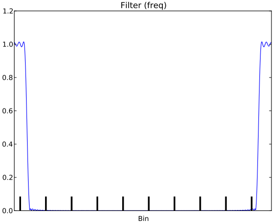
Filters



- Let G be Gaussian with $\sigma = B\sqrt{\log n}$
- H be box-car filter of length n/B.

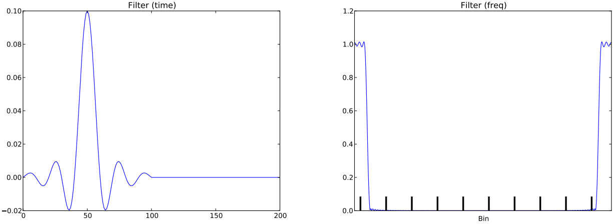
Filters





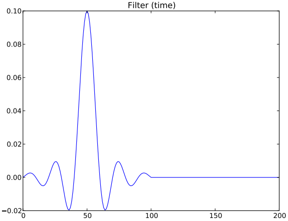
- Let G be Gaussian with $\sigma = B\sqrt{\log n}$
- H be box-car filter of length n/B.
- Use $\widehat{F} = \widehat{G} * H$.

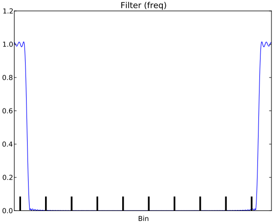
Filters



- Let G be Gaussian with $\sigma = B\sqrt{\log n}$
- H be box-car filter of length n/B.
- Use $\widehat{F} = \widehat{G} * H$.
- Hashes correctly to one bucket, leaks to at most 1 bucket.

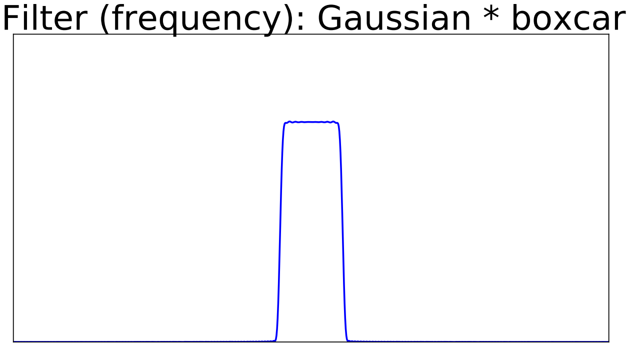
Filters



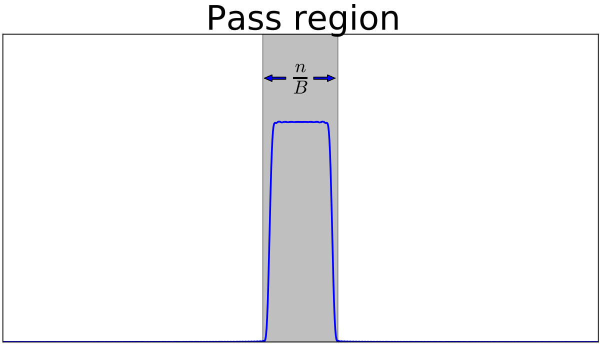


- Let G be Gaussian with $\sigma = B\sqrt{\log n}$
- H be box-car filter of length n/B.
- Use $\widehat{F} = \widehat{G} * H$.
- Hashes correctly to one bucket, leaks to at most 1 bucket.

Properties of filter

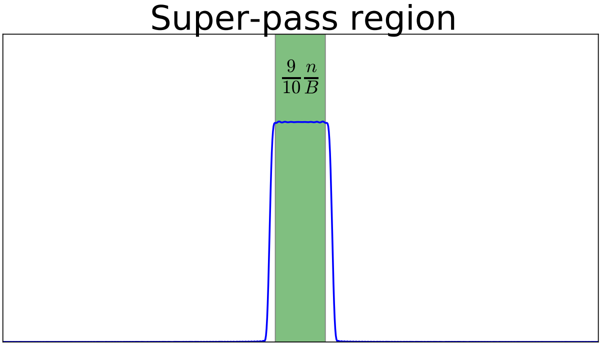


Properties of filter



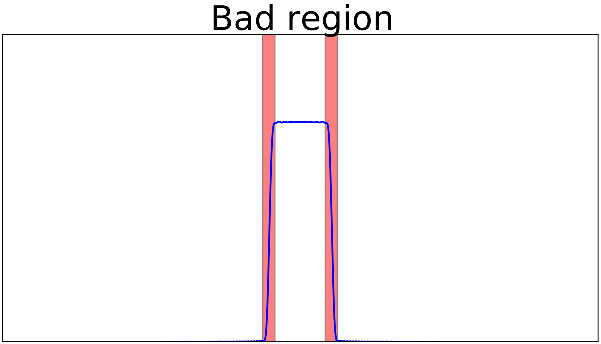
- "Pass region" of size n/B, outside which is negligible δ.

Properties of filter



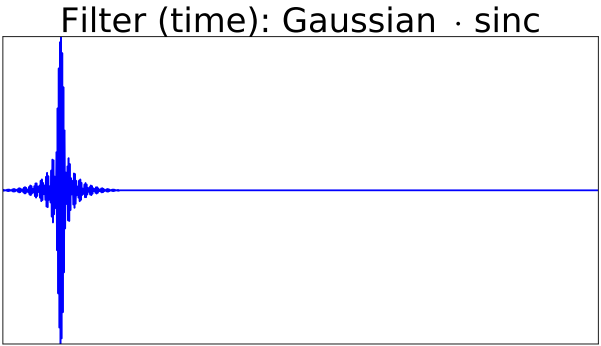
- "Pass region" of size n/B, outside which is negligible δ.
- "Super-pass region", where ≈ 1.

Properties of filter



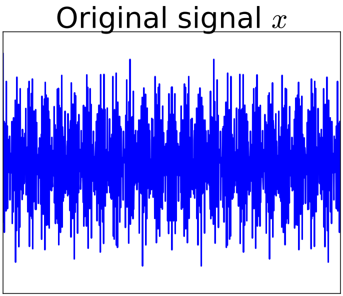
- "Pass region" of size n/B, outside which is negligible δ.
- "Super-pass region", where ≈ 1.
- Small fraction (say 10%) is "bad region" with intermediate value.

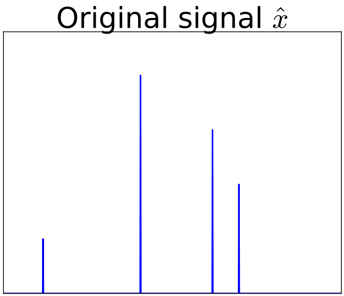
Properties of filter



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

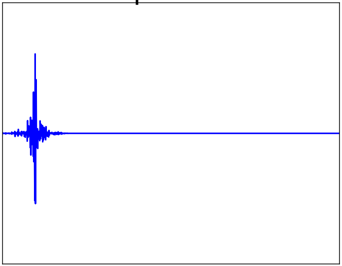
Algorithm for *exactly sparse* signals



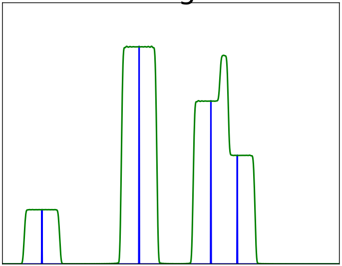


Algorithm for *exactly sparse* signals

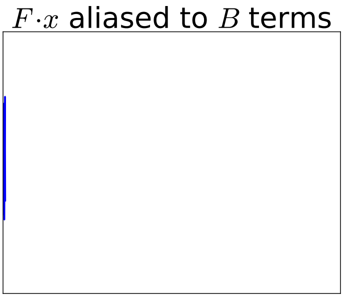
Computed $F \cdot x$

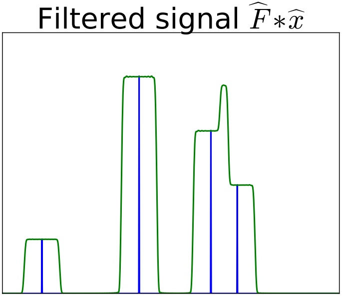


Filtered signal $\widehat{F} * \widehat{x}$

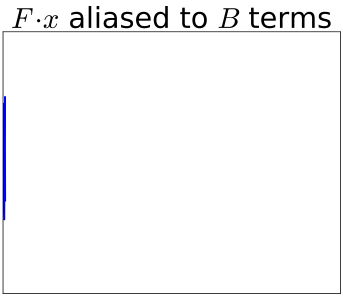


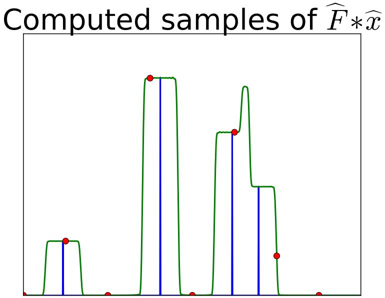
Algorithm for *exactly sparse* signals



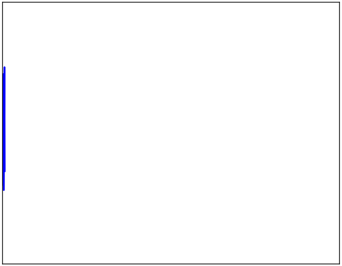


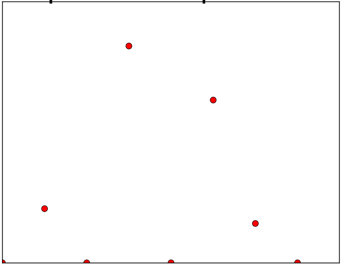
Algorithm for *exactly sparse* signals





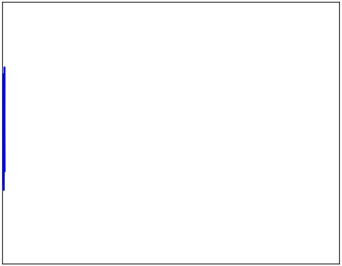
Algorithm for *exactly sparse* signals



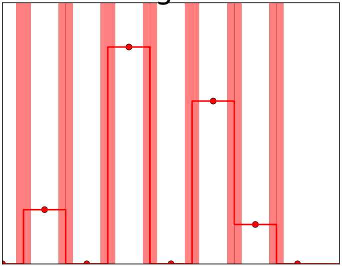


Algorithm for *exactly sparse* signals

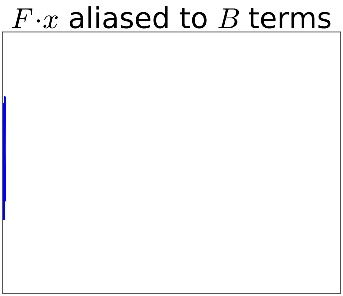
$F \cdot x$ aliased to B terms

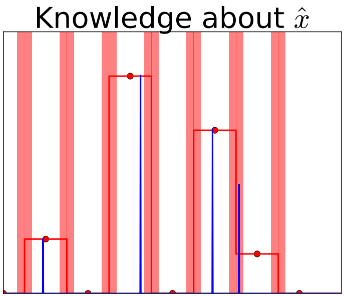


Knowledge about $\hat{x}$

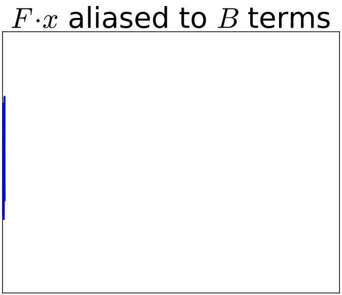


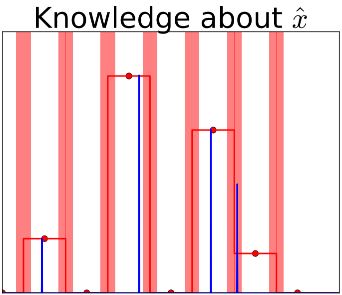
Algorithm for *exactly sparse* signals





Algorithm for *exactly sparse* signals





Lemma

If i is alone in its bucket and in the "super-pass" region,

$$u_{h(i)} = \widehat{x}_i.$$

Computing u takes $O(B \log n)$ time.

Algorithm for *perfectly sparse* signals

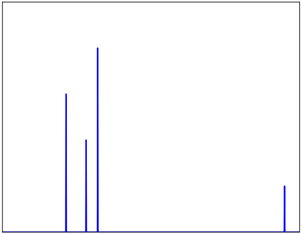
Lemma

If i is alone in its bucket and in the "super-pass" region,

$$u_{h(i)} = \widehat{x}_i.$$

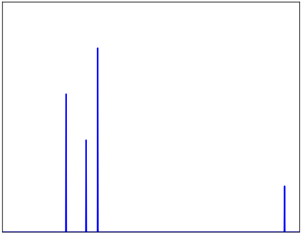
- Time-shift x by one and repeat: $u'_{h(i)} = \widehat{x}_i \omega^i$.
- Divide to find i.

Permutation in time and frequency



- Can recover coordinates that are alone in their bucket and in the super-pass region.
- What if coordinates are near each other?

Permutation in time and frequency



- Can recover coordinates that are alone in their bucket and in the super-pass region.
- What if coordinates are near each other?
- Define the "permutation"

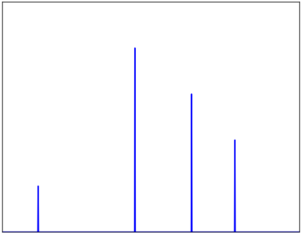
$$(P_{a,b}x)_i = x_{ai}\omega^{-ib}.$$

Then

$$\widehat{(P_{a,b}x)}_{ai+b} = \widehat{x}_i.$$

- For random a and b, each i is probably "well-hashed."

Permutation in time and frequency



- Can recover coordinates that are alone in their bucket and in the super-pass region.
- What if coordinates are near each other?
- Define the “permutation”

$$(P_{a,b}x)_i = x_{ai}\omega^{-ib}.$$

  Then

$$(\widehat{P_{a,b}x})_{ai+b} = \widehat{x}_i.$$

- For random a and b, each i is probably “well-hashed.”

Overall algorithm

- Weak sparse recovery:
  - Permute with random a, b.
  - Hash to u
  - Time shift by one, hash to u'.
  - For $j \in [B]$
    - Choose i^* by $u'_j / u_j = \omega^{i^*}$.
    - Set $\widehat{x'}_{i^*} = u_j$.

Overall algorithm

- Weak sparse recovery:
  - Permute with random a, b.
  - Hash to u
  - Time shift by one, hash to u'.
  - For $j \in [B]$
    - Choose i^* by $u'_j / u_j = \omega^{i^*}$.
    - Set $\widehat{x'}_{i^*} = u_j$.
- Full sparse recovery:
  - $\widehat{x'} \leftarrow \texttt{WeakRecovery}(x, k)$
  - $k \rightarrow k/2$, $x \rightarrow (x - x')$, repeat.

Overall algorithm

- Weak sparse recovery:
  - Permute with random a, b.
  - Hash to u
  - Time shift by one, hash to u'.
  - For $j \in [B]$
    - ★ Choose i^* by $u'_j / u_j = \omega^{i^*}$.
    - ★ Set $\widehat{x'}_{i*} = u_j$.
- Full sparse recovery:
  - $\widehat{x'} \leftarrow \texttt{WeakRecovery}(x, k)$
  - $k \rightarrow k/2$, $x \rightarrow (x - x')$, repeat.
- Time dominated by hash to $B_r = k/2^r$ buckets in round r:
  - $B_r \log n$ to hash x.
  - Hashing $\widehat{x'}$ takes $O(|\text{supp}(\widehat{x'})|) = O(k)$.

Overall algorithm

- Weak sparse recovery:
  - Permute with random a, b.
  - Hash to u
  - Time shift by one, hash to u'.
  - For $j \in [B]$
    - Choose i^* by $u'_j / u_j = \omega^{i^*}$.
    - Set $\widehat{x'}_{i^*} = u_j$.
- Full sparse recovery:
  - $\widehat{x'} \leftarrow \texttt{WeakRecovery}(x, k)$
  - $k \rightarrow k/2$, $x \rightarrow (x - x')$, repeat.
- Time dominated by hash to $B_r = k/2^r$ buckets in round r:
  - $B_r \log n$ to hash x.
  - Hashing $\widehat{x'}$ takes $O(|\text{supp}(\widehat{x'})|) = O(k)$.
- Time $\sum(\frac{k}{2^r} \log n + k) = O(k \log n)$.

Outline

1. Introduction
2. Special case: exactly sparse signals
3. General case: approximately sparse signals
4. Experiments

Nearly sparse signals

- What happens if only 90% of the mass lies in top k coordinates, not 100%?
- Want to find most "heavy" coordinates i with $|\widehat{x}_i|^2 > \|x_{\text{tail}}\|_2^2/k$.

Nearly sparse signals

- What happens if only 90% of the mass lies in top k coordinates, not 100%?
- Want to find most "heavy" coordinates i with $|\widehat{x}_i|^2 > \|x_{\text{tail}}\|_2^2/k$.

Lemma

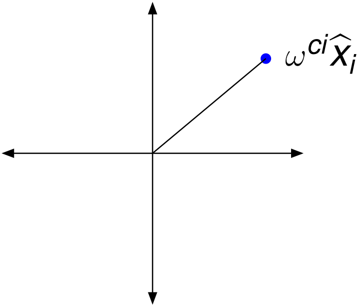
Each i is "well-hashed" with large constant probability over the permutation (a, b). If i is well-hashed, then with time shift c we have

$$u_{h(i)} = \widehat{x}_i \omega^{ci} + \eta$$

so that for random c, the noise η is bounded by

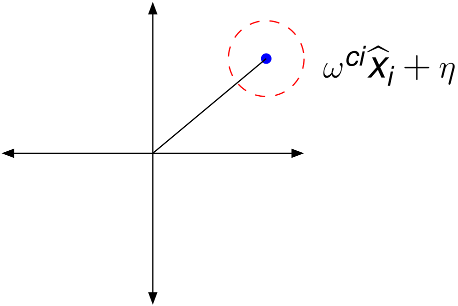
$$E[|\eta|^2] \lesssim \|x_{tail}\|_2^2/B$$

Recovering well-hashed i



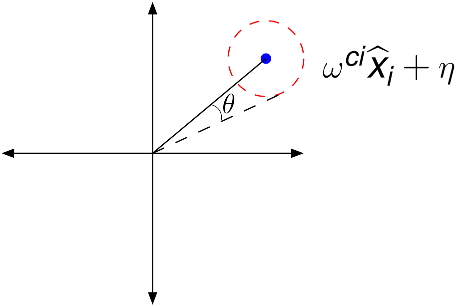
- With good probability over c, get $u_{h(i)} = \widehat{x}_i \omega^{c\pi(i)} + \eta$ with $|\eta| < |\widehat{x}_i|/10$.

Recovering well-hashed i



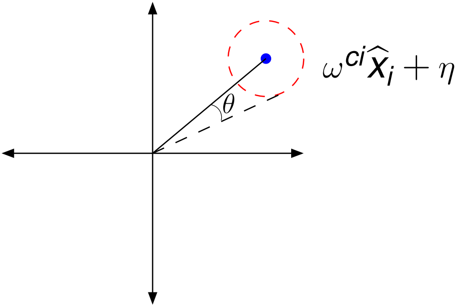
- With good probability over c, get $u_{h(i)} = \widehat{x}_i \omega^{c\pi(i)} + \eta$ with $|\eta| < |\widehat{x}_i|/10$.

Recovering well-hashed i



- With good probability over c, get $u_{h(i)} = \widehat{x}_i \omega^{c\pi(i)} + \eta$ with $|\eta| < |\widehat{x}_i|/10$.
- Phase error $|\theta| \leq \sin^{-1}(\frac{|\eta|}{|\widehat{x}_i|}) < 0.11$.

Recovering well-hashed i

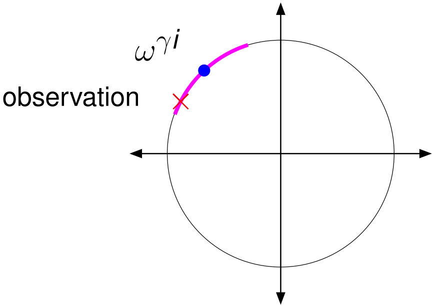


- With good probability over c, get $u_{h(i)} = \widehat{x}_i \omega^{c\pi(i)} + \eta$ with $|\eta| < |\widehat{x}_i|/10$.
- Phase error $|\theta| \leq \sin^{-1}(\frac{|\eta|}{|\widehat{x}_i|}) < 0.11$.
- True for random c. For a fixed γ, run on c and $c + \gamma$ to observe

$$\omega^{\gamma\pi(i)}$$

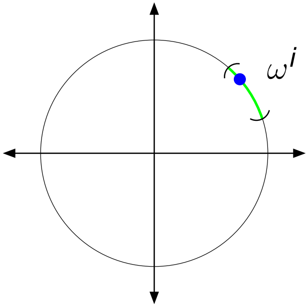
to within 0.22.

Recovering well-hashed i



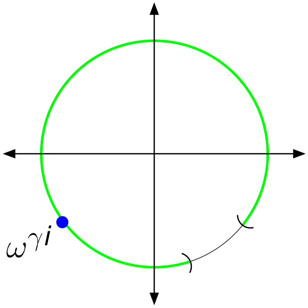
- Find i from n/k possibilities in bucket.
- Choose any γ, then observe $\omega^{\gamma i}$ to within ± 0.1 radians.
- Constant number of bits, so hope for $\Theta(\log(n/k))$ observations.

Recovering well-hashed i



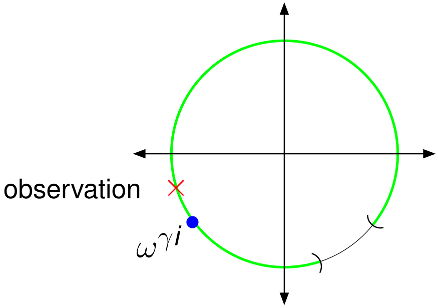
- We know i to within R.
- Set $\gamma = \lfloor n/R \rfloor$.
- Restrict and repeat, $\log(n/k)$ times.

Recovering well-hashed i



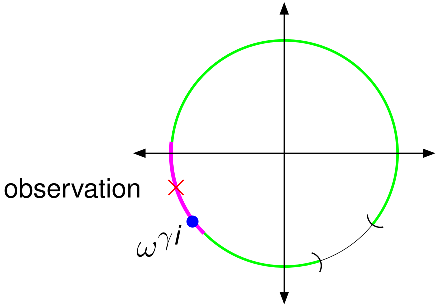
- We know i to within R.
- Set $\gamma = \lfloor n/R \rfloor$.
- Restrict and repeat, $\log(n/k)$ times.

Recovering well-hashed i



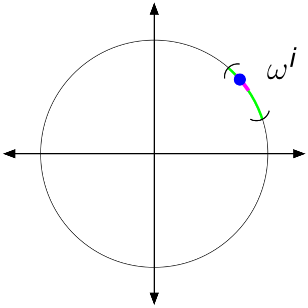
- We know i to within R.
- Set $\gamma = \lfloor n/R \rfloor$.
- Restrict and repeat, $\log(n/k)$ times.

Recovering well-hashed i



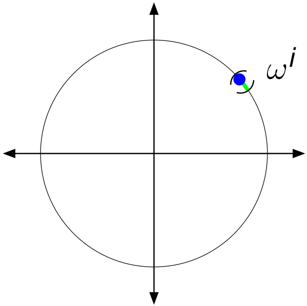
- We know i to within R.
- Set $\gamma = \lfloor n/R \rfloor$.
- Restrict and repeat, $\log(n/k)$ times.

Recovering well-hashed i



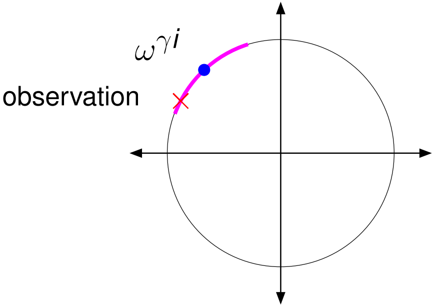
- We know i to within R.
- Set $\gamma = \lfloor n/R \rfloor$.
- Restrict and repeat, $\log(n/k)$ times.

Recovering well-hashed i



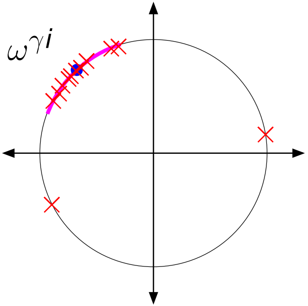
- We know i to within R.
- Set $\gamma = \lfloor n/R \rfloor$.
- Restrict and repeat, $\log(n/k)$ times.

Problem: constant failure probability per measurement



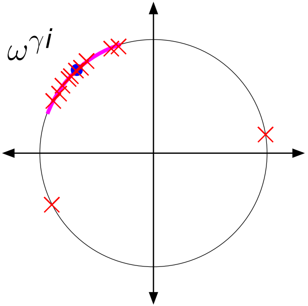
- We only estimate $\omega^{\gamma i}$ well with 90% probability.
- Some of the $\log(n/k)$ restrictions will go awry.

Problem: constant failure probability per measurement



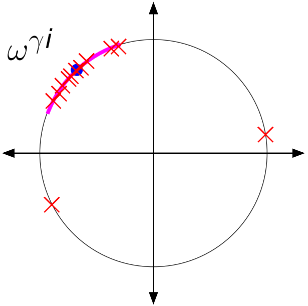
- We only estimate $\omega^{\gamma i}$ well with 90% probability.
- Some of the $\log(n/k)$ restrictions will go awry.

Problem: constant failure probability per measurement



- We only estimate $\omega^{\gamma i}$ well with 90% probability.
- Some of the $\log(n/k)$ restrictions will go awry.
- Two options:
  - Median of $O(\log\log(n/k))$ estimates.

Problem: constant failure probability per measurement



- We only estimate $\omega^{\gamma i}$ well with 90% probability.
- Some of the $\log(n/k)$ restrictions will go awry.
- Two options:
  - Median of $O(\log\log(n/k))$ estimates.
  - Can avoid the loss: learn $\log\log(n/k)$ bits at a time.

General k-sparse algorithm

- Shown how to find most heavy hitters.

General k-sparse algorithm

- Shown how to find most heavy hitters.
- Straightforward to estimate their value.

General k-sparse algorithm

- Shown how to find most heavy hitters.
- Straightforward to estimate their value.
- Gives "weak sparse recovery": k-sparse $\widehat{x'}$ such that $\widehat{x - x'}$ is $k/2$-sparse.

General k-sparse algorithm

- Shown how to find most heavy hitters.
- Straightforward to estimate their value.
- Gives "weak sparse recovery": k-sparse $\widehat{x'}$ such that $\widehat{x - x'}$ is $k/2$-sparse.
  - With a little additional noise [Gilbert-Li-Porat-Strauss '10]

General k-sparse algorithm

- Shown how to find most heavy hitters.
- Straightforward to estimate their value.
- Gives "weak sparse recovery": k-sparse $\widehat{x'}$ such that $\widehat{x - x'}$ is $k/2$-sparse.
  - With a little additional noise [Gilbert-Li-Porat-Strauss '10]
- Repeat on $\widehat{x - x'}$, with $k \to k/2$.

General k-sparse algorithm

- Shown how to find most heavy hitters.
- Straightforward to estimate their value.
- Gives "weak sparse recovery": k-sparse $\widehat{x'}$ such that $\widehat{x - x'}$ is $k/2$-sparse.
  - With a little additional noise [Gilbert-Li-Porat-Strauss '10]
- Repeat on $\widehat{x - x'}$, with $k \to k/2$.
- Takes $O((B_i \log n + k) \log(n/B_i))$ time in round i, with B_i buckets.

General k-sparse algorithm

- Shown how to find most heavy hitters.
- Straightforward to estimate their value.
- Gives "weak sparse recovery": k-sparse $\widehat{x'}$ such that $\widehat{x - x'}$ is $k/2$-sparse.
  - With a little additional noise [Gilbert-Li-Porat-Strauss '10]
- Repeat on $\widehat{x - x'}$, with $k \to k/2$.
- Takes $O((B_i \log n + k) \log(n/B_i))$ time in round i, with B_i buckets.
  - Previous recursion: $B_i \asymp k_i \asymp k/2^i$ gives

$$k \log n \log k \gg n \log n$$

General k-sparse algorithm

- Shown how to find most heavy hitters.
- Straightforward to estimate their value.
- Gives "weak sparse recovery": k-sparse $\widehat{x'}$ such that $\widehat{x - x'}$ is $k/2$-sparse.
  - With a little additional noise [Gilbert-Li-Porat-Strauss '10]
- Repeat on $\widehat{x - x'}$, with $k \to k/2$.
- Takes $O((B_i \log n + k) \log(n/B_i))$ time in round i, with B_i buckets.
  - Previous recursion: $B_i \asymp k_i \asymp k/2^i$ gives

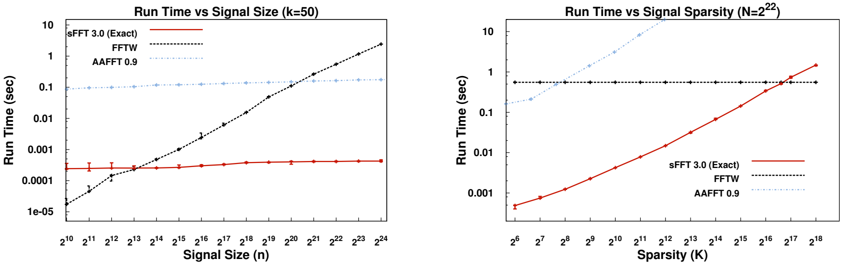
$$k \log n \log k \gg n \log n$$

  - Instead: $B_i \asymp k/i^{\Theta(1)}$, $k_i \asymp k/i!$ gives

$$k \log n \log(n/k)$$

Outline

Empirical performance of exact sparse algorithm

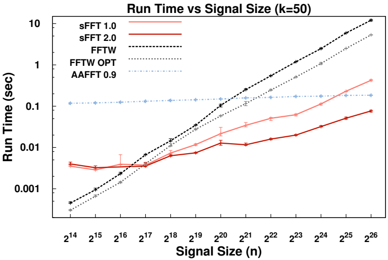


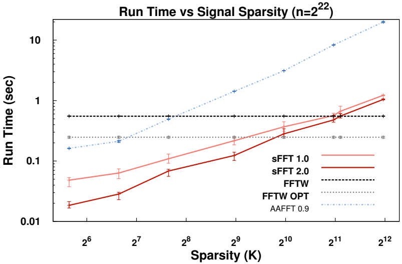
- Compare to FFTW, previous best sublinear algorithm (AAFFT).
- Faster than FFTW for $k/n < 3\%$.

Conclusions and Future Work

- $O(k \log n)$ for exactly sparse $\widehat{x}$
- $O(k \log \frac{n}{k} \log n)$ for approximation.
- Beats FFTW for $k/n < 3\%$ (in the exact case).
- Open problems:
  - Can we get $k \log n$ for approximate recovery?
  - Hadamard matrix / FFT over finite fields?
  - n not a power of 2?
  - Higher probability of success without $\log(1/\delta)$ slowdown?
  - Stronger approximation guarantee, like ℓ_∞/ℓ_2?
  - Better recovery of off-grid frequencies?

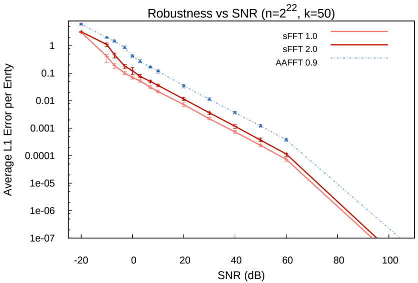
SODA empirical Performance: runtime





- Compare to FFTW, previous best sublinear algorithm (AAFFT).
- Offer a heuristic that improves time to $\widetilde{O}(n^{1/3}k^{2/3})$.
  - Filter from [Mansour '92].
  - Can't rerandomize, might miss elements.
- Faster than FFTW for $n/k > 2{,}000$.
- Faster than AAFFT for $n/k < 1{,}000{,}000$.

SODA empirical Performance: noise



- Just like in Count-Sketch, algorithm is noise tolerant.

Saving a $\log\log(n/k)$ factor

- Could use only $\log(n/k)$ samples by taking random γ:
  - For $\tau' \neq \tau$, $\omega^{\gamma(\tau'-\tau)}$ uniform over circle.
  - Hence $\omega^{\gamma\tau'}$ probably far from the observations.
  - Distinguish among n/k possibilities with $\log(n/k)$ samples.
- Takes $n/k \log(n/k)$ time to test all possibilities.
- Idea: mix the two approaches.
  - Split region into $\log(n/k)$ subregions of size w.
  - Choose random $\gamma \in [\frac{n}{8w}, \frac{n}{4w}]$.
  - Small enough that subregions remain local.
  - Large enough that far subregions roughly uniform.
  - Identify subregion exhaustively: $\log\log(n/k)$ measurements and $\log(n/k)\log\log(n/k)$ time.
  - Repeat $\log_{\log(n/k)}(n/k)$ times to identify τ.
  - Total $\log(n/k)$ measurements, $\log^2(n/k)$ time.