

FFT on Multipoles (FFTM)

FFT accelerated SLFMM

Lecture 23

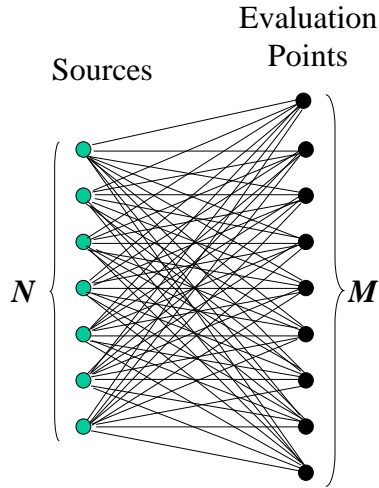
FFTM (Fast Fourier Transform on Multipoles)

- FFTM is, in fact, SLFMM speeded up with the FFT, which brings its complexity to $O(N\log N)$.

E. T. Ong, K. M. Lim, K. H. Lee and H. P. Lee:
A fast algorithm for three-dimensional potential fields
calculation: fast Fourier transform on multipoles
*Journal of Computational Physics, Volume 192, Issue
1, 20 November 2003, Pages 244-261*

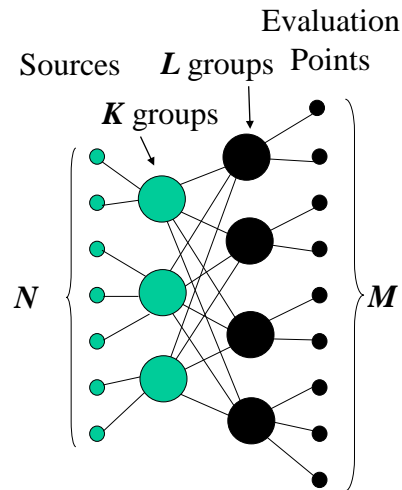
Idea of a Single Level FMM

Standard algorithm



Total number of operations: $O(NM)$

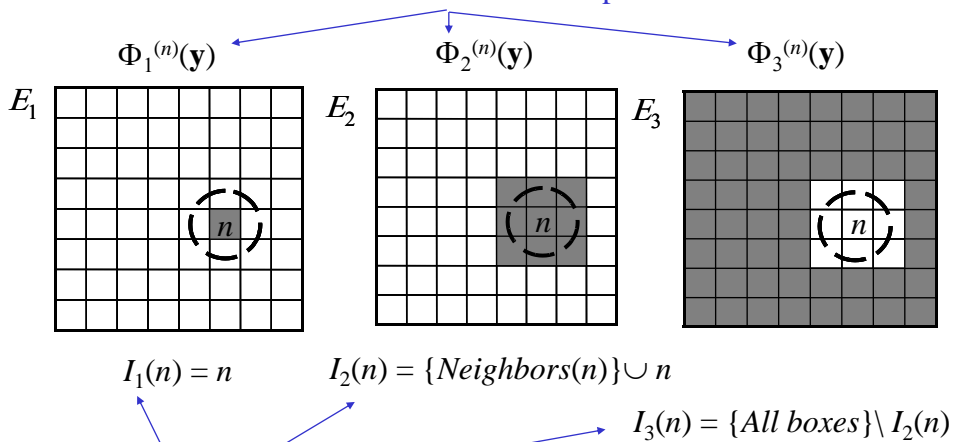
SLFMM



Total number of operations: $O(N+M+KL)$

Spatial Domains

Potentials due to sources in these spatial domains



Boxes with these numbers belong to these spatial domains

Definition of potentials

$$\Phi_1^{(n)}(\mathbf{y}) = \sum_{\mathbf{x}_i \in E_1(n)} u_i \Phi(\mathbf{y}, \mathbf{x}_i),$$

$$\Phi_2^{(n)}(\mathbf{y}) = \sum_{\mathbf{x}_i \in E_2(n)} u_i \Phi(\mathbf{y}, \mathbf{x}_i),$$

$$\Phi_3^{(n)}(\mathbf{y}) = \sum_{\mathbf{x}_i \in E_3(n)} u_i \Phi(\mathbf{y}, \mathbf{x}_i),$$

Since domains $E_2(n)$ and $E_3(n)$ are complimentary:

$$\Phi(\mathbf{y}) = \sum_{i=1}^{\infty} u_i \Phi(\mathbf{y}, \mathbf{x}_i) = \sum_{\mathbf{x}_i \in E_2(n) \cup E_3(n)} u_i \Phi(\mathbf{y}, \mathbf{x}_i) = \Phi_2^{(n)}(\mathbf{y}) + \Phi_3^{(n)}(\mathbf{y})$$

for arbitrary n .

SLFMM Algorithm

Step 1. Generate S-expansion coefficients

for each box

$$\Phi_1^{(n)}(\mathbf{x}) = \mathbf{C}^{(n)} \circ \mathbf{S}(\mathbf{x} - \mathbf{x}_c^{(n)}),$$

$$\mathbf{C}^{(n)} = \sum_{\mathbf{x}_i \in E_1(n)} u_i \mathbf{B}(\mathbf{x}_i, \mathbf{x}_c^{(n)}).$$

loop over all non-empty source boxes

For $n \in \text{NonEmptySource}$

Get $\mathbf{x}_c^{(n)}$, the center of the box;

$\mathbf{C}^{(n)} = \mathbf{0}$;

For $\mathbf{x}_i \in E_1(n)$ loop over all sources in the box

Get $\mathbf{B}(\mathbf{x}_i, \mathbf{x}_c^{(n)})$, the S-expansion coefficients near the center of the box;

$\mathbf{C}^{(n)} = \mathbf{C}^{(n)} + u_i \mathbf{B}(\mathbf{x}_i, \mathbf{x}_c^{(n)})$;

End;

End;

Implementation can be different!

All we need is to get $\mathbf{C}^{(n)}$.

SLFMM Algorithm

Step 2. (S|R)-translate expansion coefficients

$$\Phi_{\mathcal{I}_3(n)}^{(n)}(\mathbf{y}) = \mathbf{D}^{(n)} \circ \mathbf{R}(\mathbf{y} - \mathbf{x}_c^{(n)}),$$

$$\mathbf{D}^{(n)} = \sum_{m \in \mathcal{I}_3(n)} (\mathbf{S|R})(\mathbf{x}_c^{(n)} - \mathbf{x}_c^{(m)}) \mathbf{C}^{(m)}.$$

loop over all non-empty
evaluation boxes

For $n \in \text{NonEmptyEvaluation}$

Get $\mathbf{x}_c^{(n)}$, the center of the box;

$\mathbf{D}^{(n)} = \mathbf{0}$;

loop over all non-empty source boxes

For $m \in \mathcal{I}_3(n)$ ← outside the neighborhood of the n -th box

Get $\mathbf{x}_c^{(m)}$, the center of the box;

$\mathbf{D}^{(n)} = \mathbf{D}^{(n)} + (\mathbf{S|R})(\mathbf{x}_c^{(n)} - \mathbf{x}_c^{(m)}) \mathbf{C}^{(m)}$;

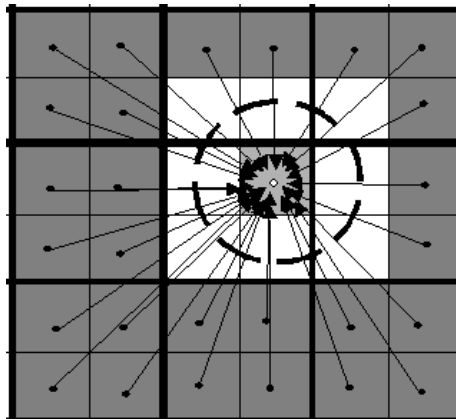
End;

End;

Implementation can be different!

All we need is to get $\mathbf{D}^{(n)}$.

S|R-translation



SLFMM Algorithm

Step 3. Final Summation

$$v_j = \hat{\Phi}(\mathbf{y}_j) = \sum_{\mathbf{x}_i \in E_2(n)} \hat{\Phi}(\mathbf{y}_j, \mathbf{x}_i) + \mathbf{D}^{(n)} \circ \mathbf{R}(\mathbf{y}_j - \mathbf{x}_c^{(n)}), \quad \mathbf{y}_j \in E_1(n).$$

For $n \in \text{NonEmptyEvaluation}$ ← loop over all boxes containing evaluation points
 Get $\mathbf{x}_c^{(n)}$, the center of the box;
For $\mathbf{y}_j \in E_1(n)$ ← loop over all evaluation points in the box
 $v_j = \mathbf{D}^{(n)} \circ \mathbf{R}(\mathbf{y}_j - \mathbf{x}_c^{(n)})$;
For $\mathbf{x}_i \in E_2(n)$ ← loop over all sources in the neighborhood of the n -th box
 $v_j = v_j + \Phi(\mathbf{y}_j, \mathbf{x}_i)$;
End;
End;
End;

Implementation can be different!
All we need is to get v_j

FFTM

$$D_n^{(l)} = \sum_{m=0}^{p-1} \left[\sum_{k \in I_3(l)} (S|R)_{nm}(x_{*l} - x_{*k}) C_m^{(k)} \right], \quad l = 1, \dots, K, \quad n = 0, \dots, p-1.$$

Redefine:

$$(S|R)_{nm}(x_{*l} - x_{*k}) = \begin{cases} (S|R)_{nm}(x_{*l} - x_{*k}), & k \in I_3(l), \\ 0, & k \notin I_3(l). \end{cases}$$

This is a regular function sampled on a grid. One can use the FFT for convolution!

Convolution and FFT

$$f(y_j) = \sum_{k=0}^{N-1} \Phi(y_j - x_k) u_k, \quad j = 0, \dots, M-1.$$

$$\Phi(y) = \sum_{l=0}^{L-1} c_l e^{ily}.$$

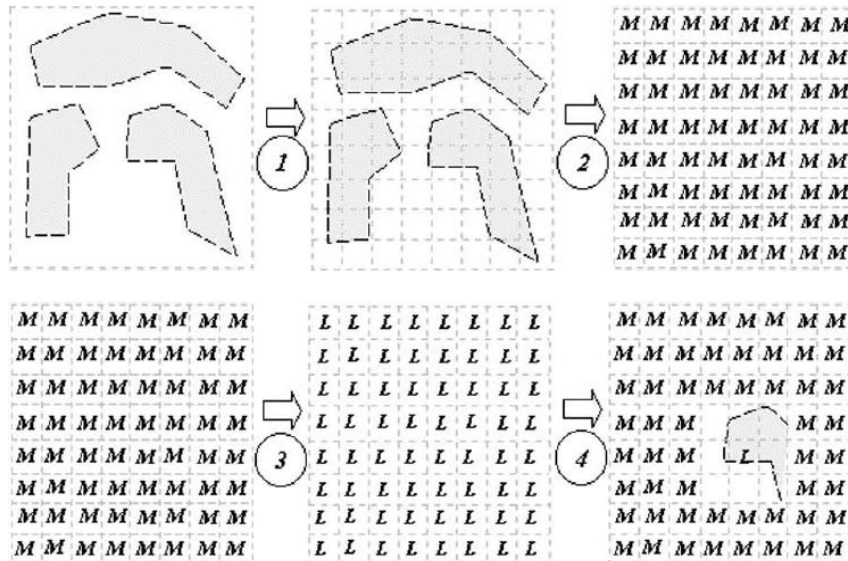
$$f(y_j) = \sum_{k=0}^{N-1} \sum_{l=0}^{L-1} c_l e^{il(y_j - x_k)} u_k = \sum_{k=0}^{N-1} \sum_{l=0}^{L-1} c_l e^{ily_j} e^{-ilx_k} u_k = \sum_{l=0}^{L-1} c_l e^{ily_j} \sum_{k=0}^{N-1} e^{-ilx_k} u_k.$$

$$b_l = \sum_{k=0}^{N-1} e^{-ilx_k} u_k, \quad a_l = c_l b_l, \quad l = 0, \dots, L-1,$$

$$f(y_j) = \sum_{l=0}^{L-1} a_l e^{ily_j}, \quad j = 0, \dots, M-1.$$

This can be extended to d-
dimensions

- So, for any kernel, which can be approximated by Fourier series $O(N \log N)$ algorithm exist to compute convolution.



Comparison of the FFTM and FMM

- FFTM has better error bounds;
- FFTM can be faster for relatively uniform distributions;
- FFTM does not require hierarchical data structures;
- But... for non-uniform distributions FMM, perhaps, is faster...

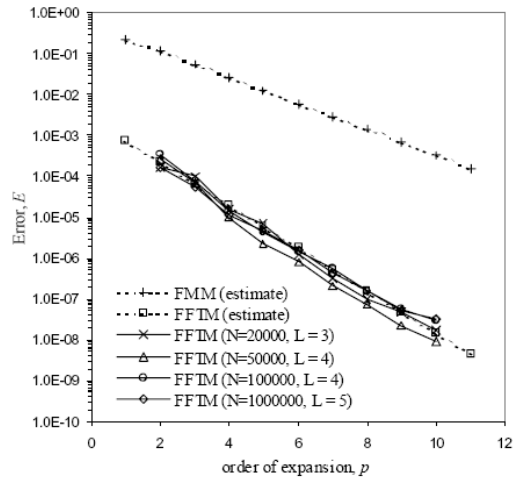


Fig. 4. Convergence behavior of FFTM for cube example.

From Ong, et al. (2003).
 Comparison with a non-efficient FMM

Non Uniform FFT

Lecture 23

NUFFT

- Uniform and Nonuniform Discrete Fourier Transforms
 - DFT, FFT, NUFFT, IDFT, IFFT, INUFFT
 - Statement of the problems
- INUFFT
 - Data Structure
 - Kernel Decomposition
 - “Middleman” part of the algorithm
 - FMM part of the algorithm
 - Error bounds
 - Complexity and Optimization
- Some Numerical Results
 - Error Analysis
 - Performance

Band-limited functions

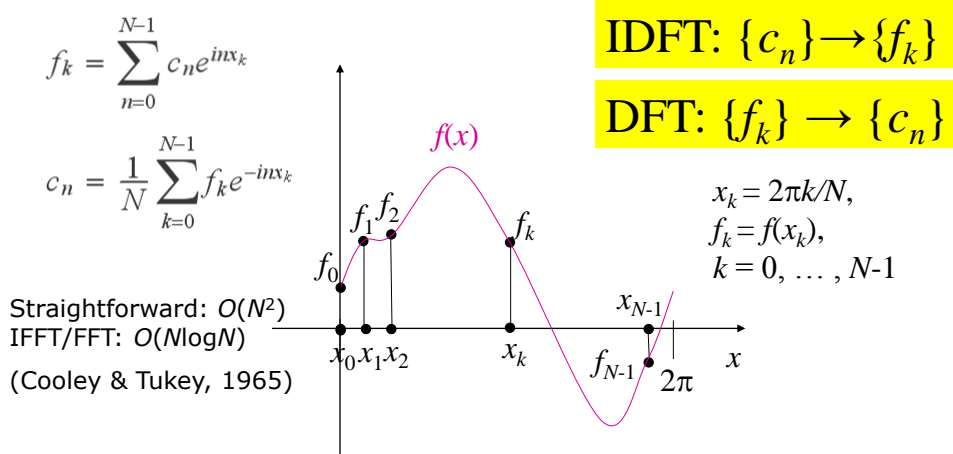
$$f(x) = \sum_{n=0}^{N-1} c_n e^{inx}$$

Diagram illustrating the equation $f(x) = \sum_{n=0}^{N-1} c_n e^{inx}$ with annotations:

- $f(x)$: 2π -periodic function (complex)
- c_n : Fourier coefficients (complex)
- $N-1$: bandwidth
- e^{inx} : real

some people like functions
$$F(x) = f(x)e^{-iNx/2} = \sum_{n=-N/2}^{N/2-1} d_n e^{inx} \quad (d_n = c_{n-N/2})$$

Inverse and Forward Discrete Fourier Transform




Fast Fourier Transform

- Divide and conquer type $O(N \log N)$ algorithm;
- Exact in exact arithmetic;
- Revolutionized computations:
 - Signal processing;
 - Scientific computing;
 - Large data sets handling/compression;
 - Many other applications...
- Requires equispaced (uniform) data

Fast Fourier Transform

- FFT Requires equispaced (uniform) data.
- In many applications data is not uniform:
 - Jitter;
 - Non-uniform grids for computations;
 - Events that occur randomly
 - Etc.
- An algorithm of complexity $O(N\log N)$ to perform the forward and inverse Fourier Transform for such cases is highly desired.

Publications on the NUFFT

- Dutt & Rokhlin (1993):
 - Interpolation using Gaussians;
 - Forward problem $O(N\log N)$;
 - Inverse problem: iteratively $O(N_{\text{iter}}N\log N)$.
 - Dutt & Rokhlin (1995):
 - FMM for kernel $\cot(t/2)$ and $\ln(\sin(t/2))$; 
 - Forward problem $O(N\log N)$;
 - Inverse problem: $O(N\log N)$.
 - Beylkin (1995):
 - Interpolation using multiresolution analysis (MRA).
 - Liu & Nguyen (1997,1998):
 - Least square approximations;
 - Forward problem $O(N\log N)$;
 - Inverse problem: iteratively, $O(N_{\text{iter}}N\log N)$.
- OUR WORK IS CLOSE TO THIS ONE, WHILE FACTORIZATION AND DATA STRUCTURE ARE DIFFERENT

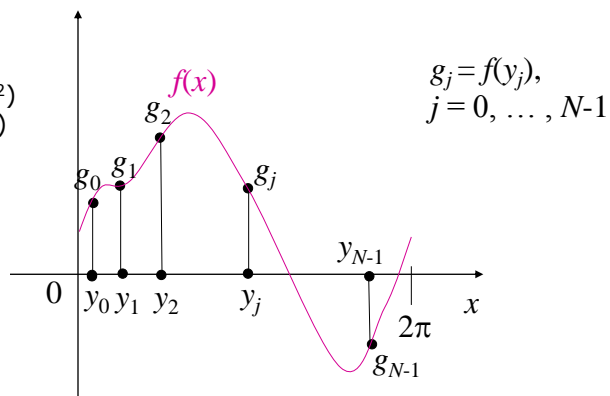
Statement of the Problem

Inverse Nonuniform Discrete Fourier Transform

INUDFT: $\{c_n\} \rightarrow \{g_j\}$

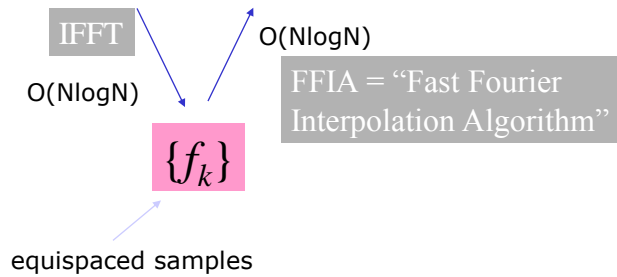
$$g_j = \sum_{n=0}^{N-1} c_n e^{iny_j}$$

Straightforward: $O(N^2)$
Looking for: $O(N \log N)$
(INUFFT)



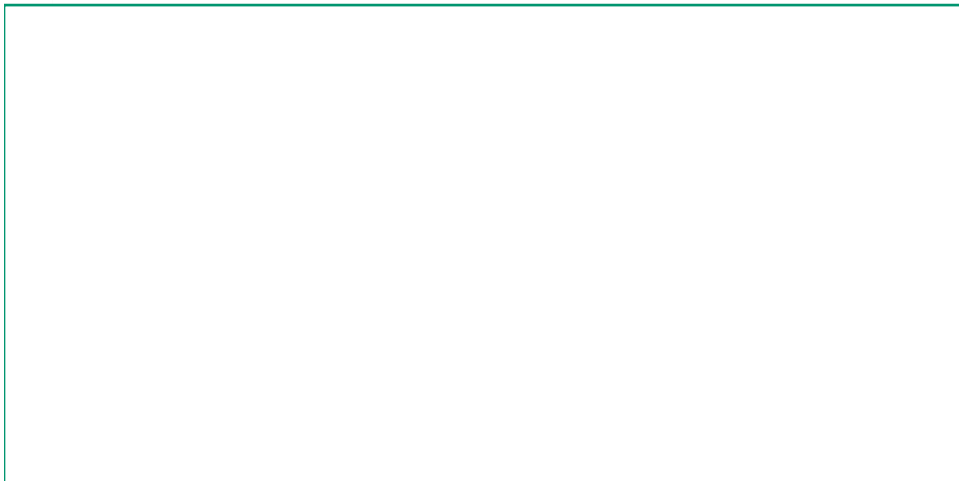
INUDFT: Reduction to Interpolation Problem

INUFFT: $\{c_n\} \rightarrow \{g_j\}$



FFIA

$$g_j = f(y_j) = \sum_{n=0}^{N-1} c_n e^{iny_j} = \sum_{k=0}^{N-1} \left[\frac{1}{N} \sum_{n=0}^{N-1} e^{-ins_k} e^{iny_j} \right] f_k, \quad j = 0, \dots, N-1,$$



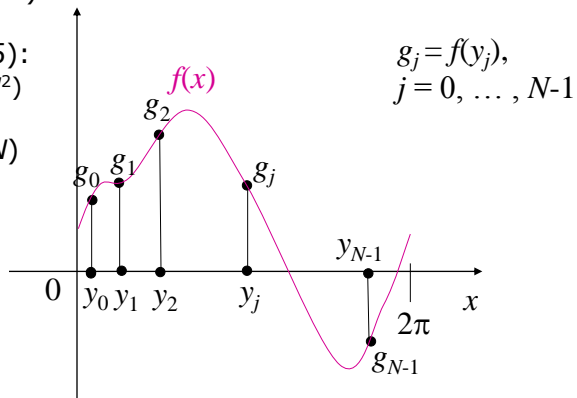
Forward Nonuniform Discrete Fourier Transform

NUDFT: $\{g_j\} \rightarrow \{c_n\}$

Straightforward matrix inversion: $O(N^3)$

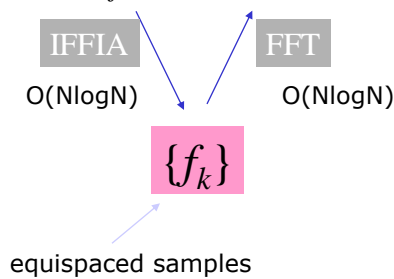
Dutt & Rokhlin (1995):
Straightforward: $O(N^2)$

Looking for: $O(N \log N)$
(NUFFT)



NUDFT: Reduction to Inverse Interpolation Problem

NUFFT: $\{g_j\} \rightarrow \{c_n\}$



Inverse Interpolation Problem

$$\{f_k\} = \{L_{kj}\}\{g_j\}, \quad \{L_{kj}\} = \{K_{jk}\}^{-1}$$

Theorem (Dutt & Rokhlin, 1995):

$$L_{kj} = C_k G(x_k - y_j) D_j, \quad k, j = 0, \dots, N-1,$$

$$C_k = 2i(-1)^k \prod_{l=0}^{N-1} \sin \frac{x_k - y_l}{2},$$

$$D_j = e^{-iNy_j/2} \prod_{l=0, l \neq j}^{N-1} \left(\sin \frac{y_j - y_l}{2} \right)^{-1}.$$

Computation of Products

$$C_k = 2i(-1)^k e^{\Phi(x_k)}, \quad D_k = e^{-iNy_k/2} e^{-\Psi_k}, \quad k = 1, \dots, N,$$

$$\Phi(x) = \ln \prod_{j=0}^{N-1} \sin \frac{x - y_j}{2} = \sum_{j=0}^{N-1} \ln \left(\sin \frac{x - y_j}{2} \right),$$

$$\Psi_k = \ln \prod_{\substack{j=0, \\ j \neq k}}^{N-1} \sin \frac{y_k - y_j}{2} = \sum_{\substack{j=0, \\ j \neq k}}^{N-1} \ln \left(\sin \frac{y_k - y_j}{2} \right).$$

$$\Phi(x)|_{x=y_k} \rightarrow \ln \left(\sin \frac{x - y_k}{2} \right) + \Psi_k.$$

$$\Phi'(x) = \frac{1}{2} \sum_{j=0}^{N-1} \cot \frac{x - y_j}{2} = \frac{1}{2} \sum_{j=0}^{N-1} H(x - y_j) = \phi(x),$$

$$\Phi(x) = \int_{x_*}^x \phi(x) dx + \Phi(x_*),$$

FFIA and IFFIA can be reduced to
computation of sums:

$$h(y_j) = \sum_{k=0}^{N-1} H(y_j - x_k) f_k$$

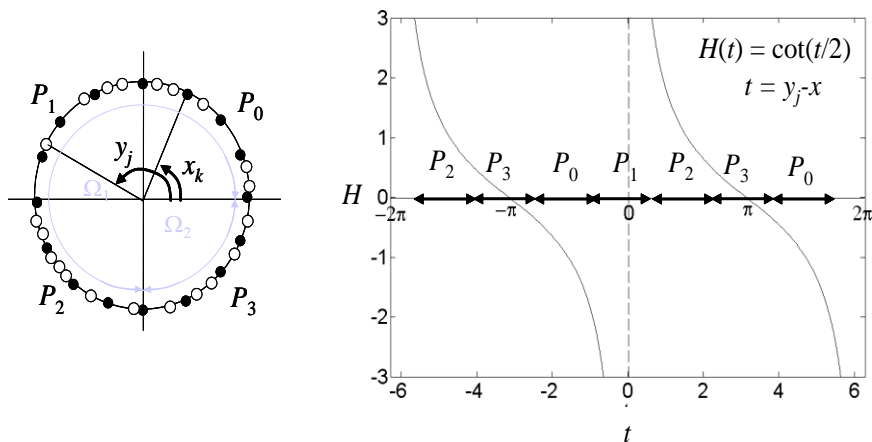
$$e(y_j) = \int_{y_*}^{y_j} \left[\sum_{k=0}^{N-1} H(y - x_k) \mathbf{d}y \right] + e(y_*)$$

$$H(t) = \cot\left(\frac{t}{2}\right)$$

Here x_k and y_j can be arbitrary (non-uniform)
We can perform fast summation using the Fast Multipole Methods

Solution of the Problem

Kernel Decomposition



Kernel Decomposition (2)

B_{2m} – Bernoulli numbers

$$H(t) = \cot \frac{t}{2} = H_1(t) + H_2(t), \quad H_1(t) = \frac{2}{t}, \quad H_2(t) = -2 \sum_{m=1}^{\infty} \frac{|B_{2m}|}{(2m)!} t^{2m-1}, \quad |t| < 2\pi,$$

$$x_k \in \Omega_1(P_n) : -\pi \leq y_j - \tilde{x}_k \leq \pi, \quad |t| = |y_j - \tilde{x}_k| \leq \pi.$$

$$h(y_j) = h^{(1)}(y_j) + h^{(2)}(y_j) = \sum_{x_k \in \Omega_1(P_n)} f_k H(y_j - \tilde{x}_k) + \sum_{x_k \in \Omega_2(P_n)} f_k H(y_j - \tilde{x}_k),$$

$$H(y_j - \tilde{x}_k) = \cot \frac{y_j - \tilde{x}_k}{2} = -\tan \frac{t}{2} = -2 \sum_{m=1}^{\infty} \frac{(2^{2m} - 1)|B_{2m}|}{(2m)!} t^{2m-1}, \quad |t| < \pi,$$

$$x_k \in \Omega_2(P_n) : -\pi/2 \leq y_j - \hat{x}_k \leq \pi/2, \quad |t| = |y_j - \hat{x}_k| \leq \pi/2.$$

Bernoulli Numbers

Generating Function:

$$\frac{t}{e^t - 1} = \sum_{n=0}^{\infty} B_n \frac{t^n}{n!}, \quad |t| < 2\pi;$$

Relation to the Riemann Zeta Function:

$$|B_{2m}| = \frac{2(2m)!}{(2\pi)^{2m}} \zeta(2m), \quad \zeta(s) = \sum_{k=1}^{\infty} k^{-s}, \quad m = 1, 2, \dots;$$

Bounds:

$$\frac{2(2m)!}{(2\pi)^{2m}} < |B_{2m}| < \frac{2(2m)!}{(2\pi)^{2m}} \frac{1}{1 - 2^{1-2m}}, \quad m = 1, 2, \dots;$$

Rational. First 60 numbers can be found in Abramowitz and Stegun (1964):

$$B_2 = \frac{1}{6}, \quad B_4 = -\frac{1}{30}, \quad B_6 = \frac{1}{42}, \quad B_8 = -\frac{1}{30}, \quad B_{10} = \frac{5}{66}, \quad B_{12} = -\frac{691}{2730}, \dots$$

Factorization of the regular part of the Kernel

$$t^{2m-1} = [(y_j - x_c^{(n)}) - (\bar{x}_k - x_c^{(n)})]^{2m-1} = \sum_{l=0}^{2m-1} \frac{(2m-1)!}{l!(2m-1-l)!} (y_j - x_c^{(n)})^l (x_c^{(n)} - \bar{x}_k)^{2m-1-l}.$$

$$H_2^{(q)}(t) = -2 \sum_{m=1}^q \frac{|B_{2m}|}{(2m)!} t^{2m-1} = - \sum_{m=1}^q \frac{|B_{2m}|}{m} \sum_{l=0}^{2m-1} \frac{1}{l!(2m-1-l)!} (y_j - x_c^{(n)})^l (x_c^{(n)} - \bar{x}_k)^{2m-1-l},$$

$$h^{(1)}(y_j) = \sum_{x_k \in \Omega_1(\mathcal{F}_n)} f_k H_1(y_j - x_k) - \sum_{l=0}^{2q-1} \frac{a_l}{l!} (y_j - x_c^{(n)})^l + \delta_q^{(1)},$$

$$h^{(2)}(y_j) = - \sum_{l=0}^{2q-1} \frac{b_l}{l!} (y_j - x_c^{(n)})^l + \delta_q^{(2)},$$

$$a_i = \sum_{m=\lfloor i/2 \rfloor + 1}^q \frac{|B_{2m}|}{m} a_{2m-i}^{(1)}, \quad a_i^{(1)} = \frac{1}{i!} \sum_{x_k \in \Omega_1(\mathcal{F}_n)} f_k (x_c^{(n)} - \bar{x}_k)^i,$$

$$b_i = \sum_{m=\lfloor i/2 \rfloor + 1}^q \frac{|B_{2m}|}{m} (2^{2m} - 1) a_{2m-i}^{(2)}, \quad a_i^{(2)} = \frac{1}{i!} \sum_{x_k \in \Omega_2(\mathcal{F}_n)} f_k (x_c^{(n)} - \hat{x}_k)^i.$$

Error bounds for the regular part

$$|\epsilon_q^{(1)}| = |H_2(t) - H_2^{(q)}(t)| = 2 \left| \sum_{m=q+1}^{\infty} \frac{|B_{2m}| t^{2m-1}}{(2m)!} \right| < \frac{2^{2-2q}}{3\pi(1-2^{-2q-1})} = 2\epsilon_q, \quad |t| < \pi,$$

$$|\epsilon_q^{(2)}| = 2 \left| \sum_{m=q+1}^{\infty} \frac{(2^{2m}-1)|B_{2m}|}{(2m)!} t^{2m-1} \right| < \frac{2^{3-2q}}{3\pi(1-2^{-2q-1})} = 4\epsilon_q, \quad |t| < \pi/2,$$

$$|\delta_q| = |\delta_q^{(1)} + \delta_q^{(2)}| < |\delta_q^{(1)}| + |\delta_q^{(2)}| < \frac{3N}{4} 2\epsilon_q + \frac{N}{4} 4\epsilon_q = \frac{5N\epsilon_q}{2}.$$

For FFIA:

$$|\epsilon_q^{reg}| = \left| \sum_{k=0}^{N-1} (K_{jk}^{reg} - K_{jk}^{(q)reg}) f_k \right| < \frac{2}{N} \frac{1}{2} |\delta_q| = \frac{5\epsilon_q}{2}.$$

Solution

Middleman for the regular part

Since we have polynomial factorization of the regular part for the entire domain, we can apply the "Middleman" method (the fastest possible) to compute this part.

Complexity: $O(qN)$:

For the forward interpolation problem (FFIA):
 $q \sim \log_4(1/\epsilon)$ (does not depend on N);

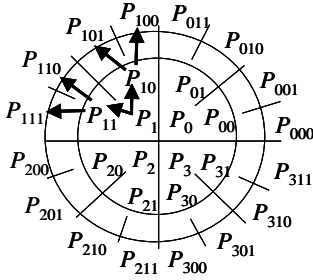
For the inverse interpolation problem (IFFIA):
 $q \sim \log_4(N/\epsilon)$.

Multilevel FMM for the singular part

Kernel Function:

$$H_1(y - \tilde{x}_k) = \frac{2}{y - \tilde{x}_k}.$$

Data Structure with
Circular Topology



Error Bound (Gumerov, Duraiswami & Borovikov, 2003):

$$|\epsilon_p^{(FMM)}| < \frac{10}{L_{\min}} N_s 3^{-p},$$

$$L_{\min} = (3\pi/2)2^{-l_{\max}}, \quad N_s = 3N/4.$$

$$|\epsilon_p^{(FMM)}| < N \frac{10}{2\pi} 2^{l_{\max}} 3^{-p},$$

Error for the FFIA:

$$|\epsilon_p^{sing}| = \left| \sum_{k=0}^{N-1} (K_{jk}^{sing} - K_{jk}^{(p)sing}) f_k \right| < \frac{2}{N} \frac{1}{2} |\epsilon_p^{(FMM)}| = \frac{5}{\pi} 2^{l_{\max}} 3^{-p}.$$

Complexity and optimization of the FMM

$$Complexity_{FMM} = O\left(2Np + \frac{3N^2}{2^{l_{\max}}} + \frac{6p^2}{2^{-l_{\max}}}\right),$$

$$l_{\max}^{(opt)} \sim \log_2 \frac{N}{p\sqrt{2}},$$

$$C_{FFIA}^{(opt)} = O\left(2N[p(3\sqrt{2} + 1) + 2q]\right).$$

$$C_{FFIA}^{(opt)} = O(N \log \epsilon^{-1}),$$

$$C_{IFFIA}^{(opt)} = O(N[\log \epsilon^{-1} + \log N]).$$

Some performance results

