

Simple recursive implementation of fast multipole method

P. B. Visscher^{*,1}, D. M. Apalkov²

Department of Physics and Astronomy and the MINT Center, U. of Alabama

Abstract

In this paper we present an implementation of the well known “fast multipole” method for the efficient calculation of dipole fields, that uses polynomials in the Cartesian coordinates rather than spherical harmonics. This has considerable efficiency and simplicity advantages. We have implemented it in the context of an arbitrary hierarchical system of cells – no periodic mesh is required. The implementation is in terms of recursive functions. Results are given for application to micromagnetic simulation.

Key words: fast multipole method

PACS: 07.05Tp, 75.40.Mg, 65C20

1 Introduction and Motivation

“Fast multipole” methods [1] (FMM’s) have been used for about 15 years [2], to speed up large scale simulations involving long ranged forces. The advantage of this method is that the required time is of order N , where N is the number of objects (atoms, computational cells, etc.) that are interacting, instead of order N^2 like brute-force calculation of all pair interactions. Fast multipole methods have not been used until quite recently [3], [4], [5], [6], and [7] in

* Corresponding author.

Email addresses: visscher@ua.edu (P. B. Visscher), apalk001@ua.edu (D. M. Apalkov).

¹ Supported by DOE grant DE-FG02-98ER45714 and NSF grant ECS-008534 to the U. of Alabama

² Supported by NSF grant DMR-0213985 to the U. of Alabama and by the DOE Computational Materials Science Network

micromagnetics, although similarly motivated hierarchical methods have been used [8].

The adoption of the FMM has been slowed somewhat by a perception that the coefficient of N is so large that the FMM becomes faster than the order- N^2 method only for very large N . We find that this is true only when one considers very high order multipoles (values of order 20 have often been used [9]) and that at a level of accuracy that makes any sense in an application such as micromagnetics (where other uncertainties may make it pointless to go beyond second or third order) the FMM gives substantial advantages for most system sizes currently studied.

Another barrier to the adoption of the FMM is that it is conventionally implemented using spherical harmonics. This substantially increases both the complexity and the execution time of a code. We present in Sec. 3 an automatic algorithm for manipulation of Cartesian multipoles. Our method can be consistently implemented using only real polynomials in the cartesian coordinates, and this implementation is substantially simpler and more efficient than the usual complex spherical-harmonic implementation, or previous Cartesian implementations [10], [11].

A second innovation in the present paper is that we do the hierarchical traversals of the system in a very simple way, using recursive functions. This implementation works for an arbitrary hierarchical system, and does not assume a particular geometric structure (e.g., cubes as in the original Greengard-Rokhlin implementation [1]).

2 Fast multipole method: recursive implementation

The FMM is a method for calculating the field at each point of a system, due to all the sources in the system. To do this by “brute force” requires a number of computations of order N^2 . We can reduce this by lumping sources together into large cells (one such “source cell” is shown in Fig. 1) whose field is computed by a multipole expansion (e.g., Eq. (13) below) rather than one at a time.

We can also gain efficiency by lumping the field points together: the field due to a distant source is smooth over the “field cell” shown in Fig. 1 and can be expanded in a power series (Taylor expansion) within this cell. To find the field at a particular point in the field cell, it is faster to evaluate the Taylor series than to evaluate the multipole expansion.

For a given cell (for example, the one labeled “this” in Fig. 2a, there is a

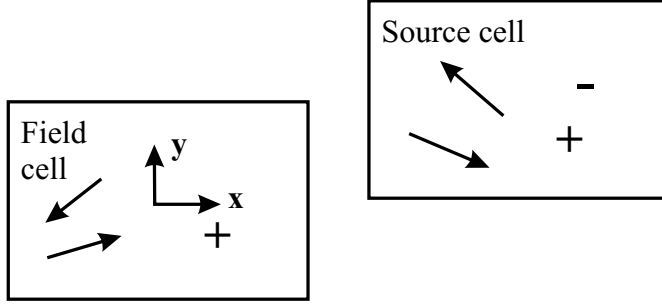


Fig. 1. Schematic system illustrating the advantage of the fast multipole method: the multipole moment of the charges and/or dipoles in the "source cell" is calculated, and used to calculate a Taylor expansion of the potential (and hence of the field) about the origin of the axis system shown in the "field cell".

certain set of sources which are far enough away that their fields are smooth over "this" cell, one of which (labeled "far source") is shown. The circle schematically separates the near from the far sources. The idea of the FMM is that this smooth field (in the form of a Taylor expansion) needs to be calculated only once, and used for all the objects inside "this" cell. The near sources are handled separately as discussed below.

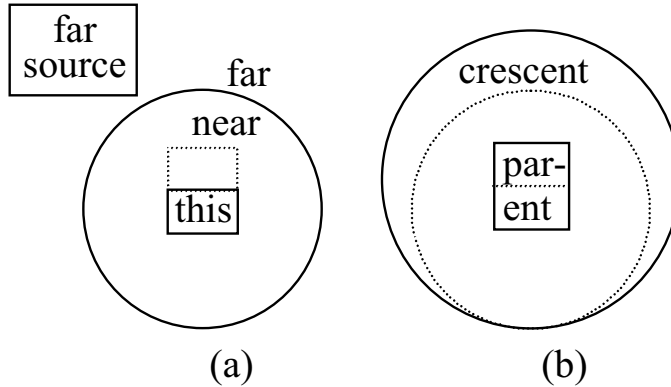


Fig. 2. (a) Schematic diagram illustrating sources that are "near" and "far" from a field cell ("this"). (b) The corresponding diagram for the parent cell of "this", showing how the child's smooth field is calculated from that of the parent by adding sources in the crescent region.

So far, this is not very complicated – what makes the FMM difficult is the logistics of deciding which sources are included in the Taylor expansion, making sure all sources are included somehow, deciding how big the field cell is, etc. In the original implementation [1], this is determined by a set of geometrical rules: the fields of the cells that touch the field cell (even just at a corner) are not smooth, more distant ones are smooth. In the present implementation, we avoid some of the complexity by using a very simple recursive procedure that does not rely on any particular geometrical structure, such as a cubic lattice. We do not claim that this is more efficient than using a geometrically prescriptive algorithm, just that it is comparably efficient and much simpler.

All FMM implementations begin by describing the system as a hierarchical tree. We will assume a binary tree (Fig. 3) for simplicity – the method would work just as well if the cells had more than two children.

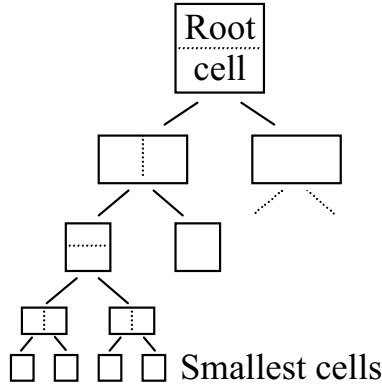


Fig. 3. The binary tree describing the system. The top (root) cell represents the entire system, and the lowest (leaf) nodes represent the computational cells of our discretization.

The root cell at the top of Fig. 3 consists of the entire system, and the lowest (“leaf”) cells are individual particles in a particle simulation, or computational cells in the case of a continuum simulation.

Our recursive procedure for calculating fields is depicted schematically in Fig. 2. In Fig. 2a we show a cell (“this”) with the circle separating near sources from the far sources whose field is smooth. Our objective is to write a function, which we will call “updateStat()”, which calculates an object called SmoothField, consisting of the Taylor coefficients of the smooth field in “this” cell. The procedure is recursive in the sense that we assume that the smooth field of the parent cell (shown in 2b) is known. We need to add to this the fields of sources far from “this” but near to the parent, i.e., those in the crescent-shaped region in 2b. We will store these sources in a list called “Partners”.

Thus the recursive function “updateStat()” first sets this cell’s SmoothField equal to that of its parent, then adds the fields of the sources in the Partners list to get the correct SmoothField for this cell. The function then recursively calls itself twice, once for each child. We need only call “updateStat()” explicitly for the **root** cell – this recursive call will work its way down the tree to calculate the smooth field in **every** cell. When we get down to the smallest cells we must sometimes calculate the fields of a few near neighbors by hand, in a continuum context. In the case of point monopoles or dipoles, this is not necessary because Eq. (20) or (21) (below) is exact in these cases. To begin the recursion, we must know the SmoothField of the root cell, i.e. the field of the sources that are far from everything. In a system with only internal sources, this smooth field is zero; if our system is subjected to an external field, the latter is the root’s smooth field.

Clearly this algorithm has the advantage of simplicity – it can be implemented in a few lines, using the C++ Standard Template Library (STL) for the lists. However, we have not yet specified how the Partners list is generated, or how we determine whether a source cell is far enough away that its field is smooth within a field cell. We derive in the Appendix a simple criterion for the latter: the field is smooth if the “opening angle” of the field cell, defined as (diameter of field cell)/(center-to-center distance), as well as the opening angle of the source cell, is less than some critical value α , typically of order 0.5.

The procedure for creating the Partners list is also recursive. For each cell, such as “this” in Fig. 2a, we start with a list of its parent’s near sources (the interior of the large circle in Fig. 2b) – these are the ones this cell and its children will have to take into account. This list (parent’s near sources) may as well be initially stored in the same list object, “Partners”. We need to traverse this list and decide which sources must remain in the Partners list, i.e., which are in the crescent region. This is determined by the opening angle criterion: if a source is too close, it is copied to the Partners list of both children of this cell, and removed from “this” Partners list. We will refer to the function that does this as “CullPartners()”. At the end of this list traversal, the children’s Partners lists are correctly set up (containing all near sources of their parent, “this”), ready for the two recursive calls to CullPartners() for the children. One detail we have not considered is that we would like the Partner sources to be similar in size to “this” cell, and the partners of the parent might be too large. Thus if one of the parent’s near sources is also near to “this” cell, we check its size and if it is larger than this cell, we split it in half and put both halves at the end of the list so they are checked again. The splitting may have made one or both of the source’s children “far” from this, so they can be left on the Partner list; otherwise they are passed to “this”’s children. Thus the sources that end up being passed down to the children are no larger than “this” cell. [The simple C++ code that accomplishes the above may actually be easier to follow than a verbal description – it is available at <http://bama.ua.edu/~visscher/mumag/code>.]

The “Partners” lists that result from the above recursive algorithm are similar to those created in the conventional Greengard-Rokhlin algorithm. Figure 4 shows the Partners lists for one small cell and its ancestors, for a two-dimensional square lattice. Part (a) of the figure shows the root cell; we pick one of the children to be “this” cell in the next part of the figure. In parts (a-c) the Partners list is empty – no cells are far enough from “this”. The number of near sources increases to 4 due to subdivision. In part (d) there are 3 cells far enough away from “this” that their fields are smooth in “this” and they are labeled “partner” to indicate that they are on the Partners list. The “near” cells (bounded by the heavy line) must be passed to the children via their Partners list.

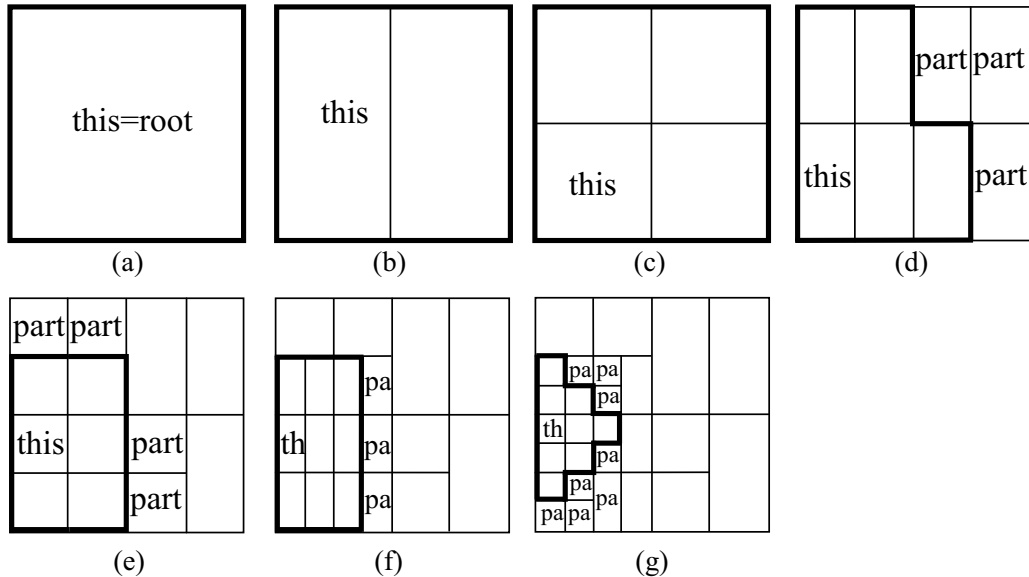


Fig. 4. Illustration of how `CullPartners()` generates the partner lists as we move recursively down the tree. In each part of the figure, one cell is designated "this" (or "th"). Heavy line bounds "near" sources. Source cells on the Partners list of "this" cell are labeled "Partner", "Part", or "Pa".

Unlabeled cells outside this are in the parent's SmoothField.

The important point here is that the pattern of cell interactions in Fig. 4 is basically the same as that generated by a non-recursive algorithm: the difference is that the recursive algorithm can be implemented in a few lines and with very simple logic, whereas in the Greengard-Rokhlin approach the programmer has to generate Fig. 4 by hand and hard-wire the geometrical relationships into a more complicated algorithm.

The reader will note that at the last stage (Fig. 4g) when "this" cell and all of its near neighbors are the smallest cells in the hierarchy, the field due to these near sources is not smooth. In this case the interaction must be handled by hand in a problem-dependent way. In a system of point charges or dipoles, interactions between all pairs of charges can be calculated explicitly. In our present continuum micromagnetic system, we need to calculate an effective field at this cell – we will describe below (Sec. 4) how this is done.

The above recursive formulation is independent of any special structure of the cell lattice, and can in fact be applied to systems whose computational cells are irregular. With a procedure for rearranging the tree occasionally, it can be applied to systems whose components move around, such as colloids.

3 Cartesian-coordinate formulation of the fast multipole method

We will describe the magnetic field \mathbf{H} in each cell of our system by a power series (Taylor) expansion of the scalar potential; the scalar potential can be defined as long as there is no free current [12]. In most implementations of the FMM [2], the scalar potential is expanded in terms of spherical harmonics. This requires computing the spherical harmonics, complicating the code and increasing the execution time. The usual motivation for using spherical harmonics is that there are fewer basis functions to a given order l : there are $(l+1)^2$ spherical harmonics of order $\leq l$ but we show in Appendix B that there are $(l+1)(l+2)(l+3)/6$ in a general Cartesian expansion. The difference is because the spherical harmonics span only harmonic functions (solutions of the Laplace equation). There are constraints on the Cartesian expansion coefficients of a harmonic function; we can choose a subset of $(l+1)^2$ of them from which the rest can be computed very rapidly. Thus it is possible to remove this inefficiency of the Cartesian expansion; however, since this complicates the code and the inefficiency is less than a factor of two up to order 8 (see Table 1), we have constructed an initial implementation without taking advantage of this.

Table 1

Comparison of number of components for spherical harmonic and cartesian expansions.

order l	0	1	2	3	4	5	6	7	8
# of sph. harm of order l	1	3	5	7	9	11	13	15	17
NumBelow[l] = # of order $\leq l$	1	4	9	16	25	36	49	64	81
# of Cartesian monomials of order l	1	3	6	10	15	21	28	36	45
# of Cartesian monomials of order $\leq l$	1	4	10	20	35	56	84	120	165

We will use the shorthand notation $\mathbf{r}^{\mathbf{n}}$ for an arbitrary Cartesian monomial:

$$\mathbf{r}^{\mathbf{n}} \equiv x^{n_x} y^{n_y} z^{n_z} \tag{1}$$

so that an arbitrary Taylor expansion of the scalar potential $V(\mathbf{r})$ can be written

$$V(\mathbf{r}) = \sum_{\mathbf{n}} \frac{1}{\mathbf{n}!} V_{\mathbf{n}} \mathbf{r}^{\mathbf{n}} \tag{2}$$

where we have normalized the Taylor coefficient $V_{\mathbf{n}}$ by the factor $\mathbf{n}! \equiv n_x! n_y! n_z!$,

so it is defined by

$$V_{\mathbf{n}} = \left(\frac{\partial^{\mathbf{n}}}{\partial \mathbf{r}^{\mathbf{n}}} V(\mathbf{r}) \right)_{\mathbf{r}=\mathbf{0}} \quad (3)$$

We will denote the total order $n_x + n_y + n_z$ by n , and retain terms for which $n \leq l$, where l is a maximum order.

To compute this Taylor expansion efficiently, we sequentially number the possible values of the integer vector \mathbf{n} by an index $i_{\mathbf{n}}$, as in Table 2 (the exact sequence is not important as long as the total order n increases monotonically with i).

Table 2

Sequential numbering of multipole components \mathbf{n} .

$i_{\mathbf{n}}$	0	1	2	3	4	5	6
\mathbf{n}	(0,0,0)	(1,0,0)	(0,1,0)	(0,0,1)	(2,0,0)	(1,1,0)	(0,2,0)

Then the coefficients $V_{\mathbf{n}}$ can be stored in a linear array, which we have implemented as a C++ object called an “mpole”; to order $l = 5$ its size is only 56, from Table 1 .

3.1 Translation of origin

When we compute the smooth field over a cell from the smooth field of its parent, we need to be able to translate the origin, i.e., to expand in powers of \mathbf{r} the expression

$$\begin{aligned} V(\mathbf{r} + \mathbf{c}) &= \sum_{\mathbf{n}} \frac{1}{\mathbf{n}!} V_{\mathbf{n}} (\mathbf{r} + \mathbf{c})^{\mathbf{n}} = \sum_{\mathbf{n}} \frac{1}{\mathbf{n}!} V_{\mathbf{n}} (\mathbf{x} + \mathbf{c}_x)^{n_x} \dots \\ &= \sum_{\mathbf{n}} \frac{1}{\mathbf{n}!} V_{\mathbf{n}} \sum_{m_x} \frac{n_x!}{m_x!(n_x-m_x)!} x^{m_x} c_x^{n_x-m_x} \dots \\ &= \sum_{\mathbf{n}} \frac{1}{\mathbf{n}!} V_{\mathbf{n}} \sum_{\mathbf{m}} \frac{\mathbf{n}!}{\mathbf{m}!(\mathbf{n}-\mathbf{m})!} \mathbf{r}^{\mathbf{m}} \mathbf{c}^{\mathbf{n}-\mathbf{m}} = \sum_{\mathbf{m}} \frac{1}{\mathbf{m}!} V'_{\mathbf{m}} \mathbf{r}^{\mathbf{m}}. \end{aligned} \quad (4)$$

where

$$V'_{\mathbf{m}} = \sum_{\mathbf{n}} \frac{1}{(\mathbf{n} - \mathbf{m})!} V_{\mathbf{n}} \mathbf{c}^{\mathbf{n}-\mathbf{m}} = \sum_{\mathbf{p}} \frac{1}{\mathbf{p}!} V_{\mathbf{m}+\mathbf{p}} \mathbf{c}^{\mathbf{p}} \quad (5)$$

gives the Taylor expansion about point \mathbf{c} in terms of that around the origin.

We will refer to this as the “down-convolution” (because the high-order $V_{\mathbf{m}+\mathbf{p}}$ contributes to the lower-order $V'_{\mathbf{m}}$) of the two mpoles V and \mathbf{c}' where

$$\mathbf{c}'^{\mathbf{p}} \equiv \frac{1}{\mathbf{p}!} \mathbf{c}^{\mathbf{p}} = 1, c_x, c_y, c_z, \frac{1}{2}c_x^2, \dots \quad (6)$$

To compute the down-convolution, we store the sequential index $i_{\mathbf{m}+\mathbf{p}}$ in an array $\text{ConvSum}[i_{\mathbf{m}}][i_{\mathbf{p}}]$, so that Eq. (5) can be computed as

$$V'_{\mathbf{m}} = \sum_{i_{\mathbf{p}}=0}^{\text{NumBelow}[l-m]-1} V_{\text{ConvSum}[i_{\mathbf{m}}][i_{\mathbf{p}}]} \mathbf{c}'^{\mathbf{p}} \quad (7)$$

In principle the number of terms is infinite, but because we store V only up to l th order, the number of terms, $\text{NumBelow}[l-m]$, is the largest index $i_{\mathbf{p}}$ for which the total order $m+p \leq l$, and is given in Table 1.

As in the conventional spherical-harmonic formulation, it will turn out that the other operations needed for the FMM can also be expressed as convolutions.

3.2 Taylor expansion of field of a multipole

We define the multipole moments of a charge distribution $\rho(\mathbf{r})$ to be

$$\mathbf{Q}_{\mathbf{n}} \equiv \frac{1}{\mathbf{n}!} \int \rho(\mathbf{r}) \mathbf{r}^{\mathbf{n}} d^3\mathbf{r} \quad (8)$$

This is a C++ object of class `mpole`. The first-order components are exactly the usual magnetic dipole moment. In our application the charge is magnetic, but of course this formalism works equally well for electric or gravitational charge. Even though a closed system cannot have a net zeroth order magnetic monopole moment ($\mathbf{Q}_{(0,0,0)}$), we may want to subdivide a system into parts that do, so we will not exclude this possibility.

We want to be able to shift the origin of the multipole Q . It is easy to see that this can be done using an equation similar to Eq. (5) above: the moment Q about the origin is given in terms of moments Q' about the point \mathbf{c} , by

$$Q_{\mathbf{n}} = \sum_{\mathbf{p}} \frac{\mathbf{c}^{\mathbf{p}}}{\mathbf{p}!} Q'_{\mathbf{n}-\mathbf{p}} = \sum_{\mathbf{p}} \mathbf{c}'^{\mathbf{p}} Q'_{\mathbf{n}-\mathbf{p}} \quad (9)$$

Unlike the down-convolution, this “up-convolution” has a finite number of terms – the sum is over \mathbf{p} ’s of order less than n . The up-convolution can be

calculated using the same arrays we used for the down-convolution; specifically, for each pair of indices \mathbf{p} and \mathbf{m} , $Q_{\mathbf{m}+\mathbf{p}} = Q_{\text{ConvSum}[i_{\mathbf{m}}][i_{\mathbf{p}}]}$ has a term $\mathbf{c}^{\mathbf{p}} Q'_{i_{\mathbf{m}}}$.

The magnetic scalar potential produced by this distribution at position \mathbf{c} is

$$V(\mathbf{c}) \equiv \frac{1}{4\pi} \int \frac{\rho(\mathbf{r})}{|\mathbf{c} - \mathbf{r}|} d^3\mathbf{r} \quad (10)$$

(We are using SI units; as is conventional, we do not include a μ_0 factor in V , so the energy per unit magnetic charge is $\mu_0 V$. The electrostatic case is obtained by inserting a factor $1/\epsilon_0$).

If we now expand $1/4\pi|\mathbf{c} - \mathbf{r}|$ in a Taylor series:

$$\frac{1}{4\pi|\mathbf{c} - \mathbf{r}|} = \sum_{\mathbf{p}} \frac{1}{\mathbf{p}!} D_{\mathbf{p}}(\mathbf{c}) \mathbf{r}^{\mathbf{p}} \quad (11)$$

where $D_{\mathbf{p}}$ is the \mathbf{p} -th derivative of the Coulomb potential,

$$D_{\mathbf{p}}(\mathbf{c}) \equiv \frac{\partial^{\mathbf{p}}}{\partial(-\mathbf{c})^{\mathbf{p}}} \frac{1}{4\pi|\mathbf{c}|} \quad (12)$$

we obtain

$$V(\mathbf{c}) = \sum_{\mathbf{p}} D_{\mathbf{p}}(\mathbf{c}) Q_{\mathbf{p}} \quad (13)$$

(The minus sign in Eq. (12) is required in order to avoid them in Eqs. (11) and (13).) We can also think of $D_{\mathbf{p}}$ as the coefficients of a Taylor expansion near the origin of the potential due to a monopole at \mathbf{c} .

The coefficients $V_{\mathbf{m}}$ in a Taylor expansion (Eq. (2)) of this potential about \mathbf{c} are given by

$$V_{\mathbf{m}} = \frac{\partial^{\mathbf{m}}}{\partial \mathbf{c}^{\mathbf{m}}} V(\mathbf{c}) = (-1)^{\mathbf{m}} \sum_{\mathbf{p}} D_{\mathbf{p}+\mathbf{m}}(\mathbf{c}) Q_{\mathbf{p}} \quad (14)$$

Note that this down-convolution has essentially the same form as Eq. (5) above; the same function can be used to compute it.

What remains is to calculate the derivative $D_{\mathbf{n}}$ of the Coulomb potential. This can be done recursively [11]. We will prove by mathematical induction

that $D_{\mathbf{n}}(\mathbf{r})$ has the form

$$D_{\mathbf{n}}(\mathbf{r}) = \frac{1}{r^{2n+1}} \sum_{\mathbf{p}} F_{\mathbf{n}}[\mathbf{p}] \mathbf{r}^{\mathbf{p}} \quad (15)$$

This is clearly true for the case $\mathbf{n} = 0$, with $F_0[\mathbf{p}] = 0$ unless $\mathbf{p} = 0$, and $F_0[\mathbf{0}] = 1/4\pi$. Assume it is true for the vector index \mathbf{n} , and consider the higher order index $\mathbf{n} + \hat{\mathbf{x}}$, where $\hat{\mathbf{x}} = (1, 0, 0)$. We have

$$\begin{aligned} D_{\mathbf{n} + \hat{\mathbf{x}}}(\mathbf{r}) &= -\frac{\partial}{\partial x} D_{\mathbf{n}}(\mathbf{r}) = -\sum_{\mathbf{n}} F_{\mathbf{n}}[\mathbf{p}] \frac{\partial}{\partial x} \frac{x^{p_x} y^{p_y} z^{p_z}}{r^{2n+1}} \\ &= \sum_{\mathbf{p}} F_{\mathbf{n}}[\mathbf{p}] \left[\frac{-p_x r^2 \mathbf{r}^{\mathbf{p} - \hat{\mathbf{x}}} + (2n+1)x \mathbf{r}^{\mathbf{p}}}{r^{2n+3}} \right] \\ &= \sum_{\mathbf{p}} F_{\mathbf{n}}[\mathbf{p}] \left[\frac{-p_x \mathbf{r}^{\mathbf{p} - \hat{\mathbf{x}} + 2\hat{\mathbf{y}}} - p_x \mathbf{r}^{\mathbf{p} - \hat{\mathbf{x}} + 2\hat{\mathbf{z}}} + (2n+1-p_x) \mathbf{r}^{\mathbf{p} + \hat{\mathbf{x}}}}{r^{2n+3}} \right] \end{aligned} \quad (16)$$

Since we want this to equal Eq. (15) with \mathbf{n} replaced by $\mathbf{n} + \hat{\mathbf{x}}$, or

$$D_{\mathbf{n}}(\mathbf{r}) = \frac{1}{r^{2n+3}} \sum_{\mathbf{p}'} F_{\mathbf{n} + \hat{\mathbf{x}}}[\mathbf{p}'] \mathbf{r}^{\mathbf{p}'} \quad (17)$$

we find that $F_{\mathbf{n}}[\mathbf{p}]$ contributes to the higher-order $F_{\mathbf{n} + \hat{\mathbf{x}}}[\mathbf{p}']$ for three values of \mathbf{p}' (three powers of \mathbf{r}), namely

$$\begin{aligned} \mathbf{p}' = \mathbf{p} + \hat{\mathbf{x}} : \quad & F_{\mathbf{n} + \hat{\mathbf{x}}}[\mathbf{p} + \hat{\mathbf{x}}] + = F_{\mathbf{n}}[\mathbf{p}] (2n + 1 - p_x) \\ \mathbf{p}' = \mathbf{p} - \hat{\mathbf{x}} + 2\hat{\mathbf{y}} : \quad & F_{\mathbf{n} + \hat{\mathbf{x}}}[\mathbf{p} - \hat{\mathbf{x}} + 2\hat{\mathbf{y}}] + = -F_{\mathbf{n}}[\mathbf{p}] p_x \\ \mathbf{p}' = \mathbf{p} - \hat{\mathbf{x}} + 2\hat{\mathbf{z}} : \quad & F_{\mathbf{n} + \hat{\mathbf{x}}}[\mathbf{p} - \hat{\mathbf{x}} + 2\hat{\mathbf{z}}] + = -F_{\mathbf{n}}[\mathbf{p}] p_x \end{aligned} \quad (18)$$

Here “+ = ” means, as in C++, that $F_{\mathbf{n} + \hat{\mathbf{x}}}[\mathbf{p}']$ may have contributions from other values of \mathbf{p} , which we accumulate as we loop over \mathbf{p} . The first few values obtained from Eq. (18) are $F_x[x] \equiv F_{(1,0,0)}[(1, 0, 0)] = 1/4\pi$, $F_{xx}[xx] \equiv F_{(2,0,0)}[(2, 0, 0)] = 2/4\pi$, and $F_{xx}[yy] \equiv -1/4\pi$. Since the array $F_{\mathbf{n}}[\mathbf{p}]$ is relatively sparse, we store it in the form of a list of \mathbf{p} s for each \mathbf{n} : for $j = 0 \dots \text{DerTerms}[i_{\mathbf{n}}]$, we store the vector \mathbf{p} as $\text{DerPower}[i_{\mathbf{n}}][j]$, and $F_{\mathbf{n}}[\mathbf{p}]$ itself as $\text{DerCoeff}[i_{\mathbf{n}}][j]$. (“Der” stands for Derivative.) This makes the calculation of $D_{\mathbf{n}}$ very fast:

$$D_{\mathbf{n}}(\mathbf{r}) = \frac{1}{4\pi r^{2n+1}} \sum_{j=0}^{\text{DerTerms}[i_{\mathbf{n}}]} \text{DerCoeff}[i_{\mathbf{n}}][j] * \mathbf{r}^{\text{DerPower}[i_{\mathbf{n}}][j]} \quad (19)$$

A formula like Eq. (15) was used previously by Shimada et al [10] but the coefficients $F_{\mathbf{n}}[\mathbf{p}]$ were all calculated by hand, unlike the present automatic recursive method.

The rather abstract Eq. (14) contains within it all the familiar results of magnetostatics: for a magnetic monopole Q_0 at the origin, Eq. (14) gives the inverse-square law for the magnetic field at \mathbf{r} :

$$H_x(\mathbf{r}) = -V_x = D_x Q_0 = \frac{Q_0 x}{4\pi r^3} \quad (20)$$

(here V_x means $V_{(1,0,0)}$, and we have used $D_x = x/4\pi r^3$). For a dipole (Q_x, Q_y, Q_z) at the origin,

$$H_x(\mathbf{r}) = D_{xx}Q_x + D_{xy}Q_y + D_{xz}Q_z \quad (21)$$

where for example $D_{xx} = \frac{2x^2 - y^2 - z^2}{4\pi r^5}$; this is the usual dipole field. As a higher-order example, we give the case $\mathbf{n} = (2, 1, 0)$:

$$D_{xxy}(\mathbf{r}) = \frac{-\partial^3}{\partial x^2 \partial y} \frac{1}{4\pi r} = \frac{1}{4\pi r^7} [12x^2 y - 3y^3 - 3yz^2] \quad (22)$$

Note that we have written the equations explicitly only for the case of recursion in the x -direction in \mathbf{n} -space – to get to $\mathbf{n} = (0, 0, 1)$, for example, we need to write similar equations for recursion in the z -direction.

Now we have all of the necessary ingredients for a FMM calculation: Eq. (14) to calculate Taylor expansions of multipoles at cell centers, and Eq. (5) to shift them to the centers of descendant cells.

4 Application to continuum micromagnetic system

The above method is applicable to systems of discrete charges or dipoles, but we will describe here the application to the case of a continuum micromagnetic system. In micromagnetics, one divides a ferromagnetic object into cells (often cubes). The rate of change $d\mathbf{M}/dt$ of the average magnetization in a cell is assumed to be given by the Landau-Lifshitz equation [13], in terms of the total magnetostatic field (usually also averaged over the cell) created by the magnetizations (usually assumed uniform) of all the other cells. Thus the problem is to calculate this magnetostatic field. The FMM algorithm described above calculates these automatically, except for the fields of very near neighboring cells [for example, the five in Fig. 4g]. The average field in a cell depends linearly on the uniform magnetization of a source cell, in a way formally identical to Eq. (21) above; the coefficients D_{ij} are referred to in the literature [14] as the “micromagnetic kernel”. This can be incorporated into the FMM algorithm in a very simple way. In the function `CullPartners`, when

a partner of the parent is “near” so we would like to pass it to the children of “this” cell (see Fig. (2)), but “this” cell is already a smallest cell, we put it on a new list called “NearPartners” whose fields will be computed from the kernel rather than from Eq. (14). These kernels can be calculated analytically [15], but we have calculated them using a recursive technique [16].

In Fig. (5), we give the computation times for a FMM calculation as a function of the number $N = L^3$ of cells in a system of length L cells, for various values of the opening angle α and the maximum order p . For $\alpha L < 1$ and $p = 1$, clearly the algorithm reverts to the usual pairwise $O(N^2)$ algorithm, so the line has slope 2(???). It remains $O(N^2)$ for $p > 1$, but the prefactor (intercept) increases. Error estimate?

$$\epsilon = \alpha^p?$$

fig showing realistic? Give alpha in figs.

Fig. 5. Logarithmic plot of FMM computation time as a function of number of cells, for several values of the opening angle α and the order p .

To estimate the optimal values for the opening angle and order p , we show in Fig. (6) the computation time as a function of order, for various fixed values of the error.

Fig. 6. Plot of computation time vs. order p , for various values of the error estimate ϵ (Eq. ...).

It can be seen that the ...

5 Verification for NIST standard problem #2 (MRAM switching)

As a test of the FMM code described above, we have written a C++ program that implements the recursive Cartesian FMM algorithm described above, and applied it to a rectangular slab of magnetic material (an MRAM element – this is NIST standard problem number 2 [17]). Fig. XX shows the resulting hysteresis curve, for $\alpha = 0.5$ and several choices of multipole order p ? Convergence with respect to cell size is a separate and complicated issue [17], which we do not address here.

A Smoothness criterion

Here we will determine how far apart a source and field cell must be for the field of the source to be “smooth” over the field cell. Suppose the multipole expansion due to a source cell is truncated after the p th order term (the 2^p -pole). The first omitted multipole moment is of order qr^{p+1} where r is the radius of the source cell (the maximum distance between a corner and the center) and q is a quantity with units of charge. Thus the error in the field at a field point due to multipole truncation is approximately qr^p/d^{p+2} where d is the distance between centers. This is proportional to $(\alpha/2)^{p+1}$ where α is the opening angle $2r/d$ of the source cell as seen by the field cell. The error due to truncation of the Taylor series at order p is similarly proportional to the $p + 1$ st power of the opening angle of the field cell. Thus we will consider two cells “far apart” if both opening angles are less than a chosen maximum value.

B Counting Cartesian components

In this Appendix we justify the statement that the number of monomials [Eq. (1); this is also the number of ordered triples of nonnegative integers (n_x, n_y, n_z)] of total order $\leq l$ is $(l + 1)(l + 2)(l + 3)/6$. To see this, imagine placing l coins in a row on a table. Place a red pencil between a pair of adjacent coins, to the left of the first coin, or to the right of the l^{th} coin ($l + 1$ possible positions). Then place a green pencil somewhere among the $l + 1$ objects on the table ($l + 2$ possible positions – we consider the left and right side of the red pencil as different positions). Finally, place a blue pencil among the $l + 2$ objects now on the table ($l + 3$ positions). There are now $(l + 1)(l + 2)(l + 3)$ possible configurations, but those related by the 6 permutations of the three pencils are equivalent. That is, if we replace the colored pencils by identical black ones, there are only $(l + 1)(l + 2)(l + 3)/6$ configurations. These are in one-to-one correspondence with the monomials: set n_x equal to the number of coins to the left of the first black pencil, n_y equal to the number between that pencil and the second, and similarly for n_z .

References

- [1] L. Greengard and V. Rokhlin, A fast algorithm for particle simulations, *J. Comp. Phys.* **73**, 325-348 (1987). The word “fast” in this context can be regarded as a synonym for “hierarchical”, referring to a class of methods, of which the earliest is the Fast Fourier Transform [J. W. Cooley and J. W. Tukey, *Math. Comput.* **19**,

- 297 (1965); P. B. Visscher, The FFT: Fourier Transforming One Bit at a Time, *Computers in Physics* **10**, 438-443 (1996)] that speed up an order(N) process to order($\log N$) by formulating it in a hierarchical way. A very clear application of the Greengard-Rokhlin method to micromagnetics is given by C. Seberino and H. N. Bertram, Concise, Efficient Three-Dimensional Fast Multipole Method for Micromagnetics, *IEEE Trans. Magn.* **37**, 1078 (2001).
- [2] J. Shimada , H. Kaneko, and T. Takada, *J. Comput. Chem.* **14**, 867 (1993).
- [3] J. Blue and M. Scheinfein, Using multipoles decreases computation time for magnetostatic self-energy, *IEEE Trans. Magn.* **27**, 4778 (1991).
- [4] S. Yuan and H. N. Bertram, Fast adaptive algorithms for Micromagnetics, *IEEE Trans. Magn.* **28**, 2031 (1992).
- [5] Y. Gunal and P. B. Visscher, Brownian Dynamics Simulation of Magnetic Colloid Aggregation, *IEEE Trans. Magn.* **32** 4049-4051 (1996).
- [6] G. Brown, M. Novotny, and P. Rikvold, Langevin simulation of thermally activated magnetization reversal in nanoscale pillars, *Phys. Rev. B* **64**, 134422 (2001).
- [7] C. Seberino and N. Bertram, Concise, Efficient Three-Dimensional Fast Multipole Method for Micromagnetics, *IEEE Trans. Magn.* **37**, 1078 (2001).
- [8] J. J. Miles and B. K. Middleton, A hierarchical micromagnetic model of longitudinal thin film recording media, *J. Magn. Mag. Mat.* **95**, 9-108 (1991).
- [9] K. E. Schmidt and M. A. Lee, *J. Stat. Phys.* **63**, 1223 (1991).
- [10] J. Shimada, H. Kaneko, and T. Takada, Performance of Fast Multipole Algorithms for Calculating Electrostatic Interactions in Biomacromolecular Simulations, *J. Comput. Chem.* **15**, 28-43 (1994).
- [11] A recursive formula for $D_{\mathbf{p}}(\mathbf{c})$ (Eq. (12)) has been given in M. Chalacombe, E. Schwegler, and J. Almlöf, Recurrence relations for calculation of the Cartesian multipole tensor, *Chem. Phys. Lett.* **241**, 67-72 (1995), but the recursion must be performed for each value of \mathbf{c} .
- [12] J. D. Jackson, *Classical Electrodynamics*, (Wiley, 3rd Edition, 1999), p. 195.
- [13] R. F. Soohoo, Magnetic thin films, publisher's name,date.
- [14] A. J. Newell, W. Williams, and D. J. Dunlop, A Generalization of the Demagnetizing Tensor for Nonuniform Magnetization, *J. Geophys. Res.* **98**, 9951-9555(1993).
- [15] H. Fukushima, Y. Nakatani, and N. Hayashi, "Volume Average Demagnetizing Tensor of Rectangular Prisms", *IEEE Trans. Magn.* **34**, 193-198 (1998).
- [16] D. M. Apalkov and P. B. Visscher, "Simple iterative calculation of micromagnetic kernels", *J. Appl. Phys.* **93**, 15 May 2003.
- [17] Standard Problem #2 is described at the web site <http://www.ctcms.nist.gov/~rdm/toc.html#standards>.