

Group Project: An FMM toolbox for MATLAB

CMSC 878R/AMSC 698 R/ MAIT 627, Spring 2008

Ramani Duraiswami and Nail Gumerov

April 10, 2008

Abstract

The goal of this project is to provide an FMM toolbox (publicly available) integrated in to Matlab. The goal is correctness and a full understanding of concepts. Five sub-projects are identified, and a schedule for the completion is given.

1 Introduction

The goal of this joint project is to create a collection of FMM-related functions that are tied in to MATLAB. The tasks are divided in to five pieces, and each task is assigned to a main person (responsible for coding), a secondary person (responsible for documentation/testing). Each student will have the role of a main person on one part and that of the secondary person on another part. Finally at the end of the project, each student must build one example case using the overall code. This project will involve substantial work, and you should plan on spending several hours on it weekly.

The focus is on correctness and concepts, and prototyping. Some functions might also be converted in to MEX form (with the native code written in Fortran or C).

A piece of advice is to understand the calling of functions in Matlab, and how you can have variable arguments, variable function names, etc. Also since Matlab is organized column major as far as matrix storage is concerned, your algorithms should respect this.

You can share your comments, code and documentation via the following wiki page for the project https://wiki.cs.umd.edu/Matlab_FMM. Create accounts for yourselves and post a test message. Weekly reports also have to be posted here (and read by all of you).

2 Project Pieces

2.1 Data Structures and the Overall FMM Algorithm

This follows closely from the homework on data structures, where you were asked to build a set of functions. You will build the same functions here as well. In the homework, the functions only had to work in 1-D. Here you have to make sure that the functions work in 1, 2 and 3-D.

In addition you will build the overall FMM algorithm (upward pass, downward pass and final summation; skipping empty boxes). One piece of advice, is to keep the piece with the direct computations separate, since there are some opportunities for separate speed up of this piece.

2.2 Translation Operators

A package of S expansions, R expansions, $S|S$, $S|R$ and $R|R$ translations for various commonly used “FMM”-able functions in 1, 2 and 3-D will be built.

This will be seeded by the 1-D Cauchy kernel we have been doing homework on.

Later we will add the following

- 2D Laplace
- 3D Laplace
- 2D Helmholtz
- Faster versions of the 2D/3D Laplace.

The functions and expansions must be all tested and demonstrated.

2.3 Adaptive Versions of the FMM

Adaptive versions of the FMM are discussed in two papers. Cheng et al. in the Journal of Computational Physics, and in a chapter in our book. There are also two Java implementations of adaptive FMM in 2D, as previous course projects. Running the adaptive version also changes the running of the overall FMM algorithm since the downward pass can be substantially modified.

2.4 Using the FMM with Matlab Routines

The FMM is mainly used in application problems, including the following

1. Time dependent inter particle computations (e.g. in stellar dynamics, molecular dynamics, vortex element methods)
2. Dense linear system solutions (e.g., in function fitting, boundary element methods, equivalent source methods etc.). For such problems source and evaluation points are the same.
3. Eigen value problems (e.g., stability computations, acoustics, etc.)

2.4.1 Existing Matlab Routines and the FMM

Thus it would be useful to use the rich Matlab environment and implement some test applications. Many Matlab routines allow a user defined routine for matrix vector products. These include the following

- **pcg**: Preconditioned conjugate gradient
- **gmres**: Preconditioned generalized minimum residual solution of a linear system (usually non-symmetric).
- **eigs**: Determining a few eigenvalues and eigenvectors

There are several others that could also be called.

In addition, while the routine **condest** and **normest** do not provide a matrix-vector function call link, they also rely on mat-vecs.

The goal is to use the FMM with these.

2.4.2 Time Integrators

In addition we can compute particle dynamics via the FMM using some standard integrators. Standard integrators will have the form

$$\frac{d}{dt} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} f_1(x_1, \dots, x_n, t) \\ f_2(x_1, \dots, x_n, t) \\ \vdots \\ f_n(x_1, \dots, x_n, t) \end{pmatrix}$$

Incorporate the FMM in to a simple Euler computation of N-body calculations in 2D, and then extend it to higher order integrators.

2.4.3 Preconditioning

Preconditioning is important in many computations for fast convergence. A particularly simple preconditioner may be appropriate to use with the FMM, and is described here. Consider a matrix Φ . Its inverse can be computed by solving N linear systems with the right hand sides of these systems being formed by the columns of the identity matrix. If each of the solution vectors are stacked as the columns of a matrix, this matrix is the inverse. Now, as discussed in class the best preconditioner is the inverse, but it is too expensive to compute. The best practical preconditioners are approximate inverses that are inexpensive to compute.

Each row of the matrix in a FMM matrix vector product can be associated with a single evaluation point (and for a linear system solution, that source point). The idea of the preconditioner is to take a small group of q points (e.g., the points we do direct evaluation with, plus some close by points), and solve a small linear system, at $O(q^3)$ cost. We do this for all N points, and create an approximate inverse matrix by extending the q vector to a N vector by adding zeroes at the locations not considered. Stacking these columns, we have in principle an approximate preconditioner, which requires $O(Nq)$ storage. To apply this preconditioner it takes $O(q^2N)$ operations.

We will try out this preconditioner, and we can use the condition estimator to check how much the condition number improves..

2.5 Use of the Improved Fast Gauss Transform

The improved fast Gauss transform code is available on sourceforge and computes the gaussian sum. The goal is to build a Matlab interface to this package.

3 Schedule

Selection of pieces: April 10

General Matlab Framework: All: April 15

Implemented 1-D Framework: All: April 17

Familiarization with Matlab, Wiki: All: April 17

First Report on