# A simplified $\mathcal{O}(n \log n)$ direct method for inverse n-body problems.

Thomas Ben Thompson

December 6, 2014

## Introduction

Many problems in physics, statistics, mathematics and computer science can be formulated as a generalized n-body problem, where the goal is to compute sums of the form:

$$f_i = \sum_{j=1}^{M} K(\|x_i - y_j\|)u_j \quad \forall i = 1, \ldots, N \tag{1}$$

where, often $x_i$ are thought of as "observation" particle locations, $y_j$ as "source" particle locations, $u_j$ are the (known) source strengths and $K(\|x - y\|)$ is an (distance-dependent) interaction kernel. The problem can be thought of as a matrix-vector product with a special form of matrix.

Many of the efficient techniques for this problem were originally formulated for the classical n-body problem in astrophysics, where, with $u_j$ as the mass of particle $j$,

$$K(\|x_i - y_j\|) = \frac{1}{\|x_i - y_j\|^2} \tag{2}$$

representing the Newtonian gravitational influence of a unit mass at $x_j$ on a unit mass at $y_j$.

A naive algorithm to compute these sums requires time $O(NM)$. Except in special cases, this is a lower bound on the running time for any exact algorithm. However, few applications require exact evaluation. Even computing the sums to floating point precision can be done in $\widetilde{O}(N + M)$ because $K(x, y)$ varies slowly for large $\|x - y\|$. In other words, the problem is much easier because, under the condition of two particles being well-separated, the strength of the interaction between two particles is weakly dependent on the exact position of those particles. I will describe the Barnes-Hut "treecode" (CITE barnes-hut) method for evaluating these sums in time $O(N \log M + M \log M)$. I will also show how a few modifications to the treecode method gives the influential fast multipole method (CITE Greengard).

A variation on the N-body problem switches the knowns and unknowns such that the goal is the compute $u_j$ given $f_i$. Later, I will give an example where this problem arises in the solution of the partial differential equations describing faulting and flexure in Earth's crust (CITE CRUSE). Two other applications are in the inversion of dense covariance matrices in machine learning and in radial basis function approximation.

Analogous to the representation of the n-body problem as a matrix-vector product, this **"inverse" n-body problem** can be represented by a linear system:

$$Ku = f \tag{3}$$

$$K_{ij} = K(\|x_i - y_j\|) \tag{4}$$

A naive application of Gaussian elimination or an LU decomposition to solve this problem requires $O(N^3)$ time and $O(N^2)$ space (CITE GOLUB VAN LOAN). In some applications, $N > 1,000,000$, making an $O(N^3)$ algorithm prohibitively expensive.

Given a solution to the n-body problem, a fast algorithm to the inverse n-body problem can be built using an iterative linear solver. The time cost of an iterative method is $O(IV)$ where $V$ is the cost of a

matrix-vector product, and $I$ is the number of iterations. Using a fast n-body algorithm, $V = O(N)$, giving $O(IN)$. $I$ is normally some function of the condition number and the iterative solver used. For solving a single well conditioned problems, this is suitable. However, for a poorly conditioned problem, or for a case where the linear system must be solved with many different vectors $f$, a method that directly inverts the matrix (called a "direct" method, Gaussian elimination is an example) would be preferable. Recent research has produced direct methods that are able to take advantage of special matrix structure, specifically a hierarchically low-rank property, to reduce the time cost to $O(n^{3/2})$ or $O(n)$ (citepMartinsson2005, Greengard2009, Ambikasaran2014).

I will discuss a recent extension to the fast multipole method that performs a direct matrix inversion in $O(N)$ time. Due to the complexity of the algorithm, I will only cover the general ideas, instead developing a simplification that achieves suboptimal $O(N \log N)$ time, while being easier to implement and understand.

# Treecodes

The fundamental idea of a treecode or the a fast multipole method is to approximate far-field interactions with the goal of decoupling summation over sources and evaluation of each observation. I follow the presentation by CITEFONGANDDARVE. First, I will consider a case in which this decoupling can be performed globally.

TODO: Better description of the interpolation process.

Without loss of generality, I assume that we are considering a 1D problem, $x_i, y_j \in [-1, 1]$ and that $M = N$. Suppose that the kernel, $K(\|x - y\|)$, is smooth everywhere, with no singularities. In this case, a interpolative approximation should be high-accuracy:

$$K(\|x - y\|) = \sum_{l=1}^{p} K(\|x - \tilde{y}_l\|) v_l(\tilde{y}_l, y) \tag{5}$$

where $\tilde{y}_l$ are the nodes of the interpolation. The value of this low-rank approximation is that summation over $j$ and evaluation of each $i$ can be decoupled:

$$f_i = \sum_{j=1}^{N} \left( \sum_{l=1}^{p} K(\|x - \tilde{y}_l\|) v_l(\tilde{y}_l, y) \right) u_j = \sum_{l=1}^{p} K(\|x - \tilde{y}_l\|) \sum_{j=1}^{N} v_l(\tilde{y}_l, y) u_j \tag{6}$$

Now, by evaluating the inner sum once and storing the result, the full sums can be evaluated in time $O(N+pM)$. This mathematical approximation and re-arrangement can be physically interpreted as bundling all the sources together and computing the effect at an observation point from this bundle instead of the original individual points.

The trouble is that most $K(\|x-y\|)$ are not smooth when $x = y$. The gravitational potential $\frac{1}{\|x-y\|^2}$, for example, has a singularity at $x = y$. Now, the idea of far-field interaction of "well-separatedness" becomes important. If we consider points from two disjoint subintervals $x \in [a_x, b_x]$ and $y \in [a_y, b_y]$, then this restricted $K(\|x-y\|)$ is once again smooth, because there is no pair of points in those subintervals such that $x = y$. So, the plan for the treecode code is to hierarchically divide space, and bundle the points in each subdivision together. Then, far-field interactions will be computed from these bundles.

The algorithm is:

1. Subdivide the interval $[-1, 1]$ until no interval has more than $A$ point.

2. Compute the weights of the interpolation for each subinterval $I$ in the leaves of the tree. This step is often called the P2M operator, short for "particle to multipole".

$$W_m^I = \sum_{y_j \in I} v_m(\tilde{y}_m, y_j) u_j \quad \forall m = 1, \ldots, p \tag{7}$$

3. Recurse up the tree and compute the weights, treating the weights at a node's child nodes just like the individual particles were treated in step 2. This step is often called the M2M operator, short for "multipole to multipole".

$$W_m^I = \sum_{J,\text{child of } I} \sum_k v_m(\tilde{y}_m, \tilde{y}_k) W_k^J \quad \forall m = 1, \ldots, p \tag{8}$$

4. For each observation particle, $x_i$, perform a depth-first traversal of the tree. Define far-field nodes to obey the inequality $\frac{D}{R} \leq \theta$, where $D$ is the diameter/width of the node's interval, and $R$ is the distance from the center of the interval to the current observation particle. At any step of the traversal there are three possibilities:

   (a) The current node is a far-field node. Compute the interaction between its weights and $x_i$.

   (b) The current node is a near-field non-leaf node. Recurse to each of its children.

   (c) The current node is a near-field leaf node. Directly compute the interaction between the source particles $y_j$ and $x_i$.

The analysis of this algorithm is relatively straightforward. Using a structure like a kd-tree for step 1 results in a maximum depth of $O(\log N)$ and $O(N)$ time at each level for median-finding using a linear time median finding algorithm, resulting in an initialization time of $O(N \log N)$. Steps 2 requires one step per particle-weight interaction: $O(pN)$ time. Step 3 requires one step per weight at each level. With $O(N)$ internal nodes in the tree and $p$ weights per node, the cost is $O(pN)$. Per observation point, step 4, requires $O(Af(\theta) \log N)$ total interactions, where $f(\theta)$ is some small constant factor depending on the far-field threshold, $\theta$. As a result, the whole algorithm requires $O(N \log N)$ time.

I have ignored the analysis of the error incurred by the approximation, which can be influenced by both the nearfield threshold, $\theta$ and the order of approximation, $p$. For details, see CITEYOKOTA!

The fast multipole method is an algorithmic improvement over the treecode method. The fundamental idea in the treecode is to compute interaction between a particle and a group of particles. The fast multipole method takes this a step further and computes interactions between **pairs of groups** of far-field particles. The asymptotic cost of an evaluation becomes $O(N)$, although tree construction remains $O(N \log N)$.

## Inverse Treecode Method

Efficient techniques for the inverse n-body problem will rely on the same key idea as the fast n-body/matrix-vector product method above. That is, far-field iteractions must be approximated and compressed.

CITEAMBIKASARAN describes a method for direct inversion of the matrix produced by a fast multipole method. The method is fundamentally a Gaussian elimination process with two main caveats:

1. Sparse extension: Because all operations performed are linear (additions and multiplications), the action of the treecode method I have described (also the action of a fast multipole method) can be encoded as a large sparse matrix with additional variables corresponding to the weights, $W_m^I$. This matrix is necessarily sparse because $O(N \log N)$ operations are performed by the treecode and a non-sparse matrix would represent $\omega(N \log N)$ operations.

2. Compression of fill-in: Gaussian elimination proceeds by adding and subtract multiples of rows or columns of a matrix. This process does not maintain the sparsity pattern of the matrix. What may have been 0 to begin with could be "filled-in" due to an elimination step

## Application: Integral Equations

In my research, I solve the partial differential equations describing Earth's crustal deformation in order to estimate seismic hazard and understand the mechanics of earthquakes. Commonly used volumetric methods

like the finite element method produce large sparse matrices that must be inverted. However, representing complex geometries is difficult with volumetric methods. To avoid this problem, I translate the partial differential equations into a convolution over the boundary of the domain – a boundary integral equation (CITEPCruse1969). Because the integral equation explicitly consider the boundaries of the domain instead of the volume, treating faults and other surfaces within the Earth becomes much easier. The result, called the Somigliana identity is:

$$\vec{u}(x) + \lim_{\tilde{x}\to x} \int_{\partial\Omega} T^*(\tilde{x}, y)\vec{u}(y)dS = \lim_{\tilde{x}\to x} \int_{\partial\Omega} U^*(\tilde{x}, y)\vec{t}(y)dS \quad \forall x \in \partial\Omega \tag{9}$$

where $\Omega$ is the region of interest, $\partial\Omega$ is its boundary, $U^*(x, y) = O(\frac{1}{\|x-y\|})$ is an operator that represents the displacement at $x$ given a force at $y$, $T^*(x, y) = O(\frac{1}{\|x-y\|^2})$ is an operator that represents the displacement at $x$ given a displacement at $y$, $t(x)$ are the forces and $u(x)$ are the displacements in the region. Given knowledge of either $u(x)$ or $t(x)$, the integral equation can be solved for the other. For simple problems, this integral equation could be solved by hand. However, for complex geometries, this continuous problem needs to be discretized to be solved as a linear algebra problem on a computer.

To discretize the problem, I first represent the surface, $\partial\Omega$, via a triangular mesh $\partial\Omega \approx \cup T_i$. Then, I represent both $u(x)$ and $t(x)$ as a piecewise-polynomial interpolation over this mesh. I described this interpolation in terms of a basis $\phi_i(x)$, where each basis function has a support over only one triangle. The coefficients of these polynomials $u_i$ and $t_i$ will be the unknowns in the final linear system. Rewriting Somigliana's identity in terms of this interpolation:

$$\sum_j u_j\phi_j(x) + \lim_{\tilde{x}\to x} \int_{\partial\Omega} T^*(\tilde{x}, y) \sum_j u_j\phi_j(y)dS = \lim_{\tilde{x}\to x} \int_{\partial\Omega} U^*(\tilde{x}, y) \sum_j t_j\phi_j(y)dS \quad \forall x \in \partial\Omega \tag{10}$$

$$\tag{11}$$

Finally, this integral equation cannot be enforced everywhere. That would require an infinite number of "constraints", each being a row in the final linear system. Instead, I enforce the integral equation in a least squares weighted average sense where each basis function of the interpolation takes a turn as the weighting function. For the Somigliana identity, I get:

$$\int_{\partial\Omega} \phi_i(x) \sum_j u_j\phi_j(x)dS_x + \int_{\partial\Omega} \phi_i(x) \lim_{\tilde{x}\to x} \int_{\partial\Omega} T^*(\tilde{x}, y) \sum_j u_j\phi_j(y)dS_y dS_x =$$
$$\int_{\partial\Omega} \phi_i(x) \lim_{\tilde{x}\to x} \int_{\partial\Omega} U^*(\tilde{x}, y) \sum_j t_j\phi_j(y)dS_y dS_x \quad \forall i \tag{12}$$

By rearranging 12, the linear system emerges from behind the curtain:

$$\sum_j u_j \int_{\partial\Omega} \phi_i(x)\phi_j(x)dS_x + \sum_j u_j \int_{\partial\Omega} \phi_i(x) \lim_{\tilde{x}\to x} \int_{\partial\Omega} T^*(\tilde{x}, y)\phi_j(y)dS_y dS_x =$$
$$\sum_j t_j \int_{\partial\Omega} \phi_i(x) \lim_{\tilde{x}\to x} \int_{\partial\Omega} U^*(\tilde{x}, y)\phi_j(y)dS_y dS_x \quad \forall i \tag{13}$$

This can be written:

$$\sum_j (M_{ij}u_j + G_{ij}u_j - H_{ij}t_j) = 0 \quad \forall i \tag{14}$$

$$M_{ij} = \int_{\partial\Omega} \phi_i(x)\phi_j(x)dS_x \tag{15}$$

$$G_{ij} = \int_{\partial\Omega} \phi_i(x) \lim_{\tilde{x}\to x} \int_{\partial\Omega} T^*(\tilde{x}, y)\phi_j(y)dS_y dS_x \tag{16}$$

$$H_{ij} = \int_{\partial\Omega} \phi_i(x) \lim_{\tilde{x}\to x} \int_{\partial\Omega} U^*(\tilde{x}, y)\phi_j(y)dS_y dS_x \tag{17}$$

For $n$ basis functions, this is a linear system of $n$ equations and $2n$ unknowns. So, for every triangle on the surface, either displacement or traction must be known, while the other will be solved for. Supposing that $u_j$ is known and $t_j$ is unknown, the problem can be written in the standard form for an n-body problem with $K_{ij} = H_{ij}$ and $f_i = \sum_j (M_{ij} + G_{ij})u_j$. Actually computing the matrix entries is a complex task that I won't cover here, but fundamentally relies on quadrature methods like Gaussian quadrature.

TODO: discuss actual scientific applications?

# Conclusion

I've discussed fast methods for the matrix-vector products and matrix inversions that arise in n-body and inverse n-body problems. I presented a simplification of a recent fast inversion method, the inverse fast multipole method, that achieves close to the same time bound. This may be useful for pedagogical purposes and as an implementational stepping stone to more complex, but asymptotically faster methods. Despite a $O(N \log N)$ complexity, because of their comparative simplicity, treecodes have been found to be faster than $O(N)$ fast multipole methods for reasonable $N$ in certain domains. These solutions to generalized n-body problems are widely applicable, especially in numerical simulation of physical processes. I am actively implementing this method. The current state of the code is available at `https://github.com/tbenthompson/direct_inv`.