Sparse Recovery and Fourier Sampling

Eric Price

MIT

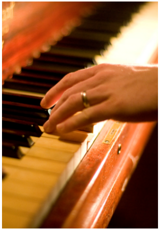
The Fourier Transform

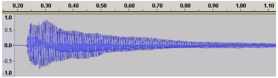
Conversion between time and frequency domains

Time Domain

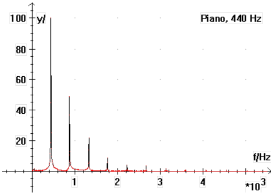
Frequency Domain

Fourier Transform ⟷





Displacement of Air

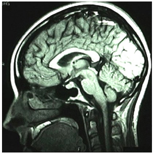


Concert A

The Fourier Transform is Ubiquitous

Audio

Video



Medical Imaging

Radar

GPS

Oil Exploration

Computing the Discrete Fourier Transform

- How to compute $\widehat{x} = Fx$?

Computing the Discrete Fourier Transform

- How to compute $\widehat{x} = Fx$?
- Naive multiplication: $O(n^2)$.

Computing the Discrete Fourier Transform

- How to compute $\widehat{x} = Fx$?
- Naive multiplication: $O(n^2)$.
- Fast Fourier Transform: $O(n \log n)$ time. [Cooley-Tukey, 1965]

Computing the Discrete Fourier Transform

- How to compute $\widehat{x} = Fx$?
- Naive multiplication: $O(n^2)$.
- Fast Fourier Transform: $O(n \log n)$ time. [Cooley-Tukey, 1965]

> *[T]he method greatly reduces the tediousness of mechanical calculations.*
>
> – Carl Friedrich Gauss, 1805

Computing the Discrete Fourier Transform

- How to compute $\widehat{x} = Fx$?
- Naive multiplication: $O(n^2)$.
- Fast Fourier Transform: $O(n \log n)$ time. [Cooley-Tukey, 1965]

> *[T]he method greatly reduces the tediousness of mechanical calculations.*
>
> – Carl Friedrich Gauss, 1805

- By hand: $22n \log n$ seconds. [Danielson-Lanczos, 1942]

Computing the Discrete Fourier Transform

- How to compute $\widehat{x} = Fx$?
- Naive multiplication: $O(n^2)$.
- Fast Fourier Transform: $O(n \log n)$ time. [Cooley-Tukey, 1965]

> *[T]he method greatly reduces the tediousness of mechanical calculations.*
>
> – Carl Friedrich Gauss, 1805

- By hand: $22n \log n$ seconds. [Danielson-Lanczos, 1942]
- Can we do better?

Computing the Discrete Fourier Transform

- How to compute $\widehat{x} = Fx$?
- Naive multiplication: $O(n^2)$.
- Fast Fourier Transform: $O(n \log n)$ time. [Cooley-Tukey, 1965]

> *[T]he method greatly reduces the tediousness of mechanical calculations.*
>
> – Carl Friedrich Gauss, 1805

- By hand: $22n \log n$ seconds. [Danielson-Lanczos, 1942]
- Can we do *much* better?

Computing the Discrete Fourier Transform

- How to compute $\widehat{x} = Fx$?
- Naive multiplication: $O(n^2)$.
- Fast Fourier Transform: $O(n \log n)$ time. [Cooley-Tukey, 1965]

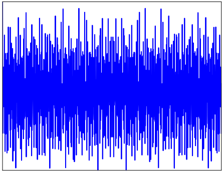
> *[T]he method greatly reduces the tediousness of mechanical calculations.*
>
> – Carl Friedrich Gauss, 1805

- By hand: $22 n \log n$ seconds. [Danielson-Lanczos, 1942]
- Can we do *much* better?

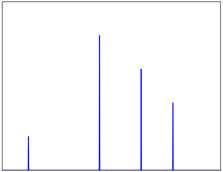
When can we compute the Fourier Transform in *sublinear* time?

Idea: Leverage *Sparsity*

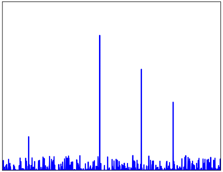
Often the Fourier transform is dominated by a small number of peaks:



Time Signal



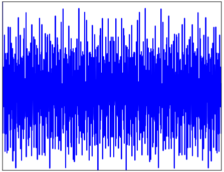
Frequency
(Exactly sparse)



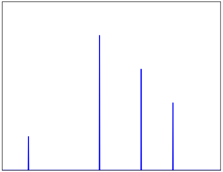
Frequency
(Approximately sparse)

Idea: Leverage *Sparsity*

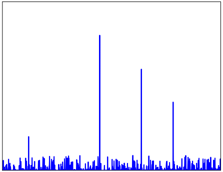
Often the Fourier transform is dominated by a small number of peaks:



Time Signal



Frequency
(Exactly sparse)

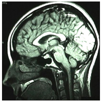


Frequency
(Approximately sparse)

Sparsity is common:

Audio

Video



Medical Imaging

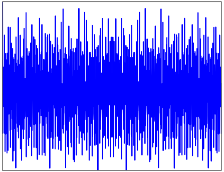
Radar

GPS

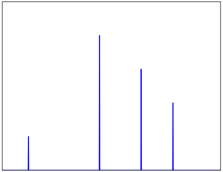
Oil Exploration

Idea: Leverage *Sparsity*

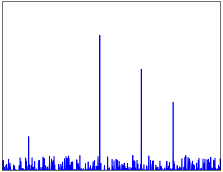
Often the Fourier transform is dominated by a small number of peaks:



Time Signal



Frequency
(Exactly sparse)



Frequency
(Approximately sparse)

Sparsity is common:

Goal of this work: a *sparse* Fourier transform

Faster Fourier Transform on sparse data.

Talk Outline

1. Sparse Fourier Transform
   - Overview
   - Technical Details

Talk Outline

1. Sparse Fourier Transform
   - Overview
   - Technical Details
2. Beyond: Sparse Recovery / Compressive Sensing
   - Overview
   - Adaptivity
   - Conclusion

Talk Outline

1. Sparse Fourier Transform
   - Overview
   - Technical Details

2. Beyond: Sparse Recovery / Compressive Sensing
   - Overview
   - Adaptivity
   - Conclusion

My Contributions

Goal: Compute the Fourier transform $\widehat{x} = Fx$ when $\widehat{x}$ is k-sparse.

- Theory:
  - The fastest algorithm for Fourier transforms of sparse data.
  - The only algorithms faster than FFT for all $k = o(n)$.

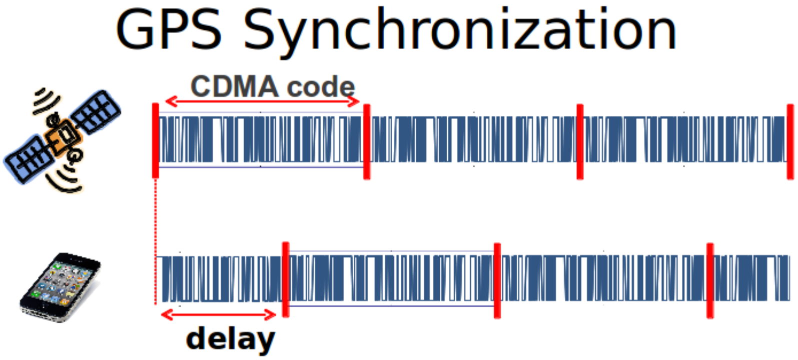
My Contributions

Goal: Compute the Fourier transform $\widehat{x} = Fx$ when $\widehat{x}$ is k-sparse.

- Theory:
  - The fastest algorithm for Fourier transforms of sparse data.
  - The only algorithms faster than FFT for all $k = o(n)$.
- Practice:
  - Implementation is faster than FFTW for a wide range of inputs.
  - Orders of magnitude faster than previous sparse Fourier transforms.
  - Useful in multiple applications.

Applications of ideas

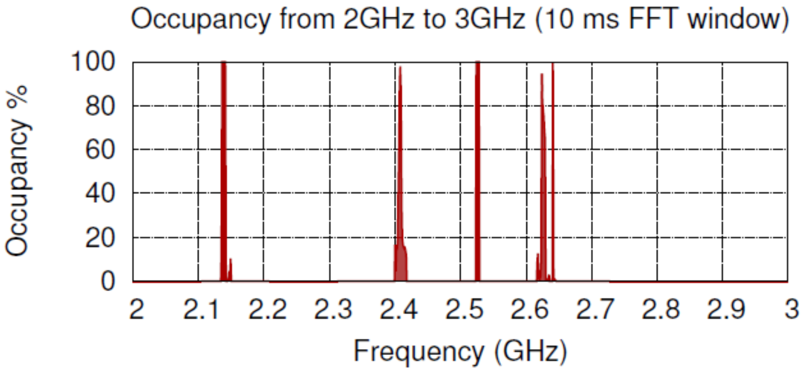
http://groups.csail.mit.edu/netmit/sFFT/workshop.html



- GPS [HAKI]: 2$\times$ faster

Applications of ideas

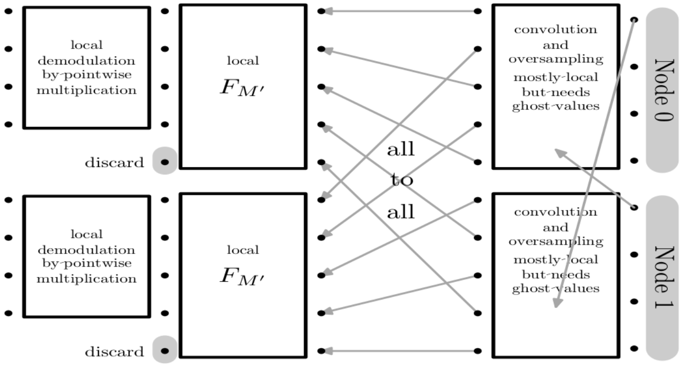
http://groups.csail.mit.edu/netmit/sFFT/workshop.html



- GPS [HAKI]: 2× faster
- Spectrum sensing [HSAHK]: 6× lower sampling rate

Applications of ideas

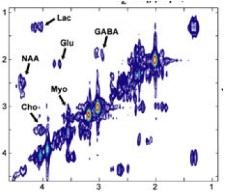
http://groups.csail.mit.edu/netmit/sFFT/workshop.html



- GPS [HAKI]: 2× faster
- Spectrum sensing [HSAHK]: 6× lower sampling rate
- Dense FFT over clusters [TPKP]: 2× faster

Applications of ideas

http://groups.csail.mit.edu/netmit/sFFT/workshop.html



- GPS [HAKI]: 2$\times$ faster
- Spectrum sensing [HSAHK]: 6$\times$ lower sampling rate
- Dense FFT over clusters [TPKP]: 2$\times$ faster
- ...

Talk Outline

1. Sparse Fourier Transform
   - Overview
   - Technical Details
2. Beyond: Sparse Recovery / Compressive Sensing
   - Overview
   - Adaptivity
   - Conclusion

Theoretical Results

For a signal of size *n* with *k* large frequencies

- First on Boolean cube [GL89, KM92, L93]

Theoretical Results

For a signal of size n with k large frequencies

- First on Boolean cube [GL89, KM92, L93]
- Adapted to complexes [Mansour '92, GGIMS02, AGS03, GMS05, Iwen '10, Akavia '10]

Theoretical Results

For a signal of size n with k large frequencies

- First on Boolean cube [GL89, KM92, L93]
- Adapted to complexes [Mansour '92, GGIMS02, AGS03, GMS05, Iwen '10, Akavia '10]
  - All take at least $k \log^4 n$ time.
  - Only better than FFT if $k \ll n / \log^3 n$.

Theoretical Results

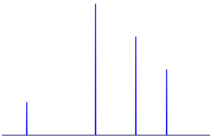
For a signal of size n with k large frequencies

- First on Boolean cube [GL89, KM92, L93]
- Adapted to complexes [Mansour '92, GGIMS02, AGS03, GMS05, Iwen '10, Akavia '10]
    - All take at least $k \log^4 n$ time.
    - Only better than FFT if $k \ll n / \log^3 n$.
- Our results [HIKP12a, HIKP12b]

Theoretical Results

For a signal of size n with k large frequencies

- First on Boolean cube [GL89, KM92, L93]
- Adapted to complexes [Mansour '92, GGIMS02, AGS03, GMS05, Iwen '10, Akavia '10]
  - All take at least $k \log^4 n$ time.
  - Only better than FFT if $k \ll n/\log^3 n$.

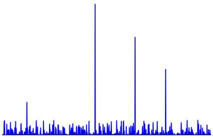


- Our results [HIKP12a, HIKP12b]
  - Exactly k-sparse: $O(k \log n)$
    - Optimal if FFT is optimal.

Theoretical Results

For a signal of size n with k large frequencies

- First on Boolean cube [GL89, KM92, L93]
- Adapted to complexes [Mansour '92, GGIMS02, AGS03, GMS05, Iwen '10, Akavia '10]
  - All take at least $k \log^4 n$ time.
  - Only better than FFT if $k \ll n/\log^3 n$.



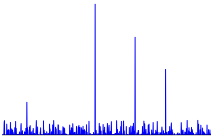
- Our results [HIKP12a, HIKP12b]
  - Exactly k-sparse: $O(k \log n)$
    - ★ Optimal if FFT is optimal.
  - Approximately k-sparse: $O(k \log(n/k) \log n)$

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

Theoretical Results

For a signal of size n with k large frequencies

- First on Boolean cube [GL89, KM92, L93]
- Adapted to complexes [Mansour '92, GGIMS02, AGS03, GMS05, Iwen '10, Akavia '10]
  - All take at least $k \log^4 n$ time.
  - Only better than FFT if $k \ll n/\log^3 n$.



- Our results [HIKP12a, HIKP12b]
  - Exactly k-sparse: $O(k \log n)$
    - Optimal if FFT is optimal.
  - Approximately k-sparse: $O(k \log(n/k) \log n)$

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

  - Better than FFT for any $k = o(n)$

Discrete Fourier Transform (DFT) Definition

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

$$\widehat{x}_i = \frac{1}{n}\sum_j \omega^{-ij} x_j \quad \text{for} \quad \omega = e^{\tau \mathbf{i}/n}$$

Discrete Fourier Transform (DFT) Definition

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

$$\widehat{x}_i = \frac{1}{n} \sum_j \omega^{-ij} x_j \quad \text{for} \quad \omega = e^{\tau \mathbf{i}/n}$$

(where τ is the circle constant 6.283...)

Discrete Fourier Transform (DFT) Definition

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

$$\widehat{x}_i = \frac{1}{n} \sum_j \omega^{-ij} x_j \quad \text{for} \quad \omega = e^{\tau \mathbf{i}/n}$$

$$\widehat{x} = F x \quad \text{for} \quad F_{ij} = \omega^{-ij}/n$$

(where τ is the circle constant 6.283...)

Discrete Fourier Transform (DFT) Definition

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

$$\widehat{x}_i = \frac{1}{n}\sum_j \omega^{-ij} x_j \quad \text{for} \quad \omega = e^{\tau \mathbf{i}/n}$$

$$\widehat{x} = F x \quad \text{for} \quad F_{ij} = \omega^{-ij}/n$$

- Inverse transform almost identical:

(where τ is the circle constant 6.283...)

Discrete Fourier Transform (DFT) Definition

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

$$\widehat{x}_i = \frac{1}{n} \sum_j \omega^{-ij} x_j \quad \text{for} \quad \omega = e^{\tau \mathbf{i}/n}$$

$$\widehat{x} = F x \quad \text{for} \quad F_{ij} = \omega^{-ij}/n$$

- Inverse transform almost identical:

$$x_i = \sum_j \omega^{ij} \widehat{x}_j$$

  - $\omega \to \omega^{-1}$, scale

(where τ is the circle constant 6.283...)

Discrete Fourier Transform (DFT) Definition

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

$$\widehat{x}_i = \frac{1}{n}\sum_j \omega^{-ij} x_j \quad \text{for} \quad \omega = e^{\tau \mathbf{i}/n}$$

$$\widehat{x} = F x \quad \text{for} \quad F_{ij} = \omega^{-ij}/n$$

- Inverse transform almost identical:

$$x_i = \sum_j \omega^{ij} \widehat{x}_j$$

  - $\omega \to \omega^{-1}$, scale
- Lots of nice properties

(where τ is the circle constant 6.283...)

Discrete Fourier Transform (DFT) Definition

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

$$\widehat{x}_i = \frac{1}{n} \sum_j \omega^{-ij} x_j \quad \text{for} \quad \omega = e^{\tau \mathbf{i}/n}$$

$$\widehat{x} = F x \quad \text{for} \quad F_{ij} = \omega^{-ij}/n$$

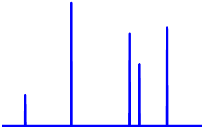
- Inverse transform almost identical:

$$x_i = \sum_j \omega^{ij} \widehat{x}_j$$

  - $\omega \to \omega^{-1}$, scale
- Lots of nice properties
  - Convolution $\longleftrightarrow$ Multiplication

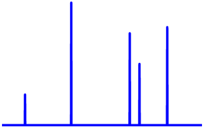
(where τ is the circle constant 6.283...)

Algorithm



Simpler case: $\widehat{x}$ is *exactly* k-sparse.

Algorithm

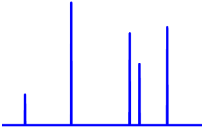


Simpler case: $\widehat{x}$ is *exactly* k-sparse.

Theorem

We can compute $\widehat{x}$ *in* $O(k \log n)$ *expected time.*

Algorithm



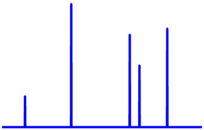
Simpler case: $\widehat{x}$ is *exactly* k-sparse.

Theorem

We can compute $\widehat{x}$ *in* $O(k \log n)$ *expected time.*

Still kind of hard.

Algorithm

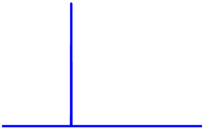


Simpler case: $\widehat{x}$ is *exactly* k-sparse.

Theorem

We can compute $\widehat{x}$ in $O(k \log n)$ expected time.

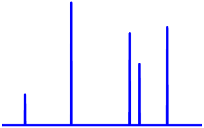
Still kind of hard.



Simplest case: $\widehat{x}$ is exactly 1-sparse.

Algorithm

Simpler case: $\widehat{x}$ is *exactly* k-sparse.

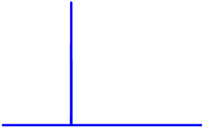


Theorem

We can compute $\widehat{x}$ in $O(k \log n)$ expected time.

Still kind of hard.

Simplest case: $\widehat{x}$ is exactly 1-sparse.



Lemma

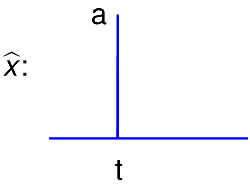
We can compute a 1-sparse $\widehat{x}$ in $O(1)$ time.

Algorithm for $k = 1$

Lemma

We can compute a 1-sparse $\widehat{x}$ in $O(1)$ time.

$$\widehat{x}_i = \begin{cases} a & \text{if } i = t \\ 0 & \text{otherwise} \end{cases}$$

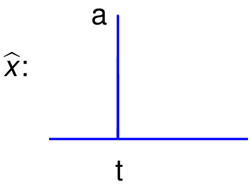


Algorithm for $k = 1$

Lemma

We can compute a 1-sparse $\widehat{x}$ in $O(1)$ time.

$$\widehat{x}_i = \begin{cases} a & \text{if } i = t \\ 0 & \text{otherwise} \end{cases}$$



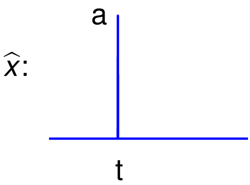
- Then $x = (a, a\omega^t, a\omega^{2t}, a\omega^{3t}, \ldots, a\omega^{(n-1)t})$.

Algorithm for $k = 1$

Lemma

We can compute a 1-sparse $\widehat{x}$ in $O(1)$ time.

$$\widehat{x}_i = \begin{cases} a & \text{if } i = t \\ 0 & \text{otherwise} \end{cases}$$



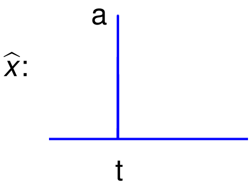
- Then $x = (a, a\omega^t, a\omega^{2t}, a\omega^{3t}, \ldots, a\omega^{(n-1)t})$.

$$x_0 = a$$

Algorithm for $k = 1$

Lemma

We can compute a 1-sparse $\widehat{x}$ in $O(1)$ time.



$$\widehat{x}_i = \begin{cases} a & \text{if } i = t \\ 0 & \text{otherwise} \end{cases}$$

- Then $x = (a, a\omega^t, a\omega^{2t}, a\omega^{3t}, \ldots, a\omega^{(n-1)t})$.

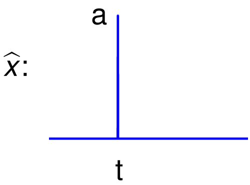
$$x_0 = a \qquad\qquad x_1 = a\omega^t$$

Algorithm for $k = 1$

Lemma

We can compute a 1-sparse $\widehat{x}$ in $O(1)$ time.

$$\widehat{x}_i = \begin{cases} a & \text{if } i = t \\ 0 & \text{otherwise} \end{cases}$$



- Then $x = (a, a\omega^t, a\omega^{2t}, a\omega^{3t}, \ldots, a\omega^{(n-1)t})$.

$$x_0 = a \qquad\qquad x_1 = a\omega^t$$

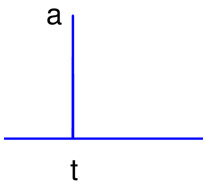
- $x_1/x_0 = \omega^t \implies t$.

Algorithm for $k = 1$

Lemma

We can compute a 1-sparse $\widehat{x}$ in $O(1)$ time.

$$\widehat{x}_i = \begin{cases} a & \text{if } i = t \\ 0 & \text{otherwise} \end{cases}$$

$\widehat{x}$: 

- Then $x = (a, a\omega^t, a\omega^{2t}, a\omega^{3t}, \ldots, a\omega^{(n-1)t})$.

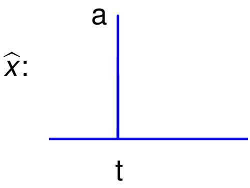
$$x_0 = a \qquad\qquad x_1 = a\omega^t$$

- $x_1/x_0 = \omega^t \implies t$. $\blacksquare$

Algorithm for $k = 1$

Lemma

We can compute a 1-sparse $\widehat{x}$ in $O(1)$ time.



$$\widehat{x}_i = \begin{cases} a & \text{if } i = t \\ 0 & \text{otherwise} \end{cases}$$

- Then $x = (a, a\omega^t, a\omega^{2t}, a\omega^{3t}, \ldots, a\omega^{(n-1)t})$.

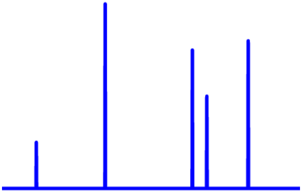
$$x_0 = a \qquad\qquad x_1 = a\omega^t$$

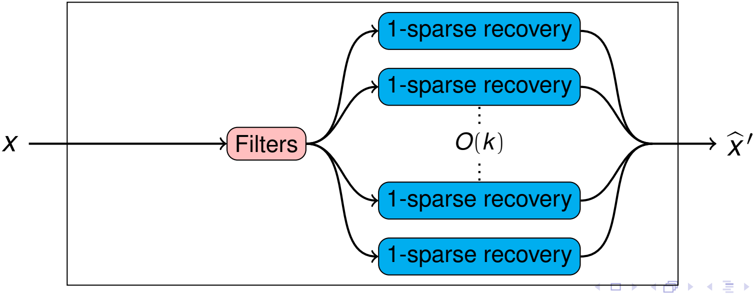
- $x_1 / x_0 = \omega^t \implies t$. ■

- (Related to OFDM, Prony's method, matrix pencil.)

Algorithm for general k

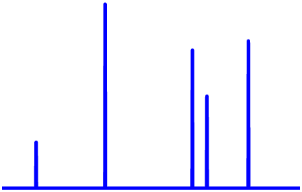
- Reduce general k to $k = 1$.

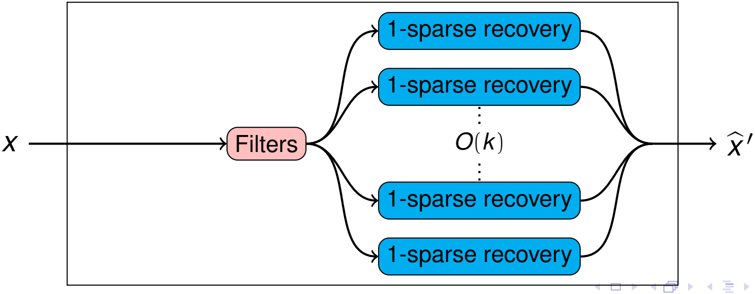




Algorithm for general k

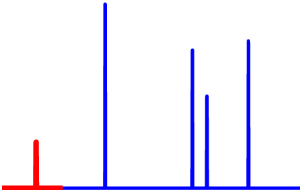
- Reduce general k to $k = 1$.
- "Filters": partition frequencies into $O(k)$ buckets.

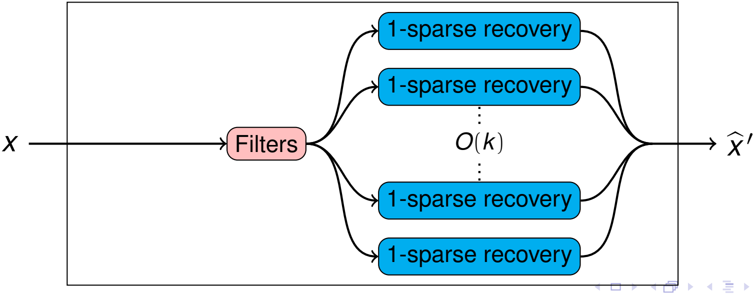




Algorithm for general k

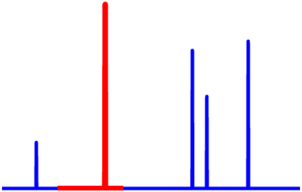
- Reduce general k to $k = 1$.
- "Filters": partition frequencies into $O(k)$ buckets.

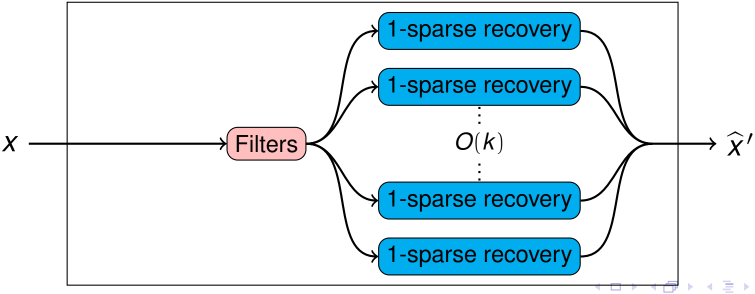




Algorithm for general k

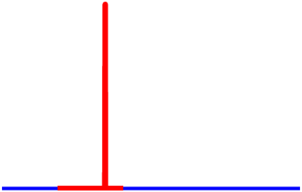
- Reduce general k to $k = 1$.
- "Filters": partition frequencies into $O(k)$ buckets.

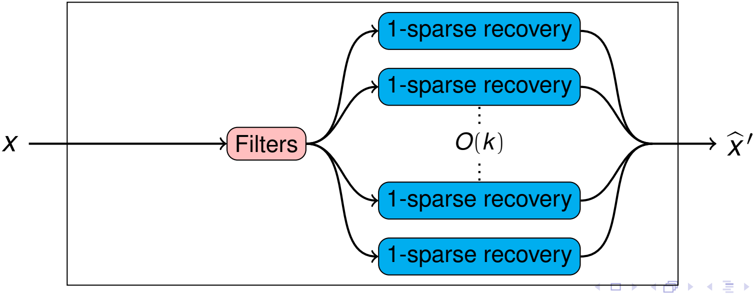




Algorithm for general k

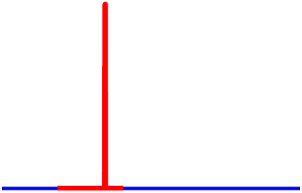
- Reduce general k to $k = 1$.
- "Filters": partition frequencies into $O(k)$ buckets.
  - Sample from time domain of each bucket with $O(\log n)$ overhead.

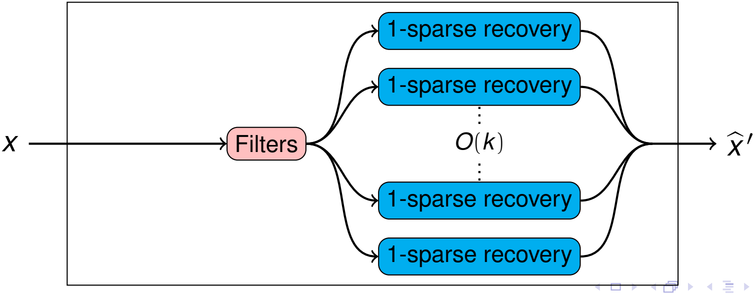




Algorithm for general k

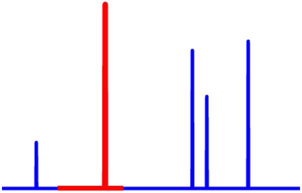
- Reduce general k to $k = 1$.
- "Filters": partition frequencies into $O(k)$ buckets.
  - Sample from time domain of each bucket with $O(\log n)$ overhead.
  - Recovered by $k = 1$ algorithm

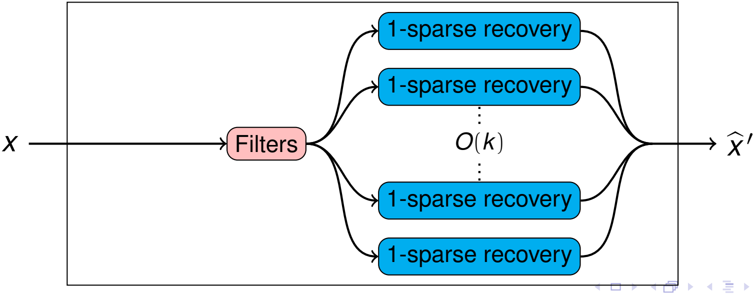




Algorithm for general k

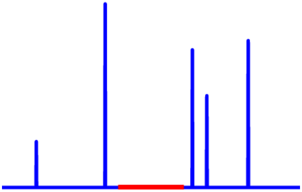
- Reduce general k to $k = 1$.
- "Filters": partition frequencies into $O(k)$ buckets.
  - Sample from time domain of each bucket with $O(\log n)$ overhead.
  - Recovered by $k = 1$ algorithm
- Most frequencies alone in bucket.

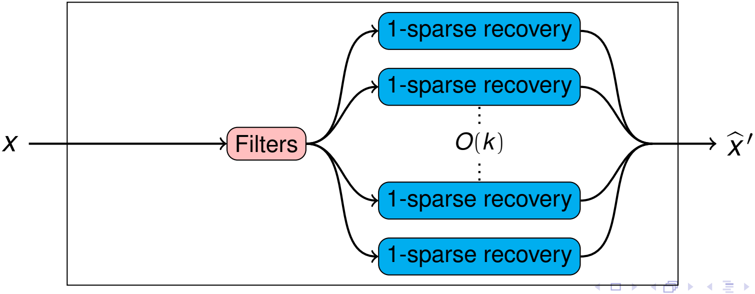




Algorithm for general k

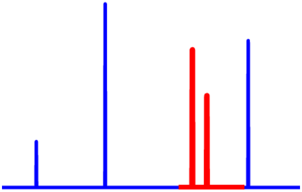
- Reduce general k to $k = 1$.
- "Filters": partition frequencies into $O(k)$ buckets.
  - Sample from time domain of each bucket with $O(\log n)$ overhead.
  - Recovered by $k = 1$ algorithm
- Most frequencies alone in bucket.

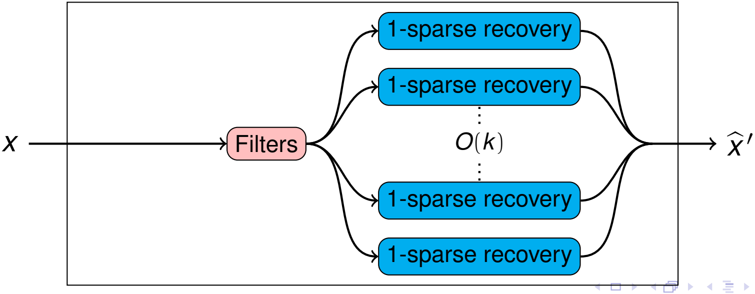




Algorithm for general k

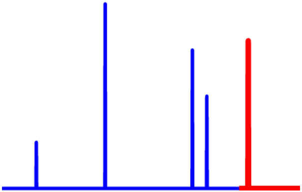
- Reduce general k to $k = 1$.
- "Filters": partition frequencies into $O(k)$ buckets.
  - Sample from time domain of each bucket with $O(\log n)$ overhead.
  - Recovered by $k = 1$ algorithm
- Most frequencies alone in bucket.

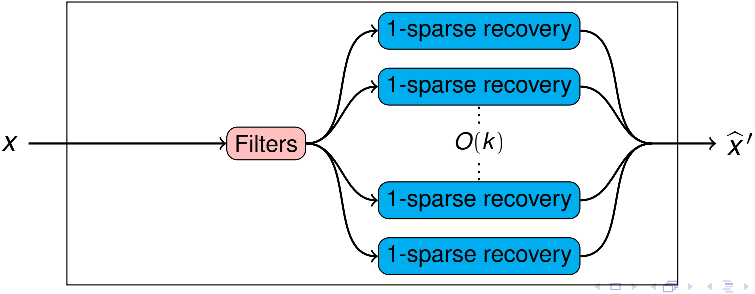




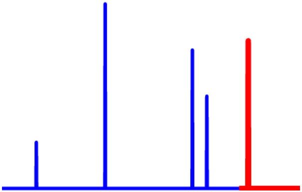
Algorithm for general k

- Reduce general k to $k = 1$.
- "Filters": partition frequencies into $O(k)$ buckets.
  - Sample from time domain of each bucket with $O(\log n)$ overhead.
  - Recovered by $k = 1$ algorithm
- Most frequencies alone in bucket.

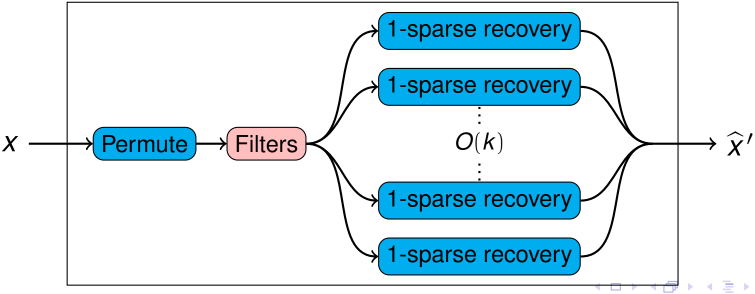




Algorithm for general k

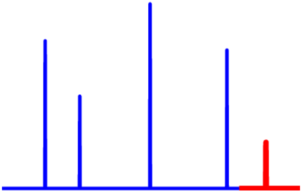


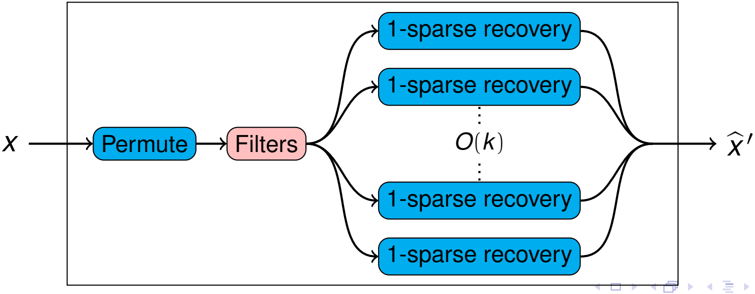
- Reduce general k to $k = 1$.
- "Filters": partition frequencies into $O(k)$ buckets.
  - Sample from time domain of each bucket with $O(\log n)$ overhead.
  - Recovered by $k = 1$ algorithm
- Most frequencies alone in bucket.
- Random permutation



Algorithm for general k

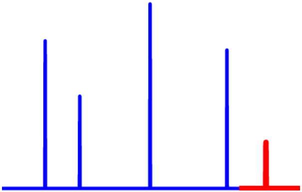
- Reduce general k to $k = 1$.
- "Filters": partition frequencies into $O(k)$ buckets.
  - Sample from time domain of each bucket with $O(\log n)$ overhead.
  - Recovered by $k = 1$ algorithm
- Most frequencies alone in bucket.
- Random permutation

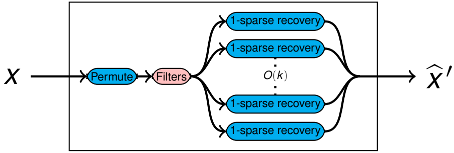




Algorithm for general k

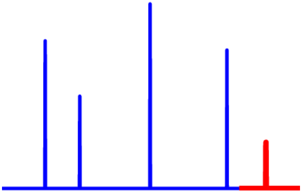
- Reduce general k to $k = 1$.
- "Filters": partition frequencies into $O(k)$ buckets.
  - Sample from time domain of each bucket with $O(\log n)$ overhead.
  - Recovered by $k = 1$ algorithm
- Most frequencies alone in bucket.
- Random permutation

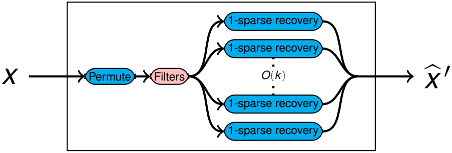




Algorithm for general k

- Reduce general k to $k = 1$.
- "Filters": partition frequencies into $O(k)$ buckets.
  - Sample from time domain of each bucket with $O(\log n)$ overhead.
  - Recovered by $k = 1$ algorithm
- Most frequencies alone in bucket.
- Random permutation



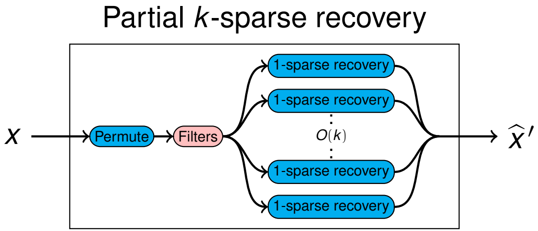


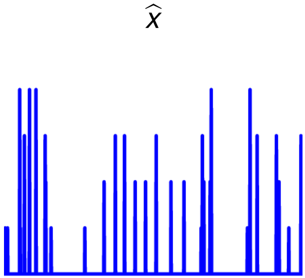
Recovers *most* of $\widehat{x}$:

Lemma (Partial sparse recovery)

In $O(k \log n)$ expected time, we can compute an estimate $\widehat{x}'$ such that $\widehat{x} - \widehat{x}'$ is $k/2$-sparse.

Overall outline

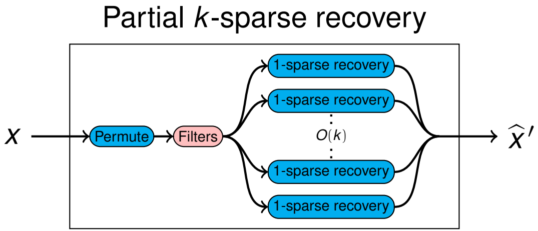


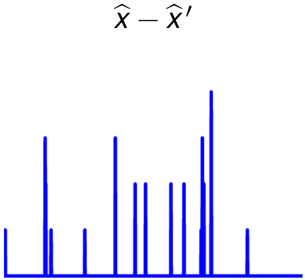


Lemma (Partial sparse recovery)

In $O(k \log n)$ expected time, we can compute an estimate $\widehat{x}'$ such that $\widehat{x} - \widehat{x}'$ is $k/2$-sparse.

Overall outline





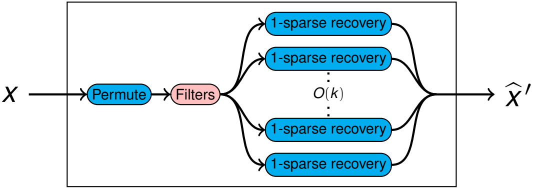
Lemma (Partial sparse recovery)

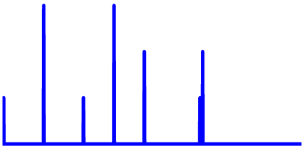
In $O(k \log n)$ expected time, we can compute an estimate $\widehat{x}'$ such that $\widehat{x} - \widehat{x}'$ is $k/2$-sparse.

Overall outline

$\widehat{x} - \widehat{x}'$

Partial k-sparse recovery





Lemma (Partial sparse recovery)

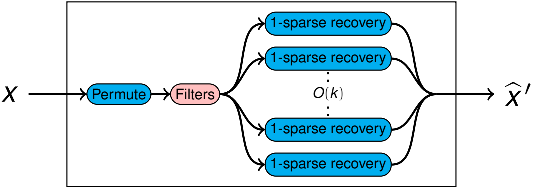
In $O(k \log n)$ expected time, we can compute an estimate $\widehat{x}'$ such that $\widehat{x} - \widehat{x}'$ is $k/2$-sparse.

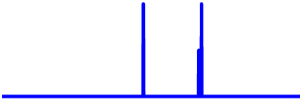
Repeat, $k \to k/2 \to k/4 \to \cdots$

Overall outline

$\widehat{x} - \widehat{x}'$

Partial k-sparse recovery





Lemma (Partial sparse recovery)

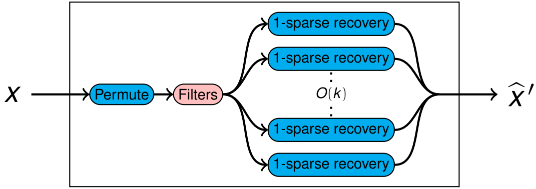
In $O(k \log n)$ expected time, we can compute an estimate $\widehat{x}'$ such that $\widehat{x} - \widehat{x}'$ is $k/2$-sparse.

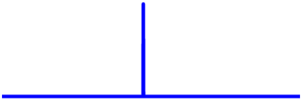
Repeat, $k \to k/2 \to k/4 \to \cdots$

Overall outline

$\widehat{x} - \widehat{x}'$

Partial k-sparse recovery





Lemma (Partial sparse recovery)

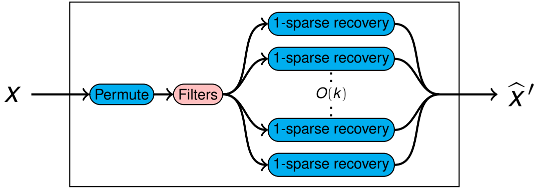
In $O(k \log n)$ expected time, we can compute an estimate $\widehat{x}'$ such that $\widehat{x} - \widehat{x}'$ is $k/2$-sparse.

Repeat, $k \to k/2 \to k/4 \to \cdots$

Overall outline

$\widehat{x} - \widehat{x}'$

Partial k-sparse recovery



Lemma (Partial sparse recovery)

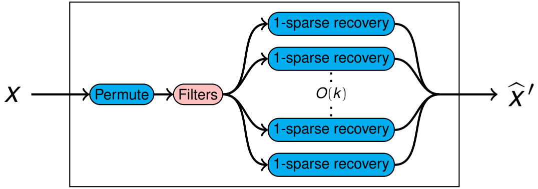
In $O(k \log n)$ expected time, we can compute an estimate $\widehat{x}'$ such that $\widehat{x} - \widehat{x}'$ is $k/2$-sparse.

Repeat, $k \to k/2 \to k/4 \to \cdots$

Overall outline

$\widehat{x} - \widehat{x}'$

Partial k-sparse recovery



Lemma (Partial sparse recovery)

In $O(k \log n)$ expected time, we can compute an estimate $\widehat{x}'$ such that $\widehat{x} - \widehat{x}'$ is $k/2$-sparse.

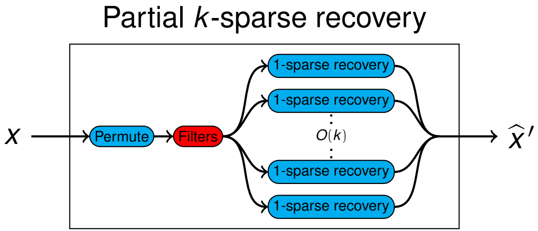
Repeat, $k \to k/2 \to k/4 \to \cdots$

Theorem

We can compute $\widehat{x}$ in $O(k \log n)$ expected time.

Overall outline

$\widehat{x} - \widehat{x}'$



Lemma (Partial sparse recovery)

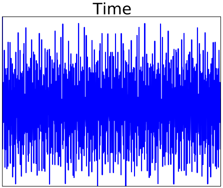
In $O(k \log n)$ expected time, we can compute an estimate $\widehat{x}'$ such that $\widehat{x} - \widehat{x}'$ is $k/2$-sparse.

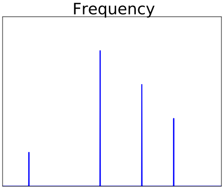
Repeat, $k \to k/2 \to k/4 \to \cdots$

Theorem

We can compute $\widehat{x}$ in $O(k \log n)$ expected time.

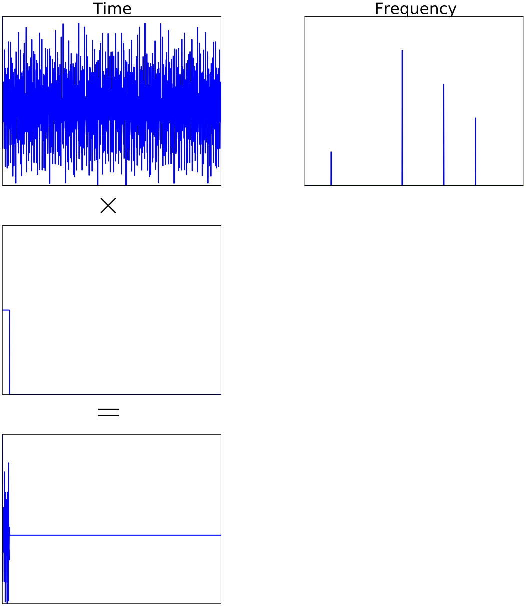
How can you isolate frequencies?





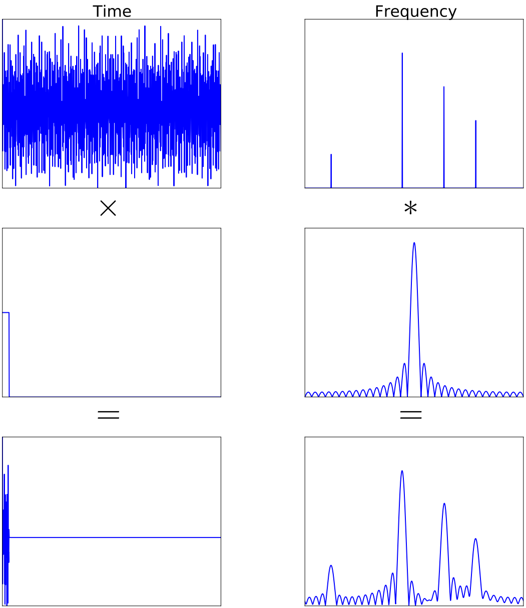
n-dimensional DFT:
$O(n \log n)$
$x \to \widehat{x}$

How can you isolate frequencies?



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

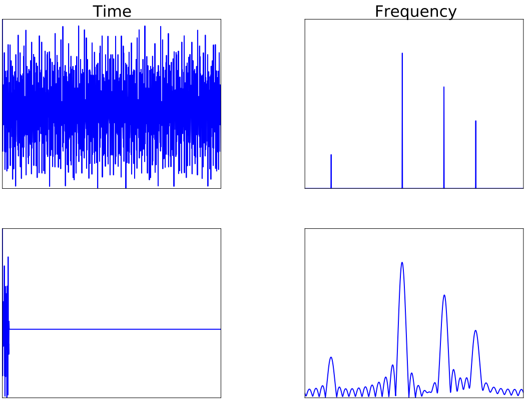
How can you isolate frequencies?



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

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

How can you isolate frequencies?



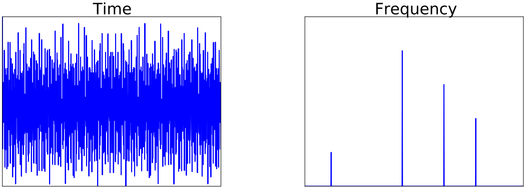
n-dimensional DFT: $O(n \log n)$

$x \rightarrow \widehat{x}$

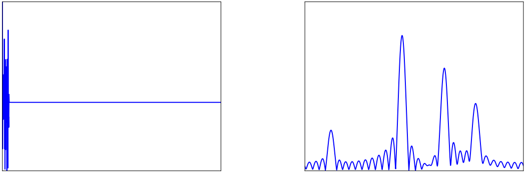
n-dimensional DFT of first k terms: $O(n \log n)$

$x \cdot \text{rect} \rightarrow \widehat{x} * \text{sinc}$.

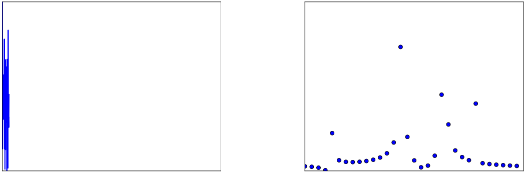
How can you isolate frequencies?



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

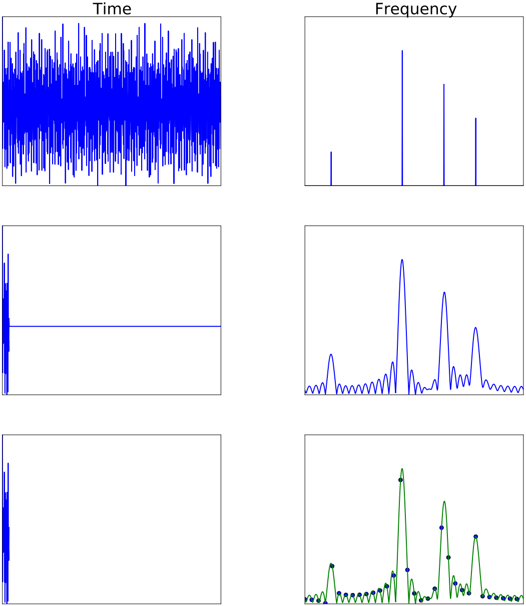


n-dimensional DFT of first k terms: $O(n \log n)$
$x \cdot \text{rect} \to \widehat{x} * \text{sinc}$.



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

How can you isolate frequencies?



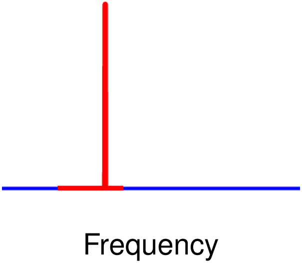
n-dimensional DFT: $O(n \log n)$
$x \to \widehat{x}$

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

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

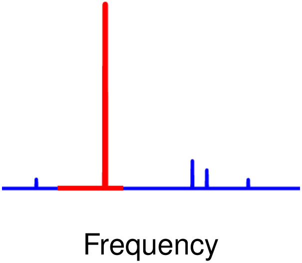
The issue

We want to isolate frequencies.

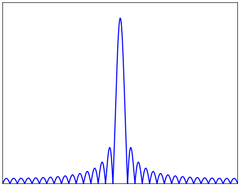


The issue

We want to isolate frequencies.

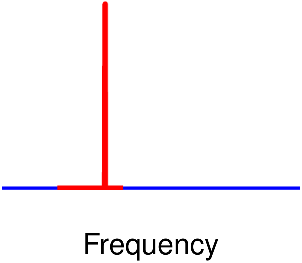


The sinc filter “leaks”.
Contamination from other buckets.

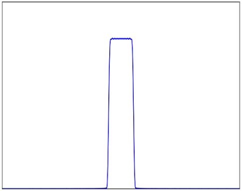


The issue

We want to isolate frequencies.



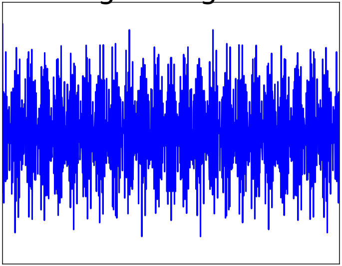
The sinc filter “leaks”.
Contamination from other buckets.



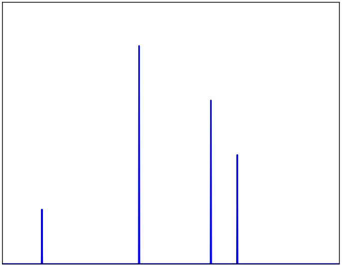
We introduce a better filter:
(Gaussian / prolate spheroidal sequence) convolved with rectangle.

Algorithm for *exactly sparse* signals

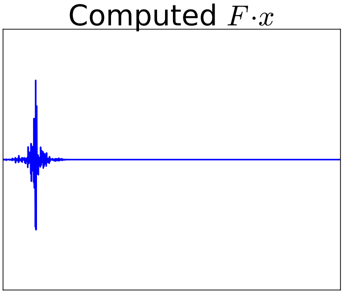
Original signal x

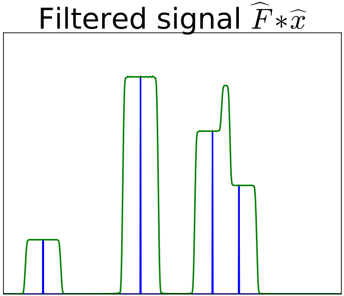


Goal $\hat{x}$



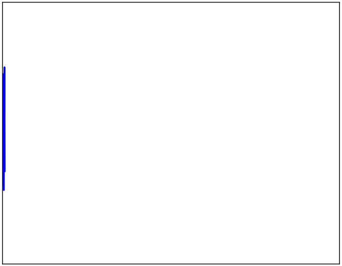
Algorithm for *exactly sparse* signals



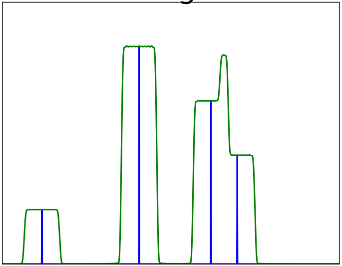


Algorithm for *exactly sparse* signals

$F \cdot x$ aliased to k terms

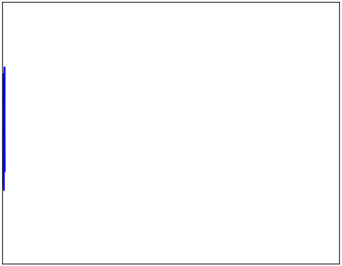


Filtered signal $\widehat{F} * \widehat{x}$

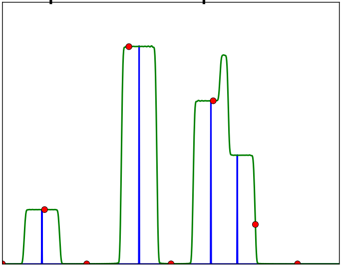


Algorithm for *exactly sparse* signals

$F \cdot x$ aliased to k terms

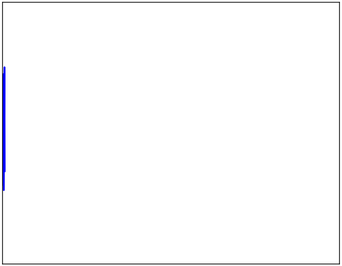


Computed samples of $\widehat{F} * \widehat{x}$

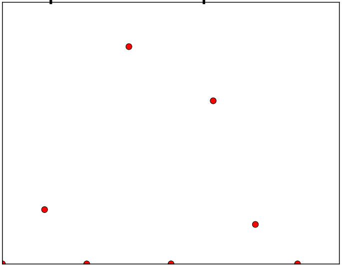


Algorithm for *exactly sparse* signals

$F \cdot x$ aliased to k terms

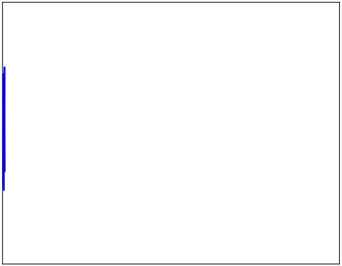


Computed samples of $\widehat{F} * \widehat{x}$

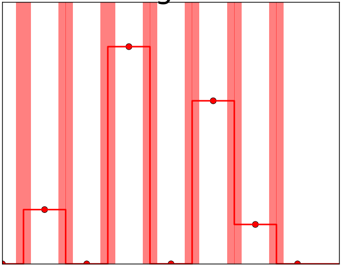


Algorithm for *exactly sparse* signals

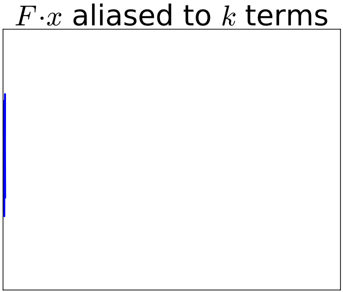
$F \cdot x$ aliased to k terms

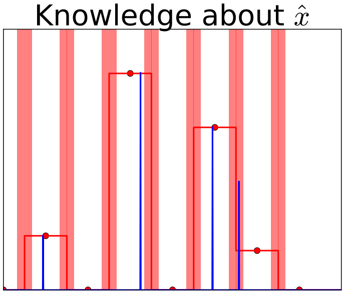


Knowledge about $\hat{x}$

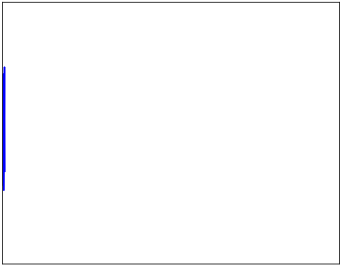


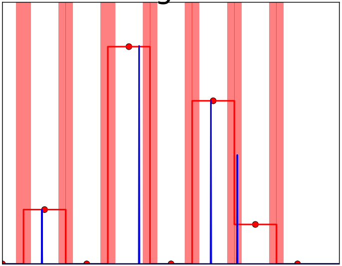
Algorithm for *exactly sparse* signals





Algorithm for *exactly sparse* signals





Lemma

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

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

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

Algorithm for *exactly sparse* signals

Lemma

For most t, the value b we compute for its bucket satisfies

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

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

Algorithm for *exactly sparse* signals

Lemma

For most t, the value b we compute for its bucket satisfies

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

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

- Time-shift x by one and repeat: $b' = \widehat{x}_t \omega^t$.
- Divide to get $b'/b = \omega^t$

Algorithm for *exactly sparse* signals

Lemma

For most t, the value b we compute for its bucket satisfies

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

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

- Time-shift x by one and repeat: $b' = \widehat{x}_t \omega^t$.
- Divide to get $b'/b = \omega^t \implies$ can compute t.

Algorithm for *exactly sparse* signals

Lemma

For most t, the value b we compute for its bucket satisfies

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

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

- Time-shift x by one and repeat: $b' = \widehat{x}_t \omega^t$.
- Divide to get $b'/b = \omega^t \implies$ can compute t.
  - Just like our 1-sparse recovery algorithm, $x_1/x_0 = \omega^t$.

Algorithm for *exactly sparse* signals

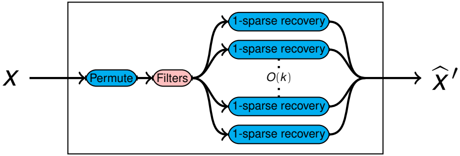
Lemma

For most t, the value b we compute for its bucket satisfies

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

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

- Time-shift x by one and repeat: $b' = \widehat{x}_t \omega^t$.
- Divide to get $b'/b = \omega^t \implies$ can compute t.
  - Just like our 1-sparse recovery algorithm, $x_1/x_0 = \omega^t$.
- Gives partial sparse recovery: $\widehat{x}'$ such that $\widehat{x} - \widehat{x}'$ is $k/2$-sparse.



Algorithm for *exactly sparse* signals

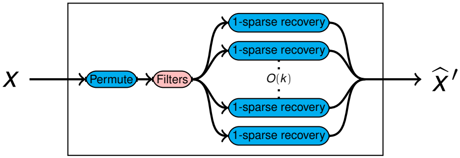
Lemma

For most t, the value b we compute for its bucket satisfies

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

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

- Time-shift x by one and repeat: $b' = \widehat{x}_t \omega^t$.
- Divide to get $b'/b = \omega^t \implies$ can compute t.
  - Just like our 1-sparse recovery algorithm, $x_1/x_0 = \omega^t$.
- Gives partial sparse recovery: $\widehat{x}'$ such that $\widehat{x} - \widehat{x}'$ is $k/2$-sparse.



- Repeat $k \to k/2 \to k/4 \to \cdots$

Algorithm for *exactly sparse* signals

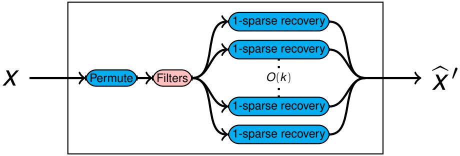
Lemma

For most t, the value b we compute for its bucket satisfies

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

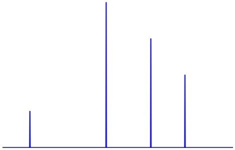
Computing the b for all $O(k)$ buckets takes $O(k \log n)$ time.

- Time-shift x by one and repeat: $b' = \widehat{x}_t \omega^t$.
- Divide to get $b'/b = \omega^t \implies$ can compute t.
  - Just like our 1-sparse recovery algorithm, $x_1/x_0 = \omega^t$.
- Gives partial sparse recovery: $\widehat{x}'$ such that $\widehat{x} - \widehat{x}'$ is $k/2$-sparse.



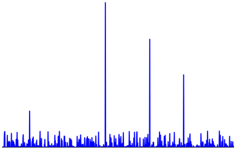
- Repeat $k \to k/2 \to k/4 \to \cdots$
- $O(k \log n)$ time sparse Fourier transform.

Algorithm for *approximately sparse* signals

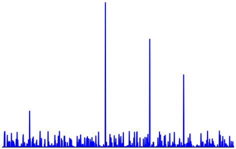


Algorithm for *approximately sparse* signals

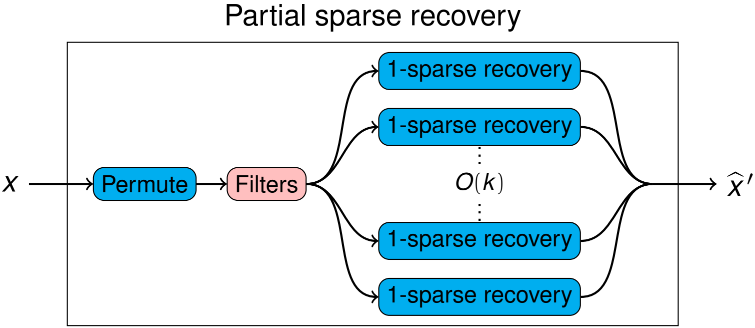
- What changes with noise?



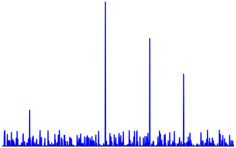
Algorithm for *approximately sparse* signals



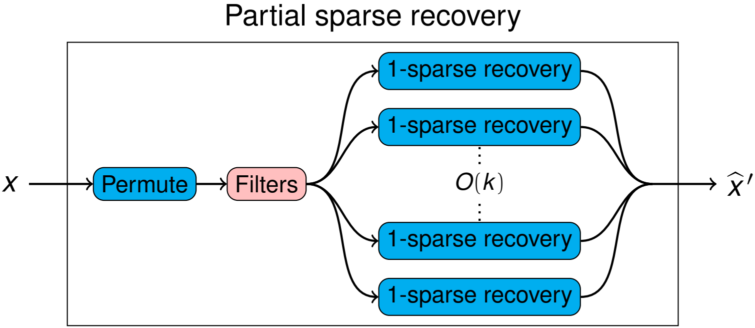
- What changes with noise?
- Identical architecture:



Algorithm for *approximately sparse* signals



- What changes with noise?
- Identical architecture:



- Just requires robust 1-sparse recovery.

Algorithm for *approximately sparse* signals: $k = 1$

Lemma

Suppose $\widehat{x}$ is approximately 1-sparse:

$$|\widehat{x}_t| / \|\widehat{x}\|_2 \geqslant 90\%.$$

Then we can recover it with $O(\log n)$ samples and $O(\log^2 n)$ time.

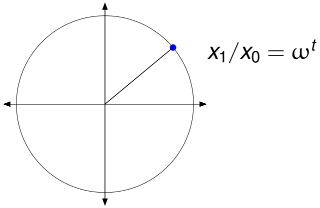
Algorithm for *approximately sparse* signals: $k = 1$

Lemma

Suppose $\widehat{x}$ is approximately 1-sparse:

$$|\widehat{x}_t| / \|\widehat{x}\|_2 \geqslant 90\%.$$

Then we can recover it with $O(\log n)$ samples and $O(\log^2 n)$ time.



- With exact sparsity: $\log n$ bits in a single measurement.

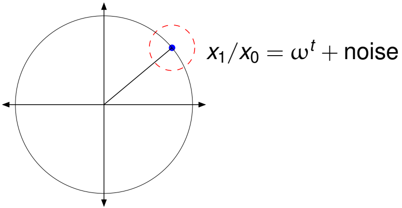
Algorithm for *approximately sparse* signals: $k = 1$

Lemma

Suppose $\widehat{x}$ is approximately 1-sparse:

$$|\widehat{x}_t|/\|\widehat{x}\|_2 \geqslant 90\%.$$

Then we can recover it with $O(\log n)$ samples and $O(\log^2 n)$ time.



- With exact sparsity: $\log n$ bits in a single measurement.
- With noise: only constant number of useful bits.

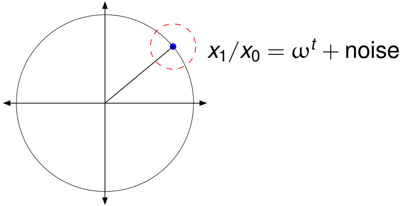
Algorithm for *approximately sparse* signals: $k = 1$

Lemma

Suppose $\widehat{x}$ is approximately 1-sparse:

$$|\widehat{x}_t| / \|\widehat{x}\|_2 \geqslant 90\%.$$

Then we can recover it with $O(\log n)$ samples and $O(\log^2 n)$ time.



- With exact sparsity: $\log n$ bits in a single measurement.
- With noise: only constant number of useful bits.
- Choose $\Theta(\log n)$ time shifts c to recover i.

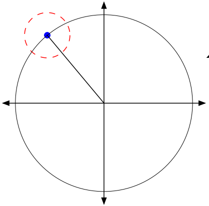
Algorithm for *approximately sparse* signals: $k = 1$

Lemma

Suppose $\widehat{x}$ is approximately 1-sparse:

$$|\widehat{x}_t| / \|\widehat{x}\|_2 \geqslant 90\%.$$

Then we can recover it with $O(\log n)$ samples and $O(\log^2 n)$ time.



$$x_{c_2}/x_0 = \omega^{c_2 t} + \text{noise}$$

- With exact sparsity: $\log n$ bits in a single measurement.
- With noise: only constant number of useful bits.
- Choose $\Theta(\log n)$ time shifts c to recover i.

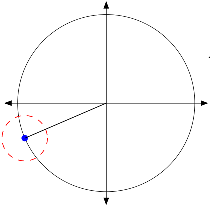
Algorithm for *approximately sparse* signals: $k = 1$

Lemma

Suppose $\widehat{x}$ is approximately 1-sparse:

$$|\widehat{x}_t| / \|\widehat{x}\|_2 \geqslant 90\%.$$

Then we can recover it with $O(\log n)$ samples and $O(\log^2 n)$ time.



- With exact sparsity: $\log n$ bits in a single measurement.
- With noise: only constant number of useful bits.
- Choose $\Theta(\log n)$ time shifts c to recover i.

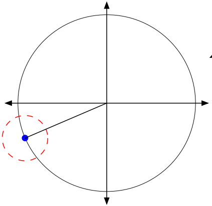
Algorithm for *approximately sparse* signals: $k = 1$

Lemma

Suppose $\widehat{x}$ is approximately 1-sparse:

$$|\widehat{x}_t| / \|\widehat{x}\|_2 \geqslant 90\%.$$

Then we can recover it with $O(\log n)$ samples and $O(\log^2 n)$ time.



$x_{c_3}/x_0 = \omega^{c_3 t} + \text{noise}$

- With exact sparsity: $\log n$ bits in a single measurement.
- With noise: only constant number of useful bits.
- Choose $\Theta(\log n)$ time shifts c to recover i.
- Error correcting code with efficient recovery $\implies$ Lemma. ∎

Algorithm for *approximately sparse* signals: general k

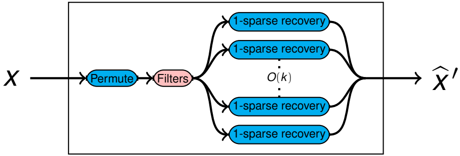
Lemma

If $\widehat{x}$ is approximately 1*-sparse, we can recover it with* $O(\log n)$ *samples and* $O(\log^2 n)$ *time.*

Algorithm for *approximately sparse* signals: general k

Lemma

If $\widehat{x}$ is approximately 1-sparse, we can recover it with $O(\log n)$ samples and $O(\log^2 n)$ time.

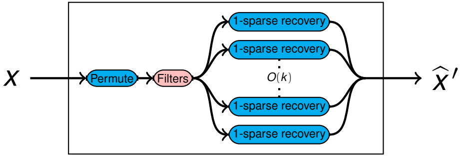


Reduce k-sparse to 1-sparse on buckets of size n/k, with $\log n$ overhead per sample.

Algorithm for *approximately sparse* signals: general k

Lemma

If $\widehat{x}$ is approximately 1-sparse, we can recover it with $O(\log n)$ samples and $O(\log^2 n)$ time.



Reduce k-sparse to 1-sparse on buckets of size n/k, with $\log n$ overhead per sample.

Theorem

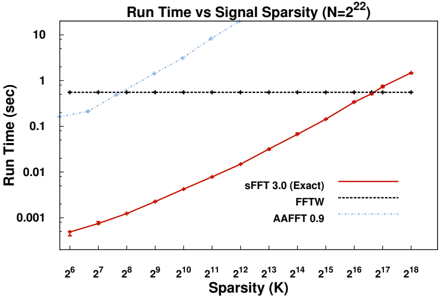
If $\widehat{x}$ is approximately k-sparse, we can recover it in $O(k \log(n/k) \log n)$ time.

Empirical performance

- Compare to
  - FFTW, the "Fastest Fourier Transform in the West"
  - AAFFT, the [GMS05] sparse Fourier transform.

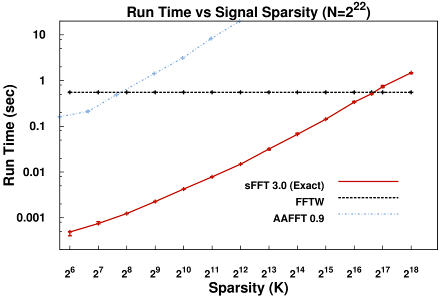
Empirical performance

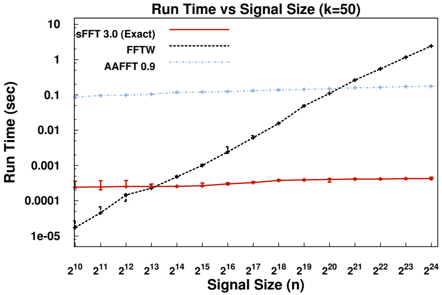
- Compare to
  - FFTW, the "Fastest Fourier Transform in the West"
  - AAFFT, the [GMS05] sparse Fourier transform.



Empirical performance

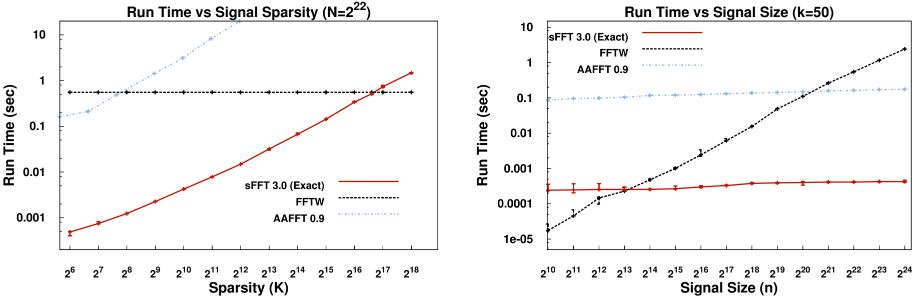
- Compare to
  - FFTW, the “Fastest Fourier Transform in the West”
  - AAFFT, the [GMS05] sparse Fourier transform.





Empirical performance

- Compare to
  - FFTW, the “Fastest Fourier Transform in the West”
  - AAFFT, the [GMS05] sparse Fourier transform.



- Faster than FFTW for wide range of values.

Recap of Sparse Fourier Transform

- Theory:
  - The fastest algorithm for Fourier transforms of sparse data.
  - The only algorithms faster than FFT for all $k = o(n)$.

Recap of Sparse Fourier Transform

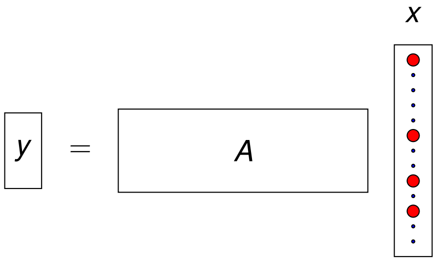
- Theory:
  - The fastest algorithm for Fourier transforms of sparse data.
  - The only algorithms faster than FFT for all $k = o(n)$.
- Practice:
  - Implementation is faster than FFTW for a wide range of inputs.
  - Orders of magnitude faster than previous sparse Fourier transforms.
  - Useful in multiple applications.

Talk Outline

1. Sparse Fourier Transform
   - Overview
   - Technical Details
2. Beyond: Sparse Recovery / Compressive Sensing
   - Overview
   - Adaptivity
   - Conclusion

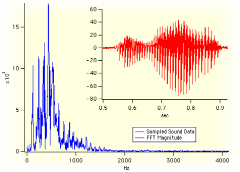
Sparse Recovery / Compressive Sensing

Robustly recover sparse x from linear measurements $y = Ax$.

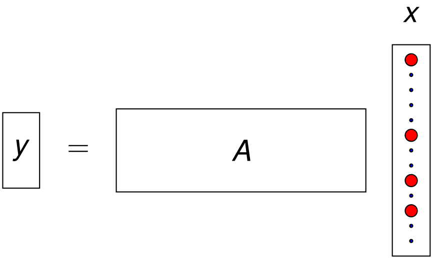


Sparse Recovery / Compressive Sensing

Robustly recover sparse x from linear measurements $y = Ax$.

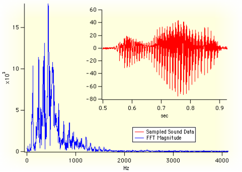


Sparse Fourier



Sparse Recovery / Compressive Sensing

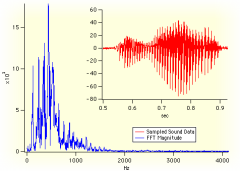
Robustly recover sparse x from linear measurements $y = Ax$.



Sparse Fourier

Sparse Recovery / Compressive Sensing

Robustly recover sparse x from linear measurements $y = Ax$.

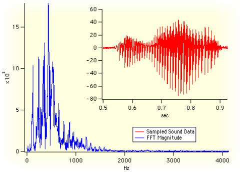


Sparse Fourier

MRI

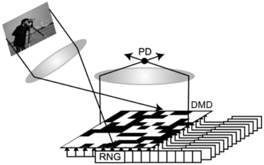
Sparse Recovery / Compressive Sensing

Robustly recover sparse x from linear measurements $y = Ax$.



Sparse Fourier

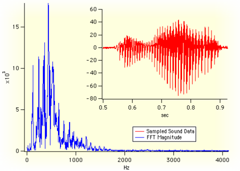
MRI



Single-Pixel Camera

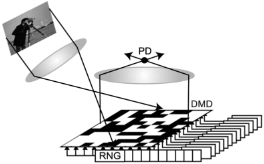
Sparse Recovery / Compressive Sensing

Robustly recover sparse x from linear measurements $y = Ax$.



Sparse Fourier

MRI



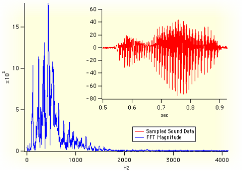
Single-Pixel Camera

Streaming Algorithms

$A(x + \Delta) = Ax + A\Delta$

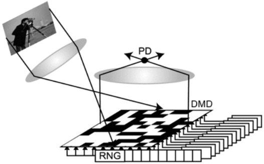
Sparse Recovery / Compressive Sensing

Robustly recover sparse x from linear measurements $y = Ax$.



Sparse Fourier

MRI



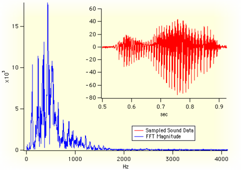
Single-Pixel Camera

Streaming Algorithms
$A(x + \Delta) = Ax + A\Delta$

Genetic Testing

My Contributions

- Sparse Fourier: minimize time complexity [HIKP12b, HIKP12a]

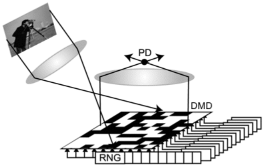


My Contributions

- Sparse Fourier: minimize time complexity [HIKP12b, HIKP12a]
- MRI: minimize Fourier *sample* complexity [GHIKPS13, IKP14]

My Contributions

- Sparse Fourier: minimize time complexity [HIKP12b, HIKP12a]
- MRI: minimize Fourier *sample* complexity [GHIKPS13, IKP14]
- Camera: use Earth-Mover Distance metric [IP11, GIP10, GIPR11]



My Contributions

- Sparse Fourier: minimize time complexity [HIKP12b, HIKP12a]
- MRI: minimize Fourier *sample* complexity [GHIKPS13, IKP14]
- Camera: use Earth-Mover Distance metric [IP11, GIP10, GIPR11]
- Streaming: improved analysis of Count-Sketch [MP14, PW11, P11]

My Contributions

- Sparse Fourier: minimize time complexity [HIKP12b, HIKP12a]
- MRI: minimize Fourier *sample* complexity [GHIKPS13, IKP14]
- Camera: use Earth-Mover Distance metric [IP11, GIP10, GIPR11]
- Streaming: improved analysis of Count-Sketch [MP14, PW11, P11]
- Genetic testing: first asymptotic gain using adaptivity [IPW11, PW13]

My Contributions

- Sparse Fourier: minimize time complexity [HIKP12b, HIKP12a]
- MRI: minimize Fourier *sample* complexity [GHIKPS13, IKP14]
- Camera: use Earth-Mover Distance metric [IP11, GIP10, GIPR11]
- Streaming: improved analysis of Count-Sketch [MP14, PW11, P11]
- Genetic testing: first asymptotic gain using adaptivity [IPW11, PW13]

Adaptive Sparse Recovery Model

- Unknown approximately k-sparse vector $x \in \mathbb{R}^n$.

Adaptive Sparse Recovery Model

- Unknown approximately k-sparse vector $x \in \mathbb{R}^n$.
- Choose $v \in \mathbb{R}^n$, observe $y = \langle v, x \rangle$.

Adaptive Sparse Recovery Model

- Unknown approximately k-sparse vector $x \in \mathbb{R}^n$.
- Choose $v \in \mathbb{R}^n$, observe $y = \langle v, x \rangle$.
- Choose another v and repeat as needed.

Adaptive Sparse Recovery Model

- Unknown approximately k-sparse vector $x \in \mathbb{R}^n$.
- Choose $v \in \mathbb{R}^n$, observe $y = \langle v, x \rangle$.
- Choose another v and repeat as needed.
- Output x' satisfying

$$\|x' - x\|_2 < (1 + \epsilon) \min_{k\text{-sparse } x_{(k)}} \|x - x_{(k)}\|_2$$

Adaptive Sparse Recovery Model

- Unknown approximately k-sparse vector $x \in \mathbb{R}^n$.
- Choose $v \in \mathbb{R}^n$, observe $y = \langle v, x \rangle$.
- Choose another v and repeat as needed.
- Output x' satisfying

$$\|x' - x\|_2 < (1 + \epsilon) \min_{k\text{-sparse } x_{(k)}} \|x - x_{(k)}\|_2$$

- Nonadaptively: $\Theta(k \log(n/k))$ measurements necessary and sufficient. [Candès-Romberg-Tao '06, DIPW '10]

Adaptive Sparse Recovery Model

- Unknown approximately k-sparse vector $x \in \mathbb{R}^n$.
- Choose $v \in \mathbb{R}^n$, observe $y = \langle v, x \rangle$.
- Choose another v and repeat as needed.
- Output x' satisfying

$$\|x' - x\|_2 < (1 + \epsilon) \min_{k\text{-sparse } x_{(k)}} \|x - x_{(k)}\|_2$$

- Nonadaptively: $\Theta(k \log(n/k))$ measurements necessary and sufficient. [Candès-Romberg-Tao '06, DIPW '10]
- Natural question: does adaptivity help?

Adaptive Sparse Recovery Model

- Unknown approximately k-sparse vector $x \in \mathbb{R}^n$.
- Choose $v \in \mathbb{R}^n$, observe $y = \langle v, x \rangle$.
- Choose another v and repeat as needed.
- Output x' satisfying

$$\|x' - x\|_2 < (1 + \epsilon) \min_{k\text{-sparse } x_{(k)}} \|x - x_{(k)}\|_2$$

- Nonadaptively: $\Theta(k \log(n/k))$ measurements necessary and sufficient. [Candès-Romberg-Tao '06, DIPW '10]
- Natural question: does adaptivity help?
  - Studied in [MSW08, JXC08, CHNR08, AWZ08, HCN09, ACD11, ...]

Adaptive Sparse Recovery Model

- Unknown approximately k-sparse vector $x \in \mathbb{R}^n$.
- Choose $v \in \mathbb{R}^n$, observe $y = \langle v, x \rangle$.
- Choose another v and repeat as needed.
- Output x' satisfying

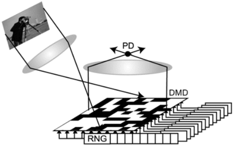
$$\|x' - x\|_2 < (1 + \epsilon) \min_{k\text{-sparse } x_{(k)}} \|x - x_{(k)}\|_2$$

- Nonadaptively: $\Theta(k \log(n/k))$ measurements necessary and sufficient. [Candès-Romberg-Tao '06, DIPW '10]
- Natural question: does adaptivity help?
  - Studied in [MSW08, JXC08, CHNR08, AWZ08, HCN09, ACD11, ...]
- First asymptotic improvement: $O(k \log \log(n/k))$ measurements. [IPW '11]

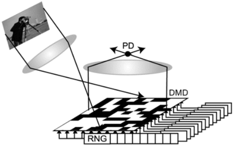
Applications of Adaptivity

Applications of Adaptivity

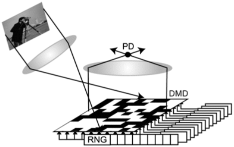
Applications of Adaptivity



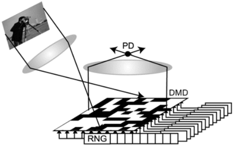
Applications of Adaptivity



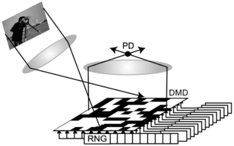
Applications of Adaptivity



Applications of Adaptivity



Applications of Adaptivity



Outline of Algorithm

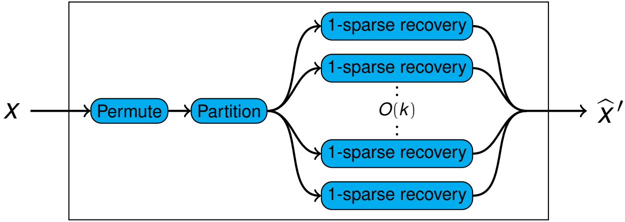
Theorem

Adaptive k-sparse recovery is possible with $O(k \log \log(n/k))$ measurements.

Outline of Algorithm

Theorem

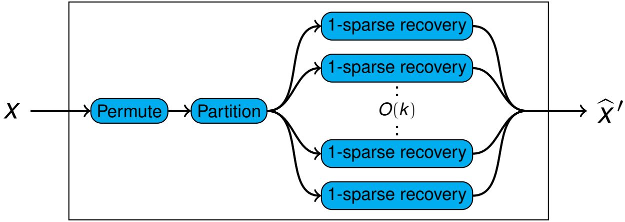
Adaptive k-sparse recovery is possible with $O(k \log\log(n/k))$ measurements.



Outline of Algorithm

Theorem

Adaptive k-sparse recovery is possible with $O(k \log\log(n/k))$ measurements.



Suffices to solve for $k = 1$:

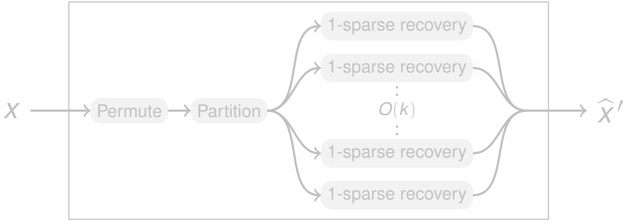
Lemma

Adaptive 1-sparse recovery is possible with $O(\log\log n)$ measurements.

Outline of Algorithm

Theorem

Adaptive k-sparse recovery is possible with $O(k \log\log(n/k))$ measurements.



Suffices to solve for $k = 1$:

Lemma

Adaptive 1-sparse recovery is possible with $O(\log\log n)$ measurements.

1-sparse recovery: non-adaptive lower bound

Lemma

Adaptive 1*-sparse recovery is possible with* $O(\log \log n)$ *measurements.*

1-sparse recovery: non-adaptive lower bound

Lemma

Adaptive 1*-sparse recovery is possible with* $O(\log \log n)$ *measurements.*

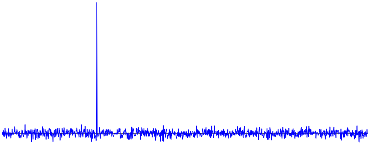
- Non-adaptive lower bound: why is this hard?

1-sparse recovery: non-adaptive lower bound

Lemma

Adaptive 1-sparse recovery is possible with $O(\log\log n)$ measurements.

- Non-adaptive lower bound: why is this hard?
- Hard case: x is random e_i plus Gaussian noise w with $\|w\|_2 \approx 1$.

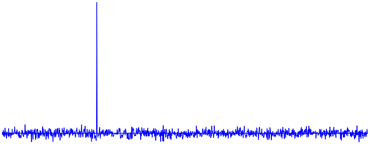


1-sparse recovery: non-adaptive lower bound

Lemma

Adaptive 1*-sparse recovery is possible with* $O(\log\log n)$ *measurements.*

- Non-adaptive lower bound: why is this hard?
- Hard case: x is random e_i plus Gaussian noise w with $\|w\|_2 \approx 1$.



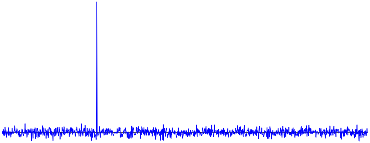
- Robust recovery must locate i.

1-sparse recovery: non-adaptive lower bound

Lemma

Adaptive 1-sparse recovery is possible with $O(\log \log n)$ measurements.

- Non-adaptive lower bound: why is this hard?
- Hard case: x is random e_i plus Gaussian noise w with $\|w\|_2 \approx 1$.



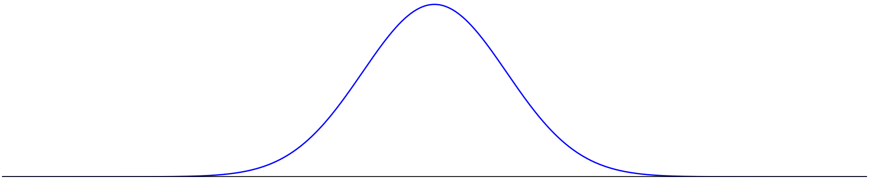
- Robust recovery must locate i.
- Observations $\langle v, x \rangle = v_i + \langle v, w \rangle = v_i + \frac{\|v\|_2}{\sqrt{n}} z$, for $z \sim N(0, 1)$.

1-sparse recovery: non-adaptive lower bound

- Observe $\langle v, x \rangle = v_i + \frac{\|v\|_2}{\sqrt{n}} z$, where $z \sim N(0, 1)$

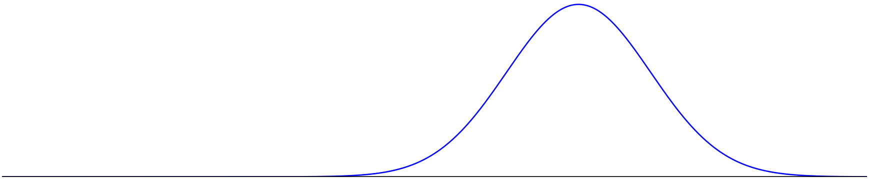
1-sparse recovery: non-adaptive lower bound

- Observe $\langle v, x\rangle = v_i + \frac{\|v\|_2}{\sqrt{n}} z$, where $z \sim N(0, 1)$



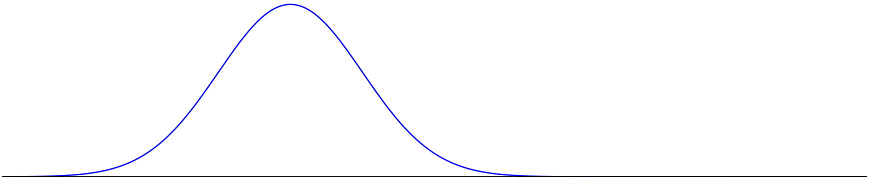
1-sparse recovery: non-adaptive lower bound

- Observe $\langle v, x \rangle = v_i + \frac{\|v\|_2}{\sqrt{n}} z$, where $z \sim N(0, 1)$



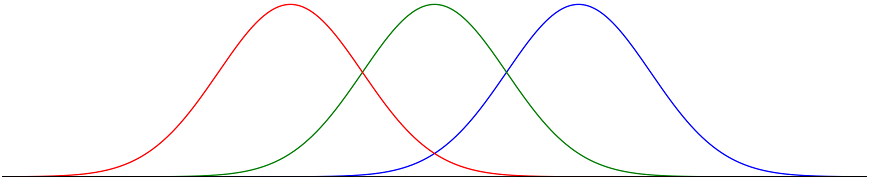
1-sparse recovery: non-adaptive lower bound

- Observe $\langle v, x \rangle = v_i + \frac{\|v\|_2}{\sqrt{n}} z$, where $z \sim N(0, 1)$



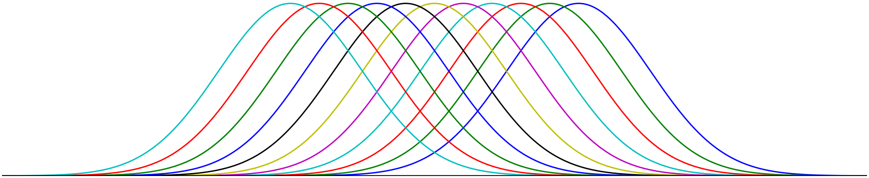
1-sparse recovery: non-adaptive lower bound

- Observe $\langle v, x \rangle = v_i + \frac{\|v\|_2}{\sqrt{n}} z$, where $z \sim N(0,1)$



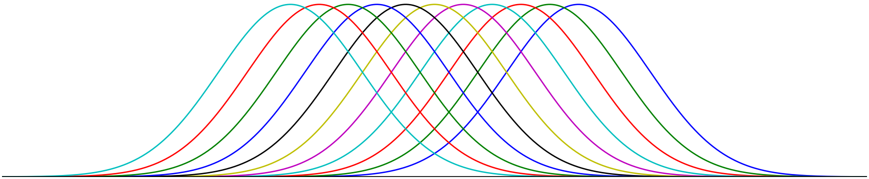
1-sparse recovery: non-adaptive lower bound

- Observe $\langle v, x\rangle = v_i + \frac{\|v\|_2}{\sqrt{n}} z$, where $z \sim N(0, 1)$



1-sparse recovery: non-adaptive lower bound

- Observe $\langle v, x\rangle = v_i + \frac{\|v\|_2}{\sqrt{n}} z$, where $z \sim N(0, 1)$



- Shannon 1948: information capacity

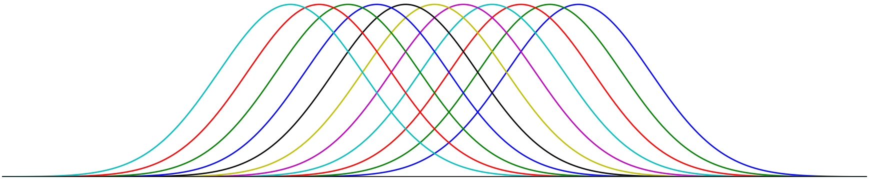
$$I(i, \langle v, x\rangle) \leqslant \frac{1}{2}\log(1 + \text{SNR})$$

where *SNR* denotes the "signal-to-noise ratio,"

$$SNR = \frac{\mathbb{E}[\text{signal}^2]}{\mathbb{E}[\text{noise}^2]} = \frac{\mathbb{E}[v_i^2]}{\|v\|_2^2/n} = 1$$

1-sparse recovery: non-adaptive lower bound

- Observe $\langle v, x \rangle = v_i + \frac{\|v\|_2}{\sqrt{n}} z$, where $z \sim N(0, 1)$



- Shannon 1948: information capacity

$$I(i, \langle v, x \rangle) \leqslant \frac{1}{2} \log(1 + \mathrm{SNR})$$

where *SNR* denotes the "signal-to-noise ratio,"

$$SNR = \frac{\mathbb{E}[\text{signal}^2]}{\mathbb{E}[\text{noise}^2]} = \frac{\mathbb{E}[v_i^2]}{\|v\|_2^2 / n} = 1$$

- Finding i needs $\Omega(\log n)$ non-adaptive measurements.

1-sparse recovery: changes in adaptive setting

- Information capacity

$$I(i, \langle v, x \rangle) \leqslant \frac{1}{2} \log(1 + \mathrm{SNR}).$$

where *SNR* denotes the "signal-to-noise ratio,"

$$SNR = \frac{\mathbb{E}[v_i^2]}{\|v\|_2^2 / n}.$$

1-sparse recovery: changes in adaptive setting

- Information capacity

$$I(i, \langle v, x \rangle) \leqslant \frac{1}{2} \log(1 + \mathrm{SNR}).$$

where *SNR* denotes the "signal-to-noise ratio,"

$$SNR = \frac{\mathbb{E}[v_i^2]}{\|v\|_2^2 / n}.$$

- If i is independent of v, this is $O(1)$.

1-sparse recovery: changes in adaptive setting

- Information capacity

$$I(i, \langle v, x \rangle) \leqslant \frac{1}{2} \log(1 + \mathrm{SNR}).$$

where *SNR* denotes the "signal-to-noise ratio,"

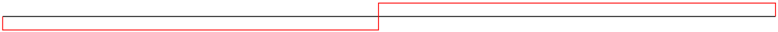
$$SNR = \frac{\mathbb{E}[v_i^2]}{\|v\|_2^2 / n}.$$

- If i is independent of v, this is $O(1)$.
- As we learn about i, we can increase the SNR.

1-sparse recovery: idea

$x = e_i + w$

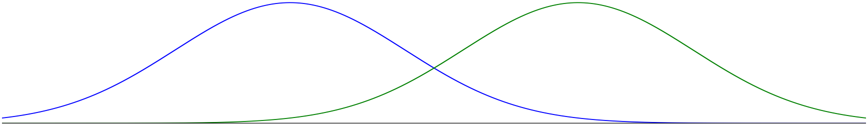
v



$SNR = 2$

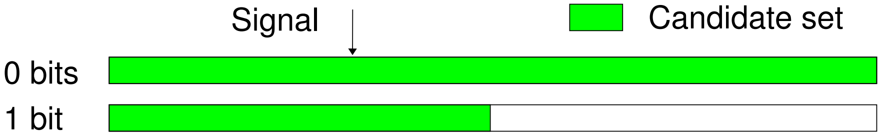
$I(i, \langle v, x \rangle) \leqslant \log SNR = 1$

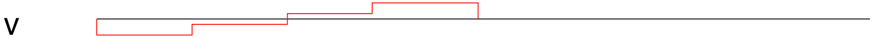
$\langle v, x \rangle = v_i + \langle v, w \rangle$



1-sparse recovery: idea

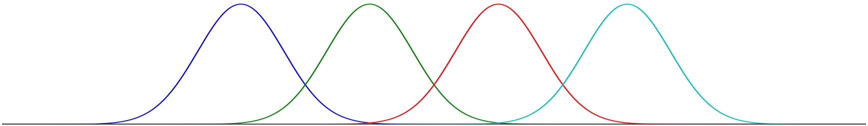
$$x = e_i + w$$





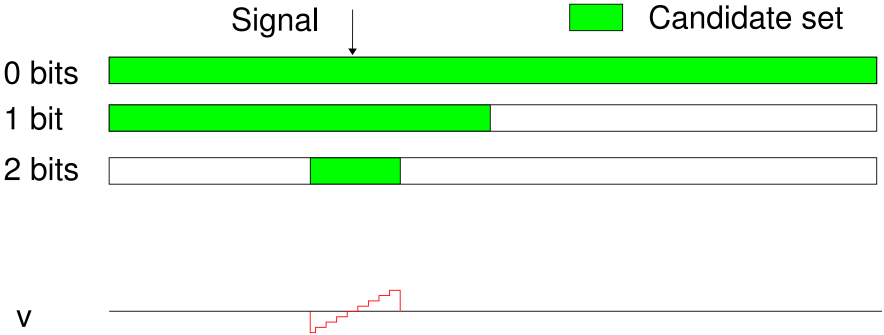
$SNR = 2^2$ $\qquad$ $I(i, \langle v, x \rangle) \leqslant \log SNR = 2$

$$\langle v, x \rangle = v_i + \langle v, w \rangle$$



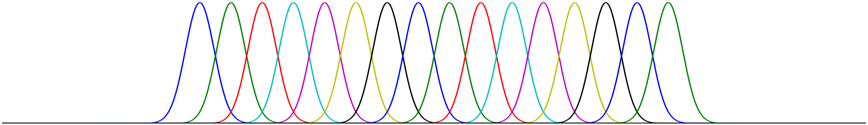
1-sparse recovery: idea

$$x = e_i + w$$



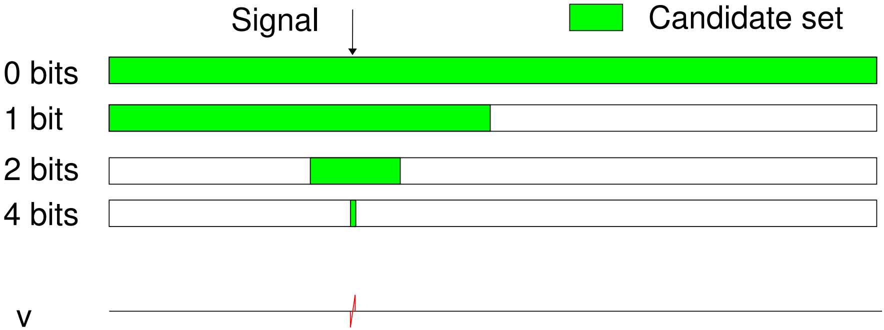
$SNR = 2^4$ $\qquad$ $I(i, \langle v, x \rangle) \leqslant \log SNR = 4$

$$\langle v, x \rangle = v_i + \langle v, w \rangle$$



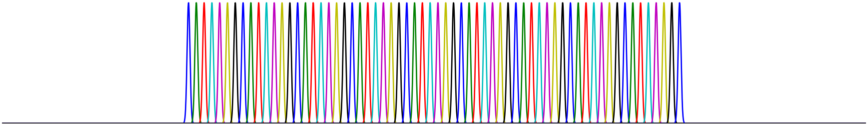
1-sparse recovery: idea

$$x = e_i + w$$



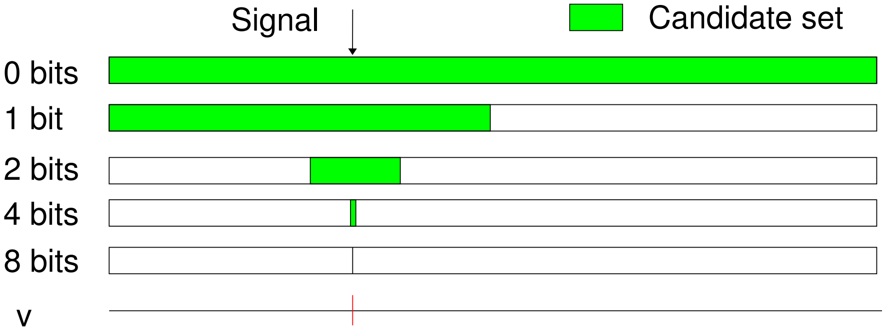
$SNR = 2^8$ $\qquad I(i, \langle v, x \rangle) \leqslant \log SNR = 8$

$$\langle v, x \rangle = v_i + \langle v, w \rangle$$



1-sparse recovery: idea

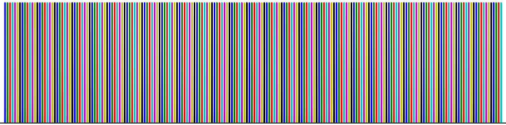
$$x = e_i + w$$



$SNR = 2^{16}$

$I(i, \langle v, x \rangle) \leqslant \log SNR = 16$

$$\langle v, x \rangle = v_i + \langle v, w \rangle$$



1-sparse recovery

Lemma (IPW11)

Adaptive 1-sparse recovery takes $O(\log \log n)$ measurements.

1-sparse recovery

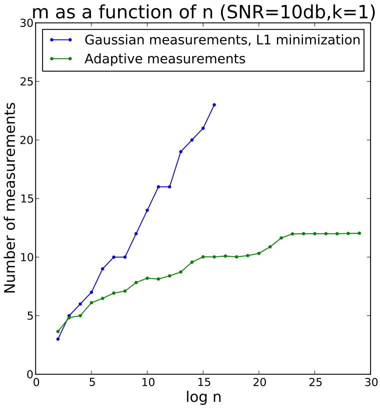
Lemma (IPW11, PW13)

Adaptive 1*-sparse recovery takes* $\Theta(\log\log n)$ *measurements.*

1-sparse recovery

Lemma (IPW11, PW13)

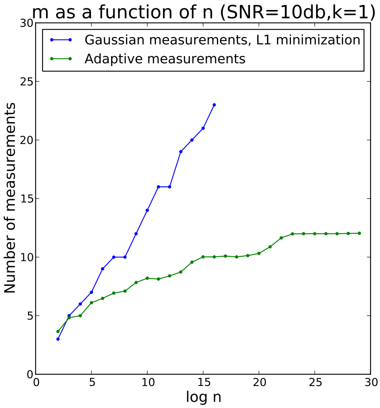
Adaptive 1*-sparse recovery takes* $\Theta(\log \log n)$ *measurements.*

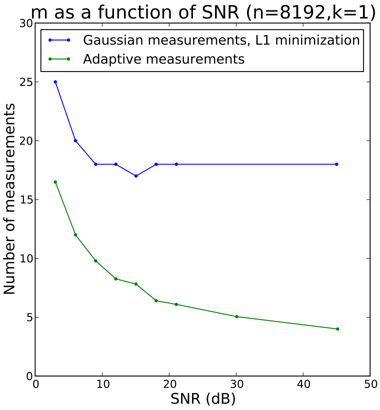


1-sparse recovery

Lemma (IPW11, PW13)

Adaptive 1*-sparse recovery takes* $\Theta(\log \log n)$ *measurements.*

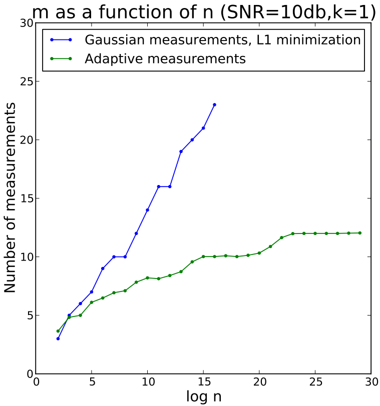


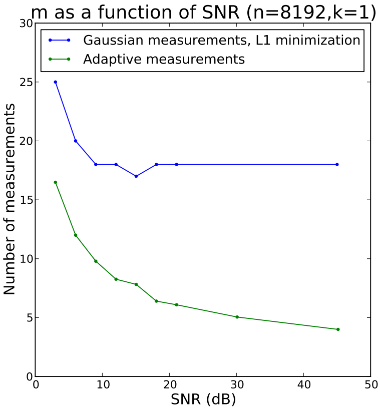


1-sparse recovery

Lemma (IPW11, PW13)

Adaptive 1*-sparse recovery takes* $\Theta(\log\log n)$ *measurements.*





Gives $\Theta(k \log\log(n/k))$ k-sparse recovery via general framework.

Summary

- Sparse Fourier transform
  - Fastest algorithm for Fourier transforms on sparse data
  - Already has applications with substantial improvements

Summary

- Sparse Fourier transform
  - Fastest algorithm for Fourier transforms on sparse data
  - Already has applications with substantial improvements
- Broader sparse recovery theory
  - Sparse Fourier: minimize time complexity [HIKP12]
  - MRI: minimize Fourier *sample* complexity [GHIKPS13, IKP14]
  - Camera: use Earth-Mover Distance metric [IP11, GIP10, GIPR11]
  - Streaming: improved analysis of Count-Sketch [MP14, PW11, P11]
  - Genetic testing: first asymptotic gain using adaptivity [IPW11, PW13]

Summary

- Sparse Fourier transform
  - Fastest algorithm for Fourier transforms on sparse data
  - Already has applications with substantial improvements
- Broader sparse recovery theory
  - Sparse Fourier: minimize time complexity [HIKP12]
  - MRI: minimize Fourier *sample* complexity [GHIKPS13, IKP14]
  - Camera: use Earth-Mover Distance metric [IP11, GIP10, GIPR11]
  - Streaming: improved analysis of Count-Sketch [MP14, PW11, P11]
  - Genetic testing: first asymptotic gain using adaptivity [IPW11, PW13]
- Lower bounds
  - Based on Gaussian channel capacity: tight bounds, extensible to adaptive settings.
  - Based on communication complexity: extends to ℓ_1 setting.

Summary

- Sparse Fourier transform
  - Fastest algorithm for Fourier transforms on sparse data
  - Already has applications with substantial improvements
- Broader sparse recovery theory
  - Sparse Fourier: minimize time complexity [HIKP12]
  - MRI: minimize Fourier *sample* complexity [GHIKPS13, IKP14]
  - Camera: use Earth-Mover Distance metric [IP11, GIP10, GIPR11]
  - Streaming: improved analysis of Count-Sketch [MP14, PW11, P11]
  - Genetic testing: first asymptotic gain using adaptivity [IPW11, PW13]
- Lower bounds
  - Based on Gaussian channel capacity: tight bounds, extensible to adaptive settings.
  - Based on communication complexity: extends to ℓ_1 setting.

Thank You

The Future

- Make sparse Fourier applicable to more problems

The Future

- Make sparse Fourier applicable to more problems
  - Better sample complexity

The Future

- Make sparse Fourier applicable to more problems
  - Better sample complexity
  - Incorporate stronger notions of structure

The Future

- Make sparse Fourier applicable to more problems
  - Better sample complexity
  - Incorporate stronger notions of structure
- Tight constants in compressive sensing

The Future

- Make sparse Fourier applicable to more problems
  - Better sample complexity
  - Incorporate stronger notions of structure
- Tight constants in compressive sensing
  - Analogous to channel capacity in coding theory.

The Future

- Make sparse Fourier applicable to more problems
  - Better sample complexity
  - Incorporate stronger notions of structure
- Tight constants in compressive sensing
  - Analogous to channel capacity in coding theory.
  - Lower bound techniques, from information theory, should be strong enough.