A Hybrid Generalized Minimal Residual (GMRES) Algorithm for Solving FEM Systems Generated by the High Frequency Structure Simulator (HFSS) Code

Youssry Botros
John L. Volakis

Hewlett-Packard Company
1400 Fountaingrove Pkwy
Santa Rosa, CA

PROJECT INFORMATION

PROJECT TITLE: Algorithms for Phased Array Simulations

REPORT TITLE: A Hybrid Generalized Minimal Residual (GMRES) Algorithm for Solving FEM Systems Generated by the High Frequency Structure Simulator (HFSS) Code

U-M REPORT No.: 375458-1-T

CONTRACT
START DATE: July 1997
END DATE: May 1999

DATE: October 29, 1997

SPONSOR: David Wilson
Hewlett-Packard EEsof Division
M/S 2US-H
1400 Fountaingrove Pkwy
Santa Rosa, CA 95403-1799

SPONSOR
GRANT/CONTRACT No.: Round Table Agreement Dated 6/01/97

U-M PRINCIPAL
INVESTIGATOR: John L. Volakis
EECS Dept.
University of Michigan
1301 Beal Ave
Ann Arbor, MI 48109-2122
Phone: (313) 764-0500 FAX: (313) 747-2106
volakis@umich.edu
http://www-personal.engin.umich.edu/~volakis/

CONTRIBUTORS
TO THIS REPORT: Youssry Botros and John Volakis

A Hybrid Generalized Minimal Residual (GMRES) Algorithm for Solving FEM Systems Generated by the High Frequency Structure Simulator (HFSS) Code

Youssry Y. Botros and John L. Volakis

Radiation Laboratory

Department of Electrical Engineering

and Computer Science,

The University of Michigan

Ann Arbor, MI 48109-2212

Email ybotros@engin.umich.edu

Abstract

In this work, we developed a robust, efficient and reliable hybrid iterative solver for solving finite element systems modeled by the High Frequency Structure Simulator (HFSS) software. There is a need for such solvers when large and complicated geometries are modeled using HFSS. In this case, the number of equations (unknowns) become significantly large and conventional direct solvers can not be utilized because of their unaffordable memory and CPU requirements. In the developed solver, two versions of the Generalized Minimal Residual (GMRES) family were employed, namely, the GMRES and the Flexible GMRES (FGMRES). The GMRES iterative solver was chosen because of its robustness, efficiency and the complete avoidance of breakdown or nearly breakdown situations. Due to the unknown nature, size and condition of the FEM systems generated by HFSS, the code should be automatically adapted to the FEM system to avoid any memory or CPU waste. To speed up convergence, preconditioning of the FEM matrix is needed. Preconditioners vary in complexity from the simple Diagonal Preconditioner (DPC) to the complicated Approximate Inverse Preconditioner (AIPC). The code can detect the system condition so that the control parameters, the amount of preconditioning needed and the preconditioning technique can be set. This feature of adapting, changing and optimizing the preconditioner saves considerable amount of memory and CPU. Several examples representing actual systems generated by the HFSS are presented to demonstrate the performance of the suggested hybrid solver.

1 Introduction

The High Frequency Structure Simulator (HFSS) package is a Computer Aided Design (CAD) package that characterizes the performance of microwave circuits and structures by finding the steady state response. The Finite Element Method (FEM) is employed as the numerical technique for computing the fields in all parts of the computational domain. Several types of microwave structures can be modeled, generated and characterized using HFSS. Among them, waveguides, microstrip circuits, strip lines, wire and aperture antennas, coaxial cables and any combined structure from all of them. After the structure is drawn using the CAD package, three main steps are performed. The first is geometry sampling (discretization). In this process, the whole domain (body) is decomposed into small subdomains each representing a small finite element (tetrahedron) where the field quantities have certain form of variation within this element. It should be noted that HFSS employs second order **H1** elements for meshing the domain. The second step includes the assembly of the finite element equations into a global linear system. Finally, this system is solved by a direct method based on LU decomposition.

For large geometries or complicated structures with small geometrical details, the size of the assembled FEM system increases dramatically and the LU solver becomes inefficient because of their high memory requirements. Therefore, there is a serious need for using solvers with low memory requirements. Iterative solvers appear more attractive from this point of view. In this report, we discuss the possibility of using such solvers in the HFSS code to model large problems. In spite of their low memory requirements, iterative solvers often suffer from two major drawbacks. The first is the lack of guaranteed convergence particularly for poorly conditioned systems. Also, the convergence varies widely among

different systems. Two systems of the same size may have completely different conditions and thus their convergence behavior will be significantly different. Another disadvantage of iterative solvers is the often irregular convergence behavior which can become erratic for some routines. For example, the Biconjugate Gradient (BCG) and the Quasi Minimal Residual (QMR) algorithms do not guarantee convergence. However, Generalized Minimal Residual (GMRES) family of solvers are among the robust algorithms and have been successful for ill conditioned systems. The systems generated by HFSS have a wide variation in both size, condition, nature and sparsity. Because of the severe difficulty in knowing or detecting these features, iterative solvers used for dealing with the FEM system should acquire two features. The first of these features is the solver robustness with no breakdown or nearly breakdown possibility and the ability to deal with indefinite systems. The second feature is that of efficiency, even for poorly conditioned systems. That is, the solver should work well for all physical problems and structures. Thus, we are looking for a solver which remains efficient and reliable for all electromagnetic problems.

In this work, our goal is to design and test a solver that is capable of achieving all the aforementioned requirements. Although systems generated by HFSS have wide variations in size, condition and nature, they share some common features. Among them

1. They are indefinite (non positive or semi definite). This is because the second component of the wave equation (which has the ($-k^2$ factor) biases negatively the eigenvalue spectrum. Of course, this becomes more of a serious issue for larger frequencies.

2. The FEM matrices generated by the HFSS may be ill-conditioned. This is due to the following:

    - Second order elements (used in HFSS) deteriorates the matrix condition. Al-

though, higher order elements increase the simulation accuracy, the system condition deteriorates.

- Non uniform meshing of the geometry affects the condition of the final system. This occurs when modeling large geometries with small details.
- Use of Absorbing Boundary Conditions (ABC's) [1]-[3] for mesh truncation also degrades the matrix condition. This is a characteristic feature of the ABC truncation scheme. A better way of truncating the domain is using an integral operator (non local in time and space).

Based on the above facts and our requirements, we looked at various kinds of iterative solvers as well as preconditioners. We implemented and tested several matrices of different sizes, conditions and configurations using the various solvers. After an extensive study, we reached the conclusion that a hybrid GMRES-FGMRES solver with a partial Approximate Inverse Preconditioner (AIPC) is the most acceptable scheme for dealing with linear systems modeled by HFSS.

This report is organized as follows: we introduce the ordinary GMRES iterative solver by presenting the algorithm, its main features and apply it to some HFSS systems. Preconditioning is discussed next along with issues such as complexity, storage, CPU requirements and convergence. At the end, we present our suggested solver and preconditioner.

2 GMRES Solver

In this section, we summarize the basic features of the GMRES solver and provide an understanding of its convergence history. The reasons behind the choice of the GMRES type

of solvers as the preferred solver for solving large ill and well conditioned systems are:

1. GMRES type of solvers are robust and efficient with monotonic convergence characteristics.

2. They guarantee convergence even for badly conditioned system.

3. They lead to the smallest error among all solvers for a fixed number of iterations.

4. They provide room for optimization and adaptation by controlling certain local parameters.

5. They are characterized by a predictable error behavior. This feature will be extensively used to get information about the FEM systems from the initial iterations.

In the following subsection, we summarize the basic features of the GMRES solver and introduce its parameters.

2.1 GMRES Algorithm

The GMRES solver employs the projection method to obtain $\boldsymbol{x}$ satisfying the following linear set of equations

$$\boldsymbol{A}\ \boldsymbol{x} = \boldsymbol{b} \tag{1}$$

where $\boldsymbol{A}$ is the system (FEM) matrix of size $(n \times n)$, $\boldsymbol{x}$ is the solution vector of length n and $\boldsymbol{b}$ is the feed (excitation) vector of. We seek an approximate solution $\boldsymbol{x_m}$ from an affine subspace $\boldsymbol{x_o} + \boldsymbol{K_m}$ of dimension m by imposing the Petrov-Galerkin condition

$$\{b - A\ x_m\ \} \perp L_m \tag{2}$$

where L_m is another subspace of dimension m. Here x_o is the initial guess that is completely arbitrary. The subspace K_m is the Krylov subspace given by

$$K_m(A, r_o) = span\{r_o, Ar_o, A^2 r_o, ...A^{m-1} r_o\} \tag{3}$$

with $r_o = b - A\,x_o$. The different versions of the Krylov subspace methods arise from different choices of the subspace L_m and from the manner in which the system matrix is preconditioned. The GMRES solver is a projection method based on taking $L_m = A\,K_m$. GMRES iterations minimize the residual norm over all vectors in $x_o + K_m$. Without going through the mathematical details, this minimization is based on finding a set of m basis functions (search vectors) which span the solution space. At each iteration, the projection of A on all basis functions are evaluated and m steps of the Arnoldi Modified Gram-Schmidt (MGS) procedures [4] are executed to obtain an estimate of the solution along with the basis functions (m vectors) for the next iteration. The GMRES iterations continue until convergence is achieved. The number of search vectors m per restart is the crucial parameter for the convergence. In general, larger values of m lead to smaller residuals and hence faster convergence. However, the GMRES CPU and memory costs are strongly related to m. For all types of GMRES solvers, the memory cost is $O(mn)$ and the computational cost is $O(m^2 n)$. Thus, there is a strong interest in keeping m as small as possible for efficiency purposes. The GMRES algorithm is given by

Initialize x

$r_o = b - A\,x$

$\boldsymbol{resd} = \sqrt{\boldsymbol{r_o^*.r_o}}$

$\boldsymbol{v_1} = \boldsymbol{r_o/resd}$

Define the $m+1 \times m$ matrix $\boldsymbol{H_m} = \{\boldsymbol{h_{ij}}\ , \boldsymbol{1 <= i <= m+1\ and\ 1 <= j <= m}\}$

Set $\boldsymbol{H_m} = \boldsymbol{0}$

For $j = 1, 2, ..., m$ Do

$\boldsymbol{w_j} = \boldsymbol{A\,M\,z_j}$ where $\boldsymbol{M}$ is the PC

For $i = 1, 2, ...$ Do

$\boldsymbol{h_{i,j}} = \boldsymbol{(w_j, v_i)}$

$\boldsymbol{w_j} = \boldsymbol{w_j - h_{i,j} v_i}$

End

$\boldsymbol{h_{j+1,j}} = \sqrt{\boldsymbol{w_j^*.w_j}}$

$\boldsymbol{v_{j+1}} = \boldsymbol{w_j/h_{j+1,j}}$

End

Compute $\boldsymbol{y_m}$ to minimize $\sqrt{\boldsymbol{s^*.s}}$ where $\boldsymbol{s = resd * e1 - H * y}$

where $\boldsymbol{e1} = \boldsymbol{[1\,0\,0\,0\,.....0\,0]^T}$, and the length of e1 is $\boldsymbol{m+1 \times m}$

and

$\boldsymbol{x = x + M\,V_m\,y_m}$

We will now address the convergence of GMRES and its dependence on m. To make convenient comparisons between different cases, the solver is run for a fixed number of iterations. At the end of each iteration, the error is monitored. The examples used in the study represent actual and real microwave structures such as antennas and waveguides. Results displaying the convergence as a function of m with the corresponding matrix structure are given in Figures 1 to 7. The following systems were examined

1. **System I** has 254 unknowns (n=254) and 7074 non zero elements of 7074 (nz=7074). This corresponds to approximately 11% of matrix fill. For this system, the GMRES algorithm was tested for m= 5, 10, 20, 30 and 40 without preconditioning. The sparsity pattern of the matrix **A** is displayed in Figure 2 and the error history data are given in Figure 1.

2. **System II** has 4108 unknowns (n=4108) and 147876 non zeros elements (nz= 147876). This corresponds to approximately 0.88% of matrix fill. The GMRES algorithm was tested for m = 5, 10, 20, 30 and 40 without preconditioning. The sparsity pattern of this system is displayed in Figure 4 and the error history data are given in Figure 3.

3. **System III** has almost 42 K unknowns (n=41750) and 1447686 non zeros elements (nz= 1447686). This gives about 0.083% of matrix fill. The GMRES algorithm was tested for m = 5, 10, 20 and 40 with no preconditioning. The sparsity pattern of this system is displayed in Figure 6 and the corresponding error history data are shown in figure 5 for each m.

4. **System IV** has almost 110 K unknowns (n=10980) and 2.8 million non zeros elements of (nz= 2.8 M). This indicates about 0.0231 % of matrix filling. The GMRES algorithm was tested for m = 5 and 20 without preconditioning. The corresponding error history data are given in figure 7.

From all these graphs, we can conclude the following about the performance of GMRES when used to solve systems modeled and generated by HFSS.

- For all systems, the error is monotonically decreasing. This reflects the theoretical fact that GMRES guarantees convergence. However, in some cases, for very low tolerance,

convergence is achieved only after substantially more iterations.

- From Figures 1, 3, 5 and 7, the error decreases significantly as m increases. This observation was expected due to the use of sufficiently large set of basis to represent the function.

- The error history is strongly related to the condition of the system. The better the system condition is, the lower the error will be for a fixed number of search vectors per restart.

From the presented data, it is essential to have an estimate for m before executing the GMRES iterations. If this number is lower than the threshold or minimum value, convergence will be extremely slow and may not be achieved at all. On the other hand, if m is too high, storage and CPU are wasted. The optimal value of m is directly related to two main factors, the condition and the size of the matrix. From our examples, we found that the system condition has a strong impact on the convergence. Preconditioning can play an important role for poorly conditioned systems and this topic will be addressed in the following section.

3 Preconditioners

Preconditioners are usually applied to the FEM systems to improve their condition and hence speed up convergence. There are two ways to precondition the FEM system by a certain matrix $\boldsymbol{M}$, namely

1. Right Preconditioning (RPC), in which $\boldsymbol{M}$ is applied to the system as follows

$$\boldsymbol{A}\,\boldsymbol{M}\,\boldsymbol{M}^{-1}\,\boldsymbol{x} = \boldsymbol{b} \tag{4}$$

and the new linear system will take the form

$$\boldsymbol{A_1}\,\boldsymbol{u} = \boldsymbol{b} \tag{5}$$

where $\boldsymbol{u} = \boldsymbol{M^{-1}}\,\boldsymbol{x}$ and $\boldsymbol{A_1} = \boldsymbol{A}\,\boldsymbol{M}$. It should be observed that RPC does not affect the right hand side of the linear system.

2. Left Preconditioning (LPC), in which $\boldsymbol{M}$ is applied to the system as follows

$$\boldsymbol{M}\,\boldsymbol{A}\,\boldsymbol{x} = \boldsymbol{M}\,\boldsymbol{b} \tag{6}$$

and the system to be solved will be

$$\boldsymbol{A_1}\,\boldsymbol{x} = \boldsymbol{b_1} \tag{7}$$

where $\boldsymbol{A_1} = \boldsymbol{M}\,\boldsymbol{A}$ and $\boldsymbol{b_1} = \boldsymbol{Mb}$. Here, the right hand side of the linear system is changed.

The ideal preconditioner should have the following features:

- It should be inexpensive in memory and CPU costs. This means that the PC should not consume large storage in the memory and in the same time, it should be constructed in few operations.
- It should work for all FEM systems and with all solvers.
- It should not be dependent on any other parameter or function. This gives the opportunity to use the preconditioner with all systems generated by HFSS.

3.1 Diagonal Preconditioner DPC

There are different types of preconditioning schemes with variable complexity and cost. The Diagonal Preconditioner (DPC) is the simplest one because it involves a scaling for each row by the diagonal element. However, it typically delivers a speed up of 30% to 60%. This value depends on the system condition, specified tolerance and the sparsity pattern. As shown in Figure 9, the DPC achieves substantial convergence improvements. From this graph, we can see that the convergence improvement is truly impressive since the DPC leads to more than 50 dB error reduction.

3.2 Approximate Inverse Preconditioner AIPC

For the general situation, where the matrix $\boldsymbol{A}$ is indefinite, standard preconditioning techniques may fail due to code breakdown. Also, when $\boldsymbol{A}$ is not diagonally dominant, most preconditioners (such as diagonal and ILU) are not as effective. If $\boldsymbol{A}$ has large nonsymmetric parts, the error using traditional preconditioners will be quite high. Therefore our goal is to find a preconditioner satisfying the following requirements:

1. It should work efficiently for poorly conditioned and highly indefinite systems.

2. It should retain its robustness even if $\boldsymbol{A}$ is not diagonally dominant.

The idea behind the Approximate Inverse Preconditioner (AIPC) depends on finding a sparse matrix $\boldsymbol{M}$ which minimizes the Frobenius norm of the residual of the matrix $\boldsymbol{R}$ given by

$$\boldsymbol{R} = \boldsymbol{I} - \boldsymbol{A}\,\boldsymbol{M} \tag{8}$$

where $\boldsymbol{I}$ is the identity matrix and $\boldsymbol{M}$ is the AIPC. The objective function to be minimized is given by

$$\boldsymbol{F(M)} = \sum_{j=1}^{j=n} ||\boldsymbol{I_j} - \boldsymbol{A}\,\boldsymbol{M_j}||_2^2 \tag{9}$$

where $\boldsymbol{I_j}$ is the $\boldsymbol{j^{th}}$ column of the identity matrix and $\boldsymbol{M_j}$ is also the $\boldsymbol{j^{th}}$ column of the initial guess matrix $\boldsymbol{M}$. Note that $||.||_2$ denotes the Eucledian norm of the matrix.

According to [4], this minimization can be achieved in two different ways. The first is referred to as *Global Iteration* approach which treats the matrix M as an unknown sparse matrix and minimizes the objective function given by (9). One of the well known techniques that utilizes this method is the *Global Steepest Descent Method.* The pseudo-algorithm for this method is as follows:

Initialize $\boldsymbol{M}$

For $i = 1$ till convergence, Do

$\boldsymbol{R = I - AM}$, where $\boldsymbol{I}$ is the identity matrix

$\boldsymbol{G = A^T R}$

$\boldsymbol{\alpha = ||G||_F^2 / ||A\,G||_F^2}$

$\boldsymbol{M = M + \alpha G}$

Apply Numerical Dropping to $\boldsymbol{M}$

EndDo

where $||.||_F$ denotes the Frobinus norm. As expected, the drawback of this technique is its high CPU time and memory cost (order n^2) which are impractical and inefficient.

On the other hand, the *Column Oriented Algorithms* minimize the norm of the individual columns of $\boldsymbol{R}$ given by

$$\boldsymbol{R_j} = \boldsymbol{I_j} - \boldsymbol{A}\,\boldsymbol{M_j} \tag{10}$$

There are many ways to achieve this minimization such as the application of the Conjugate Gradient (CG) or the Minimal Residual (MR) iteration methods. We found that the MR minimization algorithm performed better in most cases. The idea of column minimization can be explained as follows:

1. Minimize (10) for each column of $\boldsymbol{R}$. Two matrix vector products are needed for norm minimization step per column. Thus the CPU cost of the AIPC increases dramatically with the number of minimization steps per column.

2. After minimizing the norm of all columns of $\boldsymbol{R}$, we obtain a good estimate of the AIPC.

3. At each preconditioning step, the density of $\boldsymbol{M}$ will increase and thus the memory requirement will be higher and not affordable. A way to overcome this problem will be addressed later.

4 Ideal Solver for Large Systems

If we have a large system with no *a priori* information about its condition and size, the most robust and efficient way to solve this system is to perform the AIPC evaluation first, then apply it to the GMRES solver with sufficient m. The total memory and CPU costs for this technique will be addressed in the following subsections. Figure 9 shows the outstanding performance of the AIPC against the DPC and the unpreconditioned case. For the same tolerance, GMRES with AIPC preconditioning scheme converges dramatically faster than

the DPC preconditioned version (see Figure 9). However, constructing the AIPC before starting the GMRES iterations is expensive in both memory and CPU.

4.1 Memory costs

For a system of size n generated by HFSS, the storage needed for the matrix $\boldsymbol{A}$ will be on the order of $20n$ to $25n$. However, the total storage requirements depend on the employed algorithm and the utilized preconditioner. In designing the AIPC preconditioner, the following facts must be taken into account:

1. Usually $\boldsymbol{M}$ becomes denser as the number of norm minimization steps per column n_i increases. Typically, for $n_i > 2$, $\boldsymbol{M}$ becomes denser and may take up to $10\boldsymbol{A}$. This deteriorates significantly the advantage of utilizing iterative solvers. For the case where only one iteration is performed with an initial PC as the DPC, the size of M is exactly the same as $\boldsymbol{A}$ ($20n$ to $25n$).

2. For the GMRES iterations, m vectors each of length n are needed to be stored. Their storage is $O(mn)$.

3. Three additional vectors are needed to perform the entire GMRES iteration loop.

Thus, the total storage of the GMRES solver with the AIPC will be as follows:

$$Total\ Storage = Storage\ for A + Storage\ for M + (m+3)n \tag{11}$$

Typically, if $\boldsymbol{A}$ and $\boldsymbol{M}$ each requires $25n$ complex numbers, the maximum storage requirement will be on the order of

$$Storage = 25n + 25n + (m+3)n \tag{12}$$

From this equation, we observe that

- The PC size is a major contributer to the memory cost and we may therefore choose to have a less accurate AIPC rather than increasing its size. Thus, $\boldsymbol{M_j}$ in (10) can be the column obtained after only one iteration.
- Numerical dropping (not storing matrix or vector elements with small magnitudes) may be applied to the basis functions (m ones) to reduce the storage requirements.

4.2 CPU costs

We measure the CPU time by evaluating the number of matrix vector products for all iterations. For faster execution, it is essential to minimize the total number of matrix vector products (MVP). In our case, we form the whole AIPC and then perform some GMRES iterations, the total number of matrix vector products are determined as follows:

1. Two matrix vector products are needed per preconditioning iteration per column. For one norm minimization step per column, $2n$ MVPs are needed.
2. One matrix vector product per GMRES iteration (m iterations) is performed.
3. One matrix vector product is needed for the residual evaluation.

Therefore, the total MVP can be approximated as

$$\boldsymbol{MVP = 2n + mI_{gmres}} \tag{13}$$

where $\boldsymbol{I_{gmres}}$ is the total number of global GMRES iterations.

5 Practical Issues

As indicated in the previous sections, a robust iterative solver can be formed by considering the following:

- Start with the matrix $\boldsymbol{A}$ and an initial guess for the PC, construct a sparse AIPC with the same size and sparsity as $\boldsymbol{A}$. A good initial PC is the DPC.
- Perform GMRES iterations with large m and apply numerical dropping for the residual vectors.
- The memory cost of this technique is on the order of $\boldsymbol{2A - 3A}$. This is due to that the size of the PC is exactly the same as $\boldsymbol{A}$ (based on one minimization step per column) and the basis functions (m=30-40 typical) storage is less than $\boldsymbol{A}$.
- The total MVPs for all iterations is given by

$$\boldsymbol{MVP_{tot} = I_{gmres}(m+1) + 2n} \tag{14}$$

  The first term can be regarded as the GMRES term while the second one is the pre-conditioning one. The ratio between the second term to the first is much higher than unity for large systems. This increases the CPU time and slows down convergence.

However, this approach for solving large systems with unknown conditions and structures, although robust, is inefficient due to the following drawbacks:

- The CPU cost of the AIPC construction is extremely high. The ratio between the second to the first term of equation (14) is in the range of 100 or more, implying that the AIPC will dominate the CPU time.

- For good and moderately conditioned systems, the AIPC may not be needed at all. Typically, convergence can be achieved in a few iterations using the DPC. This is due to the robustness of the GMRES solver in addition to having large values for m.

- Even for ill conditioned systems, calculating the entire AIPC matrix is wasteful. In such cases, partial formation of the AIPC may be enough to speedup convergence. The percentage of the AIPC matrix to be formed depends on the system condition and required tolerance. Typically, formation of 10% to 40% of the AIPC may be enough to achieve convergence.

Changing the PC at each iteration step is an approach which allows improvements in the PC. The Flexible Generalized Minimal Residual (FGMRES) solver is particularly suited for this task. An overview of this solver is given in the following section.

6 Flexible Generalized Minimal Residual (FGMRES) Algorithm

In all preconditioners discussed in the previous sections, it was implicitly assumed that the preconditioning matrix $\boldsymbol{M}$ is fixed at all steps of the solver. However, in many cases, $\boldsymbol{M}$ may not be a constant operator and therefore, the iterative solver preconditioned with constant operators will not converge. *Flexible* iterative solvers permit variations or changes of the preconditioner from one step to another . One of these solvers is the Flexible GMRES. The FGMRES pseudo code is given below

Initialize x

$\boldsymbol{r_o = b - A\,x}$

$\boldsymbol{resd} = \sqrt{\boldsymbol{r_o^*.r_o}}$

$\boldsymbol{v_1} = \boldsymbol{r_o/resd}$

Define the $\boldsymbol{m+1 \times m}$ matrix $\boldsymbol{H_m} = \{\boldsymbol{h_{ij}}\ , \boldsymbol{1 <= i <= m+1\ \ and 1 <= j <= m}\}$

Set $\boldsymbol{H_m = 0}$

For $j = 1, 2, ..., m$ Do

$\boldsymbol{z_j} = \boldsymbol{M_j\, z_j}$

$\boldsymbol{w_j} = \boldsymbol{A\, z_j}$

For $i = 1, 2, ...$ Do

$\boldsymbol{h_{i,j}} = \boldsymbol{(w_j, v_i)}$

$\boldsymbol{w_j} = \boldsymbol{w_j - h_{i,j} v_i}$

End

$\boldsymbol{h_{j+1,j}} = \sqrt{\boldsymbol{w_j^*.w_j}}$

$\boldsymbol{v_{j+1}} = \boldsymbol{w_j/h_{j+1,j}}$

End

Compute $\boldsymbol{y_m}$ to minimize $\sqrt{\boldsymbol{s^*.s}}$ where $\boldsymbol{s = resd * e1 - H * y}$

and $\boldsymbol{e1} = [\boldsymbol{1\,0\,0\,0 .....0\,0}]^{\boldsymbol{T}}$, length of e1 is $\boldsymbol{m+1 \times m}$

and

$\boldsymbol{x} = \boldsymbol{x} + \boldsymbol{Z_m\, y_m}$

Note that the matrix $\boldsymbol{Z_m}$ is formed from all the vectors $\boldsymbol{z_j}$ where $j = 1, 2, ..m$. The main advantage of FGMRES is the ability to change or modify the preconditioner at each iteration step. The expense of this flexibility is the additional storage requirement for m vectors (each with length n).

7 Effects of m

As displayed in the first set of graphs, when the GMRES solver was tested without any preconditioning, the effect of m was significant. Therefore, it is recommended to increase the number of restarts as much as possible. However, this increases the storage as well as the CPU time significantly. To overcome this problem, it was observed that early iterations of GMRES are more sensitive to m. Therefore it is recommended to start with a large m and then reduce it for the subsequent iterations. It should be noted that the variation of m throughout the iteration can take several forms. However, all of them should satisfy the following

- High changes for m between different iterations are not recommended. Such changes deteriorate the convergence.
- A minimum value (dc value) for m is recommended regardless of n to avoid a situation when m vanishes or becomes very small. In the latter case, convergence may not be achieved at all.

One way of selecting m is by using the following vcriterian

$$m_i = am_{i-1} + d \tag{15}$$

where $\boldsymbol{a}$ and $\boldsymbol{d}$ are constants and $\boldsymbol{i}$ is the iteration number. These constants can be adjusted for each class of systems or problem type.

8 Suggested Flexible Solver

In accordance with the memory and CPU requirements of the iterative GMRES-AIPC, we developed an efficient way of dealing with systems generated by HFSS. Our approach involves three main stages. The error is monitored in the first few iterations. If convergence is not achieved, we start the second stage by turning on the the AIPC and by changing (increasing) m. In the AIPC part, we start by applying the norm minimization to columns with high norms. This means that the AIPC is generated gradually by adding more columns to the initial AIPC at each iteration. Using this technique, a final solver will be formed from the following modules:

8.1 Module I: GMRES-DPC Solver

After setting m (usually 40 works for most systems) and both $\boldsymbol{a}$ and $\boldsymbol{d}$, we perform few (less than five) GMRES iteration with diagonal preconditioning. Convergence is checked at the end of each iterations and the error at each iteration is recorded. If convergence is not achieved within these iterations, the error analysis section is activated and parameters are reset to speed up convergence.

The advantages of executing the initial GMRES iterations with DPC can be summarized as follows:

- Significant improvements in the system condition are achieved using the simple DPC without memory or CPU cost.
- For well or moderately conditioned systems, convergence is likely to be achieved during these few GMRES iterations and hence no additional memory or CPU operations will

be wasted.

- In the case of GMRES, only one matrix vector product is needed per GMRES iteration. This gives a total of approximately mI_{gmres} MVPs in the GMRES algorithm, where I_{gmres} is the total number of global iterations before convergence. Note that this number of MVPs represents almost half of the that needed by the FGMRES algorithm.

- Starting with large m reduces the initial error and increases the chance for faster convergence. Reducing m in the subsequent iterations helps in achieving substantial savings in memory and CPU operations.

- The error vector obtained from the GMRES-DPC iterations gives an indication about the system condition as well as its convergence characteristics. For example, in Figure 8, two systems generated by HFSS were solved by the same GMRES-DPC algorithm with the same m. The error was recorded for five iterations and the following was found:

  System A

  - Number of equations: 10980.
  - Number of non zero elements in the matrix: 4.4 million elements.
  - System Condition: moderate.
  - Initial error:-57 dB.
  - Type of solver: GMRES-DPC.

  System B

  - Number of equations: 82740.

- Number of non zero elements in the matrix: 2.8 million elements.
- System condition: extremely ill.
- Initial error is: -11 dB.
- Type of solver: GMRES-DPC.

From the observed error of the first few iterations, we can conclude that system **B** was much more poorly conditioned than system **A**. On the other hand, system **B** requires a strong preconditioner and possibly higher m to achieve convergence.

8.2 Module II: Error Analysis

After performing a few GMRES-DPC iterations, one of the following situations is likely to occur:

1. Convergence is achieved during the first few iterations of the GMRES-DPC part. Therefore, there is no need for increasing m or introducing the AIPC. This situation typically occurs with well and moderately conditioned systems.

2. Convergence is nearly achieved. This can be deduced by checking two important values (quantities). One is the error decay rate and the other is the latest error. If there is a considerable decay rate and the final error is close to the prespecified tolerance, it is advisable to continue with more GMRES-DPC steps.

3. The error after these iterations is much higher than the specified tolerance and the corresponding error decay rate is low. In this case, we proceed with partial formation of the AIPC up to a certain portion (say 50% or less). We perform this preconditioner

on steps by constructing a small part of the AIPC (say 10%) at each iteration. Also, we may need to increase m.

8.3 Module III: FGMRES-AIPC solver

The last part of the proposed solver is a combination between FGMRES and the AIPC. After the error analysis was completed in module II, certain data should be supplied as inputs to the FGMRES-AIPC part. These data include

- The initial m for the FGMRES-AIPC part..
- The constants specifying the linear change in m (defined by $\boldsymbol{a}$ and $\boldsymbol{d}$).
- Percentage of the AIPC to be constructed at each FGMRES iteration.
- Upper bound of the preconditioning percentage.

It should be noted that the FGMRES section has the following features

1. Two matrix vector products are performed per FGMRES step.
2. Guaranteed convergence.
3. Unnecessary preconditioning and CPU operations are completely avoided.
4. Level of preconditioning is automatically related to the system condition.

9 Hybrid Solver Cost

In this section, we give an estimate of the hybrid solver cost and make comparisons with direct solvers. The memory cost will be calculated in terms of the maximum storage requirements

for the worst case scenario. The CPU cost is given as a function of the number of operations.

9.1 Memory Cost

The worst case scenario occurs when the whole AIPC is constructed and used in conjunction with the FGMRES solver. In this case, the memory cost will be as follows:

1. The AIPC matrix has the same configuration as $\boldsymbol{A}$. For the systems generated by HFSS, the average number of non zero elements per column/row is 20 to 25. Thus, the maximum memory requirement for the AIPC will be $25n$.

2. Storage of the $2m$ basis functions, each of length n is needed. These vectors are usually sparse (not highly sparse). Moreover, numerical dropping increases their sparsity, thus reducing their storage requirements. At worst, the storage of these basis functions is $O(2mn)$.

3. Three additional vectors each of length n are needed, requiring $3n$ storage.

Summarizing, the maximum storage is

$$Maximum\ \ Storage = 25\,n + 2\,m_{eff}\,n\ +\ 3\,n = (25\ +\ m_{eff}\ +\ 3)\,n \tag{16}$$

where m_{eff} is the effective (average) number of search vectors per restart and it is given by

$$\boldsymbol{m_{eff}} = \frac{1}{\boldsymbol{I_{tot}}} \sum_{\boldsymbol{i=1}}^{\boldsymbol{i=I_{tot}}} \boldsymbol{m_i} \tag{17}$$

where $\boldsymbol{I_{tot}}$ is the total number of iterations in both GMRES and FGMRES parts of the code and $\boldsymbol{m_i}$ is the number of search vectors within the $\boldsymbol{i^{th}}$ iteration. Typically, the required storage is much less than the maximum value given by (18). This is due to that

the complete AIPC is not needed and the basis functions are sparse either by nature or by numerical dropping. Testing different systems modeled by HFSS showed that the typical storage requirement is

$$Typical\ \ Storage = (25 + c\, m_{eff} + 3\, c)\, n \tag{18}$$

where c is the sparsity percentage factor and ranges from zero to one.

Direct solvers for sparse systems need memory on the order of $\mathbf{10}\boldsymbol{A}\boldsymbol{to}\mathbf{15}\boldsymbol{A}$ with approximately $\mathbf{12}\boldsymbol{A}$. Since $\boldsymbol{A}$ has an approximate size of $25n$, the total storage for direct solvers will be of order $300n$. Thus, from (18), the memory ratio between the developed iterative solver and the direct one is given by

$$memory\ \ ratio = \frac{25 + (m_{eff} + 3)\, c}{300} \tag{19}$$

Figure 10 graphs this ratio as a function of the average (effective) m for different values of c from (.1 to 1). As shown in this figure, hybrid solver offers significantly lower memory requirements than direct ones.

9.2 CPU cost

The CPU cost of the hybrid solver will be expressed in terms of the total number of operations. Again, the worst case scenario is assumed. The following operations determine the CPU cost

1. For AIPC, almost $O(p^2 n)$ operations are required, where p is the matrix bandwidth (typically from 20 to 25).

2. For FGMRES, the maximum CPU cost is on the order of $O(m_{eff}^2 n)$. Therefore, the total cost for all FGMRES iterations is given by

$$Number\ \ of\ \ operations = p^2 n + (m_{eff}^2)\, n\, I_{fgmres} \tag{20}$$

where I_{fgmres} is the number of iterations in the FGMRES part of the solver. The total number of operations for a sparse direct solver is $O(q^2 n)$ where q is the lower (or upper) bandwidth. Typically for HFSS systems, q is around 400. Consequently, the CPU ratio of the hybrid and direct solvers is

$$CPU\ \ ratio = \frac{625 + m_{eff}^2 I_{fgmres}}{160000} \tag{21}$$

Figure 11 plots this ratio for different number of iterations and different values of m. Clearly as the effective m increases, the CPU time increases and approaches that of the direct solver. Keeping the total number of iterations small (less than 10) makes the developed hybrid iterative solver a more attractive choice.

References

[1] Webb and Kanellopoulos, " Absorbing boundary conditions for the finite element solution of the vector wave equation," *Microwave Opt. Tech. Lett.* Vol. 2, pp. 370–372, 1989

[2] A. Chatterjee and J.L.Volakis, "Conformal absorbing boundary conditions for the vector wave equation," *Microwave Opt. Tech. Lett.* Vol. 6, pp. 886–889,1993

[3] Katz, D.S, Thiele, E.T., and Taflove, A. "Validation and Extension to Three dimensions of the Berenger PML Absorbing Boundary Condition for FD-TD Meshes," *IEEE Microwave and Guided Wave Letters*, August, 1994, p.268-270.

[4] Y. Saad *Iterative Methods for Sparse Linear System,* PWS publishing co., 1996.

List of Figures

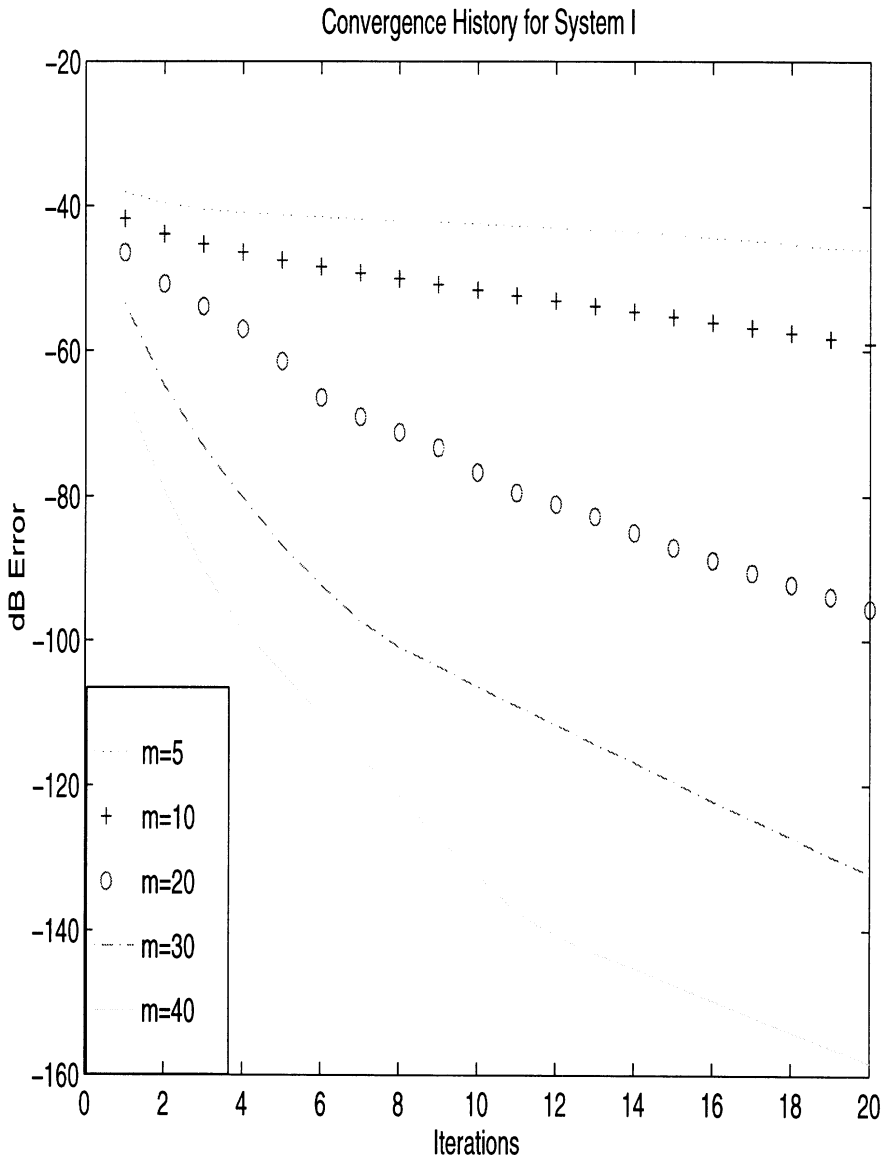

Figure 1: Effect of m on the convergence of **System I**

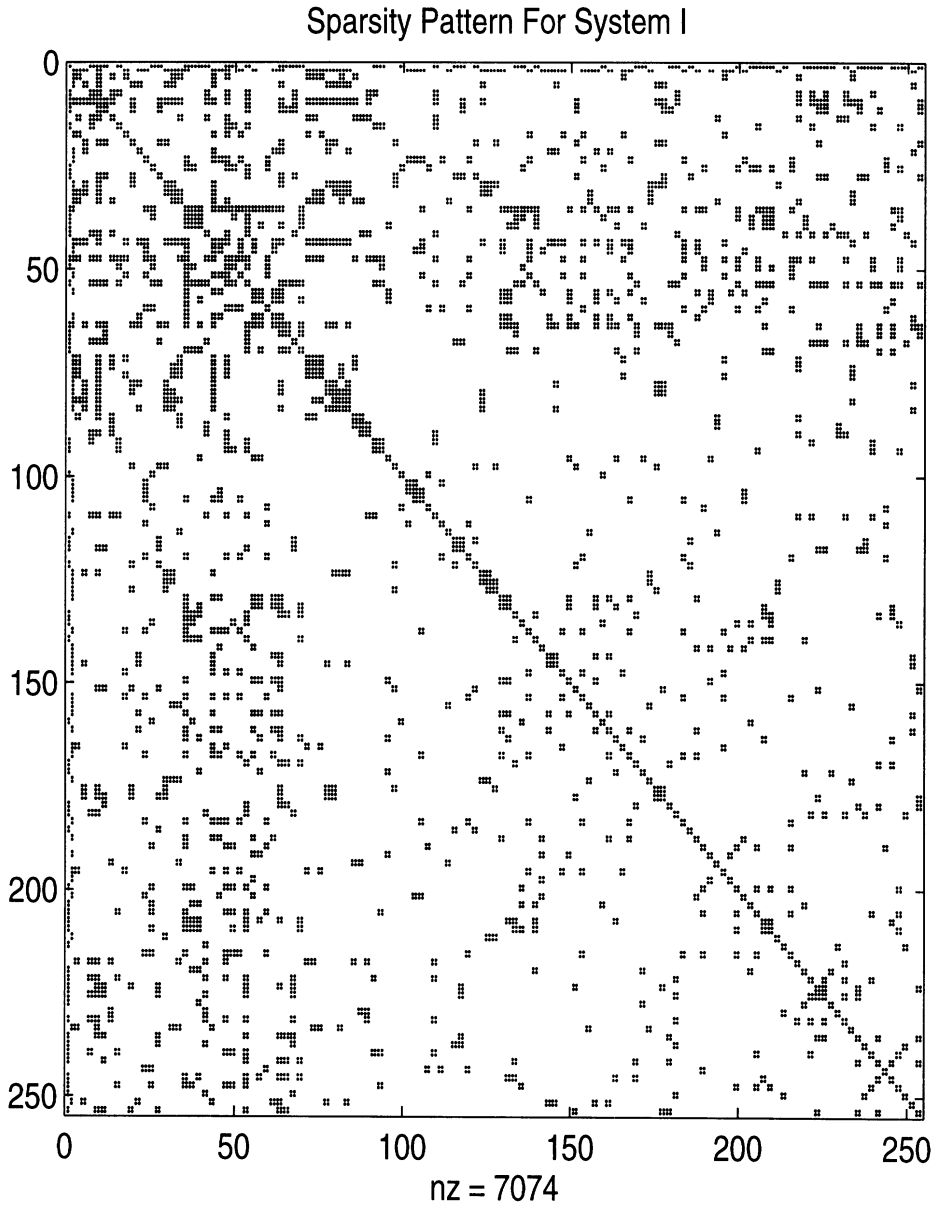

Figure 2: Sparsity Pattern of **System I**

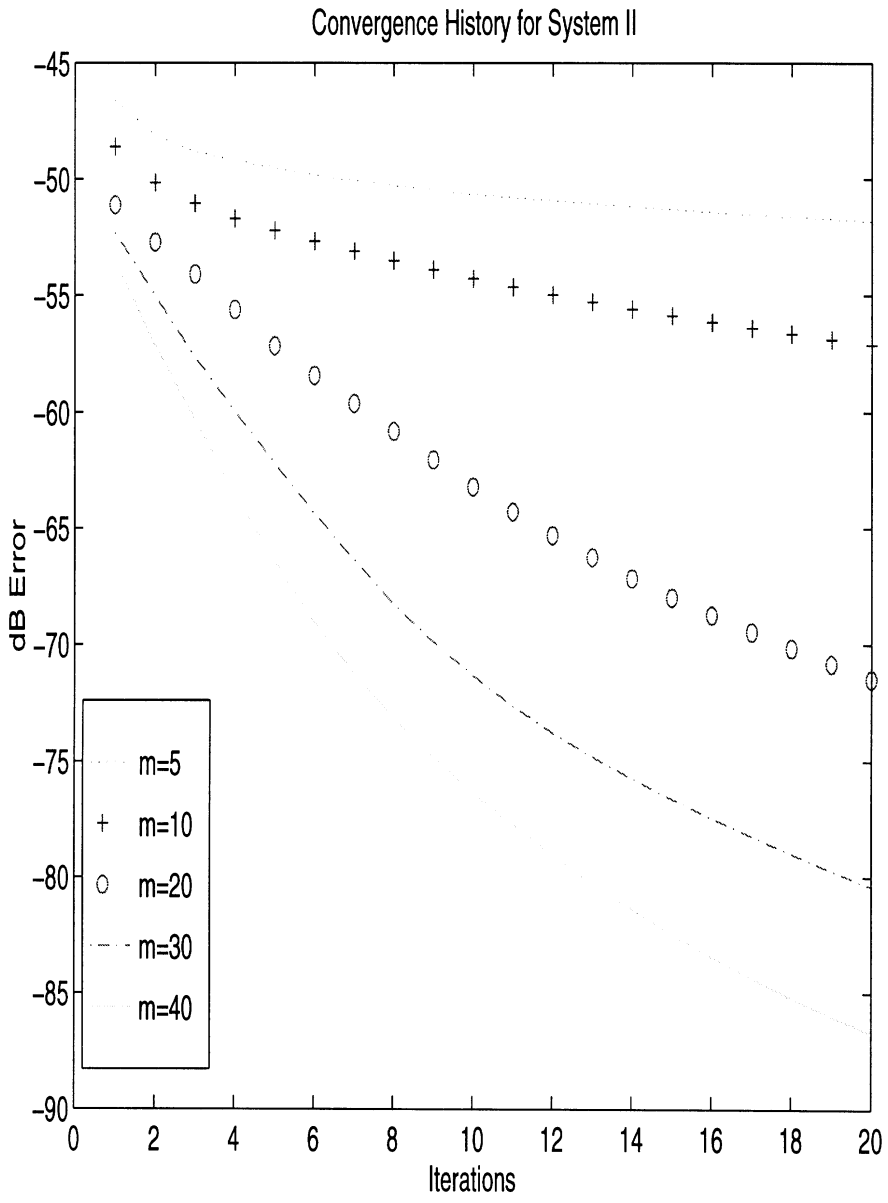

Figure 3: Effect of m on the convergence of **System II**

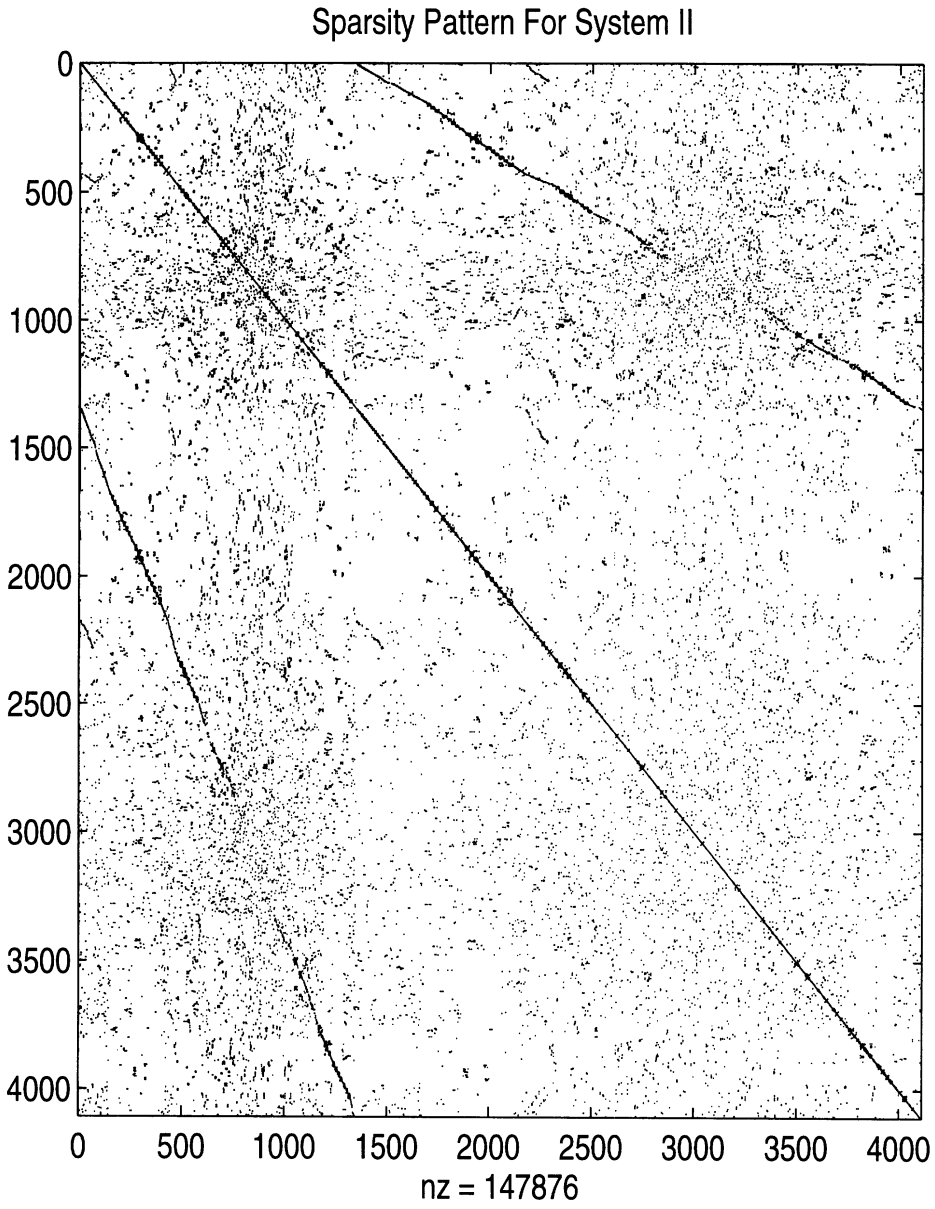

Figure 4: Sparsity Pattern of **System II**

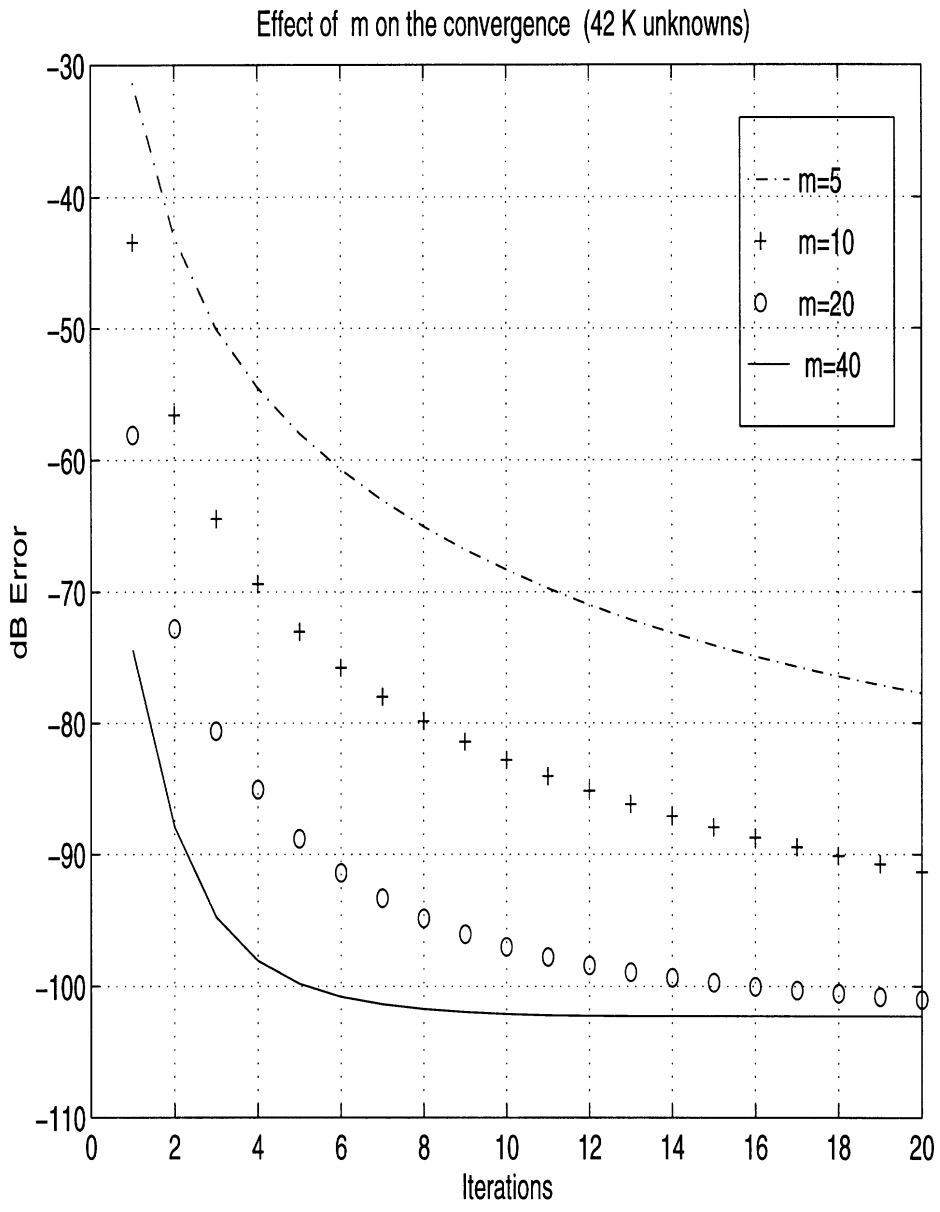

Figure 5: Effect m on the convergence of **System III**

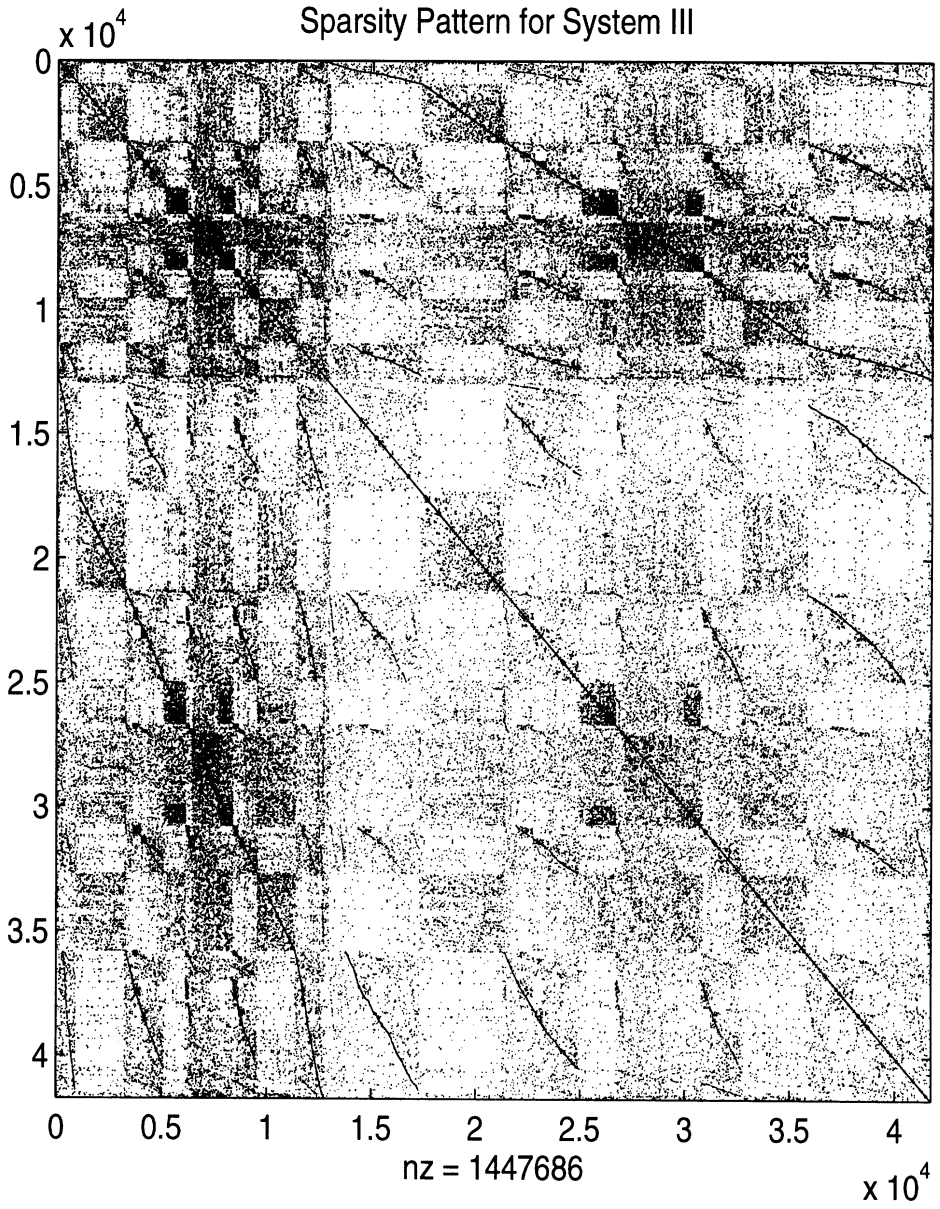

Figure 6: Sparsity Pattern of **System III**

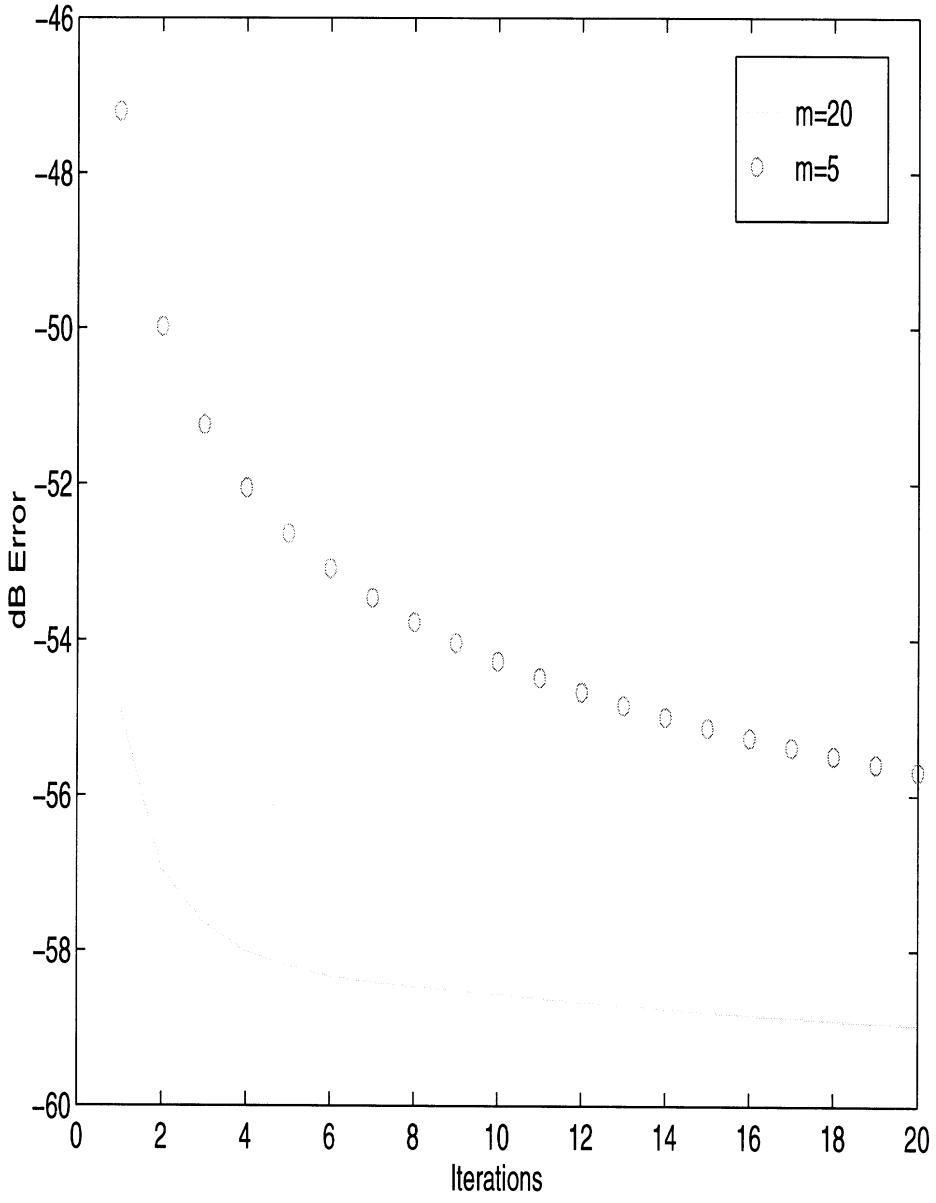

Figure 7: Effect m on the convergence of **System V**

Figure 8: Convergence performance for two systems, **System A** and **System B**. System A is represented by the upper line while System B is given by the lower one.

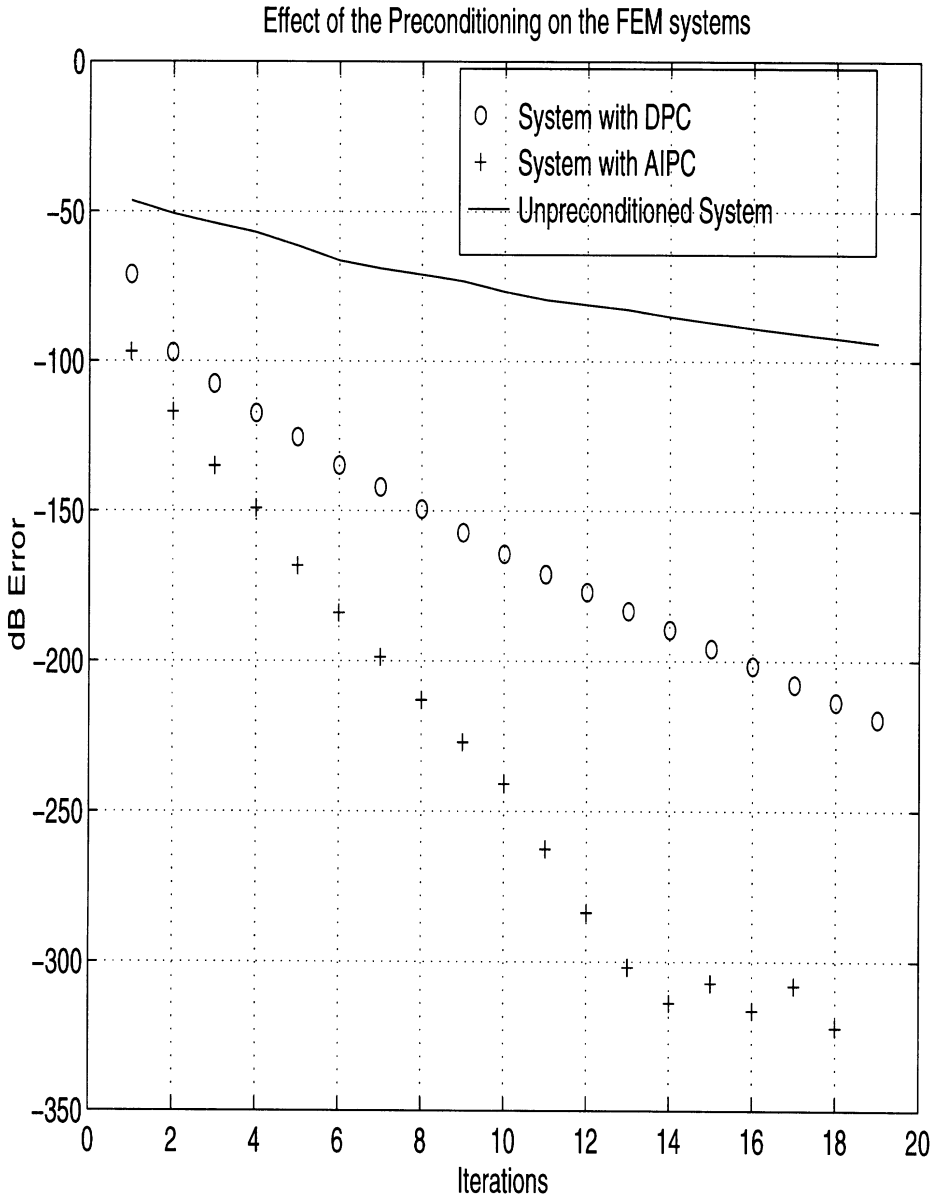

Figure 9: Comparison between the GMRES solver in the unpreconditioned case, the DPC case and the AIPC case

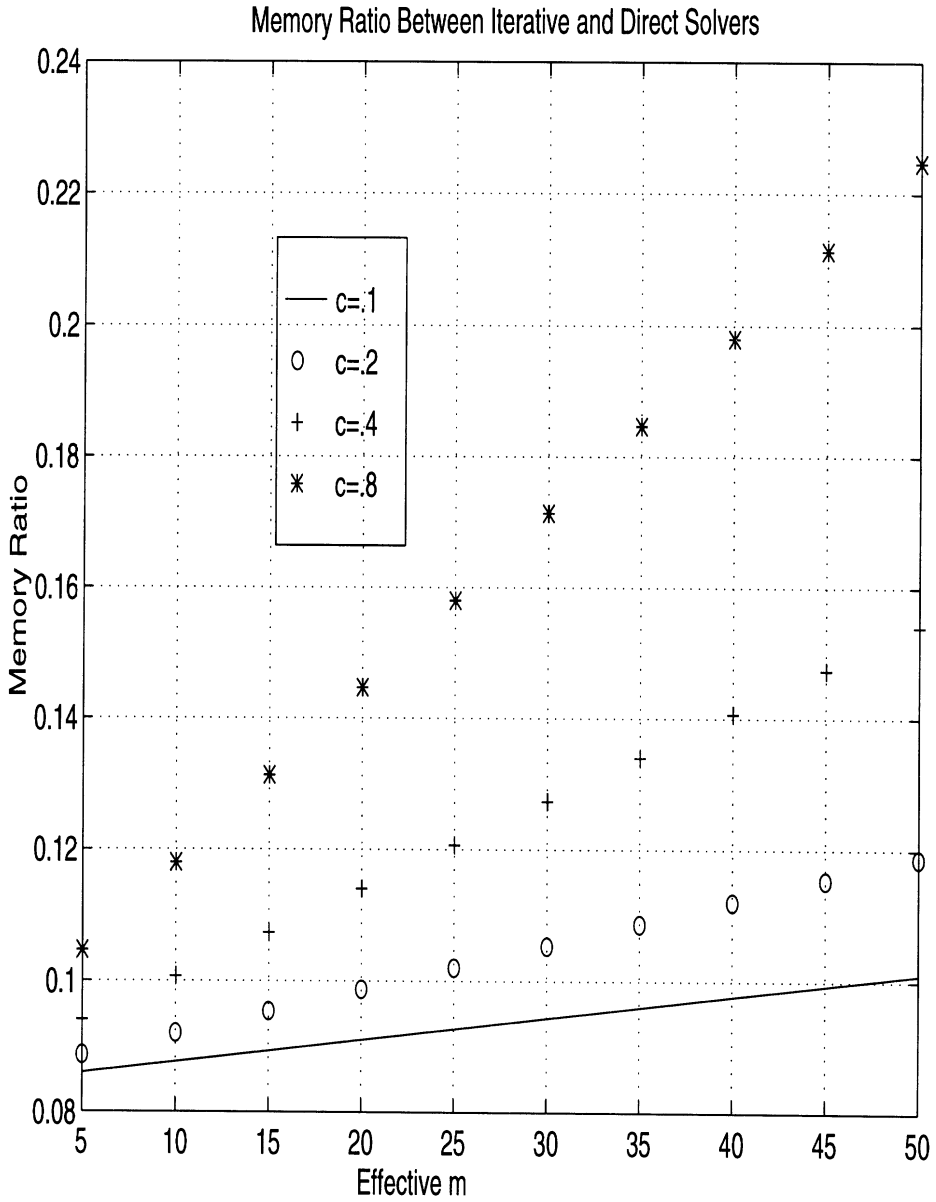

Figure 10: Memory Ratio Between the Suggested Hybrid S and Direct Solvers; c is the Sparsity Factor and varies between 0.1 and 0.8

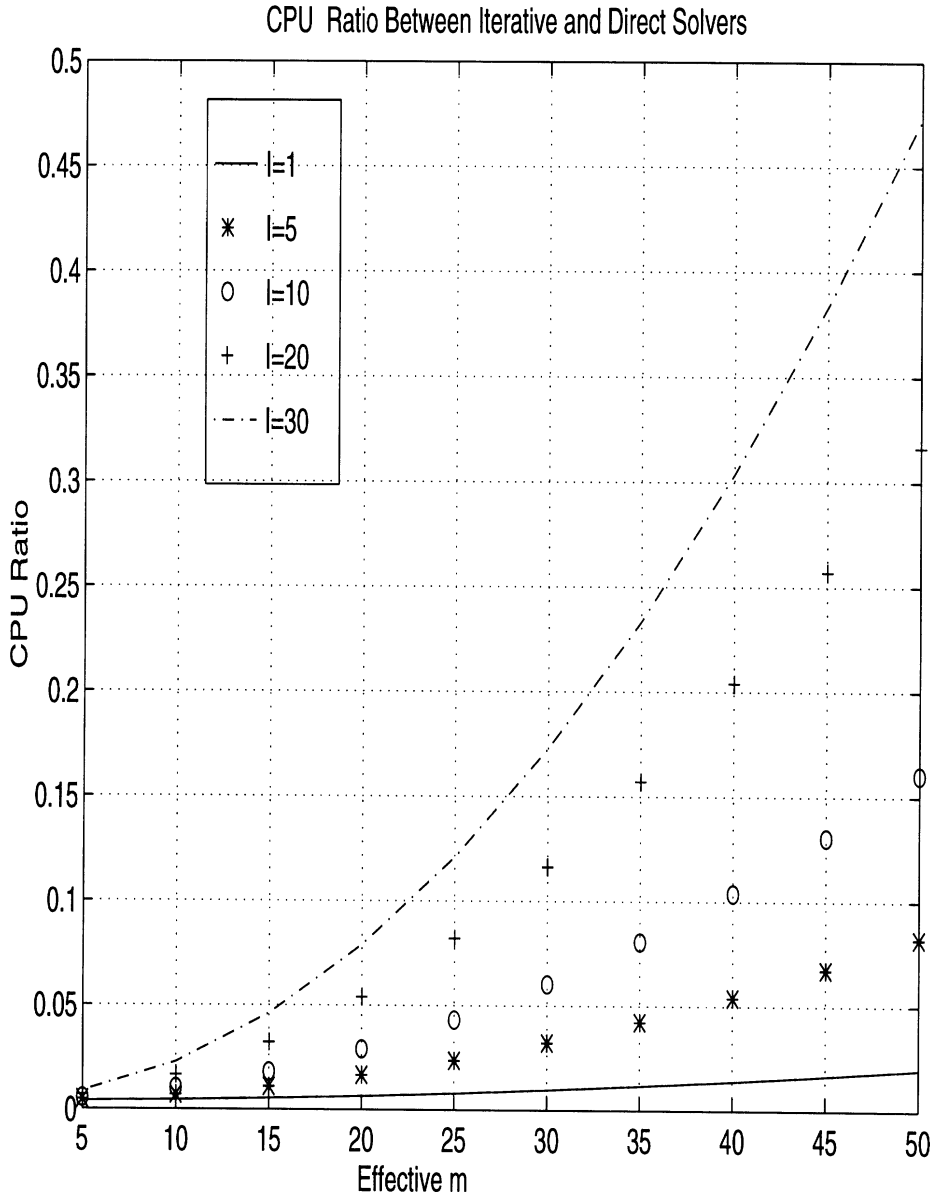

Figure 11: CPU Ratio Between the Suggested Hybrid Solver and Direct Solvers; I is the total number of iterations