Recursive Tree Algorithms for Orthogonal Matrix Generation and Matrix-Vector Multiplications in Rigid Body Simulations

Fuhui Fang *

April 5, 2016

Abstract

In this paper, we study a numerical linear algebra problem arising from the efficient simulation of Brownian dynamics with hydrodynamics interactions where molecules are modeled as ensembles of rigid bodies. Given the first 6 rows of a matrix Q of size $3n \times 3n$ describing how the force on each of the n particles in a rigid body P can be mapped to the 6 entries in P's *resultant force* and *torque*, we show how the remaining $3n - 6$ rows of vectors can be constructed explicitly using $O(n \log(n))$ operations and storage, so that (1) they form an orthonormal basis and (2) they are orthogonal to each of the first 6 vectors. For applications where only the matrix-vector multiplications are needed, without forming Q, we introduce $O(n)$ recursive tree algorithms for computing both $Q \cdot \vec{v}$ and $Q^T \cdot \vec{v}$ for an arbitrary vector $\vec{v}$. Preliminary numerical results are presented to demonstrate the performance and accuracy of the numerical algorithms.

1 Problem Statement

Assume a rigid body P consists of n unit-mass particles P_j, $j = 1, \cdots, n$ and P's mass center is located at the origin $\vec{O} = (0, 0, 0)^T$. Further assume an external force $\vec{f_j} = (f_j^1, f_j^2, f_j^3)^T$ is applied on particle P_j located at $\vec{r_j} = (x_j, y_j, z_j)^T$, then the

*This project was supported by the Michael P. and Jean W. Carter Research Fund administered by Honors Carolina.

resultant force $\vec{f}_P$ and *torque* $\vec{\tau}_P$ of rigid body P are given by

$$\vec{f}_P = \sum_{j=1}^{n} \vec{f}_j, \quad \vec{\tau}_P = \sum_{j=1}^{n} \vec{r}_j \times \vec{f}_j, \tag{1.1}$$

and the corresponding matrix form is given by

$$\begin{bmatrix} \vec{f}_P \\ \vec{\tau}_P \end{bmatrix}_{6\times 1} = \begin{bmatrix} I & I & \dots & I \\ A_1 & A_2 & \dots & A_n \end{bmatrix}_{6\times 3n} \begin{bmatrix} \vec{f}_1 \\ \vec{f}_2 \\ \cdots \\ \vec{f}_n \end{bmatrix}_{3n\times 1} = Z \begin{bmatrix} \vec{f}_1 \\ \vec{f}_2 \\ \cdots \\ \vec{f}_n \end{bmatrix}, \tag{1.2}$$

where the $6 \times 3n$ matrix Z is defined as

$$Z = \begin{bmatrix} \vec{z}_1^T \\ \cdots \\ \vec{z}_6^T \end{bmatrix} = \begin{bmatrix} I & I & \dots & I \\ A_1 & A_2 & \dots & A_n \end{bmatrix} \tag{1.3}$$

which will be referred to as the *Z-matrix* of P, I is the 3×3 Identity matrix, and A_i represents the translation matrix of $A_i \cdot \vec{f}_i \triangleq r_i \times \vec{f}_i$ which can be explicitly written as

$$A_i = \begin{bmatrix} 0 & -z_i & y_i \\ z_i & 0 & -x_i \\ -y_i & x_i & 0 \end{bmatrix}.$$

Note that when the Gram-Schmidt process is applied on the 6 row-vectors of Z, an orthonormal basis $\{\vec{q}_1, \vec{q}_2, \cdots, \vec{q}_k\}$ can be derived using an upper triangular matrix R of size $6 \times k$ as in

$$Q_Z \triangleq [\vec{q}_1, \vec{q}_2, \cdots \vec{q}_k]^T = (R^T)^{-1} \cdot Z.$$

In this paper, we introduce three recursive tree algorithms to address the following three questions:

Q1: Given the $6 \times 3n$ *Z-matrix* Z (or equivalently $Q_Z = [\vec{q}_1, \vec{q}_2, \cdots \vec{q}_k]^T$) of a rigid body P consisting of n particles, how to **efficiently** construct the remaining $3n - k$ orthonormal vectors $\tilde{Q} = [\vec{q}_{k+1}, \vec{q}_{k+2}, \cdots, \vec{q}_{3n}]^T$, such that the matrix $Q_{3n\times 3n} = \begin{bmatrix} Q_Z \\ \tilde{Q} \end{bmatrix}$ is an orthogonal matrix?

Q2: For any given set of forces applied on the n particles, how to **efficiently** compute the matrix-vector multiplication

$$Q_{3n\times 3n} \begin{bmatrix} \vec{f}_1 \\ \cdots \\ \vec{f}_n \end{bmatrix}_{3n\times 1} \quad ? \tag{1.4}$$

Q3: For any given vector $\vec{v}_{3n\times 1}$, how to **efficiently** compute $Q^{-1}\vec{v} = Q^T\vec{v}$?

2 Background

The three questions arise in our research efforts to design optimal numerical tools for simulating the Brownian dynamics of biomolecules with hydrodynamics interactions, where a complex molecular system is modeled by multiple (e.g., hundreds or thousands) rigid bodies to reduce the numerical "stiffness" due to the local chemical bond type interactions between atoms that cause very high frequency oscillations (which require extremely small time step sizes when marching in time). Instead of a thorough listing of existing literature on the Brownian dynamics models and hydrodynamics interactions, we focus on the "bead model" in [9] in which a molecular system is modeled by m rigid bodies with given external forces $\{\vec{f}_j\}$ and torques $\{\vec{\tau}_j\}$, $j = 1, \cdots, m$, in each step of the dynamics simulation. To simplify the discussions and notations, we further assume that the *Z-matrix* of each rigid body has full rank 6 (lower rank cases can be handled similarly, but notations will be more involved). Under this assumption, the unknowns are then the size 3 translational velocity vectors $\{\vec{v}_j\}$ and angular velocity vectors $\{\vec{\omega}_j\}$, $j = 1, \cdots, m$, which determine how each rigid body moves. There are therefore a total of $6m$ unknowns, the same as the number of entries in the given forces and torques.

To simulate the hydrodynamics interactions of the molecular system, each rigid body is further modeled by a number of "beads" (usually of the same radius) placed on its surface. This can be roughly considered as a particle-type method for solving the boundary value Stokes equations. When n_j beads are used for rigid body j, the total number of beads is then $n = \sum_{j=1}^{m} n_j$. The bead model assumes that the hydrodynamics interactions satisfy $D\vec{f} = \vec{v}$, where $D_{3n\times 3n}$ is the Rotne-Prager-Yamakawa tensor whose entries are determined by the bead locations (see, e.g., [1, 2]), $\vec{f}$ is a vector of size $3n$ consisting of n force vectors, each of which is of size 3 representing the force on each bead, and $\vec{v}$ is a vector of size $3n$ consisting of the three velocity components of each bead. Adding the rigid body constraints, the bead model can then be described by

$$Z_{6m\times 3n} D^{-1}_{3n\times 3n} Z^T_{3n\times 6m} \begin{bmatrix} \begin{bmatrix} \vec{v}_1 \\ \vec{\omega}_1 \end{bmatrix} \\ \vdots \\ \begin{bmatrix} \vec{v}_m \\ \vec{\omega}_m \end{bmatrix} \end{bmatrix}_{6m\times 1} = \begin{bmatrix} \begin{bmatrix} \vec{f}_1 \\ \vec{\tau}_1 \end{bmatrix} \\ \vdots \\ \begin{bmatrix} \vec{f}_m \\ \vec{\tau}_m \end{bmatrix} \end{bmatrix}_{6m\times 1} \tag{2.1}$$

In the formula, $Z_{6m\times 3n}$ is a block diagonal matrix and each diagonal block is the $6 \times 3n_j$ *Z-matrix* of rigid body j in Eq. (1.3).

The physical meaning of Eq. (2.1) is that for the unknown velocity vector $\vec{v}_{r\_body} \triangleq$

$\left[\vec{v}_1^T, \vec{\omega}_1^T, \cdots, \vec{v}_m^T, \vec{\omega}_m^T\right]^T$ of the rigid bodies, $\vec{v}_{bead} \triangleq Z_{3n\times 6m}^T \vec{v}_{r\_body}$ gives the velocity vector of all the beads, $\vec{f}_{bead} \triangleq D_{3n\times 3n}^{-1}\vec{v}_{bead}$ is then the force-and-torque vector of all the beads due to the hydrodynamics interactions modeled by the Rotne-Prager-Yamakawa tensor. When the resultant force and torque vector of the rigid bodies is derived using the mapping $\vec{f}_{r\_body} \triangleq Z_{6m\times 3n}\vec{f}_{bead}$, it has to match the given external force and torque vector as in $\vec{f}_{r\_body} = \vec{f}_{ext} \triangleq \left[\vec{f}_1^T, \vec{\tau}_1^T, \cdots, \vec{f}_m^T, \vec{\tau}_m^T\right]^T$. We want to mention that instead of using the *Z-matrix* for each rigid body in Eq. (2.1), one can also consider the diagonalized formulation

$$Q_{Z\,6m\times 3n} D_{3n\times 3n}^{-1} Q_{Z\,3n\times 6m}^T \tilde{v}_{6m\times 1} = \tilde{f}_{6m\times 1}, \tag{2.2}$$

where each diagonal block Q_{Z_j} in Q_Z contains the orthonormal basis vectors of Z_j, and $\tilde{v}$ and $\tilde{f}$ are the corresponding vectors after performing the same orthogonalization procedure on the original $\vec{f}_{ext}$ and $\vec{v}_{r\_body}$ vectors. One numerical difficulty when solving Eq. (2.1) or Eq. (2.2) is the calculation of D^{-1}: as D is a dense matrix, for large n, even with the acceleration of the fast direct solvers [3, 6] or H-matrix based techniques [4, 5], computing D^{-1} is simply too expensive in the dynamics simulations as the solution of the bead model is required at each and every time marching step.

To overcome this hurdle, in the following, we show how to reformulate Eq. (2.2) to a different linear system. The technique for Eq. (2.1) is very similar and we neglect the details in this paper. The first step is to form the orthogonal matrix $\left[\begin{array}{cc} Q_Z^T & \tilde{Q}^T \end{array}\right]_{3n\times 3n}$, and add $3n-6m$ zeros to $\tilde{v}_{6m\times 1}$, to form

$$Q_{Z\,6m\times 3n} D_{3n\times 3n}^{-1} \left[\begin{array}{cc} Q_Z^T & \tilde{Q}^T \end{array}\right]_{3n\times 3n} \left[\begin{array}{c} \tilde{v} \\ \vec{0} \end{array}\right]_{3n\times 1} = \tilde{f}_{6m\times 1}.$$

Next, we add $3n-6m$ new equations to the original system to form

$$\left[\begin{array}{c} Q_Z \\ \tilde{Q} \end{array}\right]_{3n\times 3n} D_{3n\times 3n}^{-1} \left[\begin{array}{cc} Q_Z^T & \tilde{Q}^T \end{array}\right]_{3n\times 3n} \left[\begin{array}{c} \tilde{v} \\ 0 \end{array}\right]_{3n\times 1} = \left[\begin{array}{c} \tilde{f} \\ \vec{g} \end{array}\right]_{3n\times 1}, \tag{2.3}$$

where the vector $\vec{g}_{(3n-6m)\times 1}$ contains $3n-6m$ *unknown* numbers. We multiply both sides by $\left[\begin{array}{cc} Q_Z^T & \tilde{Q}^T \end{array}\right]$ to get

$$D^{-1} \left[\begin{array}{cc} Q_Z^T & \tilde{Q}^T \end{array}\right] \left[\begin{array}{c} \tilde{v} \\ 0 \end{array}\right] = \left[\begin{array}{cc} Q_Z^T & \tilde{Q}^T \end{array}\right] \left[\begin{array}{c} \tilde{f} \\ \vec{g} \end{array}\right] = \left[Q_Z^T \tilde{f} + \tilde{Q}^T \vec{g}\right], \tag{2.4}$$

where we used the fact that $\left[\begin{array}{cc} Q_Z^T & \tilde{Q}^T \end{array}\right]$ is an orthogonal matrix and its inverse is simply its transpose. Then, we multiply both sides of the equation by D to get

$$\left[\begin{array}{cc} Q_Z^T & \tilde{Q}^T \end{array}\right] \left[\begin{array}{c} \tilde{v} \\ \vec{0} \end{array}\right] = D\left[Q^T \tilde{f} + \tilde{Q}^T \vec{g}\right].$$

Finally we multiply the equation by $\left[\begin{array}{cc} Q_Z^T & \tilde{Q}^T \end{array}\right]^T$ to get

$$\left[\begin{array}{c} \tilde{v} \\ \vec{0} \end{array}\right] = \left[\begin{array}{c} Q_Z \\ \tilde{Q} \end{array}\right] D \left[Q_Z^T \tilde{f} + \tilde{Q}^T \vec{g}\right] = \left[\begin{array}{c} QD\left(Q_Z^T \tilde{f} + \tilde{Q}^T \vec{g}\right) \\ \tilde{Q} D\left(Q_Z^T \tilde{f} + \tilde{Q}^T \vec{g}\right) \end{array}\right]. \tag{2.5}$$

In the new formulation, for the given vector $\tilde{f}_{6m\times 1}$, one first finds the unknown vector $\vec{g}$ by solving

$$\vec{0} = \tilde{Q} D Q_Z^T \tilde{f} + \tilde{Q} D \tilde{Q}^T \vec{g} \tag{2.6}$$

using a preconditioned Krylov subspace iterative method, and then computes the velocity vector $\tilde{v}_{6m\times 1}$ by evaluating

$$\tilde{v} = Q_Z D(Q^T \tilde{f} + \tilde{Q}^T \vec{g}). \tag{2.7}$$

Note that the fundamental building blocks required when applying a Krylov subspace method for the new formulation are the fast matrix-vector multiplication algorithms for computing (a) $D \cdot \vec{v}$, (b) $\left[\begin{array}{cc} Q_Z^T & \tilde{Q}^T \end{array}\right] \cdot \vec{v}$, and (c) $\left[\begin{array}{cc} Q_Z^T & \tilde{Q}^T \end{array}\right]^T \cdot \vec{v}$ for any given vector $\vec{v}$. For (a), one of the authors and collaborators has developed fast $O(n)$ algorithms using the fast multipole method (FMM) [7, 8]. In this paper, we introduce $O(n)$ recursive tree algorithms for (b) and (c). We want to mention that other reformulations for Eq. (2.2) or Eq. (2.1) are also possible, including techniques based on the Schur complement. These alternatives are also being studied and compared with the approach in this paper. Another closely related research topic is the design of effective preconditioners for Eq. (2.6). Results along these directions will be discussed in future papers.

3 Divide-and-Conquer and Recursive Tree Algorithms

The divide-and-conquer technique on the tree structure is the basis of many efficient algorithms for different types of applications. Examples include the fast Fourier transform (FFT) (parent polynomial is divided into child polynomials recursively until the degree of the child polynomial is low enough for direct computation, and results from child polynomials are combined to generate parent's result), multigrid methods (upward and downward passes are performed on the tree structure with information exchange between child and parent levels, to effectively reduce the errors in the solution), and fast multipole methods (FMM) (data are compressed and transmitted between parent and child boxes in an adaptive spatial tree structure). Assuming the

data are approximately uniformly distributed, the adaptive tree usually has $O(\log(n))$ levels, making it efficient to process and transmit any data on the tree structure with better numerical stability features. In this section, we discuss how to use the divide-and-conquer technique on the tree structures to design optimal algorithms for the 3 questions in Sec. 1.

3.1 Adaptive Tree Structure

We first discuss how to generate a spatial adaptive tree when simulating a molecular system modeled by multiple rigid bodies in the bead model. We assume each rigid body is "discretized" into a number of beads as discussed in Sec. 2 to capture the hydrodynamics interactions between rigid bodies. A hierarchical partition is then performed to divide the beads domain into nested cubical boxes, where the root box is the smallest bounding box that contains all the beads. Without loss of generality, the root box is normalized to size 1 in each side. The root box is partitioned along each dimension in the middle, resulting in 8 child boxes in 3 dimensions. A child box becomes a leaf node in the tree structure if it contains no more than s beads. To simplify the discussions, we set $s = 1$ in this paper. In our implementation, other s values are also allowed after changing how leaf nodes are processed in the code. If a child box contains more than s beads, it will be further partitioned and this procedure continues recursively. If a child box is empty (with no beads inside), it will be pruned off from the tree.

This hierarchical spatial partition procedure results in an adaptive tree structure with L levels. At level $l = 0$, the root node is the bounding box of size 1, and we refer to the non-empty boxes of size $1/2^l$ as the nodes at level l of the recursive partition procedure. Each parent node is connected to each of its children and this parent-child relation can be represented by an "edge" in the tree structure. Note that each node may contain either $\leq s$ beads, or a rigid body consisting of $> s$ beads. Also, when the beads are approximately uniformly distributed, the number of levels is asymptotically $L = O(\log(n))$, and the number of nodes is $O(n)$. These estimates are not correct for very special bead distributions which are unlikely to happen in real applications.

We want to mention that the octree in 3D can be modified to a binary tree, for example, by first combine every 2 of the 8 child nodes of a parent P to form 4 "ghost" nodes, and then combine every 2 of the 4 ghost nodes to form 2 child nodes of P. It is easy to see that the number of levels in the modified binary tree is no more than $3 \times L$. In this paper, we focus on the binary tree, i.e., every parent node has at most two child nodes, to simplify the notations and description of the algorithm.

3.2 Q1: Explicit Orthogonal Matrix Generation

Without loss of generality, we consider a single (instead of m) rigid body consisting of n beads and discuss Q1: *Given the rank* $= 6$ *and size* $= 6 \times 3n$ *Z-matrix* Z *(or* $Q_Z = [\vec{q}_1, \vec{q}_2, \cdots \vec{q}_6]^T$*), how to* **efficiently** *construct the remaining* $3n-6$ *orthonormal vectors* $\tilde{Q} = [\vec{q}_7, \vec{q}_8, \cdots, \vec{q}_{3n}]^T$*, such that* $Q = \left[Q_Z^T, \tilde{Q}^T\right]^T$ *form an orthogonal matrix?*

Note that there are multiple choices of $\tilde{Q}$. For efficiency and stability considerations, we study how to utilize the divide-and-conquer strategy on the adaptive tree and generate a special structured $\tilde{Q}$. We start from a 2-level setting where the parent rigid body P at level 0 is partitioned into two child nodes at level 1, child X with n_X beads and child Y containing n_Y beads ($n_X + n_Y = n$), and

$$Z_X = \begin{bmatrix} I & I & \dots & I \\ A_1 & A_2 & \dots & A_{n_X} \end{bmatrix}_{6\times 3n_X}, \quad \textit{and} \quad Z_Y = \begin{bmatrix} I & I & \dots & I \\ B_1 & B_2 & \dots & B_{n_Y} \end{bmatrix}_{6\times 3n_Y}$$

are the *Z-matrices* of X and Y, respectively. We assume both Z_X and Z_Y have full rank (=6) to simplify the discussion and notation, and the orthogonal sub-matrices $\tilde{Q}_X$ (size $(3n_X - 6) \times 3n_X$) and $\tilde{Q}_Y$ (size $(3n_Y - 6) \times 3n_Y$) satisfying $\tilde{Q}_X \perp Z_X$ and $\tilde{Q}_Y \perp Z_Y$ are already constructed for child X and Y, respectively. The key question in a divide-and-conquer algorithm then becomes **"how to find parent** P**'s** $\tilde{Q}$ **matrix (size** $(3n-6) \times 3n$**) such that** $\tilde{Q} \perp Z_P$**, where** $Z_P = [Z_X, Z_Y]$ **is** P**'s** *Z-matrix* **"?**

One way to answer the question is to study the matrix

$$H = \begin{bmatrix} Z_X & Z_Y \\ 0_{6\times 3n_X} & Z_Y \\ \tilde{Q}_X & 0_{(3n_X-6)\times 3n_Y} \\ 0_{(3n_Y-6)\times 3n_X} & \tilde{Q}_Y \end{bmatrix}_{3n\times 3n}. \tag{3.1}$$

It is straightforward to verify that the matrix is full rank, and the vectors in the lower $3n - 12$ rows

$$\begin{bmatrix} \tilde{Q}_X & 0_{(3n_X-6)\times 3n_Y} \\ 0_{(3n_Y-6)\times 3n_X} & \tilde{Q}_Y \end{bmatrix}_{(3n-12)\times 3n}$$

of H are normalized, orthogonal to each other, and orthogonal to the vectors in the first 12 rows of H. This means that parent P's $\tilde{Q}$ can readily "receive" the lower $3n - 12$ rows of vectors from its two children. For the remaining 6 row vectors in $\tilde{Q}$ (referred to as the *Residue Vectors* in the remainder of this paper), a Gram-Schmidt procedure on the first 12 row vectors can be performed and the last 6 orthonormal vectors will be orthogonal to the vectors in Z and the $3n - 12$ vectors from the children. The Gram-Schmidt procedure requires approximately $O(n)$ operations and $O(n)$ storage for explicitly generating all the *Residue Vectors* .

We want to mention that when either Z_X or Z_Y is not full rank (=6), for example, when each child only contains 1 bead (referred to as the bead-bead case), we have $k = rank(Z) = 5$ and $rank(Z_X) = rank(Z_Y) = 3$, and the dimension of the *Residue Vectors* is then $3 + 3 - 5 = 1$; when child X has only one bead ($rank(Z_X) = 3$) and child Y contains a rigid body with $l = rank(Z_Y)$ being either $l = 5$ or $l = 6$, assuming the parent's *Z-matrix* has rank k ($k = 5$ or $k = 6$), then the *Residue Vectors* has dimension $l + 3 - k$. The computations in these special cases are briefly discussed next.

Bead+Bead: When a parent has two child nodes, each contains only one bead, then the rank of the *Z-matrix* of each node is 3, the parent's *Z-matrix* has rank 5, and the dimension of the *Residue Vectors* is $3 + 3 - 5 = 1$. The vector can be explicitly computed by normalizing the vector

$$\left[\frac{x_1 - x_2}{z_2 - z_1}, \frac{y_1 - y_2,}{z_2 - z_1}, -1, \frac{-(x_1 - x_2)}{z_2 - z_1}, \frac{-(y_1 - y_2)}{z_2 - z_1}, 1\right],$$

where the vectors (x_i, y_i, z_i) $(i = 1, 2)$ are the bead locations.

Bead+Rigid Body: Consider a parent node with two children. Child X has more than 1 bead and child Y contains only a single bead. When computing parent's *Residue Vectors* , instead of the $12 \times 3n$ matrix $\begin{bmatrix} Z_X & Z_Y \\ 0 & Z_Y \end{bmatrix}$, one can also applying the Gram-Schmidt procedure on the $9 \times 3n$ matrix $\begin{bmatrix} Z_X & Z_Y \\ 0 & I \end{bmatrix}$, where I is the 3×3 Identity matrix.

In the pseudo-code presented in Algorithm 1, we describe a recursive implementation of the divide-and-conquer technique using the function *Q_gen* to explicitly generate the orthogonal matrix Q by an upward pass in the tree structure. To estimate the algorithm complexity and storage requirement, we consider a system with n beads and a tree with $O(\log(n))$ levels. To generate the *Residue Vectors* for all the nodes in one level, by taking advantage of the *sparse structures* (many zeros in the vector which are unnecessary to calculate or store), approximately $O(n)$ operations (to compute all the required inner products and corresponding *Residue Vectors* for all nodes in this level) and $O(n)$ storage are required. As there are $O(\log(n))$ levels, the algorithm therefore has $O(n \log(n))$ complexity and $O(n \log(n))$ memory requirement.

3.3 Upward pass: Algorithm for Computing $Q \cdot \vec{v}$ in Q2

In the Brownian dynamics application in Sec. 2, instead of explicitly generating Q and Q^T, one only needs the final matrix-vector multiplication results. In this subsection, we discuss how to apply the divide-and-conquer technique to further reduce the

Algorithm 1 Upward Pass to Explicitly Generate Q

```
1: function Q_GEN(node P)
2:     if node P is childless then
3:         form P's Z-matrix Z_P.
4:     else
5:         find child nodes X and Y of node P.
6:         Q_GEN(node X)
7:         Q_GEN(node Y)
8:         form P's Z-matrix using child nodes' Z-matrices Z_X and Z_Y.
9:         perform Gram-Schmidt procedure on first 12 rows of H.
10:        if node P is root node then
11:            output all 12 orthogonal vectors.
12:        else
13:            output the last 6 (or less) Residue Vectors .
```

complexity and storage to the asymptotically optimal $O(n)$ when explicit Q is not required. We first introduce the following definitions and theorems.

Definition 1 *For a node P containing n beads in the tree structure, its Info-set is defined as the size 6 vector*

$$\mathbf{M}_P = \begin{bmatrix} I & I & \dots & I \\ A_1 & A_2 & \dots & A_n \end{bmatrix} \begin{bmatrix} \vec{f}_1 \\ \vdots \\ \vec{f}_n \end{bmatrix} = Z \begin{bmatrix} \vec{f}_1 \\ \vdots \\ \vec{f}_n \end{bmatrix}, \tag{3.2}$$

where $\vec{f}_j = \left[f_j^1, f_j^2, f_j^3\right]^T$ is the force acting on bead j and Z is the node's Z-matrix .

Similar to the multipole expansions in the fast multipole method (FMM), the size 6 *Info-set* from child nodes can be combined to form parent's *Info-set* , as described by the following theorem.

Theorem 2 *Assume a parent node P contains 2 child nodes, Child X with n_X beads and Child Y with n_Y beads, respectively. Also assume X and Y's Info-set are given by*

$$\mathbf{M}_X = \begin{bmatrix} I & \cdots & I \\ A_1 & \cdots & A_{n_X} \end{bmatrix} \begin{bmatrix} \vec{f}_1 \\ \vdots \\ \vec{f}_{n_X} \end{bmatrix} = Z_X \begin{bmatrix} \vec{f}_1 \\ \vdots \\ \vec{f}_{n_X} \end{bmatrix},$$

$$\mathbf{M}_Y = \begin{bmatrix} I & \cdots & I \\ B_1 & \cdots & B_{n_Y} \end{bmatrix} \begin{bmatrix} \tilde{f}_1 \\ \vdots \\ \tilde{f}_{n_Y} \end{bmatrix} = Z_Y \begin{bmatrix} \tilde{f}_1 \\ \vdots \\ \tilde{f}_{n_Y} \end{bmatrix},$$

Then the parent P's Info-set defined as

$$\mathbf{M}_P = \begin{bmatrix} Z_X & Z_Y \end{bmatrix} \begin{bmatrix} \vec{f}_X \\ \tilde{f}_Y \end{bmatrix} = \begin{bmatrix} I & \cdots & I & I & \cdots & I \\ A_1 & \cdots & A_{n_X} & B_1 & \cdots & B_{n_Y} \end{bmatrix} \begin{bmatrix} \vec{f}_1 \\ \vdots \\ \tilde{f}_{n_Y} \end{bmatrix} \tag{3.3}$$

can be computed from its two children as

$$\mathbf{M}_P = \mathbf{M}_X + \mathbf{M}_Y. \tag{3.4}$$

Definition 3 *For a node P containing n beads in the tree structure, its Inertia Matrix N_P is defined as the 6×6 matrix*

$$\mathbf{N}_P = \begin{bmatrix} I & \cdots & I \\ A_1 & \cdots & A_n \end{bmatrix} \begin{bmatrix} I & A_1^T \\ \vdots & \vdots \\ I & A_n^T \end{bmatrix} = Z_P \cdot Z_P^T. \tag{3.5}$$

We have the following theorem for constructing the *Inertia Matrix* .

Theorem 4 *Assume a parent node P contains 2 child nodes, Child X with n_X beads and Child Y with n_Y beads, respectively. Also assume child X's Inertia Matrix is*

$$\mathbf{N}_X = \begin{bmatrix} I & \cdots & I \\ A_1 & \cdots & A_{n_X} \end{bmatrix} \begin{bmatrix} I & A_1^T \\ \vdots & \vdots \\ I & A_{n_X}^T \end{bmatrix} = Z_X \cdot Z_X^T,$$

and Child B's Inertia Matrix is

$$\mathbf{N}_Y = \begin{bmatrix} I & \cdots & I \\ B_1 & \cdots & B_{n_Y} \end{bmatrix} \begin{bmatrix} I & B_1^T \\ \vdots & \vdots \\ I & B_{n_Y}^T \end{bmatrix} = Z_Y \cdot Z_Y^T.$$

Then parent P's Inertia Matrix defined as

$$\mathbf{N}_P = \begin{bmatrix} I & \cdots & I & I & \cdots & I \\ A_1 & \cdots & A_{n_X} & B_1 & \cdots & B_{n_Y} \end{bmatrix} \begin{bmatrix} I & A_1^T \\ \vdots & \vdots \\ I & A_{n_X}^T \\ I & B_1^T \\ \vdots & \vdots \\ I & B_{n_Y}^T \end{bmatrix} = Z_P \cdot Z_P^T \tag{3.6}$$

can be computed by

$$\mathbf{N}_P = \mathbf{N}_X + \mathbf{N}_Y.$$

The *Inertia Matrix* contains all the necessary information for generating the *Residue Vectors* from the *Z-matrix* . As discussed in previous subsection, for a parent node P, its $\tilde{Q}$ consists of two parts: the $3n-12$ orthonormal vectors from child nodes X and Y, and the *Residue Vectors* derived from the Gram-Schmidt procedure on the 12 row vectors in H as in

$$\begin{aligned} H_{1:12} &= \left[\begin{array}{ccc:ccc} I & \dots & I & I & \dots & I \\ A_1 & \dots & A_{n_X} & B_1 & \dots & B_{n_Y} \\ \hdashline & & & I & \dots & I \\ & 0 & & B_1 & \dots & B_{n_Y} \end{array}\right]_{12\times 3n} = \left[\begin{array}{cc} Z_X & Z_Y \\ 0 & Z_Y \end{array}\right] \\ &= \left[\begin{array}{c:c} R_{11}^T & 0 \\ R_{21}^T & R_{22}^T \end{array}\right]_{12\times(k+l)} \left[\begin{array}{c} Q_Z \\ \hdashline \tilde{Q}_0 \end{array}\right]_{(k+l)\times 3n} \\ &= R^T \cdot Q, \end{aligned} \tag{3.7}$$

where Z_X and Z_Y are the *Z-matrices* of child nodes X and Y, respectively, the upper triangular matrix R is partitioned into four blocks, R_{11} is of size $6 \times k$ where k is the dimension of P's *Z-matrix* , R_{22} is of size $6 \times l$ where l is the dimension of the subspace formed by the *Residue Vectors* , Q is partitioned into two row blocks, Q_Z corresponds to the first k orthogonal vectors such that row vectors in Q_Z span the same subspace as row vectors in parent's *Z-matrix* , and $\tilde{Q}_0$ contains the *Residue Vectors* of dimension l. For most settings, $k = l = 6$, other possible values of k and l include 3 and 5 as briefly discussed in previous subsection for special settings.

In the Gram-Schmidt procedure, if we require the diagonal entries of R_{11} and R_{22} to be positive, then Q and R will be uniquely determined. As it is unnecessary to explicitly generate the *Residue Vectors* , we define the *Residue* as a vector of size $l \leq 6$, denoted by $\vec{\epsilon}_P$, to store the product of the *Residue Vectors* with a given vector $\vec{v}$ as follows.

Definition 5 *The Residue of a parent node P is defined as the product of its Residue Vectors and a given vector $\vec{v}_{3n\times 1}$ denoted as $\vec{\epsilon}_P = \tilde{Q}_0 \cdot \vec{v} = \tilde{Q}_0 \cdot [\vec{v}_X^T, \vec{v}_Y^T]^T$, where $\vec{v}_X$ and $\vec{v}_Y$ are entries in $\vec{v}$ corresponding to children X and Y, respectively.*

Notice that for parent P,

$$\left[\begin{array}{cc} Z_X & Z_Y \\ 0 & Z_Y \end{array}\right] \cdot \vec{v} = \left[\begin{array}{c} \mathbf{M}_P \\ \mathbf{M}_Y \end{array}\right] = \left[\begin{array}{c:c} R_{11}^T & 0 \\ R_{21}^T & R_{22}^T \end{array}\right] \left[\begin{array}{c} Q_Z \cdot \vec{v} \\ \tilde{Q}_0 \cdot \vec{v} \end{array}\right], \tag{3.8}$$

where $\mathbf{M}_P = \left[\begin{array}{cc} Z_X & Z_Y \end{array}\right] \cdot \vec{v}$ and $\mathbf{M}_Y = Z_Y \cdot \vec{v}_Y$ are respectively parent P and Child Y's *Info-set* . The $12 \times (l+k)$ matrix R can be computed from the *Inertia Matrix* $\mathbf{N}$

as

$$H_{1:12} \cdot H_{1:12}^T = \begin{bmatrix} Z_X & Z_Y \\ 0 & Z_Y \end{bmatrix} \cdot \begin{bmatrix} Z_X & Z_Y \\ 0 & Z_Y \end{bmatrix}^T = \begin{bmatrix} \mathbf{N}_P & \mathbf{N}_Y \\ \mathbf{N}_Y & \mathbf{N}_Y \end{bmatrix} = R^T Q \cdot Q^T R = R^T \cdot R, \tag{3.9}$$

where $\mathbf{N}_P$ and $\mathbf{N}_Y$ are the symmetric *Inertia Matrix* of parent P and child Y, respectively, and Q and R are the QR-decomposition of the first 12 rows of H as shown in Eq. (3.7). For the most common case when $l = k = 6$, as

$$\begin{aligned} R_{11}^T \cdot Q_Z \cdot \vec{v} &= \mathbf{M}_P, \\ R_{21}^T \cdot Q_Z \cdot \vec{v} + R_{22}^T \cdot \tilde{Q}_0 \cdot \vec{v} &= \mathbf{M}_Y, \end{aligned}$$

one can work out $\vec{\epsilon}_P$ explicitly as

$$\vec{\epsilon}_P = \tilde{Q}_0 \cdot \vec{v} = (R_{22}^T)^{-1}(\mathbf{M}_Y - R_{21}^T \cdot Q_Z \vec{v}) = (R_{22}^T)^{-1}(\mathbf{M}_Y - R_{21}^T \cdot (R_{11}^T)^{-1} \cdot \mathbf{M}_P). \tag{3.10}$$

In the formula, one only needs R (computed from the *Inertia Matrix* $\mathbf{N}$) and *Info-set* $\mathbf{M}$ to get $\vec{\epsilon}_P$, the explicit form of the *Residue Vectors* is not required.

It is interesting to compare above definitions with those in the fast multipole methods (FMM). Each node's *Info-set* $\mathbf{M}$ collects and compresses information from beads, and sends to its parent. This is very similar to the "multipole coefficients" in FMM. The *Inertia Matrix* $\mathbf{N}$ stores the necessary information for finding the translation operator (R matrix) for deriving the *Residue* $\vec{\epsilon}$ from $\mathbf{M}$. Using a procedure similar to the FMM upward pass, the residues of all the nodes can be computed using the recursive function **compute_residue** in Algorithm 2. The complexity and storage of the algorithm can be estimated by checking the operations and memory requirement for each node in the tree structure. Notice that the *Info-set* is a vector of size 6, the *Inertia Matrix* is a matrix of size at most 6×6, and the size of the *Residue* is no more than 6. Also, as only the matrix-vector multiplication results are required and it is unnecessary to compute or store the *Residue Vectors* . The operation counts and storage for the algorithm is therefore proportional to the total number of nodes in the tree structure. For most bead distributions, as the number of nodes in the tree structure is proportional to the number of beads n, the algorithm complexity and storage are both $O(n)$.

3.4 Downward Pass: Algorithm for Computing $Q^T\vec{v}$ in Q3

We address **Q3:** the efficient computation of the matrix-vector multiplication $Q^T\vec{v}$ in this section. When Q^T is considered as a vector of column vectors, $Q^T\vec{v}$ is the linear combination of these column vectors where $\vec{v}$ contains the coupling coefficients.

Algorithm 2 Upward Pass to Compute $Q \cdot \vec{v}$

```
function COMPUTE_RESIDUE(node P)
    if node P is childless then
        compute Info-set M_P and Inertia Matrix N_P.
    else
        find child nodes X and Y of node P.
        COMPUTE_RESIDUE(node X)
        COMPUTE_RESIDUE(node Y)
        construct P's Info-set M_P and Inertia Matrix N_P from children's.
        compute P's Residue ε⃗ using Eq. (3.10).
```

We start from an observation from rigid body dynamics. Given the translational velocity $\vec{V}$ and angular velocity $\vec{\omega}$ of a rigid body P consisting of n beads, and assuming that the reference point is the origin $\vec{O}$, then the velocity of bead i in the rigid body can be computed by

$$\vec{v}_i = \vec{V} + \vec{\omega} \times \vec{r}_i.$$

In matrix form, the bead velocities are given by

$$\begin{bmatrix} \vec{v}_1 \\ \cdots \\ \vec{v}_n \end{bmatrix}_{3n\times 1} = \begin{bmatrix} I & A_1^T \\ \vdots & \vdots \\ I & A_n^T \end{bmatrix}_{3n\times 6} \begin{bmatrix} \vec{V} \\ \vec{\omega} \end{bmatrix}_{6\times 1} = Z^T \cdot \begin{bmatrix} \vec{V} \\ \vec{\omega} \end{bmatrix}_{6\times 1}, \tag{3.11}$$

where Z is the *Z-matrix* of P. Notice that only 6 numbers in $\vec{V}$ and $\vec{\omega}$ are needed to construct all beads' velocities, assuming the location of each bead is known. We therefore introduce the definition of *RV-set* (rigid body velocity set), a vector containing the 6 numbers.

Definition 6 *The RV-set of a rigid body P consisting of n beads is defined as a vector of size 6 given by $\mathbf{L} = \left[\vec{V}^T, \vec{\omega}^T\right]^T$ describing how the rigid body moves. The velocities of the beads in the rigid body are given in Eq. (3.11).*

In Eq. (3.11), instead of the *Z-matrix* Z, one can also change the basis and use the corresponding Q_Z containing the orthonormal vectors $\{q_1^T, q_2^T, \cdots, q_k^T\}$, where $k = rank(Z)$. Then for any given vector $\vec{v}$ of size k, a *RV-set* vector $\mathbf{L}$ of size 6 can be found (may not be unique) such that $Z^T \cdot \mathbf{L} = Q_Z^T \vec{v} = \sum_{j=0}^{k} q_j^T \cdot v_j$. This observation provides a way to store the necessary information in the linear combination of the vectors $\{q_j^T,\ j = 1, \cdots, k\}$ with coupling coefficients $\{v_j,\ j = 1, \cdots, k\}$ into the *RV-set* $\mathbf{L}$, i.e., similar to the "local expansion coefficients" in FMM, the *RV-set* can be

used to store the compressed information. Next, we show how the stored information can be passed to the child nodes in an adaptive tree.

Theorem 7 *Assume a parent rigid body P is partitioned to child X with Z-matrix Z_X and child Y with Z-matrix Z_Y, and parent P's Z-matrix is $Z_P = [Z_X, Z_Y]$. Also assume that*

$$\begin{bmatrix} Z_X^T & 0 \\ Z_Y^T & Z_Y^T \end{bmatrix} = \begin{bmatrix} Q_Z^T & \tilde{Q}_0^T \end{bmatrix} \begin{bmatrix} R_{11} & R_{21} \\ 0 & R_{22} \end{bmatrix} = \begin{bmatrix} Q_Z^T R_{11}, & Q_Z^T R_{21} + \tilde{Q}_0^T R_{22} \end{bmatrix} \quad (3.12)$$

where R_{11} and R_{22} are upper triangular matrices computed using the Inertia Matrix . Then for any vector $\vec{v} = [v_1, \cdots, v_k]^T$ of the same size as $rank(\tilde{Q}_0)$, there exist RV-set vectors $\tilde{\mathbf{L}_X}$ and $\tilde{\mathbf{L}_Y}$ for child X and Y, respectively, such that

$$\tilde{Q}_0^T \vec{v} = \begin{bmatrix} Z_X^T \cdot \tilde{\mathbf{L}_X} \\ Z_Y^T \cdot \tilde{\mathbf{L}_Y} \end{bmatrix}, \quad (3.13)$$

where

$$\begin{cases} \tilde{\mathbf{L}_X} = -R_{11}^{-1} R_{21} R_{22}^{-1} \vec{v}, \\ \tilde{\mathbf{L}_Y} = (I - R_{11}^{-1} R_{21}) R_{22}^{-1} \vec{v}, \end{cases} \quad (3.14)$$

i.e., without explicitly using $\tilde{Q}_0$, $\tilde{Q}_0^T \vec{v}$ can be compressed and stored in the RV-set of child nodes.

Theorem 7 suggests a recursive algorithm on the tree structure to compress the matrix-vector multiplication results, store in each node's *RV-set* , and send to its child nodes. At the root node, consider the QR decomposition of the *Z-matrix* $Z_{root}^T = Q_{root}^T R_{root}$ where R_{root} is computed using the *Inertia Matrix* $\mathbf{N}_{root}$, and the linear combination of vectors in $Q_{Z_{root}}$ with coupling coefficients $\vec{v}_{root}$ can be be stored as the root node's *RV-set* given by $\mathbf{L}_{root} = R_{root}^{-1} \vec{v}_{root}$. For any child node, the linear combination of the orthogonal vectors from all parent nodes can be computed using its parent's *RV-set* plus linear combination of its parent's *Residue Vectors* with given coupling coefficients, and can be stored as child's *RV-set* , as shown in the following theorem.

Theorem 8 *Consider a parent rigid body P with child nodes X and Y in the tree structure and assume P's RV-set $\mathbf{L}_P$ satisfies $Q_P^T \vec{v}_1 = Z_P^T \cdot \mathbf{L}_P$ where Q_P represents all the orthogonal vectors from P's parents and grandparents and $\vec{v}_1$ is the coupling coefficient vector. Further assume that*

$$\tilde{Q}_0^T \vec{v}_2 = \begin{bmatrix} Z_X^T \cdot \tilde{\mathbf{L}}_X \\ Z_Y^T \cdot \tilde{\mathbf{L}}_Y \end{bmatrix}$$

where $\tilde{Q}_0$ contains P's Residue Vectors , $\vec{v}_2$ is the corresponding coupling coefficient vector, and $\tilde{\mathbf{L}}_X$ and $\tilde{\mathbf{L}}_Y$ are computed using Eq. (3.14). Then for the two child nodes X and Y, the linear combination of the orthogonal vectors from Q_P and $\tilde{Q}_0$ can be stored in child nodes' RV-set $\mathbf{L}_X$ and $\mathbf{L}_Y$ as in

$$\begin{bmatrix} Q_P^T & \tilde{Q}_0^T \end{bmatrix} \cdot \begin{bmatrix} \vec{v}_1 \\ \vec{v}_2 \end{bmatrix} = Q_P^T \vec{v}_1 + \tilde{Q}_0^T \vec{v}_2 = Z_P^T \cdot \mathbf{L}_P + \begin{bmatrix} Z_X^T \cdot \tilde{\mathbf{L}}_X \\ Z_Y^T \cdot \tilde{\mathbf{L}}_Y \end{bmatrix} = \begin{bmatrix} Z_X^T \cdot \mathbf{L}_X \\ Z_Y^T \cdot \mathbf{L}_Y \end{bmatrix}, \quad (3.15)$$

where $\mathbf{L}_X = \mathbf{L}_P + \tilde{\mathbf{L}}_X$, $\mathbf{L}_Y = \mathbf{L}_P + \tilde{\mathbf{L}}_Y$.

This theorem shows how parent's information can be combined with the information from *Residue Vectors* and sent to its children. The recursive algorithm implementation is presented as a pseudo-code in Algorithm 3. In the algorithm, we assume the R matrix for each node is already computed in the upward pass discussed in Sec. 3.3.

Algorithm 3 Downward Pass to compute $Q^T \cdot \vec{v}$

```
1:  function COMBINE_ORTHO_VEC(node P)
2:      if node P is root node then
3:          compute P's RV-set using L_root = R_root^{-1} v_root.
4:      else
5:          compute P's RV-set using Theorem 8.
6:
7:      if node P is leaf node then
8:          output the linear combination result for the leaf node.
9:      else
10:         find child nodes X and Y.
11:         COMBINE_ORTHO_VEC(node X)
12:         COMPINE_ORTHO_VEC(node Y)
```

For the special cases when $rank(Z) < 6$, same as the upward pass in previous subsection, the formulas and calculations can be simplified and we neglect the details here. Also, similar to the downward pass in the fast multipole method, the *RV-set* can be considered as the "local expansion" of the algorithm, and it contains information from both parent and special features during parent-children communications. The special features are computed using the translation operator determined by the *Inertia Matrix* collected in the upward pass. Finally, as a constant amount of operations (and storage) are required for each node, the complexity of the recursive algorithm is $O(n)$.

4 Numerical Results

As a prototype, we implement the recursive algorithms with Matlab, and preliminary numerical results for the first two algorithms (explicit Q and implicit Q methods) are presented in this section. More numerical experiments will be conducted in future papers.

4.1 Performance Measurement

As in previous analysis, in a system with n beads, if we explicitly generate the orthogonal matrix Q and compute the matrix-vector multiplication $Q \cdot \vec{v}$ recursively, then the required storage is $O(n \log(n))$. With each choice of n ($n = 100, 200, \cdots, 1000$), we run the algorithm ten times and obtain the average memory s used. Then we plot the required memory against the number of beads with a line of best fit using lease squares. As shown in Figure 1, the data points perfectly fit $s = 32.1563n \log(n) - 1628.7007$, which corresponds to the $O(n \log(n))$ storage reqired.

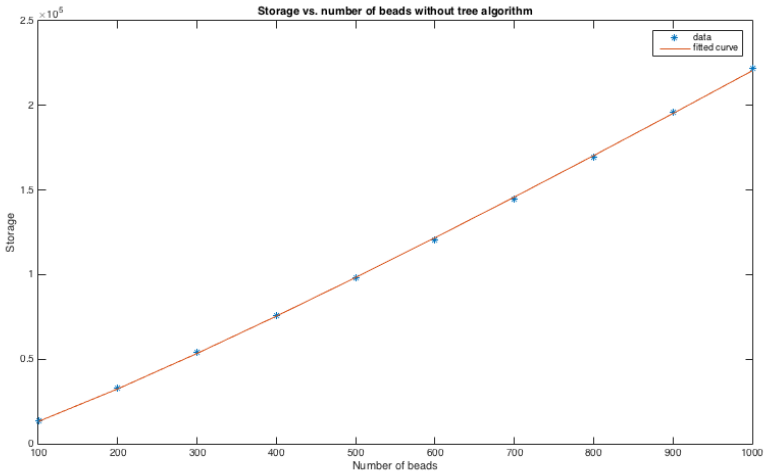

Figure 1: Explicit Q Method, fitted line: $s = 32.1563n \log(n) - 1628.7007$

To measure performance, we also compute the total CPU time (in seconds) for each run, and plot them against n. Although the time recorded is not accurate, as other systems also take up the CPU at the same time, we minimize the error by making the environment setting the same each time. The data appear to roughly fit

$t = 0.0155n\log(n) + 53.6008$ in Figure 2. This result verifies that the time complexity for the explicit Q method is $O(n\log(n))$. One problem with this figure is that the data points are located above the fitted curve when $n = 100, 200$. Future work is required to improve the implementation.

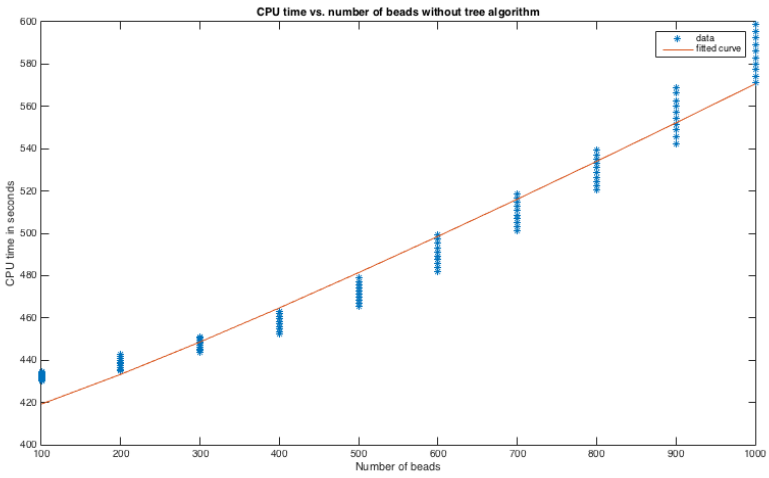

Figure 2: Explicit Q Method, fitted line: $t = 0.0155n\log(n) + 53.6008$

Similarly, we run the second algorithm (the implicit Q method) ten times with the same choices of n, and we present the data points in Figure 3 with a fitted line $t = 0.0011n + 0.1763$, which demonstrates the complexity of the implicit Q method is the asymptotically optimal $O(n)$. The data points appear to be more spread out as n increases. One explanation is that as the required storage increases, it may take more time to get the information from memory.

Table 1: Error Analysis: $Q \cdot Q^T - I$

Number of beads	100	200	300	400	500
Error	1.7e-13	6.3e-14	7.3e-13	5.5e-13	6.2e-13
Number of beads	600	700	800	900	1000
Error	1.0e-13	1.0e-13	2.4e-13	4.8e-13	4.8e-13

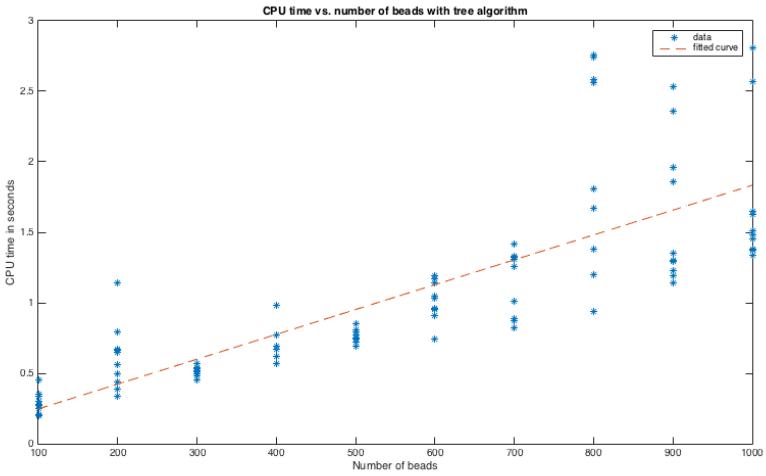

Figure 3: Implicit Q Method, fitted line: $t = 0.0011n + 0.1763$

4.2 Accuracy Measurement

We want to demonstrate that the Q generated explicitly is indeed orthogonal. In Table 1, we show that $Q \cdot Q^T I$ is close to machine precision, where I is the $3n \times 3n$ identity matrix. Then we present the accuracy comparison from the two algorithms in Table 2 that shows that our algorithms guarantee the results are the same with a precision of at least 10 decimal digits. This number is less than machine precision, probably resulting from the QR-decomposition. If we create a matrix A by $A = Q \cdot R$, where Q is an orthogonal matrix and R is an upper triangular matrix, and then we perform QR-decomposition on A to form $\tilde{Q}$ and $\tilde{R}$, then the norm of $(Q - \tilde{Q})$ and the norm of $(R - \tilde{R})$ may be zero with a precision of only 10 decimal digits. In other words, the Q we generate explicitly and the Q used in the direct calculation of $Q \cdot \vec{v}$ are different, although both of them are orthogonal and satisfy the equation of the QR-decomosition. Finally, we verify that $Q^T \cdot Qv = v$ and present the results in Table 3. $Q \cdot v$ is obtained implicitly and Q^T is the transpose of the Q we generate explicitly.

Table 2: Error Analysis: Explicit Q Method vs. Implicit Q Method

Number of beads	100	200	300	400	500
Error	5.5e-12	1.2e-11	2.8e-11	7.5e-11	9.5e-11
Number of beads	600	700	800	900	1000
Error	2.5e-10	4.7e-10	1.0e-10	9.5e-11	2.2e-10

Table 3: Error Analysis: $Q^T \cdot Qv - v$

Number of beads	100	200	300	400	500
Error	9.5e-13	2.4e-12	1.2e-11	4.2e-11	1.9e-11
Number of beads	600	700	800	900	1000
Error	1.4e-10	3.4e-11	3.2e-11	5.9e-11	2.4e-11

References

[1] GK Batchelor. Brownian diffusion of particles with hydrodynamic interaction. *Journal of Fluid Mechanics*, 74(01):1–29, 1976.

[2] Donald L Ermak and JA McCammon. Brownian dynamics with hydrodynamic interactions. *The Journal of chemical physics*, 69(4):1352–1360, 1978.

[3] Leslie Greengard, Denis Gueyffier, Per-Gunnar Martinsson, and Vladimir Rokhlin. Fast direct solvers for integral equations in complex three-dimensional domains. *Acta Numerica*, 18:243–275, 2009.

[4] Wolfgang Hackbusch. A sparse matrix arithmetic based on\ cal h-matrices. part i: Introduction to {\ Cal H}-matrices. *Computing*, 62(2):89–108, 1999.

[5] Wolfgang Hackbusch and Boris N Khoromskij. A sparse ?-matrix arithmetic. *Computing*, 64(1):21–47, 2000.

[6] Kenneth L Ho and Leslie Greengard. A fast direct solver for structured linear systems by recursive skeletonization. *SIAM Journal on Scientific Computing*, 34(5):A2507–A2532, 2012.

[7] Shidong Jiang, Zhi Liang, and Jingfang Huang. A fast algorithm for brownian dynamics simulation with hydrodynamic interactions. *Mathematics of Computation*, 82(283):1631–1645, 2013.

[8] Zhi Liang, Zydrunas Gimbutas, Leslie Greengard, Jingfang Huang, and Shidong Jiang. A fast multipole method for the rotne–prager–yamakawa tensor and its applications. *Journal of Computational Physics*, 234:133–139, 2013.

[9] Nuo Wang, Gary A Huber, and J Andrew McCammon. Assessing the two-body diffusion tensor calculated by the bead models. *The Journal of chemical physics*, 138(20):204117, 2013.