Application of Multipole Methods to Two Matrix Eigenproblems

Alex Solomonoff
Institute for Math and its Applications
University of Minnesota
Minneapolis, MN 55455

November 11, 1993

Abstract

For two different types of matrices, arrowhead matrices and rank-one perturbations of diagonal matrices, their eigenvalues are the roots of a function essentially of the form $h(x) = \sum_i q_i/(x - x_i)$. It is shown that by using multipole methods to evaluate $h(x)$ we can speed up the calculation of their eigenvalues. An improvement is seen for matrices as small as 70×70. In addition, multipole methods can be used to efficiently multiply the matrix of eigenvectors by a vector.

Key Words: Multipole Methods, Symmetric Eigenproblem, Rank-one Perturbation
AMS Subject Classifications: 65F15, 65F, 31C20

1 Introduction

Numerical evaluation of the function $h(x) = \sum_i q_i/(x-x_i)$ is a task that occurs in several different situations. One example is computing the eigenvalues of arrowhead matrices. Arrowhead matrices are of the form

$$A = \begin{pmatrix} D & z \\ z^t & \rho \end{pmatrix}, \tag{1}$$

where D is an $n \times n$ diagonal matrix, z a vector, and ρ a scalar. Such matrices occur in the description of radiationless transitions in isolated molecules [2] and oscillators vibrationally coupled with a Fermi liquid [3]. O'Leary and Stewart [1] show that the eigenvalues of this matrix are the roots of the *secular function*

$$\phi_{ah}(\lambda) = \rho - \lambda - \sum_{i=1}^{n} \frac{z_i^2}{d_i - \lambda}. \tag{2}$$

The name is due to Golub [5]. The d_i are the diagonal elements of D and z_i are the elements of the vector z. They suggest applying a root-finding algorithm to $\phi_{ah}(\lambda)$ such as bisection as a method of computing the eigenvalues. They showed that there is exactly one root in each of the intervals $[d_i, d_{i+1}]$, and the remaining two roots are in the intervals $[-\infty, d_1]$ and $[d_n, \infty]$. We are assuming here that the d_i are sorted in increasing order, so that all these intervals are disjoint. We can assume this without loss of generality.

It requires $O(n)$ operations to evaluate $\phi_{ah}(\lambda)$ once, and there are $n+1$ eigenvalues, so $O(n^2 r)$ operations are needed to compute all the eigenvalues of this matrix, where r is the average number of function evaluations needed by the root-finding algorithm.

Another appearance of $h(x)$ is in the divide-and-conquer parallel algorithm for computing the eigenvalues of symmetric tridiagonal matrices, see Dongarra and Sorenson [4]. An important part of this is computing the eigenvalues of matrices of the form

$$A = D + \rho z z^t, \tag{3}$$

where the forms of D, z and ρ are as before. In [4] it is shown that the eigenvalues of $D + \rho z z^t$ are the roots of the function

$$\phi_{dc}(\lambda) = 1 + \rho \sum_{i=1}^{n} \frac{z_i^2}{d_i - \lambda}. \tag{4}$$

Dongarra and Sorenson also suggest using a root-finding algorithm to calculate the eigenvalues of this matrix. The roots are bracketed in the same intervals as in arrowhead matrices, except that if $\rho > 0$ there is a root in $[d_n, \infty]$ and none in $[-\infty, d_1]$, and vice versa if $\rho < 0$.

2 Multipole Methods

The main point of this paper is to point out that $h(x)$ can be evaluated rapidly using multipole methods, and since there are only trivial differences between $h(x)$ and the $\phi(\lambda)$, both of these eigenproblems can be sped up by using multipole methods. Actually, in

a note added in the galley proofs of [12], Bini and Pan noticed the same thing. Here, we explore the idea in more detail. In section 3, we do some numerical experiments which show that the multipole method is faster than the direct summation for reasonably small values of n. In section 4 we show how to use multipole methods to rapidly multiply the eigenvector matrix of A by a vector. This is crucial for the divide-and-conquer eigenvalue algorithm.

The function $h(x)$ appears in several other situations, and has been evaluated there using multipole methods. Examples are approximating the derivative of a function by polynomial interpolation [6], evaluation of the Riemann zeta function [7], Cauchy singular integral equations [8], and conformal mapping [9].

Multipole methods have been described in many other works, [10] [11] so we will only summarize their properties here. Fast and slow multipole methods exploit the idea that, far away from the x_i, $h(x)$ is a smooth, slowly changing function and so can be approximated by something simpler.

2.1 The Basic Multipole Algorithm

We will begin by describing the simplest version of multipole. This is a 1-dimensional, non-adaptive, slow multipole method. We will call each (q_i, x_i) a particle, with strength q_i and location x_i. Suppose that all the particles are in the interval $[0, 1]$. This is divided into a tree of subintervals. On the zero-th level there is the one interval $[0, 1]$, and the intervals on the i-th level of the tree are generated by cutting each interval on the $i-1$-th level into two equal halves, which are called its children. The contribution of the particles in each subinterval I is approximated by a multipole function of the form

$$m(x) = \sum_{k=0}^{p} \frac{a_k}{(x-c)^{k+1}} \tag{5}$$

where c is the center of I, and the coefficients are determined by interpolating at $x = \infty$, which gives [10, Theorem 2.1.1],

$$a_k = \sum_{z_i \in I} q_i (z_i - c)^k, \tag{6}$$

The degree p of the multipole is chosen so that the approximation is accurate enough for any x which is not in the subinterval itself or in either of the two subintervals of the same size which are its nearest neighbors. The error of the multipole approximation is approximately

$$\text{err} = \left(\frac{|x_{inside} - c|}{|x_{outside} - c|} \right)^p \tag{7}$$

where x_{inside} is the location of the particle inside the interval farthest from the center, and $x_{outside}$ is the nearest point to c that the multipole will be evaluated at. For our simple geometry, this ratio is at least 3, and so

$$p = -\log_3 \epsilon \tag{8}$$

where ϵ is the machine precision. This expression gives $p = 15$ for single precision, and $p = 33$ for double precision. However, for unknown reasons, we found that smaller values of p were adequate in our numerical experiments.

In all of our experiments, we chose i_{max} so that the number of particles in the smallest intervals was approximately p. This gives

$$i_{max} = \text{int}(\log \frac{n}{p}) \approx \log n - \log\log \frac{1}{\epsilon} \tag{9}$$

On the highest level of refinement, the multipole functions are constructed by directly summing the contribution of the particles in the subinterval. If there are l particles in the interval, this takes pl operations. At coarser levels, the function coefficients are generated by fusing together the two multipoles which are the children of a subinteval. If a_{k1}, a_{k2}, b_k are the child and parent multipole coefficients, and c_1, c_2, c_p are the centers of the respective intervals, then [10, Lemma 2.2.1],

$$b_l = \sum_{i=1}^{2} \sum_{k=0}^{l} a_{ki} (c_i - c_p)^{l-k} \binom{l}{k}, \tag{10}$$

which requires $O(p^2)$ operations. Since the smallest subintervals each contain about p particles, the whole multipole generation process takes $O(np)$ operations.

To evaluate the function at a point x, the contribution of the particles which are in the same subinterval as x are summed directly. The particles in the two neighboring intervals are also summed directly. The contribution of all other particles is approximated by a sum of different multipole functions on different levels of the tree. The choice of which multipoles to use in this sum is organized by a concept called an *interaction list*. The interaction list of an interval I is all intervals of the same size which are the children of the neighbors of I's parents, but are not neighbors of I. Suppose x is in I. First the multipoles of the intervals on I's interaction list are summed, then those on the interaction list of I's parent, and I's parents' parent, all the way up to the top of the tree. If n_{il} is the number of intervals on an interaction list, (3 in one dimension, 27 in two), then $O(i_{max}\, n_{il})$ multipoles are required, and so the number of operations required to evaluate $h(x)$ at n points is $O(pn) + O(pn \log n)$.

The derivative $h'(x)$ can also be rapidly computed using multipole methods. First the local interaction function is analytically differentiated. Then the multipole functions $m(x)$ are analytically differentiated and added together. No new multipole coefficients need to be generated.

2.2 Variations of the Basic Algorithm

The multipole algorithm we have described here works well if the particles are distributed evenly in the interval. Usually they will not be. If that happens, then some intervals will have many more than p particles, and some will have few or none. This will reduce the efficiency of the algorithm. This is remedied by the adaptive multipole algorithm. Instead of splitting all the subintervals in half i_{max} times, intervals are split if they contain more than p particles and not otherwise. So some parts of the tree go deeper than others. For simplicity, we have not used this here, but in almost any real application, adaptive multipole would be necessary.

This process generalizes to two or more dimensions fairly easily. In two dimensions we have complex arithmetic; the particles are located in the complex plane, and they have complex strength. Instead of intervals we have squares. However then it is less efficient than in the 1-d case. There are two reasons for this. One is that in higher dimensions, a cube has more near neighbors that have to be dealt with carefully. In d dimensions the number of cubes on the interaction list of a cube is $6^d - 3^d$. The other reason is that cubes resemble spheres less and less as the dimension increases. This degrades the convergence of the multipole expansions. In d dimensions the error is

$$\text{err} = (\frac{3}{\sqrt{d}})^p \tag{11}$$

for a p-th order multipole expansion.

The fast multipole method is a variant which is able to evaluate $h(x)$ at n points in $O(n)$ time. However the constant is quite large, and in most applications it is only competitive with other approaches if n is very large. It works by constructing local polynomial approximations to the contribution of all distant particles. The cost of constructing the local polynomials is $O(npn_{il})$. Then the cost of each function evaluation is only $O(p)$. We have not used this method, mainly because the programming is quite complex. In these matrix eigenproblems it would probably be the fastest algorithm at quite reasonable values of n.

In most multipole applications, such as trajectories of vortices in fluid, the particles (point vortices here) move with each time step and so the process of generating all the multipoles must be repeated at each time step. However our particles do not move and their strength does not change. If r is the number of function evaluations required for the root-finder to find one root, then this means that the cost of the multipole generation stage is amortized over the rn function evaluations, rather than n function evaluations as is usual. This makes multipole methods more attractive for this problem than it is for most other applications. It also makes fast multipole methods particularly attractive.

3 Numerical Results

We constructed random matrices of the form $D + \rho zz^t$, where the d_i were uniformly distributed in the interval [0,1] and the z_i^2 uniformly distributed in the interval $[0.01, 1.01]$, and with $\rho = 1$. We used Newton's method to find the $n-1$ interior roots of the secular function $\phi_{dc}(\lambda)$. This computation was performed twice, once computing $\phi_{dc}(\lambda)$ by the basic summation, and once using the simple multipole method described in section 2.1. The CPU time for these computations for a range of different matrix sizes is shown in figures 1 and 2. The degree of the multipoles used was $p = 8$ for single precision computations, and $p = 21$ for double precision. All computations were performed on a Sun Sparcstation 1.

We can see from the figures that the multipole algorithm is faster than the basic summation algorithm if the matrix size is bigger than about 70 in single precision and about 100 for double precision. The difference between single and double precision is presumably due to the larger value for p. We can also see from the figures that the curves corresponding to the multipole calculations are close to being straight lines. This suggests that most of the CPU time in the multipole calculations was spent in the multipole generation stage and the local particle summation stage. Both of these operations are $O(np)$.

4 Action of Eigenvectors on a Vector

This section is about calculating the vector $w = Q^t v$, where Q is the matrix whose columns are the eigenvectors of $A = D + \rho zz^t$, and v is some prespecified vector. This task is required in the divide-and-conquer parallel eigenvalue algorithm, where v is either e_1 or e_n. We will only carry out the derivation

for the rank-one perturbation matrices, but the process is similar for arrowhead matrices.

We want to do this without actually constructing Q, since that would require $O(n^2)$ storage and at least $O(n^2)$ time. Since there only n outputs to this problem, we would hope to do better than this. We can, using multipole methods.

Suppose $q,\ \lambda$ are an eigenpair of A, then

$$(D + \rho z z^t)q = \lambda q. \tag{12}$$

If we rearrange terms to get

$$(D - \lambda I)q = -\rho(z^t q)z, \tag{13}$$

we can see that the i-th eigenvector q is

$$q_i = \theta_i q_{0i} \tag{14}$$

where $\theta_i = (q_{0i}^t q_{0i})^{-\frac{1}{2}}$ is a normalizing constant, and

$$q_{0i} = (D - \lambda_i I)^{-1} z \tag{15}$$

The divide-and-conquer algorithm requires that we calculate the first or last element of each eigenvector. We can calculate the lth element of q_{0i} in $O(1)$ operations. This is

$$q_{oil} = \frac{z_l}{d_l - \lambda_i}. \tag{16}$$

Unfortunately the calculation of θ_i seems to require calculating all elements of q_{0i}, which would make the computation of $Q^t e_l$ an $O(n^2)$ computation, which we want to avoid. We can speed this up using multipole methods.

$$\begin{aligned} \theta_i^{-2} &= q_{0i}^t q_{0i} && (17)\\ &= z^t (D - \lambda_i I)^{-t} (D - \lambda_i I)^{-1} z && (18)\\ &= \sum_{i=1}^{n} \frac{z_i^2}{(d_i - \lambda_i)^2} && (19) \end{aligned}$$

But this is just

$$\theta_i^{-2} = \frac{1}{\rho} \frac{d}{d\lambda} \phi(\lambda_i) \tag{20}$$

and we can compute this rapidly using multipole methods. So calculating one element of $Q^t e_l$ involves one evaluation of ϕ' which takes $O(\log n)$ operations, and $O(1)$ operations to calculate $q_{0i,l}$. So calculating all of $Q^t e_l$ takes $O(n \log n)$ operations.

To calculate $Q^t v$, where v is an arbitrary vector, is more complicated, but can also be done in $O(n \log n)$ operations using multipole methods.

Equation (14) is

$$q_i = \theta_i (D - \lambda_i I)^{-1} z \tag{21}$$

which we multiply by the vector v to get the ith element of $Q^t v$:

$$\begin{aligned} q_i^t v &= \theta_i z^t (D - \lambda_i I)^{-1} v && (22)\\ &= \theta_i \sum_{j=1}^{n} \frac{z_j v_j}{d_j - \lambda_i}. && (23) \end{aligned}$$

If we write

$$\xi(\lambda) = \sum_{j=1}^{n} \frac{z_j v_j}{d_j - \lambda}, \tag{24}$$

then

$$Q^t v = \{\theta_i \, \xi(\lambda_i)\}_{i=1}^{n} \tag{25}$$

The function $\xi(\lambda)$ can be rapidly evaluated using multipole methods, and so $Q^t v$ can be computed in $O(n \log n)$ operations. However, unlike computing $\phi'(\lambda)$, this requires generating a new set of multipole coefficients for ξ.

It is also possible to efficiently compute Qv. This is a linear combination of eigenvectors, i.e.

$$w = Qv = \sum_{i=1}^{n} v_i \theta_i (D - \lambda_i I)^{-1} z \tag{26}$$

If we look at the lth element of this we have

$$\begin{aligned} w_l &= \sum_{i=1}^{n} v_i \theta_i \frac{1}{d_l - \lambda_i} z_l && (27)\\ &= -z_l \sum_{i=1}^{n} \frac{v_i \theta_i}{\lambda_i - d_l} && (28) \end{aligned}$$

and so

$$w = \{-z_l \, \zeta(d_l)\}_{l=1}^{n} \tag{29}$$

where

$$\zeta(d) = \sum_{i=1}^{n} \frac{v_i \theta_i}{\lambda_i - d} \tag{30}$$

and this function can be evaluated using multipoles. In this case we also have to construct new multipole coefficients. However this time the particles are located at the λ_i rather than at the d_i.

5 Generalization to Complex and Nonsymmetric Problems

The arguments in [1] and [4] that were used to show that the roots of the secular functions are eigenvalues generalize to nonsymmetric and complex matrices rather easily. In the case of Hermitian matrices, D and ρ are real, and so the only change is that the numerators in $\phi(\lambda)$ change from z_i^2 to $|z_i|^2$. The real

unsymmetric case is more complex. The eigenvalues of the matrix

$$A = \begin{pmatrix} D & v \\ u^t & \rho \end{pmatrix} \tag{31}$$

are the roots of the function

$$\phi_{ah}(\lambda) = \rho - \lambda - \sum_{i=1}^{n} \frac{u_i v_i}{d_i - \lambda} \tag{32}$$

and the eigenvalues of the matrix $D + \rho uv^t$ are the roots of

$$\phi_{dc}(\lambda) = 1 - \rho \sum_{i=1}^{n} \frac{u_i v_i}{d_i - \lambda}. \tag{33}$$

In both cases, if $u_i v_i > 0 \ \forall i$, then no algorithmic changes are needed. However, if some $u_i v_i$ are negative then the $\phi(\lambda)$ can still be evaluated using multipole methods, but some of the roots might be complex, and so a root-finding algorithm that works in the complex plain will be needed. Unfortunately, the nice root localization that we have in the symmetric case is completely lost in the unsymmetric case. In the symmetric eigenproblem, each interval $[d_i, d_{i+1}]$ contains exactly one root. But the unsymmetric eigenproblem, the roots can be anywhere on the real axis, or in complex conjugate pairs anywhere in the complex plane. About all we can say is that if the sign of $u_i v_i$ is the same as the sign of $u_{i+1} v_{i+1}$ then the interval $[d_i, d_{i+1}]$ has an odd number of real roots, and if the sign is different then the interval has an even number of real roots.

If A is complex non-Hermitian, then the particles lie in the complex plane, and a 2-d multipole routine is needed. In that case I have no idea how to isolate the roots.

6 Conclusions

We have shown that at least for symmetric problems, multipole methods can be effectively used to speed up the calculation of the eigenvalues of two kinds of matrices. Moreover, the algorithm is faster than the basic one for matrices that are reasonably small, i.e. 100×100 or less. With 2000×2000 matrices, ten times faster execution is possible.

If we had used a fast multipole algorithm, probably there would have been additional improvement.

In most practical applications, the eigenvalues of the matrix will not be uniformly distributed over an interval. Probably they will be clustered together in some places, and spread thin in others. In such cases an adaptive version of multipole methods would be necessary. We have used a non-adaptive code here only for simplicity.

If an effective method could be found for localizing eigenvalues, this algorithm could be extended to unsymmetric and complex matrices.

References

[1] D.P. O'Leary, G.W. Stewart, *Computing the Eigenvalues and Eigenvectors of Symmetric Arrowhead Matrices*, J. Comp. Phys. **90**, pp. 497-505 (1990).

[2] M. Bixon, J. Jortner, *Intramolecular Radiationless Transitions*, J. Chem. Phys. **48**, pp. 715-26 (1968).

[3] J.W. Gadzuk, *Localized Vibrational Modes in Fermi Liquids. General Theory*, Phys. Rev. B, **24**, pp. 1651-1663 (1981).

[4] J.J. Dongarra, D.C. Sorenson, *A Fully Parallel Algorithm for the Symmetric Eigenvalue Problem*, SIAM J. Sci. Stat. Comput., **8**, #2, s139-s154 (March 1987).

[5] G.H. Golub, *Some Modified Matrix Eigenvalue Problems*, SIAM Rev. **15**, pp. 318-334 (1973).

[6] J. Boyd, *Multipole expansions and Pseudospectral Cardinal Functions: A New Generalization of the Fast Fourier Transform*, J. Comp. Phys. **103**, 1, pp. 184-186 (1992).

[7] A.M. Odlyzko, A. Schönhage, *Fast Algorithms for Multiple Evaluation of the Riemann Zeta Function*, Trans. Amer. Math. Soc. **309** #2, pp. 797-809, (1988).

[8] A. Gerasoulis, *A Fast Algorithm for the Multiplication of Generalized Hilbert Matrices with Vectors*, Math. Comp. **50**,#181, pp. 79-188, (1988).

[9] S.T. O'Donnel, V. Rokhlin, *A Fast Algorithm for the Numerical Evaluation of Conformal mappings*, SIAM J. Sci. Stat. Comput., **10**, #3, pp. 475-487, May 1989.

[10] L. Greengard, *The Rapid Evaluation of Potential Fields in Particle System*, MIT Press, 1988.

[11] L. Van Dommeln, E.A. Rundensteiner, *Fast, Adaptive Summation of Point Forces in the Two-Dimensional Poisson Equation*, J. Comp. Phys., **83**, pp.126-147, (1989).

[12] D. Bini, V. Pan, *Practical Improvement of the Divide-and-Conquer Eigenvalue Algorithms*, Computing, **48**, pp. 109-123 (1992).

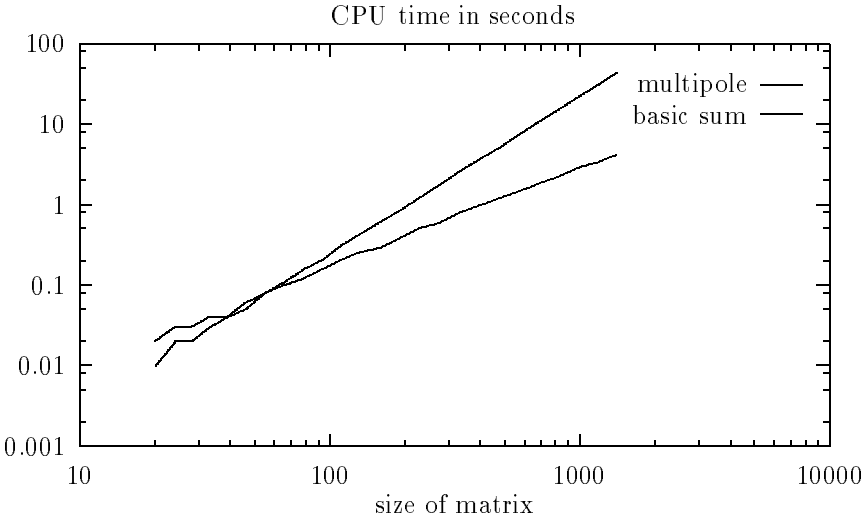

Figure 1: Performance of basic and multipole algorithms in single precision.

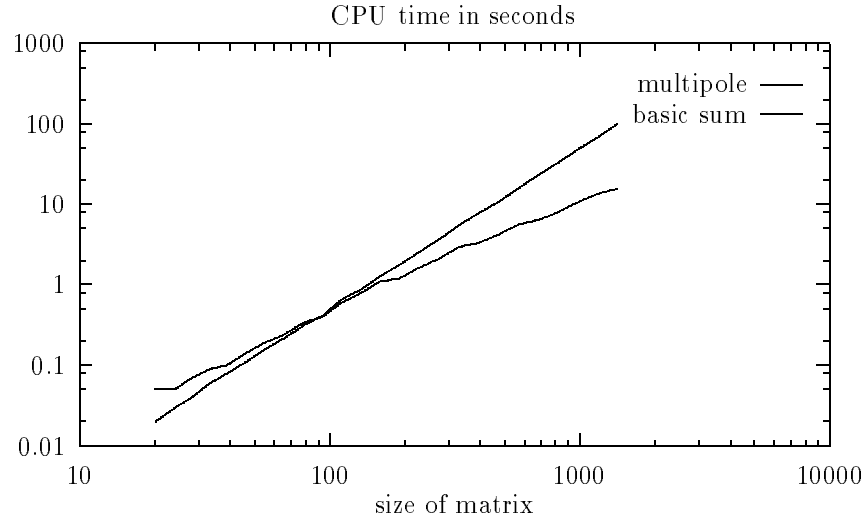

Figure 2: Performance of basic and multipole algorithms in double precision.