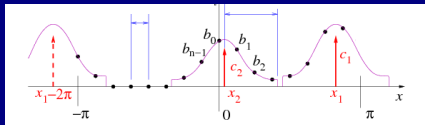


Building a better non-uniform fast Fourier transform

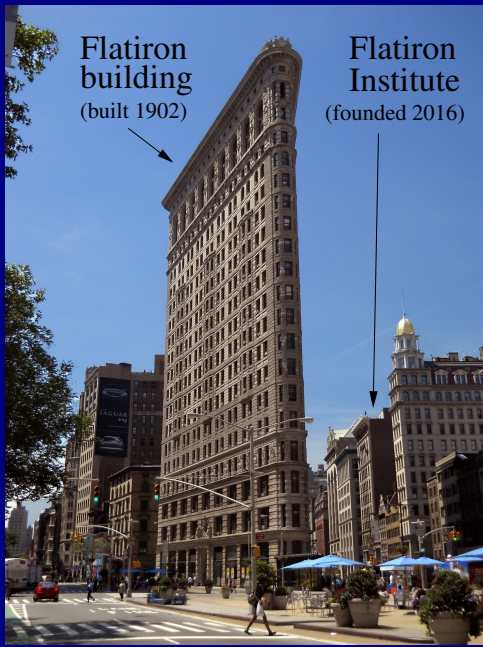
Princeton Applied and Computational Math Colloquium, 2/3/20

Alex Barnett Center for Computational Mathematics, Flatiron Institute

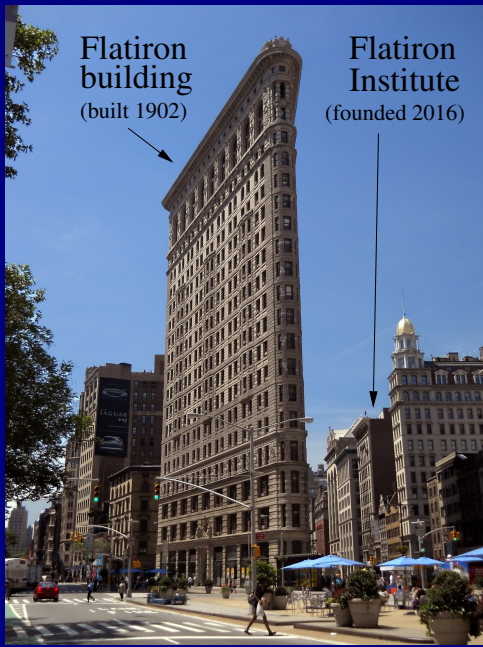
Joint work with Jeremy Magland (CCM), Ludvig af Klinteberg (KTH),
Yu-hsuan (Melody) Shih (NYU)



What is Flatiron?



What is Flatiron?



Flatiron
building
(built 1902)

Flatiron
Institute
(founded 2016)

a division of the
Simons Foundation

\approx 200 people

basic research in
computational...

Biology (CCB)

Astrophysics (CCA)

Quantum physics (CCQ)

Mathematics (CCM) (us)

Center for Computational Mathematics (CCM)

Formed late 2018. Still $< 0.4\times$ final size. Current areas:

- numerical PDE eg: fast algorithms, integral eqns.
- signal processing scattering transform, NUFFT library
- imaging cryo-EM, phase retrieval
- statistics & optimization ocean microbiome, inference, MCMC
- neuroscience tools calcium imaging, spike sorting
- deep learning eg: interatomic forces, symmetries

Basic research in methods, software release, consulting to other 3 Centers

Junior and senior research scientists, software engineers, postdocs, consultants, internships, visitors. Interdisciplinary.

Center for Computational Mathematics (CCM)

Formed late 2018. Still $< 0.4\times$ final size. Current areas:

- numerical PDE eg: fast algorithms, integral eqns. *
- signal processing scattering transform, NUFFT library *
- imaging cryo-EM, phase retrieval *
- statistics & optimization ocean microbiome, inference, MCMC
- neuroscience tools calcium imaging, spike sorting *
- deep learning eg: interatomic forces, symmetries

Basic research in methods, software release, consulting to other 3 Centers

Junior and senior research scientists, software engineers, postdocs, consultants, internships, visitors. Interdisciplinary.

Center for Computational Mathematics (CCM)

Formed late 2018. Still $< 0.4\times$ final size. Current areas:

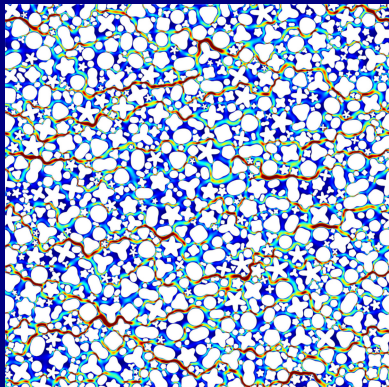
- numerical PDE eg: fast algorithms, integral eqns. *
- signal processing scattering transform, NUFFT library * today
- imaging cryo-EM, phase retrieval *
- statistics & optimization ocean microbiome, inference, MCMC
- neuroscience tools calcium imaging, spike sorting *
- deep learning eg: interatomic forces, symmetries

Basic research in methods, software release, consulting to other 3 Centers

Junior and senior research scientists, software engineers, postdocs, consultants, internships, visitors. Interdisciplinary.

Hint of another project at CCM. . .

Fast boundary integral equation tools for, eg, Stokes flows in complex geom:



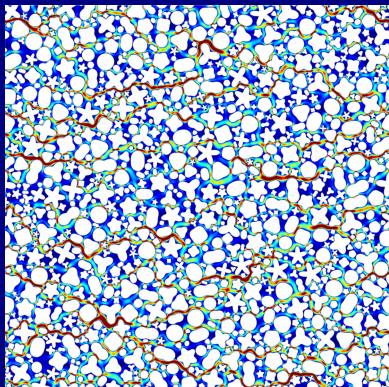
shown: 1 unit cell, doubly-periodic

10^3 obstacles, 10-digit accuracy

(B-Marple-Veerapaneni-Zhao '18)

Hint of another project at CCM. . .

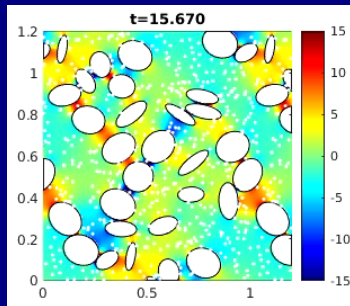
Fast boundary integral equation tools for, eg, Stokes flows in complex geom:



shown: 1 unit cell, doubly-periodic

10^3 obstacles, 10-digit accuracy

(B-Marple-Veerapaneni-Zhao '18)



suspension of rigid particles

time-dep. rheology, doubly-periodic

(B-Wang, '19)

Ask me about this in person :)

Overview of today

<https://github.com/flatironinstitute/finufft>

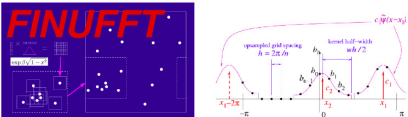
finufft 1.1.1 documentation » next | index

Next topic
Installation

This Page
Show Source

Quick search

Flatiron Institute Nonuniform Fast Fourier Transform



FINUFFT is a set of libraries to compute efficiently three types of nonuniform fast Fourier transform (NUFFT) to a specified precision, in one, two, or three dimensions, on a multi-core shared-memory machine. The library has a very simple interface, does not need any precomputation step, is written in C++ (using OpenMP and FFTW), and has wrappers to C, fortran, MATLAB, octave, and python. As an example, given M arbitrary real numbers x_j and complex numbers c_j , with $j = 1, \dots, M$, and a requested integer number of modes N , the 1D type-1 (aka "adjoint") transform evaluates the N numbers

$$f_k = \sum_{j=1}^M c_j e^{ikx_j}, \quad \text{for } k \in \mathbb{Z}, \quad -N/2 \leq k \leq N/2 - 1. \quad (1)$$

Overview of today

<https://github.com/flatironinstitute/finufft>

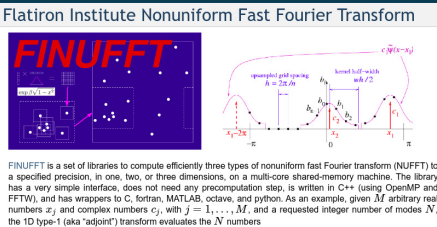
finufft 1.1.1 documentation » next | index

Next topic
Installation

This Page
Show Source

Quick search Go

Flatiron Institute Nonuniform Fast Fourier Transform



FINUFFT is a set of libraries to compute efficiently three types of nonuniform fast Fourier transform (NUFFT) to a specified precision, in one, two, or three dimensions, on a multi-core shared-memory machine. The library has a very simple interface, does not need any precomputation step, is written in C++ (using OpenMP and FFTW), and has wrappers to C, fortran, MATLAB, octave, and python. As an example, given M arbitrary real numbers x_j and complex numbers c_j , with $j = 1, \dots, M$, and a requested integer number of modes N , the 1D type-1 (aka "adjoint") transform evaluates the N numbers

$$f_k = \sum_{j=1}^M c_j e^{ikx_j}, \quad \text{for } k \in \mathbb{Z}, \quad -N/2 \leq k \leq N/2 - 1. \quad (1)$$

We released in 2018

What does it do?

Overview of today

<https://github.com/flatironinstitute/finufft>

finufft 1.1.1 documentation » next | index

Flatiron Institute Nonuniform Fast Fourier Transform

FINUFFT

FINUFFT is a set of libraries to compute efficiently three types of nonuniform fast Fourier transform (NUFFT) to a specified precision, in one, two, or three dimensions, on a multi-core shared-memory machine. The library has a very simple interface, does not need any precomputation step, is written in C++ (using OpenMP and FFTW), and has wrappers to C, fortran, MATLAB, octave, and python. As an example, given M arbitrary real numbers x_j and complex numbers c_j , with $j = 1, \dots, M$, and a requested integer number of modes N , the 1D type-1 (aka "adjoint") transform evaluates the N numbers

$$f_k = \sum_{j=1}^M c_j e^{ikx_j}, \quad \text{for } k \in \mathbb{Z}, \quad -N/2 \leq k \leq N/2 - 1. \quad (1)$$

We released in 2018

What does it do?

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

But... surely there already exist libraries?

eg NFFT from Chemnitz ([Potts–Keiner–Kunis](#)), NUFFT from NYU ([Lee–Greengard](#))

Ours is faster in large-scale 2D/3D settings, less RAM, simpler to use

Overview of today

<https://github.com/flatironinstitute/finufft>

finufft 1.1.1 documentation » next | index

Flatiron Institute Nonuniform Fast Fourier Transform

FINUFFT

FINUFFT is a set of libraries to compute efficiently three types of nonuniform fast Fourier transform (NUFFT) to a specified precision, in one, two, or three dimensions, on a multi-core shared-memory machine. The library has a very simple interface, does not need any precomputation step, is written in C++ (using OpenMP and FFTW), and has wrappers to C, fortran, MATLAB, octave, and python. As an example, given M arbitrary real numbers x_j and complex numbers c_j , with $j = 1, \dots, M$, and a requested integer number of modes N , the 1D type-1 (aka "adjoint") transform evaluates the N numbers

$$f_k = \sum_{j=1}^M c_j e^{ikx_j}, \quad \text{for } k \in \mathbb{Z}, \quad -N/2 \leq k \leq N/2 - 1. \quad (1)$$

We released in 2018

What does it do?

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

But... surely there already exist libraries?

eg NFFT from Chemnitz ([Potts–Keiner–Kunis](#)), NUFFT from NYU ([Lee–Greengard](#))

Ours is faster in large-scale 2D/3D settings, less RAM, 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. $f_k = \sum_{j=0}^{N-1} e^{ik\frac{2\pi j}{N}} c_j$, $-N/2 \leq k < N/2$

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

Recap the discrete Fourier transform (DFT)

Task: eval. $f_k = \sum_{j=0}^{N-1} e^{ik\frac{2\pi j}{N}} c_j$, $-N/2 \leq k < N/2$ 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. $f_k = \sum_{j=0}^{N-1} e^{ik\frac{2\pi j}{N}} c_j$, $-N/2 \leq k < N/2$ **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. $f_k = \sum_{j=0}^{N-1} e^{ik\frac{2\pi j}{N}} c_j$, $-N/2 \leq k < N/2$ 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 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. $f_k = \sum_{j=0}^{N-1} e^{ik\frac{2\pi j}{N}} c_j$, $-N/2 \leq k < N/2$ 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 \mathbf{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: f smooth $\Leftrightarrow \hat{f}_n$ decays for $|n|$ large \Leftrightarrow 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. $f_k = \sum_{j=0}^{N-1} e^{ik\frac{2\pi j}{N}} c_j$, $-N/2 \leq k < N/2$ 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 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: f smooth $\Leftrightarrow \hat{f}_n$ decays for $|n|$ large \Leftrightarrow 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 $f_k = \sum_{j=1}^M e^{ikx_j} c_j$, $-N/2 \leq k < N/2$

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

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 $f_k = \sum_{j=1}^M e^{ikx_j} c_j$, $-N/2 \leq k < N/2$

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

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

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 $f_k = \sum_{j=1}^M e^{ikx_j} c_j$, $-N/2 \leq k < N/2$

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

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

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

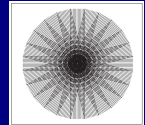
Type-3: NU to NU, also needs NU output freqs $\{s_k\}_{k=1}^N$

evaluates $f_k = \sum_{j=1}^M e^{is_k x_j} c_j$, $k = 1, \dots, N$ 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 fast exponential sums needed in many areas

- Given large # of point masses (eg stars), what is Fourier spectrum?
3D type-1 (NU to U) eg, universe simulation in periodic cube
- Magnetic resonance imaging (MRI)
2D/3D type-2: models signal at NU \mathbf{k} -space pts, given U image
- Image recon. from Fourier data: coherent diffraction (CDI), etc
- Making mol. dyn. or PDEs spatially periodic (“particle-mesh Ewald”)
- PDEs: spectral interpolation from U to NU grids; applying heat kernels



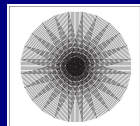
These fast exponential sums needed in many areas

- Given large # of point masses (eg stars), what is Fourier spectrum?

3D type-1 (NU to U) eg, universe simulation in periodic cube

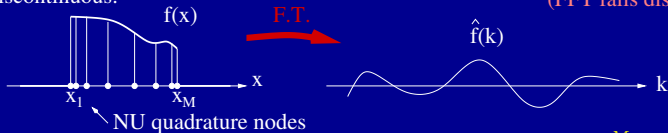
- Magnetic resonance imaging (MRI)

2D/3D type-2: models signal at NU k -space pts, given U image



- Image recon. from Fourier data: coherent diffraction (CDI), etc
- Making mol. dyn. or PDEs spatially periodic (“particle-mesh Ewald”)
- PDEs: spectral interpolation from U to NU grids; applying heat kernels
- Type-1 can compute accurate Fourier *transforms* of a non-smooth $f(\mathbf{x})$:

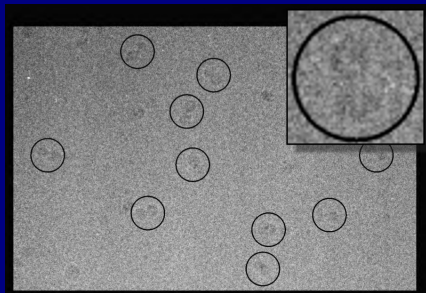
Eg, f discontinuous:



Apply good quadrature (eg Gauss) to Fourier integral $\hat{f}(k) := \int_{-\infty}^{\infty} e^{ikx} f(x) dx \approx \sum_{j=1}^M e^{ikx_j} f(x_j) w_j$

NUFFT in cryo-EM protein reconstruction

Much recent algorithmic progress in going from stack of images like this...

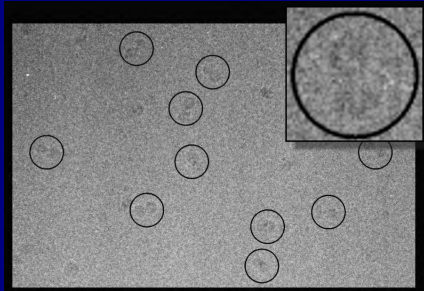


v. noisy TEM micrographs: $\text{SNR} \sim 0.1$

gives $\sim 10^5$ projections of ρ at unknown angles

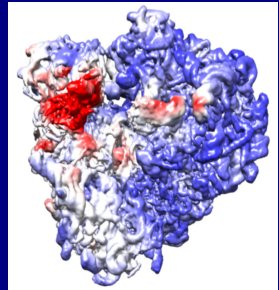
NUFFT in cryo-EM protein reconstruction

Much recent algorithmic progress in going from stack of images like this...



v. noisy TEM micrographs: $\text{SNR} \sim 0.1$
gives $\sim 10^5$ projections of ρ at unknown angles

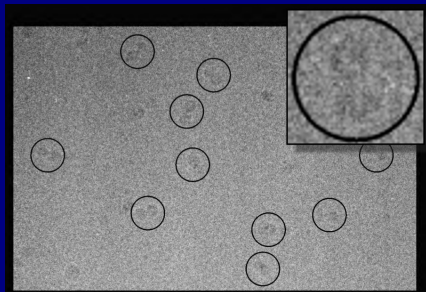
to this...



3D electron density map $\rho(\mathbf{x})$; now 2\AA poss.
80S ribosome, color=variance (Andén, '18)

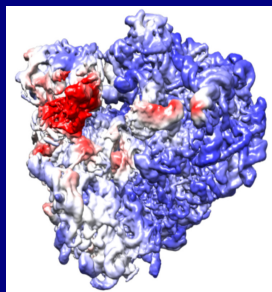
NUFFT in cryo-EM protein reconstruction

Much recent algorithmic progress in going from stack of images like this...



v. noisy TEM micrographs: SNR ~ 0.1
gives $\sim 10^5$ projections of ρ at unknown angles

to this...



3D electron density map $\rho(\mathbf{x})$; now 2Å poss.
80S ribosome, color=variance (Andén, '18)

NUFFT App. I

$\sim 10^{10}$ data

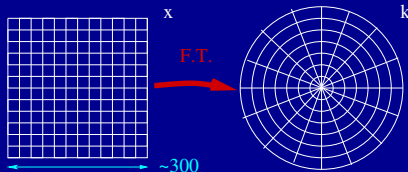
2D type-2: eval. $\hat{f}(\mathbf{k}_j)$ on polar grid

why? allows accurate image rotations

We enable 10^8 pts/sec on desktop

2D image f on U grid

NU Fourier samples



Cryo-EM NUFFT App. II

Many algs. to reconstruct $\rho(\mathbf{x})$ alternate between: “hard expectation maximization”

- a) given current ρ guess, fit best Euler proj. angle for each noisy image
- b) given their angles, use all images to get ρ noisy CT imaging task (inverse Radon)

Cryo-EM NUFFT App. II

Many algs. to reconstruct $\rho(\mathbf{x})$ alternate between: “hard expectation maximization”

a) given current ρ guess, fit best Euler proj. angle for each noisy image

b) given their angles, use all images to get ρ noisy CT imaging task (inverse Radon)

Step b) exploits Fourier Slice Theorem $\int \rho(x, y, z) dz \xrightarrow{2D.F.T.} \hat{\rho}(k_x, k_y, 0)$

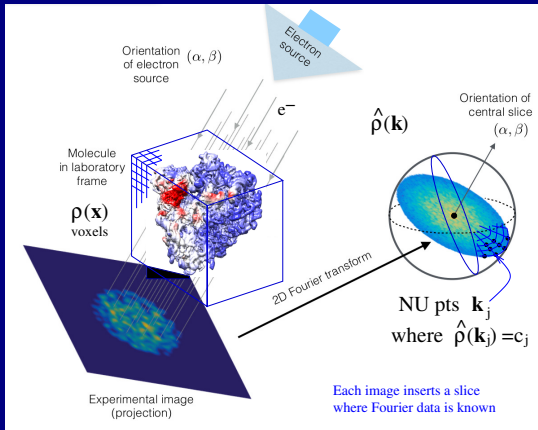
Cryo-EM NUFFT App. II

Many algs. to reconstruct $\rho(\mathbf{x})$ alternate between: “hard expectation maximization”

a) given current ρ guess, fit best Euler proj. angle for each noisy image

b) given their angles, use all images to get ρ **noisy CT imaging task (inverse Radon)**

Step b) exploits Fourier Slice Theorem $\int \rho(x, y, z) dz \xrightarrow{2D\text{ F.T.}} \hat{\rho}(k_x, k_y, 0)$

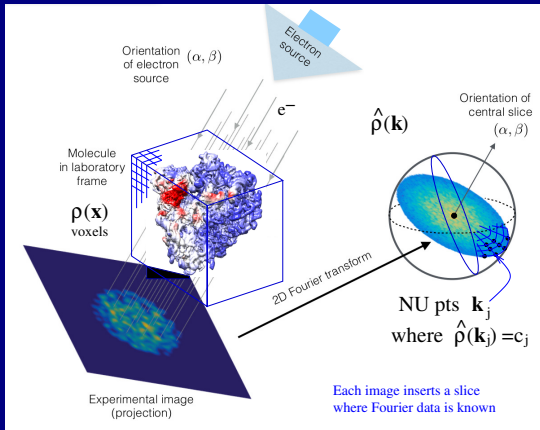


Cryo-EM NUFFT App. II

Many algs. to reconstruct $\rho(\mathbf{x})$ alternate between: “hard expectation maximization”

- given current ρ guess, fit best Euler proj. angle for each noisy image
- given their angles, use all images to get ρ noisy CT imaging task (inverse Radon)

Step b) exploits Fourier Slice Theorem $\int \rho(x, y, z) dz \xrightarrow{2D\text{ F.T.}} \hat{\rho}(k_x, k_y, 0)$



Solve in least-squares sense:

$$A\rho = c$$

c all Fourier slice data

ρ unknown density at voxels

A matrix of 3D type-2 to \mathbf{k}_j

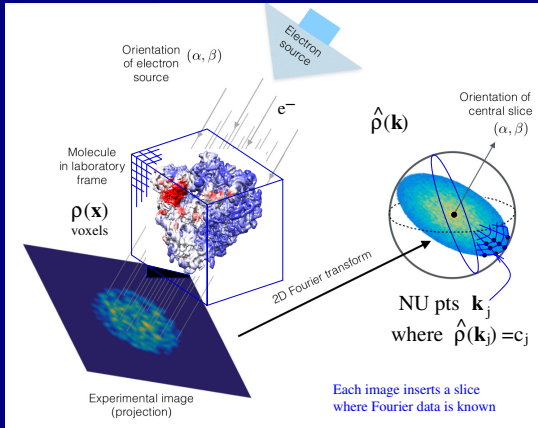
“inverse NUFFT” (harder!)

Cryo-EM NUFFT App. II

Many algs. to reconstruct $\rho(\mathbf{x})$ alternate between: “hard expectation maximization”

- given current ρ guess, fit best Euler proj. angle for each noisy image
- given their angles, use all images to get ρ noisy CT imaging task (inverse Radon)

Step b) exploits Fourier Slice Theorem $\int \rho(x, y, z) dz \xrightarrow{2D\text{ F.T.}} \hat{\rho}(k_x, k_y, 0)$



Solve in least-squares sense:

$$A\rho = c$$

c all Fourier slice data

ρ unknown density at voxels

A matrix of 3D type-2 to \mathbf{k}_j

“inverse NUFFT” (harder!)

$$\text{CG iter. on } (A^* A)\rho = A^* c$$

Toeplitz (apply via FFT) 3D type-1

ASPIRE (Singer et al.) uses FINUFFT

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 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

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 $\{\hat{b}_k\}_{k=-N_f/2}^{N_f/2-1}$

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

Three steps:

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

- Add each strength c_j onto fine-grid cell w/ location \tilde{x}_j nearest x_j
- Call this vector $\{b_l\}_{l=0}^{N_f-1}$: take its size- N_f FFT to get $\{\hat{b}_k\}_{k=-N_f/2}^{N_f/2-1}$
- Keep just low-freq outputs: $\tilde{f}_k = \hat{b}_k$, 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: $\tilde{f}_k = \hat{b}_k$, for $-N/2 \leq k < N/2$

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:

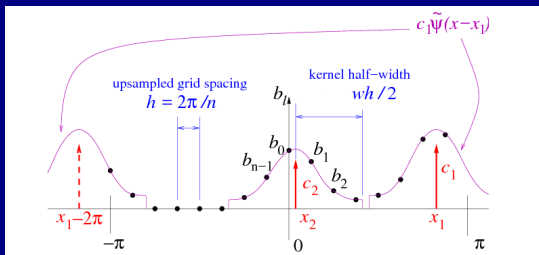
Set up “not-as-fine” grid on $[0, 2\pi)$, $N_f = \sigma N$, upsampling $\sigma \approx 2$

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:

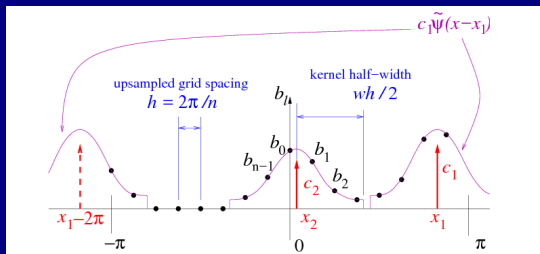
Set up “not-as-fine” grid on $[0, 2\pi)$, $N_f = \sigma N$, upsampling $\sigma \approx 2$

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:

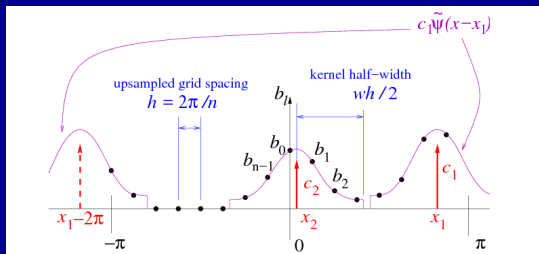
Set up “not-as-fine” grid on $[0, 2\pi)$, $N_f = \sigma N$, upsampling $\sigma \approx 2$

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 $-N/2 \leq k < N/2$

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?

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?

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 :(

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

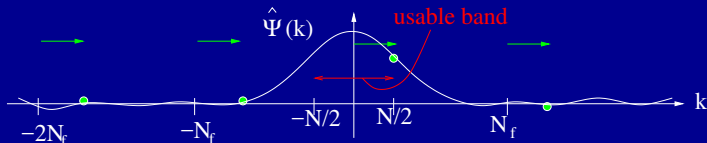
B) ψ should be smooth. Why? So $\sum_{j=1}^M c_j \psi(x - x_j)$ is also

recall smooth \rightarrow small DFT aliasing error

A) and B) are conflicting requirements :(

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

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



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–af Klinteberg '19, B '20)

(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–af Klinteberg '19, B '20)

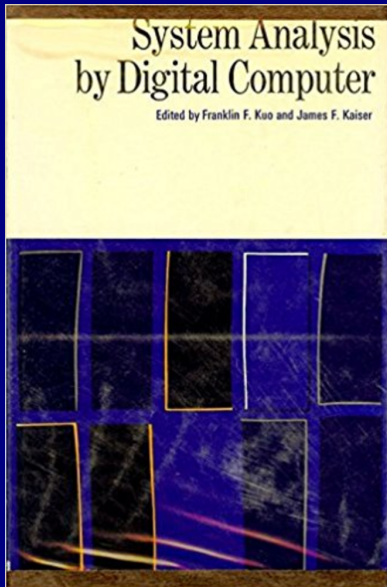
But what on earth is Kaiser–Bessel ?

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...)

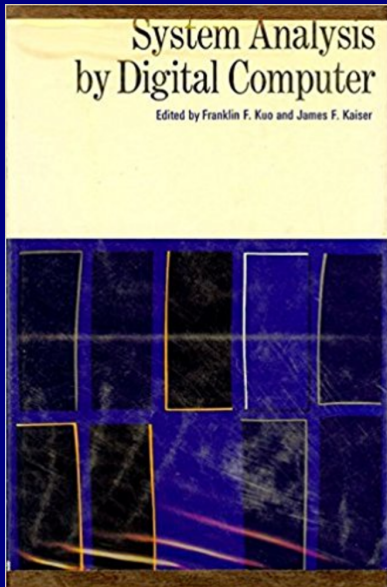
Story of the “Kaiser–Bessel” kernel

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

$$H_1^*(z) = a_0 w(\omega) + \frac{1}{2} \sum_{n=1}^N [a_n w(nT)] [z^n + z^{-n}], \quad m \text{ even} \quad (7.15)$$

or

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

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

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

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]

$$w(t) = \begin{cases} \frac{I_0 \left[\frac{\omega_a}{I_0(\omega_a \tau)} \sqrt{\tau^2 - t^2} \right]}{I_0(\omega_a \tau)}, & |t| < \tau \\ 0, & |t| > \tau \end{cases} \quad (7.18)$$

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

$$W(j\omega) = \frac{2}{I_0(\omega_a \tau)} \frac{\text{sinc} \left[\tau \sqrt{\omega^2 - \omega_a^2} \right]}{\sqrt{\omega^2 - \omega_a^2}} \quad (7.19)$$

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 -sinc 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 -sinc 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 -sinc 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

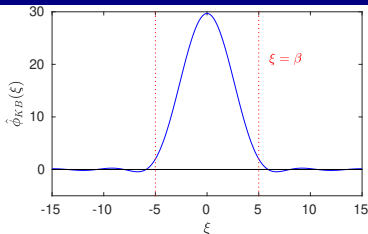
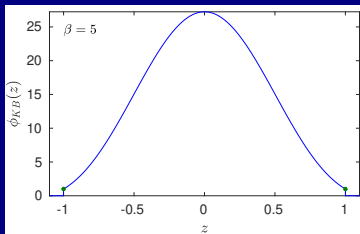
Kaiser–Bessel Fourier transform pair

The truncated kernel

$$\phi_{KB}(z) := \begin{cases} I_0(\beta\sqrt{1-z^2}), & |z| \leq 1 \\ 0, & \text{otherwise} \end{cases} \quad \text{we scale } z := 2x/wh$$

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=NU417xFhQdA&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...

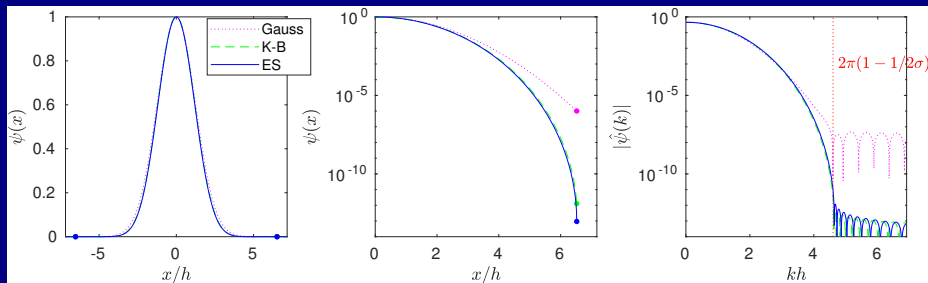
So one day I went in Ben's office and his chalkboard was just filled with equations. Way down in the left-hand corner of Ben's chalkboard was this transform pair, the I_0 -sinh transform pair. I didn't know what I_0 was, I said, "Ben, what's I_0 ?" He came back with "Oh, that's the modified Bessel function 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)

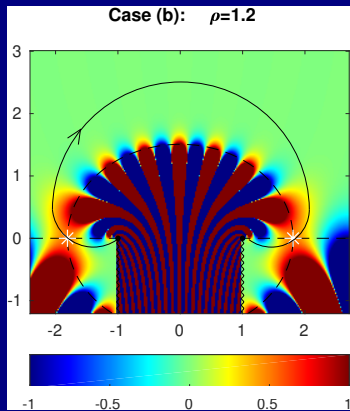
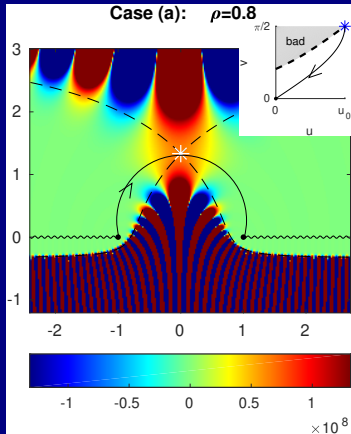
2) Analytic consequence: one has to work with the FT integral directly...

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

- same exponential convergence rate as Kaiser–Bessel, and as PSWF (Fuchs '64)
- consequence: $w \approx 7$ gives accuracy $\epsilon = 10^{-6}$, $w \approx 13$ 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... (B '20 [arxiv:2001.09405](https://arxiv.org/abs/2001.09405))

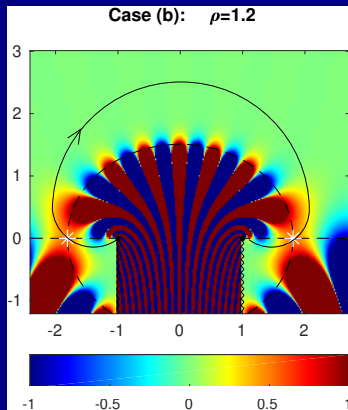
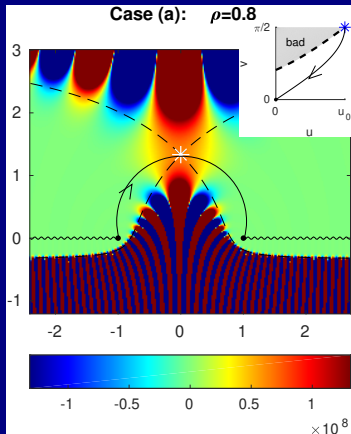
One proof ingredient

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

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

Implementation

- 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

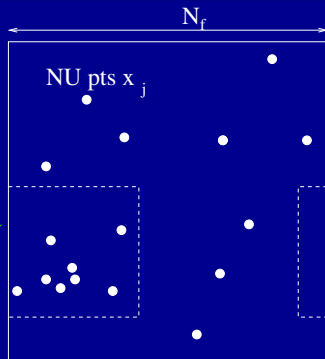
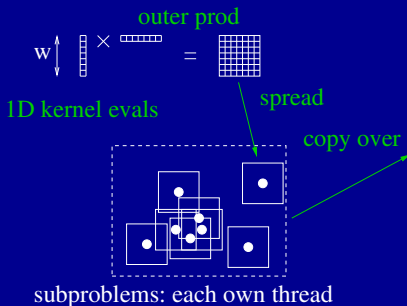
- Type 1,2,3 for dimensions $d = 1, 2, 3$: nine routines
- C++/OpenMP/SIMD, shared mem, calls FFTW. Apache license
- Wrappers to C, Fortran, MATLAB, octave, python, julia
- Cache-aware multithreaded spreading:

Type-2 easy: parallelize over *bin-sorted* NU pts

no collisions reading from U blocks

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

2D case, type-1, spread to fine grid:



Fast $e^{\beta\sqrt{1-x^2}}$ kernel approximation

Problem: for some chips & compilers, $\exp(x)$ slow (40 M evals/sec/core)
But kernel evals lie on fine h grid \rightarrow piecewise polynomial, Horner's rule:

```
File Edit Options Buffers Tools C Help
// Code generated by gen_all_horner_C_code.m in finufft/develop
// Authors: Alex Barnett & Ludvig af Klinteberg.
// (C) 2018, The Simons Foundation, Inc.
if (w==2) {
  FLT c0[] = {4.5147043243215343E+01, 4.5147043243215336E+01};
  FLT c1[] = {5.7408070938221300E+01, -5.7408070938221293E+01};
  FLT c2[] = {-1.8395117920046662E+00, -1.8395117920046617E+00};
  FLT c3[] = {-2.0382426253182079E+01, 2.0382426253182079E+01};
  FLT c4[] = {-2.0940804433577291E+00, -2.0940804433577358E+00};
  FLT c5[] = {3.1328044596872613E+00, -3.1328044596872546E+00};
  for (int i=0; i<2; i++) ker[i] = c0[i] + z*(c1[i] + z*(c2[i] + z*(c3[i] + z*(c4[i] + z*(c5[i])))));
} else if (w==3) {
  FLT c0[] = {1.5653991189315124E+02, 8.8006872410780340E+02, 1.5653991189967161E+02};
  FLT c1[] = {3.1653018869611071E+02, 2.1722031447974492E-14, -3.1653018868907077E+02};
  FLT c2[] = {1.7742692790454473E+02, -3.3149255274727807E+02, 1.7742692791117116E+02};
  FLT c3[] = {-1.5357716116473128E+01, -5.1917435849174007E-16, 1.5357716122720189E+01};
  FLT c4[] = {-3.7757583061523604E+01, 5.3222970968867436E+01, -3.7757583054647363E+01};
  FLT c5[] = {-3.9654011076088960E+00, 6.0642442697108023E-14, 3.9654011139270056E+00};
  FLT c6[] = {3.3694352031960180E+00, -4.8817394017826032E+00, 3.3694352094301192E+00};
  for (int i=0; i<3; i++) ker[i] = c0[i] + z*(c1[i] + z*(c2[i] + z*(c3[i] + z*(c4[i] + z*(c5[i] + z*(c6[i])))));
} else if (w==4) {
  FLT c0[] = {5.4284366850213223E+02, 1.0073871433088403E+04, 1.0073871433088401E+04, 5.4284366850213223E+02};
  FLT c1[] = {1.4650917259256937E+03, 6.1905285583602872E+03, -6.1905285583602890E+03, -1.4650917259256942E+03};
  FLT c2[] = {1.4186910680718343E+03, -1.3995339862725584E+03, -1.3995339862725591E+03, 1.4186910680718338E+03};
  FLT c3[] = {5.1133995502497419E+02, -1.4191608683682987E+03, 1.4191608683682980E+03, -5.1133995502497419E+02};
  FLT c4[] = {-4.82936226441173549E+01, 3.9393732546136526E+01, 3.9393732546137308E+01, -4.82936226441173634E+01};
  FLT c5[] = {-7.838686780232118E+01, 1.4918904800408907E+02, -1.4918904800408754E+02, 7.838686780232175E+01};
  FLT c6[] = {-1.0039212571700762E+01, 5.0626747735616444E+00, 5.0626747735613531E+00, -1.0039212571700721E+01};
  FLT c7[] = {4.7282853097645736E+00, -9.5966330409183929E+00, 9.5966330409170837E+00, -4.7282853097647068E+00};
  for (int i=0; i<4; i++) ker[i] = c0[i] + z*(c1[i] + z*(c2[i] + z*(c3[i] + z*(c4[i] + z*(c5[i] + z*(c6[i] + z*(c7[i])))));
} else if (w==5) {
  FLT c0[] = {9.9223677575398324E+02, 3.7794697666613341E+04, 9.8715771010760523E+04, 3.7794697666613290E+04, 9.922367757539849E+02};
}
-- ker_horner_allw_loop.c Top L19 Git-master (C/L WS Abbrev)
```

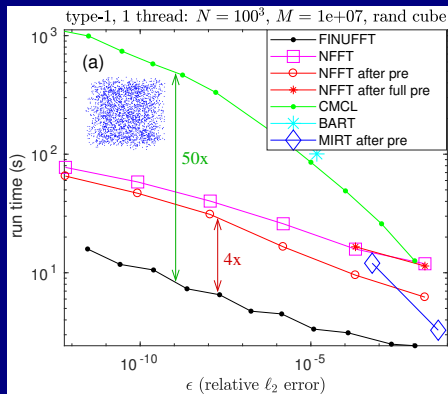
- even GCC compiler SIMD-vectorizes this; get 400-700 M evals/sec/core

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. $\operatorname{sinh} \sqrt{1-x^2}$

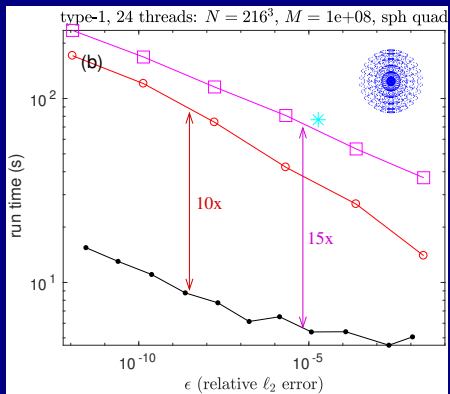
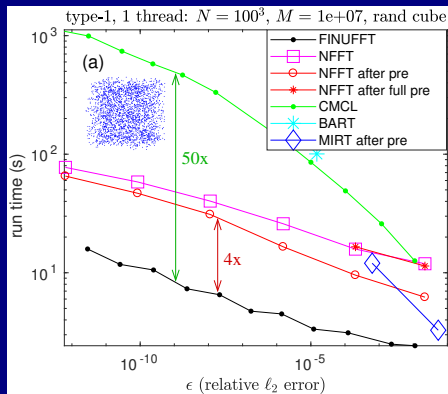
Performance: 3D Type-1 (the most dramatic)

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



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

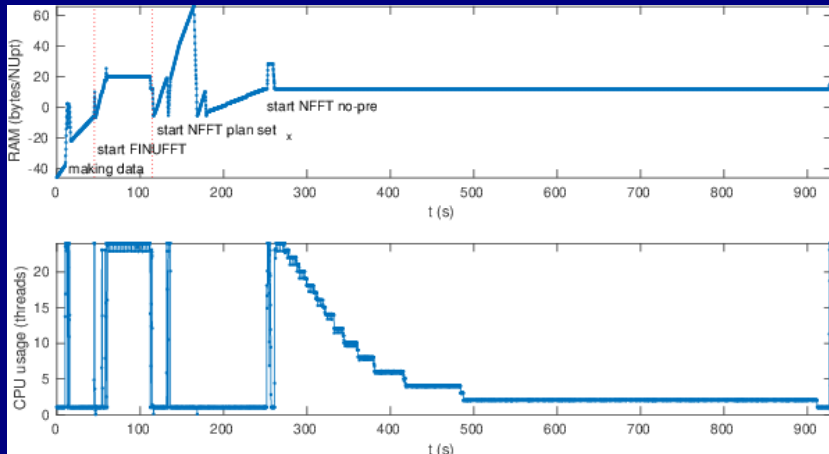
- Compare FINUFFT to
- CMCL NUFFFT (single-threaded, Gaussian kernel)
 - NFFT (multi-threaded, “backwards” Kaiser–Bessel) ie they eval. $\text{sinh } \sqrt{1-x^2}$



- 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:



Conclusions

NUFFT is a key tool with many scientific applications including cryo-EM

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, thread-balanced implementation, many other tricks

Conclusions

NUFFT is a key tool with many scientific applications including cryo-EM

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, thread-balanced implementation, many other tricks

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

(B-Magland-af Klinteberg, SISC, '19; B '20 [arxiv:2001.09405](https://arxiv.org/abs/2001.09405))

<https://github.com/flatironinstitute/finufft>

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

Ongoing:

- GPU version gets up to 4e8 NU pts/sec (intern: Yu-hsuan Melody Shih, NYU)
- “guru” interface, efficient for repeated calls (intern: Andrea Malleo)
- math: “why” are PSWF and K-B so close to $e^{\beta\sqrt{1-z^2}}$? no, it's not WKB...

EXTRA SLIDES

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 SIMD operations: SSE, SSE2, AVX, etc