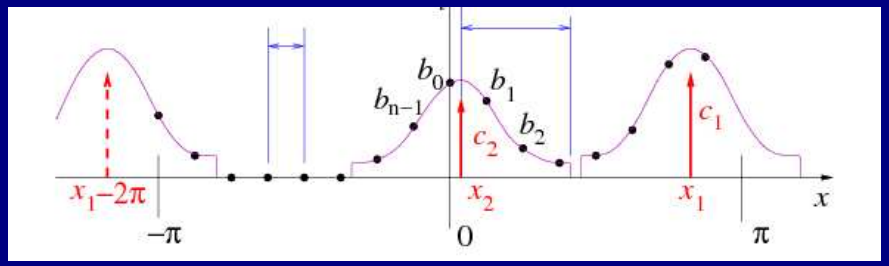
Building a better non-uniform fast Fourier transform

ICERM 3/12/18

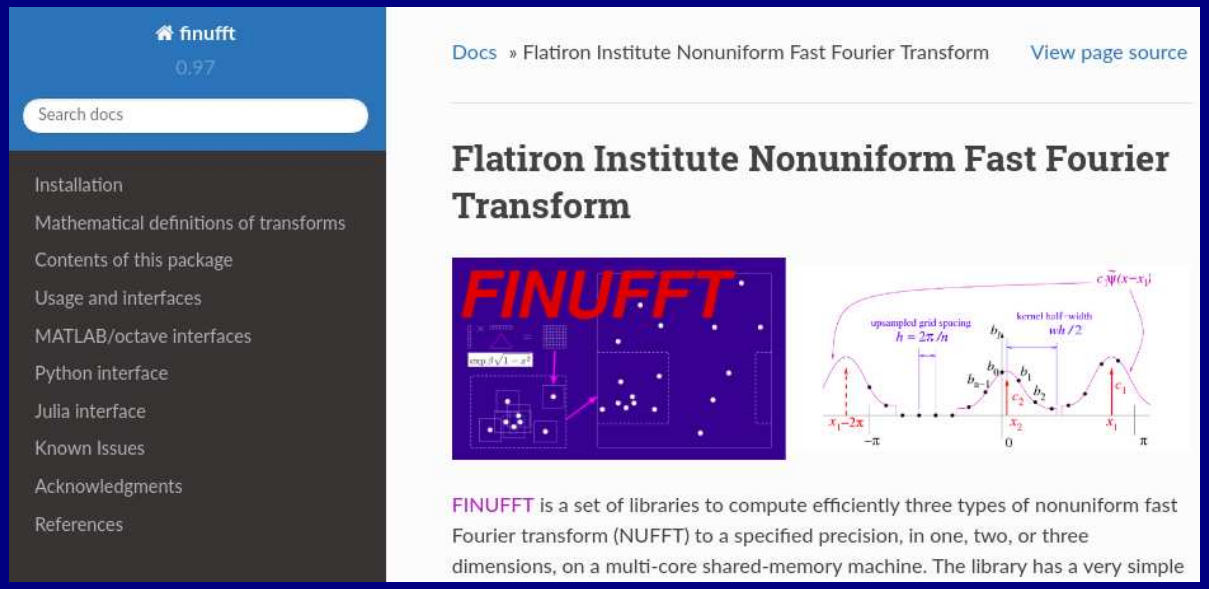
Alex Barnett (Center for Computational Biology, Flatiron Institute)

This work is collaboration with Jeremy Magland.
We benefited from much discussions and/or codes, including:
Leslie Greengard, Ludvig af Klinteberg, Zydrunas Gimbutas, Marina Spivak,
Joakim Andén, and David Stein



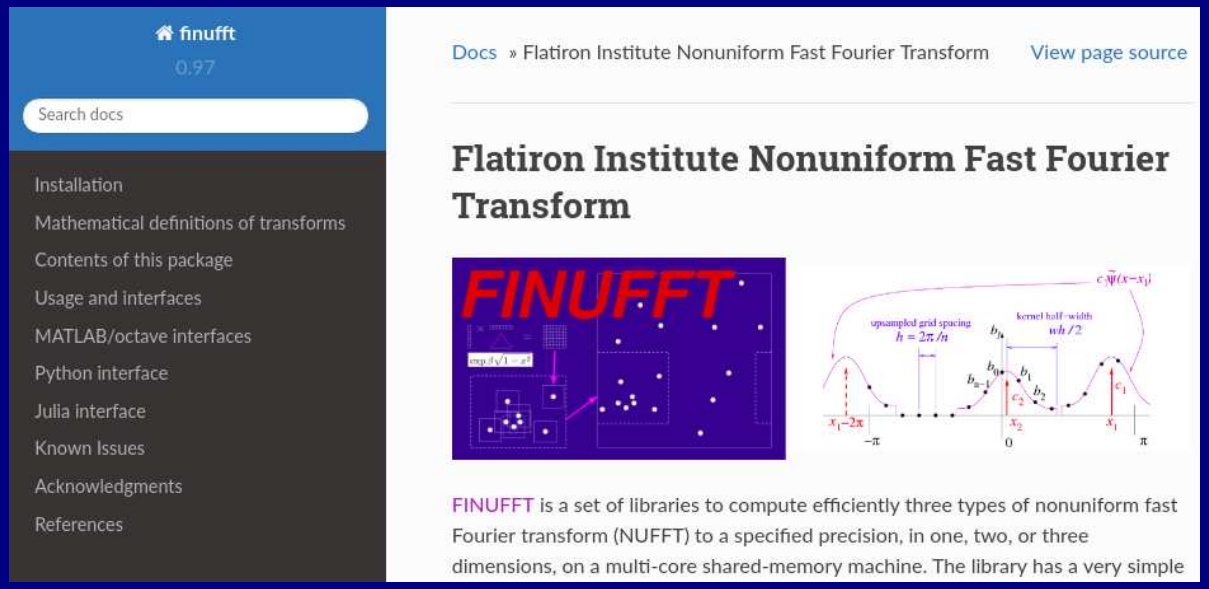
Overview

We are releasing `https://github.com/ahbarnett/finufft`



Overview

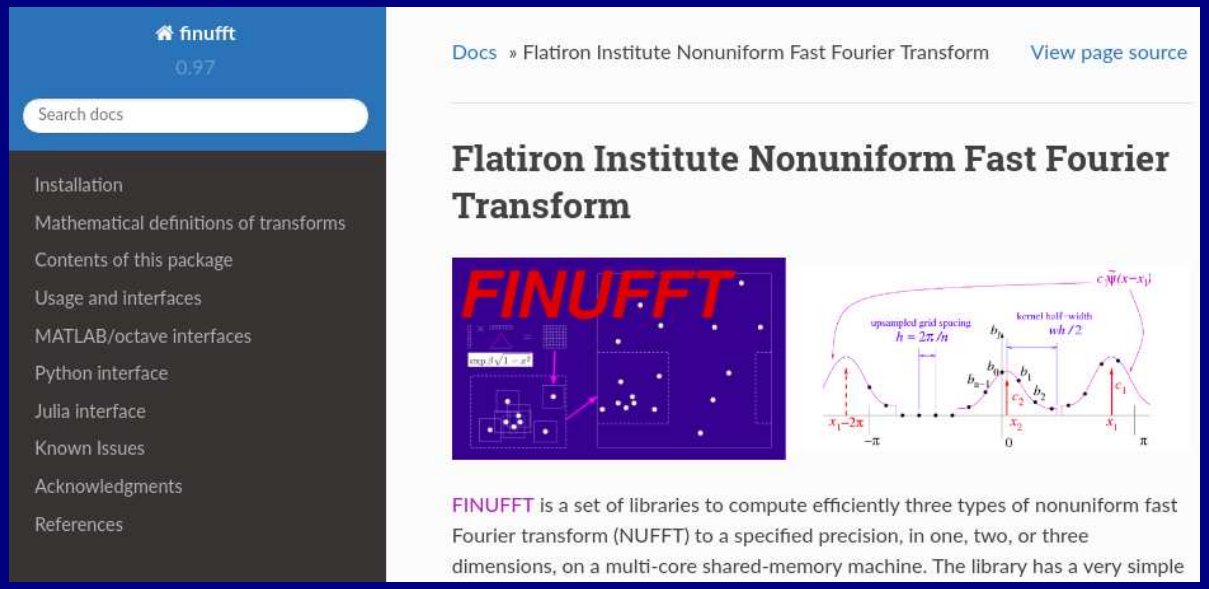
We are releasing `https://github.com/ahbarnett/finufft`



What does it do?

Overview

We are releasing `https://github.com/ahbarnett/finufft`



What does it do?

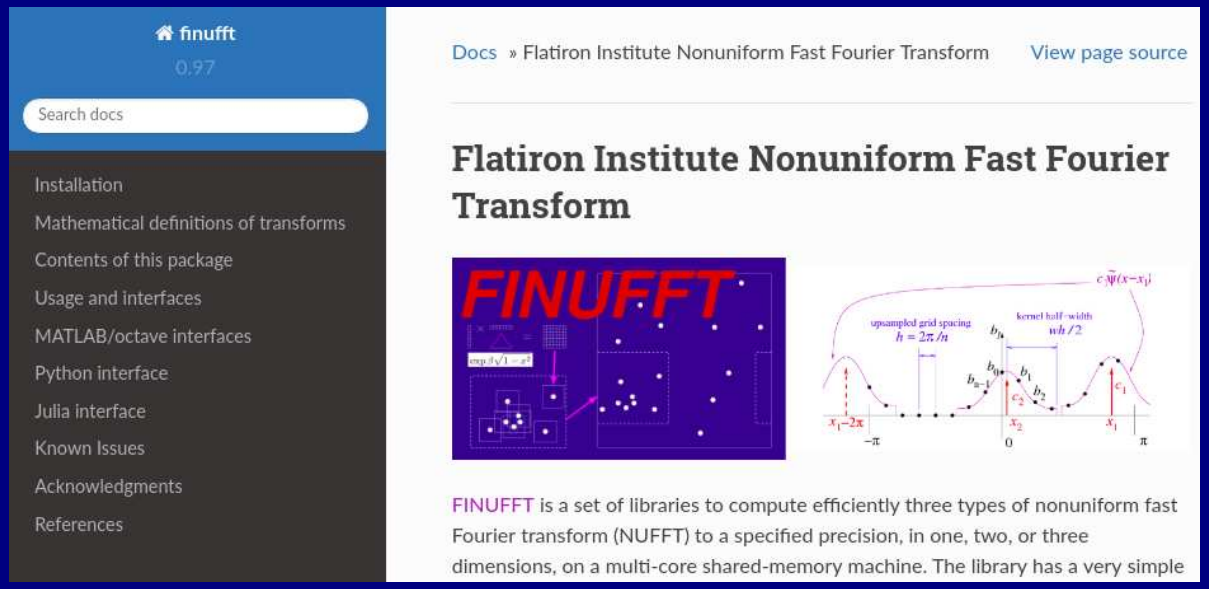
"Fourier analysis of non-uniformly spaced data at close to FFT speeds"

But. . . there already exist libraries?
eg NFFT from Chemnitz (Potts–Keiner–Kunis), NUFFT from NYU (Lee–Greengard)

Ours is faster in large-scale 2D and 3D settings & simpler to use

Overview

We are releasing `https://github.com/ahbarnett/finufft`



What does it do?

"Fourier analysis of non-uniformly spaced data at close to FFT speeds"

But. . . there already exist libraries?
eg NFFT from Chemnitz (Potts–Keiner–Kunis), NUFFT from NYU (Lee–Greengard)

Ours is faster in large-scale 2D and 3D settings & simpler to use

Goals: show some math and engineering behind why, give applications. . .

. . . and explain how “Tex” Logan—one of the best bluegrass fiddle players in the country—is key to the story:

Recap the discrete Fourier transform (DFT)

Task: eval. $$f_k = \sum_{j=0}^{N-1} e^{ik\frac{2\pi j}{N}} c_j \ , \quad -N/2 \le k < N/2$$

FFT is $\mathcal{O}(N \log N)$

Recap the discrete Fourier transform (DFT)

Task: eval. $$f_k = \sum_{j=0}^{N-1} e^{ik\frac{2\pi j}{N}} c_j \ , \quad -N/2 \le k < N/2$$

FFT is $\mathcal{O}(N \log N)$

Uses: 1) Discrete convolution: eg, filtering, PDE solve, regular-sampled data

Recap the discrete Fourier transform (DFT)

Task: eval. $$f_k = \sum_{j=0}^{N-1} e^{ik\frac{2\pi j}{N}} c_j \ , \quad -N/2 \le k < N/2$$ FFT is $\mathcal{O}(N \log N)$

Uses: 1) Discrete convolution: eg, filtering, PDE solve, regular-sampled data

2) Given uniform (U) samples of a smooth 2π-periodic func f,
estimate its Fourier *series* coeffs $\hat{f}_n$, ie so $f(x) = \sum_{n\in\mathbb{Z}} \hat{f}_n e^{-inx}$

Recap the discrete Fourier transform (DFT)

Task: eval. $f_k = \sum_{j=0}^{N-1} e^{ik\frac{2\pi j}{N}} c_j \ , \quad -N/2 \le k < N/2$ FFT is $\mathcal{O}(N \log N)$

Uses: 1) Discrete convolution: eg, filtering, PDE solve, regular-sampled data

2) Given uniform (U) samples of a smooth 2π-periodic func f,
estimate its Fourier *series* coeffs $\hat{f}_n$, ie so $f(x) = \sum_{n\in\mathbb{Z}} \hat{f}_n e^{-inx}$

Vector $\boldsymbol{c}$ of samples, $c_j = \frac{1}{N} f\left(\frac{2\pi j}{N}\right)$

Easy to show its DFT is $f_k = \cdots + \hat{f}_{k-N} + \hat{f}_k + \hat{f}_{k+N} + \hat{f}_{k+2N} + \cdots$

$= \hat{f}_k$ desired $+ \sum_{m\neq 0} \hat{f}_{k+mN}$ aliasing error due to discrete sampling

Recap the discrete Fourier transform (DFT)

Task: eval. $f_k = \sum_{j=0}^{N-1} e^{ik\frac{2\pi j}{N}} c_j \ , \quad -N/2 \le k < N/2$ FFT is $\mathcal{O}(N \log N)$

Uses: 1) Discrete convolution: eg, filtering, PDE solve, regular-sampled data

2) Given uniform (U) samples of a smooth 2π-periodic func f,
estimate its Fourier *series* coeffs $\hat{f}_n$, ie so $f(x) = \sum_{n\in\mathbb{Z}} \hat{f}_n e^{-inx}$

Vector $\boldsymbol{c}$ of samples, $c_j = \frac{1}{N} f\left(\frac{2\pi j}{N}\right)$

Easy to show its DFT is $f_k = \cdots + \hat{f}_{k-N} + \hat{f}_k + \hat{f}_{k+N} + \hat{f}_{k+2N} + \cdots$

$= \hat{f}_k$ desired $+ \sum_{m\neq 0} \hat{f}_{k+mN}$ aliasing error due to discrete sampling

Key: $\boxed{f \text{ smooth} \Leftrightarrow \hat{f}_n \text{ decays for } |n| \text{ large} \Leftrightarrow \text{aliasing error small}}$

eg $(d/dx)^p f$ bounded $\Rightarrow \hat{f}_n = \mathcal{O}(1/|n|^p)$ p-th order convergence of error vs N

Recap the discrete Fourier transform (DFT)

Task: eval. $f_k = \sum_{j=0}^{N-1} e^{ik\frac{2\pi j}{N}} c_j \ , \quad -N/2 \le k < N/2$ FFT is $\mathcal{O}(N \log N)$

Uses: 1) Discrete convolution: eg, filtering, PDE solve, regular-sampled data

2) Given uniform (U) samples of a smooth 2π-periodic func f,
estimate its Fourier *series* coeffs $\hat{f}_n$, ie so $f(x) = \sum_{n\in\mathbb{Z}} \hat{f}_n e^{-inx}$

Vector $\boldsymbol{c}$ of samples, $c_j = \frac{1}{N} f\left(\frac{2\pi j}{N}\right)$

Easy to show its DFT is $f_k = \cdots + \hat{f}_{k-N} + \hat{f}_k + \hat{f}_{k+N} + \hat{f}_{k+2N} + \cdots$

$= \hat{f}_k$ desired $+ \sum_{m\neq 0} \hat{f}_{k+mN}$ aliasing error due to discrete sampling

Key: $\boxed{f \text{ smooth} \ \Leftrightarrow \ \hat{f}_n \text{ decays for } |n| \text{ large} \ \Leftrightarrow \ \text{aliasing error small}}$

eg $(d/dx)^p f$ bounded $\Rightarrow \hat{f}_n = \mathcal{O}(1/|n|^p)$ p-th order convergence of error vs N

3) Estimate power spectrum of *non-periodic* signal on U grid

must first multiply by a "good" *window function*

Contrast: what does NUFFT compute?

Inputs: $\{x_j\}_{j=1}^M$ NU (non-uniform) points in $[0, 2\pi)$
$\{c_j\}_{j=1}^M$ complex strengths, N number of modes

Three flavors of task:

Type-1: NU to U, evaluates $$f_k = \sum_{j=1}^{M} e^{ikx_j} c_j \ , \quad -N/2 \le k < N/2$$

F series coeffs of sum of point masses. Generalizes the DFT (was case $x_j = 2\pi j/N$)

Contrast: what does NUFFT compute?

Inputs: $\{x_j\}_{j=1}^M$ NU (non-uniform) points in $[0, 2\pi)$
$\{c_j\}_{j=1}^M$ complex strengths, N number of modes

Three flavors of task:

Type-1: NU to U, evaluates $$f_k = \sum_{j=1}^{M} e^{ikx_j} c_j \ , \quad -N/2 \le k < N/2$$

F series coeffs of sum of point masses. Generalizes the DFT (was case $x_j = 2\pi j/N$)

Type-2: U to NU, evaluates $$c_j = \sum_{-N/2 \le k < N/2} e^{ikx_j} f_k \ , \quad j = 1, \ldots, M$$

Evaluate usual F series at arbitrary targets. Is adjoint (but not inverse!) of type-1

Contrast: what does NUFFT compute?

Inputs: $\{x_j\}_{j=1}^M$ NU (non-uniform) points in $[0, 2\pi)$
$\{c_j\}_{j=1}^M$ complex strengths, N number of modes

Three flavors of task:

Type-1: NU to U, evaluates $$f_k = \sum_{j=1}^{M} e^{ikx_j} c_j \ , \quad -N/2 \le k < N/2$$

F series coeffs of sum of point masses. Generalizes the DFT (was case $x_j = 2\pi j/N$)

Type-2: U to NU, evaluates $$c_j = \sum_{-N/2 \le k < N/2} e^{ikx_j} f_k \ , \quad j = 1, \ldots, M$$

Evaluate usual F series at arbitrary targets. Is adjoint (but not inverse!) of type-1

Type-3: NU to NU, also needs NU output freqs $\{s_k\}_{k=1}^N$

evaluates $$f_k = \sum_{j=1}^{M} e^{is_k x_j} c_j \ , \quad k = 1, \ldots, N$$ general exponential sum

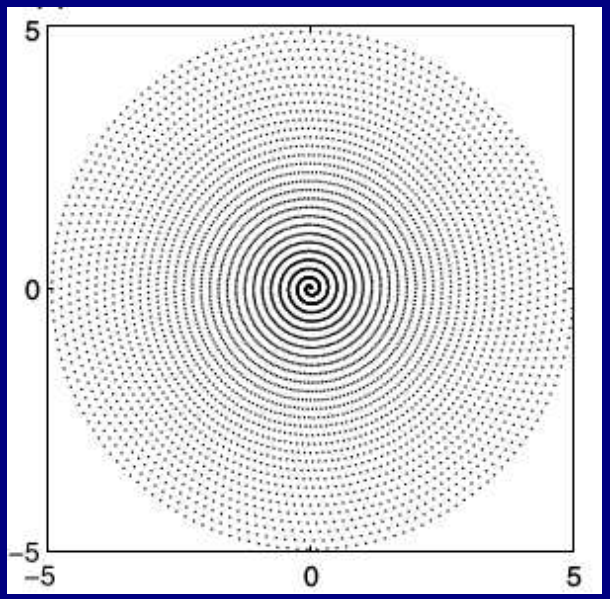
- For dimension $d = 2, 3$ (etc), replace kx_j by $\mathbf{k} \cdot \mathbf{x}_j = k_1 x_j + k_2 y_j$, etc

These tasks crop up a lot

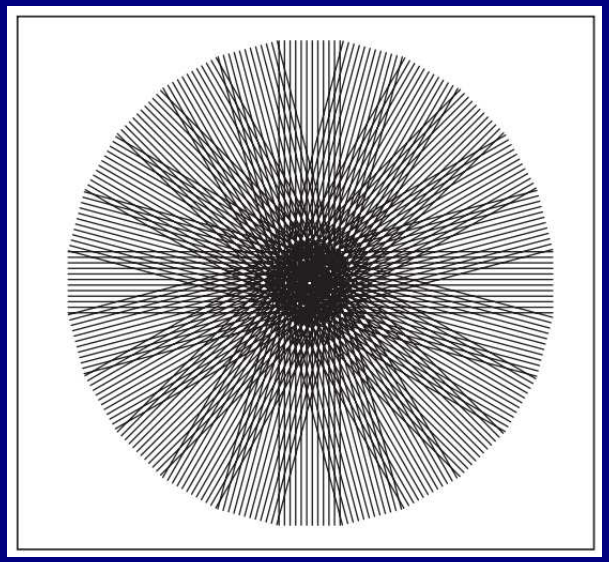
- Magnetic resonance imaging (MRI).

f is unknown 2D image; seek its vector of values $\boldsymbol{f}$ on a U grid

Given data $y_j = \hat{f}(\mathbf{k}_j)$ at NU set of Fourier pts $\mathbf{k}_j$:



spiral imaging



PROPELLER

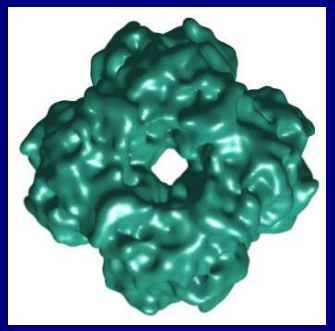
Evaluating the forward model, ie eval. $\mathbf{y} = A\boldsymbol{f}$, is a 2D type-2 NUFFT.

Reconstruct $\boldsymbol{f}$ by iteratively solving this (rect, ill-cond) linear system (eg Fessler)

Even computing a good preconditioner for this lin. sys. needs the NUFFT (Greengard–Inati '06)

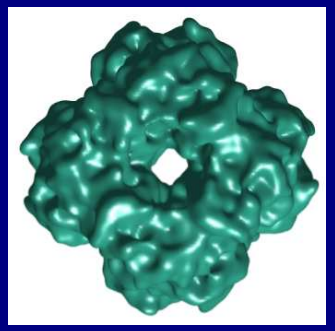
- Cryo electron microscopy (EM) algorithms (how we got into this)

  We need to go between U voxel grid and NU spherical 3D **k**-space reps.



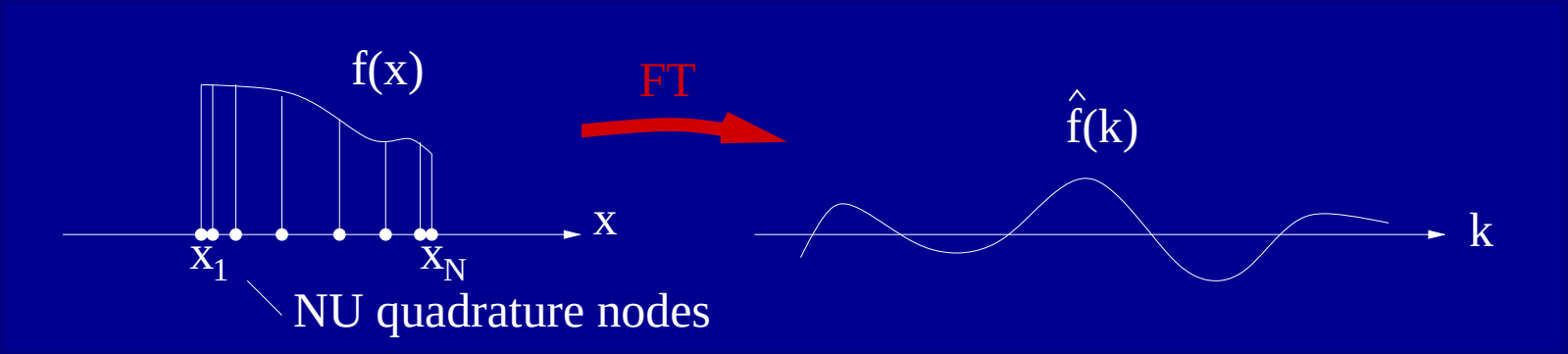
- Cryo electron microscopy (EM) algorithms (how we got into this)

  We need to go between U voxel grid and NU spherical 3D **k**-space reps.



This is an example of...

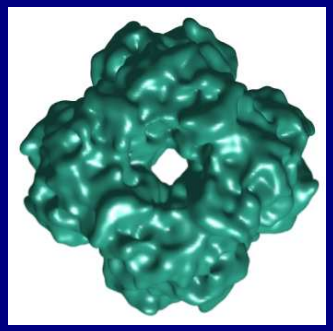
- computing actual Fourier *transforms* of non-smooth $f(\mathbf{x})$ accurately:



Apply a good quadrature rule (eg Gauss) to the Fourier integral $\hat{f}(k) := \int_{-\infty}^{\infty} e^{ikx} f(x) dx$

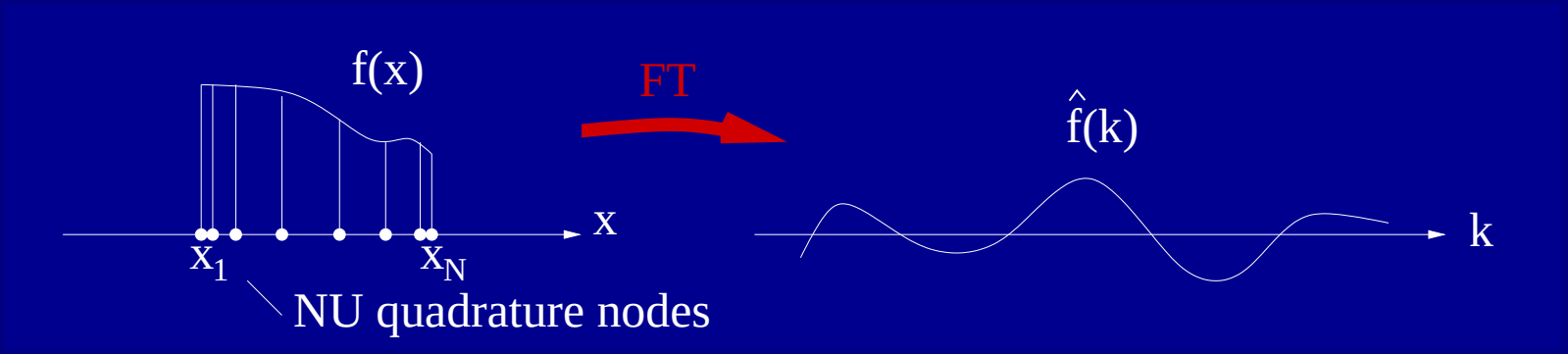
- Cryo electron microscopy (EM) algorithms (how we got into this)

  We need to go between U voxel grid and NU spherical 3D **k**-space reps.



This is an example of...

- computing actual Fourier *transforms* of non-smooth $f(\mathbf{x})$ accurately:



Apply a good quadrature rule (eg Gauss) to the Fourier integral $\hat{f}(k) := \int_{-\infty}^{\infty} e^{ikx} f(x) dx$

- Coherent diffraction imaging again given Fourier data on NU pts (Ewald spheres)
- PDEs: interpolation from U to NU coords grids, applying heat kernels
- Making PDEs, mol. dyn, spatially periodic ("particle-mesh Ewald")
- Given large # of point masses (eg stars), what is Fourier spectrum?

A super-crude 1D type-1 NUFFT: “snap to fine grid”

Three steps:

Set up fine grid on $[0, 2\pi)$, spacing $h = 2\pi/N_f, \quad N_f > N$

A super-crude 1D type-1 NUFFT: "snap to fine grid"

Three steps: Set up fine grid on $[0, 2\pi)$, spacing $h = 2\pi/N_f, \quad N_f > N$

a) Add each strength c_j onto fine-grid cell w/ location $\tilde{x}_j$ nearest x_j.

A super-crude 1D type-1 NUFFT: "snap to fine grid"

Three steps: Set up fine grid on $[0, 2\pi)$, spacing $h = 2\pi/N_f, \quad N_f > N$

a) Add each strength c_j onto fine-grid cell w/ location $\tilde{x}_j$ nearest x_j.

b) Call this vector $\{b_l\}_{l=0}^{N_f-1}$: take its size-N_f FFT to get $\{\hat{b}_k\}_{k=-N_f/2}^{N_f/2-1}$.

A super-crude 1D type-1 NUFFT: "snap to fine grid"

Three steps: Set up fine grid on $[0, 2\pi)$, spacing $h = 2\pi/N_f, \quad N_f > N$

a) Add each strength c_j onto fine-grid cell w/ location $\tilde{x}_j$ nearest x_j.

b) Call this vector $\{b_l\}_{l=0}^{N_f-1}$: take its size-N_f FFT to get $\{\hat{b}_k\}_{k=-N_f/2}^{N_f/2-1}$.

c) Keep just low-freq outputs: $\tilde{f}_k = \hat{b}_k, \quad$ for $-N/2 \leq k < N/2$

A super-crude 1D type-1 NUFFT: "snap to fine grid"

Three steps: Set up fine grid on $[0, 2\pi)$, spacing $h = 2\pi/N_f$, $N_f > N$

a) Add each strength c_j onto fine-grid cell w/ location $\tilde{x}_j$ nearest x_j.

b) Call this vector $\{b_l\}_{l=0}^{N_f-1}$: take its size-N_f FFT to get $\{\hat{b}_k\}_{k=-N_f/2}^{N_f/2-1}$.

c) Keep just low-freq outputs: $\tilde{f}_k = \hat{b}_k, \quad$ for $-N/2 \le k < N/2$

What is error? High freqs $|k| = N/2$ are the worst: relative error thus

$$e^{i\frac{N}{2}x_j} - e^{i\frac{N}{2}\tilde{x}_j} = \mathcal{O}(Nh) = \mathcal{O}(N/N_f)$$

- 1st-order convergent: eg error 10^{-1} needs $N_f \approx 10N$.
- in 3D needs ${N_f}^3 \approx 10^3 N^3$ 1000× slower than plain FFT, for 1-digit accuracy! Terrible!

And yet the idea of dumping onto fine grid is actually good...

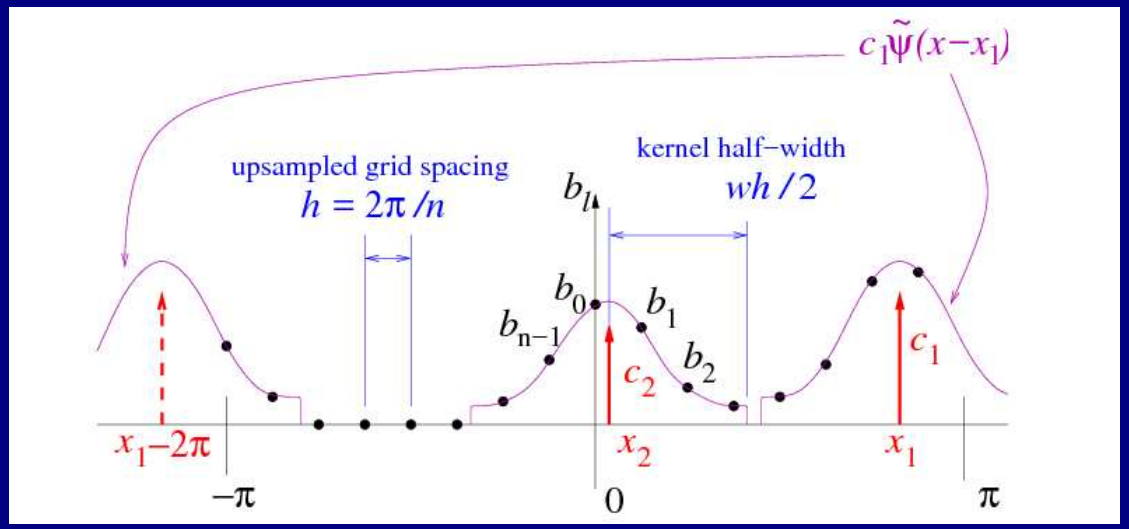
But need much more rapid convergence!

1D type-1 NUFFT algorithm

Three steps: Set up "not-as-fine" grid on $[0, 2\pi)$, $N_f = \sigma N$, upsampling $\sigma \approx 2$

Pick a *spreading kernel* $\psi(x)$ support must be only a few h wide

a) Spread each spike c_j onto fine grid $b_l = \sum_{j=1}^{M} c_j \psi(lh - x_j)$ detail: periodize

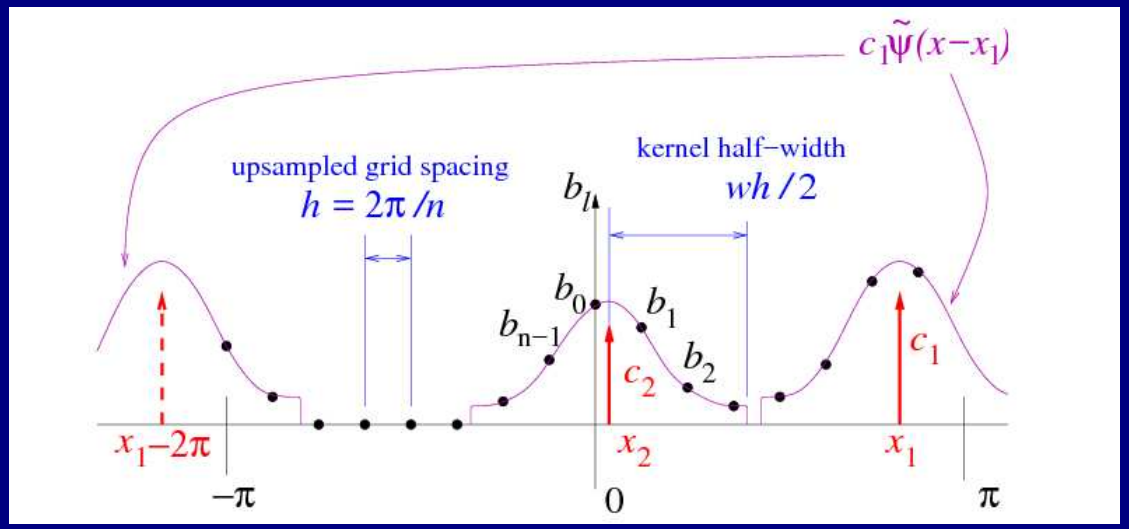


1D type-1 NUFFT algorithm

Three steps: Set up "not-as-fine" grid on $[0, 2\pi)$, $N_f = \sigma N$, upsampling $\sigma \approx 2$

Pick a *spreading kernel* $\psi(x)$ support must be only a few h wide

a) Spread each spike c_j onto fine grid $b_l = \sum_{j=1}^{M} c_j \psi(lh - x_j)$ detail: periodize



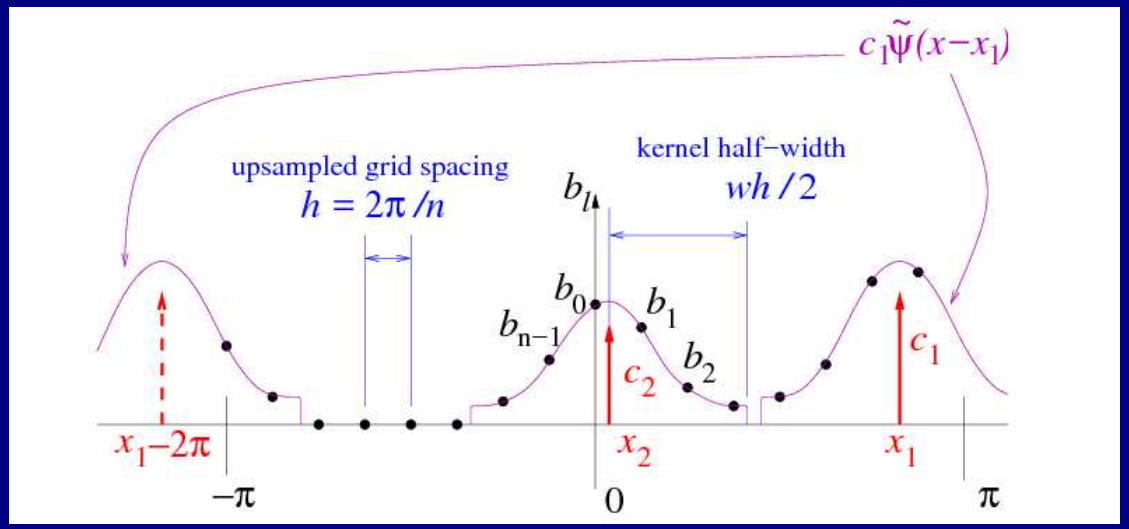
b) Do size-N_f FFT to get $\hat{b}_k$

1D type-1 NUFFT algorithm

Three steps: Set up "not-as-fine" grid on $[0, 2\pi)$, $N_f = \sigma N$, upsampling $\sigma \approx 2$

Pick a *spreading kernel* $\psi(x)$ support must be only a few h wide

a) Spread each spike c_j onto fine grid $b_l = \sum_{j=1}^{M} c_j \psi(lh - x_j)$ detail: periodize



b) Do size-N_f FFT to get $\hat{b}_k$

c) Correct for spreading: $\tilde{f}_k = \frac{1}{\hat{\psi}(k)} \hat{b}_k$, for $-N/2 \le k < N/2$

Why? since you convolved sum of point masses $\sum_{j=1}^{M} c_j \delta(x - x_j)$ with $\psi(x)$,

undo by deconvolving: dividing by kernel in Fourier domain

Type-2 similar; type-3 needs more upsampling (by σ^2 not σ)

What makes a good kernel $\psi(x)$?

A) ψ should have small support. Why? Spreading costs $\mathcal{O}(w^d M)$ flops

kernel is w grid pts wide

What makes a good kernel $\psi(x)$?

A) ψ should have small support. Why? Spreading costs $\mathcal{O}(w^d M)$ flops

kernel is w grid pts wide

B) ψ should be smooth. Why? So $\sum_{j=1}^{M} c_j \psi(x - x_j)$ is also

recall smooth $\rightarrow$ small DFT aliasing error

What makes a good kernel $\psi(x)$?

A) ψ should have small support. Why? Spreading costs $\mathcal{O}(w^d M)$ flops

kernel is w grid pts wide

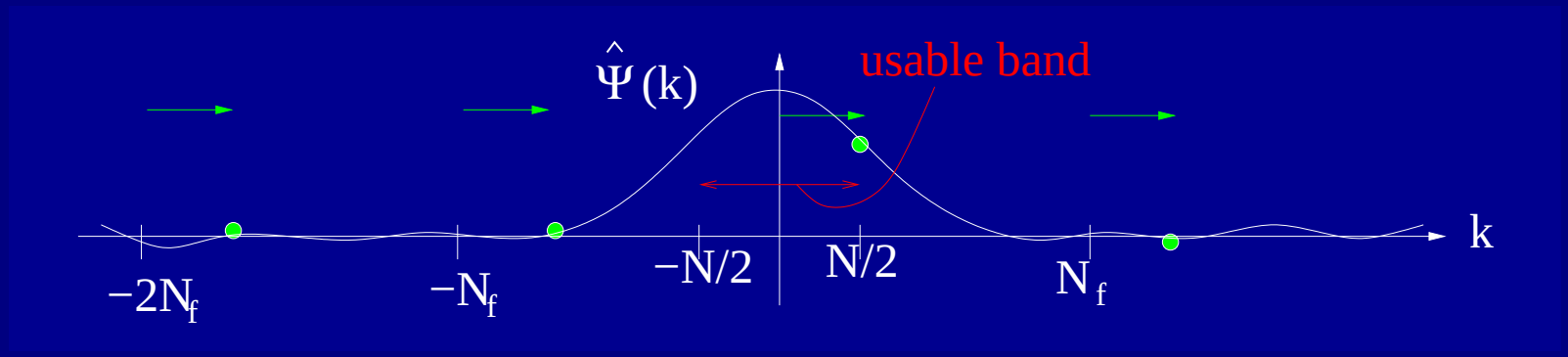
B) ψ should be smooth. Why? So $\sum_{j=1}^{M} c_j \psi(x - x_j)$ is also

recall smooth $\rightarrow$ small DFT aliasing error

A) and B) are conflicting requirements :(

Rigorous error analysis: $\quad |\tilde{f}_k - f_k| \leq \epsilon \|\boldsymbol{c}\|_1$

$$\text{where} \quad \epsilon = \max_{|k| \leq N/2, x \in \mathbb{R}} \frac{1}{|\hat{\psi}(k)|} \Big| \sum_{m \neq 0} \hat{\psi}(k + mN_f) e^{i(k+mN_f)x} \Big|$$



want $\hat{\psi}$ large in $|k| < N/2$, small for $|k| > N_f - N/2$

(Partial) history of the NUFFT

- Interpolation of F series to NU pts, astrophysical (Boyd '80s, Press–Rybicki '89)
- Gaussian kernel case $\psi(x) = e^{-\alpha x^2}$ (Dutt–Rokhlin '93, Elbel–Steidl '98)

  rigorous proof of exponential convergence vs w, ie # digits $= \log_{10}(1/\epsilon) \approx 0.5w$
- Realization there's a close-to-optimal kernel ("Kaiser–Bessel") (Jackson '91)

  nearly twice the convergence rate: $\log_{10}(1/\epsilon) \approx 0.9w$

  rigorous analysis (Fourmont '99, Fessler '02, Potts–Kunis... '02)
- Fast gridding for Gaussian case by cutting e^x evals (Greengard–Lee '04)
- Low-rank factorization version (Ruiz-Antolín–Townsend '16)

  uses $\sim w^d$ (not-upsampled!) FFT calls
- Simpler kernel, same rate as K–B, rigorous analysis (B–Magland)

(Partial) history of the NUFFT

- Interpolation of F series to NU pts, astrophysical (Boyd '80s, Press–Rybicki '89)
- Gaussian kernel case $\psi(x) = e^{-\alpha x^2}$ (Dutt–Rokhlin '93, Elbel–Steidl '98)

  rigorous proof of exponential convergence vs w, ie # digits $= \log_{10}(1/\epsilon) \approx 0.5w$
- Realization there's a close-to-optimal kernel ("Kaiser–Bessel") (Jackson '91)

  nearly twice the convergence rate: $\log_{10}(1/\epsilon) \approx 0.9w$

  rigorous analysis (Fourmont '99, Fessler '02, Potts–Kunis. . . '02)
- Fast gridding for Gaussian case by cutting e^x evals (Greengard–Lee '04)
- Low-rank factorization version (Ruiz-Antolín–Townsend '16)

  uses $\sim w^d$ (not-upsampled!) FFT calls
- Simpler kernel, same rate as K–B, rigorous analysis (B–Magland)

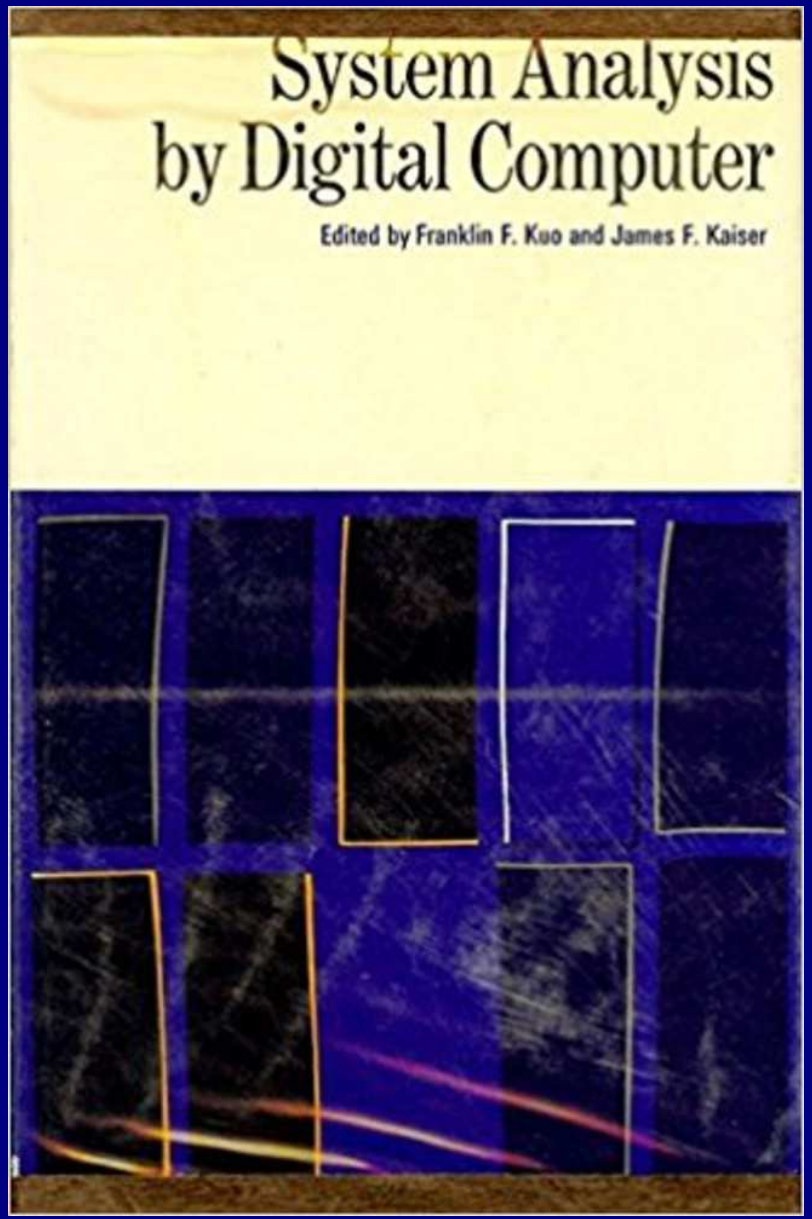
But what on earth is Kaiser–Bessel ?

Turns out requirements A) and B) v. close to those for good *window funcs*

Recall a window func. designed to make non-periodic signal pretend to be periodic. . .

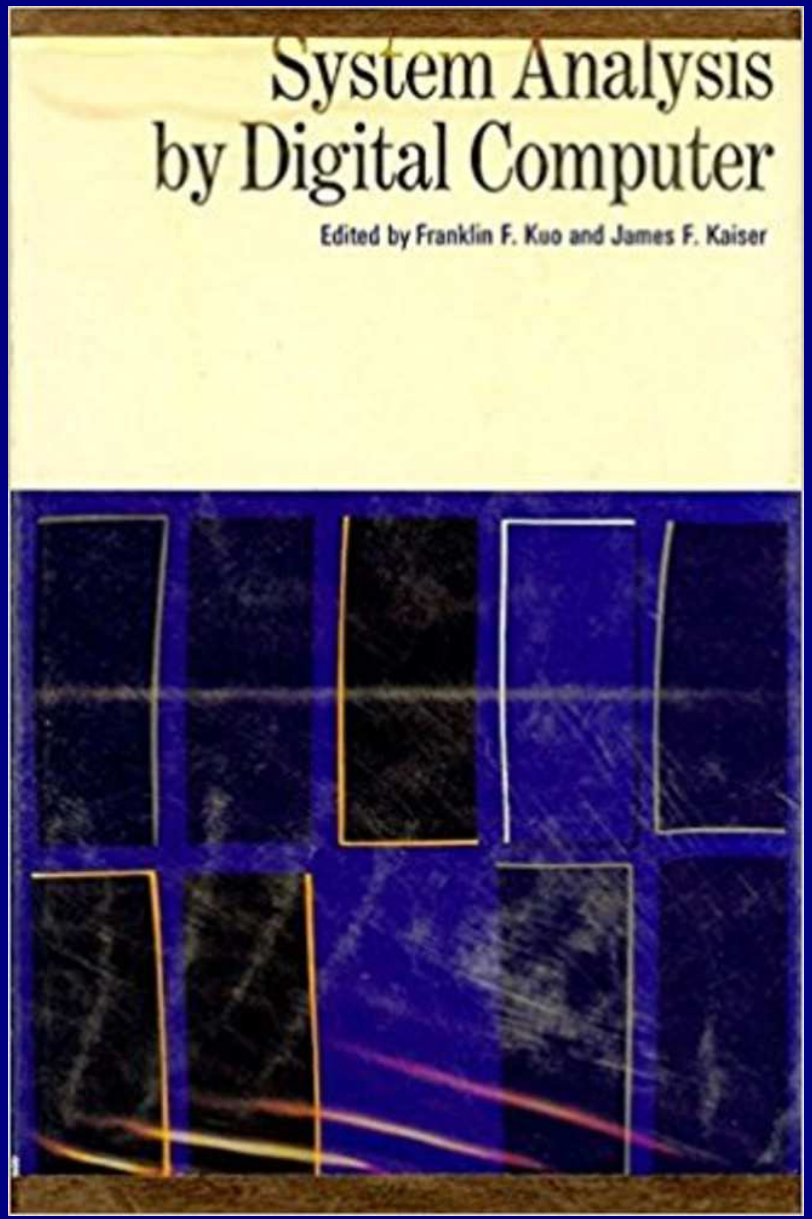
Story of the “Kaiser–Bessel” kernel

People cite this obscure 1966 book for Kaiser–Bessel:



Story of the “Kaiser–Bessel” kernel

People cite this obscure 1966 book for Kaiser–Bessel:



(had to buy 2nd-hand) Let’s open. . .

First appearance of Kaiser–Bessel in print

$$H_1^*(z) = a_o w(o) + \tfrac{1}{2} \sum_{n=1}^{N} [a_n w(nT)][z^n + z^{-n}], \quad m \text{ even} \tag{7.15}$$

or

$$H_1^*(z) = \tfrac{1}{2} \sum_{n=1}^{N} [b_n w(nT)][z^n - z^{-n}], \quad m \text{ odd} \tag{7.16}$$

where N is the greatest integer in (τ/T).

An especially simple weighting function which may be used is Hamming's [13, p. 95-99] window function defined by

$$w_h(t) = \begin{cases} 0.54 + 0.46 \cos(\pi t/\tau), & |t| < \tau \\ 0 & , |t| > \tau \end{cases} \tag{7.17}$$

For this function 99.96% of its energy lies in the band $|\omega| \leq 2\pi/\tau$ with the peak amplitude of the side lobes of $W_h(j\omega)$ being less than 1% of the peak.

Other specific window functions [38, 39] may be used to advantage. One especially flexible *family* of weighting functions with nearly optimum† characteristics is given by the Fourier cosine transform pair [40]

$$w(t) = \begin{cases} \dfrac{I_o\left[\omega_a \sqrt{\tau^2 - t^2}\right]}{I_o(\omega_a \tau)} & |t| < \tau \\ 0 & |t| > \tau \end{cases} \tag{7.18}$$

† This family of window functions was "discovered" by Kaiser in 1962 following a discussion with B. F. Logan of the Bell Telephone Laboratories.

$$W(j\omega) = \frac{2}{I_o(\omega_a \tau)} \frac{\sin\left[\tau \sqrt{\omega^2 - \omega_a^2}\right]}{\sqrt{\omega^2 - \omega_a^2}} \tag{7.19}$$

where I_0 is the modified Bessel function of the first kind and order zero. By varying the product $\omega_a \tau$ the energy in the central lobe and the amplitudes of the side lobes can be changed. The usual range on values of $\omega_a \tau$ is $4 < \omega_a \tau < 9$ corresponding to a range of side lobe peak heights of 3.1% down to 0.047%. Figure 7.7 shows w(t) and $W(j\omega)$ for $\omega_a \tau = 6.0$ and 8.5.

The flexibility of this set of window functions is now illustrated. Setting $\omega_a \tau = \pi\sqrt{3} = 5.4414$ places the first zero of $W(j\omega)$ at 2π on a normalized scale, the same location as that of the cosine transform of the Hamming window [13, p. 95-99]. The closeness of the resulting windows is immediately apparent. The Hamming window, while having a slightly lower peak amplitude on the first two side lobes, continues to oscillate approximately sinusoidally with a slowly diminishing amplitude. On the other hand the I_0-sinh window with $\omega_a \tau = \sqrt{3}\pi$ has a slightly greater magnitude for the first two side lobes but the amplitude of the sinusoidal oscillations of the window tails diminishes much more rapidly. The Hamming window has all but 0.037% of its energy in the main lobe while the I_0-sinh window with $\omega_a \tau = \sqrt{3}\pi$ has all but 0.012% of its energy in the central lobe.

As a second example, the Blackman window, his "not very serious proposal" [13, p. 98], has its first zero at 3π and corresponds to the I_0-sinh window with $\omega_a \tau = 2\sqrt{2}\pi = 8.885$. The nearly equivalent value of $\omega_a \tau = 8.50$ can be used for comparison. This function has its first zero at $2\pi(1.44226)$ rather than at 3π. Again the closeness of these two window functions can be observed by comparing our Fig. 7.7c with

What does: *"discovered" by Kaiser following a discussion with B. F. Logan* mean?

Kaiser–Bessel Fourier transform pair

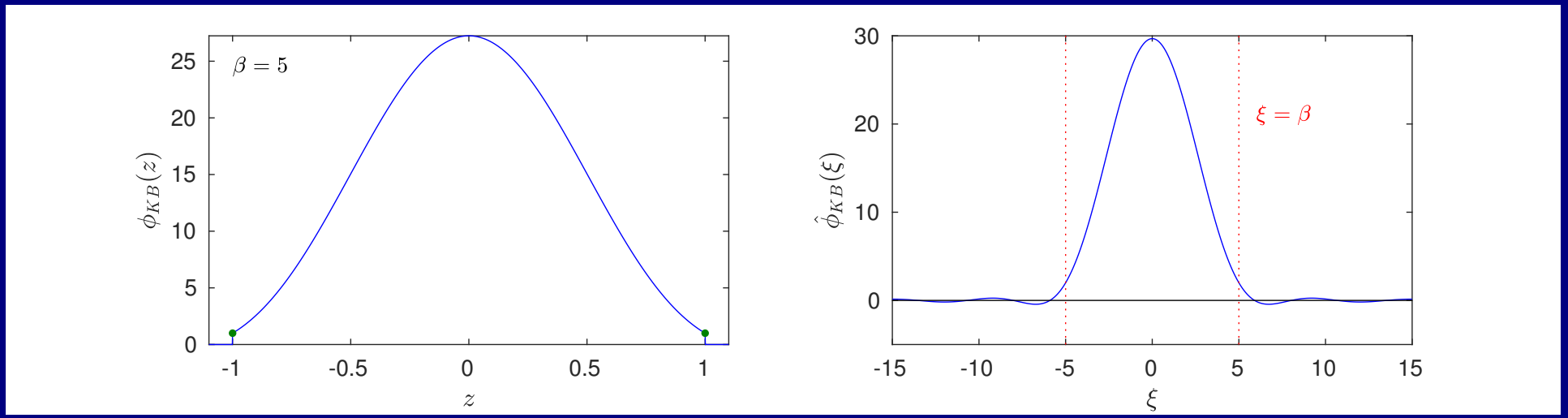
The truncated kernel

$$\phi_{KB}(z) \;:=\; \begin{cases} I_0(\beta\sqrt{1-z^2}), & |z| \le 1 \\ 0, & \text{otherwise} \end{cases}$$

we scale $z := 2x/wh$

is the FT of

$$\hat{\phi}_{KB}(\xi) \;=\; 2\frac{\sinh\sqrt{\beta^2-\xi^2}}{\sqrt{\beta^2-\xi^2}}$$



Still unknown to Gradshteyn–Ryzhik, Bateman, Prudnikov, Wolfram ®, Maple ® . . .

Over to James Kaiser, Bell Labs (interviewed in ’97)

About 1960-’61, Henry Pollak, who was department head in the math research area at Bell Labs, and two of his staff, Henry Landau, and Dave Slepian, solved the problem of finding that set of functions that had maximum energy in the main lobe consistent with certain roll-offs in the side lobes.

Over to James Kaiser, Bell Labs (interviewed in ’97)

About 1960-’61, Henry Pollak, who was department head in the math research area at Bell Labs, and two of his staff, Henry Landau, and Dave Slepian, solved the problem of finding that set of functions that had maximum energy in the main lobe consistent with certain roll-offs in the side lobes.

They showed that the functions that came out of that were the prolate spheroidal wave functions . . . The program was about 600 lines of FORTRAN.

One of the nice features of Bell Laboratories is there were a lot very bright people around, and one of the fellows that I always enjoyed talking to was Ben Logan, a tall Texan from Big Spring, Texas.

Over to James Kaiser, Bell Labs (interviewed in ’97)

About 1960-’61, Henry Pollak, who was department head in the math research area at Bell Labs, and two of his staff, Henry Landau, and Dave Slepian, solved the problem of finding that set of functions that had maximum energy in the main lobe consistent with certain roll-offs in the side lobes.

They showed that the functions that came out of that were the prolate spheroidal wave functions . . . The program was about 600 lines of FORTRAN.

One of the nice features of Bell Laboratories is there were a lot very bright people around, and one of the fellows that I always enjoyed talking to was Ben Logan, a tall Texan from Big Spring, Texas.

https://www.youtube.com/watch?v=NU4l7xFhQdA&t=72s

Back to Kaiser…

So one day I went in Ben's office and his chalkboard was just filled with equations. Way down in the left-hand corner of Ben's chalkboard was this transform pair, the I_0–sinh transform pair. I didn't know what I_0 was, I said, "Ben, what's I_0?" He came back with "Oh, that's the modified Bessel function of the first kind and order zero." I said, "Thanks a lot, Ben, but what is that?" He said, "You know, it's just a basic Bessel function but with purely imaginary argument." So I copied down the transform pair and went back to my office.

Back to Kaiser…

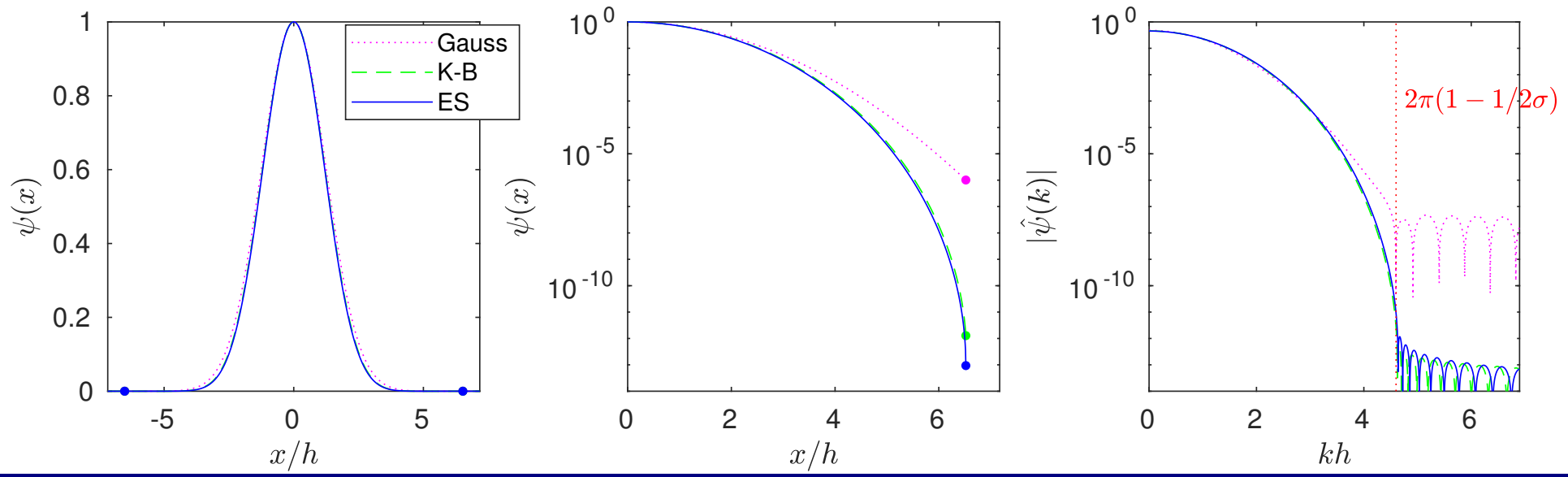
So one day I went in Ben's office and his chalkboard was just filled with equations. Way down in the left-hand corner of Ben's chalkboard was this transform pair, the I_0–sinh transform pair. I didn't know what I_0 was, I said, "Ben, what's I_0?" He came back with "Oh, that's the modified Bessel function of the first kind and order zero." I said, "Thanks a lot, Ben, but what is that?" He said, "You know, it's just a basic Bessel function but with purely imaginary argument." So I copied down the transform pair and went back to my office.

I wrote a program … got the data back and when I compared the I_0 function to the prolate, I said, "What's going on here? They look almost identical!" The answers were within about a tenth of a percent of one another. One program [PSWF] required 600 lines of code and the other ten or twelve lines of code!

P.S. we now have a kernel needing < 1 line of code …

Compare the kernels

Plot the kernels for support of $w = 13$ fine grid points:



- very hard to distinguish on linear plot! Decays differ on log plot
- Kaiser–Bessel: tail of FT is at 10^{-12}
- best truncated Gaussian has tail only at 10^{-7}
- "ES" is our new kernel; v. close to KB

Our new ES ("exp of sqrt") kernel

$$\psi_{ES}(x) := e^{\beta\sqrt{1-z^2}}, \quad z := 2x/wh \in [-1, 1], \quad \text{zero otherwise.}$$

(found via numerical tinkering: simplifying the I_0)

Our new ES ("exp of sqrt") kernel

$$\psi_{ES}(x) := e^{\beta\sqrt{1-z^2}}, \quad z := 2x/wh \in [-1,1], \quad \text{zero otherwise.}$$

(found via numerical tinkering: simplifying the I_0)

- its Fourier transform $\hat{\psi}_{ES}$ has *no known formula*

1) Numerical consequence: use quadrature on FT to eval. $\frac{1}{\hat{\psi}_{ES}}$ for step c)

Our new ES ("exp of sqrt") kernel

$$\psi_{ES}(x) := e^{\beta\sqrt{1-z^2}}, \quad z := 2x/wh \in [-1,1], \quad \text{zero otherwise.}$$

(found via numerical tinkering: simplifying the I_0)

- its Fourier transform $\hat{\psi}_{ES}$ has *no known formula*

1) Numerical consequence: use quadrature on FT to eval. $\frac{1}{\hat{\psi}_{ES}}$ for step c)

2) Analytic consequence: one has to work with the FT integral directly...

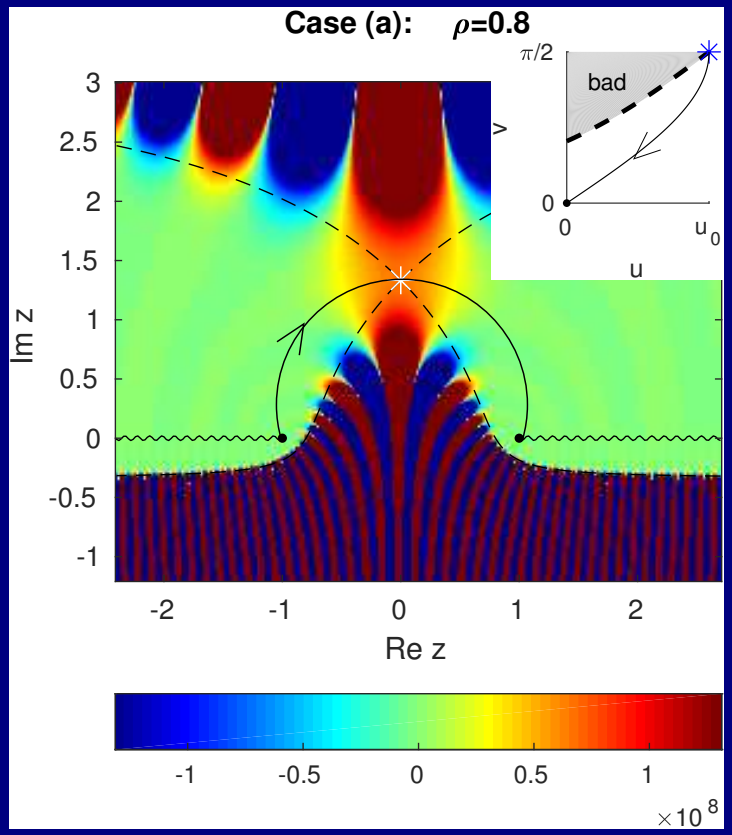
We prove essentially $\quad \epsilon = \mathcal{O}\big(\sqrt{w}e^{-\pi w\sqrt{1-1/\sigma}}\big) \quad$ as kernel width $w \to \infty$

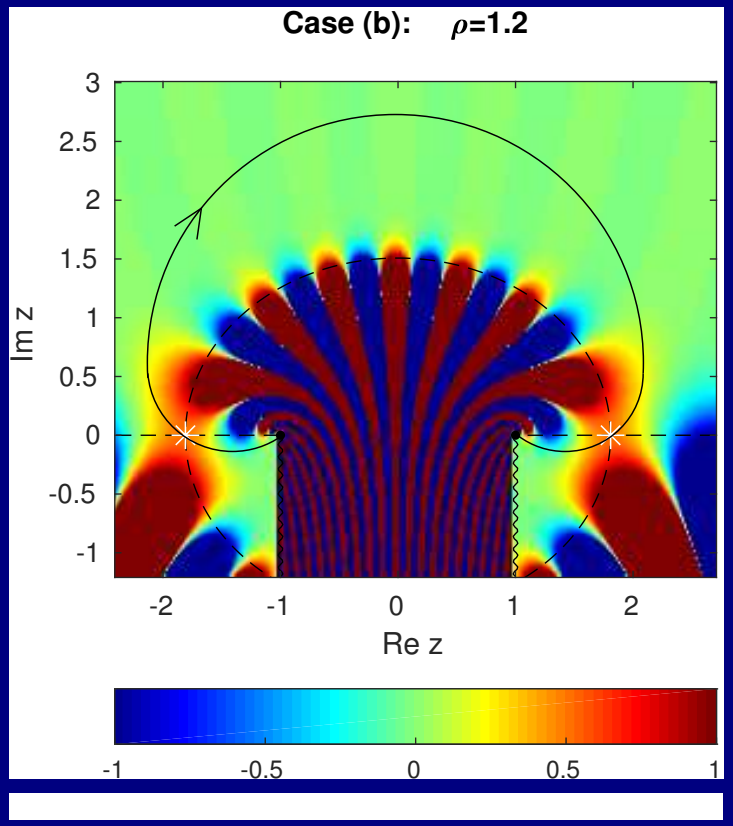
- same exponential convergence rate as Kaiser–Bessel, and as PSWF (Fuchs '64)
- consequence: $\quad w \approx 7$ gives accuracy $\epsilon = 10^{-6}$, $\quad w \approx 13$ gets $\epsilon = 10^{-12}$.
- However, evaluation now requires only one sqrt, one e^x, couple of mults.
- proof is 8 pages: contour integrals split into parts, sums into various parts, bounding the conditionally-convergent tail sum...

One proof ingredient

Asymptotics (in β) of the Fourier transform $\hat{\psi}(\rho\beta) = \int_{-1}^{1} e^{\beta(\sqrt{1-z^2}-i\rho z)} dz$

via deforming to complex plane, steepest descent (saddle pts) (Olver)

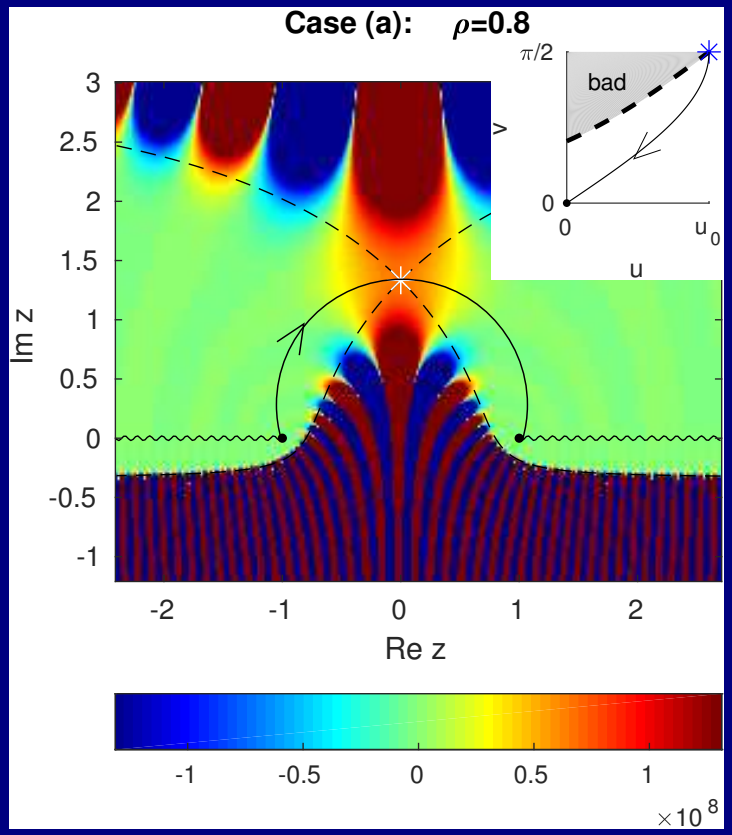


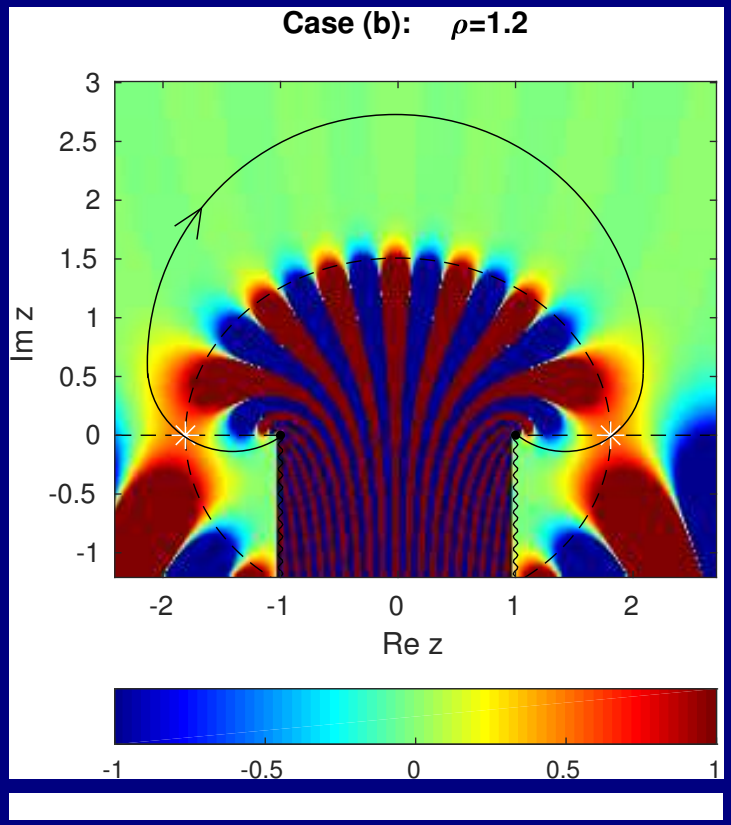


One proof ingredient

Asymptotics (in β) of the Fourier transform $\hat{\psi}(\rho\beta) = \int_{-1}^{1} e^{\beta(\sqrt{1-z^2}-i\rho z)} dz$

via deforming to complex plane, steepest descent (saddle pts) (Olver)





Summary: new approx. FT pair

K–B & PSWF also may be interpreted this way

" exp(semicircle) $\xleftrightarrow{\text{FT}}$ exp(semicircle) + exponentially small tail "

Implementation aspects

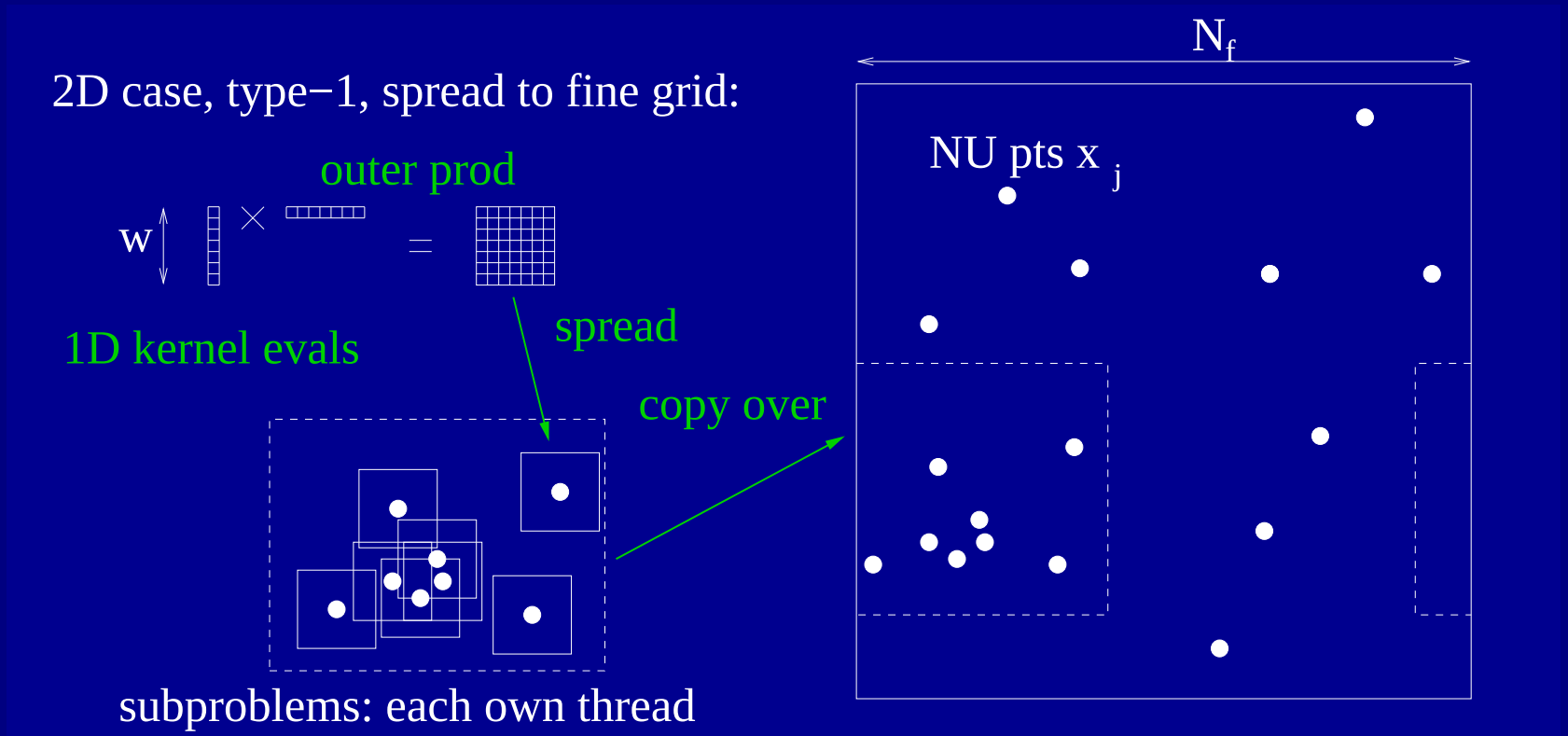
- Type 1,2,3 for dimensions $d = 1, 2, 3$: nine routines
- C++/OpenMP/SIMD, shared mem, calls FFTW. Apache license
- Wrappers to C, Fortran, MATLAB, octave, python, julia

Implementation aspects

- Type 1,2,3 for dimensions $d = 1, 2, 3$: nine routines
- C++/OpenMP/SIMD, shared mem, calls FFTW. Apache license
- Wrappers to C, Fortran, MATLAB, octave, python, julia
- Cache-aware multithreaded spreading:

Type-2 easy: parallelize over *bin-sorted* NU pts no collisons reading from U blocks

Type-1 not so: writes collide load-balancing, slow index-wrapping, $\leq 10^4$ NU pts per subprob:



Performance: 3D Type-1 (the most dramatic)

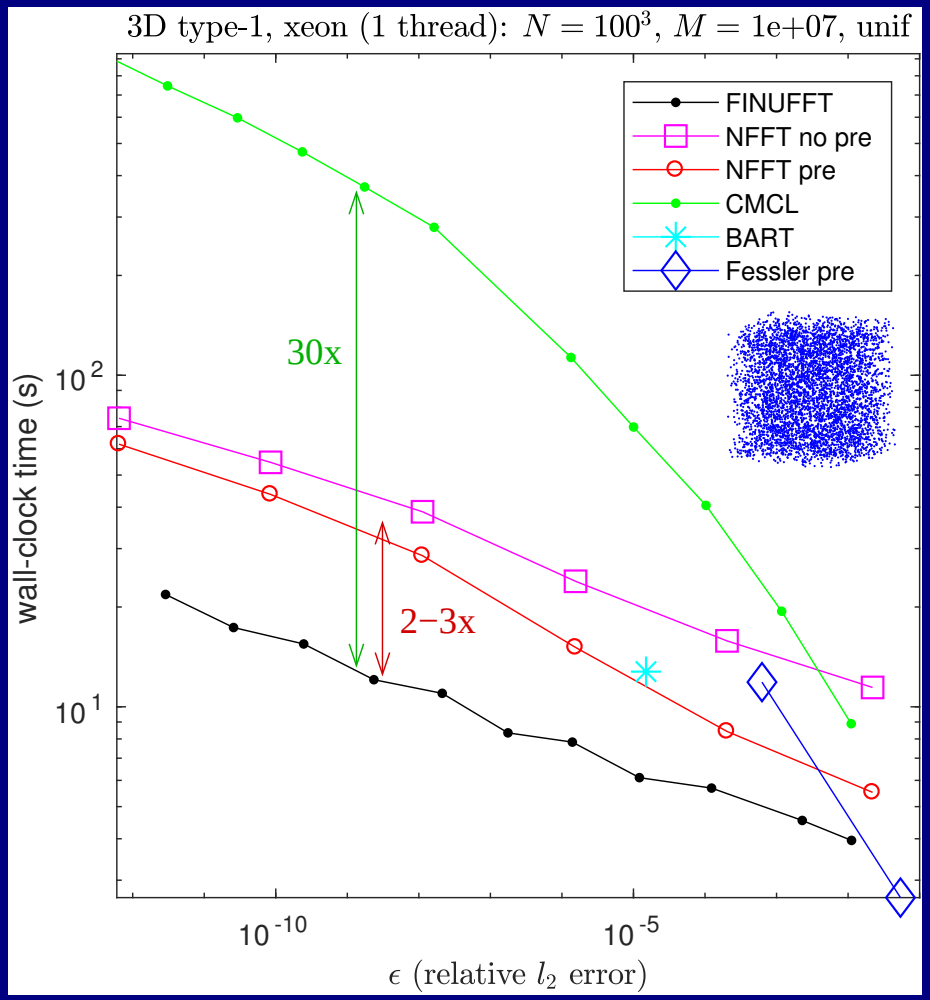
Compare FINUFFT to

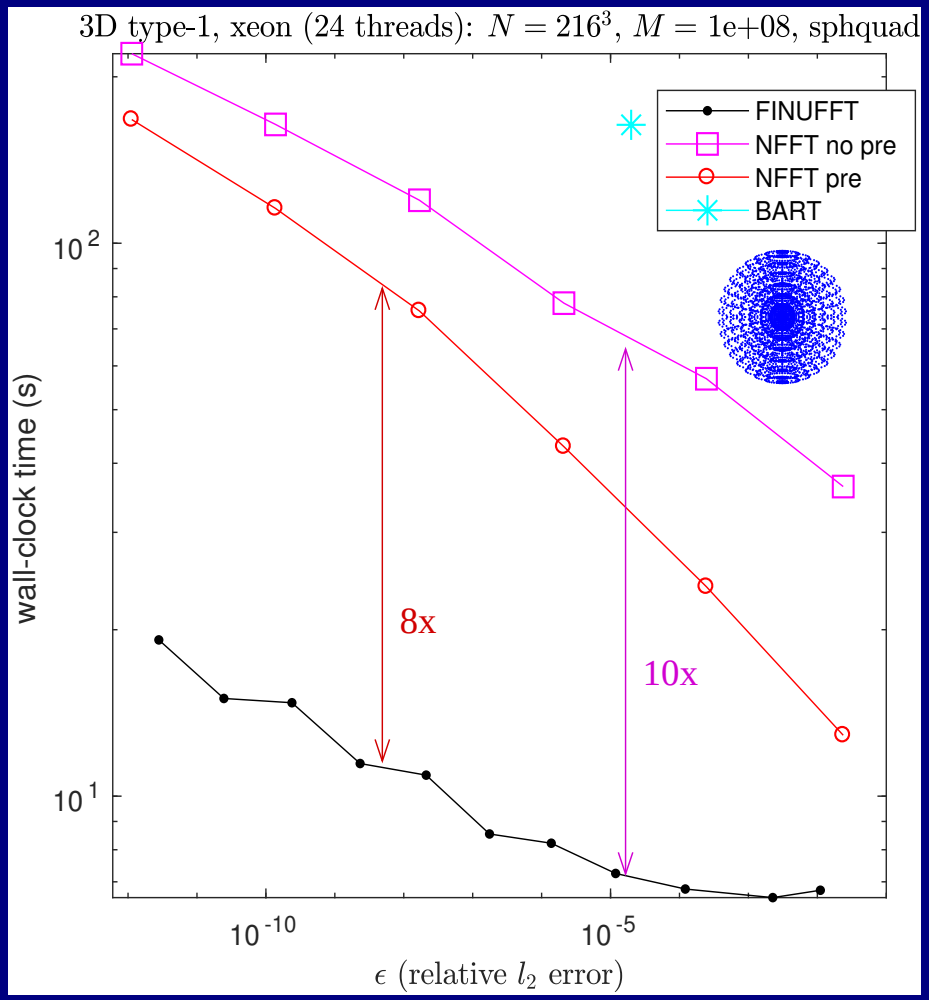
- CMCL NUFFT (single-threaded, Gaussian kernel)
- NFFT (multi-threaded, "backwards" Kaiser–Bessel)

ie they eval. sinch $\sqrt{1-x^2}$

Performance: 3D Type-1 (the most dramatic)

Compare FINUFFT to
- CMCL NUFFT (single-threaded, Gaussian kernel)
- NFFT (multi-threaded, "backwards" Kaiser–Bessel) ie they eval. sinch $\sqrt{1-x^2}$

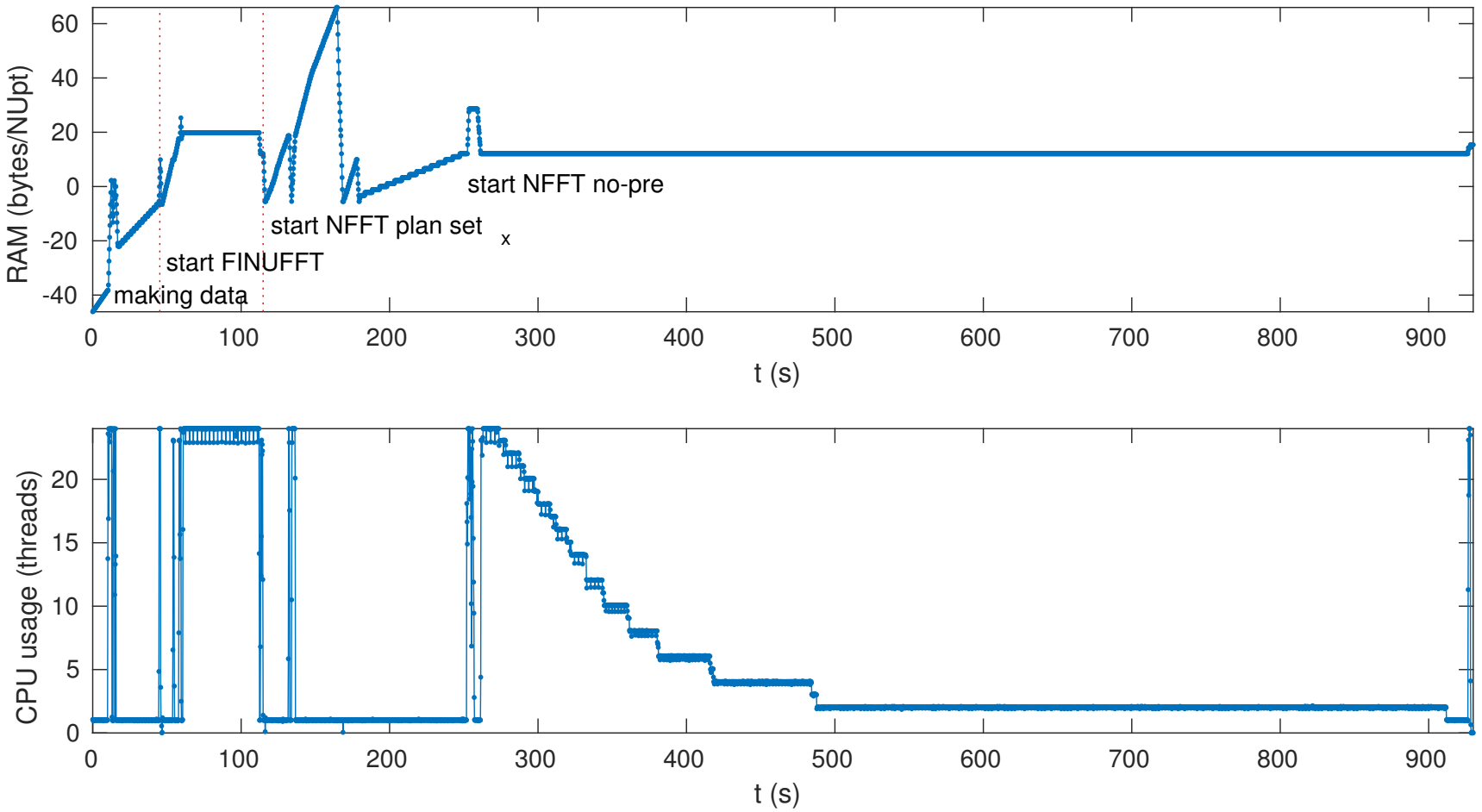




- all scale as $\mathcal{O}(M|\log\epsilon|^d + N\log N)$; it's about prefactors and RAM usage
- at $M = 10^8$: we need only 2 GB, vs NFFT pre needs 60 GB at high acc.

3D Type-1: RAM & CPU usage for non-uniform density

We use all threads efficiently, vs NFFT assigns threads to fixed x-slices:



done from MATLAB via `https://github.com/ahbarnett/memorygraph`

Conclusions

NUFFT is a key tool with many scientific computing applications

We speed up and simplify the NUFFT using...

- mathematics: creation and rigorous analysis of new kernel func ψ
- no analytic $\hat{\psi}$ need be known: instead use numerical quadrature
- cache-aware and thread-balanced implementation

Conclusions

NUFFT is a key tool with many scientific computing applications

We speed up and simplify the NUFFT using...

- mathematics: creation and rigorous analysis of new kernel func ψ
- no analytic $\hat{\psi}$ need be known: instead use numerical quadrature
- cache-aware and thread-balanced implementation

Result: FINUFFT (Flatiron Institute Non-Uniform Fast Fourier Transform)

`https://github.com/ahbarnett/finufft`

fast, simple to install and use. Send me bug reports & feature req's

Future:

- GPU spreader (build upon promising work of: Kunis–Kunis '12, Ou '17)
- math: "why" are PSWF and K–B so close to $e^{\beta\sqrt{1-z^2}}$? no, it's not WKB...

EXTRA SLIDES

Ongoing: Intel vector optimizations

Vector intrinsics accelerate by up to 2×: (Ludvig af Klinteberg)

```
{
  out[0] = 0.0; out[1] = 0.0;
#if defined(VECT) && !defined(SINGLE)
   __m128d vec_out = _mm_setzero_pd();
#endif
  if (i1>=0 && i1+ns<=N1 && i2>=0 && i2+ns<=N2 && i3>=0 && i3+ns<=N3) {
    // no wrapping: avoid ptrs
    for (int dz=0; dz<ns; dz++) {
      BIGINT oz = N1*N2*(i3+dz);        // offset due to z
      for (int dy=0; dy<ns; dy++) {
        BIGINT j = oz + N1*(i2+dy) + i1;
        FLT ker23 = ker2[dy]*ker3[dz];
        for (int dx=0; dx<ns; dx++) {
          FLT k = ker1[dx]*ker23;
#if defined(VECT) && !defined(SINGLE)
          __m128d vec_k = _mm_set1_pd(k);
          __m128d vec_val = _mm_load_pd(du+2*j);
          vec_out = _mm_add_pd(vec_out, _mm_mul_pd(vec_k, vec_val));
#else
          out[0] += du[2*j] * k;
          out[1] += du[2*j+1] * k;
#endif
          ++j;
        }
      }
    }
  } else {                          // wraps somewhere: use ptr list (slower)
```

- exploit SSE, SSE2, AVX, etc, common to 99% of CPUs.