FLATIRON INSTITUTE
Center for Computational Mathematics

Equispaced Fourier representations for efficient Gaussian process regression from a billion data points

Alex Barnett[1]

Courant Computational Mathematics and Scientific Computing Seminar, 11/11/22

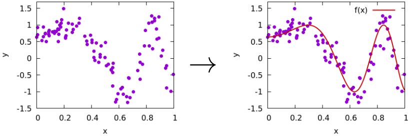
Work joint with Philip Greengard (stats @ Columbia) and Manas Rachh (CCM @ Flatiron)

Task: interpolation from noisy scattered data

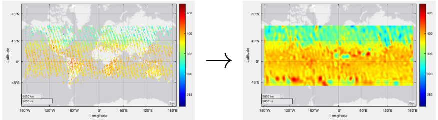
Given points $x_1, \ldots, x_N \in D \subset \mathbb{R}^d$ — e.g. $D = [0,1]^d$ domain

where meas. $y_n = f(x_n) + \epsilon_n$, noise $\epsilon_n \sim \mathcal{N}(0, \sigma^2)$ — scalar $y_n \in \mathbb{R}$

Recover underlying function $f \in C(D)$? — a.k.a. "kriging"



$d = 1$, e.g. time series, here toy $N = 10^2$



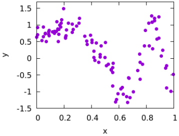
$d = 2$, geospatial (CO_2 satellite data), $N > 10^6$

Task: interpolation from noisy scattered data

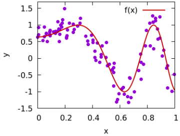
Given points $x_1, \ldots, x_N \in D \subset \mathbb{R}^d$ e.g. $D = [0,1]^d$ domain

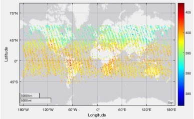
where meas. $y_n = f(x_n) + \epsilon_n$, noise $\epsilon_n \sim \mathcal{N}(0, \sigma^2)$ scalar $y_n \in \mathbb{R}$

Recover underlying function $f \in C(D)$? a.k.a. "kriging"

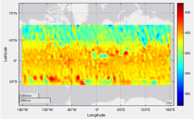


$\longrightarrow$





$\longrightarrow$



$d = 1$, e.g. time series, here toy $N = 10^2$

$d = 2$, geospatial (CO_2 satellite data), $N > 10^6$

- Need assumption on f, usually some order of *smoothness*

Noise-free case ($\sigma=0$): local/global polynomial interp. linear, cubic, barycentric
radial basis funcs, etc

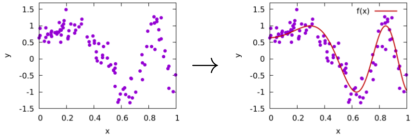
Noisy case: make f itself stochastic, recover *distribution* over f's

Task: interpolation from noisy scattered data

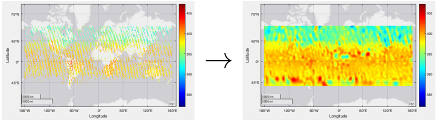
Given points $x_1, \ldots, x_N \in D \subset \mathbb{R}^d$ e.g. $D = [0,1]^d$ domain

where meas. $y_n = f(x_n) + \epsilon_n$, noise $\epsilon_n \sim \mathcal{N}(0, \sigma^2)$ scalar $y_n \in \mathbb{R}$

Recover underlying function $f \in C(D)$? a.k.a. "kriging"



$d = 1$, e.g. time series, here toy $N = 10^2$



$d = 2$, geospatial (CO_2 satellite data), $N > 10^6$

- Need assumption on f, usually some order of *smoothness*

Noise-free case (σ=0): local/global polynomial interp. linear, cubic, barycentric
radial basis funcs, etc

Noisy case: make f itself stochastic, recover *distribution* over f's

"Gaussian process" **prior** distn. on f, characterized by: mean $\equiv 0$,

2-point covar. given by some kernel: $\mathbb{E}\, f(x) f(x') = k(x, x')$, $x, x' \in D$

Likelihood of data vector $\mathbf{y} := \{y_n\}_{n=1}^N$ also Gaussian noise $\mathbf{y} | f(\mathbf{x}) \sim \mathcal{N}(0, \sigma^2 I)$

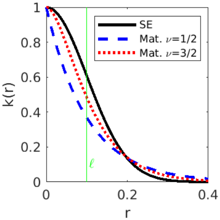
$\Rightarrow$ Bayes' theorem now specifies Gaussian **posterior** on f: "GP regression"

GP regression: kernels & posterior mean

Typical kernels $k(x - x')$ translation-invariant, isotropic $r = \|x - x'\|$, local (lengthscale $\ell > 0$):

- $k(r) = e^{-r^2/2\ell^2}$ "Squared exponential"
- $k(r) \propto (\frac{\sqrt{2\nu}r}{\ell})^\nu K_\nu(\frac{\sqrt{2\nu}r}{\ell})$ Matérn, smoothness $\nu \geq \frac{1}{2}$

$K_\nu(z)$ is modified Bessel func.

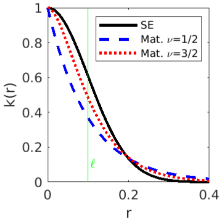


GP regression: kernels & posterior mean

Typical kernels $k(x-x')$ translation-invariant, isotropic $r = \|x-x'\|$, local (lengthscale $\ell > 0$):

- $k(r) = e^{-r^2/2\ell^2}$ "Squared exponential"
- $k(r) \propto (\frac{\sqrt{2\nu}r}{\ell})^\nu K_\nu(\frac{\sqrt{2\nu}r}{\ell})$ Matérn, smoothness $\nu \geq \frac{1}{2}$

$K_\nu(z)$ is modified Bessel func.



Posterior over functions $f \in C(D)$ is ∞-dim pdf! Summarize by...

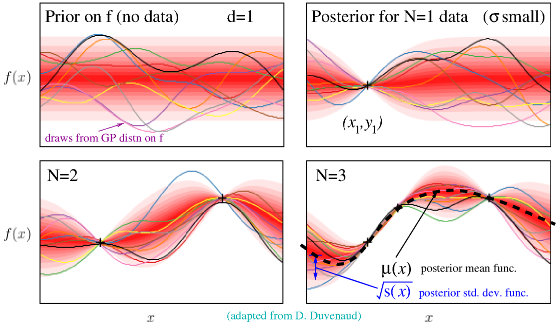
Marginal pdf at each $x \in D$:

shown as red density here →

Since everything is Gaussian,

$$f(x) \sim \mathcal{N}(\mu(x), s(x))$$

- $\mu(x)$ a common *predictor* of f at new "test" targets x



Linear system for posterior mean $\mu(x)$

Bayes thm. says: condition the joint pdf (data & unknowns) on the data $\mathbf{y}$

Linear system for posterior mean $\mu(x)$

Bayes thm. says: condition the joint pdf (data & unknowns) on the data $\mathbf{y}$

notation: kernel matrix $K \in \mathbb{R}^{N \times N}$, i.e., elements $K_{nl} := k(x_n, x_l), \quad n, l = 1, \ldots, N$

column vector of kernels to one new target x is $\mathbf{k}_x := \{k(x, x_n)\}_{n=1}^N$

joint pdf is $$\begin{bmatrix} \mathbf{y} \\ f(x) \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mathbf{0} \\ 0 \end{bmatrix}, \begin{bmatrix} K + \sigma^2 I & \mathbf{k}_x \\ \mathbf{k}_x^T & k(\mathbf{0}) \end{bmatrix} \right)$$

FLATIRON INSTITUTE
Center for Computational Mathematics

Linear system for posterior mean $\mu(x)$

Bayes thm. says: condition the joint pdf (data & unknowns) on the data $\mathbf{y}$

notation: kernel matrix $K \in \mathbb{R}^{N\times N}$, i.e., elements $K_{nl} := k(x_n, x_l)$, $\quad n, l = 1, \dots, N$

column vector of kernels to one new target x is $\mathbf{k}_x := \{k(x, x_n)\}_{n=1}^N$

joint pdf is $$\begin{bmatrix} \mathbf{y} \\ f(x) \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mathbf{0} \\ 0 \end{bmatrix}, \begin{bmatrix} K + \sigma^2 I & \mathbf{k}_x \\ \mathbf{k}_x^T & k(\mathbf{0}) \end{bmatrix} \right)$$

I skip formula for conditional of zero-mean multivariate Gaussian. . . (Schur complement)

Result for marginal at that target: $\quad f(x) \sim \mathcal{N}(\mu(x), s(x))$

with posterior mean func. $\quad \mu(x) = \sum_{n=1}^N k(x, x_n)\alpha_n = \mathbf{k}_x^T \boldsymbol{\alpha}$

where $\boldsymbol{\alpha} = \{\alpha_n\}_{n=1}^N$ is unique solution to

$$(K + \sigma^2 I)\boldsymbol{\alpha} = \mathbf{y}$$

"function space" linear system, $N \times N$ symm. pos. def.

- dense direct ("exact") solution costs $\mathcal{O}(N^3)$ time, $\mathcal{O}(N^2)$ RAM
  limits data size to $N \sim 10^4$ on single machine :(
- led to many approximate methods that scale better with N . . .

FLATIRON INSTITUTE
Center for Computational Mathematics

Previous methods to tackle larger data size N

1) Iterative solve via matvecs with $K + \sigma^2 I$ conjugate gradient, dense $\mathcal{O}(N^2 n_{\text{iter}})$

– low-rank approx. $K \approx K_{N,M}(K_{M,M})^{-1}K_{M,N}$ (Nyström '30)

via M "inducing points", subset of $\{x_n\}_{n=1}^N$, or new pts. $\mathcal{O}(NM^2 n_{\text{iter}})$

Previous methods to tackle larger data size N

1) Iterative solve via matvecs with $K + \sigma^2 I$ conjugate gradient, dense $\mathcal{O}(N^2 n_{\text{iter}})$

– low-rank approx. $K \approx K_{N,M}(K_{M,M})^{-1}K_{M,N}$ (Nyström '30)

via M "inducing points", subset of $\{x_n\}_{n=1}^N$, or new pts. $\mathcal{O}(NM^2 n_{\text{iter}})$

2) Fast direct solvers (Hackbusch, Rokhlin, Martinsson, Ying, Ho, O'Neil, Gillman, etc)

– off-diagonal blocks of K approx. low-rank (various: HODLR, $\mathcal{H}$-mat, HBS...)

– hierarchical inversion of blocks: compressed $(K + \sigma^2 I)^{-1}$ e.g. $\mathcal{O}(N^{3/2})$ 2D

Previous methods to tackle larger data size N

1) Iterative solve via matvecs with $K + \sigma^2 I$ conjugate gradient, dense $\mathcal{O}(N^2 n_{\text{iter}})$

– low-rank approx. $K \approx K_{N,M}(K_{M,M})^{-1}K_{M,N}$ (Nyström '30)

via M "inducing points", subset of $\{x_n\}_{n=1}^N$, or new pts. $\mathcal{O}(NM^2 n_{\text{iter}})$

2) Fast direct solvers (Hackbusch, Rokhlin, Martinsson, Ying, Ho, O'Neil, Gillman, etc)

– off-diagonal blocks of K approx. low-rank (various: HODLR, $\mathcal{H}$-mat, HBS...)

– hierarchical inversion of blocks: compressed $(K + \sigma^2 I)^{-1}$ e.g. $\mathcal{O}(N^{3/2})$ 2D

3) Transform to $M \times M$ system: coeffs. of M basis funcs. "weight space"

– subset of regressors, "sparse" GPs $\mathcal{O}(NM^2)$ to fill, then solve *indep* of N

– e.g., Fourier $e^{i\xi \cdot x}$ basis; full power not used (Hensman '17, P Greengard '21)

Previous methods to tackle larger data size N

1) Iterative solve via matvecs with $K + \sigma^2 I$ conjugate gradient, dense $\mathcal{O}(N^2 n_{\text{iter}})$

– low-rank approx. $K \approx K_{N,M}(K_{M,M})^{-1}K_{M,N}$ (Nyström '30)

via M "inducing points", subset of $\{x_n\}_{n=1}^N$, or new pts. $\mathcal{O}(NM^2 n_{\text{iter}})$

2) Fast direct solvers (Hackbusch, Rokhlin, Martinsson, Ying, Ho, O'Neil, Gillman, etc)

– off-diagonal blocks of K approx. low-rank (various: HODLR, $\mathcal{H}$-mat, HBS...)

– hierarchical inversion of blocks: compressed $(K + \sigma^2 I)^{-1}$ e.g. $\mathcal{O}(N^{3/2})$ 2D

3) Transform to $M \times M$ system: coeffs. of M basis funcs. "weight space"

– subset of regressors, "sparse" GPs $\mathcal{O}(NM^2)$ to fill, then solve *indep* of N

– e.g., Fourier $e^{i\xi \cdot x}$ basis; full power not used (Hensman '17, P Greengard '21)

State of the art (for $d > 1$) max out at $N \sim 10^7$, 1 hour, on desktop/GPU

Our method: class 3, Fourier, exploits fast fill and fast CG apply, $\mathcal{O}(N)$

We focus on d "small" ($d \leq 3$): t-series and spatial (geo) statistics

We will achieve $N = 10^9$ in e.g. 2 minutes on desktop...

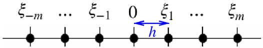
Factorizing a translationally-invariant kernel

Fourier transform $\hat{k}(\xi) := \int_{\mathbb{R}^d} k(x) e^{-2\pi i \xi \cdot x} dx$ $\quad \geq 0, \forall \xi \in \mathbb{R}^d$ for "positive" kernel

Factorizing a translationally-invariant kernel

Fourier transform $\hat{k}(\xi) := \int_{\mathbb{R}^d} k(x) e^{-2\pi i \xi \cdot x} dx$ $\geq 0, \forall \xi \in \mathbb{R}^d$ for "positive" kernel

Apply trapezoid quadrature to inverse FT:



$d = 1$

$$k(x-x') = \int \hat{k}(\xi) e^{2\pi i \xi (x-x')} d\xi \approx \sum_{j=-m}^{j=m} h \hat{k}(\xi_j) e^{2\pi i \xi_j (x-x')} = \sum_{j=1}^{M} \phi_j(x) \overline{\phi_j(x')}$$

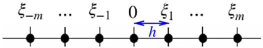
where "basis funcs" are $\phi_j(x) = \sqrt{h^d \hat{k}(\xi_j)} e^{2\pi i \xi_j \cdot x}$

For $d > 1$: equispaced product grid, size $M = (2m+1)^d$, label freqs. ξ_j, $m = 1, \ldots, M$

Factorizing a translationally-invariant kernel

Fourier transform $\hat{k}(\xi) := \int_{\mathbb{R}^d} k(x) e^{-2\pi i \xi \cdot x} dx$ $\quad \geq 0, \forall \xi \in \mathbb{R}^d$ for "positive" kernel

Apply trapezoid quadrature to inverse FT:



$d = 1$

$$k(x-x') = \int \hat{k}(\xi) e^{2\pi i \xi (x-x')} d\xi \approx \sum_{j=-m}^{j=m} h \hat{k}(\xi_j) e^{2\pi i \xi_j (x-x')} = \sum_{j=1}^{M} \phi_j(x) \overline{\phi_j(x')}$$

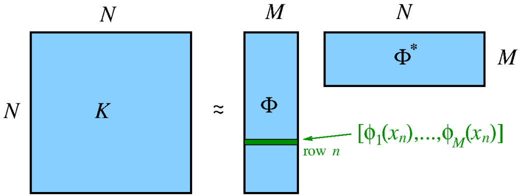
where "basis funcs" are $\phi_j(x) = \sqrt{h^d \hat{k}(\xi_j)} e^{2\pi i \xi_j \cdot x}$

For $d > 1$: equispaced product grid, size $M = (2m+1)^d$, label freqs. ξ_j, $m = 1, \ldots, M$

"Design" matrix $\Phi \in \mathbb{C}^{N \times M}$

elements $\Phi_{nj} := \phi_j(x_n)$

Then $K \approx \Phi \Phi^*$ (low-rank):



Can rigorously bound this approximation error, given k and $\hat{k}$ decay...

Kernel approximation error I

true kernel: $k(x - x')$ e.g. squared-exponential, Matérn

its Fourier grid approx: $\tilde{k}(x - x') = \sum_{j=1}^{M} \phi_j(x)\overline{\phi_j(x')}$

Kernel approximation error I

true kernel: $k(x - x')$ e.g. squared-exponential, Matérn

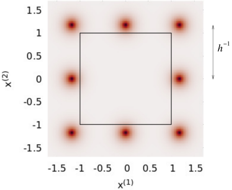
its Fourier grid approx: $\tilde{k}(x - x') = \sum_{j=1}^{M} \phi_j(x)\overline{\phi_j(x')}$

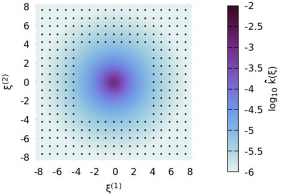
Lemma (pointwise error of truncated equispaced Fourier quadrature):

$$\tilde{k}(x) - k(x) = \underbrace{\sum_{\mathbf{n}\in\mathbb{Z}^d,\,\mathbf{n}\neq\mathbf{0}} k\left(x+\frac{\mathbf{n}}{h}\right)}_{\text{aliasing error: } k \text{ decay}} - \underbrace{\sum_{\mathbf{j}\in\mathbb{Z}^d,\,\mathbf{j}\notin\text{grid}} h^d\,\hat{k}(h\mathbf{j})\,e^{2\pi i h\mathbf{j}\cdot x}}_{\text{truncation error: } \hat{k} \text{ decay}}$$

Simple proof:

Poisson summation formula





Kernel approximation error I

true kernel: $k(x - x')$ e.g. squared-exponential, Matérn

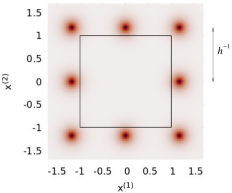
its Fourier grid approx: $\tilde{k}(x - x') = \sum_{j=1}^{M} \phi_j(x)\overline{\phi_j(x')}$

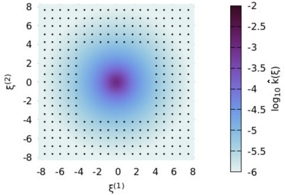
Lemma (pointwise error of truncated equispaced Fourier quadrature):

$$\tilde{k}(x) - k(x) = \underbrace{\sum_{\mathbf{n}\in\mathbb{Z}^d,\ \mathbf{n}\neq\mathbf{0}} k\left(x + \frac{\mathbf{n}}{h}\right)}_{\text{aliasing error: } k \text{ decay}} - \underbrace{\sum_{\mathbf{j}\in\mathbb{Z}^d,\ \mathbf{j}\notin\text{grid}} h^d \hat{k}(h\mathbf{j}) e^{2\pi i h\mathbf{j}\cdot x}}_{\text{truncation error: } \hat{k} \text{ decay}}$$

Simple proof:

Poisson summation formula





Seek uniform bnd $|\tilde{k}(x) - k(x)| \leq \varepsilon$ $\forall$ displacements $x \in D \ominus D = [-1, 1]^d$

Ideas: take worst-case x in aliasing error, discard phases in trunc. error

Kernel approximation error II

Result: theorems bounding ε, uniform approx. error for two kernel families

recall numerical params: Fourier grid spacing h, grid size $M = (2m+1)^d$

Thm (squared-exponential kernel):

exponential convergence in m

$$\varepsilon \le 2d\,3^d e^{-\frac{1}{2}\left(\frac{h^{-1}-1}{\ell}\right)^2} + 2d\,4^d e^{-2(\pi \ell h m)^2}$$

aliasing

truncation

Kernel approximation error II

Result: theorems bounding ε, uniform approx. error for two kernel families

recall numerical params: Fourier grid spacing h, grid size $M = (2m+1)^d$

Thm (squared-exponential kernel):

exponential convergence in m

$$\varepsilon \ \le \ \underbrace{2d\, 3^d e^{-\frac{1}{2}\left(\frac{h^{-1}-1}{\ell}\right)^2}}_{\text{aliasing}} + \underbrace{2d\, 4^d e^{-2(\pi \ell h m)^2}}_{\text{truncation}}$$

Thm (Matérn kernel, smoothness ν):

merely algebraic convergence, $m^{-2\nu}$

$$\varepsilon \ \le \ \underbrace{4d\, 3^{d-1} \frac{2^{1-\nu}}{\Gamma(\nu)} (4\nu)^\nu e^{2\nu} K_\nu(4\nu) e^{-\sqrt{\frac{\nu}{2d}}\frac{h^{-1}-1}{\ell}}}_{\text{aliasing}} + \underbrace{\frac{\nu^{\nu-1} d\, 5^{d-1}}{2^\nu \pi^{d/2+2\nu}} \frac{\Gamma(\nu+1/2)}{\Gamma(\nu)} \frac{1}{(h\ell m)^{2\nu}}}_{\text{truncation}}$$

Explicit constants! Proofs not trivial. Tools: bounding lattice sums by integrals, induction on dimension d, new bounds on K_ν Bessel funcs, 4 pages, some of August...

Corollaries: recipes to choose h and m to rigorously achieve tolerance ε

SE easy, but Matérn at low ν needs big grid (in practice instead use heuristic L_2-estimate)

Converting to a "weight-space" linear system

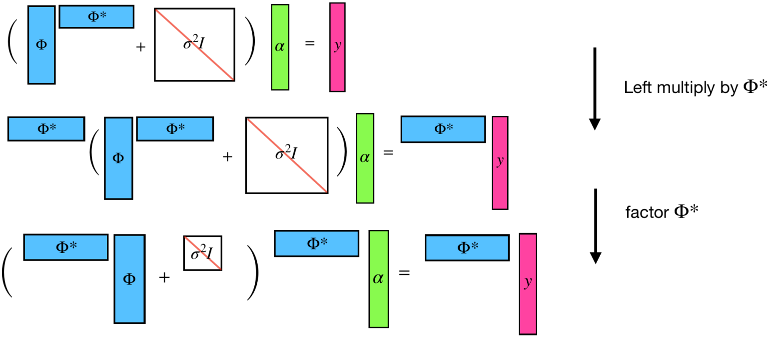
Recall "function-space" linear system $(K + \sigma^2 I)\boldsymbol{\alpha} = \mathbf{y}$

We just showed low-rank approx. $K \approx \Phi\Phi^*$ where can push error $\varepsilon \to 0$

Converting to a "weight-space" linear system

Recall "function-space" linear system $(K + \sigma^2 I)\boldsymbol{\alpha} = \mathbf{y}$

We just showed low-rank approx. $K \approx \Phi\Phi^*$ where can push error $\varepsilon \to 0$



got equiv. dual system: $(\Phi^*\Phi + \sigma^2 I)\boldsymbol{\beta} = \Phi^*\mathbf{y}$ $M \times M$, "weight space"

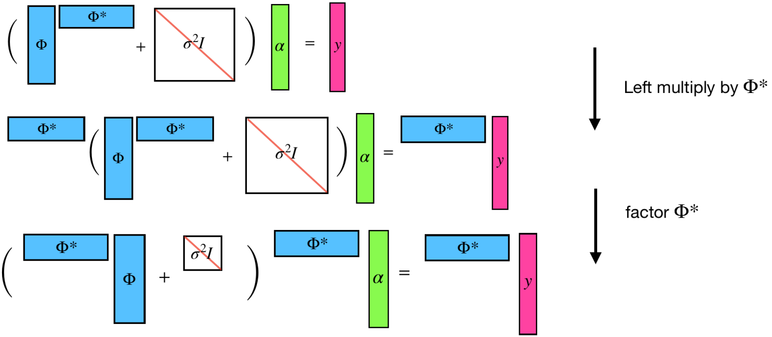
Solve for $\boldsymbol{\beta}$, is just basis coeffs of posterior mean $\mu(x) = \sum_{j=1}^{M} \beta_j \phi_j(x)$

Why? use $\boldsymbol{\beta} = \Phi^*\boldsymbol{\alpha}$: $\sum_j \beta_j \phi_j(x) = \sum_n \sum_j \phi_j(x)\overline{\phi_j(x_n)}\alpha_n = \sum_n k(x, x_n)\alpha_n = \mu(x)$

Converting to a "weight-space" linear system

Recall "function-space" linear system $(K + \sigma^2 I)\boldsymbol{\alpha} = \mathbf{y}$

We just showed low-rank approx. $K \approx \Phi\Phi^*$ where can push error $\varepsilon \to 0$



got equiv. dual system: $(\Phi^*\Phi + \sigma^2 I)\boldsymbol{\beta} = \Phi^*\mathbf{y}$ $M \times M$, "weight space"

Solve for $\boldsymbol{\beta}$, is just basis coeffs of posterior mean $\mu(x) = \sum_{j=1}^{M} \beta_j \phi_j(x)$

Why? use $\boldsymbol{\beta} = \Phi^*\boldsymbol{\alpha}$: $\sum_j \beta_j \phi_j(x) = \sum_n \sum_j \phi_j(x)\overline{\phi_j(x_n)}\alpha_n = \sum_n k(x, x_n)\alpha_n = \mu(x)$

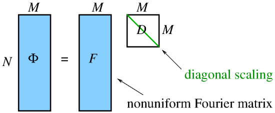
Huge advantages: i) M is indep of data size N, and have fast $\mu(x)$ eval.

ii) $\Phi^*\Phi$ and $\Phi^*\mathbf{y}$ have special structure so can form and apply fast. . .

Fast algorithm to solve in weight space

Recall linear system $(\Phi^*\Phi + \sigma^2 I)\boldsymbol{\beta} = \Phi^*\mathbf{y}$

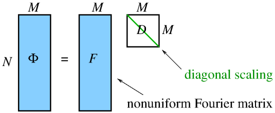
with $\Phi_{nj} = \phi_j(x_n) = e^{2\pi i \xi_j \cdot x_n} \sqrt{h^d \hat{k}(\xi_j)} =: F_{nj} D_{jj}$



Fast algorithm to solve in weight space

Recall linear system $(\Phi^*\Phi + \sigma^2 I)\boldsymbol{\beta} = \Phi^*\mathbf{y}$

with $\Phi_{nj} = \phi_j(x_n) = e^{2\pi i \xi_j \cdot x_n}\sqrt{h^d \hat{k}(\xi_j)} =: F_{nj} D_{jj}$



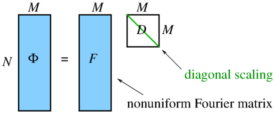
- Filling RHS: need $(\Phi^*\mathbf{y})_j = D_{jj}\sum_{n=1}^{N} e^{2\pi i \xi_j \cdot x_n} y_n, \quad j = 1, \ldots, M$

  Is a d-dimensional *nonuniform FFT*: generalization of FFT
  Can be done to accuracy ε, cost $\mathcal{O}(N \log^d(1/\varepsilon) + M \log M)$
  Uniform (equispaced) target grid $\xi_j = h\mathbf{j}$: "type 1" NUFFT (NU$\to$U)

Fast algorithm to solve in weight space

Recall linear system $(\Phi^*\Phi + \sigma^2 I)\boldsymbol{\beta} = \Phi^*\mathbf{y}$

with $\Phi_{nj} = \phi_j(x_n) = e^{2\pi i \xi_j \cdot x_n}\sqrt{h^d \hat{k}(\xi_j)} =: F_{nj} D_{jj}$

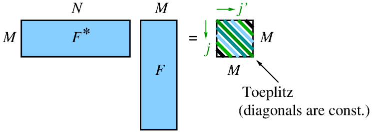


- Filling RHS: need $(\Phi^*\mathbf{y})_j = D_{jj}\sum_{n=1}^{N} e^{2\pi i \xi_j \cdot x_n} y_n, \quad j = 1, \dots, M$

  Is a d-dimensional *nonuniform FFT*: generalization of FFT
  Can be done to accuracy ε, cost $\mathcal{O}(N \log^d(1/\varepsilon) + M \log M)$
  Uniform (equispaced) target grid $\xi_j = h\mathbf{j}$: "type 1" NUFFT (NU→U)

- $(F^*F)_{\mathbf{j}\mathbf{j}'} = \sum_{n=1}^{N} e^{2\pi i h(\mathbf{j}'-\mathbf{j})\cdot x_n}$

  dep. only on $\mathbf{j}' - \mathbf{j}$



  Filling vector $\mathbf{v} \in \mathbb{C}^{(4m+1)^d}$ giving diagonals is another type 1 NUFFT!
  Matvec with F^*F is d-dim. *convolution* with $\mathbf{v}$: use padded plain FFT
  Apply system matrix $(D^*F^*FD + \sigma^2 I)$ in $\mathcal{O}(M \log M)$, per iteration

Note: Toeplitz property *only* because chose equispaced quadrature

a known idea in medical Fourier imaging (CT, MRI, cryo-EM), curiously with $\xi \leftrightarrows x$!

Equispaced Fourier GP (EFGP) algorithm summary

Inputs: kernel k, tolerance ε, points $\{x_n\}_{n=1}^N$, data $\{y_n\}_{n=1}^N$

1. Deduce grid params h then $M = (2m+1)^d$, from kernel and ε
2. Precompute RHS $\Phi^* \mathbf{y}$ via type 1 NUFFT with strengths $\{y_n\}$

   use ε as NUFFT tolerance
3. Precompute Toeplitz vector $\mathbf{v}$ via type 1 NUFFT with unit strengths
4. Use conjugate gradient to solve WS system $(\Phi^*\Phi + \sigma^2 I)\boldsymbol{\beta} = \Phi^* \mathbf{y}$

   use ε as relative residual criterion
5. Evaluate posterior mean $\mu(x) = \sum_{j=1}^M \beta_j D_{jj} e^{2\pi i h \mathbf{j} \cdot x}$ wherever you like

   a single "type 2" NUFFT (U→NU): cheap for huge number of targets x

Note: only two passes through size-N data; rest is quasilinear in M

Superior scaling to any other known algorithm (SKI, fast direct, etc)

However, prefactor also important — now show results comparisons. . .

The competition. The error metrics

We compare EFGP to three state-of-the-art GP solvers w/ software:

- SKI (structured kernel interpolation) (Wilson '15) in GPyTorch (Gardner '19)
  Cart. grid of inducing points $\rightarrow$ FFT-accel matvec, iterative CG solve of FS lin. sys.
- FLAM (fast linear algebra in MATLAB) (Ho '20) as used by (Minden '17)
  fast direct, FS: recursive skeletonization, interpolative decomp., annulus of proxy points
- RLCM (recursively low-rank compressed matrices) (Chen '21)
  fast direct, FS: hierarchical Nyström approx, pos. def., claims $\mathcal{O}(N)$ cost, in C++

The competition. The error metrics

We compare EFGP to three state-of-the-art GP solvers w/ software:

- SKI (structured kernel interpolation) (Wilson '15) in GPyTorch (Gardner '19)
  Cart. grid of inducing points $\to$ FFT-accel matvec, iterative CG solve of FS lin. sys.
- FLAM (fast linear algebra in MATLAB) (Ho '20) as used by (Minden '17)
  fast direct, FS: recursive skeletonization, interpolative decomp., annulus of proxy points
- RLCM (recursively low-rank compressed matrices) (Chen '21)
  fast direct, FS: hierarchical Nyström approx, pos. def., claims $\mathcal{O}(N)$ cost, in C++

Meaningful error metrics? Recall goal to recover $f(x)$ from $\{(x_n, y_n)\}$

- RMSE (typical in ML & kriging): root mean square prediction error
  $x_1^*, \ldots x_P^*$ new held-out points, $y_1^*, \ldots y_P^*$ data, RMSE $:= \left(\frac{1}{P}\sum_{n=1}^{P}[\mu(x_n^*) - y_n^*]^2\right)^{1/2}$

The competition. The error metrics

We compare EFGP to three state-of-the-art GP solvers w/ software:

- SKI (structured kernel interpolation) (Wilson '15) in GPyTorch (Gardner '19)
  Cart. grid of inducing points $\rightarrow$ FFT-accel matvec, iterative CG solve of FS lin. sys.
- FLAM (fast linear algebra in MATLAB) (Ho '20) as used by (Minden '17)
  fast direct, FS: recursive skeletonization, interpolative decomp., annulus of proxy points
- RLCM (recursively low-rank compressed matrices) (Chen '21)
  fast direct, FS: hierarchical Nyström approx, pos. def., claims $\mathcal{O}(N)$ cost, in C++

Meaningful error metrics? Recall goal to recover $f(x)$ from $\{(x_n, y_n)\}$

- RMSE (typical in ML & kriging): root mean square prediction error
  $x_1^*, \dots x_P^*$ new held-out points, $y_1^*, \dots y_P^*$ data, RMSE $:= \left(\frac{1}{P}\sum_{n=1}^{P}[\mu(x_n^*) - y_n^*]^2\right)^{1/2}$

  Problem: as approx GP becomes exact, RMSE $\rightarrow \mathcal{O}(\sigma)$, not zero :(

The competition. The error metrics

We compare EFGP to three state-of-the-art GP solvers w/ software:

- SKI (structured kernel interpolation) (Wilson '15) in GPyTorch (Gardner '19)
  Cart. grid of inducing points → FFT-accel matvec, iterative CG solve of FS lin. sys.
- FLAM (fast linear algebra in MATLAB) (Ho '20) as used by (Minden '17)
  fast direct, FS: recursive skeletonization, interpolative decomp., annulus of proxy points
- RLCM (recursively low-rank compressed matrices) (Chen '21)
  fast direct, FS: hierarchical Nyström approx, pos. def., claims $\mathcal{O}(N)$ cost, in C++

Meaningful error metrics? Recall goal to recover $f(x)$ from $\{(x_n, y_n)\}$

- RMSE (typical in ML & kriging): root mean square prediction error
  $x_1^*, \dots x_P^*$ new held-out points, $y_1^*, \dots y_P^*$ data, $\text{RMSE} := \left(\frac{1}{P}\sum_{n=1}^{P}[\mu(x_n^*) - y_n^*]^2\right)^{1/2}$

  Problem: as approx GP becomes exact, RMSE $\to \mathcal{O}(\sigma)$, not zero :(
- Estimated error in posterior mean. $\text{EEPM}_{\text{new}} := \left(\frac{1}{P}\sum_{n=1}^{P}[\mu(x_n^*) - \mu_{\text{ex}}(x_n^*)]^2\right)^{1/2}$
  Converges → 0. "exact" regression μ_{ex} found by convergence study of trusted method

The competition. The error metrics

We compare EFGP to three state-of-the-art GP solvers w/ software:

- SKI (structured kernel interpolation) (Wilson '15) in GPyTorch (Gardner '19)
  Cart. grid of inducing points $\to$ FFT-accel matvec, iterative CG solve of FS lin. sys.
- FLAM (fast linear algebra in MATLAB) (Ho '20) as used by (Minden '17)
  fast direct, FS: recursive skeletonization, interpolative decomp., annulus of proxy points
- RLCM (recursively low-rank compressed matrices) (Chen '21)
  fast direct, FS: hierarchical Nyström approx, pos. def., claims $\mathcal{O}(N)$ cost, in C++

Meaningful error metrics? Recall goal to recover $f(x)$ from $\{(x_n, y_n)\}$

- RMSE (typical in ML & kriging): root mean square prediction error
  $x_1^*, \dots x_P^*$ new held-out points, $y_1^*, \dots y_P^*$ data, $\text{RMSE} := \left(\frac{1}{P}\sum_{n=1}^{P}[\mu(x_n^*) - y_n^*]^2\right)^{1/2}$

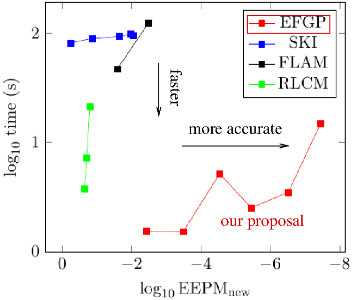
  Problem: as approx GP becomes exact, RMSE $\to \mathcal{O}(\sigma)$, not zero :(
- Estimated error in posterior mean. $\text{EEPM}_{\text{new}} := \left(\frac{1}{P}\sum_{n=1}^{P}[\mu(x_n^*) - \mu_{\text{ex}}(x_n^*)]^2\right)^{1/2}$
  Converges $\to 0$. "exact" regression μ_{ex} found by convergence study of trusted method
- But…error in $f(x)$ recovery? e.g. $\left(\frac{1}{q}\sum_{n=1}^{q}[\mu(x_n^*) - f(x_n^*)]^2\right)^{1/2}$
  Measures success of (even exact!) GP regression as a tool. Unused? Future study…

Results: CPU time vs accuracy achieved

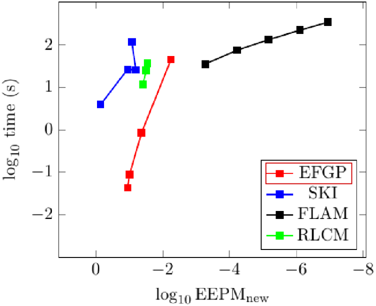
Synthetic $N = 10^5$ data points, iid uniform random in $[0,1]^d$

$f(x) = \sin(\omega \cdot x + a), \quad y_n = f(x_n) + \varepsilon_n, \quad \varepsilon_n$ iid Gaussian, $\sigma = 0.5$

For each method we vary a tolerance param (ε, rank, etc..) to get curve:



3D, squared-exponential kernel, $\ell = 0.1$



2D, Matérn-1/2 kernel, $\ell = 0.1$

- SE (left): EFGP 100× faster at 2-digit acc, can go to many digits

  recall SE smooth kernel, $\hat{k}$ super-exp. decay: very easy for Fourier method

- Matérn $\nu=\frac{1}{2}$ (right): FLAM best for high-acc (3+ digits)

  $\hat{k} \sim |\xi|^{-1-d}$, hardest for Fourier, yet EFGP 100× faster at 1-digit acc.

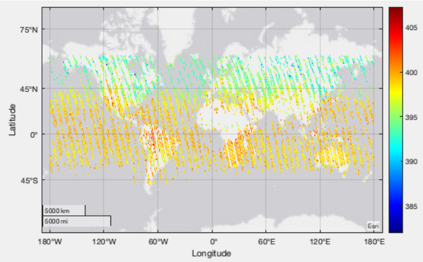
Results: atmospheric ppm CO_2 satellite data in $d = 2$

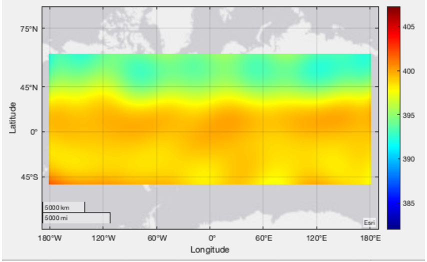
Geostatistics is fast if smooth kernel:

$N \approx 1.4 \times 10^6$ (Cressie '18)

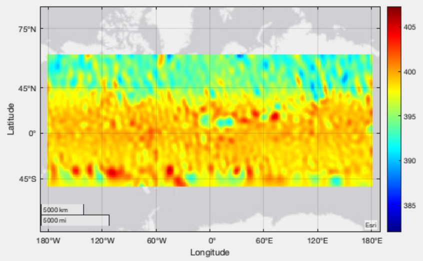
2 weeks data, patchy coverage

demean, then use SE kernel, $\sigma = 1$





$\ell = 50$: 0.5 s, $\text{EEPM}_{\text{new}} = 0.002$



$\ell = 5$: 5 s, $\text{EEPM}_{\text{new}} = 0.0002$

- In applications: often need repeat for $> 10^3$ time slices. . .

Results: large scale tests with nearest competitor (FLAM)

Synthetic 2D data, Matérn-$\frac{3}{2}$ kernel $\ell = 0.1$:

Alg	σ	ε	N	m	iters	tot (s)	mem (GB)	EEPM	$\mathrm{EEPM}_{\mathrm{new}}$	RMSE
EFGP	0.1	10^{-5}	3×10^6	94	2853	9	0.1	4.6×10^{-3}	4.6×10^{-3}	1.0×10^{-1}
EFGP	0.1	10^{-7}	3×10^6	346	9481	517	0.1	2.0×10^{-4}	1.9×10^{-4}	1.0×10^{-1}
FLAM	0.1	10^{-7}	3×10^6			384	9.1	5.4×10^{-5}	3.0×10^{-4}	1.0×10^{-1}
EFGP	0.1	10^{-5}	10^7	94	2634	10	0.3	3.9×10^{-3}	3.9×10^{-3}	1.0×10^{-1}
EFGP	0.1	10^{-7}	10^7	346	15398	878	0.7	3.4×10^{-4}	3.4×10^{-4}	1.0×10^{-1}
FLAM	0.1	10^{-7}	10^7			1272	25.0	8.0×10^{-5}	4.6×10^{-4}	1.0×10^{-1}
EFGP	0.1	10^{-5}	3×10^7	94	1915	9	2.6	3.1×10^{-3}	3.1×10^{-3}	1.0×10^{-1}
EFGP	0.1	10^{-7}	3×10^7	346	23792	1315	2.8	5.4×10^{-4}	5.4×10^{-4}	1.0×10^{-1}
FLAM	0.1	10^{-7}	3×10^7			3328	54.6	1.0×10^{-4}	7.7×10^{-4}	1.0×10^{-1}
EFGP	0.1	10^{-5}	10^8	94	1393	14	9.3	2.3×10^{-3}	2.3×10^{-3}	1.0×10^{-1}
EFGP	0.1	10^{-7}	10^8	346	35905	2055	9.5	7.6×10^{-4}	7.6×10^{-4}	1.0×10^{-1}
EFGP	0.1	10^{-5}	10^9	94	1027	103	96.7	1.2×10^{-3}	1.2×10^{-3}	1.0×10^{-1}
EFGP	0.1	10^{-7}	10^9	346	66199	4048	97.0	7.9×10^{-4}	7.9×10^{-4}	1.0×10^{-1}

- EFGP RAM scaling $\mathcal{O}(N)$, and 20–100$\times$ less than FLAM

  12-core desktop w/ 192 GB: could not run FLAM for $N > 10^8$
- EFGP becomes $3\times$ faster at $N = 3 \times 10^7$ and comparable accuracy
- If happy with 3-digit accuracy, EFGP does $N = 10^9$ in 2 minutes
- But: iteration count gets huge as decrease ε (why?)

Conditioning of the linear systems

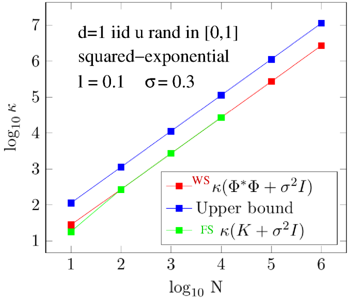
By kth conjugate gradient iter, error $\leq c\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^k \approx ce^{-2k/\sqrt{\kappa}}$ κ=cond. num.

In EFGP we care about WS $\kappa(\Phi^*\Phi + \sigma^2 I)$

Empirically we see this grows closely to its upper bound $\kappa(K + \sigma^2 I) \leq \frac{N}{\sigma^2} + 1$

pf easy: $\|K\| \leq \|K\|_F \leq N$, and $K \succcurlyeq 0$ by pos. kernel

FS and WS cond. num. similar, and bad!



Huge bnd: eg $N = 10^7$, $\sigma = 0.1$ gives $\kappa \leq 10^9$, $n_{\text{iter}} \lesssim 10^5$ for $\varepsilon = 10^{-5}$

consequence: all digits can be lost in single-precision arithmetic!

Conditioning of the linear systems

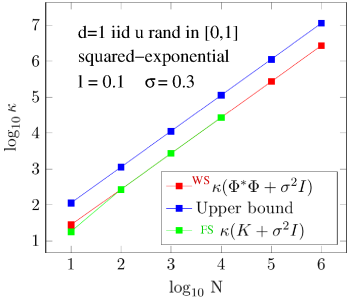
By kth conjugate gradient iter, error $\le c\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^k \approx ce^{-2k/\sqrt{\kappa}}$ κ=cond. num.

In EFGP we care about WS $\kappa(\Phi^*\Phi + \sigma^2 I)$

Empirically we see this grows closely to its upper bound $\kappa(K + \sigma^2 I) \le \frac{N}{\sigma^2} + 1$

pf easy: $\|K\| \le \|K\|_F \le N$, and $K \succcurlyeq 0$ by pos. kernel

FS and WS cond. num. similar, and bad!



Huge bnd: eg $N = 10^7$, $\sigma = 0.1$ gives $\kappa \le 10^9$, $n_{\text{iter}} \lesssim 10^5$ for $\varepsilon = 10^{-5}$

consequence: all digits can be lost in single-precision arithmetic!

Mystery 1: we observe *non-geometric* CG residual norm decay $\varepsilon \sim 1/k^2$

Mystery 2: can show GP regression *problem* has (abs.) cond. num. of 1

So, FS or WS methods handle ill-cond. sys to solve well-cond. prob... IMHO not good!

Much to explore, preconditioning...

How do nonuniform FFTs? our FINUFFT library

http://finufft.readthedocs.io (Barnett-Magland-af Klinteberg SISC '19)

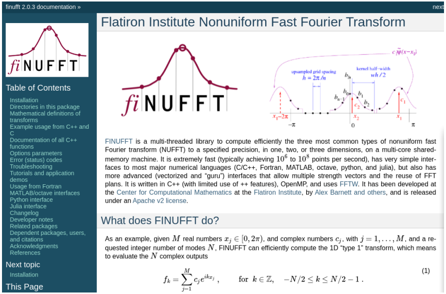

finufft 2.0.3 documentation » next

Table of Contents

Installation
Directories in this package
Mathematical definitions of transforms
Example usage from C++ and C
Documentation of all C++ functions
Options parameters
Error (status) codes
Troubleshooting
Tutorials and application demos
Usage from Fortran
MATLAB/octave interfaces
Python interface
Julia interface
Changelog
Developer notes
Related packages
Dependent packages, users, and citations
Acknowledgments
References

Next topic

Installation

This Page

Flatiron Institute Nonuniform Fast Fourier Transform

FINUFFT is a multi-threaded library to compute efficiently the three most common types of nonuniform fast Fourier transform (NUFFT) to a specified precision, in one, two, or three dimensions, on a multi-core shared-memory machine. It is extremely fast (typically achieving 10^6 to 10^8 points per second), has very simple interfaces to most major numerical languages (C/C++, Fortran, MATLAB, octave, python, and julia), but also has more advanced (vectorized and "guru") interfaces that allow multiple strength vectors and the reuse of FFT plans. It is written in C++ (with limited use of ++ features), OpenMP, and uses FFTW. It has been developed at the Center for Computational Mathematics at the Flatiron Institute, by Alex Barnett and others, and is released under an Apache v2 license.

What does FINUFFT do?

As an example, given M real numbers $x_j \in [0, 2\pi)$, and complex numbers c_j, with $j = 1, \ldots, M$, and a requested integer number of modes N, FINUFFT can efficiently compute the 1D "type 1" transform, which means to evaluate the N complex outputs

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

v1.0 released 2018, now v2.1.0

Types 1,2,3, in $d = 1, 2, 3$ dims

multithreaded C++, C API, wrappers:

Fortran, Python, MATLAB/Octave, Julia

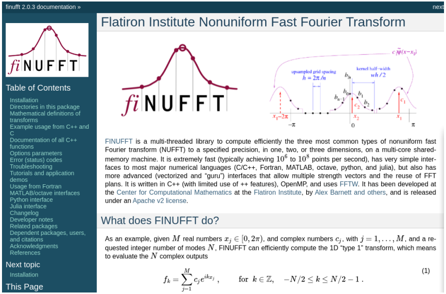
$\sim$ 5 devs; $\sim$ 20 contributors

160 GitHub stars

MRI, cryo-EM, PDE, sig. proc.

How do nonuniform FFTs? our FINUFFT library

`http://finufft.readthedocs.io` (Barnett-Magland-af Klinteberg SISC '19)


finufft 2.0.3 documentation » next

Flatiron Institute Nonuniform Fast Fourier Transform

Table of Contents

Next topic

Installation

This Page

FINUFFT is a multi-threaded library to compute efficiently the three most common types of nonuniform fast Fourier transform (NUFFT) to a specified precision, in one, two, or three dimensions, on a multi-core shared-memory machine. It is extremely fast (typically achieving 10^6 to 10^8 points per second), has very simple interfaces to most major numerical languages (C/C++, Fortran, MATLAB, octave, python, and julia), but also has more advanced (vectorized and "guru") interfaces that allow multiple strength vectors and the reuse of FFT plans. It is written in C++ (with limited use of ++ features), OpenMP, and uses FFTW. It has been developed at the Center for Computational Mathematics at the Flatiron Institute, by Alex Barnett and others, and is released under an Apache v2 license.

What does FINUFFT do?

As an example, given M real numbers $x_j \in [0, 2\pi)$, and complex numbers c_j, with $j = 1, \ldots, M$, and a requested integer number of modes N, FINUFFT can efficiently compute the 1D "type 1" transform, which means to evaluate the N complex outputs

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

v1.0 released 2018, now v2.1.0

Types 1,2,3, in $d = 1, 2, 3$ dims

multithreaded C++, C API, wrappers:

Fortran, Python, MATLAB/Octave, Julia

$\sim$ 5 devs; $\sim$ 20 contributors

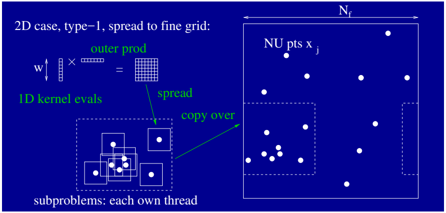
160 GitHub stars

MRI, cryo-EM, PDE, sig. proc.

Standard alg: spread $\to$ upsampled FFT $\to$ diagonal correction (type 1)

- new spreading kernel $e^{\beta\sqrt{1-x^2}}$
- piecewise polynomial Horner eval.
- SIMD-vectorized
- bin-sort for load-balanced spread

Typ: 10^7 NU pts/s, laptop, $\varepsilon = 10^{-6}$



Conclusions

GP regression popular for interpolation (kriging) from noisy scattered data

- We fix its poor scaling, allowing data size to $N \sim 10^9$ in minutes
- Equispaced quadrature in Fourier space $\rightarrow$ iter. solve for the weights
- One pass through data in $\mathcal{O}(N + M \log M)$; fast $M \log M$ per iter.
- Dimension d "low" (say $d \leq 6$); not for high-dim ML apps.

Preprint: `http://arxiv.org/abs/2210.10210`

MATLAB pkg: `http://github.com/flatironinstitute/gp-shootout`

Conclusions

GP regression popular for interpolation (kriging) from noisy scattered data

- We fix its poor scaling, allowing data size to $N \sim 10^9$ in minutes
- Equispaced quadrature in Fourier space $\rightarrow$ iter. solve for the weights
- One pass through data in $\mathcal{O}(N + M \log M)$; fast $M \log M$ per iter.
- Dimension d "low" (say $d \leq 6$); not for high-dim ML apps.

Preprint: `http://arxiv.org/abs/2210.10210`

MATLAB pkg: `http://github.com/flatironinstitute/gp-shootout`

Preliminary work (new area for me, 2022). Many things to do:

- Work with application users, release more than just research code
- Estimation of parameters $(\ell, \nu, \sigma, \ldots)$ max likelihood needs fast $\det(K + \sigma^2 I)$

  for now could estimate by cross-validation
- preconditioning (or fast direct solve) for Toeplitz+diagonal system
- Is GP regression (kriging) actually a *local* problem? Feels like it!

  but not in general: $(\text{banded matrix})^{-1} \neq$ banded matrix

FLATIRON INSTITUTE
Center for Computational Mathematics