High-Performance Interpolation of Stellarator Magnetic Fields

Paul Probert

***Abstract*—We present a new algorithm for interpolation of static current-free magnetic fields based on the use of specialized fourth-degree vector polynomial functions (Maxwell elements) which solve the static current-free Maxwell equations. This method provides sufficient accuracy for the demands of stellarator applications and retains a great speed advantage over direct Biot–Savart calculation of the fields.**

***Index Terms*—Interpolation, magnetic confinement, numerical analysis, plasma confinement, toroidal magnetic fields.**

I. Introduction

In STELLARATOR research, it is often necessary to evaluate the magnetic field produced by a complicated set of external coils. To look for magnetic islands, evaluate particle drifts and losses, or generate magnetic coordinate systems [1], one may need exceptional accuracy in the calculation. For example, magnetic islands with a minor radius of just a few electron gyroradii can affect electron transport. To calculate field line positions to this accuracy for many orbits around the machine can require relative errors to be quite small. For example, given N orbits around a machine with a major radius of R, a magnetic field of B, and an error in the field of $\triangle B$, the field line position error is roughly $\triangle p = 2\pi RN\triangle B/B$. This is assuming a perfect field line integrator and only estimates the error due to the error in the calculated $\mathbf{B}$. If B is 1 T, the major radius is 1 m, and the electron energy is 1 keV, then to get the position error that is comparable to the electron gyroradius for 1000 orbits requires a relative field error $\triangle B/B \sim 10^{-8}$.

The most straightforward way to do this is by means of Biot–Savart integration [2] over the coil current distribution. The computational cost of this can be high, however, particularly when the coils are complicated and the discrete representation of the coils is fine grained. This leads one to consider precalculating the magnetic field on some suitable 3-D grid. One can then find an approximate field at an arbitrary field point inside the grid by interpolation. We report here a new method of interpolating the stored field and its gradient with good speed and accuracy.

Manuscript received August 24, 2010; revised November 12, 2010; accepted December 17, 2010. This work was supported by U.S. Department of Energy under Grant DE-FG02-93ER54222.

The author is with the HSX Plasma Laboratory, Electrical and Computer Engineering Department, University of Wisconsin–Madison, Madison, WI 53706-1691 USA.

Digital Object Identifier 10.1109/TPS.2011.2105890

II. Maxwell Element Method

For our purposes here, we always calculate the field gradient along with the field. Because of the Maxwell equations for a static magnetic field in a vacuum, $\nabla \cdot \mathbf{B} = 0$, and $\nabla \times \mathbf{B} = 0$. The gradient of $\mathbf{B}$ can be represented by a symmetric traceless matrix with five independent numbers. Thus, the field and its gradient at any given point can be represented by a vector Y_l, $l = 1, \ldots, 8$, comprising three components of $\mathbf{B}$ and five independent components of $\nabla\mathbf{B}$

$$\mathbf{Y}(\mathbf{p}) = \Big[B_x(\mathbf{p}), B_y(\mathbf{p}), B_z(\mathbf{p}), (\nabla\mathbf{B}(\mathbf{p}))_{xx}, (\nabla\mathbf{B}(\mathbf{p}))_{xy}, (\nabla\mathbf{B}(\mathbf{p}))_{xz}, (\nabla\mathbf{B}(\mathbf{p}))_{yy}, (\nabla\mathbf{B}(\mathbf{p}))_{yz}\Big]$$

where the general field point is $\mathbf{p} = x\hat{x} + y\hat{y} + z\hat{z}$. Here, we use the standard meaning of $\nabla\mathbf{B}_{ij} = \partial B_i/\partial p_j$. Suppose that we have these $\mathbf{Y}$ values precomputed and stored at regular grid positions. We seek a general polynomial to express the field value at some point $\mathbf{p} = (x, y, z)$. We expand about some point $\mathbf{p}_0 = (x_0, y_0, z_0)$

$$\begin{aligned}\mathbf{B} = \hat{\mathbf{x}} &\sum_{i,j,k=0}^{i+j+k\le D} a_{ijk}(x-x_0)^i(y-y_0)^j(z-z_0)^k \\ + \hat{\mathbf{y}} &\sum_{i,j,k=0}^{i+j+k\le D} b_{ijk}(x-x_0)^i(y-y_0)^j(z-z_0)^k \\ + \hat{\mathbf{z}} &\sum_{i,j,k=0}^{i+j+k\le D} c_{ijk}(x-x_0)^i(y-y_0)^j(z-z_0)^k \qquad (1)\end{aligned}$$

where D is the maximum degree of the polynomials. We must solve for the coefficients a, b, and c in terms of the stored field at surrounding grid points. In order to ensure that the grid is not too fine (the storage required goes like the cube of the inverse grid spacing) while still maintaining accuracy, we seek the highest possible degree polynomials.

The upper limit on the degree D that we can use is set by the need to solve for the coefficients a, b, and c using a limited amount of data. In our case of fourth-degree polynomials, we get a total of 105 unknown coefficients. If we wish to use $\mathbf{B}$ data at the eight nearest points, including the gradients of $\mathbf{B}$, this supplies only 64 known values to the fitting procedure. The main idea of this paper is to reduce the number of terms in these polynomials by applying Maxwell's equations to the polynomials and solving for the coefficients, reducing the number of independent coefficients from 105, in the case of fourth-degree polynomials, to 35. The technique is simple but algebraically tedious. Applying $\nabla \cdot \mathbf{B} = 0$ and $\nabla \times \mathbf{B} = 0$ to (1), the result is a system of four polynomial equations. Requiring these to hold at all x, y, and z gives sufficient information to solve for sets of the a, b, and c coefficients. In the case of fourth-degree polynomials, we obtain 35 separate sets of these coefficients which solve these equations. We refer to these 35 vector polynomial functions as "Maxwell elements," which are similar to the concept of elements in numerical partial differential equations and finite-element analysis [3]. These polynomials are also closely related to the "harmonic polynomials" for solving Laplace and wave equations [10], [11]. We group them hereinafter according to the highest powers of the coordinates in the element (each comma-separated line of coefficient assignments represents one element).

Constant elements

$$a_{000} = 1$$
$$b_{000} = 1$$
$$c_{000} = 1.$$

Linear elements

$$b_{001} = 1, \quad c_{010} = 1$$
$$a_{001} = 1, \quad c_{100} = 1$$
$$a_{010} = 1, \quad b_{100} = 1$$
$$a_{100} = 1, \quad c_{001} = -1$$
$$a_{100} = 1, \quad b_{010} = -1.$$

Quadratic elements

$$a_{011} = 1, \quad b_{101} = 1, \quad c_{110} = 1$$
$$a_{020} = 1, \quad a_{200} = -1, \quad b_{110} = 2$$
$$a_{002} = 1, \quad a_{200} = -1, \quad c_{101} = 2$$
$$b_{002} = 1, \quad b_{020} = -1, \quad c_{011} = 2$$
$$a_{110} = 2, \quad b_{200} = 1, \quad b_{020} = -1$$
$$b_{011} = 2, \quad c_{020} = 1, \quad c_{002} = -1$$
$$a_{101} = 2, \quad c_{200} = 1, \quad c_{002} = -1.$$

Cubic elements

$$a_{111}=6, \quad b_{021}=-3, \quad b_{201}=3, \quad c_{030}=-1, \quad c_{210}=3$$
$$a_{111}=6, \quad b_{003}=-1, \quad b_{201}=3, \quad c_{210}=3, \quad c_{012}=-3$$
$$a_{210}=3, \quad a_{012}=-3, \quad b_{300}=1, \quad b_{102}=-3, \quad c_{111}=-6$$
$$a_{030}=1, \quad a_{012}=-3, \quad b_{120}=3, \quad b_{102}=-3, \quad c_{111}=-6$$
$$a_{300}=2, \quad a_{120}=-3, \quad a_{102}=-3, \quad b_{210}=-3, \quad b_{012}=3, \quad c_{201}=-3, \quad c_{021}=3$$
$$a_{102}=3, \quad a_{120}=-3, \quad b_{030}=2, \quad b_{012}=-3, \quad b_{210}=-3, \quad c_{201}=3, \quad c_{021}=-3$$
$$a_{120}=3, \quad a_{102}=-3, \quad b_{210}=3, \quad b_{012}=-3, \quad c_{003}=2, \quad c_{201}=-3, \quad c_{021}=-3$$
$$a_{003}=1, \quad a_{021}=-3, \quad b_{111}=-6, \quad c_{102}=3, \quad c_{120}=-3$$
$$a_{201}=-3, \quad a_{021}=3, \quad b_{111}=6, \quad c_{300}=-1, \quad c_{120}=3.$$

Quartic elements

$$a_{400}=1, \quad a_{202}=-3, \quad a_{220}=-3, \quad a_{022}=3, \quad b_{310}=-2, \quad b_{112}=6, \quad c_{301}=-2, \quad c_{121}=6$$
$$a_{040}=1, \quad a_{202}=3, \quad a_{220}=-3, \quad a_{022}=-3, \quad b_{310}=-2, \quad b_{130}=4, \quad b_{112}=-6, \quad c_{301}=2, \quad c_{121}=-6$$
$$a_{004}=1, \quad a_{202}=-3, \quad a_{220}=3, \quad a_{022}=-3, \quad b_{310}=2, \quad b_{112}=-6, \quad c_{301}=-2, \quad c_{121}=-6, \quad c_{103}=4$$
$$a_{112}=-6, \quad a_{310}=4, \quad a_{130}=-2, \quad b_{400}=1, \quad b_{202}=-3, \quad b_{022}=3, \quad b_{220}=-3, \quad c_{211}=-6, \quad c_{031}=2$$
$$a_{112}=6, \quad a_{130}=-2, \quad b_{040}=1, \quad b_{202}=3, \quad b_{022}=-3, \quad b_{220}=-3, \quad c_{211}=6, \quad c_{031}=-2$$
$$a_{112}=-6, \quad a_{130}=2, \quad b_{004}=1, \quad b_{202}=-3, \quad b_{022}=-3, \quad b_{220}=3, \quad c_{013}=4, \quad c_{211}=-6, \quad c_{031}=-2$$
$$a_{103}=-2, \quad a_{121}=-6, \quad a_{301}=4, \quad b_{013}=2, \quad b_{211}=-6, \quad c_{400}=1, \quad c_{220}=-3, \quad c_{202}=-3, \quad c_{022}=3$$
$$a_{103}=2, \quad a_{121}=-6, \quad b_{013}=-2, \quad b_{031}=4, \quad b_{211}=-6, \quad c_{040}=1, \quad c_{220}=-3, \quad c_{202}=3, \quad c_{022}=-3$$
$$a_{103}=-2, \quad a_{121}=6, \quad b_{013}=-2, \quad b_{211}=6, \quad c_{004}=1, \quad c_{220}=3, \quad c_{202}=-3, \quad c_{022}=-3$$
$$a_{211}=3, \quad a_{013}=-1, \quad b_{103}=-1, \quad b_{301}=1, \quad c_{112}=-3, \quad c_{310}=1$$
$$a_{031}=1, \quad a_{013}=-1, \quad b_{121}=3, \quad b_{103}=-1, \quad c_{112}=-3, \quad c_{130}=1.$$

Note that, by "element," we refer to a vector function which is a polynomial in x, y, z as in (1). By substituting a_{ijk}, b_{ijk}, and c_{ijk} from any of these 35 lines into (1), one obtains an element. We can compute the gradients of each of these elements analytically and create the quantities Y_l designating element m as $Y_l^m(\mathbf{p})$, $m = 1, \ldots, 35$, $l = 1, \ldots, 8$. When interpolating at some field point, we use the stored data at the eight vertices $\mathbf{p_n}$ of the cube surrounding the field point $\mathbf{p}$. For the expansion point $\mathbf{p}_0$ in (1), we use the average position of the eight vertices. The grid need not be regular, and the points $\mathbf{p_n}$ need not lie exactly on the vertices of a cube. Indeed, in our implementation, the grid points lie on a regular grid in cylindrical coordinate space, even though the field data are stored as components in the Cartesian x, y, z space. The precomputed grid data are generated by running a Biot–Savart routine which uses a model of the stellarator coils as a sequence of straight current filaments [2]. Let us represent the data $Y_l(\mathbf{p})$ at our field point $\mathbf{p}$ as a sum over the $M = 35$ elements Y_l^m as $Y_l(\mathbf{p}) = \sum_{m=1}^{M} f_m Y_l^m(\mathbf{p})$.

We find the coefficients f_m by solving a standard linear least squares problem [4]. Specifically, we let our error χ be given by

$$\chi^2 = \sum_{l=1}^{L}\sum_{n=1}^{N}\left[Y_l(\mathbf{p}_n) - \sum_{m=1}^{M} f_m Y_l^m(\mathbf{p}_n)\right]^2.$$

To find the least squares solution, we differentiate this with respect to the M coefficients f_m and set the result to zero, obtaining a matrix problem $Af = g$, where

$$A_{uv} = \sum_{l=1}^{L}\sum_{n=1}^{N} Y_l^u(\mathbf{p}_n)Y_l^v(\mathbf{p}_n)$$

$$g_u = \sum_{l=1}^{L}\sum_{n=1}^{N} Y_l(\mathbf{p}_n)Y_l^u(\mathbf{p}_n).$$

In our routine, this 35 by 35 matrix problem is solved for each interpolation, yielding the f_m coefficients valid for the grid cell containing the field point $\mathbf{p}$, from which the extrapolated field and its gradient can be quickly calculated. The size of this matrix problem is small enough such that modern desktop processors, using optimized linear algebra routines such as Atlas [5], can solve the problem entirely in cache memory. Indeed, we have found by profiling our code that the time spent in solving this matrix problem is negligibly small. Most of the time seems to be spent in calculating the values of the elements to fill the A_{uv} matrix. As a speedup, we save the coefficients in a buffer in case the next call to the interpolation routine requests the field at a point in the same cell. We note that, as a further speedup, one could presolve for the f_m coefficients for all the cells in the grid and store all these sets of coefficients. The resulting table would be 35/8 times as large, but the routine would then be very fast. The code for our routine was written in the open-source language Octave [6]. Many of the subroutines, such as those for evaluating the elements Y_l^m and for generating the matrix A_{uv} and the vector g_u, were written as extensions in the C++ language for speed. We can also run these programs using the commercial software Matlab [7], which is a closed-source language that is mostly compatible with Octave. The extension routines for Matlab are written in C.

To test the accuracy of the interpolation, we constructed a test coil consisting of a square loop that is 0.5 m on a side, and our field point is located about 2 cm from the corner of the loop, where interpolation is difficult due to strong variations in the field there. Grids were constructed around this point using many different grid spacings, and the error in the interpolated field was evaluated. The results of this test are shown in Fig. 1. Also shown, for comparison, are interpolations using Octave's built-in interp3 routine, using both a linear interpolation and a cubic spline interpolation. We see that the Maxwell element interpolation gives a significant advantage in accuracy and in the dependence of the error on the grid spacing. Another advantage of this method is that the results are explicitly curl and divergence free (within a roundoff error), which can be an advantage in some calculations.

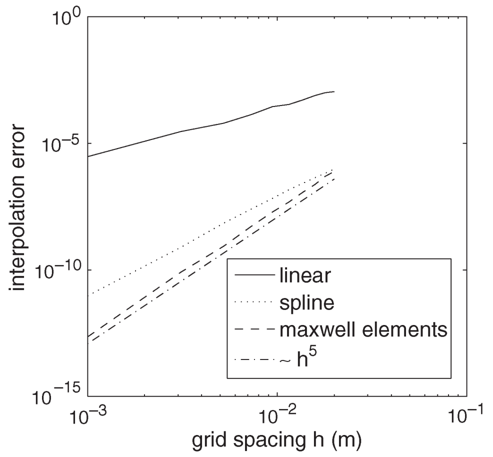


Fig. 1. Interpolation errors versus grid spacing, as the magnitude of the vector difference between the interpolated field and the exact result, divided by the magnitude of the exact field. The test problem is described in the text. Also shown is a line proportional to h^5. This verifies that the Maxwell element method used here, which employs fourth-order polynomials, has an error that scales properly with the grid spacing h.

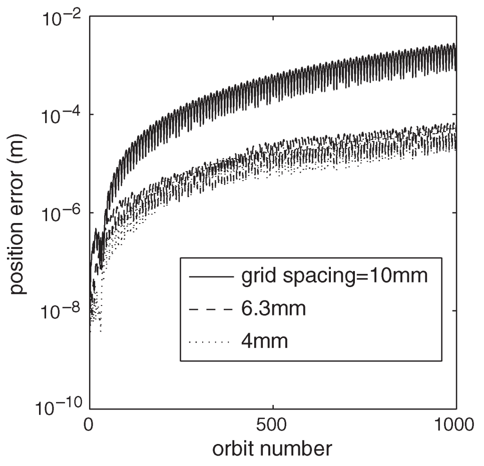


Fig. 2. Results of field line tracing for many orbits around the HSX stellarator. The error is the magnitude of the difference of the field line position at a fixed toroidal angle relative to that calculated from Biot–Savart integration. The line tracing integrations were performed using Octave's lsode package [9] with relative and absolute tolerances of 10^{-10}.

We have developed this code primarily to conduct research on the HSX stellarator [8]. This machine has a major radius of roughly 1.2 m and a minor radius of 0.15 m. We usually operate the machine in a configuration with stellarator symmetry, which allows us to generate a grid which only covers one-half field period. With a 1-cm grid spacing, the stored grid is roughly 14 MB in size. A typical desktop machine can fill such a table in a few hours. The interpolation error scales as the fifth power of the grid spacing so that, by reducing the grid spacing from 1 to 0.63 cm, we gain about a factor of ten in accuracy. Such a reduction in the grid spacing, however, leads to grid tables that are about four times as large. To find the optimum grid spacing, we generated tables at three different grid spacings and then ran a field line tracing program for 1000 orbits around the machine. To compare, we ran the same tracing using the Biot–Savart representation of the coils directly. The resulting position errors are shown in Fig. 2. We see that both 6.3- and 4-mm tables give similar results. We interpret this to mean that other errors, such as roundoff or integration errors, dominate over interpolation errors with these finer tables. We also note that the errors in position for the 6.3- and 4-mm tables, even after 1000 orbits, are less than the electron gyroradius in our machine ($B = 1\ \text{T}, T_e \sim 1000\ \text{eV} \rightarrow r_{\text{ge}} = 10^{-4}\ \text{m}$). There is little or no slowdown when using tables with the finer grids. Also, since the gradients of the Maxwell elements are known from analytical differentiation, there is almost no speed penalty when requesting the gradient of the field during an interpolation.

We note that the time required for these integrations is roughly a factor of 17 faster using our interpolation routine compared to the Biot–Savart field evaluation. This is with the coil representation consisting of 48 coils, each with 840 straight current filaments representing the turns in the coil. Using a more complicated and accurate coil representation would slow down the Biot–Savart method and lengthen the time to fill the tables initially, but it would have no effect on the interpolation speed.

There exist related methods for interpolating vector functions on a grid. By representing the magnetic field as the gradient of a scalar [12] or as the curl of a vector potential (in [13], tricubic splines are used to interpolate the vector potential, and the field is then found by analytically differentiating those spline formulas), one can do an essentially similar task as what we do here. It is possible that, by storing only the minimal components of the vector potential on the grid (by employing gauge freedom), one could cut down on the memory requirements. However, due to differentiation, the $\mathbf{B}$ field will be represented by a polynomial of one degree less than the vector potential, forcing the polynomial fitting of the vector potential on the grid to a higher degree in order to obtain the same precision. We were also motivated to stick with a grid representation of $\mathbf{B}$ due to the existence of Biot–Savart codes for generating $\mathbf{B}$ and its gradient for our machine.

An important restriction on this method is that it is only applicable to regions where the local current density is zero. Thus, it is useful to calculate the magnetic field due to coil sets, but the contribution from plasma currents must be done separately by other means. Usually, in the course of a computation, the plasma current distribution is a variable quantity, not known with great precision, and its magnetic field is a small fraction of that due to the coils, so such a restriction is not severe. To remove the current-free restriction on our method would essentially destroy it by removing our ability to restrict our fitting parameters to a manageable set of only 35 numbers.

III. Conclusion

We have developed an interpolation scheme for 3-D vector magnetic fields from external coils. This method exploits the mathematical properties of the field (that it solves the static current-free Maxwell equations for the magnetic field) to efficiently fit fourth-degree polynomials to the known data. The scheme results in a code which is sufficiently accurate for most stellarator physics calculations while providing significant speed improvement over Biot–Savart calculations.

References

[1] X. Bonnin, A. Mutzke, C. Nuhrenberg, J. Nuhrenberg, and R. Schneider, "Calculation of magnetic coordinates for stellarator fields including islands and the scrape-off layer," *Nucl. Fusion*, vol. 45, no. 1, pp. 22–29, Jan. 2005.

[2] J. D. Hanson and S. P. Hirshman, "Compact expressions for the Biot–Savart fields of a filamentary segment," *Phys. Plasmas*, vol. 9, no. 10, pp. 4410–4412, Oct. 2002.

[3] G. Strang and G. Fix, *An Analysis of the Finite Element Method*. Wellesley, MA: Wellesly-Cambridge, 1973.

[4] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in C*, 2nd ed. New York: Cambridge Univ. Press, 1992.

[5] R. C. Whaley and J. J. Dongarra, *Conference on High Performance Networking and Computing*. Los Alamitos, CA: IEEE Comput. Soc. Press, 1998, pp. 1–27.

[6] J. W. Eaton, D. Bateman, and S. Hauberg, *GNU Octave Manual Version 3*. Bristol, U.K.: Network Theory Ltd., 2009.

[7] The MathWorks, Inc., *MATLAB, The Language of Technical Computing*, 6th ed. Natick, MA: MathWorks, Inc., 2002.

[8] F. S. B. Anderson, A. F. Almagri, D. T. Anderson, P. G. Mathews, J. N. Talmadge, and J. L. Shohet, "Helically symmetric experiment, (HSX) goals design and status," *Fusion Technol.*, vol. 27, p. 273, 1995.

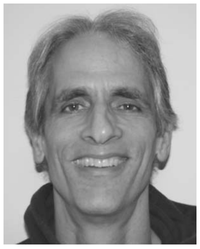
[9] K. Radhakrishnan and A. C. Hindmarsh, "Description and use of LSODE, the Livermore solver for ordinary differential equations," Lawrence Livermore Nat. Lab., Livermore, CA, Tech. Rep. UCRL-ID-113855, Dec. 1993.

[10] E. P. Miles, Jr. and E. Williams, "Basic sets of polynomials for the iterated laplace and wave equations," *Duke Math. J.*, vol. 26, no. 1, p. 3540, 1959.

[11] E. P. Miles, Jr. and E. Williams, "A note on basic sets of homogeneous harmonic polynomials," *Proc. Amer. Math. Soc.*, vol. 6, no. 2, p. 191, 1955.

[12] H. Wind, "Processing magnetic field data," *J. Comput. Phys.*, vol. 2, no. 3, pp. 274–278, Feb. 1968.

[13] J. M. Finn and L. Chacon, "Volume preserving integrators for solenoidal fields on a grid," *Phys. Plasmas*, vol. 12, no. 5, p. 054 503, May 2005.



Paul Probert received the Ph.D. degree from the University of Wisconsin, Madison, in 1985.

He is currently an Associate Scientist with the HSX Plasma Laboratory and the Pegasus Toroidal Experiment, University of Wisconsin. He maintains gyrotrons, develops instrumentation, writes data acquisition software, and works on RF heating. He worked on the Phaedrus-B tandem mirror experiment and designed and built the poloidal field systems for the Phaedrus-T tokamak. He was also heavily involved with the RF systems on that experiment.