Computational Issues in Damping Identification for Large Scale Problems*

Deborah F. Pilkey

Engineering Science & Mechanics
Virginia Polytechnic Institute
Blacksburg, Virginia 24061-0219
`dpilkey@vt.edu`

Kevin P. Roe

ICASE
NASA Langley Research Center
Hampton, Virginia 23681-0001
`kproe@icase.edu`

Daniel J. Inman

Engineering Science & Mechanics
Virginia Polytechnic Institute
Blacksburg, Virginia 24061-0219
`dinman@vt.edu`

Abstract

Damage detection and diagnostic techniques using vibration responses that depend on analytical models provide more information about a structure's integrity than those that are not model based. The drawback of these approaches is that some form of workable model is required. Typically, models of practical structures and their corresponding computational effort are very large. One method of detecting damage in a structure is to measure excess energy dissipation, which can be seen in damping matrices. Calculating damping matrices is important because there is a correspondence between a change in the damping matrices and the health of a structure. The objective of this research is to investigate the numerical problems associated with computing damping matrices using inverse methods.

Two damping identification methods are tested for efficiency in large-scale applications. One is an iterative routine, and the other a least squares method. Numerical simulations have been performed on multiple degree-of-freedom models to test the effectiveness of the algorithm and the usefulness of parallel computation for the problems. High Performance Fortran is used to parallelize the algorithm. Tests were performed using the IBM-SP2 at NASA Ames Research Center.

The least squares method tested incurs high communication costs, which reduces the benefit of high performance computing. This method's memory requirement grows at a very rapid rate meaning that larger problems can quickly exceed available computer memory. The iterative method's memory requirement grows at a much slower pace and is able to handle problems with 500+ degrees of freedom on a single processor. This method benefits from parallelization, and significant speedup can be seen for problems of 100+ degrees-of-freedom.

*This research was supported by the National Aeronautics and Space Administration under NASA Contract Nos. NASA-19480 and NAS1-97046, while the authors were in residence at ICASE, NASA Langley Research Center, Hampton, VA 23681-0001.

1 Introduction

The synthesis of damping in structural systems and machines is extremely important if the model is to be used in predicting transient responses, transmissibility, decay times, or other characteristics in design and analysis that are dominated by energy dissipation. Methods for determining the mass and stiffness matrices of a system are more straightforward than those for determining the damping matrix, as they represent quantities which can be measured and evaluated by static tests. Damping, on the other hand, must be determined by dynamic testing. This makes the process of modeling and experimental verification difficult. It is assumed here that acceptable models of the mass are available and that it is desired to use either the eigenvalue and eigenvector, or transfer function information to construct a damping matrix. This is known as an inverse problem.

One application of the inverse problem is diagnostics. This idea is to test for changes in a structure's properties by looking at changes in measurable values such as mode shapes or frequencies. Here the underlying assumption is that changes in the damping values correspond to some sort of change in the structure's health.

A factor not yet investigated in damping identification is how the problem size affects the ability to solve the inverse problem. Although most example problems used in published papers are able to prove that a method is capable of producing the desired results for small problems, it is usually not mentioned whether a particular method is still viable when the problem size is increased to practical limits with many degrees of freedom. Most inverse problems in damping identification require some sort of iteration or optimization procedure. These can become very costly with the increasing problem size in regards to execution time and computer memory limitations.

The increased execution time and required computer memory creates a need for high performance computing. Parallel machines can provide a decreased execution time when the workload is distributed among multiple processors and if the data is also distributed then the per processor memory requirement may be decreased as well. At least two ways of programming parallel computers are available. Hand-coded message passing can be done and may achieve excellent performance at the price of a long conversion process. This approach forces the programmer to explicitly handle the low-level details of the required communication and to hardwire the choices into the code. This makes the program difficult to understand and inflexible. The other method is for a compiler to handle the parallelization of a sequential code. High Performance Fortran (HPF) allows the programmer to parallelize a sequential code while only requiring the addition of a few data distribution directives at the top of the code. HPF has a much lower learning curve and handles most of the low level optimizations without the user's knowledge. This means good performance can be achieved while abstracting away the details.

In the following paper, Section 2 describes several methods used in damping identification. Section 3 is an introduction to the capabilities and uses of High Performance Fortran. Next, two recent representative damping identification routines are described in detail. A sample problem is presented, and the parallelization procedure is explained for both methods. Examples are followed by results and conclusions.

2 Background

The problem being investigated assumes a structural system consisting of mass ($\mathbf{M}$), damping ($\mathbf{C}$), and stiffness ($\mathbf{K}$) matrices such that the response *x(t)* satisfies

$$\mathbf{M}\ddot{x} + \mathbf{C}\dot{x} + \mathbf{K}x = \mathbf{0}, \tag{1}$$

where x is an n by 1 vector varying with time, representing the displacements of the masses in a lumped mass system (n is the number of degrees of freedom). The vectors $\ddot{x}$ and $\dot{x}$ represent the acceleration and velocity respectively of the lumped masses.

The idea behind an inverse problem is to find the physical parameters of a system (mass, damping, and stiffness) from its behavior using measurements such as forced responses and natural frequencies. Damping identification is an inverse problem in which the damping matrix is the desired result.

Matrix identification can be done by several methods, many of which are described below. Several approaches make limiting assumptions such as diagonal or proportional damping matrices, while others assume the experimental data is incomplete and use only partial eigensystems to solve for damping.

Caravani and Thomson (1974) developed a method for damping identification specific to viscous damping. In their iterative method, the goal is to minimize the real and ideal response vectors. This method is meant to solve relatively simple problems and requires that the "true" response be known *a priori*. Fritzen (1986) optimizes a loss function in his Instrument Variable method. This method requires a full set of measurement information, but works well when noise is added to the system. It is not necessary for this method to have a prior model of the system. This instrument variable method is iterative, but requires few iterations. It is in the frequency domain, and works best when damping is large. Several methods are combined in a paper by Roemer and Mook (1992). Noisy measurement data are used to identify properties of lumped parameter systems by combining two time domain techniques and an estimation technique. Hasselman (1972) uses perturbation theory to solve for the damping matrix of the systems with small, proportional damping. Linear viscous damping is assumed.

3 Introduction to High Performance Fortran

High Performance Fortran (HPF) was designed to provide a portable extension to Fortran 90 for writing data parallel applications. It includes features for mapping data to parallel processors, specifying data parallel operations, and methods for interfacing HPF programs to other programming paradigms. It is expected that HPF will be a standard programming language for computationally intensive applications on many types of machines, including massively parallel MIMD (Multiple Instruction Multiple Data) and SIMD (Single Instruction Multiple Data) multiprocessors as well as traditional vector processors.

Features of HPF include compiler directives, parallel constructs, array intrinsics, and escape mechanisms for interfacing with other languages and libraries. Compiler directives are structured comments that assert facts or suggest implementation strategies about a program to the compiler (Koelbel, et al, 1994). HPF directives allow the programmer to specify how to assign array elements to the memory of processors. Data mapping can be accomplished by using `DISTRIBUTE` statements which partition the data over the processors in a `BLOCK`, `CYCLIC`, or general `CYCLIC(m)` manner. In a `BLOCK` distribution, contiguous blocks of the array are distributed across the processors. In a `CYCLIC` distribution, array elements are distributed among processors in a round-robin fashion. In a `CYCLIC(m)` distribution, contiguous data blocks of size `m` are distributed cyclically. The `ALIGN` directive tells the compiler how different arrays are aligned with each other to reduce interprocessor communication. The combination of alignment and distribution defines how the arrays are mapped. Data parallel operations can be specified using parallel constructs such as `FORALL` and `INDEPENDENT`.

4 Least Squares method

Chen, Ju, and Tsuei (1996) developed a method in the frequency domain that can solve for the damping matrix independent of the mass and stiffness matrices, using data obtained from a frequency response function. Taking into account the symmetry of the damping matrix, the damping matrix can be compressed into the vector:

$$\mathbf{\bar{c}} = [\mathbf{c_{11}}\ \mathbf{c_{21}}\ \mathbf{c_{22}}\ \mathbf{c_{31}}\ \ldots]^{\mathbf{T}}$$

without losing any information contained in the full matrix. A flow chart of this method is presented below where the entries of $\mathbf{\bar{V}}$ and $\mathbf{\bar{q}}$ are rearranged to solve for $\mathbf{\bar{c}}$. Then, premultiplying by the transpose of $\mathbf{\bar{V}}$, a least squares method can be used to solve for $\mathbf{\bar{c}}$.

Figure 1 illustrates the Least Squares Method, where $\mathbf{H_I}$ is the imaginary part and $\mathbf{H_R}$ is the real part of the transfer function. Also, ω is the natural frequency of the system.

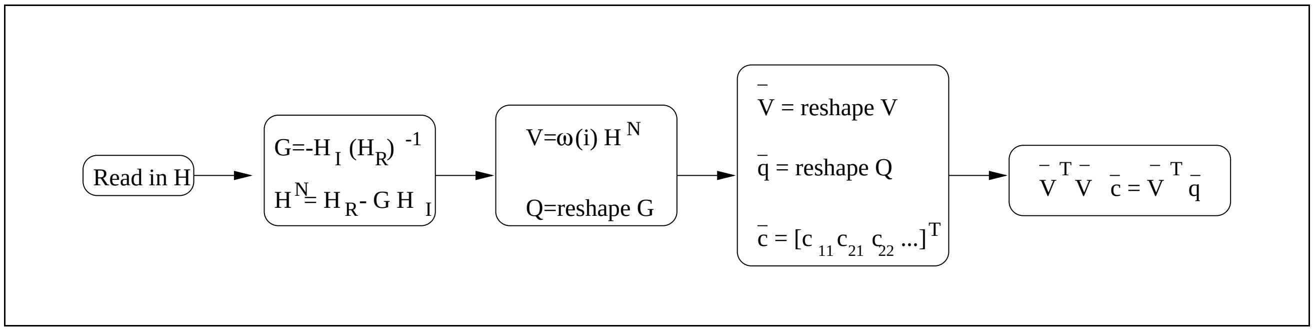

Figure 1: Least Squares Method

Since the complex transfer function ($\mathbf{H}$) is measurable, it is considered known information for this method. This method works perfectly when no noise is present. When noise is present, it is necessary to sweep through several of the frequencies in the range of interest to ensure accurate results. In doing so, the computational requirements of the problem increase dramatically. The size of the matrices involved in solving this problem are as large as $mn^2 \times n(n+1)/2$, where n is the degrees of freedom and m is the number of frequencies used in the sweep. This translates to a matrix of $O(mn^4)$ elements, which can quickly fill a computers memory when n is increased.

5 Iterative Method

Lancaster (1961) presented a set of equations that was later revised and developed by Pilkey and Inman (1997) that can solve for the damping matrix given only the mass, complex eigenvalues and eigenvectors.

The method is as follows: The first step involves guessing an initial damping matrix ($\mathbf{C_0}$). This can be any reasonable guess, such as the identity matrix. Next the eigenvectors are normalized using

$$\phi_{\mathbf{i}}{}'(\mathbf{2M}\lambda_{\mathbf{i}} + \mathbf{C_0})\phi_{\mathbf{i}} = \mathbf{1} \tag{2}$$

The damping matrix ($\mathbf{C}$) is then solved for using the following equation

$$\mathbf{C} = -\mathbf{M}(\mathbf{\Phi\Lambda^2\Phi'} + \mathbf{\bar{\Phi}\bar{\Lambda}^2\bar{\Phi}'})\mathbf{M} \tag{3}$$

where $\mathbf{M}$ is the mass matrix, $\mathbf{\Phi}$ is the matrix of eigenvectors, and $\mathbf{\Lambda}$ is the diagonal matrix of eigenvalues. Since the initial guess for $\mathbf{C}$ is not going to match the new value of $\mathbf{C}$, it is necessary to iterate. In the next iteration, the eigenvectors are again normalized, this time using the initial mass matrix and the updated damping matrix.

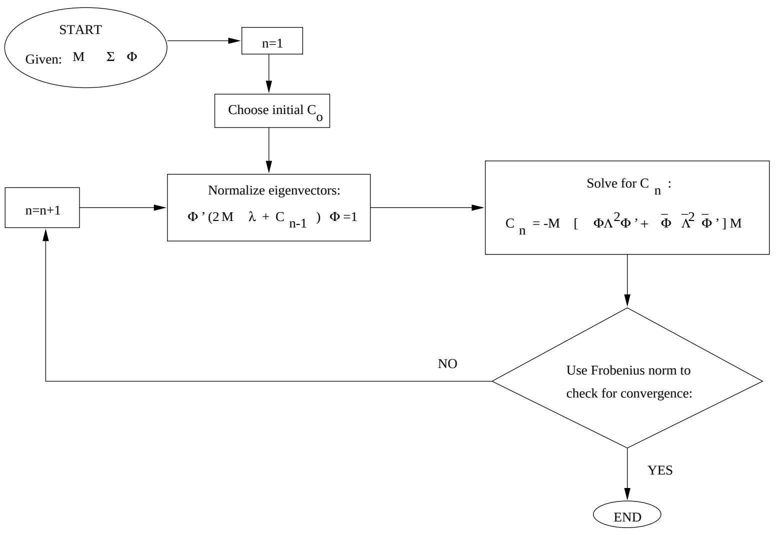

Figure 2: Iterative Procedure

$$\boldsymbol{\phi_i}'(\mathbf{2M}\boldsymbol{\lambda_i} + \mathbf{C_1})\boldsymbol{\phi_i} = \mathbf{1} \tag{4}$$

The damping is again calculated using equation 3. The iterative procedure continues using the updated damping matrix each time to normalize the eigenvectors until the difference between the Frobenius norm of successive damping matrices is small enough to declare convergence.

6 Problem

The problem investigated is shown in Figure 3. It is an n-degree-of-freedom lumped mass model. The equation of motion for this system is equation 1. **M** is a diagonal mass matrix whose elements, $m_1, m_2, ...$ are all 10. The values of k_i are all 1 and the values of c_i are all 0.1 except for c_1, which is 0.5. The matrix **C** is formed using these values as illustrated below.

$$\begin{bmatrix} c_1+c_2 & -c_2 & 0 & 0 & ... & 0 \\ -c_2 & c_2+c_3 & -c_3 & 0 & ... & 0 \\ 0 & -c_3 & c_3+c_4 & -c_4 & ... & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & 0 & -c_{n-1} & c_{n-1}+c_n & -c_n \\ 0 & 0 & 0 & 0 & -c_n & c_n \end{bmatrix} \tag{5}$$

The **K** matrix is in the same form as the **C** matrix. The **C** and **K** matrix are tridiagonal for this example, but this code was written to handle damping and stiffness matrices that have different structures. this means that the code will be performing full matrix calculations even though this example only uses tridiagonal matrices for damping and stiffness.

Both methods were tested with and without noise. Noise can be expected when experimental measurements are taken, and comes from the instrumentation. Instruments cause noise when they process

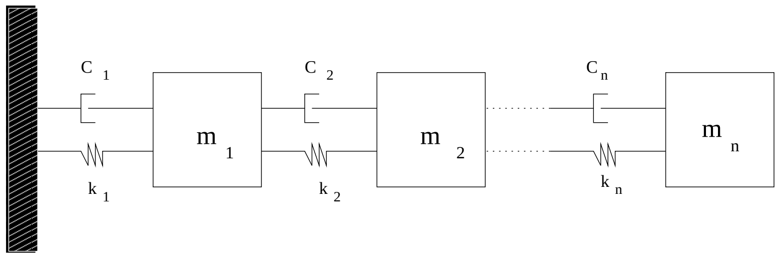

Figure 3: n degree-of-freedom lumped mass system

or average the frequency response function. The purpose of this paper is not to study noise, but to see how computational aspects of these methods deal with real situations. A problem arose with adding equal amounts of noise to both methods because the methods require different types of data. While the least squares method requires the frequency response function, the iterative method uses estimates from the frequency response function. Since it is not possible to calculate the complex eigensystem directly from the frequency response function, noise was simulated by adding 1 percent normally distributed random error directly to the eigensystem for this case. In the case of the least squares method, white noise could be added directly to the transfer function. The error attributed to noise is formed by generating a normally distributed random matrix the size of the transfer matrix that is scaled to have a 2-norm that is 1 percent of the 2-norm of the original transfer function matrix. The transfer matrix and the error matrix are added to form a slightly perturbed transfer function. This is illustrated in figure 4.

7 Parallelization

Both the Iterative and the Least Square methods were written in Fortran 90. High Performance Fortran (HPF) was considered a good choice since it required only a few high level directives to be added to aid the compiler in parallelizing the code.

7.1 Using HPF - Iterative Method

A sequential Fortran 90 code was written for the iterative method. To parallelize it, the sequential code was modified by adding HPF directives. This code was then tested to see which distribution achieved the best performance (Pilkey and Inman, 1997). The best performance was achieved when the matrix column dimension was distributed over the processors. Only this distribution was used for the final simulation.

In the sequential code, over 95% of the computational time was spent on dense matrix multiplication. Therefore, successful parallelization of the code required a significant decrease in execution time of the matrix multiplication operation. Matrix multiplication is an intrinsic function in HPF and is most likely optimized reasonably well. Although it is possible that hand coded message passing could have achieved slightly better performance, the one line matrix multiplication command in HPF/Fortran 90 was considered a good tradeoff.

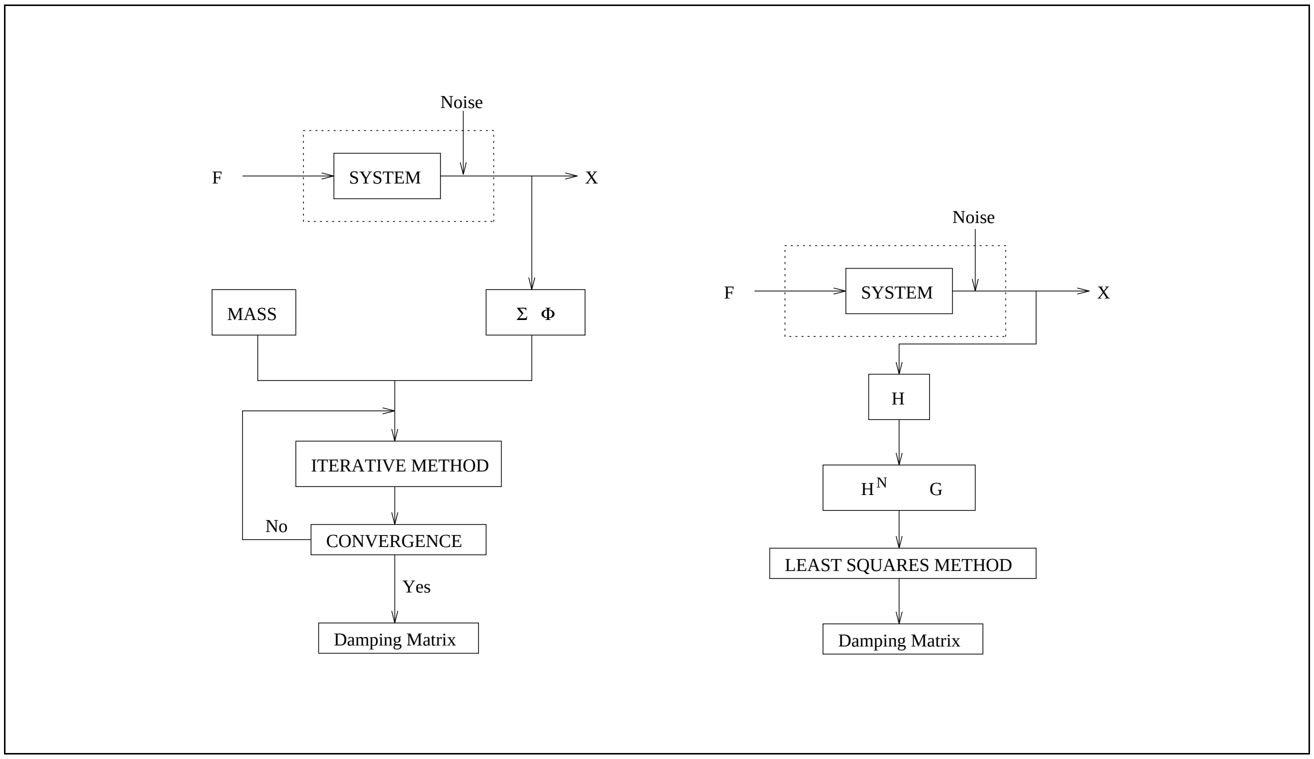

Figure 4: Noise Model for Iterative and Least Square Method

7.2 Using HPF - Least Squares Method

HPF directives were added to the least squares method described. No other modifications were made to the code. Due to the nature of the problem, the least squares method requires extensive communication between processors. Because of this, no parallelization was considered since the benefits, if any, would be small. Also, the code's parallelization would not allow a much larger degree-of-freedom problem to be considered even with the additional memory available with multiple processors, since the growth of the problem size is of $O(mn^4)$.

8 Results

Both methods are tested and timed for the problem using the IBM SP-2 at NASA Ames Research Center. The maximum problem size is determined for each method based on the memory of the majority of IBM SP-2 nodes (256 Megabytes). The performance of each method for different problem sizes is examined. Also, the effect of noise on the performance and accuracy of each method is examined.

8.1 Problem Size - Memory Requirements

The typical node on the IBM SP-2 at NASA Ames Research Center has 256 Megabytes of memory. This is the limiting factor for each method and determines how effectively the machine's memory is utilized. The least squares method use of memory increases at a rather rapid rate ($O(mn^4)$). Because of this, a 35 degree-of-freedom problem fills the machine's memory and larger problems can not be tested without utilizing the memory of multiple processors. However, this will only allow slightly larger problem sizes to be tested because the amount of memory available will grow linearly with the number of processors used, while the amount of memory required will grow at the rate $O(mn^4)$.

The iterative method is more modest in its use of memory. Problem sizes over 500 degrees of freedom can be accommodated since the memory required will grow on the $O(n^2)$. The slower increase in memory usage allows problems with a high number of degrees of freedom to be run on multiple processors.

8.2 Timing

Certain conventions were used in timing the results. First, 5-10 timings were taken for each test to get an average since there were fluctuations in the timings on the IBM SP-2. Second, only the actual computation was timed; reading the data files is not included in the timings. This was omitted because it is inherently sequential and we have not currently examined parallel I/O. Future work may include parallel I/O since the amount of time to read 500+ degree of freedom data sets can become significant. Lastly, a fixed number of iterations was used to judge parallel speedup. Although the problem may have converged prior to the fixed number of iterations, the time required to reach the iteration number appears in 2.

8.3 Performance

The problem was tested for many different degrees of freedom using both methods. Tests conducted on the IBM SP-2 using sequential Fortran 90 are presented in Table 1 and Figure 5. Note that parallel speedup refers to the p processor HPF time relative to the one processor HPF time. Sequential speedup refers to the p processor HPF time compared to the sequential Fortran 90 time. The results show that the least squares method is competitive with the iterative method up to 20 degrees of freedom. After that, the execution time increases at a much higher rate than the iterative method until it no longer fits in memory. The maximum speedup obtained (Table 2) for: $n = 100$ is 1.70 using 4 processors, $n = 250$ is 3.79 for 8 processors, and $n = 500$ is 6.36 for 16 processors.

8.4 Error Associated with Noise

Calculations for accuracy and convergence were made using the Frobenius norm, which is defined in equation 6.

$$\|\mathbf{C}\|_F = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n}|c_{ij}|^2} \tag{6}$$

With this measure, both routines presented negligible error without noise added to the system. When 1 percent noise is added (Table 3), the deviation from the exact known Frobenius norm for the iterative method ranges from 0.52% (for $n = 500$) to 0.83% (for $n = 20$). The least squares routine did not produce acceptable results when 1 percent noise was added to this problem for more than 5 degrees of freedom. Notice that there was very little change in execution time when the system with noise was tested. Because of this, the parallel HPF timings were almost identical to the system without noise. In addition, when the least squares method was applied, the memory requirement increased. This is due to the number (m) of additional frequencies needed for accurate results. Correspondingly, only problems with lower degrees of freedom are able to fit in the computer's memory.

9 Conclusions

Two methods for damping identification of an n degree-of-freedom lumped mass model have been examined. The least squares method has been shown to obtain the damping matrix for problems with

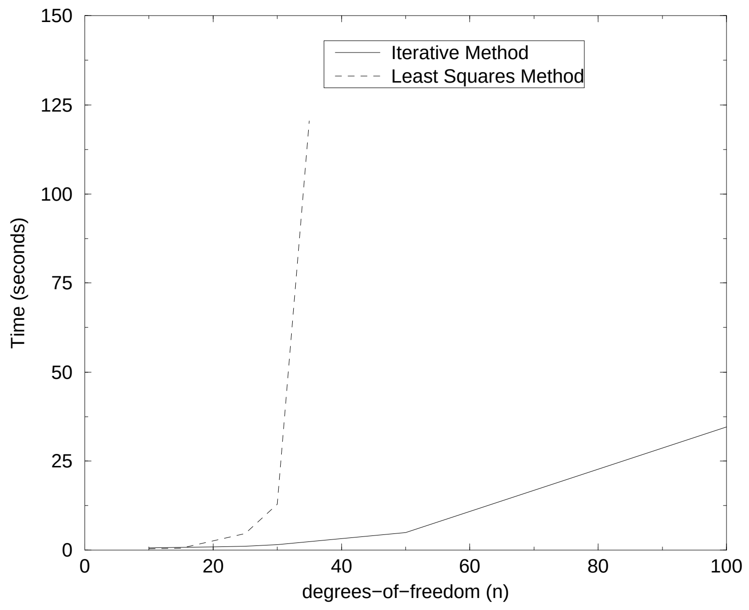

Figure 5: Time for one processor to complete multiple degree of freedom problem

n	Exact	LS Method		Iterative Method	
	F-Norm	time(s)	F-Norm	time(s)	F-Norm
10	0.72973	0.36	0.72973	0.75	0.72973
15	0.91241	0.60	0.91241	0.88	0.91241
20	1.06419	2.61	1.06419	0.93	1.06419
25	1.19687	4.76	1.19687	1.06	1.19687
30	1.31625	12.92	1.31624	1.21	1.31624
35	1.42566	120.58	1.42566	1.48	1.42566
50	1.71245	Memory Exceeded		2.59	1.71246
100	2.43567	Memory Exceeded		16.08	2.43568
250	3.86426	Memory Exceeded		243.32	3.86427
500	5.47107	Memory Exceeded		1901.92	5.47110

Table 1: Problem tested with no noise on a single processor using Fortran 90

problem size, $n = 100$				
procs	time(s)	F-Norm	parallel speedup	sequential speedup
1	19.53	2.43568	1.00	0.82
2	12.22	2.43568	1.60	1.32
4	9.44	2.43568	2.07	1.70
8	11.26	2.43568	1.73	1.43
problem size, $n = 250$				
procs	time(s)	F-Norm	parallel speedup	sequential speedup
1	286.53	3.86427	1.00	0.85
2	152.36	3.86427	1.88	1.59
4	87.59	3.86427	3.27	2.78
8	64.18	3.86427	4.46	3.79
16	73.47	3.86427	3.90	3.31
problem size, $n = 500$				
procs	time(s)	F-Norm	parallel speedup	sequential speedup
1	2267.43	5.47106	1.00	0.83
2	1134.71	5.47106	2.00	1.68
4	611.52	5.47106	3.71	3.11
8	369.09	5.47106	6.14	5.15
16	298.58	5.47106	7.59	6.36
32	440.14	5.47106	5.15	4.32

Table 2: Problem tested with no noise using High Performance Fortran

n	Exact	Iterative Method		
	F-Norm	time(s)	F-Norm	Percent Error
10	0.72973	0.78	0.73492	0.71
15	0.91241	0.90	0.91756	0.56
20	1.06419	0.96	1.07301	0.83
25	1.19687	1.05	1.20535	0.71
30	1.31625	1.21	1.32442	0.62
35	1.42566	1.52	1.43411	0.59
50	1.71245	2.81	1.72318	0.63
100	2.43567	16.12	2.44996	0.59
250	3.86426	243.41	3.88629	0.57
500	5.47107	1901.05	5.49944	0.52

Table 3: Problem tested with 1% noise using Fortran 90

limited numbers of degrees-of-freedom. The least squares method is restrictive because of its large computer memory requirements. When 1% noise is added to the system, the least squares method does not produce results with an acceptable amount of error for problems with many degrees of freedom. The iterative method is able to obtain the damping matrix for much larger problems. When 1% noise is added to the system, the memory requirement of this method does not change as in the least squares method. In addition, a reasonable level of accuracy is maintained when noise is added into the system. The iterative method also runs with a much lower execution time. The only down side to the iterative method is that it may be more difficult to apply because it requires knowledge of the complex eigenvalues and eigenvectors, while the least squares method only requires knowledge of the complex transfer function.

High performance computing, specifically parallel computing, may be beneficial to the above methods. For the least squares method, a larger degree-of-freedom problem may be examined. However, because of the rapidly increasing memory requirement of this method the benefit of parallelization may be marginal. The benefit of using parallel processors with the iterative method is much greater for two reasons. If data is distributed among multiple processors, then much larger degree-of-freedom problems may be examined. This means that problems with over 1000 degrees of freedom may be testable. In the future, methods should be devised and examined for scalability to ensure that realistic problems can be solved.

10 Acknowledgments

This work was done using the IBM SP-2 at NASA Langley Research Center and NASA Ames Research Center.

The authors wish to acknowledge the support of the Virginia/ICASE/LaRC program in High Performance Computing. Support was also provided to the first author by the Virginia Space Grant Consortium, and the first and last author by the Army Research Office

References

[1] Caravani, P., Thomson, W. T., 1974, "Identification of Damping Coefficients in Multidimensional Linear Systems", *ASME Journal of Applied Mechanics*, Vol. 41, pp 379-382.

[2] Chen, S. Y., and Ju, M. S., and Tsuei, Y. G., 1996, "Estimation of Mass, Stiffness and Damping Matrices from Frequency Response Functions", *Journal of Vibration and Acoustics*, Vol. 118, pp. 78-82.

[3] Fritzen, C.-P., 1986, "Identification of Mass, Damping, and Stiffness Matrices of Mechanical Systems", *ASME Journal of Vibration, Acoustics, Stress, and Reliability in Design*, Vol. 108, pp. 9-16.

[4] Hasselman, T.K., 1972, "A Method of Constructing a Full Modal Damping Matrix from Experimental Measurements", *AIAA Journal*, Vol. 10, pp. 526-527.

[5] High Performance FORTRAN Forum. *High Performance FORTRAN Language Specification, Version 2.0*, January 1997.

[6] Koelbel, C.H., Loveman, D.B., Schreiber, R.S., Steele, G.L., Zosel, M.E., **The High Performance FORTRAN Handbook**, The MIT Press, Cambridge, 1994.

[7] Lancaster, P. 1961, "Expression for Damping Matrices in Linear Vibration Problem", *Journal of the Aerospace Sciences*. pg. 256.

[8] Pilkey, D. F., Inman, D. J., 1997, "An Iterative Approach to Viscous Damping Matrix Identification", *Proceedings of the 15th International Modal Analysis Conference*,

[9] Roemer, M. J., and Mook, D. J., 1992, "Mass, Stiffness, and Damping Matrix Identification: An Integrated Approach", *ASME Journal of Vibration and Acoustics*, Vol. 114, pp. 358-363.