1993

NASA/ASEE SUMMER FACULTY FELLOWSHIP PROGRAM

MARSHALL SPACE FLIGHT CENTER
THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

FINITE ELEMENT BASED ELECTRIC MOTOR DESIGN OPTIMIZATION

Prepared by:	C. Warren Campbell, Ph.D., P. E.
Academic Rank:	Associate Professor
Institution and Department:	The University of Alabama in Huntsville Department of Civil and Environmental Engineering
MSFC Colleague(s):	Charles S. Cornelius Rae Ann Weir
NASA/MSFC:	
Laboratory:	Propulsion Lab
Division:	Component Development Division
Branch:	Control Mechanisms and Propellant Delivery Branch

I. INTRODUCTION

The purpose of this effort was to develop a finite element code for the analysis and design of permanent magnet electric motors. These motors would drive electromechanical actuators in advanced rocket engines. The actuators would control fuel valves and thrust vector control systems. Refurbishing the hydraulic systems of the Space Shuttle after each flight is costly and time consuming. Electromechanical actuators could replace hydraulics, improve system reliability, and reduce down time.

The organization of the code is shown in Figure 1. The motor preprocessor is a routine that does the following:

1) Receives data on the motor geometry, materials, windings, and currents
2) Generates the meshes and elements for the motor for different rotor positions
3) Renumbers the nodes for minimal storage using the minimum degree ordering algorithm
4) Dynamically allocates storage for coefficient arrays for the finite element analysis

The finite element model calculates the magnetic vector potential and stores the results in a file that can be accessed by the postprocessor.

The postprocessor will do the following:

1) Calculate flux densities and field intensities
2) Calculate torques and back emfs for the motor
3) Plot the results

The optimizer will take torques and information from the postprocessor and calculate a general objective function with internal penalty function constraints. Constraints could include magnitude of current densities, motor weight and volume, and cogging torque. Based on previous values of the objective function, the optimizer will select motor geometry for the next iteration. Optimization will continue until the motor design is optimized.

The optimization will begin with an initial motor design and will proceed toward an improved design. Care must be taken in the design of the mesh. Sometimes in finite element structural optimization, a mesh is generated which gives an accurate solution to the initial design, but as optimization proceeds, the mesh becomes too coarse for an accurate solution. Then the "optimized" design is invalid.

Clearly, the code will be very long running. Consider using

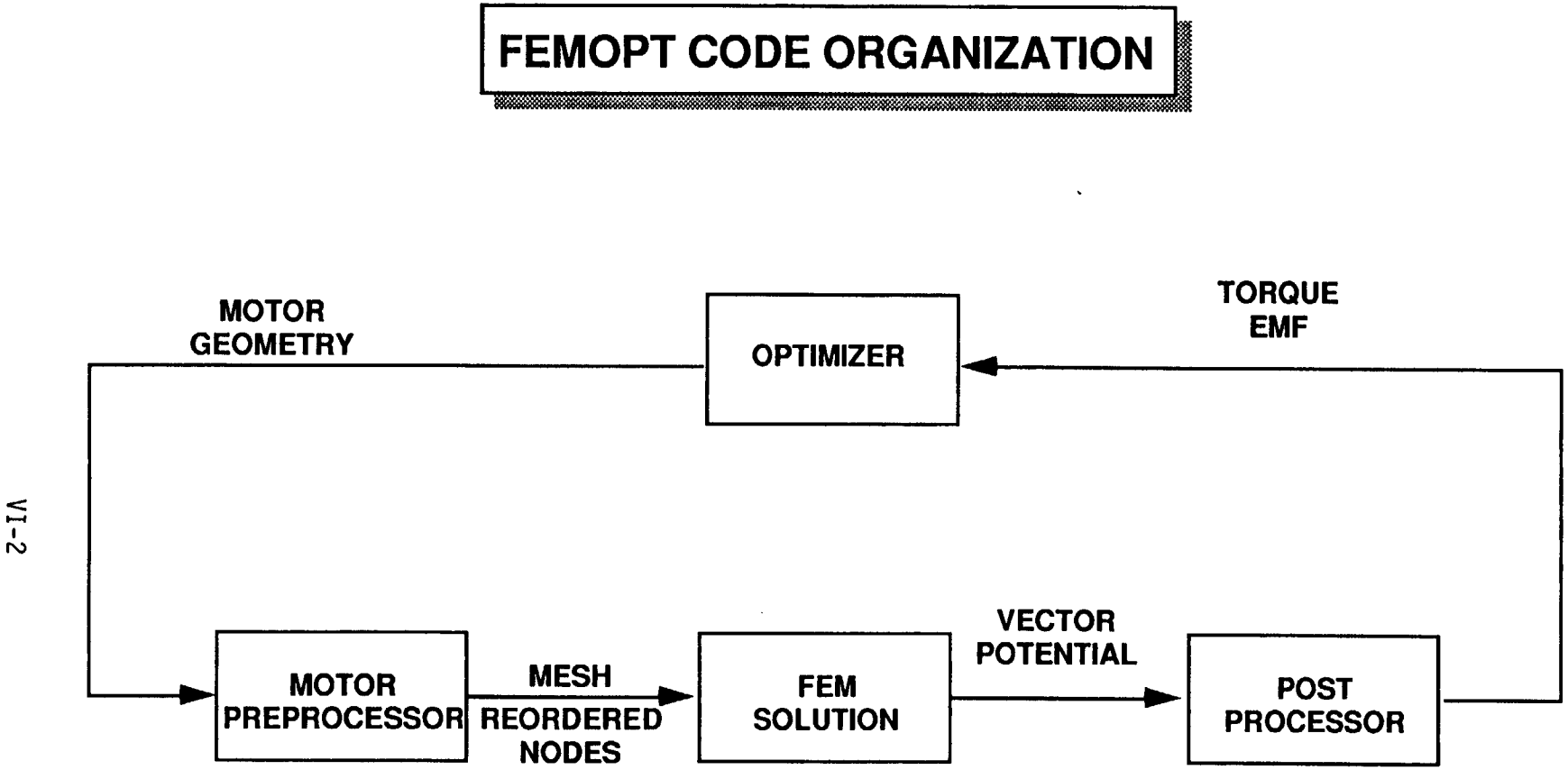

Figure 1. FEMOPT Code Structure

cogging as a constraint. For each value of the objective function the finite element code must find several solutions for different positions of the rotor.

The finite element code developed in this effort was based on the models in Silvester and Ferrari (3). The sparse matrix algorithms were taken from George and Liu (1). The optimizer will be an adaptation of code available from Numerical Recipes in C by Press, et al.(2).

II. APPROACH

The objective of this effort was to develop a finite element code with optimization that could run on a 386- or 486-class machine with up to 15,000 nodes in a two-dimensional problem. Since motors are very long compared to airgap widths and since we will not use rotor or stator skewing of magnets or teeth, the problem can be assumed to be two-dimensional. Also, these goals should be achievable without making users buy thousands of dollars of software.

Because of the ambitious goals for this project, as many of the routines as possible were based on existing code. At the beginning, I did not realize that the code in Silvester and Ferrari (3) was learning code in which coefficient arrays were dimensioned to the maximum number of nodes, that is A(maxnod, maxnod). For a 15,000 node problem (the goal for this effort), the coefficient array alone would require 15,000 by 15,000 = 225 Megawords of storage! For 4-byte words, this is a gigabyte of storage. Clearly, sparse matrix methods are required.

The need for sparse matrix methods significantly slowed the progress of the effort. Even though George and Liu is an excellent reference for solutions of finite element problems and though it has Fortran subroutines in the text, progress was extremely slow. This is because the routines in the text are spaghetti code that are extremely hard to debug and understand. The code uses variables that perform several functions and have values that change in mysterious ways at different places in the program. For these reasons, direct application of the routines would make the code difficult to understand, debug, and maintain. For these reasons, algorithms presented in George and Liu were used to write new code that was understandable, structured, and maintainable.

Borland C and C++ was chosen as the development language for many good reasons. The Borland package is inexpensive(~$300), well documented, and well written. It permits tracing line by line through the code viewing values of any variable at any point. It also allows the setting of breakpoints. The code can be executed to the breakpoints where

each variable of interest can be examined. This capability minimizes debugging effort. C was chosen because of its power. Desirable features include dynamic memory allocation, ability to implement data structures easily while writing readable code, and accessibility of computer graphics capabilities. Dynamic memory allocation means that large arrays can be created as needed, used, and then the memory deallocated for other uses. In C this is done cleanly without impact to any of the desirable features of the code. The same thing can be done in Fortran using equivalence statements, but the process can cause unexpected and untraceable errors in the code.

A strategy was found to be very useful for code development. The first step was to take simple test problems and use Mathcad (a mathematical spreadsheet easy to use and understand) to calculate values of the variables at every point in the execution of a program. With the line-by-line tracing ability of Borland C, values of the variables in the code and those calculated with Mathcad could be compared.

I also adapted an array dynamic allocation strategy from Press, et al. (3). C normally dimensions arrays from 0 to n - 1, where n is the array dimension. By the Numerical Recipes approach, arrays can be allocated from nlow to nhigh where nlow and nhigh are any values with nhigh > nlow. This is very useful in translating Fortran code with arrays dimensioned from 1 to n.

III. SUMMARY

In the first year of this task, work was done on the preprocessor and on the finite element solver. Next year the goal will be to add a nonlinear equation solver, a motor preprocessor, post processor, and optimizer.

IV. ACKNOWLEDGEMENT

Thanks are due to Charlie Cornelius and Rae Ann Weir whose support and encouragement were invaluable.

V. REFERENCES

1. George, Alan, and Liu, Joseph W., Computer Solution of Large Sparse Positive Definite Systems, Prentice-Hall, Englewood Cliffs, NJ, 1981.

2. Press, William H., et al., Numerical Recipes in C, Cambridge University Press, New York, 1990.

3. Silvester, P. P., and Ferrari, R. L., Finite Elements for Electrical Engineers, 2nd Edition, Cambridge University Press, New York, 1990.