
Lecture 1

Instructor: Carlos Pérez Arancibia

MIT
Mathematics Department
Cambridge, September 7, 2017
Basic information

- Instructor: Carlos Pérez-Arancibia (cperezar@mit.edu), office 2-336
- Office hours: E-mail instructor
- Room: 2-143
- Time: MW 1:00-2:30
- Course website: http://math.mit.edu/~cperezar/18.336.html
Important dates

› 10/06: Add date

› 11/02: Project proposal due

› 11/21 and 11/23: Mid-term meeting to discuss progress on the final report

› 12/12: Final project report due

› 12/07 and 12/12: Final project presentations
Course grades

- 50% problem sets (~4): Lowest grade is dropped. Deadline on Thursdays in class.

- 50% final project: Maximum 5 pages conference-paper style report, presentation and code. You can code your project in any language you want but I suggest you to use: Matlab, Python, Julia, C, C++ or Fortran 90.
<table>
<thead>
<tr>
<th>Function</th>
<th>Fortran</th>
<th>Julia</th>
<th>Python</th>
<th>R</th>
<th>Matlab</th>
<th>Octave</th>
<th>Mathematica</th>
<th>JavaScript</th>
<th>Go</th>
<th>LuaJIT</th>
<th>Java</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>gcc 5.1.1</td>
<td>0.4.0</td>
<td>3.4.3</td>
<td>3.2.2</td>
<td>R2015b</td>
<td>4.0.0</td>
<td>10.2.0</td>
<td>V8 3.28.71.19</td>
<td>go1.5</td>
<td>gsl-shell 2.3.1</td>
<td>1.8.0_45</td>
</tr>
<tr>
<td>fib</td>
<td>0.70</td>
<td>2.11</td>
<td>77.76</td>
<td>533.52</td>
<td>26.89</td>
<td>9324.35</td>
<td>118.53</td>
<td>3.36</td>
<td>1.86</td>
<td>1.71</td>
<td>1.21</td>
</tr>
<tr>
<td>parse_int</td>
<td>5.05</td>
<td>1.45</td>
<td>17.02</td>
<td>45.73</td>
<td>802.52</td>
<td>9581.44</td>
<td>15.02</td>
<td>6.06</td>
<td>1.20</td>
<td>5.77</td>
<td>3.35</td>
</tr>
<tr>
<td>quicksort</td>
<td>1.31</td>
<td>1.15</td>
<td>32.89</td>
<td>264.54</td>
<td>4.92</td>
<td>1866.01</td>
<td>43.23</td>
<td>2.70</td>
<td>1.29</td>
<td>2.03</td>
<td>2.60</td>
</tr>
<tr>
<td>mandel</td>
<td>0.81</td>
<td>0.79</td>
<td>15.32</td>
<td>53.16</td>
<td>7.58</td>
<td>451.81</td>
<td>5.13</td>
<td>0.66</td>
<td>1.11</td>
<td>0.67</td>
<td>1.35</td>
</tr>
<tr>
<td>pi_sum</td>
<td>1.00</td>
<td>1.00</td>
<td>21.99</td>
<td>9.56</td>
<td>1.00</td>
<td>299.31</td>
<td>1.69</td>
<td>1.01</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>rand_mat_stat</td>
<td>1.45</td>
<td>1.66</td>
<td>17.93</td>
<td>14.56</td>
<td>14.52</td>
<td>30.93</td>
<td>5.95</td>
<td>2.30</td>
<td>2.96</td>
<td>3.27</td>
<td>3.92</td>
</tr>
<tr>
<td>rand_mat_mul</td>
<td>3.48</td>
<td>1.02</td>
<td>1.14</td>
<td>1.57</td>
<td>1.12</td>
<td>1.12</td>
<td>1.30</td>
<td>15.07</td>
<td>1.42</td>
<td>1.16</td>
<td>2.36</td>
</tr>
</tbody>
</table>

**Announcement:** Prof. Steven Johnson will hold a Julia tutorial session on Monday 11th from 5 to 7 PM. Location to be announced.
Prerequisites

- Knowledge of ODEs and basic linear PDEs (Laplace, Poisson, heat and wave equations)
- Fourier transform and Fast Fourier Transform (FFT)
- Linear algebra: Gaussian elimination, LU, QR and singular value decompositions, iterative linear algebra solvers
- Basic numerical methods: Approximation theory, finite difference and finite element methods
- Programming: Matlab, Python, Julia, C, C++, Fortran 90
Class topics

- Problem formulation: PDE vs. boundary and volume integral equations

- Boundary integral equation method: Collocation, Galerkin Boundary Element Method (BEM) and Nyström methods for boundary integral equations, finite differences, spectral and finite element methods.

- Numerical methods for PDEs: Finite differences, spectral and finite element methods.

- Fast algorithms: FFT based-methods, Fast Multipole Method (FMM), Hierarchical matrices (H-Matrices), butterfly algorithms and multigrid.
Top ten algorithms of the century*

1. Monte Carlo method
2. Simplex method of linear programming
3. Krylov subspace iteration method
   - iterative solution of linear systems
4. Householder matrix decomposition
5. Fortran compiler
6. QR algorithm for eigenvalue calculation
7. Quicksort algorithm
8. Fast Fourier transform
   - fast matrix-vector products and fast linear system solution for dense structured matrices
9. Integer relation detection algorithm
10. Fast Multipole Method
    - fast matrix-vector products for dense unstructured matrices

Motivation: $N$-body problem

The celebrated $N$-body problem of astronomy is the following:

*Each particle in a system of a finite number of particles is subjected to a Newtonian gravitational attraction from all the other particles, and to no other forces. If the initial state of the system is given, how will the particles move?*

- The 2-body problem was fully solve by Johann Bernoulli in 1734
- Many famous scientists have worked on this problem including Newton, Euler, Lagrange, Poincaré, Delaunay and Moulton. There is not known explicit solutions for the $N$-body problem for $N>2$. 
Motivation: $N$-body problem

In general, we have

$$m_i \frac{d^2 \mathbf{x}_i(t)}{dt^2} = -\nabla_{\mathbf{x}_i} \Phi, \quad i = 1, \ldots, N$$

where the potential is given by

$$\Phi = \Phi_{\text{far}} + \{\Phi_{\text{near}} + \Phi_{\text{external}}\}$$

Applications:

- Classical celestial mechanics
- Plasma physics
- Molecular dynamics
- Vortex methods
Motivation: $N$-body problem

In general, we have

$$m_i \frac{d^2 \mathbf{x}_i(t)}{dt^2} = -\nabla_{\mathbf{x}_i} \Phi, \quad i = 1, \ldots, N$$

where the potential is given by

$$\Phi = \Phi_{\text{far}} + \{\Phi_{\text{near}} + \Phi_{\text{external}}\}$$

$\Phi_{\text{near}}$: Van der Walls forces that decay fast ($|\mathbf{x}_i - \mathbf{x}_j|^{-6}$ in the case of Lennard-Jones potential)

$\Phi_{\text{external}}$: It is independent of the number of particles

$\Phi_{\text{near}}$ can be evaluated in $O(N)$ operations

$\Phi_{\text{external}}$ can also be evaluated in order $O(N)$ operations
Motivation: $N$-body problem

In general, we have

$$m_i \frac{d^2 \mathbf{x}_i(t)}{dt^2} = -\nabla_{\mathbf{x}_i} \Phi, \quad i = 1, \ldots, N$$

where the potential is given by

$$\Phi = \Phi_{\text{far}} + \{\Phi_{\text{near}} + \Phi_{\text{external}}\}$$

$\Phi_{\text{near}}$: Van der Walls forces that decay fast ($|\mathbf{x}_i - \mathbf{x}_j|^{-6}$ in the case of Lennard-Jones potential)

$\Phi_{\text{external}}$: It is independent of the number of particles

$$\Phi_{\text{far}}(\mathbf{x}_i) = \sum_{j=1, i \neq j}^{N} \frac{Gm_i m_j}{|\mathbf{x}_i - \mathbf{x}_j|}$$

Slow decay: far-field interactions cannot be neglected

The overall cost of evaluating the right-hand-side is $O(N^2)$
Motivation: \(N\)-body problem

A hierarchical \(O(N \log N)\) force-calculation algorithm

Josh Barnes & Piet Hut

The Institute for Advanced Study, School of Natural Sciences, Princeton, New Jersey 08540, USA

Until recently the gravitational \(N\)-body problem has been modelled numerically either by direct integration, in which the computation needed increases as \(N^2\), or by an iterative potential method in which the number of operations grows as \(N \log N\). Here we describe a novel method of directly calculating the force on \(N\) bodies that grows only as \(N \log N\). The technique uses a tree-structured hierarchical subdivision of space into cubic cells, each of which is recursively divided into eight subcells whenever more than one particle is found to occupy the same cell. This tree is constructed anew at every time step, avoiding ambiguity and tangle. Advantages over potential-solving codes are: accurate local interactions; freedom from geometrical assumptions and restrictions; and applicability to a wide class of systems, including (proto-)planetary, stellar, galactic and cosmological ones. Advantages over previous hierarchical tree-codes include simplicity and the possibility of rigorous analysis of error. Although we concentrate here on stellar dynamical applications, our techniques of efficiently handling a large number of long-range interactions and concentrating computational effort where most needed have potential applications in other areas of astrophysics as well.

Fig. 2 Box structure induced by a three-dimensional particle distribution. This example was taken from the early stages of an encounter of two \(N = 64\) systems, and shows how the boxing algorithm can accommodate systems with arbitrarily complicated geometry. The particle distribution corresponding to a system with 32 times as many members is shown in Fig. 3.
A Fast Algorithm for Particle Simulations*

L. Greengard and V. Rokhlin

Department of Computer Science, Yale University,
New Haven, Connecticut 06520

Received June 10, 1986; revised February 5, 1987

An algorithm is presented for the rapid evaluation of the potential and force fields in systems involving large numbers of particles whose interactions are Coulombic or gravitational in nature. For a system of $N$ particles, an amount of work of the order $O(N^2)$ has traditionally been required to evaluate all pairwise interactions, unless some approximation or truncation method is used. The algorithm of the present paper requires an amount of work proportional to $N$ to evaluate all interactions to within roundoff error, making it considerably more practical for large-scale problems encountered in plasma physics, fluid dynamics, molecular dynamics, and celestial mechanics.
Motivation: $N$-body problem

Applets for the $N$-Body Problem:

Motivation: Poisson’s equation

Say we want to determine the potential produced by a given mass (charge) density:

$$\Delta u = f \quad \text{in} \quad \mathbb{R}^d \quad (f \in C_0^\infty(\mathbb{R}^d))$$

$u$: potential \quad $f$: mass (charge) density

The exact solution is given by:

$$u(x) = \int_{\mathbb{R}^d} G(x, y) f(y) \, dy, \quad x \in \mathbb{R}^d$$

where $G$ is the (free space) Green’s function of the Laplace’s equation:

$$\Delta_x G(x, y) = \delta_y \quad \text{in} \quad \mathbb{R}^d$$
Motivation: Poisson’s equation

The exact solution is given by:

\[
u(x) = \int_{\mathbb{R}^d} G(x, y) f(y) \, dy, \quad x \in \mathbb{R}^d
\]

\[
G(x, y) = \begin{cases} 
\frac{1}{2\pi} \log(|x - y|) & \text{if } d = 2, \\
-\frac{1}{4\pi|x - y|} & \text{if } d = 3
\end{cases}
\]

it’s a convolution!

Discretization:

\[
u(x_j) \approx \sum_{i} W_i G(x_j - y_i) f(y_i)
\]

\(W_i: \text{quadrature weights} \quad y_i: \text{quadrature points}\)

using the FFT this can be evaluated in \(O(N \log N)\) operations!
Motivation: Poisson’s equation

Especial case in 1D:

\[ \Delta u(x) = f(x) \quad x \in (0, 1), \]
\[ u(0) = 0, \quad u(1) = 0 \]

Spatial discretization:

\[ x_j = jh, \quad j = 0, \ldots, N + 1, \quad h = \frac{1}{N + 1}, \]

\[ x_0 = 0 \quad x_{N+1} = 1 \]

\[ x_1 \quad x_2 \quad \cdots \quad x_{N-1} \quad x_N \]
Motivation: Poisson’s equation

Especial case in 1D:

\[ \Delta u(x) = f(x) \quad x \in (0, 1), \]
\[ u(0) = 0, \quad u(1) = 0 \]

Using a finite difference discretization:

\[ u''(x_j) = \frac{u(x_j + h) - 2u(x_j) + u(x_j - h)}{h^2} + O(h^2) \]

we form the linear system:

\[ \Delta_h u_h = f_h \]

where

\[ \Delta_h = -\frac{1}{h^2} \begin{bmatrix} 2 & -1 \\ -1 & 2 & \ddots \\ & \ddots & \ddots & \ddots \\ & & \ddots & 2 & -1 \\ & & & -1 & 2 \end{bmatrix} \in \mathbb{R}^{N \times N} \]
Motivation: Poisson’s equation

Linear system: \( \Delta_h u_h = f_h \)

\[
\Delta_h = -\frac{1}{h^2} \begin{bmatrix}
2 & -1 & & & \\
-1 & 2 & \ddots & & \\
& \ddots & \ddots & \ddots & \\
& & \ddots & 2 & -1 \\
& & & -1 & 2
\end{bmatrix}
\]

Solution: \( u_h = \Delta_h^{-1} f_h \)

The overreaching goal in the design of numerical methods is to solve the problem in \( O(N) \) operations.
Motivation: Poisson’s equation

Linear system: \[ \Delta_h u_h = f_h \]

\[ \Delta_h = \begin{bmatrix} \ddots \end{bmatrix} \]

LU decomposition costs \( O(N) \) operations in this case

\[ Lx = f_h \quad U u_h = x \]
forward substitution: \( O(N) \) operations  backward substitution: \( O(N) \) operations
## Solving the Discrete Poisson Equation using Jacobi, SOR, Conjugate Gradients, and the FFT

Motivation: Poisson’s equation

… unfortunately, in general we have

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Type</th>
<th>Serial</th>
<th>PRAM</th>
<th>Storage</th>
<th>#Procs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dense LU</td>
<td>D</td>
<td>N^3</td>
<td>N</td>
<td>N^2</td>
<td>N^2</td>
</tr>
<tr>
<td>Band LU</td>
<td>D</td>
<td>N^2</td>
<td>N</td>
<td>N^(3/2)</td>
<td>N</td>
</tr>
<tr>
<td>Inv(P)*bhat</td>
<td>D</td>
<td>N^2</td>
<td>log N</td>
<td>N^2</td>
<td>N^2</td>
</tr>
<tr>
<td>Jacobi</td>
<td>I</td>
<td>N^2</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Sparse LU</td>
<td>D</td>
<td>N^(3/2)</td>
<td>N^(1/2)</td>
<td>N*log N</td>
<td>N</td>
</tr>
<tr>
<td>CG</td>
<td>I</td>
<td>N^(3/2)</td>
<td>N^(1/2)*log N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>SOR</td>
<td>I</td>
<td>N^(3/2)</td>
<td>N^(1/2)</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>FFT</td>
<td>D</td>
<td>N*log N</td>
<td>log N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Multigrid</td>
<td>I</td>
<td>N</td>
<td>(log N)^2</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Lower Bound</td>
<td>N</td>
<td>log N</td>
<td>N</td>
<td>N</td>
<td></td>
</tr>
</tbody>
</table>

Key to abbreviations:
- Dense LU: Gaussian elimination, treating P as dense
- Band LU: Gaussian elimination, treating P as zero outside a band of half-width n-1 near diagonal
- Sparse LU: Gaussian elimination, exploiting entire zero-structure of P
- Inv(P)*bhat: precompute and store inverse of P, multiply it by right-hand-side bhat
- CG: Conjugate Gradient method
- SOR: Successive Overrelaxation
- FFT: Fast Fourier Transform based method

https://people.eecs.berkeley.edu/~demmel/cs267/lecture24/lecture24.html
Two major topics of fast methods

In general, the kind of Fast Methods that we will study in this class exploit two important features of the problem:

I. Translational invariance

\[
\Delta_h = -\frac{1}{h^2} \begin{bmatrix}
2 & -1 \\
-1 & 2 \\
& & \ddots \\
& & & \ddots \\
& & & & \ddots \\
& & & & & 2 & -1 \\
& & & & -1 & 2
\end{bmatrix}
\]

constant diagonals

\[
G(x, y) = \begin{cases}
\frac{1}{2\pi} \log(|x - y|) & \text{if } d = 2, \\
-\frac{1}{4\pi |x - y|} & \text{if } d = 3
\end{cases}
\]

is a function of $|x - y|$

Fast Fourier Transform based methods are the preferred fast methods for translationally invariant problems, giving $O(N \log N)$ complexity.
Two major topics of fast methods

In general, the kind of Fast Methods that we will study in this class exploit two important features of the problem:

II. Low-rank interactions

At the continuous level:

\[ G(x, y) \approx \sum_{p=1}^{P} \phi_p(x) \psi_p(y) \]

At the discrete level:

\[ A = UV^T = \sum_{i=1}^{k} u_i v_i^T \]

... main idea behind the Fast Multiple method, the butterfly algorithm and \( \mathcal{H} \)-matrices