Fast Multipole Methods: Fundamentals & Applications

Ramani Duraiswami
Nail A. Gumerov
Week 1. Introduction.

- What are multipole methods and what is this course about.
- Problems from physics, mathematics, computer vision, statistics, etc. that can be solved with fast multipole methods.
- Historical review of the fast multipole method.
- Review of Complexity
- Homework.
Course Mechanics

- **Background Needed**
  - **Linear Algebra**
    - Matrices, Vectors, Linear Systems, LU Decomposition, Eigenvalue problem, SVD
  - **Numerical Analysis**
    - (Interpolation, Approximation, Finite Differences, Taylor Series, etc.)
  - **Programming**
    - Matlab, C/C++ and/or FORTRAN
  - For particular applications in your domain you will need to know the background material there
  - E.g., familiarity with partial differential equations or integral equations, or interpolation, or …

- **Participation essential!**
Course Requirements

• Ideally
  – you are an applied student in engineering, physics or CS who has a problem in mind and would like to explore how the FMM could be applied to it
  – or, you are Applied Math/Scientific Computation student who wants to learn this important algorithm

• At the end of the course you will have a familiarity with
  – Application of FMM to different problems
  – Data structures for FMM, and analysis related to it.

• Course focus is on applying the methods to achieve solution.
  – Theory as needed to proceed
Homework

- Will try to have it every week
- Will not be excessive
- Essential for learning --- must do as opposed to just read.
- Homework handed out last class of a week.
- Due last class of next week
- Thanksgiving week no homework
  - (you can turn in the previous homework after thanksgiving)
Quizzes

- To provide us feedback
- Experimental …
- Typically last 10 minutes of lecture on alternate Thursdays
- Will cover material from previous week
- Will account for 10 points
Projects & Exams

- There will be a final project that will require you to implement an FMM algorithm in a field of your choice, account for 20% of the grade.
- Project to be chosen latest by October 14.
  - Implementation (reimplementation) of an FMM algorithm (hints and help will be given)
  - If you already have a project in mind you can discuss it with us
- Exams
  - intermediate exam worth 10%, week of October 14
  - final exam worth 20%. Finals Week
Class web & Mailing List

- http://www.umiacs.umd.edu/~ramani/cmsc878R
- cmsc878r@umiacs.umd.edu
  - Discussions, announcements
- Homework will be posted on the web
  - No late homework (except by timely prior arrangement)
  - Hardcopy (no web or email submissions)
- Solutions will be posted after homework collected
- Links to papers etc.
Introductions

- Email addresses
- What are your interests?
- What do you want us to cover?
  - Last years outline is posted at the course web site
What is the Fast Multipole Method?

- An algorithm for achieving fast products of particular dense matrices with vectors
- Similar to the Fast Fourier Transform
  - For the FFT, matrix entries are uniformly sampled complex exponentials
- For FMM, matrix entries are
  - Derived from particular functions
  - Functions satisfy known “translation” theorems
- Name is a bit unfortunate
  - What the heck is a multipole? We will return to this …
- Why is this important?
Vectors and Matrices

\(d\) dimensional column vector \(\mathbf{x}\) and its transpose

\[
\mathbf{x} = \begin{pmatrix}
    x_1 \\
    x_2 \\
    \vdots \\
    x_d \\
\end{pmatrix}
\]

\[
\text{and} \quad \mathbf{x}^t = (x_1 \ x_2 \ \ldots \ x_d)
\]

- \(n \times d\) dimensional matrix \(\mathbf{M}\) and its transpose \(\mathbf{M}^t\)

\[
\mathbf{M} = \begin{pmatrix}
    m_{11} & m_{12} & m_{13} & \ldots & m_{1d} \\
    m_{21} & m_{22} & m_{23} & \ldots & m_{2d} \\
    \vdots & \vdots & \vdots & \ddots & \vdots \\
    m_{n1} & m_{n2} & m_{n3} & \ldots & m_{nd} \\
\end{pmatrix}
\]

\[
\text{and} \quad \mathbf{M}^t = \begin{pmatrix}
    m_{11} & m_{12} & \ldots & m_{1d} \\
    m_{12} & m_{22} & \ldots & m_{2d} \\
    \vdots & \vdots & \ddots & \vdots \\
    m_{1d} & m_{2d} & \ldots & m_{nd} \\
\end{pmatrix}
\]
Matrix vector product

\[ s_1 = m_{11} x_1 + m_{12} x_2 + \ldots + m_{1d} x_d \]
\[ s_2 = m_{21} x_1 + m_{22} x_2 + \ldots + m_{2d} x_d \]
\[ \ldots \]
\[ s_n = m_{n1} x_1 + m_{n2} x_2 + \ldots + m_{nd} x_d \]

- \( d \) products and sums per line
- \( N \) lines
- Total \( Nd \) products and \( Nd \) sums to calculate \( N \) entries

- Matrix vector product is identical to a sum
  \[ s_i = \sum_{j=1}^{d} m_{ij} x_j \]
- So algorithm for fast matrix vector products is also a fast summation algorithm
Linear Systems

- Solve a system of equations
  \[ Mx = s \]
- \( M \) is a \( N \times N \) matrix, \( x \) is a \( N \) vector, \( s \) is a \( N \) vector
- Direct solution (Gauss elimination, LU Decomposition, SVD, …) all need \( O(N^3) \) operations
- Iterative methods typically converge in \( k \) steps with each step needing a matrix vector multiply \( O(N^2) \)
  - if properly designed, \( k \ll N \)
- A fast matrix vector multiplication algorithm (\( O(N \log N) \) operations) will speed all these algorithms
Is this important?

• Argument:
  – Moore’s law: Processor speed doubles every 18 months
  – If we wait long enough the computer will get fast enough and let my inefficient algorithm tackle the problem

• Is this true?
  – Yes for algorithms with same asymptotic complexity
  – No!! For algorithms with different asymptotic complexity

• For a million variables, we would need about 16 generations of Moore’s law before a $O(N^2)$ algorithm was comparable with a $O(N)$ algorithm

• Similarly, clever problem formulation can also achieve large savings.
Memory complexity

- Sometimes we are not able to fit a problem in available memory
  - Don’t care how long solution takes, just if we can solve it
- To store a $N \times N$ matrix we need $N^2$ locations
  - 1 GB RAM = $1024^3 = 1,073,741,824$ bytes
  - => largest $N$ is 32,768
- “Out of core” algorithms copy partial results to disk, and keep only necessary part of the matrix in memory
- FMM allows reduction of memory complexity as well
  - Elements of the matrix required for the product can be generated as needed
The need for fast algorithms

- Grand challenge problems in large numbers of variables
- Simulation of physical systems
  - Electromagnetics of complex systems
  - Stellar clusters
  - Protein folding
  - Turbulence
- Learning theory
  - Kernel methods
  - Support Vector Machines
- Graphics and Vision
  - Light scattering …
• General problems in these areas can be posed in terms of millions \((10^6)\) or billions \((10^9)\) of variables
• Recall Avogadro’s number \((6.022\ 141\ 99 \times 10^{23})\) molecules/mole
Dense and Sparse matrices

- Operation estimates are for **dense matrices**.
  - Majority of elements of the matrix are **non-zero**
- However in many applications matrices are **sparse**
- A sparse **matrix** (or **vector**, or **array**) is one in which most of the elements are zero.
  - If storage space is more important than access speed, it may be preferable to store a sparse matrix as a list of (index, value) pairs.
  - For a given sparsity structure it may be possible to define a fast matrix-vector product/linear system algorithm
• Can compute

\[
\begin{bmatrix}
    a_1 & 0 & 0 & 0 & 0 \\
    0 & a_2 & 0 & 0 & 0 \\
    0 & 0 & a_3 & 0 & 0 \\
    0 & 0 & 0 & a_4 & 0 \\
    0 & 0 & 0 & 0 & a_5
\end{bmatrix}
\begin{bmatrix}
    x_1 \\
    x_2 \\
    x_3 \\
    x_4 \\
    x_5
\end{bmatrix}
=
\begin{bmatrix}
    a_1x_1 \\
    a_2x_2 \\
    a_3x_3 \\
    a_4x_4 \\
    a_5x_5
\end{bmatrix}
\]

In 5 operations instead of 25 operations
• Sparse matrices are not our concern here
Structured matrices

• Fast algorithms have been found for many dense matrices
• Typically the matrices have some “structure”
• Definition:
  – A dense matrix of order $N \times N$ is called structured if its entries depend on only $O(N)$ parameters.
• Most famous example – the fast Fourier transform
Fourier Matrices

A Fourier matrix of order $n$ is defined as the following

$$F_n = \begin{bmatrix}
1 & 1 & 1 & \cdots & 1 \\
1 & \omega_n & \omega_n^2 & \cdots & \omega_n^{n-1} \\
1 & \omega_n^2 & \omega_n^4 & \cdots & \omega_n^{2(n-1)} \\
\cdots & \cdots & \cdots & \cdots & \cdots \\
1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \cdots & \omega_n^{(n-1)(n-1)}
\end{bmatrix},$$

where

$$\omega_n = e^{-\frac{2\pi i}{n}},$$

is an $n$th root of unity.
FFT and IFFT

The *discrete Fourier transform* of a vector $x$ is the product $F_n x$.
The *inverse discrete Fourier transform* of a vector $x$ is the product $F_n^* x$.

Both products can be done efficiently using the fast Fourier transform (FFT) and the inverse fast Fourier transform (IFFT) in $O(n \log n)$ time.

The FFT has revolutionized many applications by reducing the complexity by a factor of almost $n$.

Can relate many other matrices to the Fourier Matrix
Circulant Matrices

\[ C_n = C(x_1, \ldots, x_n) = \begin{bmatrix} x_1 & x_n & x_{n-1} & \cdots & x_2 \\ x_2 & x_1 & x_n & \cdots & x_3 \\ x_3 & x_2 & x_1 & \cdots & x_4 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_n & x_{n-1} & x_{n-2} & \cdots & x_1 \end{bmatrix} \]

Toeplitz Matrices

\[ T_n = T(x_{-n+1}, \ldots, x_0, \ldots, x_{n-1}) = \begin{bmatrix} x_0 & x_1 & x_2 & \cdots & x_{n-1} \\ x_{-1} & x_0 & x_1 & \cdots & x_{n-2} \\ x_{-2} & x_{-1} & x_0 & \cdots & x_{n-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{-n+1} & x_{-n+2} & x_{-n+3} & \cdots & x_0 \end{bmatrix} \]

Hankel Matrices

\[ H_n = H(x_{-n+1}, \ldots, x_0, \ldots, x_{n-1}) = \begin{bmatrix} x_{-n+1} & x_{-n+2} & x_{-n+3} & \cdots & x_0 \\ x_{-n+2} & x_{-n+3} & x_{-n+4} & \cdots & x_1 \\ x_{-n+3} & x_{-n+4} & x_{-n+5} & \cdots & x_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_0 & x_1 & x_2 & \cdots & x_{n-1} \end{bmatrix} \]

Vandermonde Matrices

\[ V = V(x_0, x_1, \ldots, x_n) = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ x_0 & x_1 & \cdots & x_{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ x_0^{n-1} & x_1^{n-1} & \cdots & x_{n-1}^{n-1} \end{bmatrix} \]
• Modern signal processing very strongly based on the FFT
• One of the defining inventions of the 20th century
Fast Multipole Methods (FMM)

- Introduced by Rokhlin & Greengard in 1987
- Called one of the 10 most significant advances in computing of the 20th century
- Speeds up matrix-vector products (sums) of a particular type
  \[ s(x_j) = \sum_{i=1}^{N} \alpha_i \phi(x_j - x_i), \quad \{s_j\} = [\Phi_{ji}]\{\alpha_i\}. \]
- Above sum requires \(O(MN)\) operations.
- For a given precision \(\epsilon\) the FMM achieves the evaluation in \(O(M+N)\) operations.
• Can accelerate matrix vector products
  – Convert $O(N^2)$ to $O(N \log N)$
• However, can also accelerate linear system solution
  – Convert $O(N^3)$ to $O(kN \log N)$
A very simple algorithm

- Not FMM, but has some key ideas
- Consider
  \[ S(x_i) = \sum_{j=1}^{N} \alpha_j (x_i - y_j)^2 \quad i=1, \ldots, M \]
- Naïve way to evaluate the sum will require \( MN \) operations
- Instead can write the sum as
  \[ S(x_i) = \left( \sum_{j=1}^{N} \alpha_j \right) x_i^2 + \left( \sum_{j=1}^{N} \alpha_j y_j^2 \right) - 2x_i \left( \sum_{j=1}^{N} \alpha_j y_j \right) \]
  - Can evaluate each bracketed sum over \( j \) and evaluate an expression of the type
    \[ S(x_i) = \beta x_i^2 + \gamma - 2x_i \delta \]
  - Requires \( O(M+N) \) operations
- Key idea – use of analytical manipulation of series to achieve faster summation
Approximate evaluation

• FMM introduces another key idea or “philosophy”
  – In scientific computing we almost never seek exact answers
  – At best, “exact” means to “machine precision”
• So instead of solving the problem we can solve a “nearby” problem that gives “almost” the same answer
• If this “nearby” problem is much easier to solve, and we can bound the error analytically we are done.
• In the case of the FMM
  – Manipulate series to achieve approximate evaluation
  – Use analytical expression to bound the error
• FFT is exact … FMM can be arbitrarily accurate
Some FMM algorithms

• Molecular and stellar dynamics
  – Computation of force fields and dynamics
• Interpolation with Radial Basis Functions
• Solution of acoustical scattering problems
  – Helmholtz Equation
• Electromagnetic Wave scattering
  – Maxwell’s equations
• Fluid Mechanics: Potential flow, vortex flow
  – Laplace/Poisson equations
• Fast nonuniform Fourier transform
Applications – I  Interpolation

• Given a scattered data set with points and values \( \{x_i, f_i\} \)
• Build a representation of the function \( f(x) \)
  – That satisfies \( f(x_i) = f_i \)
  – Can be evaluated at new points
• One approach use “radial-basis functions”
  \[
  f(x) = \sum_{i=1}^{N} \alpha_i R(x - x_i) + p(x)
  \]
  \[
  f_j = \sum_{i=1}^{N} \alpha_i R(x_j - x_i) + p(x_j)
  \]
• Two problems
  – Determining \( \alpha_i \)
  – Knowing \( \alpha_i \) determine the product at many new points \( x_j \)
• Both can be solved via FMM (Cherrie et al, 2001)
Applications 2

- RBF interpolation

Cherrie et al 2001
Applications 3

- Sound scattering off rooms and bodies
  - Need to know the scattering properties of the head and body (our interest)

\[ \nabla^2 P + k^2 P = 0 \]
\[ \frac{\partial P}{\partial n} + i\sigma P = g \]
\[ \lim_{r \to \infty} r \left( \frac{\partial P}{\partial r} - ikP \right) = 0 \]

\[ C(x)p(x) = \int_{\Gamma_y} \left[ G(x, y, k) \frac{\partial}{\partial n_y} p(y) - \frac{\partial}{\partial n_y} G(x, y, k) p(y) \right] d\Gamma_y \]

\[ G(x, y) = \frac{e^{ik|x-y|}}{4\pi|x-y|} \]
EM wave scattering

- Similar to acoustic scattering
- Send waves and measure scattered waves
- Attempt to figure out object from the measured waves
- Need to know “Radar cross-section”
- Many applications
  - Light scattering
  - Radar
  - Antenna design
  - ....

Darve 2001
Molecular and stellar dynamics

- Many particles distributed in space
- Particles exert a force on each other
- Simplest case force obeys an inverse-square law (gravity, coulombic interaction)

\[
\frac{d^2 \mathbf{x}_i}{dt^2} = F_i,
\quad F_i = \sum_{\substack{j=1 \atop j \neq i}}^{N} q_i q_j \frac{(\mathbf{x}_i - \mathbf{x}_j)}{|\mathbf{x}_i - \mathbf{x}_j|^3}
\]

Figure 10: Slice views of the SDB cluster at time 0 and 1.16 ns. The slices are passing the sphere center with thickness of 70 Å.
Fluid mechanics

- Incompressible Navier Stokes Equation

\[ \nabla \cdot \mathbf{u} = 0 \]

\[ \rho \left( \frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} \right) + \nabla p = \mu \nabla^2 \mathbf{u} \]

\[ \mathbf{u} = \nabla \phi + \nabla \times \mathbf{A} \]

- Laplace equation for potential and Poisson equation for vorticity

- Solved via particle methods …
Asymptotic Equivalence

\[
\lim_{n \to \infty} \left( \frac{f(n)}{g(n)} \right) = 1
\]
Little Oh

- *Asymptotically smaller*:
  
  \[ f(n) = o(g(n)) \]

\[
\lim_{n \to \infty} \left( \frac{f(n)}{g(n)} \right) = 0
\]
Big Oh

• Asymptotic Order of Growth:
  • $f(n) = O(g(n))$

$$\limsup_{n \to \infty} \left( \frac{f(n)}{g(n)} \right) < \infty$$
The Oh’s

If $f = o(g)$ or $f \sim g$ then $f = O(g)$

$\lim = 0$     $\lim = 1$     $\lim < \infty$
The Oh’s

If $f = o(g)$, then $g \neq O(f)$

$$\lim_{x \to \infty} \frac{f}{g} = 0 \quad \lim_{x \to \infty} \frac{g}{f} = \infty$$
Big Oh

• Equivalently,

\[ f(n) = O(g(n)) \]

\[ \exists c, n_0 \geq 0 \ \forall n \geq n_0 \ \ |f(n)| \leq c \cdot g(n) \]
Big Oh

\[ f(x) = O(g(x)) \]
Complexity

- The most common complexities are
  - $O(1)$ - not proportional to any variable number, i.e. a fixed/constant amount of time
  - $O(N)$ - proportional to the size of $N$ (this includes a loop to $N$ and loops to constant multiples of $N$ such as $0.5N$, $2N$, $2000N$ - no matter what that is, if you double $N$ you expect (on average) the program to take twice as long)
  - $O(N^2)$ - proportional to $N$ squared (you double $N$, you expect it to take four times longer - usually two nested loops both dependent on $N$).
  - $O(\log N)$ - this is trickier to show - usually the result of binary splitting.
  - $O(N \log N)$ this is usually caused by doing $\log N$ splits but also doing $N$ amount of work at each "layer" of splitting.
Theta

Same Order of Growth:

\[ f(n) = \Theta(g(n)) \]

\[ f(n) = O(g(n)) \text{ and } g(n) = O(f(n)) \]
Log complexity

- If you half data at each stage then number of stages until you have a single item is given (roughly) by $\log_2 N$. $\Rightarrow$ binary search takes $\log_2 N$ time to find an item.

- All logs grow a constant amount apart (homework)
  - So we normally just say $\log N$ not $\log_2 N$.

- Log $N$ grows very slowly
History of FMM

- Rokhlin and Greengard
- Greengard, ACM thesis award
- Rokhlin & Greengard Steele prize
- Regular FMM
- Complexity
- Translation
- Chew, Darve, Michielssen
Brief Historical Review on Fast Multipole Methods
Outline

• Separable (Degenerate) Kernels
• Problems with Infinite Series
• First Fast Solvers
• 2D Laplace Equation
• 3D Laplace Equation
• 2D Poisson Equation
• Fast Gauss Transform
• 2D Helmholtz Equation
• 3D Helmholtz Equation
• 3D Maxwell Equations
• 1D Problems
• Other Equations
Separable (Degenerate) Kernels

Compute matrix-vector product

\[ v = Au, \]

or sums

\[ v_j = \sum_{i=1}^{N} u_i A(x_i, y_j), \quad j = 1, \ldots, M. \]

Fast computation in case of degenerate (separable) kernel

\[ A(x_i, y_j) = \sum_{m=1}^{n} \varphi_m(x_i) \psi_m(y_j). \]

\[ v_j = \sum_{i=1}^{N} u_i \sum_{m=1}^{n} \varphi_m(x_i) \psi_m(y_j) = \sum_{m=1}^{n} \psi_m(y_j) \sum_{i=1}^{N} u_i \varphi_m(x_i) = \sum_{m=1}^{n} c_m \psi_m(y_j), \]

where

\[ c_m = \sum_{i=1}^{N} u_i \varphi_m(x_i). \]

Authors: Unknown.
Problems with Infinite Series

The case of degenerate kernels is not the FMM!

Compute matrix-vector product

\[ v = Au, \]

or sums

\[ v_j = \sum_{i=1}^{N} u_i A(x_i, y_j), \quad j = 1, \ldots, M. \]

Non-degenerate kernel:

\[ A(x_i, y_j) = \sum_{m=1}^{\infty} \varphi_m(x_i) \psi_m(y_j) = \sum_{m=1}^{p} \varphi_m(x_i) \psi_m(y_j) + \text{Error}(p; x_i, y_j) \]

where \( p \) is the truncation number.

Features of the FMM:

1). Factorization is not obvious and should be selected somehow.
2). Error bounds should be established.
3). Series converge in some spatial domains. Need to have data structures and translation technique to avoid divergent series and uncontrolled error.
4) A lot of analytical work!
First Fast Solvers

Fast computation of the Laplacian gravitational fields for interstellar interactions:


2D Laplace Equation

\[ \nabla^2 \Phi = \frac{\partial^2 \Phi}{\partial x^2} + \frac{\partial^2 \Phi}{\partial y^2} = 0. \]

Fundamental solution (charge, monopole, source, free field Green’s function):

\[ G_{x_0,y_0}(x,y) = -\frac{1}{2\pi} \ln r, \quad r = \sqrt{(x-x_0)^2 + (y-y_0)^2}. \]

Satisfies

\[ \frac{\partial^2 G_{x_0,y_0}}{\partial x^2} + \frac{\partial^2 G_{x_0,y_0}}{\partial y^2} = \delta(x-x_0), \quad x=(x,y), \quad x_0=(x_0,y_0). \]

Field generated by a set of \( N \) monopoles:

\[ \Phi(x) = \sum_{i=1}^{N} q_i G_{x_i}(x) = \sum_{i=1}^{N} q_i G(x - x_i). \]


1. Introduced translation operators for 2D Laplace Equation;
2. Introduced hierarchical space subdivision based on quad-trees for data structuring in the FMM.
3. First known publications on the FMM.

Also known as MLFMA (MultiLevel Fast Multipole Algorithm)

CMSC878R/AMSC698R Fall 2003 © Gumerov & Duraiswami, 2002 - 2003
2D Laplace Equation
(Greengard’s scheme of translation)
3D Laplace Equation

\[ \nabla^2 \Phi = \frac{\partial^2 \Phi}{\partial x^2} + \frac{\partial^2 \Phi}{\partial y^2} + \frac{\partial^2 \Phi}{\partial z^2} = 0. \]

Fundamental solution (charge, monopole, source, free field Green’s function):

\[ G_{x_0,y_0,z_0}(x,y,z) = \frac{1}{4\pi r}, \quad r = \sqrt{(x-x_0)^2 + (y-y_0)^2 + (z-z_0)^2}. \]

Satisfies

\[ \frac{\partial^2 G_{x_0,y_0,z_0}}{\partial x^2} + \frac{\partial^2 G_{x_0,y_0,z_0}}{\partial y^2} + \frac{\partial^2 G_{x_0,y_0,z_0}}{\partial z^2} = -\delta(x-x_0), \quad x = (x,y,z), \quad x_0 = (x_0,y_0,z_0). \]

Field generated by a set of \( N \) monopoles:

\[ \Phi(x) = \sum_{i=1}^{N} q_i G_{x_i}(x) = \sum_{i=1}^{N} q_i G(x-x_i). \]


1). Introduced translation operators for 3D Laplace Equation;

2). Introduced hierarchical space subdivision based on oct-trees for data structuring in the FMM.

One of the latest developments:


CMSC878R/AMSC698R Fall 2003 © Gumerov & Duraiswami, 2002 - 2003
Adaptive FMM for 2D Poisson Equation

\[ \nabla^2 \Phi = \frac{\partial^2 \Phi}{\partial x^2} + \frac{\partial^2 \Phi}{\partial y^2} = F(x,y). \]


1) Introduced an adaptive quad-tree space subdivision for 2D Poisson equation. Good for very non-uniform source point distributions.
Fast Gauss Transform

\[ \Phi(x) = \sum_{i=1}^{N} q_i e^{-\|x-x_i\|^2/\sigma} \]

1). Use of the Hermit expansions and Taylor series for different domains (far field and near field).
2). Spatial grouping based on source and evaluation points location using interaction lists.


2D Helmholtz Equation

\[ \nabla^2 \Phi + k^2 \Phi = \frac{\partial^2 \Phi}{\partial x^2} + \frac{\partial^2 \Phi}{\partial y^2} + k^2 \Phi = 0. \]

Fundamental solution (charge, monopole, source, free field Green’s function):

\[ G_{x_0,y_0}(x,y) = \frac{1}{2\pi} \mathcal{H}_0^1(kr), \quad r = \sqrt{(x-x_0)^2 + (y-y_0)^2}. \]

\( \mathcal{H}_0^1(kr) \) is the first kind Hankel function.

\[ \text{Satisfies} \]

\[ \frac{\partial^2 G_{x_0,y_0}}{\partial x^2} + \frac{\partial^2 G_{x_0,y_0}}{\partial y^2} + k^2 G_{x_0,y_0} = -\delta(x-x_0), \quad x=(x,y), \quad x_0=(x_0,y_0). \]

Field generated by a set of \( N \) monopoles:

\[ \Phi(x) = \sum_{i=1}^{N} q_i G_{x_0}(x) = \sum_{i=1}^{N} q_i G(x-x_i). \]


1). Translation operators for 2D Helmholtz Equation;
2). Error bounds;
2). Spatial grouping.
3D Helmholtz Equation

\[ \nabla^2 \Phi + \kappa^2 \Phi = \frac{\partial^2 \Phi}{\partial x^2} + \frac{\partial^2 \Phi}{\partial y^2} + \frac{\partial^2 \Phi}{\partial z^2} + \kappa^2 \Phi = 0. \]

Fundamental solution (charge, monopole, source, free field Green's function):

\[ G_{x_0, y_0}(x, y) = \frac{1}{4\pi r} e^{ikr}, \quad r = \sqrt{(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2}. \]

Satisfies

\[ \frac{\partial^2 G_{x_0, y_0}}{\partial x^2} + \frac{\partial^2 G_{x_0, y_0}}{\partial y^2} + \frac{\partial^2 G_{x_0, y_0}}{\partial z^2} + \kappa^2 G = -\delta(x - x_0), \quad x = (x, y), \quad x_0 = (x_0, y_0). \]

Field generated by a set of $N$ monopoles:

\[ \Phi(x) = \sum_{i=1}^{N} q_i G(x_i) = \sum_{i=1}^{N} q_i G(x - x_i). \]

1. Translation operators for 3D Helmholtz Equation;
2. Spatial grouping.
3D Maxwell Equations

\[ \nabla \times \mathbf{E} = -\mu \frac{\partial \mathbf{H}}{\partial t}, \]
\[ \nabla \times \mathbf{H} = \varepsilon \frac{\partial \mathbf{E}}{\partial t}, \]
\[ \nabla \cdot \mathbf{E} = 0, \]
\[ \nabla \cdot \mathbf{H} = 0, \]

where \( \mathbf{E} \) and \( \mathbf{H} \) are the electric and magnetic field vectors, and \( \mu \) and \( \varepsilon \) are permeability and permittivity in the medium, respectively. In the case of vacuum we have

\[ \mu = \mu_0, \quad \varepsilon = \varepsilon_0, \quad c = (\mu_0 \varepsilon_0)^{-1/2}, \]

where \( c \) is the speed of light in a vacuum, \( c \approx 3 \times 10^8 \) m/s.


1D Problems: Interpolation, Differentiation, Integration

Fast Algorithms for Polynomial Interpolation, Integration, and Differentiation
A. Dutt, M. Gu, V. Rokhlin


1). Considered the FMM for fast Lagrange polynomial interpolation;
2). Fast summation and operations with series of polynomials.
Other Equations

- Biharmonic (Stokes Flows)
- Yukawa Potentials (molecular dynamics)
Our papers ...