Iterative Solution of Hermite Boundary Integral Equations

Abstract. Iterative methods for solving linear equations are especially useful for systems involving a large number of variables, and for matrices that are sparse (a sparse matrix is one in which most entries are zero). Such systems arise when a partial differential equation is solved by means of a cubic Hermite approximation of the corresponding boundary integral equation. The Hermite approximation relies on knowledge of the function derivatives (gradient), and these gradient equations are sparse. By employing a sparse representation of these matrices, e.g., compressed row storage, the memory and the computational cost of working with the matrix can be significantly reduced. The goal of this work has been to exploit the sparsity present in the Hermite matrices, thereby reducing the cost of the matrix vector multiplication required by the iterative solver. In this project, a compressed row storage (CRS) format has been implemented; the primary task in constructing the CRS was the determination of where the non-zero entries would be located. Timing results, comparing the computational costs of the sparse and non-sparse algorithms will be carried out as future research. This work was motivated by applications of the Hermite approximation for moving boundary problems. For these simulations, a boundary integral solution is obtained at every time step, and thus it is critical to reduce the time required by the linear solver. Other possible applications of this work include elasticity, geomechanics, electromagnetics, acoustics, low- Reynolds number hydrodynamics, biomechanics, and off- shore structures.

In computational mathematics, iterative methods are derived from an initial guess in the attempt to find successive approximations. In former work such as "A Cubic Hermite Boundary Integral Approximation", it was shown that a differentiable (cubic Hermite) boundary integral interpolation should be effective for the solution of moving boundary problems. Moving boundary problems in mainly used in science and engineering. The goal is to track the time step of a "moving boundary", e.g, modeling the breaking of a water wave, the dripping of a faucet, or even measuring the wavelength of a sound wave. The iterative Hermite algorithm used in the project deals with a $3N$ by $3N$ system of linear equations. It is known to be linear due to the form of A x = b. A represents the matrix of the general form: a_{ij} where i represents the rows and j represents the columns. The matrix-vector multiplication can then be written as:

$$\begin{pmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

For each a_{ij} in this submatrix (a matrix inside matrix) is N by N. The vector x_1 represents the unknown boundary values of potential (the work required to move a unit from a reference point to a designated point) or flux (the rate of flow through a given surface). The first block row of equations if the discretized form (defined for a finite or countable set of values) of the exterior limit boundary integral equation for the Laplace equation. The vectors x_2 and x_3 represents the gradient (vector having x and y components that are the partial derivatives of a function with respect to the variables). The second and third rows are the difference of the interior and exterior limits. In addition, the gradient equations of the interpolation (estimate of a value of a function or series between two values) algorithms are sparse. Since the gradient equations only involve singular integrals over local fields/ boundedness, A_{2j} and A_{3j} are sparse matrices. Therefore, the diagonal blocks A_{22} and A_{33} are symmetric positive definite, then $A_{22} = A_{33} = 0$. First, assume an initial guess of zero for x_2 and x_3 and solve for x_1. Lastly, update the values for x_2 and x_3 by solving for them.

Step 1:

$$A_{11}x_1 = b_1 - A_{12}x_2 - A_{13}x_3$$

Step 2:

$$A_{22}x_2 = b_2 - A_{21}x_1 - A_{23}x_3$$
$$A_{33}x_3 = b_3 - A_{31}x_1 - A_{32}x_2$$

A_{23} and A_{32} are considered the weakly (carefree) couple. Weakly couple is defined as a couple such that their greatest common divisor is one and at least one of the couple is prime. The weakly couple will not specifically affect the results so they are excluded from the sparse transformations.

In the current project, the primary task was to manipulate the advantage of the structure of a sparse matrix when determining the location of the non- zero elements. The concept of sparse data is to compress the row, column, or diagonal storage, which will significantly reduce memory usage and computing time. However, the sparse matrix storing requires a scheme for knowing how to construct the matrix. Employing the compressed row storage (CRS) format required two main steps. The first step consisted of constructing three vectors. The CRS format is free information for the general public that explains the definition and the purpose for these vectors. The value vector is simply the non- zero matrix entries. The row- pointer vector tells the row location of the non-zero(s). The column- pointer vector is the non- zero matrix entries from the column. The three vectors must be constructed in the Fortran subroutines for the remaining four sparse

matrices: A_{21}, A_{22}, A_{31}, and A_{33}. The next step is to perform a new matrix multiplication using the three vectors that were created. The bulk of the work included debugging, as usual with computer programming languages. One main task consisted of checking the results. The values of the potential, flux, and the gradient were supposed to remain the same as the non- sparse algorithms. Only a slight difference in the results was expected.

$$(Ax)_i = \sum_{j=1}^{N} a_{ij} x_j \leftrightarrow \sum_{k=k_1}^{k_2} V(k) x_{col(k)}$$

As one can see from the two matrix- vector multiplications, the summation has been reduced from *N* amount of times to a known value such as k_2.

In conclusion, timing results and comparing the computational costs of the sparse and non- sparse algorithms will be carried out. For example, if there were two *N* by *N* matrices, the non- sparse algorithms would take $2N^2$ operations per iteration. The sparse algorithms would produce a total of $6N$ operations per iteration. $2N^2$ versus $6N$ is an obvious dramatic difference. Another advantage of using sparse is the capability to solve larger problems using a desktop or a personal computer (PC). The value of *N* depends strictly upon the problem that needs to be solved. If the value of *N* is roughly estimated at

15,000, then a supercomputer might be necessary. The timing results will vary depending on the value of N, as well as the quality of the machine. Naturally, a supercomputer will run phenomenally faster than an average PC. For the application of moving boundary problems, higher accuracy and faster computing are crucially essential.

Applications of boundary element method (BEM) encompass a variety of fields. Elasticity, structural mechanics, acoustics, and cathodic protection/ offshore structures are only a few examples. Moreover, elasticity refers to the manner in which solids respond to stress, or the application of the force over a given unit area. Structural mechanics is defined as the computation of internal forces and stresses within particular structures. Acoustics is the study of the production, transmission, and reception of sound. Cathodic protection is a technique to control or prevent stress corrosion of a metal surface. It's often used in offshore structures such as rigs, drilling, and production platforms. The definitions can verify the extreme importance of time evolution. Implementation of sparse representation of the Hermite matrix will permit solving BIEs with high accuracy, without highly refined grids. The future research is targeted toward extending the result to three dimensions (3D), which will result in a more noticeable decrease in the computational costs of sparse versus non- sparse algorithms.