Faster Algorithms for Sparse Fourier Transform

Haitham Hassanieh Piotr Indyk Dina Katabi Eric Price

MIT

Material from:

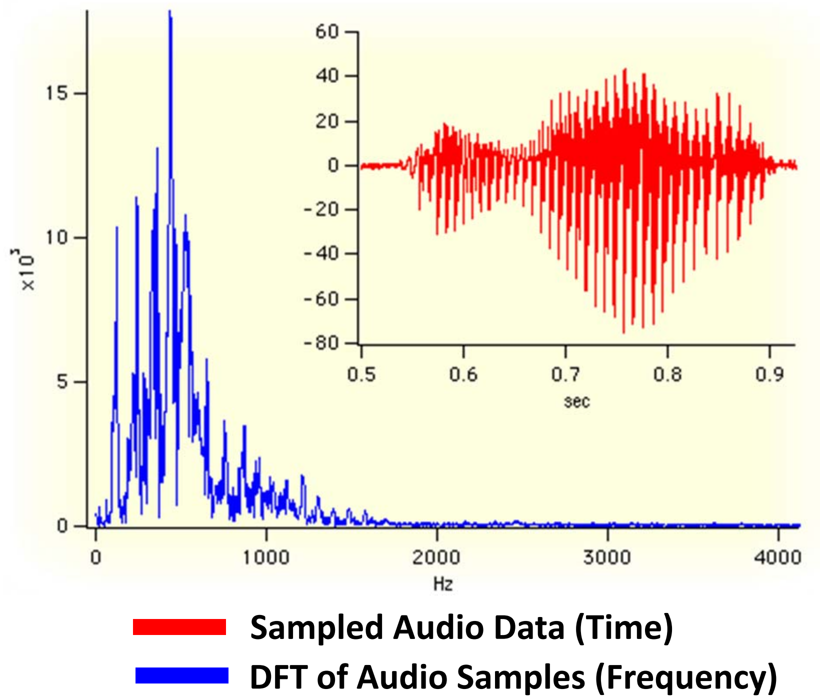
- Hassanieh, Indyk, Katabi, Price, "Simple and Practical Algorithms for Sparse Fourier Transform, SODA'12.
- Hassanieh, Indyk, Katabi, Price, "Nearly Optimal Sparse Fourier Transform", STOC'12.

The Discrete Fourier Transform

- Discrete Fourier Transform:
  - Given: a signal $\mathbf{x} \in \mathbb{C}^n$
  - Goal: compute the frequency vector $\hat{\mathbf{x}}$ such that for $f \in [1 \ldots n]$:

$$\hat{\mathbf{x}}_f = \sum \mathbf{x}_t \, e^{-\mathrm{i}\, 2\pi\, tf/n}$$

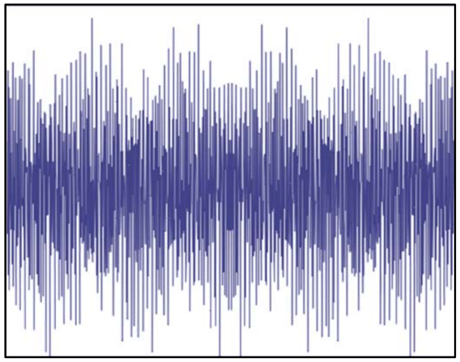
- Fundamental tool:
  - Compression (audio, image, video)
  - Signal processing
  - Data analysis
  - Wireless Communication
  - …



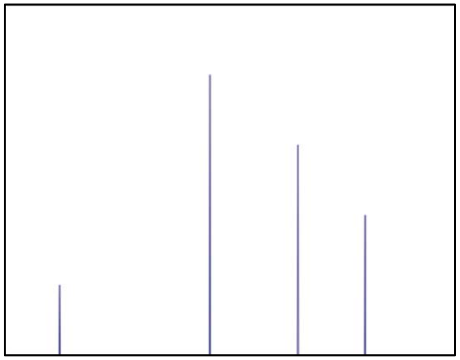
- Fastest algorithm since 1960s

FFT : $O(n \log n)$ time

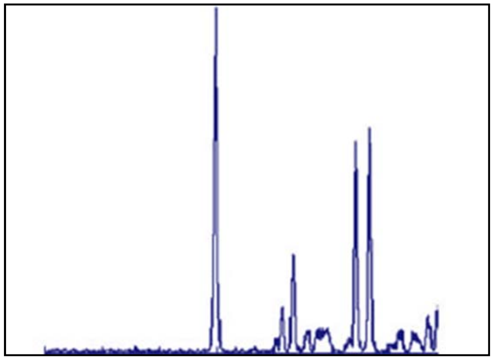
Sparse Fourier Transform



Time Domain Signal



Sparse Frequency Spectrum



Approximately Sparse Frequency Spectrum

- Often the Fourier transform is dominated by a small number of "peaks"
  - Only few of the frequency coefficients are nonzero.
  - An exactly k-sparse signal has only k nonzero frequency coefficients.
  - In practice : approximate a sparse signal using the k largest peaks.
- Problem : Can we recover the k-sparse frequency spectrum faster than FFT?

Previous Work

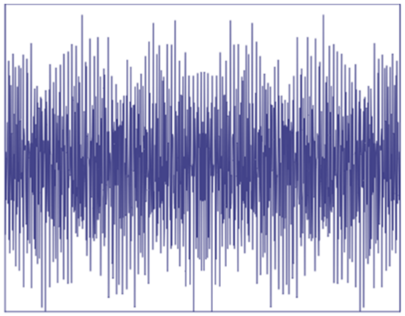
- Algorithms:
  - [KM92, Mansour92, GGIMS02, AGS03, GMS05, Iwen10, Aka10]
- Best running time: [GMS05] $O(k \log^4 n)$
  - In theory : Improves over FFT for $n/k >> \log^3 n$
  - In practice : Large constants; need $n/k > 40{,}000$ to beat FFT
- Goal:
  - Theory: improve over FFT for **all** values of $k = o(n)$
  - Practice: faster runtime than FFT.

Our results

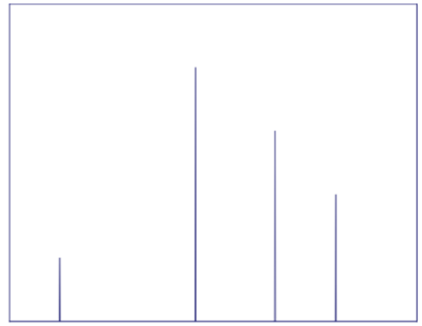
- Randomized algorithms, with large constant probability of success
- Exactly k-sparse case, recover $\hat{\mathbf{x}}$: $O(k \log n)$
  - Optimal if FFT optimal
- Approximately k-sparse case, recover $\widehat{\mathbf{x}'}$:
  - Let $\mathrm{Err}_2^k(\hat{\mathbf{x}}) = \min_{k\ sparse\ \hat{\mathbf{x}}_k} \|\hat{\mathbf{x}} - \hat{\mathbf{x}}_k\|_2$
  - l_2/l_2 guarantee $\left\|\widehat{\mathbf{x}'} - \hat{\mathbf{x}}\right\|_2 \leq c \times \mathrm{Err}_2^k(\hat{\mathbf{x}})$: $O(k \log(n) \log(n/k))$
    - Improves over FFT for any $k \ll n$
  - l_∞/l_2 guarantee $\left\|\widehat{\mathbf{x}'} - \hat{\mathbf{x}}\right\|_\infty \leq \frac{c}{\sqrt{k}} \mathrm{Err}_2^k(\hat{\mathbf{x}})$: $O(\sqrt{nk \log n} \log n)$
    - Improves over FFT for $k \ll n/\log n$

Sparse FFT - Algorithm

Intuition



Time Domain Signal

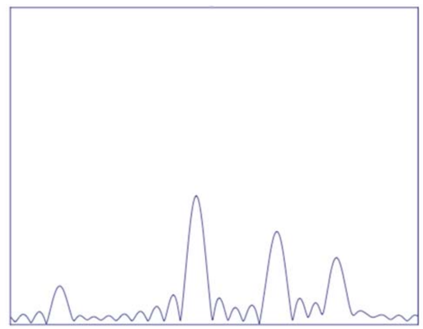


Frequency Domain

n-point DFT : $n \log(n)$

$\mathbf{x}$ ➡ $\hat{\mathbf{x}}$

Cut off Time signal

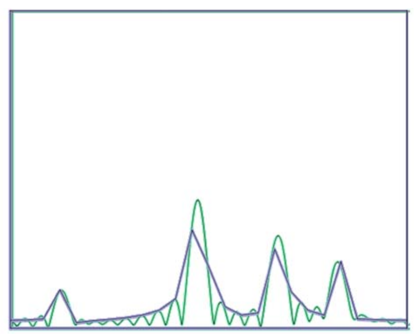


Frequency Domain

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

$\mathbf{x} \times$ **Boxcar** ➡ $\hat{\mathbf{x}} *$ **sinc**

First B samples



Frequency Domain

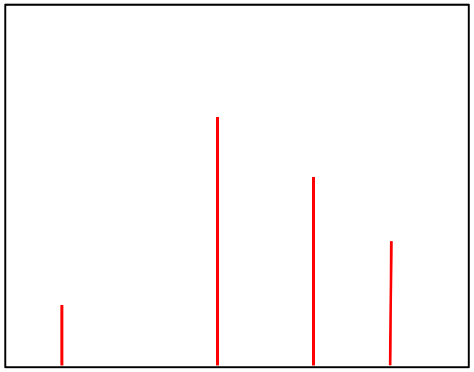
B-point DFT of first B terms: $B \log(B)$

Alias $(\mathbf{x} \times$ **Boxcar**$)$

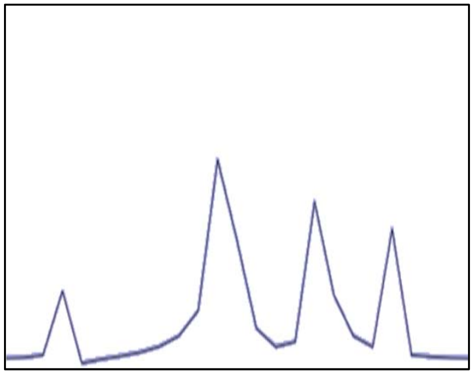
⬇

Subsample $(\hat{\mathbf{x}} *$ **sinc**$)$

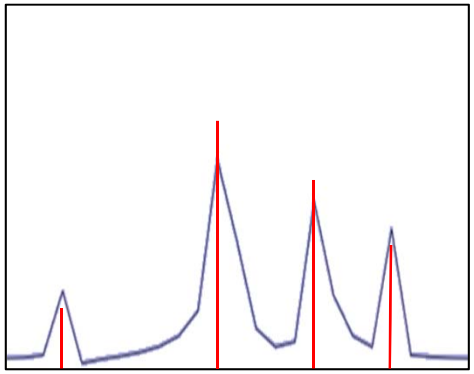
Framework



n-point DFT
of all n samples



B-point DFT
of first B sample



n frequencies hash
into B buckets

- “Hashes” the n frequency coefficients into B buckets in $O(B \log B)$ time

- n/B frequencies coefficients hash into each bucket.

- Ideally we want:
  - Value of each bucket = sum over n/B frequencies that hash to it.
  - If one large frequency in the bucket → Estimate its value from value of the bucket.

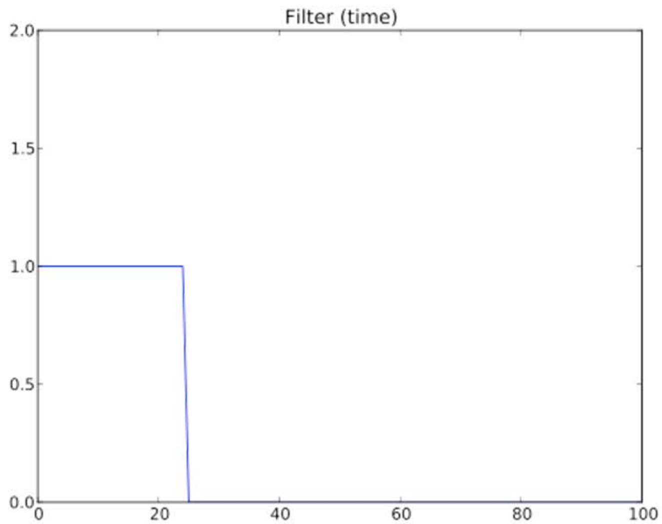
Issues

- Leakage
  - value of bucket = **Subsample** $(\hat{\mathbf{x}} * \mathbf{sinc})$
  - sum over all frequencies weighted by sinc
  - frequencies outside the bucket leak power into the bucket.

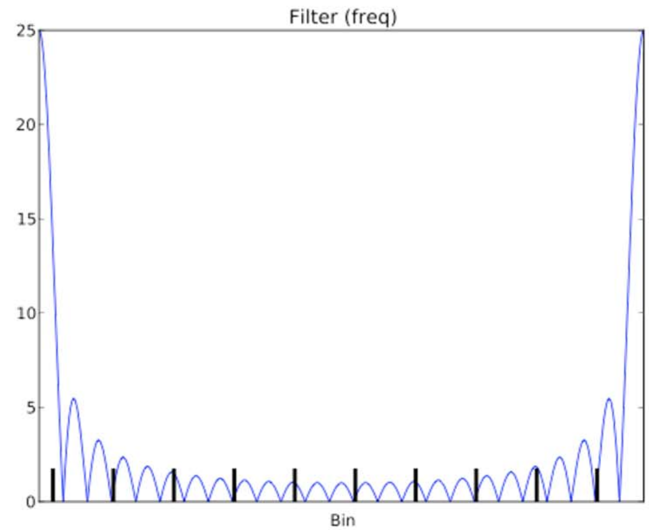
  - Replace **sinc** with a better **Filter**
  - **GOAL : Subsample** $(\hat{\mathbf{x}} * \mathbf{Filter})$ = sum only over n/B frequencies that hash to the bucket

- Given these B buckets, how can we estimate the locations and values the k large frequencies?

Filter: Sinc



$F = \text{Boxcar}$

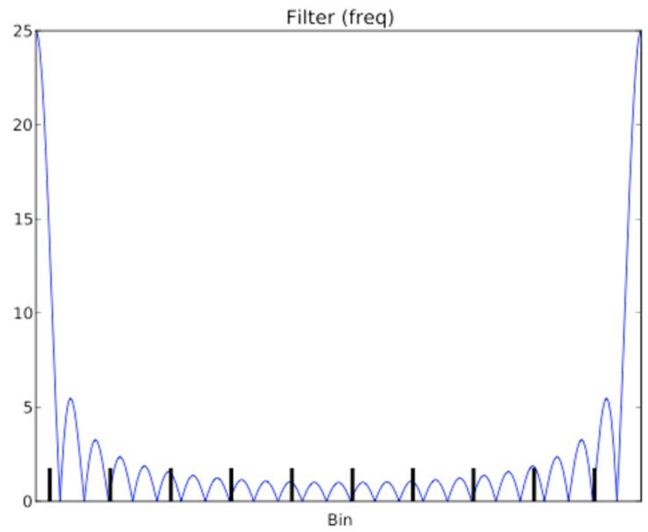


$\hat{F} = \text{Sinc}$

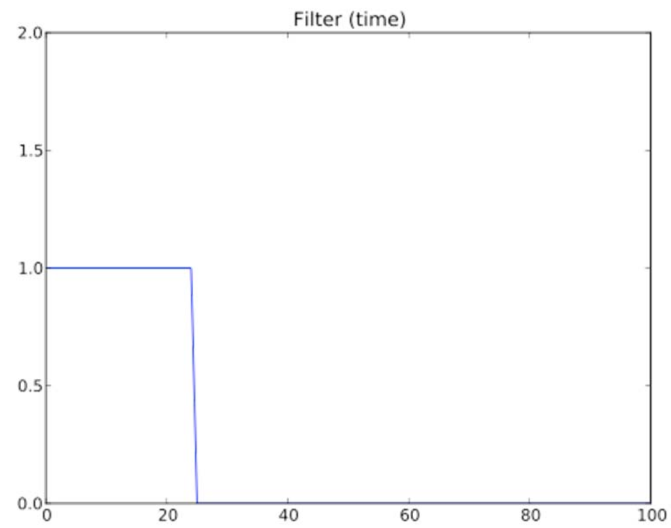
– Polynomial decay

– Leaking many buckets

Filter: Boxcar



F = Sinc

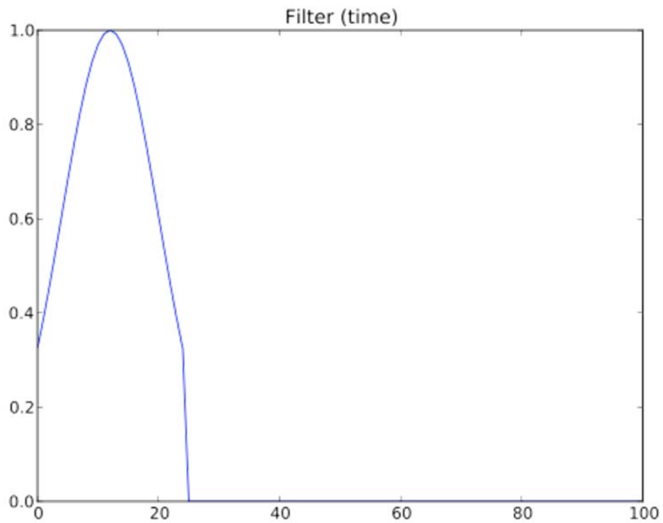


$\hat{F}$ = Boxcar

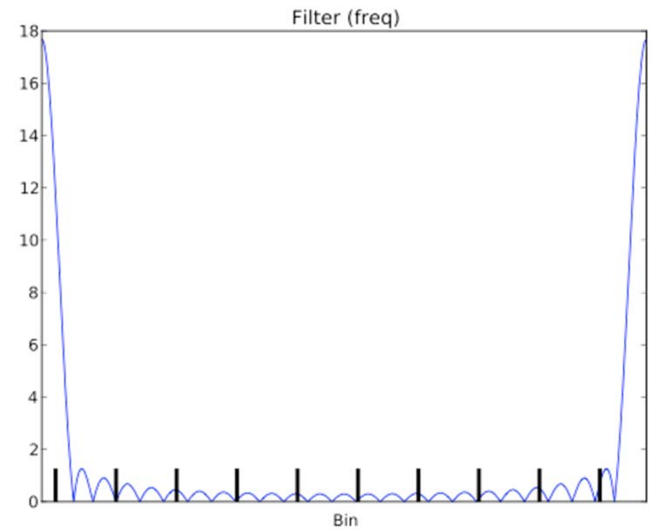
– Large support in time domain → Cannot truncate

– Need all n time domain samples

Filter: Gaussian



F = Gaussian

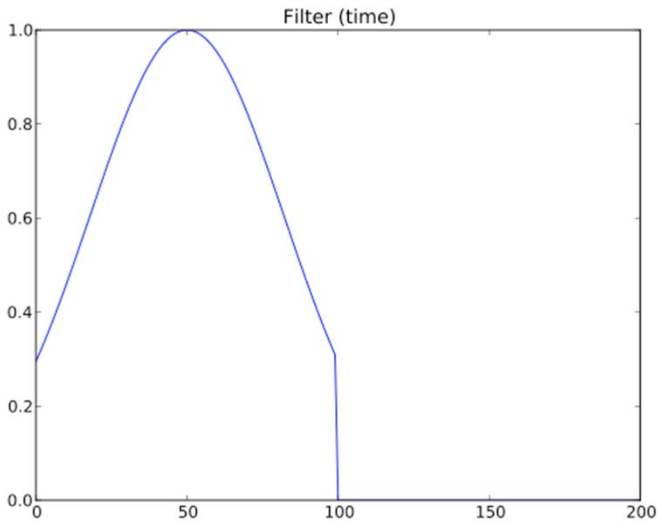


$\hat{F}$ = Gaussian

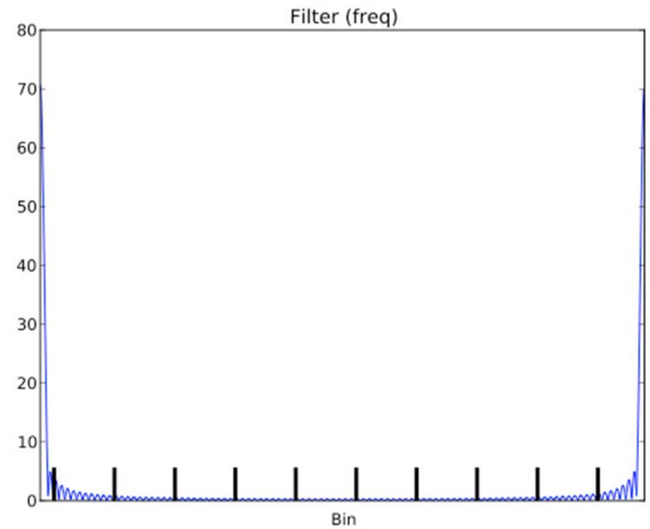
– Exponential decay

– Leaking to $\sqrt{\log n}$ buckets

Filters: Wider Gaussian



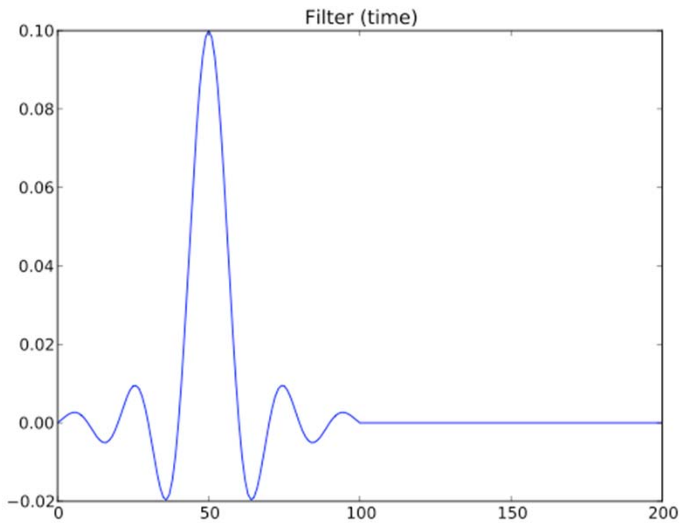
F = Wider Gaussian

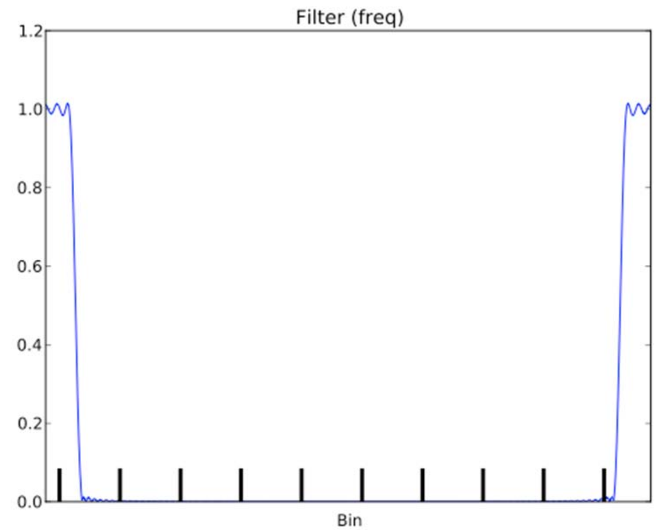


$\hat{F}$ = Narrow Gaussian

- Exponential decay
- Leaking to 0 buckets
- But trivial contribution to the correct bucket

Filters: Sinc × Gaussian





$F =$ Sinc × Gaussian $\quad\quad \hat{F} =$ Boxcar $*$ Gaussian

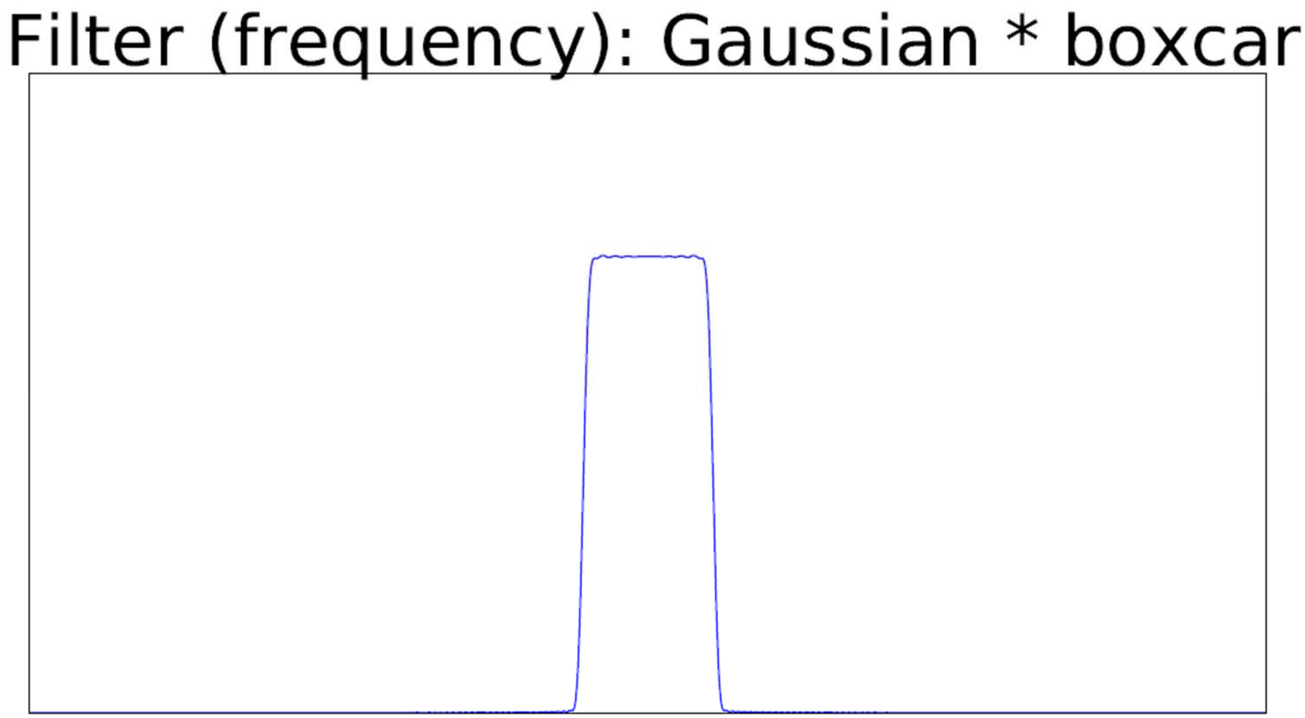
– Still exponential decay

– Leaking to at most 1 bucket

– Sufficient contribution to the correct bucket

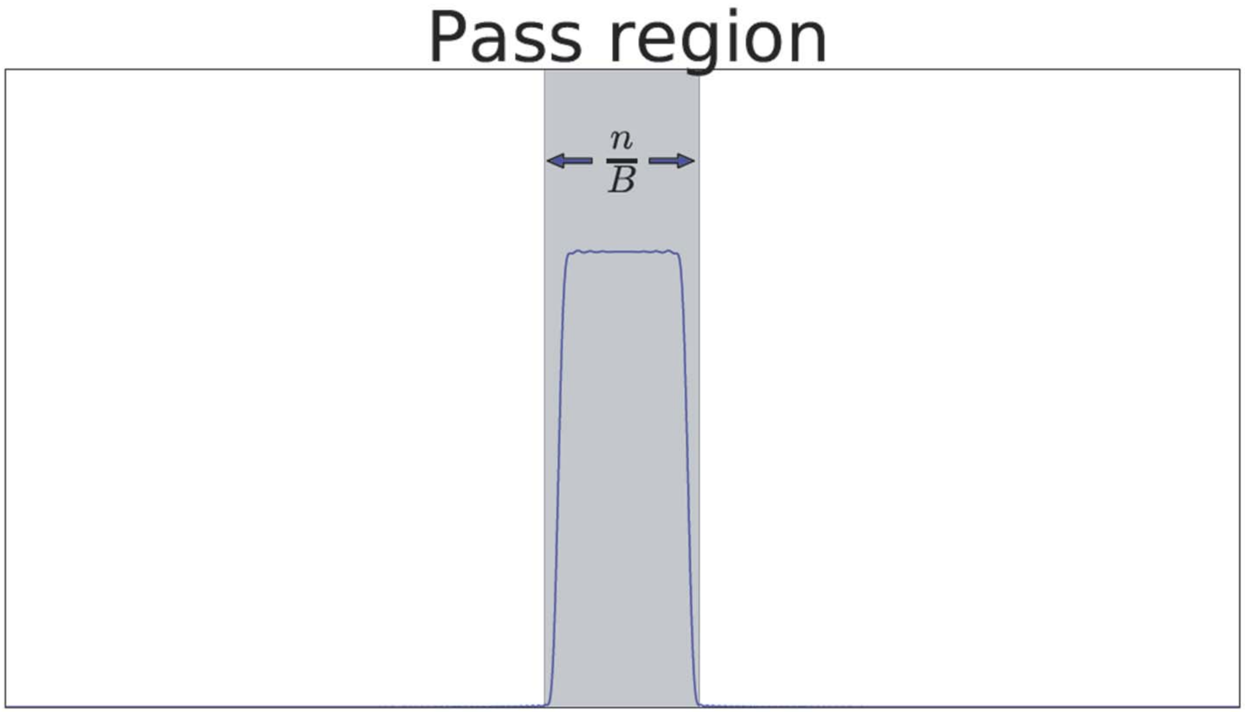
– Small support in time domain

Filters: Sinc × Gaussian



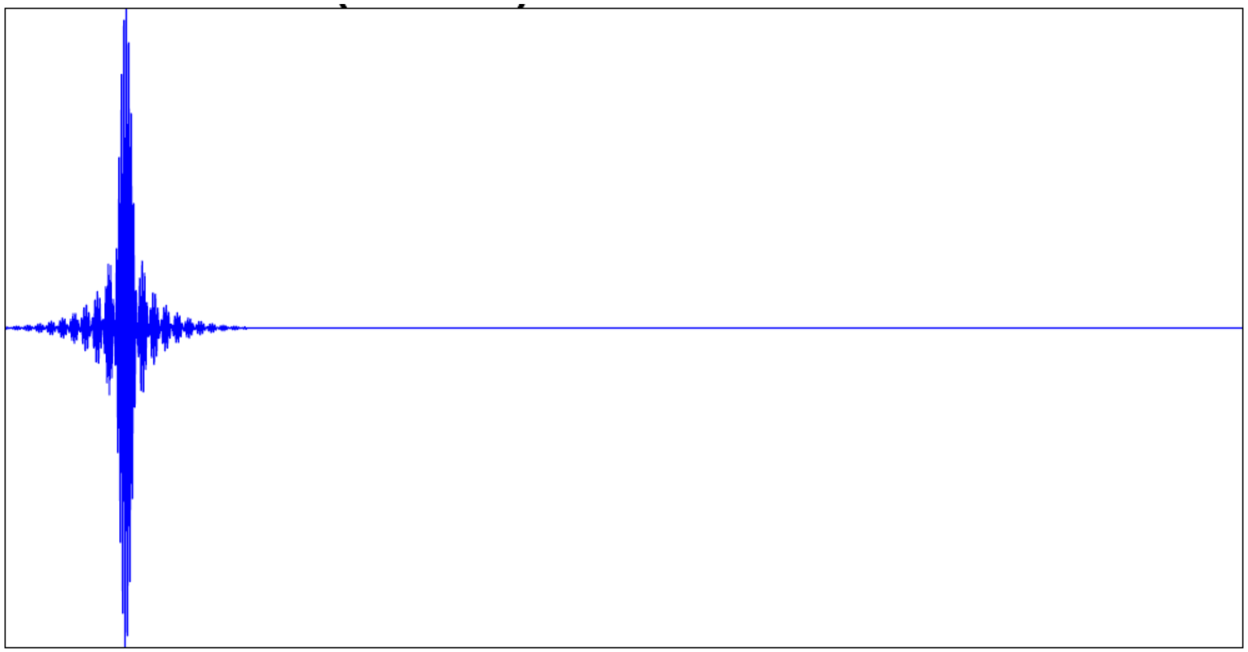
– Gaussian with standard deviation : $B\sqrt{\log n}$

Filters: Sinc × Gaussian



– Gaussian with standard deviation : $B\sqrt{\log n}$

Filters: Sinc × Gaussian



- Gaussian with standard deviation : $B\sqrt{\log n}$
- Filter F has a support of $B \log n$ in the time domain
- **Alias** $(\mathbf{x} \times F)$ into B samples

Finding the support

- $\hat{\mathbf{y}}$ = B-point DFT $(\mathbf{x} \times F)$ = Subsample $\left(\hat{\mathbf{x}} * \hat{F}\right)$
- Assume no collisions:
  - At most one large frequency hashes into each bucket.
  - Large frequency f_1 hashes to bucket b_1 :

$$\hat{\mathbf{y}}_{b_1} = \hat{\mathbf{x}}_{f_1} \times \hat{F}_{\Delta} + \text{~~leakage~~}$$

  - Recall: DFT($\mathbf{x}^{\tau}$) = $\hat{\mathbf{x}} \times e^{-\mathrm{i}\,2\pi\,\tau f/n}$

  - $\hat{\mathbf{y}}^{\tau}$ = B-point DFT $(\mathbf{x}^{\tau} \times F)$:

$$\hat{\mathbf{y}}^{\tau}{}_{b_1} = \hat{\mathbf{x}}_{f_1} \times e^{-\mathrm{i}\,2\pi\,\tau f_1/n} \times \hat{F}_{\Delta} + \text{~~leakage~~}$$

Finding the support

- $\hat{\mathbf{y}}$ = B-point DFT $(\mathbf{x} \times F)$ = Subsample $(\hat{\mathbf{x}} * \hat{F})$
- Assume no collisions:
  - At most one large frequency hashes into each bucket.
  - Large frequency f_1 hashes to bucket b_1 :

$$\hat{\mathbf{y}}_{b_1} = \hat{\mathbf{x}}_{f_1} \times \hat{F}_{\Delta}$$

$$\hat{\mathbf{y}}^{\mathbf{1}}{}_{b_1} = \hat{\mathbf{x}}_{f_1} \times e^{-\mathrm{i}\,2\pi\, f_1/n} \times \hat{F}_{\Delta}$$

$$\frac{\hat{\mathbf{y}}_{b_1}}{\hat{\mathbf{y}}^{\mathbf{1}}{}_{b_1}} = e^{-\mathrm{i}\,2\pi\, f_1/n} \rightarrow \mathrm{angle}\left(\frac{\hat{\mathbf{y}}_{b_1}}{\hat{\mathbf{y}}^{\mathbf{1}}{}_{b_1}}\right) = -2\pi\, f_1/n$$

$$f_1 = -\frac{n}{2\pi} \cdot \mathrm{angle}\left(\frac{\hat{\mathbf{y}}_{b_1}}{\hat{\mathbf{y}}^{\mathbf{1}}{}_{b_1}}\right) \bmod n \qquad \hat{\mathbf{x}}_{f_1} = \frac{\hat{\mathbf{y}}_{b_1}}{\hat{F}_{\Delta}}$$

Random Hashing

- Some Large frequencies collide:
  - Subtract and recurse
  - Small number of collisions → converges in few iterations

- Every iteration needs new random hashing:
  - Permute time domain signal → permute frequency domain
  - σ is invertible mod n :

$$\mathbf{x}'_t = \mathbf{x}_{\sigma t} \times e^{-\mathrm{i}\,2\pi\, t\beta/n} \qquad \hat{\mathbf{x}}'_f = \hat{\mathbf{x}}_{\sigma^{-1} f + \beta}$$

Algorithm

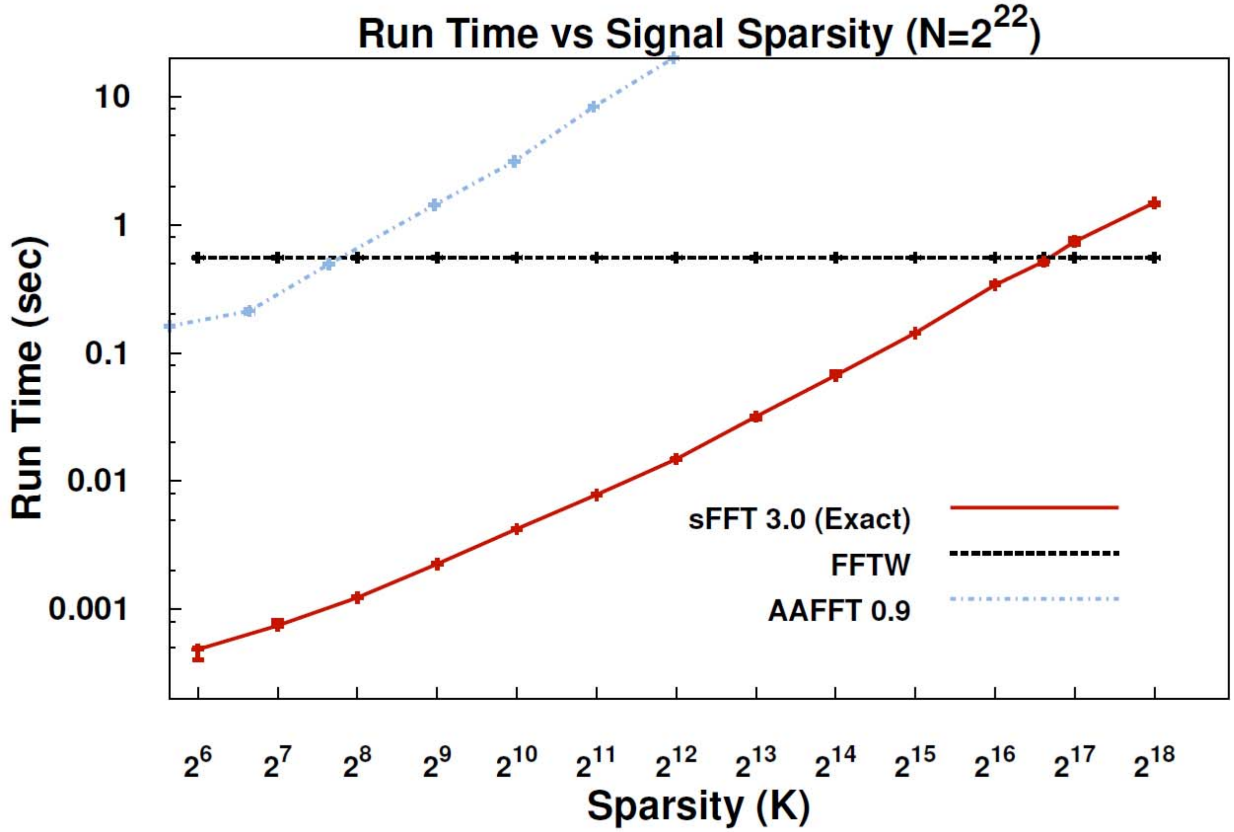
- Iteration i :
  - $B_i \propto k/\, 2^{i-1}$
  - Permute spectrum : $\mathbf{x'}_{\boldsymbol{t}} = \mathbf{x}_{\sigma \boldsymbol{t}} \times\ e^{-\mathrm{i}\,2\pi\, t\beta/n}$
  - $\hat{\mathbf{y}} = B_i$ -point DFT $(\mathbf{x}' \times F)$ = Subsample $\left(\hat{\mathbf{x}}' * \widehat{F}\right)$
  - Repeat with time shift to get $\hat{\mathbf{y}}^{\tau}$
  - Subtract large frequencies recovered in previous iterations
  - Recover locations and values of remaining large frequencies

- Iteration i recovers $k/\, 2^{i}$ of the large frequencies with probability 3/4 in O($B_i \log n$) time

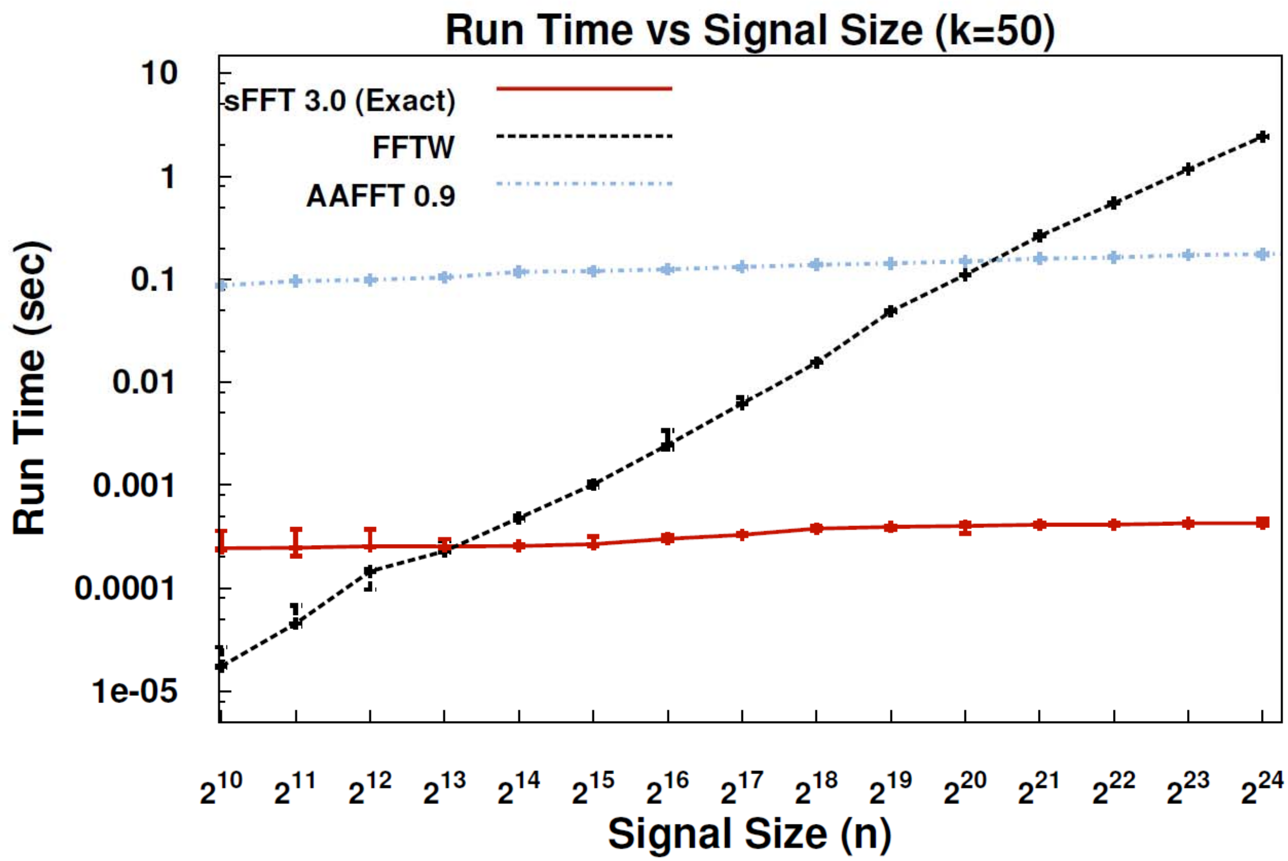
Theorem: Recover $\hat{\mathbf{x}}$ in O$(k \log n)$ with probability 3/4

Experiments
(variant exactly k-sparse algorithm)

Experiments



Experiments, ctd



Conclusions

- Sparse Fourier Transform with running times :
  - $O(k \log n)$ for exactly sparse case
  - $O(k \log(n) \log(n/k))$ for approximately sparse case
  - Improves over FFT for $k \ll n$

- Significant improvement in practice : $n/k > 32$

- $k \log n$ time for **approximately** sparse signals?
  - Not clear: $k \log(n/k)$ samples needed, extra $\log n$ for processing.