1997

Inversion of sparse matrices using Monte Carlo methods

Bassirou Chitou
Iowa State University

Follow this and additional works at: https://lib.dr.iastate.edu/rtd
Part of the Statistics and Probability Commons

Recommended Citation

This Dissertation is brought to you for free and open access by the Iowa State University Capstones, Theses and Dissertations at Iowa State University Digital Repository. It has been accepted for inclusion in Retrospective Theses and Dissertations by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.
INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.
Inversion of sparse matrices using Monte Carlo methods

by

Bassirou Chitou

A dissertation submitted to the graduate faculty
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Statistics
Major Professor: Dr. Mervyn Marasinghe

Iowa State University
Ames, Iowa
1997

Copyright © Bassirou Chitou, 1997. All rights reserved.
Graduate College
Iowa State University

This is to certify that the Ph.D. dissertation of

Bassirou Chitou

has met the dissertation requirements of Iowa State University

Signature was redacted for privacy.

Major Professor
Signature was redacted for privacy.

For the Major Program
Signature was redacted for privacy.

For the Graduate College
# TABLE OF CONTENTS

1 MOTIVATION AND LITERATURE REVIEW .................................. 1
   1.1 Definition and Notation ........................................... 1
   1.2 Motivation ...................................................... 4
   1.3 Survey of the Literature ....................................... 5
      1.3.1 Dense Matrix Algorithms ................................. 6
      1.3.2 Direct Methods ........................................... 6
      1.3.3 Iterative Methods ....................................... 17
   1.4 Focus of this Dissertation .................................... 22

2 CONDITIONING, UNBIASED ESTIMATORS, AND SWINDLE DECOMPOSITIONS ..................................................... 23
   2.1 Conditioning and Pseudo Unbiased Estimators ............ 23
      2.1.1 Conditional Swindle ................................... 25
      2.1.2 Pseudo Unbiased Estimators ............................ 28
   2.2 Swindle Decomposition ...................................... 29
   2.3 Role of Positive Definiteness and Sparsity of $W$ .... 31
   2.4 Extension to a Sub-matrix of $V$ ............................ 32

3 APPLICATION OF IMPORTANCE SAMPLING TO EVALUATE AN INTEGRAL .......................................................... 35
   3.1 Goal ............................................................ 35
   3.2 Earlier Work .................................................. 36
3.2.1 Gibbs Sampler For Matrix Inversion: David Harville (1996) ........... 36
3.3 The Importance Sampling Approach .................................................. 38
  3.3.1 The Method ................................................................................. 38
3.4 Choice Of The Importance Function .................................................. 42
  3.4.1 The Optimal Sampler ................................................................ 42
  3.4.2 Mixture Sampling ...................................................................... 44
3.5 Example and Evaluation of an Optimal Sampler .............................. 45
  3.5.1 Materials and Method ................................................................. 45
  3.5.2 Evaluation of the Importance Weights ....................................... 46
  3.5.3 Simulation Results and Interpretations ...................................... 47
3.6 Summary ......................................................................................... 50
3.7 Estimation of the Square Root of a Determinant .......................... 52

4 AN ADAPTIVE IMPORTANCE SAMPLING ALGORITHM .................... 54
  4.1 The General Algorithm ................................................................. 54
    4.1.1 Matching h and f .................................................................. 55
    4.1.2 Adaptive Importance Sampling .............................................. 56
  4.2 Estimation of the Elements of V using A.I.S ................................. 56
    4.2.1 Background Materials ............................................................ 56
    4.2.2 The Algorithm ................................................................. 58
  4.3 Example and Numerical Comparisons ......................................... 61
    4.3.1 Effect of the Starting Covariance Matrix ............................... 62
    4.3.2 Importance Sampling versus Adaptive Importance Sampling .... 63
  4.4 Summary and Discussion .............................................................. 64

5 SPARSE STORAGE, PARALLEL PROCESSING, AND APPLICATION ...... 67
  5.1 The Problem ................................................................................. 67
5.2 The Input Storage Scheme .................................. 68
  5.2.1 The Scheme .......................................... 68
  5.2.2 Advantages of the Input Storage Scheme .............. 69
  5.2.3 Disadvantages of the Input Storage Scheme .......... 70
5.3 Row Pointer/Column Index Scheme .......................... 70
  5.3.1 The Scheme .......................................... 70
  5.3.2 Advantages of the Row Pointer/Column Index Scheme .. 73
5.4 Other Storage Schemes and References ...................... 74
5.5 Application to the Tribolium Data ......................... 74
  5.5.1 Description of the Data and the Model ............... 75
  5.5.2 A and its Inverse .................................... 76
6 PARALLEL IMPLEMENTATION OF A.I.S. ....................... 78
  6.1 Adaptive Importance Sampling is Parallelizable ........ 79
  6.2 The Distributed Memory Environment .................... 80
    6.2.1 Interprocessor Communication and Load Balancing .... 80
    6.2.2 Data Structures .................................... 82
    6.2.3 Efficiency and Speed-up ............................ 83
    6.2.4 Implementation of A.I.S. on MIMD-DM ............... 83
  6.3 The Shared Memory Environment ......................... 85
    6.3.1 Description of the Shared Memory Environment ...... 85
    6.3.2 Implementation of A.I.S. under the Shared Memory Environment 88
7 CONCLUSION AND DISCUSSION ................................ 90
APPENDIX A FORTRAN CODE FOR THE IMPORTANCE SAMPLING: SPARSE FORMAT ................................. 93
APPENDIX B FORTRAN CODE FOR THE ADAPTIVE IMPORTANCE SAMPLING ........................................ 105
APPENDIX C  PVM CODE FOR PARALLEL IMPLEMENTATION OF AIS .................. 127
APPENDIX D  TEMPLATE FOR PARALLEL IMPLEMENTATION OF AIS ON THE DEC8400 .................. 129
BIBLIOGRAPHY .............................................. 131
ACKNOWLEDGMENTS ....................................... 136
# LIST OF TABLES

Table 3.1  Comparison of the Performance of 3 Importance Samplers: Nonzero Elements .................................................. 51

Table 4.1  Comparison of Starting Covariance Matrices: Diagonal Elements 65

Table 4.2  Comparison of Starting Covariance Matrices: Off diagonal Elements ............................................................... 65

Table 4.3  A.I.S. Estimates of Inverse of $W^*$: Nonzero Elements ................. 66

Table 5.1  Input Storage Scheme for the Matrix in 5.4 .............................. 69

Table 5.2  Row Pointer Scheme of the Matrix in 5.4 ................................ 71

Table 5.3  First Row of the $A^{-1}$ ........................................................ 77

Table 6.1  Efficiency and Speed-Up of the Parallel Implementation of A.I.S. in a Distributed Memory Environment ....................... 85

Table 6.2  Efficiency and Speed-Up of the Parallel Implementation of A.I.S. in a Distributed Memory Environment ....................... 89
LIST OF FIGURES

Figure 3.1  Histogram of log of Importance weights from MVN with covariance matrix \text{Diag}(1/w(i, i)) \hspace{1cm} 48

Figure 3.2  Histogram of log of importance weights from \( t_3 \) with covariance matrix \text{Diag}(1/w(i, i)) \hspace{1cm} 48

Figure 3.3  Histogram of log of importance weights from MVN with covariance matrix \text{Diag}(1/w(i, i) + 1/(w(i, i) * 2)) \hspace{1cm} 49

Figure 3.4  Histogram log of importance weights from \( t_3 \) with covariance matrix \text{Diag}(1/w(i, i) + 1/(w(i, i) * 2)) \hspace{1cm} 49
This dissertation is concerned with *large, sparse, and positive definite* matrices. The goal is to have the flexibility to compute *desired elements* of the *inverse* of such matrices, or some linear combination of these elements, without actually inverting the whole matrix. This chapter reviews the *traditional methods* available for the computation of the inverse of large sparse positive definite matrices and the *necessity for improving* these methods. However, before proceeding, let us define some terms associated with sparse matrix computations.

### 1.1 Definition and Notation

**Definition 1:**

An $n \times n$ matrix $W$ is said to be *large* if the number of rows in $W$ exceeds 30.

**Definition 2:**

A $W$ is said to be *sparse* if the percentage of nonzero entries in $W$ is less than 40 (see Evans [10]).

**Definition 3:**

An $n \times n$ matrix $W$ is *positive definite* (p.d.) if:

$$x^T W x > 0$$

for all nonzero $x \in \mathbb{R}^n$ (see Golub and Van Loan [21]).
Definition 4:

The condition number of the matrix \( W \) is defined as the product of the magnitude of \( W \) and its inverse and denoted by,

\[
\kappa(W) = \|W\| \cdot \|W^{-1}\| \tag{1.2}
\]

where, \( \| \cdot \| \) is a matrix norm.

Definition 5:

A matrix \( W \) is well conditioned if \( \kappa(W) \) is of modest size. On the other hand, if \( \kappa(W) \) is large, then \( W \) is said to be ill-conditioned and any numerical solution of \( Wx = b \) must be accepted with skepticism (see Kincaid and Cheney [27]).

Large, sparse, and p.d. matrices arise in diverse disciplines (see Grimes et al. [8]). These include partial differential equations, structural analysis, circuit analysis, least squares, mixed model equations, testing of hypotheses, econometrics, oceanography, and air traffic control.

Definition 6:

A statistical model that will be used is the so-called Mixed Model which is defined as:

\[
y = Xb + Za + e, \tag{1.3}
\]

where

\[
y = \text{an } n \times 1 \text{ vector of } n \text{ observations} \\
b = \text{an } N_f \times 1 \text{ vector of fixed effects} \\
X = \text{an } n \times N_f \text{ design matrix for fixed effect} \\
a = \text{an } N_a \times 1 \text{ vector of random effects} \\
Z = \text{an } n \times N_a \text{ design matrix for random effects, and} \\
e = \text{an } n \times 1 \text{ vector of residuals}
\]
with

\[ \text{var}(a) = G = \sigma_a^2 A, \]

where A is called the relationship matrix,

\[ \text{var}(e) = \sigma_e^2 I \]

and finally

\[ \text{cov}(a, e) = 0. \]

**Definition 7**

To obtain estimates of the unknown parameters \( \beta \) and \( \sigma_a^2 \), a set of linear equations called *Henderson's Mixed Model Equations* (MME) must be solved. The MME are given by:

\[
\begin{bmatrix}
X^T X & X^T Z \\
Z^T X & Z^T Z + G^{-1} \\
\end{bmatrix}
\begin{bmatrix}
\hat{b} \\
\hat{a} \\
\end{bmatrix}
= 
\begin{bmatrix}
X^T y \\
Z^T y \\
\end{bmatrix}
\tag{1.4}
\]

The following notations will be used throughout this document:

1. A matrix will be denoted by **BOLD UPPER CASE** letter, such as, W.
2. A vector will be denoted by **bold lower case** letter, such as, u.
3. \( u_i \), will denote the ith element of the vector u, and \( w_{ij} \) the ijth entry in W
4. \( E_x(g) \) will denote the expectation of the integrable function g under the distribution of x
1.2 Motivation

In many scientific applications, the evaluation of some entries of the inverse of a large, sparse, and p.d. matrix, or a linear combination of those entries, becomes vital. For example in the analysis of short circuit current, the bus impedance matrix must be computed (see Fagan et al.[37]). The impedance is the total opposition to current flow in an alternating current circuit. The bus impedance matrix is the inverse of the bus admittance matrix. In most all applications, the bus admittance matrix is large, sparse, symmetric and p.d. In testing the null hypothesis that the covariance matrix of a multivariate population is a diagonal matrix, the likelihood ratio statistic is the $\frac{1}{2}$th power of the determinant of the correlation matrix (see Morrison [33]). In least squares estimation for linear models, the inverse of the coefficient matrix of the normal equations must be evaluated in order to obtain the standard errors of the least squares estimators and the correlations between these estimators. In REstricted Maximum Likelihood (REML) estimation of variance components, two widely used optimization procedures are the Expectation-Maximization (EM) and the Derivative Free (DF) algorithms. The EM algorithm requires the calculation of the trace of the inverse of the coefficient matrix of the MME. The DF algorithm requires the evaluation of the determinant of the same matrix. Let us illustrate this with an example.

Example 1: Evaluation of the Trace of the Covariance Matrix in the EM Algorithm:

Let $\mathbf{W}$ be the coefficient matrix of the MME, and let $\mathbf{V}$ be its inverse. Let $\mathbf{a}$ denote the random vector of this model, $q(\mathbf{a})$ the number of levels of $\mathbf{a}$, $C^a$ the inverse of the covariance matrix of $\mathbf{a}$, and $V^{aa}$ be the block of the inverse matrix $\mathbf{V}$ corresponding to $\mathbf{a}$. The goal in EM is to evaluate:

$$
tr(\mathbf{V}^{aa}C^a) = \sum_i v_{ii}^{aa}c_{ii}^a + 2 \sum_i \sum_{j>i} v_{ij}^{aa}c_{ij}^a \quad i, j = 1, \ldots, q(\mathbf{a})
$$

(1.5)
As we can see in 1.5, the evaluation of the \( \text{tr}(V^a C^a) \) requires only the computation of the elements of \( V \) that correspond to \textbf{nonzero} elements of \( C^a \).

Similarly, the determinant needed in DF algorithm is computed as the determinant of the diagonal matrix obtained by the Cholesky factorization of \( W \).

In the Bayesian analysis of the Mixed Model (see Lin [29]), the evaluation of the \textit{posterior conditional distribution} of the random vector \( a \) requires the \textit{inversion} of the matrix:

\[
Z^TZ + A^{-1} \frac{\sigma^2_e}{\sigma^2_a}
\]  

(1.6)

Where \( Z \) is the \( N \times N_a \) design matrix for random effects, and \( A \) is the relationship matrix. The matrix \( A \) is usually large, sparse, and p.d.

In many statistical applications, the coefficient matrix of the MME is usually \textit{large}, \textit{sparse}, \textit{symmetric} and \textit{p.d.} In animal breeding applications, the order of the coefficient matrix of the MME can range from 4,000 to 1,000,000 (see Mistzal [32] and Harville [24]).

With the coefficient matrix of the MME being so large and sparse, \textit{special methods} are required for the \textit{efficient} computation of its inverse and its \textit{determinant}. Let us first survey the literature on the available methodologies.

### 1.3 Survey of the Literature

Let \( W \) be a sparse and p.d. matrix. The \textit{inverse} of \( W \) is defined as the matrix \( V \) such that:

\[
WV = WV = I.
\]  

(1.7)

That is the \( i \)th column of \( V \), say \( v_i = (v_{i1}, \ldots, v_{in})^T \) is the solution of the system of equations:

\[
Wv_i = e_i, \quad i = 1, \ldots, n
\]  

(1.8)
where \( e_i \) is the elementary vector:

\[
e_i = (0, 0, \ldots, 1, 0, \ldots, 0)^T
\]  

(1.9)

with 1 in the \( i \)th position.

(1.10)

1.3.1 Dense Matrix Algorithms

The computation of the inverse of a sparse matrix can be done using *dense matrix algorithms* such as the Single-Division method, the Partitioning method, the Bordering method, the Escalator method. V.N. Faddeeva (see [14]) gives an excellent presentation of dense methods. However, dense inverters have *two shortcomings*:

1. They do not take advantage of the *sparsity* of the matrix \( W \).

2. They are *computationally expensive*. For example, the number of multiplications and divisions required by the single-division method is \( \frac{5}{3}(n^2 + 3n - 1) \) (see [14]).

Usually, the inverse and/or the determinant of a sparse matrix can be computed either by *direct* methods or by *iterative* techniques.

1.3.2 Direct Methods

Nowadays, the evaluation of the determinant and the inversion of a large, sparse, and p.d. matrix is handled successfully by *direct methods*. Two widely used direct methods are: *the Cholesky factorization* and the *Sparse Inverse* of Takahashi et al. (see [37]).
1.3.2.1 Cholesky Factorization

The Cholesky factorization theorem states that every symmetric, p.d. matrix \( W \) in \( \mathbb{R}^{n \times n} \) has a unique decomposition:

\[
W = LL^T
\]  

(1.11)

where \( L \) is a lower triangular matrix in \( \mathbb{R}^{n \times n} \) with positive diagonal entries (see Golub and Van Loan [21]). So that, \( V \), the inverse of \( W \), is obtained by solving the following set of equations:

\[
LL^T V = I
\]  

(1.12)

where \( I \) is the identity matrix having the same dimension as \( W \).

Specifically, \( V \) is obtained by the following algorithm:

1. First, obtain the Cholesky factor \( L \) as follows (see Kincaid and Cheney in [27])

   (a) **Input** \( n, (w_{ij}) \)

   (b) **for** \( k = 1, \ldots, n \) **do**

      \[
l_{kk} \leftarrow \left( w_{kk} - \sum_{s=1}^{k-1} l_{ks}^2 \right)^{1/2}
      \]  

      (1.13)

      **for** \( i = k + 1, k + 2, \ldots, n \) **do**

      \[
l_{ik} \leftarrow \frac{w_{ik} - \sum_{s=1}^{k-1} l_{ks} l_{is}}{l_{kk}}
      \]  

      (1.14)

   **end**

   **end**

   **output** \( (l_{ij}) \)
2. Solve

\[ Ls_i = e_i \quad i = 1, \ldots, n \]  \hspace{1cm} (1.15)

3. Solve

\[ L^T v_i = s_i \quad i = 1, \ldots, n \]  \hspace{1cm} (1.16)

where \( v_i \) and \( e_i \) are as defined in Section 3. The above equations are easy to solve since \( L \) is a triangular matrix.

Example 2: Illustration of the Cholesky Factorization:

An example will make the above algorithm clear. Suppose we would like to compute \( v_5 \) the 5\(^{th} \) column of the inverse of \( W \), where \( W \) is given by:

\[
W = \begin{bmatrix}
4 & 1 & 2 & 1 & 2 \\
1 & 5 & 0 & 0 & 0 \\
2 & 0 & 3 & 0 & 0 \\
1 & 0 & 0 & 5 & 0 \\
2 & 0 & 0 & 0 & 8 \\
\end{bmatrix} \hspace{1cm} (1.17)
\]

Using the algorithm given above, we have

\[
L = \begin{bmatrix}
2.0000 \\
0.5000 & 2.1794 \\
1.0000 & -0.2294 & 1.3955 \\
0.5000 & -0.1147 & -0.3772 & 2.1435 \\
1.0000 & -0.2294 & -0.7543 & -0.3783 & 2.4971 \\
\end{bmatrix} \hspace{1cm} (1.18)
\]

Next, solve by forward substitution:
1. \[ \mathbf{Ls}_5 = \mathbf{e}_5 \] (1.19) to get
\[ \mathbf{s}_5 = (0, 0, 0, 0, .4005)^T. \] (1.20)

2. Finally, solve
\[ \mathbf{L}^T \mathbf{v}_5 = \mathbf{s}_5 \] (1.21) by backward substitution to get
\[ \mathbf{v}_5 = (-.1415, .0283, .0943, .0283, .1604)^T. \] (1.22)

1.3.2.2 The Fill-Ins Problem

The above example also illustrates the common drawback of the Cholesky factorization. That is, the Cholesky factor \( \mathbf{L} \) has nonzero entries in locations that are zeros in the lower triangular part of \( \mathbf{W} \). Those nonzero entries in \( \mathbf{L} \) are called fill-ins. For example, \( l_{32}, l_{42}, l_{43}, l_{52}, l_{53}, \) and \( l_{54} \) are all fill-ins.

There are two reasons for the occurrence of fill-ins. Firstly, as explained by George and Liu (see [18]), fill-ins are caused by the outer product scheme used in the computation of \( \mathbf{L} \). Specifically, the Cholesky factor \( \mathbf{L} \) can be obtained as follows. First, write \( \mathbf{W} \) as:
\[ \mathbf{W} = \begin{bmatrix} w_{11} & w_{21}^T \\ w_{21} & \mathbf{W}_{22} \end{bmatrix} \] (1.23)

Next, rewrite \( \mathbf{W} \) as:
\[ \mathbf{W} = \begin{bmatrix} \sqrt{w_{11}} & 0 \\ s & \mathbf{I}_{n-1} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & \mathbf{W}_{22} - \mathbf{ss}^T \end{bmatrix} \begin{bmatrix} \sqrt{w_{11}} & 0 \\ s & \mathbf{I}_{n-1} \end{bmatrix} \] (1.24)
Where \( s = w_{11}^{-1/2} w_{21} \).

The sub-matrix

\[
\bar{W}_1 = \bar{W}_{22} - ss^T
\]  

(1.25)

may have nonzero elements in locations that are zero in \( \bar{W}_{22} \).

Example 3: Illustration of the Fill-ins Problem:

A numerical example will make this clearer. Consider \( W \) as in 1.17. Let

\[
\begin{bmatrix}
5 & 0 & 0 & 0 \\
0 & 3 & 0 & 0 \\
0 & 0 & 5 & 0 \\
0 & 0 & 0 & 8
\end{bmatrix}
\]

(1.26)

and let

\[
\begin{bmatrix}
1 \\
2 \\
1 \\
2
\end{bmatrix}
\]

(1.27)

Then, the matrix \( \bar{W}_1 \) in (1.25) is:

\[
\bar{W}_1 = \bar{W}_1 - \frac{1}{4} \begin{bmatrix}
1 \\
2 \\
1 \\
2
\end{bmatrix} \begin{bmatrix}
1 & 2 & 1 & 2
\end{bmatrix}
\]

(1.28)

or

\[
\begin{bmatrix}
\frac{19}{4} & \frac{-1}{2} & \frac{-1}{4} & \frac{-1}{2} \\
\frac{-1}{2} & 2 & \frac{-1}{2} & -1 \\
\frac{-1}{4} & \frac{-1}{2} & \frac{19}{4} & \frac{-1}{2} \\
\frac{-1}{2} & -1 & \frac{-1}{2} & 7
\end{bmatrix}
\]

(1.29)
Hence, $\bar{W}_1$ has nonzero elements in locations that are zero in $W_1$. For more details on the fill-in problem in Cholesky factorization, see [18].

Secondly, as explained by Zlatev([38]), fill-ins may also occur because "one assumes that the elements of $V$ can be calculated by exact arithmetic, and that zero elements can be computed exactly." However, on computers, calculations are performed with rounding errors, so that nonzero elements will appear where exact computation are expected to produce zero elements, and hence the occurrence of fill-ins. This fact is specifically mentioned in the Numerical Algorithm Group (NAG) subroutine for computing the Cholesky factor as follows: "The Cholesky factor $L$ is the exact factor of a perturbed matrix $W + E$, where $E$ is the matrix of fill-ins" (see under F07GDF in NAG Fortran library, Demmel [3] and Golub and Van Loan [21]). For this second type of fill-ins, [38] suggests that one attempts at some stages of the computational process to avoid both the storage of small non-zero elements and the computations with such non-zero elements. The challenge here is to decide when an element is small enough to be discarded. This will depend on the problem in hand. For more details on this approach to fill-ins, see [38].

These additional nonzero entries require more storage and more computing time to calculate (1.17), (1.19), and (1.21). In other words, occurrence of fill-ins creates additional expenses in terms of storage and computing time.

### 1.3.2.3 The Good Ordering Problem

To reduce fill-ins in $L$, many ordering algorithms have been developed. For these algorithms, the task is to find a good permutation of the rows and columns of $W$ such that the matrix $L$ has very few fill-ins. These algorithms are extensively reviewed by [18] and by [37]. The most popular of these algorithms is the minimum degree algorithm, which is used in many software packages such as SPARSPACK.
Example 4: Illustration of the Good Ordering Scheme:

Let us now, illustrate, how a good ordering can dramatically reduce fill-ins in $L$. As before, let:

$$W = \begin{bmatrix} 4 & 1 & 2 & 1 & 2 \\ 1 & 5 & 0 & 0 & 0 \\ 2 & 0 & 3 & 0 & 0 \\ 1 & 0 & 0 & 5 & 0 \\ 2 & 0 & 0 & 0 & 8 \end{bmatrix} \quad (1.30)$$

and consider the following permutation matrix:

$$P = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (1.31)$$

Then,

$$\tilde{W} = PWP^T$$

$$= \begin{bmatrix} 8 & 0 & 0 & 0 & 2 \\ 0 & 5 & 0 & 0 & 1 \\ 0 & 0 & 3 & 0 & 2 \\ 0 & 0 & 0 & 5 & 1 \\ 2 & 1 & 2 & 1 & 4 \end{bmatrix} \quad (1.32)$$
and, the Cholesky factor of $\mathbf{W}$ is:

\[
\tilde{\mathbf{L}} = \begin{bmatrix}
2.8284 & & & \\
0.0000 & 2.2361 & & \\
0.0000 & 0.0000 & 1.7321 & \\
0.0000 & 0.0000 & 0.0000 & 2.2361 \\
0.7071 & 0.4472 & 1.1547 & 0.4472 & 1.3292
\end{bmatrix}
\] (1.33)

We can now see that $\tilde{\mathbf{L}}$ is as sparse as the lower triangular part of $\mathbf{W}$, i.e., $\tilde{\mathbf{L}}$ suffers no fill-in. Hence, an appropriate choice of the permutation matrix $\mathbf{P}$ provides a substantial reduction in the number of fill-ins and hence in storage and computer time. It is worth noting that in practical applications, fill-ins are not entirely eliminated but reduced to a small number.

By applying the Cholesky algorithm to the permuted matrix $\mathbf{W}$, we get:

\[
\tilde{\mathbf{v}}_5 = (-.1415, -.1132, -.3773, -.1132, .5660)^T
\] (1.34)

and

\[
\tilde{\mathbf{v}}_1 = (.1603, .0283, .0943, .0283, -.1415)^T.
\] (1.35)

\[
\mathbf{Pv}_5 = \tilde{\mathbf{v}}_1
\] (1.36)

\[
\mathbf{Pv}_5 = (.1603, .0283, .0943, .0283, -.1415)^T
\] (1.37)

which is the same as $\tilde{\mathbf{v}}_1$. 
1.3.2.4 The Sparse Bus Impedance Matrix Method

In their 1973 paper, Takahashi, Fagan, and Chen ([37]) developed an algorithm called the sparse bus impedance matrix method. The method was developed to provide the relevant terms needed in the computation of the bus impedance matrix, that is all the diagonal entries which correspond to the nodes elements and some off-diagonal entries which correspond to the actual branch elements of a short circuit network (see [37]). The algorithm is as follows:

1. Reorder the matrix \( W \) by premultiplying it by a good permutation matrix \( P \) and post multiplying by \( P^T \)

2. Obtain a root-free Cholesky decomposition of the reordered matrix \( W \). That is, rewrite \( W \) as:

\[
W = LDU
\]  
(1.38)

(a) where \( L \) is a unit lower triangular matrix

(b) \( U = L^T \) is the transpose of \( L \)

(c) \( D \) is a diagonal matrix

3. The determinant of \( W \) is the product of diagonal elements of \( D \)

4. The inverse \( V \) satisfies the equation:

\[
V = D^{-1}L^{-1} + (I - U)V \]  
(1.39)

Specifically, the elements of \( V \) are computed as follows:

\[
v_{ii} = d_{ii}^{-1} - \sum_{k=i+1}^{n} u_{ik}v_{ik} \]  
(1.40)
and
\[ v_{ij} = -\sum_{k=j+1}^{n} u_{jk}v_{ik} \quad (1.41) \]
for \( i = n, n-1, \ldots, 1 \), and \( j = i-1, i-2, \ldots, 1 \) where \( v_{ij} \) is the \( ij \)th element of \( V \) and \( u_{ij} \) is the \( ij \)th element of \( U \).

Examining the above formulas, three advantages of the *sparse inverse* method can be seen:

1. \( v_{ij} \) is computed only if \( u_{ij} \) is nonzero and this for all \( i, j = 1, \ldots, n \)

2. Since \( V \) is symmetric, *all* of its elements can be computed *without* inverting \( L \).

3. The *nonzero* elements of \( V \) that correspond to nonzero elements of \( U \) can be computed *without* computing any other elements of \( V \) (see example 5 below). Those elements of \( V \) corresponding to nonzero elements of \( U \) are called the *sparse inverse*.

Example 5: Illustration of the Sparse Bus Impedance Method:

Consider the *ordered* matrix \( \mathbf{W} \) defined as:

\[
\mathbf{W} = \begin{bmatrix}
8 & 0 & 0 & 0 & 2 \\
0 & 5 & 0 & 0 & 1 \\
0 & 0 & 3 & 0 & 2 \\
0 & 0 & 0 & 5 & 1 \\
2 & 1 & 2 & 1 & 4 \\
\end{bmatrix}
\quad (1.42)
\]

The Cholesky factorization gives:

\[
\mathbf{U} = \begin{bmatrix}
1 & 0 & 0 & 0 & 1/4 \\
0 & 1 & 0 & 0 & 1/5 \\
0 & 0 & 1 & 0 & 2/3 \\
0 & 0 & 0 & 1 & 1/5 \\
0 & 0 & 0 & 0 & 1 \\
\end{bmatrix}
\quad (1.43)
\]
and

\[ D = \text{diag}(8, 5, 3, 5, \frac{53}{30}) \quad (1.44) \]

Therefore:

1. The determinant of \( W \) is given by:

\[ |W| = 8 \times 5 \times 3 \times 5 \times \frac{53}{30} = 1600 \quad (1.45) \]

2. The elements of the \textit{sparse inverse} of \( V \) in reverse order are computed as follows:

\[ v_{55} = \frac{1}{d_{55}} = \frac{30}{53} \quad (1.46) \]
\[ v_{45} = -u_{45}v_{55} = \frac{-6}{53} \quad (1.47) \]
\[ v_{35} = -u_{35}v_{55} = \frac{-20}{53} \quad (1.48) \]
\[ v_{25} = -u_{25}v_{55} = \frac{-6}{53} \quad (1.49) \]
\[ v_{15} = -u_{15}v_{55} = \frac{-15}{106} \quad (1.50) \]

The above results agree with the ones obtained by the Cholesky factorization of the reordered matrix. However, \textit{sparse inverse} method suffers two main drawbacks:

1. \( V \) contains \textit{more} nonzero than \( W \) because of fill-ins in \( U \) as a result of the Cholesky factorization, and hence requires \textit{more} storage space and computing time than any method without fill-ins.

2. In general, the sparse inverse of \( V \) comprises only a \textit{small fraction} of the nonzero elements of \( V \) that are of interest.
1.3.2.5 Advantages and Disadvantages of Direct Methods

At this point, let us summarize what we have learned about direct methods. Direct sparse methods have four simple and easy flowing phases:

1. Find an optimal permutation matrix $P$ that will result in a sparse factorization.

2. Set the data structure for the factor $L$, such that $PWP^T = LL^T$.

3. Compute $L$.

4. To obtain the inverse of $PWP^T$, solve

$$LL^TV = I.$$ 

However, direct sparse methods share the following 3 drawbacks:

1. Finding an optimal permutation of $W$. As mentioned by Duff et al. (1986), cited by Mistzal (see [31]), optimal permutation is excessively expensive.

2. Direct Methods require more storage and computing time than iterative methods, because of the fill-ins problem.

3. Direct Methods produce estimates that are not completely accurate due to roundoff errors (see [27]).

1.3.3 Iterative Methods

In the previous section, the advantages of developing fill-ins free methods in solving large sparse positive systems were made apparent. One way to do that is to use iterative methods. This section describes some of the most widely used iterative methods and the need for their improvement.
1.3.3.1 The General Iterative Approach

An iterative method is a simple process that is applied repeatedly to generate a sequence of vectors that ideally converge to the solution (see [27]).

Kincaid and Cheney, as well as Dongarra et al. (see [6]) describe the general approach to iterative methods as follows. Suppose we want to solve the system of equations:

$$Wv = e$$  \hspace{1cm} (1.51)

1. Then, choose a certain matrix $Q$, called the splitting matrix, and rewrite (1.51) as:

$$Qv = (Q - W)v + e.$$ \hspace{1cm} (1.52)

2. This leads to the basic iteration:

$$Qv^{(k)} = Pv^{(k-1)} + e, \quad k = 1, 2, \ldots$$ \hspace{1cm} (1.53)

where

$$P = Q - W$$ \hspace{1cm} (1.54)

and $v^{(0)}$ is an initial guess.

The splitting matrix $Q$ is also called the preconditioner for the system $Wv = e$. When $Q = \text{Diag}(w_{ii})$ then the iterative method is called the Jacobi iteration and when $Q$ is the lower triangular part of $W$, including the diagonal, the iterative method is called the Gauss-Seidel iteration. Other popular methods are the Successive OverRelaxation (SOR), the Cheyshev Semi-iterative method, and some modifications of these methods.

1.3.3.2 The Conjugate Gradient Method

For a large, symmetric, and p.d. matrix, the most widely used iterative method is the conjugate gradient method of Hestenes and Stiefel [25]. The preconditioned conjugate gradient algorithm with preconditioner $Q \in \mathbb{R}^{n \times n}$ may be described as follows:
1. **Input** \( v, W, Q, e, \delta, \) and \( \epsilon \)

2. Set \( r \leftarrow e - Wv \)

3. Solve \( Qz = r \) for \( z \).

4. Set, \( s \leftarrow z, \) and \( c \leftarrow z^T r \)

5. For \( k = 1, 2, \ldots, M \) do

   (a) If \((s^T s)^{1/2} < \delta\) then stop

   (b) \( z \leftarrow Ws \)

   (c) \( t \leftarrow c/s^T z \)

   (d) \( v \leftarrow v + ts \)

   (e) \( r \leftarrow r - tz \)

   (f) Solve \( Qz = r \) for \( r \)

   (g) Set \( d \leftarrow z^T r \)

   (h) If \( d^2 < \epsilon \) then

      i. \( ep \leftarrow r^T r \)

      ii. If \( ep^2 < \epsilon \) then stop

   endif

   (i) \( s \leftarrow z + \frac{d}{c} s \)

   (j) \( c \leftarrow d \)

   (k) Output \( k, v, r \)

6. end
That is, given the matrix $W$, the preconditioner $Q$, the right-hand side vector $e$ and two reals numbers $\delta$ and $\epsilon$, the value of the unknown solution $v$ may be computed as follows:

1. Starting with an initial value $v$, compute the residual vector $r$.
2. Then, find the vector $z$ by solving $Qz = r$.
3. Set $s$ and $c$,
4. and for $k = 1, \ldots, M$, compute relevant quantities and check convergence criteria.
5. If convergence criteria are satisfied then stop. If not, iterate until either criteria are satisfied or maximum number of iterations is exceed.

The preconditioned conjugate gradient is available in many software packages such as the ITPACKV 2D of Kincaid, Oppe, and Young[1989], the NSPGG of Oppe, Joubert, and Kincaid [1988], as well as in many Fortran libraries such as IMSL, NAG, Harwell library HSL, and the IBM library ESSL. In NAG, the preconditioned conjugate gradient is obtained by calling a suite of 3 subroutines: F11GAF, F11GBF, and F11GCF. F11GAF sets up the suite, and F11GBF is the iterative solver, while F11GCF returns additional information about the computation, such as the number of iterations, convergence criteria, and so on.

Example 6: Illustration of the Preconditioned Conjugate Gradient Method:

Consider the following symmetric, p.d. matrix $W$:

$$
W = \begin{bmatrix}
8 & 0 & 0 & 0 & 2 \\
0 & 5 & 0 & 0 & 1 \\
0 & 0 & 3 & 0 & 2 \\
0 & 0 & 0 & 5 & 1 \\
2 & 1 & 2 & 1 & 4
\end{bmatrix}
$$

(1.55)
and suppose that we are interested in solving the following system:

\[ Wv_5 = e_5 \]  \hspace{1cm} (1.56)

Using the suite F11GAF, F11GBF, and F11GCF in the NAG library, we obtain:

Number of iterations for convergence = 5

and solution vector:

\[ v_5 = (.1603, .0283, .0943, .0283, -.1415)^T \]  \hspace{1cm} (1.57)

Which agrees with both the Cholesky's method and the Sparse Bus Impedance method.

1.3.3.3 Advantages and Disadvantages of Iterative Methods

In summary, iterative methods offer the following 4 advantages:

1. They involve no creation of fill-ins, since they work with the original matrix \( W \) (see David Evans [11]).

2. They do not require the storage of the original matrix \( W \). In the case of partial differential equation, the elements of \( W \) are generated as needed and are not retained after use. Thus, iterative methods have a decisive advantage over direct methods in terms of speed and demands on computer memory.

3. Therefore, iterative methods are well suited to very large, and sparse matrices.

4. They are usually stable (see [27])

On the other hand, iterative methods share the following 4 drawbacks:

1. An iterative method is suitable only for a specific class of problems.

2. Iterative algorithms such as SOR and related methods depend upon parameters that are sometimes hard to choose properly (see [21])
3. In SOR and related methods, the ordering of equations and unknowns is very crucial for the success of the method (see [21]).

4. In the conjugate gradient and related methods, the choice of the preconditioner can have a dramatic effect on the rate of convergence.

In summary, traditional methods for computing the inverse of a large, sparse, and p.d. matrix are in need of improvement. Interestingly, the computation of $W^{-1}$ can be successfully achieved via a statistical approach. This approach, which is devoid of fill-ins, good ordering, and choice of critical parameters problems, is the topic of this dissertation.

1.4 Focus of this Dissertation

The organization of this dissertation is as follows:

Chapter 2 describes how the computation of the inverse matrix can be reduced to the computation of an expectation (or integral) of well-defined random variables.

Chapter 3 shows how the Importance Sampling technique may be used to estimate the above expectations and verifies some asymptotic properties of the estimates.

Chapter 4 introduces Adaptive Importance Sampling as an improvement over the Importance Sampling technique. A better estimate of the inverse matrix may be obtained using a smaller number of Monte Carlo stimulations.

Chapter 5 deals with the problem of efficient sparse storages as a means to take advantage of the sparsity of the matrix and chapter 6 with parallel processing as a means to speed up the Adaptive Importance Sampling technique.

Finally, this dissertation ends with a conclusion and discussion for further research.
2 CONDITIONING, UNBIASED ESTIMATORS, AND
SWINDLE DECOMPOSITIONS

The goal in this chapter is to obtain an analytical expression for each element of
the inverse matrix $V$ using Swindle decompositions. To this end, in section 1, we will
establish 2 main distributional results using conditioning. Then, in section 2, we will
show how each element of the inverse matrix $V$ can be written in terms of the original
matrix $W$. Section 3 will make evident the role of the sparsity of $W$ and section 4 will
extend the results of the previous sections to any submatrix of $W$.

2.1 Conditioning and Pseudo Unbiased Estimators

Let $W$ be a large, sparse, symmetric and p.d. matrix in $R^{n \times n}$. Let $V = W^{-1}$ and

$x = (x_1, \ldots, x_n) \sim MVN(0, V)$  \hspace{1cm} (2.1)

with $v_{ij}$ be the $ij$th entry of $V$, $(i, j = 1, 2, \ldots, n)$. That is:

$v_{ij} = E_x(x_i x_j) \quad i, j = 1, 2, \ldots, n$  \hspace{1cm} (2.2)

Specifically

$v_{ii} = E_x(x_i^2) \quad i = 1, 2, \ldots, n$  \hspace{1cm} (2.3)

and

$v_{ij} = E_x(x_i x_j) \quad i \neq j = 1, 2, \ldots, n$  \hspace{1cm} (2.4)
Therefore, define

\[ \hat{v}_{ii} = \frac{1}{N} \sum_{k=1}^{N} (x_i^{(k)})^2 / N \]  

(2.5)

and

\[ \hat{v}_{ij} = \frac{1}{N} \sum_{k=1}^{N} x_i^{(k)} x_j^{(k)} / N \]  

(2.6)

where, \( x^{(k)} \), \( k = 1, \ldots, N \) are \( N \) independent draws from (2.1).

If we could take a random sample from the distribution of \( x \), \( \hat{v}_{ii} \) and \( \hat{v}_{ij} \) would have been unbiased estimators of \( v_{ii} \) and \( v_{ij} \), respectively. However, since \( V = W^{-1} \) is unknown, we cannot sample from the distribution of \( x \). Therefore, the above expectation cannot be evaluated accurately. So our next step is to obtain more practical (computable and useful) estimators, using conditioning.

To see this, let:

\[ x_{(i)} = (x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) \]  

(2.7)

\[ x_{(i,j)} = (x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_j-1, x_{j+1}, \ldots, x_n) \]  

(2.8)

then by the law of double expectation,

\[ v_{ii} = \mathbb{E}_{x_{(i)}} \left[ \mathbb{E}_{x_{(i)}} \left( x_i^2 | x_{(i)} \right) \right] \]  

(2.9)

\[ v_{ij} = \mathbb{E}_{x_{(i,j)}} \left[ \mathbb{E}_{(x_i, x_j)} \left( x_i x_j | x_{(i,j)} \right) \right] \]  

(2.10)

The above expressions of \( v_{ii} \) and \( v_{ij} \) have the following two advantages:

1. They provide Unbiased Estimators of \( v_{ii} \) and \( v_{ij} \) respectively.

2. They also provide a swindle decomposition of \( v_{ii} \) and \( v_{ij} \) respectively.
2.1.1 Conditional Swindle

Specifically, the inside expectation in (2.9) may be rewritten as:

\[ E_{x_i|x(i)} \left( x_i^2 | x(i) \right) = var(x_i|x(i)) + \left[ E(x_i|x(i)) \right]^2 \]  

(2.11)
since for any random variable Z:

\[ E(Z^2) = var(Z) + \left[ E(Z) \right]^2. \]  

(2.12)

Similarly, the inside expectation in (2.10) can be written as:

\[ E(x_i,x_j|x(i,j)) = cov \left( x_i, x_j | x(i,j) \right) + E(x_i|x(i,j))E(x_j|x(i,j)) \]  

(2.13)
since for any two random variables Y and Z,

\[ E(YZ) = COV(Y, Z) + E(Y)E(Z). \]  

(2.14)

To evaluate (2.11) and (2.13), we will make use of the following well known result in order to obtain the conditional distribution of \( x_i|x(i) \) and the joint-distribution of \( (x_i, x_j)|x(i,j) \).

Let \( V = W^{-1} \), \( x_{n+1} \sim MVN(0, V) \), and \( r \) be any positive integer between 1 and \( n-1 \) inclusive. Let \( s_1, \ldots, s_r, t_1, \ldots, t_{n-r} \) be any permutation of the first \( n \) positive integers. Define \( x_S \) to be the \( r \times 1 \) vector whose \( i \)th element is the \( s_i \)th element of \( x \), and \( x_T \) to be the \( (n-r) \times 1 \) vector whose \( j \)th element is the \( t_j \)th element of \( x \).

Let \( V_{SS} \) and \( W_{SS} \) represent the \( r \times r \) matrices whose \( ij \)th elements are \( v_{s_i,s_j} \) and \( w_{s_i,s_j} \) respectively. Similarly, let \( V_{ST} \) and \( W_{ST} \) be the \( r \times (n-r) \) matrix whose \( ij \)th element is \( v_{s_i,t_j} \) and \( w_{s_i,t_j} \) respectively. Finally, let \( V_{TT} \) be the matrix whose \( t_i t_j \)th element is \( v_{t_i,t_j} \). Then, using a well known result concerning MVN distribution (Graybill [22] and Harville [24]), the conditional distribution of \( x_S \) given \( x_T \) is:

\[ x_S|x_T \sim MVN(-W_{SS}^{-1} W_{ST} x_T, W_{SS}^{-1}) \]  

(2.15)
By letting:

\[ x_S = x_i \quad (2.16) \]
\[ x_T = x_{(i)} \quad (2.17) \]

we have

\[ W_{SS} = w_{ii} \quad (2.18) \]

and

\[ W_{ST} = (w_{i1}, \ldots, w_{i,i-1}, w_{i,i+1}, \ldots, w_{in}) \quad (2.19) \]

so that

\[ W_{SS}^{-1} W_{ST} x_T = w_{ii}^{-1} \sum_{k \neq i} w_{ik} x_k \quad (2.20) \]

Hence, by the above theorem we have:

\[ x_i|x_{(i)} \sim N\left(-w_{ii}^{-1} \sum_{k \neq i} w_{ik} x_k, \quad w_{ii}^{-1}\right) \quad (2.21) \]

Similarly, let:

\[ x_S = (x_i, x_j) \quad (2.22) \]
\[ x_T = x_{(i,j)} \quad (2.23) \]

Then, for \( i \neq j \)

\[ W_{SS} = \begin{pmatrix} w_{ii} & w_{ij} \\ w_{ij} & w_{jj} \end{pmatrix} \quad (2.24) \]
and

\[ W_{ST} = \begin{pmatrix} w_{i1}, \ldots, w_{i,i-1}, w_{i,i+1}, \ldots, w_{i,j-1}, w_{i,j+1}, \ldots, w_{i,n} \\ w_{j1}, \ldots, w_{j,i-1}, w_{j,i+1}, \ldots, w_{j,j-1}, w_{j,j+1}, \ldots, w_{j,n} \end{pmatrix} \tag{2.25} \]

so that, for \( j \neq i \)

\[-W_{SS}^{-1}W_{ST}x_T = \frac{1}{\Delta} \begin{pmatrix} w_{jj} \sum_{k \neq i, k \neq j} w_{ik} x_k - w_{ij} \sum_{s \neq j, s \neq i} w_{js} x_s \\ -w_{ij} \sum_{k \neq i, k \neq j} w_{ik} x_k + w_{ii} \sum_{s \neq j, s \neq i} w_{js} x_s \end{pmatrix} \tag{2.26} \]

where

\[ \Delta = w_{ii} w_{jj} - w_{ij}^2 \tag{2.27} \]

Therefore, by the above theorem

\[ (x_i, x_j|x_{(i,j)}) \sim \text{MVN} \left[ \mu_{S|T}, \begin{pmatrix} w_{ii} & w_{ij} \\ w_{ij} & w_{jj} \end{pmatrix}^{-1} \right] \tag{2.28} \]

where

\[ \mu_{S|T} = \frac{-1}{\Delta} \begin{pmatrix} w_{jj} \sum_{k \neq i, k \neq j} w_{ik} x_k - w_{ij} \sum_{s \neq j, s \neq i} w_{js} x_s \\ -w_{ij} \sum_{k \neq i, k \neq j} w_{ik} x_k + w_{ii} \sum_{s \neq j, s \neq i} w_{js} x_s \end{pmatrix} \tag{2.29} \]

and \( \Delta \) as defined by (2.27).

The results in (2.21) and (2.28) are very useful because they provide us with unbiased estimators and also lead to swindle decompositions of the elements of \( V \).
2.1.2 Pseudo Unbiased Estimators

From (2.21), we can see that:

\[ \text{var}(x_i|x(i)) = 1/w_{ii} \] (2.30)

and

\[ \mathbb{E}(x_i|x(i)) = \sum_{k \neq i} w_{ik}x_k \] (2.31)

and hence, using (2.11) we obtain the following random variable:

\[ z_{ii} = w_{ii}^{-1} + \left( \sum_{k \neq i} w_{ik}x_k \right)^2 \] (2.32)

Therefore,

\[ \bar{v}_{ii} = \frac{\sum_{k=1}^{N} z_{ii}^{(k)}}{N} \] (2.33)

where, \( z_{ii}^{(k)}, k = 1, \ldots, N \) are \( N \) independent draws from (2.32). Thus, if we could take random sample from the of \( z_{ii} \), then \( \bar{v}_{ii} \) would have been an unbiased estimator of \( v_{ii} \).

Similarly, using (2.28), we have:

\[ \text{cov}(x_i, x_j|x(i,j)) = -w_{ij}/\Delta \] (2.34)

\[ \mathbb{E}(x_i|x(i,j)) = \left( \sum_{k \neq i, k \neq j} w_{ik}x_k - w_{ij} \sum_{s \neq j, s \neq i} w_{js}x_s \right)/\Delta \] (2.35)

and

\[ \mathbb{E}(x_j|x(i,j)) = \left( -w_{ij} \sum_{k \neq i, k \neq j} w_{ik}x_k + w_{ii} \sum_{s \neq j, s \neq i} w_{js}x_s \right)/\Delta \] (2.36)

and hence from (2.13) we obtain the following random variable

\[ z_{ij} = -w_{ij}/\Delta + \left( \sum_{k \neq i, k \neq j} w_{ik}x_k - w_{ij} \sum_{s \neq j, s \neq i} w_{js}x_s \right) \]

\[ \times \left( -w_{ij} \sum_{k \neq i, k \neq j} w_{ik}x_k + w_{ii} \sum_{s \neq j, s \neq i} w_{js}x_s \right)/\Delta^2 \] (2.37)
Therefore,

\[
\bar{v}_{ij} = \frac{1}{N} \sum_{k=1}^{N} z_{ij}^{(k)}
\]  

(2.38)

where, \(z_{ij}^{(k)}, k = 1, \ldots, N\) are \(N\) independent draws from (2.37). Thus, if we could take random sample from the of \(z_{ij}\), then \(\bar{v}_{ij}\) would have been an unbiased estimator of \(v_{ij}\)

In summary, with conditioning we have derived estimators \(\bar{v}_{ii}\) as defined by (2.33) and \(\bar{v}_{ij}\) as given by (2.38). However, there are two drawbacks to these estimators:

1. They cannot be directly computed since they depend on the values of the random vector \(x\) whose distribution is we cannot generate from. One way to remedy this is to use the Gibbs Sampler (as suggested by [24]) to obtain sample values for the random vector \(x\). Or, use the Importance Resampling technique (Gelman et al. [16]) to obtain an approximate distribution for the vector \(x\).

2. Moreover, \(\bar{v}_{ij}\), the estimator of \(v_{ij}\) is more computationally intensive since it requires the inversion of a \(2 \times 2\) matrix.

Interestingly, better estimators of \(v_{ii}\) and \(v_{ij}\) can be obtained using the swindle decomposition, which is the topic of the next section.

2.2 Swindle Decomposition

The aim in this section is to decompose both \(v_{ii}\) and \(v_{ij}\) into two parts: a fixed components and an expectation of a random variable. To see this, recall that for \(i = 1, \ldots, n:\)

\[
v_{ii} = E_{x_{(i)}}[z_{ii}]
\]  

(2.39)
and in light of (2.32), we have

\[ v_{ii} = E\{ \psi_i \} \left[ w_{ii}^{-1} + \left( w_{ii}^{-1} \sum_{k \neq i} w_{ik} x_k \right)^2 \right] \]  

(2.40)

i.e.

\[ v_{ii} = 1/w_{ii} + \theta_{ii}/w_{ii}^2 \]  

(2.41)

where,

\[ \theta_{ii} = E\{ \psi_i \} \left( \sum_{k \neq i} w_{ik} x_k \right)^2. \]  

(2.42)

Similarly, and for \( j \neq i \)

\[ v_{ij} = E\{ \psi_{i,j} \} \left[ z_{ij} \right] \]  

(2.43)

and in light of (2.37) we have:

\[ v_{ij} = E\{ \psi_{i,j} \} \left[ -w_{ij}/\Delta + (w_{jj} \sum_{k \neq i, k \neq j} w_{ik} x_k - w_{ij} \sum_{s \neq j, s \neq i} w_{js} x_s) \left( \sum_{k \neq i} w_{ik} x_k + \sum_{s \neq j, s \neq i} w_{js} x_s \right)/\Delta^2 \right] \]  

(2.44)

i.e.

\[ v_{ij} = -w_{ij}/\Delta + \theta_{ij}/\Delta^2 \]  

(2.45)

where,

\[ \theta_{ij} = E\{ \psi_{i,j} \} \left[ (w_{jj} \sum_{k \neq i, k \neq j} w_{ik} x_k - w_{ij} \sum_{s \neq j, s \neq i} w_{js} x_s) \left( \sum_{k \neq i} w_{ik} x_k + \sum_{s \neq j, s \neq i} w_{js} x_s \right) \right] \]  

(2.46)

and \( \Delta \) is given by (2.27)

The expression of \( v_{ii} \) as given by (2.41) and \( v_{ij} \) as given by (2.45) will be called the decomposition swindle of \( v_{ii} \) and \( v_{ij} \) respectively. These decompositions are better than the expression given by (2.33) and (2.38) for the following six reasons for all \( i, j = 1, \ldots, n \).
1. Each element $v_{ij}$ of $V$ is decomposed as sum of two components:
   (a) a fixed component
   (b) an expectation of a random component

2. The variance of an estimator of $v_{ij}$ is reduced to the variance of a conditional estimator of $\theta$ as given by (2.42) and (2.46)

3. The computation of elements of $V$ is reduced to a computation of expectations.

4. These decompositions take advantage of the sparsity of $W$ in that only nonzero elements of $W$ are stored and processed. This may result in a considerable savings in storage and computing time.

5. These decompositions are devoid of fill-ins, good ordering and the problem of choosing critical parameters, since they work directly on the original matrix $W$.

6. These decompositions make relevant the role of sparsity and positive definiteness of $W$ which is the topic of our next section.

2.3 Role of Positive Definiteness and Sparsity of $W$

The expression of $\theta_{ii}$ and that of $\theta_{ij}$ point out the role of sparsity and positive definiteness of $W$ as follows:

1. Positive definiteness of $W$ insures the existence of $w_{ii}^{-1}$ for all $i = 1, \ldots, n$ ([21], page 102)

2. Since $W$ is sparse, then most of the $w_{ik}$, $i \neq k$ are equal to zeros. This reduces $\sum_{k \neq i} w_{ik} x_k$ to a few terms which results in savings in computing time and memory.
3. The exact value of $v_{ij}$ is obtained if

$$w_{ik} = w_{jk} = 0 \quad \text{for all } k \neq i \text{ and } k \neq j.$$  \hfill (2.47)

Examples of such $v_{ij}$ will be given in the next chapter.

2.4 Extension to a Sub-matrix of $V$

The results of the previous section can be generalized to the evaluation of sub-matrices of $V$. To see this, let $V_{SS}$, $V_{S_j}$, $x_S$, and $x_T$ be defined as above. Recall that (Durrett [9]):

$$\text{var}(x_S) = E[\text{var}(x_S|x_T)] + \text{var}[E(x_S|x_T)]$$  \hfill (2.48)

So that in light of the conditional distribution theorem:

$$\text{var}(x_S) = E[W^{-1}] + \text{var}[-W_{SS}^{-1}W_{ST}x_T]$$

$$= W^{-1} + W_{SS}^{-1}\text{var}[W_{ST}x_T](W_{SS}^{-1})^T$$  \hfill (2.49)

or

$$\text{var}(x_S) = W_{SS}^{-1} + W_{SS}^{-1}\theta_{SS}W_{SS}^{-1}$$  \hfill (2.50)

where

$$\theta_{SS} = E[(W_{ST}x_{ST})(W_{ST}x_{ST})^T]$$  \hfill (2.51)

since

$$E(W_{ST}x_{ST}) = W_{ST}E(x_{ST}) = 0$$  \hfill (2.52)

and

$$(W_{SS}^{-1})^T = (W_{SS}^{-1})^{-1} = W_{SS}^{-1}$$  \hfill (2.53)
and since

\[ \mathbf{W}_{SS}^T = \mathbf{W}_{SS}. \] (2.54)

Example 7: Estimation of Sub-matrix of \( \mathbf{V} \):

Let \( \mathbf{V} \) be an \( n \times n \) matrix and suppose we would like to compute the \( 2 \times 2 \) sub-matrix \( \mathbf{V}_{SS} \) of \( \mathbf{V} \) defined as follows:

\[
\mathbf{V}_{SS} = \begin{pmatrix}
  v_{11} & v_{12} \\
  v_{12} & v_{22} \\
\end{pmatrix}
\] (2.55)

Let

\[
\mathbf{x}_S = (x_1, x_2)
\] (2.56)

\[
\mathbf{x}_T = \mathbf{x}_{(1,2)}
\] (2.57)

Then

\[
\mathbf{W}_{SS} = \begin{pmatrix}
  w_{11} & w_{12} \\
  w_{12} & w_{22} \\
\end{pmatrix}
\] (2.58)

and

\[
\mathbf{W}^{-1}_{SS} = \frac{1}{\Delta} \begin{pmatrix}
  w_{22} & -w_{12} \\
  -w_{12} & w_{11} \\
\end{pmatrix}
\] (2.59)

with

\[
\Delta = w_{11}w_{22} - w_{12}^2
\] (2.60)

Also,

\[
\mathbf{W}_{ST} = \begin{pmatrix}
  w_{13}, \ldots, w_{1n} \\
  w_{23}, \ldots, w_{2n} \\
\end{pmatrix}
\] (2.61)

Therefore,

\[
\mathbf{W}_{ST} \mathbf{x}_T = \begin{pmatrix}
  \sum_{s \neq 1} w_{1s} x_{1s} \\
  \sum_{k \neq 2} w_{2k} x_{2k} \\
\end{pmatrix}
\] (2.62)
and

\[ V_{ss} = W_{ss}^{-1} + W_{ss}^{-1} \theta_{ss} W_{ss}^{-1} \]  

(2.63)

where

\[ \theta_{ss} = \begin{pmatrix} \mathbb{E}(\sum_{s \neq 1} w_{1s} x_s)^2 & \mathbb{E}(\sum_{s \neq 1} w_{1s} x_s)(\sum_{k \neq 1} w_{2k} x_{2k}) \\ \mathbb{E}(\sum_{s \neq 1} w_{1s} x_s)(\sum_{k \neq 1} w_{2k} x_{2k}) & \mathbb{E}(\sum_{k \neq 2} w_{2k} x_{2k})^2 \end{pmatrix} \]  

(2.64)

The message here is that the computation of the elements of \( V \) and/or of a sub-matrix of \( V \) can be reduced to the computation of the expectation of a certain function, \( g \), of a well defined random variable. Some examples of \( g \) are:

\[ g(x) = x_i x_j \quad \text{for all } i \text{ and } j \]  

(2.65)

\[ g(x) = \left( \sum_{k \neq i} w_{ik} x_k \right)^2 \]  

(2.66)

\[ g(x) = (w_{ij} \sum_{k \neq i, k \neq j}^n w_{ik} x_k - w_{ij} \sum_{s \neq j, s \neq i}^n w_{js} x_s) \]  

\[ \times (-w_{ij} \sum_{k \neq i, k \neq j}^n w_{ik} x_k + w_{ii} \sum_{s \neq j, s \neq i}^n w_{js} x_s) \]  

(2.67)

\[ g(x) = \left[ (W_{ST} x_{ST})(W_{ST} x_{ST})^T \right] \]  

(2.68)

In the following chapter, we will discuss Monte Carlo methods for evaluating the above expectation efficiently.
3 APPLICATION OF IMPORTANCE SAMPLING TO EVALUATE AN INTEGRAL

3.1 Goal

Let $x_{n \times 1} \sim \text{MVN}(0, V)$. At the end of the previous chapter, we have seen that the computation of the elements of $V = W^{-1}$, is reduced to the evaluation of the following quantity:

$$
\theta = E_x[g(x)]
$$

where

$$
g(x) = \left( \sum_{k \neq i} w_{ik} x_k \right)^2
$$

and, for $i \neq j$

$$
g(x) = (w_{jj} \sum_{k \neq i, k \neq j} w_{ik} x_k - w_{ij} \sum_{s \neq j, s \neq i} w_{js} x_s)(-w_{ij} \sum_{k \neq i, k \neq j} w_{ik} x_k + w_{ii} \sum_{s \neq j, s \neq i} w_{js} x_s)
$$

Let $p(x)$ be the probability density function of $x$. And express $p(x)$ as:

$$
p(x) = \frac{f(x)}{\int f(x) dx}
$$

where,

$$
f(x) = e^{-\frac{1}{2}x^T W x}
$$
Thus, $\theta$ can be expressed as a ratio of integrals. That is:

$$\theta = \int g(x)p(x)dx$$  \hspace{1cm} (3.6)

$$\theta = \frac{\int g(x)f(x)dx}{\int f(x)dx}$$  \hspace{1cm} (3.7)

In the following sections, we will examine the tools available for computing the expectation in 3.7. Our main goal in this chapter is to provide a reasonable estimate for $\theta$ using the Importance Sampling technique. However, first, let us review the numerical and Monte-Carlo techniques available for evaluating such integral.

### 3.2 Earlier Work

Gelman et al. [16], as well as Evan and Swartz [13] offer a survey of the wide varieties of methods available to evaluate $\theta$. These include numerical methods such as Simpson's Rule, Gaussian Quadrature, Laplace's Method, and Monte-Carlo methods such as Gibbs Sampler Importance Sampling, Adaptive Importance Sampling... etc.

However, as mentioned by [16], in the multivariate cases, Gaussian quadrature and asymptotic approximations are very difficult to apply. Moreover, the main reason numerical approximation such as Laplace's Method cannot be used here, is that, it is not possible to generate random samples from the normalized density $p(x) = \frac{f(x)}{\int f(x)dx}$, since $p(x)$ depends on the determinant of $V$ which is unknown. That's why, methods that allow us to generate from $p$ without actually knowing $V$ such as the Gibbs Sampler and Importance Sampling become very attractive.

#### 3.2.1 Gibbs Sampler For Matrix Inversion: David Harville(1996)

David Harville [24], in a 1996 paper, used the Gibbs Sampler to compute $\theta$. The Gibbs Sampler algorithm was developed by Geman and Geman [17]. Smith and Roberts [36] offer a detailed description in their 1993 paper. The Gibbs sampler is a numerical
method based on all possible conditional distributions. Let \( x = (x_i) \) be an \( n \times 1 \) vector whose distribution is a MVN with mean zero and variance covariance matrix \( V = (v_{ij}) \). Applied to the problem of finding \( V \), the Gibbs Sampler proceeds as follows:

1. Choose an arbitrary \( n \times 1 \) vector \( x^{(0)} \).

2. Generate \( x^{(1)} \) as follows:
   
   (a) draw \( x_1^{(1)} \) from \( N \left( \frac{\sum_{j=2}^{n} w_{1j} x_j^{(0)} }{w_{11}}, 1/w_{11} \right) \)
   
   (b) draw, \( x_2^{(1)} \) from \( N \left( \frac{w_{21} x_1^{(1)} + \sum_{j=3}^{n} w_{1j} x_j^{(0)} }{w_{22}}, 1/w_{22} \right) \)
   
   (c) draw, \( x_3^{(1)} \) from \( N \left( \frac{w_{31} x_1^{(1)} + w_{32} x_2^{(1)} + \sum_{j=4}^{n} w_{3j} x_j^{(0)} }{w_{33}}, 1/w_{33} \right) \)
   
   (d) draw \( x_i^{(1)} \) from \( N \left( \frac{\sum_{j=1}^{i-1} w_{ij} x_j^{(1)} + \sum_{j=i+1}^{n} w_{ij} x_j^{(0)} }{w_{ii}}, 1/w_{ii} \right) \)
   
   (e) draw \( x_n^{(1)} \) from \( N \left( \frac{\sum_{j=1}^{n} w_{nj} x_j^{(0)} }{w_{nn}}, 1/w_{11} \right) \)

3. Repeat step 1 and step 2 \( k \) times, to get the Gibbs sequence

\[ x^{(1)}, x^{(2)}, \ldots, x^{(k)}, \ldots \]

such as, \( x_i^{(k)} \) the \( i \)th component of the vector \( x^{(k)} \), is a draw from the univariate \( N \left( w_{ii}^{-1} \sum_{j=1}^{i-1} w_{ij} x_j^{(k)} + \sum_{j=i+1}^{n} w_{ij} x_j^{(k-1)} , 1/w_{ii} \right) \).

4. As \( k \to \infty \), \( x^{(k)} \) converges in distribution to \( x \).

5. And for any positive integer \( p \) and for sufficiently large value of \( n \), the quantity:

\[
\tilde{g} = (n - p)^{-1} \sum_{k=p+1}^{n} \left( \sum_{i \neq i} w_{ii} x_t \right)
\]  \hspace{1cm} (3.8)

provides an accurate estimate of \( \theta \) in 3.7 (see [24]).

Harville [24] pointed out three primary drawbacks of the Gibbs Sampler:
1. The successive draws are statistically dependent

2. The convergence to the limiting distribution is very slow

3. The estimators may sometimes be relatively inaccurate

In addition, when the dimension of $W$ is large, a preferred method for the evaluation of the expectation in 3.7 is Monte Carlo Integration with Importance Sampling because of its simplicity and relative ease of implementation.

### 3.3 The Importance Sampling Approach

According to [16], Importance Sampling is a simple method for obtaining approximate samples from a complicated high-dimensional distribution, or from a distribution which we cannot generate from or which is not known. Therefore, importance sampling becomes useful in this case, since the main difficulty in evaluating the expectation in 3.7 is that we cannot generate from the MVN ($0$, $V$) and $V$ is unknown.

#### 3.3.1 The Method

Recall that the quantity to be computed is:

$$\theta = \frac{\int g(x)f(x)dx}{\int f(x)dx} \quad (3.9)$$

The Importance Sampling approach proceeds as follows:

1. Choose a 'good' density $h$, that is easy to sample from. $h$ will be called Importance function or Importance Sampler. (The choice of $h$ will be discussed later.)

2. Generate a sample of size $N$ random vectors, $u_1, \ldots, u_N$, from $h$,

where each $u$ is an $n \times 1$ vector.
3. Compute for each \( u \) the importance weight:

\[
\omega(u) = \frac{f(u)}{h(u)}
\]  

(3.10)

4. Rewrite \( \theta \) as:

\[
\theta = \frac{\int g(u)\omega(u)h(u)du}{\int \omega(u)h(u)du}
\]  

(3.11)

5. Estimate \( \theta \) by:

\[
\hat{\theta} = \frac{\sum_{k=1}^{N} g(u_k)\omega(u_k)}{\sum_{k=1}^{N} \omega(u_k)}
\]  

(3.12)

Since its introduction in 1964 by Hammersley and Handscomb [23], detailed descriptions, discussions, and applications appeared in Davis and Rabinowitz [2], as well as Hop et al. [4] among others. Variance reduction techniques, such as control variates, stratified sampling, and systematic sampling, were studied by Geweke [19] and [20] and Evans [13]. Effect of the weights and dimensionality of \( W \) on the accuracy of \( \hat{\theta} \) is given by Man-Suk Oh [35].

To check the consistency and the accuracy of the estimator \( \hat{\theta} \) we will make use of the following two results given by Geweke:

**Theorem 3.1 (Geweke, 1986, page 5)** Assume that the importance function \( h \) is such that:

1. **Assumption I**: Let \( u_1, \ldots, u_N, \ldots \) be a sequence of i.i.d. random vectors of dimension \( n \) with a common distribution having a probability density function \( h(u) \).

2. **Assumption II**: The support of \( h \) is \( \mathbb{R}^n \)

3. **Assumption III**: \( E_h[g(u)] \) exists.

Under these assumptions, \( \hat{\theta} \xrightarrow{a.s.} \theta \).
Theorem 3.2 (Geweke, 1988, page 9) If the following two conditions are satisfied:

\[
E_h[(\omega(u))^2] < \infty \quad (3.13)
\]

\[
E_h[(g(u))^2\omega(u)] < \infty \quad (3.14)
\]

Then,

\[
\sqrt{N} \left( \hat{\theta} - \theta \right) \xrightarrow{\text{law}} N\left(0, \sigma^2\right) \quad (3.15)
\]

In our study, the following distributions will be considered for the importance function \( h \):

1. the Multivariate Normal distribution with a diagonal covariance matrix,
2. the Multivariate t distribution with a diagonal covariance matrix,
3. and the Mixture Sampling of the above two distributions.

Since we will be using \( N \) random samples from these distributions, assumptions I and II are satisfied.

For the problem of estimation at hand, the integrand of interest are those functions \( g \) defined in 3.2 and 3.3. In addition, the covariance matrix will be selected so that the importance function \( h \) will satisfy the following conditions:

1. \( \omega(u) < M \) a.e. on \( \mathbb{R}^n \) and for some integer \( M \)
2. And \( E_h[g(u)^2] < \infty \)

where \( g \) is defined in 3.2 and 3.3.

For this class of importance functions, the results in Theorem 3.1 and Theorem 3.2 hold for the estimator \( \hat{\theta} \) as shown in the following corollary:
Corollary 3.1 If $h$ is such that:

1. $\omega(u) < M$ a.e. on $\mathbb{R}^n$ and for some integer $M$

2. And $E_h[(g(u))^2] < \infty$

then:

$$\hat{\theta} \xrightarrow{a.s.} \theta$$

(3.16)

and

$$\sqrt{N} \left( \hat{\theta} - \theta \right) \xrightarrow{\text{Law}} N \left( 0, \sigma^2 \right)$$

(3.17)

with,

$$\sigma^2 = \frac{1}{\left( \int f(u)du \right)^2} \left[ \text{var}_h(g\omega) - 2\theta \text{cov}_h(g\omega, \omega) + \theta^2 \text{var}_h(\omega) \right]$$

(3.18)

where $\text{var}_h$ and $\text{cov}_h$ are variances and covariances as taken with respect to $h$.

Proof: It suffices to verify the conditions of Theorem 3.1 and Theorem 3.2 respectively. First, suppose

$$E_h[(g(u))^2] < \infty$$

Then,

$$[E_h[(g(u))]^2 < E_h[(g(u))^2] < \infty$$

by Jensen's inequality. And hence,

$$E_h[(g(u))] < \infty$$

Thus assumption III of Theorem 3.1 is satisfied.

Therefore, by the Theorem 3.1,

$$\hat{\theta} \xrightarrow{a.s.} \theta$$

(3.19)
Similarly, since

\[ \omega(u) < M \text{ a.e. on } R^n \text{ and for some integer } M \]

then,

\[ E_h[\omega(u)^2] < M^2 < \infty \]  \hspace{1cm} (3.20)

and

\[ E_h[g(u)^2 \omega(u)] < M E_h[g(u)^2] < \infty \]  \hspace{1cm} (3.21)

Thus, by 3.2

\[ \sqrt{N} \left( \hat{\theta} - \theta \right) \overset{\text{Law}}{\rightarrow} \mathcal{N} \left( 0, \sigma^2 \right) \]  \hspace{1cm} (3.22)

with \( \sigma^2 \) as defined in 3.18.

As shown above, both the importance weight, \( \omega \), and the asymptotic variance, \( \sigma^2 \), depend heavily on the choice of the Importance Sampler, \( h \), since the variances and covariances in 3.18 depend on \( h \). Therefore, methods for selecting good importance functions are discussed in the next section.

### 3.4 Choice Of The Importance Function

#### 3.4.1 The Optimal Sampler

In light of the current literature (see [19], [35], and [13]), a 'good' importance sampler, \( h \), should satisfy the following conditions:

1. It makes it comparatively easy to obtain pseudo random samples.

2. It possesses thicker tails than \( f \).

3. It provides a good approximation to \( f \)
4. It provides bounded importance weights.

5. It provides estimates, \( \hat{\theta} \), that have minimum asymptotic variance.

In the following, we consider the class of importance functions defined in theorem 3.1. An importance sampler which satisfies the above conditions is called an optimal importance sampler. As mentioned by Hesterberg (see [26]), there are two main reasons for an importance sampler to be optimal. Firstly, for some region of the sampling space the ratio \( \frac{h(u)}{f(u)} \) can become very small. This means the probability of sampling a random vector \( u \) from that region is very small. For example a region of the sample space where \( \frac{h(u)}{f(u)} < \frac{1}{1000} \) will be sampled less than it would be in simple random sampling with 1000 times as many replications.

Moreover, any \( u \) sampled from such regions are given large weights to compensate. This problem of large weights is common in the multivariate case. According to [35], \( \text{var}(\omega) \) increases exponentially as the dimension \( n \) of \( u \) increases. Secondly, the asymptotic variance of an estimate \( \hat{\theta} \) is:

\[
\sigma^2 = \int (g(u) - \theta)^2 \omega(u)h(u)du \tag{3.23}
\]

Therefore, large values of \( \omega \) yield large value of \( \sigma^2 \). Usually, one desires the weights \( \omega \) to be evenly distributed, as opposed to a few very large weights and many small weights. One solution to the problem of large weights is the use of mixture sampling.
3.4.2 Mixture Sampling

According to Hesterberg [26], mixture sampling consists of using a sampling distribution which is a convex combination of the true distribution $f$ and the importance function, $h_0$. Since $f$ is not known, we will use an approximate mixture. Precisely, the approximate mixture distribution is defined as:

$$h^*_\lambda(u) = \lambda f^*(u) + (1 - \lambda) h_0(u)$$  \hspace{1cm} (3.24)

with

$$0 \leq \lambda \leq 1$$  \hspace{1cm} (3.25)

where $f^*$ is an approximation to $f$ and $\lambda = 1$ means no importance sampling and $\lambda = 0$ sampling from $h_0$.

The advantage of using the mixture distribution is three-fold:

1. Firstly, it allows the sampling of rare events relatively more often.

2. Secondly, it reduces the number of replications required before the rare observations are obtained an adequate number of times. By doing so, the sample size needed to achieve a desired accuracy is reduced.

3. Thirdly, it ensures that the importance weights are bounded. In fact, in here, since we do not know $f$, we will replace $f$ by $f_0$, where $f_0$ is an approximation to $f$.

$$\omega(u) = \frac{f^*(u)}{h^*_\lambda(u)} \leq \frac{1}{\lambda}.$$  \hspace{1cm} (3.26)

Thus, $\omega$ is bounded and hence the problem of large weights as mentioned in section 3.4.1 is prevented.

In the next section, we illustrated the ideas of this section with a numerical example.
3.5 Example and Evaluation of an Optimal Sampler

3.5.1 Materials and Method

Let \( x_{n \times 1} \sim MVN(0, V) \), where \( V \) is unknown and we are interested in evaluating \( V \) using the importance sampling method. To this end, let \( W = V^{-1} \). Consider, \( W \) to be:

\[
W = \begin{bmatrix}
4 & 0 & 1 & 0 & 2 \\
0 & 5 & 0 & 0 & 0 \\
1 & 0 & 3 & 0 & 1 \\
0 & 0 & 0 & 5 & 0 \\
2 & 0 & 1 & 0 & 8
\end{bmatrix}.
\] (3.27)

Next, we will consider three families of distribution, that is, the Multivariate Normal, the Multivariate t, and the Mixture of the two, from which the importance function \( h \) will be chosen. We will also consider the following two covariance matrices:

\[
\Sigma^{(1)} = Diag(1/w(i, i))
\] (3.28)

and

\[
\Sigma^{(2)} = Diag(1/w(i, i) + 1/(w(i, i) \ast w(i, i))).
\] (3.29)

This results in six importance functions, \( h \Sigma \), where, \( \Sigma \) denotes either \( \Sigma^{(1)} \) or \( \Sigma^{(2)} \). These importance functions belong in the class defined in 3.1.

Finally, we also make use of the following estimator, known in the literature as the ratio estimator:

\[
\hat{\theta}_{ratio} = \sum_{k=1}^{N} m_{ratio,k} g(u_k)
\] (3.30)

where

\[
m_{ratio,k} = \frac{\omega(u_k)}{\sum_{k=1}^{N} \omega(u_k)}
\] (3.31)
The ratio estimate is the **weighted average** of the integrand in 3.7, with weights equal to the normalized importance weight. The ratio estimate was chosen because it is **equivariant**, that is:

\[ \hat{\theta}_{\text{ratio}}(a + b \ast g) = a + b \ast \hat{\theta}_{\text{ratio}}(g) \quad (3.32) \]

where \(a\) and \(b\) are two real scalars. This **equivariance property** is important in the computation of \(\hat{\theta}\), since it allows us to use weights that are proportional to \(\omega\) (instead of the actual \(\omega\)), especially when the densities \(f\) and/or \(h\) are known only up to a constant.

The performance of these six importance samplers, \(h_x\), in estimating the elements of \(V\) will be compared based on the histograms of their importance weights and the estimates \(\hat{\theta}\) they produce.

### 3.5.2 Evaluation of the Importance Weights

Under these three distributions, the **Importance Weights** \(\omega\) have the following form:

1. Under the Multivariate Normal:

\[ \omega(u) \propto \exp\left[-\frac{u^T(W - \Sigma)u}{2}\right] \quad (3.33) \]

2. Under the Multivariate \(t\) with \(\nu\) degree of freedom:

\[ \omega(u) \propto \exp\left[-\frac{u^T W u}{2}\right]\left[1 + \frac{u^T \Sigma u}{\nu}\right]^{\frac{(\nu + n)}{2}} \quad (3.34) \]

3. Under the **Mixture distribution**

\[ \omega(u) \propto \frac{1}{\lambda + (1 - \lambda) \Gamma(\frac{e + k}{2}) \Gamma(\frac{f}{2}) \left[\frac{2}{\nu}\right]^{\frac{k}{2}} \exp\left[\frac{u^T W u}{2}\right]\left[1 + \frac{u^T \Sigma u}{\nu}\right]} \quad (3.35) \]

As recommended by [16], to evaluate an importance sampler, it is a good practice to examine the histogram of the logarithm of the largest weights. For, estimates will be reasonable if the largest weights are not too large relative to others. On the other hand, the behavior of small weights is not of concern since they have little influence on the estimator. Here, we will display the histogram of the weights.
The Figures 3.1, 3.2, 3.3, and 3.4, suggest the following:

1. The distribution of the importance weights is *fairly symmetric* for the MVN with covariance matrix = $Diag(1/w(i,i))$ (see 3.1), and *left skew* for the other distributions (see 3.2, 3.3, and 3.4).

2. The importance weights are *bounded*. In fact, the average of the weights are: 1.1338, .6686, 2.7203, 1.6360, .7646, and .9971

3. There is no extremely higher importance weight. Therefore, estimators computed using the importance weights will be reasonable.

### 3.5.3 Simulation Results and Interpretations

For an empirical application of the importance sampling, we proceed as follows:

1. We generate 10000 i.i.d. random vectors, $u_1, \ldots, u_{10000}$ from $h\Sigma$, where $h\Sigma$ is one of the six importance functions defined in the previous section.

2. Then we evaluate for each $u$ the importance weight $\omega(u)$.

3. We also evaluate the estimate $\hat{\theta}$ as defined above and the corresponding estimators $\hat{v}_{ij}$.

4. We then repeat this for all the six $h\Sigma$ defined above.

5. *Estimators* and their *standard error* (s.e) are displayed in Table 3.1.

6. These simulations were repeated for various matrices $W$ of dimension $n = 5$, $n = 7$, $n = 10$, and $n = 397$. The results were compatible with the ones in Table 3.1. Therefore, only the results in those tables were displayed.
Figure 3.1 Histogram of log of Importance weights from MVN with covariance matrix Diag(1/w(i, i))

Figure 3.2 Histogram of log of importance weights from $t_3$ with covariance matrix Diag (1/w(i, i))
Figure 3.3 Histogram of log of importance weights from MVN with covariance matrix Diag \(1/w(i,i) + 1/(w(i,i) \times 2)\)

Figure 3.4 Histogram log of importance weights from \(t_3\) with covariance matrix Diag \(1/w(i,i) + 1/(w(i,i) \times 2)\)
Table 3.1 reveals the following:

1. For a given covariance matrix, the MV $t_3$ yields more precise estimates than the MVN and the Mixture
2. The covariance matrix $\Sigma^{(2)}$ as defined by 3.29 yields more precise estimators than the covariance matrix $\Sigma^{(1)}$ as defined by 3.28
3. The importance samplers considered yield Importance weights that are bounded
4. The Multivariate $t$ with 3 d.f. appears to be an optimal importance sampler.

3.6 Summary

In light of the results shown in Table 3.1, it can be inferred that:

1. The elements of the inverse of large, sparse, and positive definite matrix can be estimated with desired accuracy via the importance sampling approach.
2. In this approach, each element is decomposed as the sum of a fixed, known component and and expectation of a well defined random variable.
3. This approach works directly with the original matrix $W$ and thus is devoid of the problem of good ordering, fill-ins, and choice of critical parameters.
4. This method will always yield positive estimates for variances, and reasonable estimates for other estimable parameters of interest.
5. This approach is simple to apply since the the operations involved in this method are basic linear algebra such as addition, multiplication, division and. These operations are cheap in computer time.
Table 3.1 Comparison of the Performance of 3 Importance Samplers: Nonzero Elements

<table>
<thead>
<tr>
<th>Importance Sampler</th>
<th>$\hat{v}_{11}$ Estimate</th>
<th>s.e. $10^{-3}$</th>
<th>$\hat{v}_{13}$ Estimate</th>
<th>s.e. $10^{-3}$</th>
<th>$\hat{v}_{15}$ Estimate</th>
<th>s.e. $10^{-3}$</th>
<th>$\hat{v}_{22}$ Estimate</th>
<th>s.e. $10^{-3}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>MVN with $\Sigma^{(1)}$</td>
<td>.3065</td>
<td>1.5071</td>
<td>.0776</td>
<td>.3569</td>
<td>-.0655</td>
<td>.1431</td>
<td>.2000</td>
<td>.0000</td>
</tr>
<tr>
<td>$MV_{t3}$ with $\Sigma^{(1)}$</td>
<td>.3043</td>
<td>.6944</td>
<td>-.783</td>
<td>.1539</td>
<td>-.0657</td>
<td>.0935</td>
<td>.2000</td>
<td>.0000</td>
</tr>
<tr>
<td>$0.9 \cdot MVN + 0.1 \cdot MV_{t3}$ with $\Sigma^{(1)}$</td>
<td>.3050</td>
<td>.8687</td>
<td>.0794</td>
<td>.1412</td>
<td>-.0662</td>
<td>.0679</td>
<td>.2000</td>
<td>.0000</td>
</tr>
<tr>
<td>True Value</td>
<td>.3026</td>
<td></td>
<td>-.0789</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MVN $\Sigma^{(2)}$</td>
<td>.3074</td>
<td>1.0783</td>
<td>.0776</td>
<td>.2348</td>
<td>-.0655</td>
<td>.0964</td>
<td>.2000</td>
<td>.0000</td>
</tr>
<tr>
<td>$MV_{t3}$ with $\Sigma^{(2)}$</td>
<td>.3050</td>
<td>.7371</td>
<td>.0781</td>
<td>.1477</td>
<td>-.0658</td>
<td>.0882</td>
<td>.2000</td>
<td>.0000</td>
</tr>
<tr>
<td>$0.9 \cdot MVN + 0.1 \cdot MV_{t3}$ with $\Sigma^{(2)}$</td>
<td>.3052</td>
<td>.8864</td>
<td>.0786</td>
<td>.1566</td>
<td>-.0654</td>
<td>.0776</td>
<td>.2000</td>
<td>.0000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Importance Sampler</th>
<th>$\hat{v}_{33}$ Estimate</th>
<th>s.e. $10^{-3}$</th>
<th>$\hat{v}_{55}$ Estimate</th>
<th>s.e. $10^{-3}$</th>
<th>$\hat{v}_{44}$ Estimate</th>
<th>s.e. $10^{-3}$</th>
<th>$\hat{v}_{55}$ Estimate</th>
<th>s.e. $10^{-3}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>MVN with $\Sigma^{(1)}$</td>
<td>.3704</td>
<td>.7827</td>
<td>.0257</td>
<td>.6804</td>
<td>.2000</td>
<td>.0000</td>
<td>1455</td>
<td>.5959</td>
</tr>
<tr>
<td>$MV_{t3}$ with $\Sigma^{(1)}$</td>
<td>.3683</td>
<td>.4131</td>
<td>.0266</td>
<td>.3403</td>
<td>.2000</td>
<td>.0000</td>
<td>1443</td>
<td>.3298</td>
</tr>
<tr>
<td>$0.9 \cdot MVN + 0.1 \cdot MV_{t3}$ with $\Sigma^{(1)}$</td>
<td>.3739</td>
<td>.4700</td>
<td>.0285</td>
<td>.2209</td>
<td>.2000</td>
<td>.0000</td>
<td>.1465</td>
<td>.2288</td>
</tr>
<tr>
<td>True Value</td>
<td>.3684</td>
<td></td>
<td>.0263</td>
<td></td>
<td>.2000</td>
<td></td>
<td>.1447</td>
<td></td>
</tr>
<tr>
<td>MVN $\Sigma^{(2)}$</td>
<td>.3674</td>
<td>.6312</td>
<td>.0256</td>
<td>.4893</td>
<td>.2000</td>
<td>.0000</td>
<td>.1456</td>
<td>.4509</td>
</tr>
<tr>
<td>$MV_{t3}$ with $\Sigma^{(2)}$</td>
<td>.3685</td>
<td>.4064</td>
<td>.0265</td>
<td>.3355</td>
<td>.2000</td>
<td>.0000</td>
<td>.1445</td>
<td>.3286</td>
</tr>
<tr>
<td>$0.9 \cdot MVN + 0.1 \cdot MV_{t3}$ with $\Sigma^{(2)}$</td>
<td>.3739</td>
<td>.5198</td>
<td>.0262</td>
<td>.2403</td>
<td>.2000</td>
<td>.0000</td>
<td>.1468</td>
<td>.2402</td>
</tr>
</tbody>
</table>
3.7 Estimation of the Square Root of a Determinant

The evaluation of the determinant of a large, sparse, and p.d. matrix is often of interest. For example, in testing hypothesis about the covariance matrix of a multivariate population, the likelihood ratio test statistic is a one-to-one function of the determinant. Moreover, the square root of the determinant appears in the constant of integration of the common multivariate distributions such as the multivariate normal and the multivariate t. Finally, the determinant is needed in the D.F. algorithm. The goal of this section is to show how the importance sampling algorithm can be successfully used to provide an efficient estimate of the square root of the determinant of a p.d. matrix.

To see this, let $W$ be a large, sparse, symmetric and p.d. matrix in $R^{n \times n}$. Let $V = W^{-1}$. Suppose it is of interest to compute $|V|^{\frac{1}{2}}$. Let, $x$ be a random vector having a multivariate normal with mean $0$ and covariance matrix $V$. That is, the probability density function of $x$ is:

\[ p(x) = c_0 \int f(x) dx \]  \hspace{1cm} (3.36)

where

\[ c_0 = (2\pi)^{\frac{n}{2}} |V|^{\frac{-n}{2}} \]  \hspace{1cm} (3.37)

and

\[ f(x) = e^{\frac{1}{2} x^T W x} \]  \hspace{1cm} (3.38)

and since

\[ c_0 \int f(x) dx = 1 \]  \hspace{1cm} (3.39)
then

\[
(2\pi)^{\frac{n}{2}} \int f(x)dx = |V|^{\frac{1}{2}}. \tag{3.40}
\]

\[
|V|^{\frac{1}{2}} = \frac{1}{\sqrt{\int f(x)^2 dx}}. \tag{3.41}
\]

Now, let \( h \) be an optimal importance sampler and \( u \) be a random vector having the distribution of \( h \). Then:

\[
(2\pi)^{\frac{n}{2}} \int \omega(x)h(x)dx = |V|^{\frac{1}{2}} \tag{3.42}
\]

where

\[
\omega(x) = \frac{f(x)}{h(x)}, \tag{3.43}
\]

i.e.

\[
E_h[\omega(u)] = |V|^{\frac{1}{2}} \tag{3.44}
\]

and hence, an estimator of \(|V|^{\frac{1}{2}}\) is:

\[
d^* = (2\pi)^{\frac{n}{2}} \frac{1}{N} \sum_{k=1}^{N} \omega(u_k). \tag{3.45}
\]

where, \( u_1, \ldots, u_N \) is a Monte Carlo sample of size \( N \) from \( h \). The estimator \( d^* \) has desirable properties:

1. \( d^* \) is unbiased as shown in Section 3.3

2. With an optimal importance sampler and sufficiently large \( N \), \( var_h(\omega) \) may be small.

The comparison of this approach to evaluate the determinant and other methods is a topic for further research.
4 AN ADAPTIVE IMPORTANCE SAMPLING
ALGORITHM

The goal of this chapter is to describe a method for choosing a good importance function and provide evidence for the need of selecting a reasonable one.

4.1 The General Algorithm

In section 3.4 of the previous chapter, we have defined the three desirable properties a density should have to be considered as an optimal importance sampler. Finding a density that satisfies those conditions is not trivial. In practice, the choice of a 'good' \( h \) is done in the following manner:

1. First choose a parametric family of densities

\[
H = \{h_\lambda, \quad \lambda \in \Lambda\} \quad (4.1)
\]

2. Then, choose, \( h_\lambda \) in \( H \), such as it is simple to generate random number from \( h_\lambda \) and \( h_\lambda \) has a thicker tail than \( f \).

3. Choose \( \lambda \) to match some characteristics of \( f \). For example, \( \lambda \) may be the mean vector and/or the covariance matrix of \( h \).

Man-Suk Oh [35] describes three ways for choosing \( \lambda \): matching the importance function \( h \) to \( f \), Adaptive Importance Sampling, and Chaining.
### 4.1.1 Matching h and f

There are three approaches to matching [35]. The first is to match a t importance function $h$ with $f$. That is, one chooses the importance function $h$ from the $MVt$ distribution with degrees of freedom $\nu$ and covariance $\Sigma_h$ and set:

$$\Sigma_h = ((\nu - 2)/\nu)\Sigma_f$$

(4.2)

where $\Sigma_f$ denotes the covariance matrix of $f$.

The second approach is to match the minus inverse Hessian matrix of log $f$ to the minus inverse Hessian of log $h$. That is, set:

$$\Sigma_h = ((\nu + n)/\nu)(-\hat{I})_f$$

(4.3)

where $\hat{I}_f$ is the minus inverse Hessian matrix of log $f$.

The third approach to matching is to set:

$$\Sigma_h = \rho^*\Sigma_f$$

(4.4)

where $\rho^*$ is the constant that minimizes var($\omega$) with respect to $\rho$ when $h$ is chosen from the family of distribution with covariance matrix $\rho\Sigma_f$.

There are 2 drawbacks with these three approaches to matching:

1. All three approaches required the knowledge of $f$ or an approximation to $f$. However in the inversion problem in hand, $f$ is unknown.

2. In the second approach, $(\nu + n)/\nu$ increases with $n$. So that when $n$ is large, $\Sigma_h$ may be an inefficient estimator of $\Sigma_f$ unless $f$ itself is a $MVt$ [35].
4.1.2 Adaptive Importance Sampling

Adaptive Importance Sampling (A.I.S., hereinafter) was developed to address the problem of a good choice of $\lambda$. A.I.S. is an iterative importance sampling algorithm that updates the parameter of the importance function at each iteration with simple linear operations. Michael Evans [12] describes the general A.I.S. as follows:

1. Choose $h_{\lambda_0} \in H$

2. Generate, $u_1, \ldots, u_N$ from $h_{\lambda_0}$

3. Compute the estimate $\hat{\theta}_0$ using 3.12

4. Find $\lambda_1 \in \Lambda$ such that $||\theta(\lambda_1) - \hat{\theta}_0||$ is minimized.

5. Generate, $u_{N+1}, \ldots, u_{2N}$ from $h_{\lambda_1}$ and combine with $u_1, \ldots, u_N$ to obtain the estimate $\hat{\theta}_1$

6. Iterate between 4 and 5 until $\hat{\theta}_i$ is stable.

7. Use the final $h_{\lambda}$ to compute other integrals of interest.

The advantage of the above scheme is that it encodes the information learned about $h_{\lambda}$ during sampling and uses it to improve the estimation of $\theta$.

4.2 Estimation of the Elements of $V$ using A.I.S

4.2.1 Background Materials

Let $x_{n \times 1} \sim \text{MVN}(0, V)$. Suppose that we are interested in estimating the elements, $v_{ij}, \ i, j = 1, \ldots, n$ of $V$. Recall that (see Section 2.3) for $i = 1, \ldots, n$

$$v_{ii} = 1/w_{ii} + E_x[g_{ii}(x)]/w_{ii}^2 \quad (4.5)$$
where

\[ g_{ii}(x) = \left( \sum_{k \neq i} w_{ik} x_k \right)^2 \]  

where for \( i \neq j = 1, \ldots, n \)

\[ u_{ij} = -\frac{w_{ij}}{\Delta_{ij}} + E_x [g_{(ij)}(x)] / \Delta_{ij}^2 \]  

where

\[ \Delta_{ij} = w_{ii} * w_{jj} - w_{ij}^2 \]  

and

\[ g_{ij}(x) = \left( \sum_{k \neq i, k \neq j} w_{ik} x_k - w_{ij} \sum_{k \neq j, k \neq i} w_{jk} x_k \right) \left( -w_{ij} \sum_{k \neq i, k \neq j} w_{jk} x_k + w_{ii} \sum_{k \neq j, k \neq i} \right) \]  

Let

\[ g = (g_{11}, g_{12}, \ldots, g_{nn})^T \]  

and

\[ H = \{ h \Sigma, \Sigma \text{ is a diagonal, p.d. matrix} \} \]  

where \( h \Sigma \) will be chosen from either the from the multivariate normal with covariance matrix \( \Sigma \) or the multivariate t distribution with covariance matrix \( \Sigma \) and \( \nu \) degrees of freedom. Therefore, in step 4 of the previous algorithm, we will minimize the following quantity:

- \( ||\theta(\Sigma) - \theta(\Sigma_0)|| \) in the case of the multivariate normal

- \( ||\theta(\Sigma) - ((\nu - 2)/\nu)\theta(\Sigma_0)|| \) in the case of the multivariate t with \( \nu \) degrees of freedom.
Our goal in what follows is to obtain a reasonable estimate for $\mathbb{E}_x[g]$, and $\hat{\mathbb{E}}_{(k)}(g)$ will denote the sought after estimate. To this end, we make use of the following version of the A.I.S. suggested by [35]. In what follows, the subscript $(k)$ denotes the stage $k$ at which a quantity is computed. For example, $S_{(k)}$ is the value of $S$ at stage $k$, and $\omega_{(k)}$ is the value of the weight at stage $k$. In the same manner,

1. $N_{(k)}$ is the total sample size at stage $k$.

2. $\bar{\omega}_{(k)}$ is the average of all the importance weights from stage 1 to stage $k$.

3. $\text{var}_{(k)}(\omega)$ is the variance of the importance weights at stage $k$.

4. $\text{var}_{(k)}(g\omega)$ is the variance of the product of a given component $g_{ij}$ of 4.10 and $\omega$ at stage $k$.

5. $\text{cov}_{(k)}(g\omega, \omega)$ denotes the covariance between $g\omega$ and $\omega$ at stage $k$.

6. $\hat{\sigma}^2_{(k)}$ is the estimate variance of $\hat{\mathbb{E}}_{(k)}(g)$.

7. And finally, $\phi(u)$ will denote any well-defined function of $u$.

8. In the following algorithm, we need not consider $\Sigma$ of the form 4.2 since in this algorithm the parameter $(\nu - 2)/\nu$ is a constant and cancels out during the computation.

### 4.2.2 The Algorithm

1. Choose an initial guess (or estimate) $\Sigma_{(0)}$ and let $h_{(0)} = h_{\Sigma_{(0)}}$. In addition, choose a stopping rule. The choice of the stopping rule will be explained below.

2. For $k \geq 1$ draw $n_k$ i.i.d. random vectors $u_1, u_2, \ldots, u_{n_k}$ from $h_{(k-1)}$, where $h_{(k-1)}$ is the importance function with covariance matrix $\Sigma_{(k-1)}$. 
3. For each of the \( u_j, \ j = 1, \ldots, n_k \), compute

\[
\omega(k)(u) = \frac{f(u)}{h_{(k-1)}(u)}
\]  
(4.12)

where \( f \) is the unknown density with covariance \( V \).

4. Then compute the quantities:

(a) \( S(k)(g_{11}), S(k)(g_{12}), \ldots, S(k)(g_{nn}) \)

(b) \( S(k)(1) \)

where

\[
S(k)(\phi) = \sum_{j=1}^{n_k} \phi(u_j) \omega(k)(u_j).
\]  
(4.13)

5. Evaluate the following quantities

(a)

\[
N(k) = \sum_{j=1}^{k} n_j
\]  
(4.14)

\[
\bar{\omega}(k) = \frac{\sum_{j=1}^{k} S(j)(1)}{N(k)}
\]  
(4.15)

\[
\bar{\text{var}}(k)(gw) = \frac{\sum_{j=1}^{k} S(j)(g^2 \omega(j))}{N(k)} - \left[ \bar{E}(k)(g) \right]^2 \left[ \bar{\omega}(k) \right]^2
\]  
(4.16)

\[
\bar{\text{cov}}(k)(gw, w) = \frac{\sum_{j=1}^{k} S(j)(gw(j))}{N(k)} - \left[ \bar{E}(k)(g) \right] \left[ \bar{\omega}(k) \right] \left[ \bar{\omega}(k) \right]^2
\]  
(4.17)

\[
\bar{\text{var}}(k)(\omega) = \frac{\sum_{j=1}^{k} S(j)(\omega(j))}{N(k)} - \left[ \bar{\omega}(k) \right] \left[ \bar{\omega}(k) \right]^2
\]  
(4.18)

(b)

\[
\frac{\sigma(k)^2}{\bar{\omega}(k)^2} = \frac{[\bar{\text{var}}(k)(gw) - 2(\bar{E}(k)g) \bar{\text{cov}}(k)(gw, w) + (\bar{E}(k)g)^2 \bar{\text{var}}(k)(\omega)]}{[\bar{\omega}(k)]^2}
\]  
(4.19)
6. Check the following stopping rule:
\[
\frac{\hat{\text{var}}_{(k)}(\omega)}{N_{(k)}\hat{\omega}_{(k)}} \leq \left( \frac{\epsilon}{Z_{\alpha}} \right)^2
\] (4.20)

where, \( Z_{\alpha} \) is the upper \( \frac{\alpha}{2} \) quantile of the standard normal and, \( 0 < \epsilon < 1 \)

7. Decision:

(a) If 4.20 is not satisfied, then, set:
\[
\hat{\Sigma}_k = \left( \frac{S_{(k)}(g_{11})}{S_{(k)}(1)}, \frac{S_{(k)}(g_{22})}{S_{(k)}(1)}, \ldots, \frac{S_{(k)}(g_{nn})}{S_{(k)}(1)} \right)
\] (4.21)

and set \( k = k + 1 \) and go to step 2.

(b) If 4.20 is satisfied, then go to step 8.

8. Conclusion:
\[
\hat{E}_{(k)}(g) = \left( \frac{\sum_{j=1}^{k} S_{(j)}(g_{11})}{\sum_{j=1}^{k} S_{(j)}(1)}, \frac{\sum_{j=1}^{k} S_{(j)}(g_{22})}{\sum_{j=1}^{k} S_{(j)}(1)}, \ldots, \frac{\sum_{j=1}^{k} S_{(j)}(g_{nn})}{\sum_{j=1}^{k} S_{(j)}(1)} \right)
\] (4.22)

and
\[
\hat{E}_{(k)}(g) \approx N(\mathbb{E}(g), \sigma_{(k)}^2) \quad \text{as } n_k \rightarrow \infty
\] (4.23)

That is, as the sample size gets larger and larger, the estimate of the expected value, \( \hat{E}_{(k)}(g) \), is approximately normal with mean equal to the true unknown value \( \mathbb{E}(g) \) and variance \( \sigma_{(k)}^2 \) as given in 4.19. This approximation to normality makes it possible to obtain the stopping rule in 4.20 via the Chebyshev inequality. For more details on the derivation of 4.20, see [35].

As we can see, an important issue in A.I.S. is the choice of the starting parameter \( \Sigma_0 \) (or \( \lambda_0 \) in the general algorithm). For the specific problem of evaluating the elements of the inverse, possible starting matrices are the following:
1. \[
\Sigma_0^{(1)} = I_{n \times n}, 
\] (4.24)
which corresponds to \( h \) being the Standard MVt \((0, I)\) and

2. \[
\Sigma_0^{(2)} = \text{Diag}(1/w(i, i)) 
\] (4.25)
and

\[
\Sigma_0^{(3)} = \text{Diag}(1/w(i, i) + 1/(w(i, i) * 2)), 
\] (4.26)
which are suggested by the swindle decompositions of Section 3 of Chapter 2.

An alternative way for choosing \( \Sigma_0 \) is the technique of chaining suggested by Michael Evans in [12]. For more details on chaining see [12].

### 4.3 Example and Numerical Comparisons

Our goal in this section and in the context of estimating the diagonal elements of \( V \) we shall:

1. Compare the effect of different starting covariance matrices \( \Sigma_0 \) on the convergence and accuracy of the A.I.S.

2. Compare the performance of the A.I.S. with that of the ordinary Importance Sampling
4.3.1 Effect of the Starting Covariance Matrix

In this section we shall especially investigate the effect of the 3 starting covariance matrices $\Sigma_0^{(1)}$, $\Sigma_0^{(2)}$, and $\Sigma_0^{(3)}$ as defined by (4.24), (4.25), and (4.26) on the convergence and accuracy of the A.I.S. estimates. For this purpose we use the following matrix:

$$W = \begin{bmatrix}
4 & 1 & 2 & 1 & 2 \\
1 & 5 & 0 & 0 & 0 \\
2 & 0 & 3 & 0 & 0 \\
1 & 0 & 0 & 5 & 0 \\
2 & 0 & 0 & 0 & 8 \\
\end{bmatrix} \quad (4.27)$$

Then we proceed as follows:

1. We set $\Sigma_0 = \Sigma_0^{(1)}$ and $h\Sigma_0^{(1)}$ to be the multivariate $t$ with 3 d.f. and variance covariance matrix $\Sigma_0^{(1)}$.

2. Then we generate $n_1 = 300$ i.i.d., $5 \times 1$ random vectors, $u_1, \ldots, u_{300}$ from $h\Sigma_0^{(1)}$.

3. We also compute the quantities in step 3, 4, and 5 in the above algorithm, and the stopping rule in (4.20).

4. We then iterate between 1 and 7(a) of the above algorithm until (4.20) is satisfied.

5. When (4.20) is satisfied, we compute the estimators in (4.22) and compute all the 25 elements of the inverse of $W$.

6. We then, repeat the above for $\Sigma_0^{(2)}$, and $\Sigma_0^{(3)}$. The results are displayed in Table 4.1 and Table 4.2.

These studies were repeated for various matrices with dimension $n = 5$, $n = 7$, $n = 10$, and $n = 397$ and the results were compatible with the ones shown in Table 4.1 and Table 4.2. Therefore, we decided to include only the results in these tables.

Table 4.1 and Table 4.2 display the inverse of $W$ when each of the above three matrices is used as a starting value. It appears that a smaller sample size is needed to
achieve convergence when A.I.S. is started with $\Sigma_0^{(3)}$ than the first two. In other words, A.I.S. converges faster with $\Sigma_0^{(3)}$ as starting value. The message is that the closer the starting covariance matrix is to the true inverse, the faster A.I.S. will converge.

4.3.2 Importance Sampling versus Adaptive Importance Sampling

In chapter 3, we have found out that the ordinary importance sampling performs better if the covariance matrix is $\Sigma_0^{(3)}$. In the previous section we have found out that is better to start A.I.S. with $\Sigma_0^{(3)}$. Let us now see how A.I.S. performs against the ordinary importance sampling, starting both algorithms with $\Sigma_0^{(3)}$. To do this, we consider the following matrix $W^*$:

$$
W^* = \begin{bmatrix}
4 & 0 & 1 & 0 & 2 \\
0 & 5 & 0 & 0 & 0 \\
1 & 0 & 3 & 0 & 1 \\
0 & 0 & 0 & 5 & 0 \\
2 & 0 & 1 & 0 & 8
\end{bmatrix} \tag{4.28}
$$

Then we proceed as follows:

1. Beginning with the stopping rule defined in 4.20, we first run the ordinary importance sampling on $W^*$ by generating 10,800 i.i.d. and $5 \times 1$ vectors $u$'s from the multivariate $t_3$ and compute the estimate $\hat{\theta}$ and $\hat{v}_{ij}^*$.

2. We then run A.I.S. on $W^*$ as in the previous section using the same stopping rule, and computing the appropriate estimates. The results are displayed in Table 4.3.

3. These simulations were repeated for various matrices $W^*$, with dimension $n = 5$, $n = 7$, $n = 10$, $n = 397$. The results were compatible with those in Table 4.3.

Therefore, only the results in Table 4.3 were displayed.

Table 4.3 shows the results of the evaluation of the elements of the inverse of $W^*$. With roughly the same accuracy, A.I.S. achieves convergence with 4 or 5 times fewer samples
than I.S. In other words, with comparable level of accuracy, and $\Sigma_0^{(3)}$ as a starting value, A.I.S. run faster than the ordinary I.S.

4.4 Summary and Discussion

The main findings of this chapter can be summarized as follow:

1. Starting with any of the three covariances defined in 4.24, 4.25, and 4.26, A.I.S. will converge to the true unknown value. However, A.I.S. will converge faster and will be more accurate when started with $\Sigma_0^{(3)}$

2. With the same covariance matrix, A.I.S. converges with fewer iterations and has better accuracy than the ordinary importance sampling

3. The ordinary importance sampling gives reasonable results for the estimation problem at hand

4. Both A.I.S. and the ordinary importance sampling compute exactly some elements of the inverse $V$, i.e., no bias and zero mean square error. By doing so, they both take advantage of the decomposition swindle and the sparsity of $W$.

The next question is how to implement A.I.S. when $W$ is the order of 100 to 1,000,000? The issue involved in the large scale implementation of A.I.S. is two-fold:

1. storage problem: For example, if $W$ is of order 10,000, then it will require $8 \times 10,000 \times 10,000$ bits to store only $W$.

2. computing time: Again if $W$ is of order 10,000 then there are: $10,000 + 10,000 \times 9,999 \times 9,998$ functions $g_{ij}$ to be computed.

Therefore, there is a need to design an A.I.S. algorithm that takes advantage both of the sparsity of $W$ and the fact that the computation of most of the quantities needed can be done independently. One way to do that is to use an efficient storage technique and parallel processing. These two topics are discussed in the next two chapters.
### Table 1.1 Comparison of Starting Covariance Matrices: Diagonal Elements

<table>
<thead>
<tr>
<th>Method</th>
<th>Sample Size</th>
<th>$\hat{v}_{11}$ Estimate</th>
<th>s.e. 10^{-2}</th>
<th>$\hat{v}_{22}$ Estimate</th>
<th>s.e. 10^{-2}</th>
<th>$\hat{v}_{33}$ Estimate</th>
<th>s.e. 10^{-2}</th>
<th>$\hat{v}_{44}$ Estimate</th>
<th>s.e. 10^{-2}</th>
<th>$\hat{v}_{55}$ Estimate</th>
<th>s.e. 10^{-2}</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.I.S. with $\Sigma^{(1)}$</td>
<td>19800</td>
<td>.5666</td>
<td>.8607</td>
<td>.2229</td>
<td>.0509</td>
<td>.5859</td>
<td>.5619</td>
<td>.2229</td>
<td>.0509</td>
<td>.1605</td>
<td>.07898</td>
</tr>
<tr>
<td>A.I.S. with $\Sigma^{(2)}$</td>
<td>18300</td>
<td>.5676</td>
<td>1.0438</td>
<td>.2229</td>
<td>.0542</td>
<td>.5872</td>
<td>.5966</td>
<td>.2229</td>
<td>.0542</td>
<td>.1607</td>
<td>.0850</td>
</tr>
<tr>
<td>A.I.S. with $\Sigma^{(3)}$</td>
<td>17100</td>
<td>.5663</td>
<td>.9360</td>
<td>.2227</td>
<td>.0512</td>
<td>.5854</td>
<td>.5578</td>
<td>.2227</td>
<td>.0512</td>
<td>.1604</td>
<td>.0786</td>
</tr>
<tr>
<td>True Value</td>
<td></td>
<td>.5660</td>
<td></td>
<td>.2226</td>
<td></td>
<td>.5849</td>
<td></td>
<td>.2226</td>
<td></td>
<td>.1604</td>
<td></td>
</tr>
</tbody>
</table>

### Table 1.2 Comparison of Starting Covariance Matrices: Off diagonal Elements

<table>
<thead>
<tr>
<th>Method</th>
<th>Sample Size</th>
<th>$\hat{v}_{12}$ Estimate</th>
<th>s.e. 10^{-2}</th>
<th>$\hat{v}_{13}$ Estimate</th>
<th>s.e. 10^{-2}</th>
<th>$\hat{v}_{14}$ Estimate</th>
<th>s.e. 10^{-2}</th>
<th>$\hat{v}_{15}$ Estimate</th>
<th>s.e. 10^{-2}</th>
<th>$\hat{v}_{23}$ Estimate</th>
<th>s.e. 10^{-2}</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.I.S. with $\Sigma^{(1)}$</td>
<td>19800</td>
<td>-.1133</td>
<td>.1593</td>
<td>-.3864</td>
<td>.2824</td>
<td>-.1133</td>
<td>.15477</td>
<td>-.1406</td>
<td>.1856</td>
<td>.0758</td>
<td>.1684</td>
</tr>
<tr>
<td>A.I.S. with $\Sigma^{(2)}$</td>
<td>18300</td>
<td>-.1132</td>
<td>.1797</td>
<td>-.3861</td>
<td>.31091</td>
<td>-.1133</td>
<td>.1954</td>
<td>-.1406</td>
<td>.2192</td>
<td>.0762</td>
<td>.1797</td>
</tr>
<tr>
<td>A.I.S. with $\Sigma^{(3)}$</td>
<td>17100</td>
<td>-.1132</td>
<td>.1654</td>
<td>-.3815</td>
<td>.2832</td>
<td>-.1143</td>
<td>.1718</td>
<td>-.1393</td>
<td>.1979</td>
<td>.0747</td>
<td>.1675</td>
</tr>
<tr>
<td>True Value</td>
<td></td>
<td>-.1132</td>
<td></td>
<td>-.3774</td>
<td></td>
<td>-.1132</td>
<td></td>
<td>-.1415</td>
<td></td>
<td>.0755</td>
<td></td>
</tr>
<tr>
<td>Method</td>
<td>Sample Size</td>
<td>$\hat{v}_{24}$ Estimate</td>
<td>s.e. 10^{-2}</td>
<td>$\hat{v}_{25}$ Estimate</td>
<td>s.e. 10^{-2}</td>
<td>$\hat{v}_{34}$ Estimate</td>
<td>s.e. 10^{-2}</td>
<td>$\hat{v}_{35}$ Estimate</td>
<td>s.e. 10^{-2}</td>
<td>$\hat{v}_{45}$ Estimate</td>
<td>s.e. 10^{-2}</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------</td>
<td>--------------------------</td>
<td>--------------</td>
<td>--------------------------</td>
<td>--------------</td>
<td>--------------------------</td>
<td>--------------</td>
<td>--------------------------</td>
<td>--------------</td>
<td>--------------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>A.I.S. with $\Sigma^{(1)}$</td>
<td>19800</td>
<td>.0227</td>
<td>.0509</td>
<td>.0284</td>
<td>.0630</td>
<td>.0758</td>
<td>.1684</td>
<td>.0947</td>
<td>.2113</td>
<td>.0284</td>
<td>.0630</td>
</tr>
<tr>
<td>A.I.S. with $\Sigma^{(2)}$</td>
<td>18300</td>
<td>.0229</td>
<td>.0542</td>
<td>.0286</td>
<td>.0672</td>
<td>.0762</td>
<td>.1797</td>
<td>.0952</td>
<td>.2240</td>
<td>.286</td>
<td>.0672</td>
</tr>
<tr>
<td>A.I.S. with $\Sigma^{(3)}$</td>
<td>17100</td>
<td>.0227</td>
<td>.0512</td>
<td>.0284</td>
<td>.0631</td>
<td>.0762</td>
<td>.1675</td>
<td>.0952</td>
<td>.2096</td>
<td>.0286</td>
<td>.0631</td>
</tr>
<tr>
<td>True Value</td>
<td></td>
<td>.0226</td>
<td></td>
<td>.0283</td>
<td></td>
<td>.0754</td>
<td></td>
<td>.0943</td>
<td></td>
<td>.0283</td>
<td></td>
</tr>
</tbody>
</table>
Table 4.3 A.I.S. Estimates of Inverse of $W^*$: Nonzero Elements

<table>
<thead>
<tr>
<th>Method</th>
<th>Sample Size</th>
<th>$\hat{\nu}_{11}^*$</th>
<th>s.e. $10^{-3}$</th>
<th>$\hat{\nu}_{13}^*$</th>
<th>s.e. $10^{-3}$</th>
<th>$\hat{\nu}_{15}^*$</th>
<th>s.e. $10^{-3}$</th>
<th>$\hat{\nu}_{22}^*$</th>
<th>s.e. $10^{-3}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>I.S. with $\Sigma_0^{(3)}$</td>
<td>10800</td>
<td>.3050</td>
<td>.7371</td>
<td>-.0781</td>
<td>.1477</td>
<td>-.0655</td>
<td>.0882</td>
<td>.2000</td>
<td>.0000</td>
</tr>
<tr>
<td>True Value</td>
<td></td>
<td>.3026</td>
<td>-.789</td>
<td>-.0657</td>
<td>.1712</td>
<td>.2000</td>
<td>.0000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A.I.S. with $\Sigma_0^{(3)}$</td>
<td>2700</td>
<td>.3027</td>
<td>.2079</td>
<td>-.0787</td>
<td>.4206</td>
<td>-.0657</td>
<td>.1712</td>
<td>.2000</td>
<td>.0000</td>
</tr>
<tr>
<td>Method</td>
<td>Sample Size</td>
<td>$\hat{\nu}_{31}^*$</td>
<td>s.e. $10^{-3}$</td>
<td>$\hat{\nu}_{35}^*$</td>
<td>s.e. $10^{-3}$</td>
<td>$\hat{\nu}_{44}^*$</td>
<td>s.e. $10^{-3}$</td>
<td>$\hat{\nu}_{55}^*$</td>
<td>s.e. $10^{-3}$</td>
</tr>
<tr>
<td>I.S. with $\Sigma_0^{(3)}$</td>
<td>10800</td>
<td>.3696</td>
<td>.4064</td>
<td>-.0257</td>
<td>.3355</td>
<td>.2000</td>
<td>.0000</td>
<td>.1445</td>
<td>.3285</td>
</tr>
<tr>
<td>True Value</td>
<td></td>
<td>.3684</td>
<td>-.0263</td>
<td>.2000</td>
<td>.1447</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A.I.S. with $\Sigma_0^{(3)}$</td>
<td>2700</td>
<td>.3685</td>
<td>.7773</td>
<td>-.0264</td>
<td>.6388</td>
<td>.2000</td>
<td>.0000</td>
<td>.1447</td>
<td>.6097</td>
</tr>
</tbody>
</table>
5 SPARSE STORAGE, PARALLEL PROCESSING, AND APPLICATION

An important goal in the sparse matrix computations is to obtain accurate results in a reasonable amount of time. That is, design algorithms that are fast and yet accurate. One way to achieve this is to design algorithms that store and process only the non-zero element of the sparse matrix. This chapter deals with sparse storage schemes.

5.1 The Problem

One of the main challenges in sparse matrix computations is to take advantage of the sparsity of the matrix $W$ by finding an optimal technique to store the nonzero elements of $W$. To be optimal, a storage scheme should satisfy the following two conditions:

1. Be as simple as possible

2. Be efficient in performing basic linear operations such as matrix-vector product, dot product, Gaussian Elimination, Cholesky factorization and similar numerical algorithms.

Usually, these two conditions work in opposite directions (see Zlatev [38]). In addition, an optimal storage scheme depends on the type of linear operations that need to be performed. In sparse matrix computations, there are two types of operations:

1. Operations in which fill-ins occur, such as Gaussian elimination, LU factorization, and orthogonal transformation. For these operations, the storage scheme of choice
is the dynamic storage scheme of Zlatev (see [38]). A detailed explanation of these scheme is given in [38].

2. Operations in which no fill-ins occur. For such operations, two most widely used storage schemes are the input storage scheme of Zlatev et al., and the row pointer/column scheme of Duff et al.(see [10]). Both of these scheme are described next.

5.2 The Input Storage Scheme

5.2.1 The Scheme

Let nz denote the number of nonzero elements of the sparse matrix \( W \) and \( wnz \) a real array containing those nonzero elements. Also, let \( cw \) be an integer array that holds the column subscripts of the elements of \( W \) and \( rw \) be an integer array that holds the row subscripts of the elements of \( W \). In addition, let \( len \) be an integer array that holds the numbers of nonzero entries in each row of \( W \). Suppose a nonzero element \( w_{ij} \) of \( W \) is stored in the kth position, (say), in the array \( wnz \), then we have:

\[
wnz(k) = w_{ij} \quad (5.1)
\]

\[
rw(k) = i \quad (5.2)
\]

and

\[
cw(k) = j. \quad (5.3)
\]

To illustrate the input storage scheme, consider the following matrix:

\[
W = \begin{bmatrix}
4 & 0 & 1 & 0 & 2 \\
0 & 5 & 0 & 0 & 0 \\
1 & 0 & 3 & 0 & 1 \\
0 & 0 & 0 & 5 & 0 \\
2 & 0 & 1 & 0 & 8 \\
\end{bmatrix} \quad (5.4)
\]

Then, in the input storage scheme, \( W \) is stored as shown in the Table 5.1:
Table 5.1 Input Storage Scheme for the Matrix in 5.4

<table>
<thead>
<tr>
<th>Subscripts</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>cw</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>1</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>wnz</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>len</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

5.2.2 Advantages of the Input Storage Scheme

The input storage scheme has the following advantages:

1. it is simple.

2. it requires no order for storing the nonzero elements. That is the nonzero elements need not be in any particular order, such as arranged by row or column.

The input storage scheme is used in the NAG subroutines F01BRE and F04AXE (see [34]) as well as in two well-known sparse matrix packages, MA28 of Duff (see [7]) and Y12M of Wasniewski et al. (see [39]). Let us now illustrate the computation of a matrix-vector product of interest by the input storage scheme.

In the computation of the inverse of the sparse matrix $W$, we are mainly interested in matrix-vector products of the following form:

$$Wx_{(i)} = \sum_{k \neq i}^{n} w_{ik} x_k \quad \text{for } i = 1, \ldots, n$$  \hfill (5.5)

and for $i \neq j = 1, \ldots, n$

$$Wx_{(ij)} = (\sum_{k \neq i, k \neq j}^{n} w_{ik} x_k)(\sum_{k \neq i, k \neq j}^{n} w_{jk} x_k)$$  \hfill (5.6)

The computation of 5.5 using the input storage scheme may be described as follows:
For \( s = 1 : nz \)
\[
i \leftarrow \text{rw}(s)
\]
\[
k \leftarrow \text{cw}(s)
\]
\[
\text{if } i \text{ not equal } k \quad \text{sg}(i) \leftarrow \text{sg}(i) + \text{wnz}(l) \times x(k)
\]
end

5.2.3 Disadvantages of the Input Storage Scheme

The input storage scheme presents the following three drawbacks:

1. To access a nonzero element of \( W \), one must search through all of the \( nz \) elements.
   In practice, \( nz \) may be very large, in order of 10,000 to 1,000,000.

2. In addition, given two rows, \( i \) and \( j \), the input storage scheme is not efficient in locating the entry \( w_{ik} \) or \( w_{jk} \) for \( k \neq i \) and \( k \neq j \).

3. Three integer vectors and one real vector are needed to represent the sparse matrix \( W \). Fewer than 4 vectors may be sufficient.

The Row Pointer/Column Index scheme of the next section offers a more efficient alternative to the input storage scheme both in representing \( W \) as well as computing all the matrix-vector products of interest.

5.3 Row Pointer/Column Index Scheme

5.3.1 The Scheme

The row pointer/column index is row-oriented. That is, nonzero entries of each row are stored in consecutive locations, the rows are stored in order, and there is no space between rows. Concretely, let \( nz \), \( \text{wnz} \), and \( \text{wc} \) be defined as in the previous sections. In addition, let \( \text{istart} \) be an integer array that holds the location of the first entry of each
row and the first free location. Then the row pointer/column index may be described as follows:

1. The real array \( w_{nz} \) stores the nonzero elements of \( W \) by row in a consecutive order.

2. The integer array \( wc \) holds the corresponding row subscripts

3. Finally, the integer array \( istart \) holds the beginning of each row. That is, \( istart(i+1) - istart(i) = \) number of nonzero elements in row \( i \).

Let now illustrate the row pointer/column index scheme with the matrix \( W \) in 5.4

<table>
<thead>
<tr>
<th>Table 5.2 Row Pointer Scheme of the Matrix in 5.4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subscripts</td>
</tr>
<tr>
<td>( cw )</td>
</tr>
<tr>
<td>( w_{nz} )</td>
</tr>
<tr>
<td>( istart )</td>
</tr>
</tbody>
</table>

Thus, an entry, \( w_{ij} \) of \( W \) is located by the following code:

For \( i = 1:n \)

\[
\text{for } ii = istart(i):istart(i+1)-1 \\
\quad \text{if}(wc(ii) \text{ equals } i) \text{ then} \\
\quad \quad \text{wii <- w}_{nz}(ii) \\
\quad \text{else} \\
\quad \quad \text{if}(wc(ii) \text{ equals } j) \text{ wij <- w}_{nz}(ii) \\
\quad \text{endif} \\
\text{end}
\]
For example, suppose we want to locate the entry (3.5) in \( W \). Then \( i = 3 \) and \( j = 5 \).

Therefore,

\[
\text{istart}(3) = 5 \quad (5.7) \\
\text{istart}(3 + 1) - 1 = 7 \quad (5.8)
\]

It is only when \( i_1 = 7 \) that

\[
\text{cw}(i_1 = 7) = j = 3 \quad (5.9)
\]

and hence

\[
w_{35} = \text{wnz}(i_1 = 7) = 1 \quad (5.10)
\]

Therefore, the matrix-vector product in 5.5 may be computed as follow:

For \( i = 1:n \)

\[
g_{ii} \leftarrow 0
\]

for \( i_1 = \text{istart}(i): \text{istart}(i+1)-1 \)

\[
\text{if}(\text{wc}(i_1) \text{ not equal } i) \ g_{ii} \leftarrow g_{ii} + \text{wnz}(i_1) \star u(\text{wc}(i_1))
\]

end

Moreover, the matrix-vector product in 5.6 is efficiently computed with the following code:

For \( i = 1:n \)

\[
\text{for } j = i+1:n
\]
wiktu <- 0
wjktu <- 0
for k = istart(j) : istart(j+1)-1
    if(wc(k) not equal i and wc(k) not equal j)
        wjktu <- wjktu + wnz(k)*u(wc(k))
    end
* 
for l = istart(i) : istart(i+1)-1
    if(wc(l) not equal i and wc(l) not equal j)
        wiktu <- wiktu + wnz(l)*u(wc(l))
    end
* 
end
end

5.3.2 Advantages of The Row Pointer/Column Index Scheme

As seen in the previous section, the row pointer/column index scheme offers the following 4 advantages:

1. It is relatively simple to use

2. It is efficient in locating each entry of the sparse matrix W. For example, to find, \( w_{ij} \), we needed only to search only from \( \text{istart} (i) \) to \( \text{istart} (i+1) - 1 \), which will have at most \( n \) elements, compared to the \( nz \) elements of the input storage scheme.

3. It is efficient in the computation of the matrix-vector products of interest.
4. Only two integer vectors and one real vector are needed to represent each sparse matrix.

5.4 Other Storage Schemes and References

Many other storage algorithms are available. For example

1. The *Link-list* scheme of Duff (see [7])

2. The *Standard Sparse Matrix Format* of Duff et al. (see [8])

3. The *vectorizable* storage scheme used in *ITPACK*. (see [28])

4. The *band matrices* storage scheme. (see [34])

5. The *frontal and multi frontal* techniques, of I.S. Duff and J. K. Reid, cited by ([38])

6. The *graph theory* techniques. (see [18])

Among all the schemes that are available, the row sparse format is more suitable and efficient for the computation of the matrix-vector product needed in AIS.

5.5 Application to the Tribolium Data

As an application, let’s compute the inverse of the *relationship matrix*, \( \mathbf{A} \) of the Tribolium Casteneum (flour beetles) data. The relationship matrix was defined in section 1 for the MME.
5.5.1 Description of the Data and the Model

The data and their description are taken from Lin (see [29]). The experiment consists of a total of 54 matings of 73 parent flour beetles on 3 separate days. Six offspring from each mating were raised in a single bottle, giving a total of 324 second generation flour beetles. The following data were recorded on each second generation insects:

1. time, indicating whether second generation progeny are from day 1, 2, or 3 of the mating.
2. sex
3. sire(father)
4. dam(mother)
5. family: a family consists of beetles from one the 54 matings.
6. the weight

The resulting mixed linear model is:

\[ y = Xb + Z_1a + Z_2c + e \]  \hspace{1cm} (5.11)

where

\[ y = \text{a } 324 \times 1 \text{ vector of weights}, \]  \hspace{1cm} (5.12)

\[ X = \text{a } 324 \times 6 \text{ design matrix for fixed effects: time, sex, and time by sex interaction}, \]  \hspace{1cm} (5.13)

\[ Z_1 = \text{the matrix of insect random effects}, \]  \hspace{1cm} (5.14)

\[ Z_2 = \text{the } 324 \times 54 \text{ design matrix for family random effects}, \]  \hspace{1cm} (5.15)

with

\[ \text{var}(a) = G = \sigma_a^2 A, \]  \hspace{1cm} (5.16)
where \( A \) is the relationship matrix,

\[
\text{var}(c) = \sigma_c^2 I, \tag{5.17}
\]

\[
\text{var}(e) = \sigma_e^2 I, \tag{5.18}
\]

and

\[
\text{cov}(a, e) = \text{cov}(c, e) = 0 \tag{5.19}
\]

In both the classical and the Bayesian analysis of mixed model, the inverse of the relation matrix \( A \) is needed.

In the classical approach, the Henderson’s Mixed Model Equations:

\[
\begin{bmatrix}
X^T X & X^T Z_1 & X^T Z_2 \\
Z_1^T X & Z_1^T Z_1 + \frac{\sigma_e^2}{\sigma_a^2} A^{-1} & Z_1^T Z_2 \\
Z_2^T X & Z_2^T Z_1 & Z_2^T Z_2 + \frac{\sigma_e^2}{\sigma_a^2} I
\end{bmatrix}
\begin{bmatrix}
b \\
\hat{u} \\
\hat{c}
\end{bmatrix}
= \begin{bmatrix}
X^T y \\
Z_1^T y \\
Z_2^T y
\end{bmatrix} \tag{5.20}
\]

needed to be solved.

In the Bayesian approach, the relevant conditional posterior distributions need to be computed. In particular,

\[
p(u | y, c, b, \sigma_c^2, \sigma_e^2) = N \begin{pmatrix}
(Z_1^T Z_1 \\
+ \frac{\sigma_e^2}{\sigma_a^2} A^{-1})^{-1} Z_1^T (y - Xb - Z_2c), \\
\sigma_e^2 (Z_1^T Z_1 \\
+ \frac{\sigma_e^2}{\sigma_a^2} A^{-1})^{-1}
\end{pmatrix} \tag{5.21}
\]

5.5.2 \( A \) and its Inverse

As seen above, the inverse of the relationship matrix is needed both in the Bayesian and the classical analysis. The relationship matrix \( A \) is symmetric, sparse, and p.d.. Its diagonal entries, \( a_{ii}, i = 1, \ldots, 397 \) are equal to one plus the coefficient of inbreeding
(that is the expected fraction of genes common by descent). The off-diagonal entries, \(a_{ij}, i \neq j = 1, \ldots, 397\), represent the *additive genetic relationship*, that is the expected number of genes in common between the \(i\)th and \(j\)th flour beetle. For the flour beetles data, \(A\) and \(A^{-1}\) can be accessed in `/home/bchitou/stat699/fortran/sainv.dat`. The first row of \(A^{-1}\) is displayed in Table 5.3.

**Table 5.3 First Row of the \(A^{-1}\)**

<table>
<thead>
<tr>
<th>Column</th>
<th>A.I.S. Estimates</th>
<th>Std.Error</th>
<th>(10^{-3})</th>
<th>Cholesky Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10.0001</td>
<td>.6412</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>2.9989</td>
<td>2.0764</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>2.9989</td>
<td>2.0764</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>2.9989</td>
<td>2.0764</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>110</td>
<td>-1.0003</td>
<td>.4047</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>111</td>
<td>-1.0003</td>
<td>.4047</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>112</td>
<td>-1.0003</td>
<td>.4047</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>113</td>
<td>-1.0003</td>
<td>.4047</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>114</td>
<td>-1.0003</td>
<td>.4047</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>115</td>
<td>-1.0003</td>
<td>.4047</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>206</td>
<td>-1.0003</td>
<td>.4047</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>207</td>
<td>-1.0003</td>
<td>.4047</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>208</td>
<td>-1.0003</td>
<td>.4047</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>209</td>
<td>-1.0003</td>
<td>.4047</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>210</td>
<td>-1.0003</td>
<td>.4047</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>211</td>
<td>-1.0003</td>
<td>.4047</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>308</td>
<td>-1.0003</td>
<td>.4047</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>309</td>
<td>-1.0003</td>
<td>.4047</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>310</td>
<td>-1.0003</td>
<td>.4047</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>311</td>
<td>-1.0003</td>
<td>.4047</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>312</td>
<td>-1.0003</td>
<td>.4047</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>313</td>
<td>-1.0003</td>
<td>.4047</td>
<td>-1</td>
<td></td>
</tr>
</tbody>
</table>
When the number of nonzero elements of $W$ is very large (on order of 10,000 or more), A.I.S. could turn out to be slow. To achieve fast and reasonable execution time, we investigate the implementation of A.I.S. on parallel processors. Almasi and Gottlieb (see [1]) define parallel processing as a collection of processors that can communicate and cooperate to solve large problems fast. And, we are precisely interested in knowing if A.I.S. can be speeded up by combining the computing power of two or more processors. This inquiry in itself raises two basic questions:

1. Is A.I.S. suitable for parallel processing? If yes, why?

2. Under what parallel computing environments may A.I.S. be implemented? And how?

This chapter discusses the implementation of A.I.S. under two types of parallel environments. Section 2 describes the Distributed System environment and section 3 the Shared Memory environment. In section 1, we shall examine reasons for which A.I.S. is suitable for parallel processing.
6.1 Adaptive Importance Sampling is Parallelizable

There are two reasons for which A.I.S. is suitable for parallel implementation:

1. A.I.S. is a task partitioning algorithm, that is, A.I.S. consists of a set of independent computing segments. More precisely, the implementation of A.I.S. is a set of independent loops and function calls. Since different iterations of the loops are independent of each other, they may be executed on different processors simultaneously.

2. A.I.S. may also be described as a set of independent tasks that can be performed asynchronously, requiring very little communication between processors.

Dedoncker and Kapenga (see [5]), as well as Almasi and Gottlieb (see [1]), describe six main environments under which parallel implementation of adaptive integration is suitable. These include:

1. Multiple Instruction Multiple Data with Shared Memory, or MIMD-SM for short.
2. Multiple Instruction Multiple Data with Distributed Memory, or MIMD-DM
3. Single Instruction Multiple Data, or SIMD
4. Very Long Instruction Word, or VLIW
5. Vector Processor, or VP
6. And finally, Distributed System, or DS

For a detailed description of these environments, see [5] and [1]. Among the six environments cited above, there are only two available at Iowa State at the present time. These are the MIMD-DM and the MIMD-SM. In section 2 we shall introduce the DM environment and in section 3 the MIMD-SM environment.
6.2 The Distributed Memory Environment

In this section, we shall discuss how to implement A.I.S. by combining the computing power of two or more Unix work stations.

By *loosely coupling* (see [5]) two or more processors as a single *virtual* parallel machine, we obtain what is known in the literature as *Distributed Memory* system. In the *distributed memory* environment, each processor has its own *local* memory and a copy of the program to be executed. Processors communicate with each other by sending and receiving messages. There are four major issues that needed to be addressed in the implementation of A.I.S. on a distributed system. These are:

1. How do processors communicate with each other?
2. How is tasks load balanced among processors?
3. How are data distributed among the local memory of each processor?
4. How are the efficiency and the speed-up due to parallelization measured?

6.2.1 Interprocessor Communication and Load Balancing

In the distributed memory environment, communication between processors is usually handled by a message passing algorithm. One of the most-used message passing schemes is the *Parallel Virtual Machine* or *PVM*. *PVM* is a software that allows a network of heterogeneous UNIX computers to be used as a single *virtual* parallel computer. *PVM* supplies the functions to automatically start up tasks on each processor and allows them to communicate and synchronize with each other.

*PVM* was developed in 1989 at Oak Ridge National Laboratory by Al Geist et al. (see Geist et al. [15]). The *PVM* software and related packages can be freely downloaded from the netlib homepage at netlib@ornl.gov or by an anonymous FTP from cs.utk.edu pub/xnetlib.
With PVM protocol, sending and receiving messages is described as follows:

1. Each task in the virtual machine is given a task identifier (tid) which is a 32-bit integer.

2. Then, sending a message is done by a user callable function, `pvmfpsend`, available in the PVM user library. The construct of the `pvmfpsend` is:

   ```
call pvmfpsend(tid, ..., type, ...) 
```

   The routine `pvmfpsend` packs and sends an array of the specified data type to the task identified by tid.

3. Receiving a message is also handled by a user callable function, `pvmfrecv`, available in the PVM user library. The call to `pvmfrecv` is:

   ```
call pvmfrecv(tid, msgtag, ...) 
```

   The routine `pvmfrecv` will wait until a message with label `msgtag` arrived from the task identified by tid.

In addition to receiving and sending messages, communication between processors is enhanced by load balancing. A load is the size and the number of tasks assigned to each processor. A parallel computation is load balanced if each processor has roughly the same amount of computation and communication to perform at each step (see Golub and Van Loan [21]). There are many load balancing schemes, (see Geist et al. [15], Mascagni [30], and [21]). The simplest one is static load balancing. In static load balancing, the problem is divided up, and tasks are assigned to processors only once. For example, if there are 400 integrals to compute and there are 4 processors, each processor will compute 100 integrals. In general, suppose there are p processors and m tasks. Let, 

\[ m = pr + q \]

with \( 0 \leq q \leq p \). Then, processor 1, ..., processor q may handle each \( r + 1 \) tasks and processor \( q + 1, \ldots \), processor p may each handle \( r \) tasks. All tasks are active from the beginning and can communicate and coordinate with one another.
6.2.2 Data Structures

Let $x \in \mathbb{R}^n$ be a datum that will be used by the local memory of $p$ processors in the distributed environment. Write $n = pr + q$. Golub and Van Loan (see [21]) suggest two ways that data may be distributed among the $p$ processors. These are: store-by-row and store-by-column.

In a store-by-row scheme, the vector $x$ is considered as $p \times r$ matrix and each processor stores each row of that matrix in its own local memory. That is, processor 1, \ldots, processor $q$ will each house $r + 1$ components of $x$, and processor $q + 1$, \ldots, processor $p$ will each house $r$ components of $x$. An example will make this clearer. Suppose,

$$x^T = [x_1, x_2, \ldots, x_9]$$

and there are 4 processors. That is, $n = 9$, $p = 4$, and $r = 2$. In the store-by-row scheme:

1. processor 1 will get $(x_1, x_2, x_3)$
2. processor 2 will get $(x_4, x_5)$
3. processor 3 will get $(x_6, x_7)$
4. and, finally, processor 4 will get $(x_8, x_9)$

In the store-by-column scheme, the vector $x$ is considered as a $r \times p$ matrix and each processor stores each column of that matrix. For $n = 9$, $p = 4$, and $r = 2$, the store-by-column scheme is:

1. processor 1 get $x_1, x_2, x_6$
2. processor 2 get $x_3, x_7$
3. processor 3 get $x_4, x_8$
4. and, finally, processor 4 house $x_5, x_9$
6.2.3 Efficiency and Speed-up

Golub and Van Loan ([21]) defined the efficiency of a p-processor parallel algorithm as:

\[ E = \frac{T(1)}{pT(p)} \]  

(6.2)

where \( T(k) \) is the time required to run the parallel program on \( k \) processors.

The speed-up \( S \) of a p-processor program over a uniprocessor for a particular problem is defined as:

\[ S = \frac{T_{\text{seq}}}{T_{\text{par}}} = \frac{T(1)}{T(p)} \]  

(6.3)

where \( T_{\text{par}} \) is the time required to run the parallel program on \( p \) processors and \( T_{\text{seq}} \) is the total time required for running the program sequentially on a uniprocessor.

These 2 constants will be used to quantify the performance of a parallel algorithm. There are other criteria for assessing performance. These include the cost of communication, the granularity of the program, etc. For more details on these criteria, see [1] and [21].

6.2.4 Implementation of A.I.S. on MIMD-DM

The parallel implementation of A.I.S. may be illustrated as follows. Suppose that we would like to evaluate the trace \( (W^{-1}) \), i.e.,

\[ tr(W^{-1}) = \sum_{i=1}^{n} \hat{\theta}_i \quad , i = 1, \ldots, n \]  

(6.4)

where

\[ \hat{\theta}_i = \sum_{k=1}^{N} \omega_k^* g_{ii}(u_k) \]  

(6.5)

and

\[ \omega_k^* = \frac{\omega_k}{\sum_{k=1}^{N} \omega_k} \]  

(6.6)
with
\[ \omega_k = \frac{f(u_k)}{h(u_k)} \]  \hspace{1cm} (6.7)
where, \( u_k \) is a \( n \times 1 \) vector generated from the importance distribution \( h \) and \( f \) is the unknown distribution with covariance matrix \( W^{-1} \).

In addition, suppose that we have 4 processors, called: \( P_1, P_2, P_3, \) and \( P_4. \) Let \( n = 397 \) be the number of rows in \( W \), i.e. there are 397 \( \theta \)'s to be estimated. This computation can be partitioned as follows:

1. Have processor \( P_2 \) compute \( \theta_{101}, ..., \theta_{199} \), and their sum, \( S_2 = \sum_{i=101}^{199} \theta_i \); and send \( S_2 \) to processor \( P_1 \).

2. Have processor \( P_3 \) compute \( \theta_{200}, ..., \theta_{298} \), and their sum, \( S_3 = \sum_{i=200}^{298} \theta_i \); and send \( S_3 \) to processor \( P_1 \).

3. Have processor \( P_4 \) compute \( \theta_{299}, ..., \theta_{397} \), and their sum, \( S_4 = \sum_{i=299}^{397} \theta_i \); and send \( S_4 \) to processor \( P_1 \).

4. Have processor \( P_1 \) compute \( \theta_1, ..., \theta_{100} \), their sum, \( S_1 = \sum_{i=1}^{100} \theta_i \); and the total sum, \( S = S_1 + \cdots + S_4 \), and print \( S \).

Furthermore, on each processor, the computation of each \( \theta \) can be decomposed into the following four steps:

1. Generate a suitable random vector \( u_k \) from the density \( h \). This will require each processor to have \( n \) random number generators.

2. Evaluate \( \omega_k \) and prepare the vector: \( \omega^* = (\omega_1^*, ..., \omega_N^*) \). The computation of each \( \omega_k \) is done by a call to the ratio function in the Fortran subroutine library.

3. Compute \( g_i(u_k) \) and prepare the vector \( g_i = (g_i(u_k), ..., g_i(u_k)) \). The evaluation of each \( g_i \) requires a call to the dot product and the square functions in the Fortran subroutines library.
4. Finally, compute the dot product $\omega^* g_i$.

The first step is straightforward, since the $u$'s are obtained from a known distribution such as the multivariate normal or the multivariate $t$. Step 2, 3, and 4 are also straightforward since they only require calls to elementary functions (sum, ratio, dot-product) in the Fortran subroutine library. In summary, we have demonstrated that A.I.S. is suitable for parallel processing.

For the tribolium data, the PVM code for parallel implementation of A.I.S. is given in appendix C. The result on efficiency and speed-up are given in the Table 6.1:

<table>
<thead>
<tr>
<th>Number of Processors</th>
<th>Time in seconds</th>
<th>E</th>
<th>S</th>
<th>Order</th>
<th>Non-zeros</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>68</td>
<td>.34</td>
<td>1.36</td>
<td>397</td>
<td>1801</td>
</tr>
<tr>
<td>4</td>
<td>50</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

6.3 The Shared Memory Environment

This section is concerned with the implementation of A.I.S. on a Shared Memory machine. The main feature of the machine is described in section 1, and the method of implementation of A.I.S. is presented in section 2.

6.3.1 Description of the Shared Memory Environment

The second computer environment available for parallel computing at I.S.U. is a four-processor, Shared Memory machine. It is a DEC AlphaServer 8400 with 12 Gigabytes of local disk and 1 Gigabyte of memory. In addition, it has 20K bytes from primary memory caches, one for data and one for instruction, and 96K bytes secondary memory caches for both data and instruction. A cache is a form of memory storage that is automatically filled and emptied according to a fixed scheme defined by the hardware system. The
DEC 8400 exceed 1 billion floating point operations per second. Information about this machine and its usage may be found at http://www.public.iastate.edu/farm/8400.html.

In this *Shared Memory* environment, each processor has its own local memory and its own local program. However, each has equal access to a common, shared memory, by reading and writing to *shared variables* that reside in the shared memory. Reading and writing have the form:

```
reading → load( { shared memory variable }, {local memory variable } )
```

```
writing → store( {local memory variable } , { shared memory variable } )
```

Access to the shared memory is done through the *synchronization* of the total number of tasks that needed to be shared by the p processors. In this machine, the synchronization tool is the *pool-of-task* approach. In the pool-of-task scheme, a list of *remaining* tasks is maintained. When a processor completes a task, it checks the list. If the list is nonempty, the processor selects a task from the pool of the remaining task. The selected task is then removed from the list. In the other hand, if the list is empty, then no work is assigned (see [21]). To this end, a *counter* that maintains the value of the last task selected is created. This counter is updated each time a given processor takes a new task. To ensure that *at most* one processor is engaged in the update of the counter at any one instant, a special procedure called *monitor* is used. The role of the monitor is to maintain the counter that steps from 1 to the number of tasks that are to be shared by the p processors. When no more task remain, the monitor's job is over. Golub and Van Loan ([21]) describe the monitor procedure as follows:

```
monitor: nexti

protected variables: flag, count, idle.proc
condition variables: waiti
initializations : flag = 0
procedure nexti.index(p, T, more, i)
```
if flag = 0
    count = 0; flag = 1; idle.proc = 0
end
count = count + 1
if count <= T
    more = true; i = count
else
    more = false; idle.proc = idle.proc + 1
    if idle.proc <= p - 1
        delay(waiti)
    else
        flag = 0
    end
    continue(waiti)
end
end nexti.index
end nexti

That is, the monitor will output the logical variable more and the index i such as:

more = true  $\implies$ task(i) is assigned
more = false $\implies$ no task remaining

In the next section, the use of a monitor is illustrated as well as reading from and writing to the shared memory.
6.3.2 Implementation of A.I.S. under the Shared Memory Environment

In A.I.S., the basic computation is the evaluation of dot product of the form:

\[ z = [w^T u]^2 \]

where, \( w \) and \( u \in R^n \) and \( z \in R \). That basic computation in the shared memory accessible to \( p \) processors is illustrated as follows:

\begin{verbatim}
shared variables [w, u z]
glob.init[ w, u]
loc.init [p, id = my.id , n]
store( u, u_{loc} )
call nexti.index (p, T, more, i)
while  more = true
  store( w, w_{loc} )
  z_{loc} = w_{loc}^T u_{loc}
  load( z_{loc} , z)
  call nexti.index}(p, T, more, i)
end
if  id = 1
  load(z, z_{loc})
  z_{loc} = z_{loc}^2
  store(z_{loc} , z)
quit.
\end{verbatim}

The actual code for the implementation of A.I.S. under the Shared Memory environment can be found in appendix C. Applied to the 397 x 397 matrix of the tribolium data, the AIS algorithm yields the following results:
### Table 6.2 Efficiency and Speed-Up of the Parallel Implementation of A.I.S. in a Distributed Memory Environment

<table>
<thead>
<tr>
<th>Number of Processors</th>
<th>Time in seconds</th>
<th>E</th>
<th>S</th>
<th>Order</th>
<th>Non-zeros</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>67</td>
<td></td>
<td></td>
<td>397</td>
<td>1801</td>
</tr>
<tr>
<td>2</td>
<td>56</td>
<td>.60</td>
<td>1.20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>51</td>
<td>.44</td>
<td>1.31</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>48</td>
<td>.35</td>
<td>1.40</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The results of this table show that speed-up is achieved by parallelization of A.I.S., and that speed-up is increasing with the number of processors. These results are comparable to the one obtained with the distributed memory environment.
7 CONCLUSION AND DISCUSSION

This dissertation has explored some aspects of the inversion of well-conditioned, large, sparse, and p.d. matrices. For this class of matrices, the following three results have been obtained. Firstly, along with [27], [21], [18], and others, we have addressed the many drawbacks of direct methods and iterative techniques for inverting large, sparse, and p.d. matrix. These drawbacks include, \textit{fill-ins, good ordering}, and \textit{choice of critical parameters} problems.

Secondly, it has been shown that \textit{desired} elements of the inverse of a large, sparse, positive definite matrix can be evaluated with desired accuracy via a \textit{statistical approach}. In this approach, each element is decomposed as the sum of two components: a fixed quantity and an expectation of a well defined random variable. This approach work directly with the original matrix $\mathbf{W}$. Thus, it is \textit{devoid} of the good ordering, fill-ins and choice of critical parameter problems. This approach will always yield positive estimates for variances. In addition, this approach has the following advantages:

1. It is \textit{flexible}, that is, a \textit{desired entry} of the inverse matrix can be evaluated, \textit{without} computing any other entry.

2. It \textit{takes advantage} of the sparsity of the matrix by using an efficient sparse storage scheme

3. It computes the \textit{exact value} for some entries.

4. It is \textit{easily parallelizable}, which provides gains in efficiency and computing time.
Some traditional methods, such as the single division method, are not parallelizable. And some methods, such as Cholesky, are only blocked parallelizable.

The expectation in the above decomposition may be computed using either the ordinary Importance Sampling technique or the Adaptive Importance Sampling.

For moderate dimension of the matrix the ordinary importance sampling yields reasonable results when the importance sampler is the \( \mathbf{MVT}_3 \) and the corresponding covariance is equal to \( \text{Diag}(1/w_{ii} + 1/w_{ii}^2) \).

The A.I.S. may be started with three different covariance matrices: the Identity matrix, the \( \text{Diag}(1/w_{ii}) \), and the \( \text{Diag}(1/w_{ii} + 1/w_{ii}^2) \). A.I.S. provides 'better' results than the ordinary importance sampling and requires fewer iterations when started with the third matrix.

For extremely large dimension, A.I.S. may be speeded up by combining the computing power of 2 or more processors via parallel processing. The method may also be speeded up by using an efficient sparse storage scheme.

This research is in its beginning and as such it needs an influx of ideas, suggestions, and criticisms for its growth. In this context, our first step is to find a large sparse, p.d. matrix for which the Cholesky method is not successful but the A.I.S. approach presented here is. Secondly, we will present the parallel implementation of the A.I.S. approach for the Cray-T3D and the Cray-T3E using level 2 and level 3 BLAS routines. The goal here is to show that this can be done with Massive Parallel Processing Fortran and without explicit message passing. This enables us to compare the A.I.S. approach with similar work done on blocked Cholesky at Pittsburgh using Massive Parallel Interface on the same machines. Thirdly, we will compare the use of this approach to evaluate a determinant to other existing methods.

This work may be rich in research topics. For example, there are two open problems. The first is to see how the condition number of the matrix \( \mathbf{W} \) affects the performance of
the A.I.S. approach in evaluating $W^{-1}$. More precisely, one desires to see if this approach would perform better than existing methods on ill-conditioned matrices. The second is to find an *optimal search algorithm* for locating a nonzero entry of the sparse matrix. An efficient location of the nonzero element of the sparse matrix becomes a *search problem*. Therefore, finding an optimal search algorithm will lead to an *optimal storage scheme* which in turn will be very efficient in computing the matrix-vector product and other basic linear algebras of interest. The third problem is finding a *stopping criterion* that will yield accurate results with a minimum number of iterations. The real issue is to choose between a *unique* criterion that needs to be satisfied by all the entries computed or an *individual* criterion for each entry, especially when the computation is done via parallel processing. We conjecture that the latter case is the most efficient in term of accuracy and may be running time, but this remains to be proved. These two problems are future research topics.
APPENDIX A FORTRAN CODE FOR THE IMPORTANCE SAMPLING: SPARSE FORMAT

***************************************************************************
*  Adaptive Importance Sampling: Sparse Format******************************
***************************************************************************

integer i, ifail, ii, istep, nu, n, splsize, sumsize
integer i1, i2, i5, i6, j, j1, j2, nzz, nz
integer s, nr, ic, count, iseed, k, l
parameter (n = 397, nr = (n+1)*(n+2) / 2, ic =n)
parameter (nzz = 7057)

*   Local Scalars
real*8 xi2, nnu, wgt, giu, eps, seps
real*8 psumw, psumw2, tsumw, tsumw2
real*8 wbar, varw, wiotu
real*8 wiktu, wijktu, wii, wjj, wij, wjk, x
real*8 epsilon, zalpha2
real*8 g05dhf

*   Local Arrays
integer wc(nzz), istart(n+1)
real*8 avg(n), r(nr), z(n), u(n)
real*8 sig(n,n), wnz(nzz)
real*8 gw(n, n), gw2(n,n), g2w2(n,n), sg(n,n)
real*8 sumgw(n,n), sumgw2(n,n), sumg2w2(n,n)
real*8 eg(n, n), sev(n, n), v(n, n), vargw(n, n)
real*8 covgw(n, n), sig2hat(n, n), sighat(n, n)
real*8 u2w(n), sumu2w(n)
real*8 s2(n, n), s3(n, n), d(n,n)
**********Executable Statements**********
*
**********Reading the Basic Constants**********
*
read(*,*) iseed
call g05cbf(iseed)
*
read(*,*) splsize
write(*,*) 'splsize = ', splsize
write(*,99) splsize
write(*,*)
*
read(*,*) nz
write(*,*) 'number of nonzero elements of W = ', nz
write(*,99) nz
write(*,*)
*
read(*,*) istep
*
read(*,*) nnu
*
read(*,*) nu
*
read(*,*) epsilon
write(*,*) 'epsilon = ', epsilon
write(*,98) epsilon
write(*,*)
*
read(*,*) zalpha2
write(*,*) 'zalpha2 = ', zalpha2
write(*,98) zalpha2
write(*,*)
*
read(*,*) seps
write(*,*) 'seps = ', seps
write(*,98) seps
write(*,*)
*
read(*,*)
ii = 0
l = 1

***************Reading and Forming wnz, wc, and istart***************

do 900 k = 1, nz
    read(*, *) i, j, x
    wnz(k) = x
    wc(k) = j
    if(ii .ne. i) then
        istart(l) = k
        ii = i
        l = l + 1
    endif
900 continue

istart(n+1) = istart(n) + istep

write(*,*) 'istart(n) = ', istart(n)
write(*, 999) istart(n)
write(*,*)

write(*,*) 'istart(n+1) = ', istart(n+1)
write(*, 999) istart(n+1)
write(*,*)

write(*,*) 'the nonzeros w = ', wnz(i), i = 1, nz)
write(*,*)

write(*,*) ' wc = ', wc(i), i = 1, nz)
write(*,*)

write(*,*) 'istart = ', istart(i), i = 1, n+1)
write(*,*)

***************Generating Sigma***************
do 10 i = 1, n
   do 20 ii = istart(i), istart(i+1)-1
      if(wc(ii) .eq. i) then
         sig(i,i)= 1.0d0/wnz(ii) + 1.0d0/(wnz(ii)**2)
         go to 21
      endif
   20 continue
   21 continue
*     avg(i) = 0.0d0
10 continue
*
* write(*, *) 'sig = diagonal'
write(*,*)
write(*,97) (sig(i,i), i = 1,n)
write(*,*)
***********Initialization************************************************************
*      ifail = 0
      count = 0
      eps = seps / n
*      sumsize = 0
      tsumw = 0.0d0
      tsumw2 = 0.0d0
*      do 25 i = 1, n
         do 27 j = 1, n
            sumgw(i, j) = 0.0d0
            sumgw2(i, j) = 0.0d0
            sumg2w2(i, j) = 0.0d0
         27 continue
*         sumu2w(i) = 0.0d0
25 continue
*      sumu2w(i) = 0.0d0
510 continue
      psumw = 0.0d0
      psumw2 = 0.0d0
*      do 30 i = 1, n
do 40 j = 1, n
   gw(i,j) = 0.0d0
   gw2(i,j) = 0.0d0
   g2w2(i,j) = 0.0d0
40 continue
*
   u2w(i) = 0.0d0
*
30 continue
*
******Generating a random vector u from the Mvt(0, sigma)***************
*
   call g05eaf(avg, n, sig, ic, eps, r, nr, ifail)
   do 100 s = 1, splsize
      xi2 = g05dhf(nu, ifail)
      call g05ezf(z, n, r, nr, ifail)
   100 continue
*
*
   do 150 j = 1, n
      u(j) = z(j)*dsqrt(nnu) / dsqrt(xi2)
150 continue
*
*
***********Computing the weight of each u **********************
*
   call weight(u, n, nz, wc, wnz, sig, istart, wgt)
*
*
***********Summing the weight and the square of the weight of each u********
*
   psumw = psumw + wgt
   psumw2 = psumw2 + wgt**2
*
***********Computing g for each i and each u ***********************
*
   do 50 i = 1, n
      wiotu = 0.0d0
      do 60 i2 = istart(i), istart(i+1)-1
         if(wc(i2) .ne. i)
            &   wiotu = wiotu + wnz(i2)*u(wc(i2))
60 continue
*
giu = wiotu**2
gw(i,i)  = gw(i,i) + giu*wgt
gw2(i,i) = gw2(i, i) + giu*(wgt**2)
g2w2(i, i) = g2w2(i, i) + (giu**2)*(wgt**2)

*  
50  continue
*  
*  
************Computing g for each i and each j, gij of u***********
*  
*  
  do 70 i = 1, n
  do 80 j = i+1, n
       wiktu = 0.0d0
       wjkhu = 0.0d0
       do 82 k = istart(j), istart(j+1)-1
           if(wc(k) .ne. i .and. wc(k) .ne. j)
               wjkhu = wjkhu + wnz(k)* u(wc(k))
       82 continue
*  
  *  
       do 84 l = istart(i), istart(i+1)-1
           if(wc(l) .ne. i .and. wc(l) .ne. j)
               wiktu = wiktu + wnz(l)* u(wc(l))
       84 continue
*  
  *  
       do 86 j1 = istart(j), istart(j+1)-1
           if(wc(j1) .eq. j) then
               wjj = wnz(j1)
               go to 87
           endif
       86 continue
87 continue
*  
  do 88 i2 = istart(i), istart(i+1)-1
     if(wc(i2) .eq. i) then
         wii = wnz(i2)
         go to 89
    endif
  88 continue
89 continue
*  

* 
do 90 i = istart(i), istart(i+1)-1
  if(wc(1) .eq. j) then
    wij = wnz(1)
  *
    d(i, j) = wii*wjj - wij**2
    d(j, i) = d(i, j)
  *
    sg(i, j) = (wjj*wiktu - wij*wjktu)*
          & (wii*wjktu - wij*wiktu)
    sg(j, i) = sg(i, j)
  *
  go to 91
  *
  endif
  90 continue
  *
  *
  sg(i, j) = wjj*wiktu*wii*wjktu
  sg(j, i) = sg(i, j)
  *
    d(i, j) = wii*wjj
    d(j, i) = d(i, j)
  *
  *
  91 continue
  *
  *
    gw(i, j) = gw(i, j) + sg(i, j)*wgt
    gw2(i, j) = gw2(i, j) + sg(i, j)*(wgt**2)
    g2w2(i, j) = g2w2(i, j) + (sg(i, j)**2)*(wgt**2)
    gw(j, i) = gw(i, j)
    gw2(j, i) = gw2(i, j)
    g2w2(j, i) = g2w2(i, j)
  *
  80 continue
  70 continue
**
*
**
*
100 continue
**Total sum of gi and gisquare**

```fortran
100
  do 300 i = 1, n
    do 350 j = i+1, n
      sumgw(i, j) = sumgw(i, j) + gw(i, j)
      sumgw2(i, j) = sumgw2(i, j) + gw2(i, j)
      sumg2w2(i, j) = sumg2w2(i, j) + g2w2(i, j)
    350 continue
  end do
  sumgw(i,i) = sumgw(i,i) + gw(i,i)
  sumu2w(i) = sumu2w(i) + u2w(i)
300 continue
```

**Total sum of w and splsize**

```fortran
100
  tsumw = tsumw + psumw
  tsumw2 = tsumw2 + psumw2
  sumsize = sumsize + splsize
  write(*, *) 'sumsize = ', sumsize
```

**Computing wbar and varwbar**

```fortran
100
  wbar = tsumw / sumsize
  varw = tsumw2 / sumsize - wbar**2
```

**Stopping Rule and Variance of each estimate**

```fortran
100
  if(varw/(sumsize*(wbar**2)) .gt. (epsilon/zalpha2)**2) then
    do 500 i = 1, n
      eg(i,i) = sumgw(i, i)/ tsumw
    do 550 i5 = istart(i), istart(i+1)-1
      if(wc(i5) .eq. i) then
        wii = wnz(i5)
        sig(i, i) = 1.0d0/wii + eg(i,i)/(wii**2)
      550 continue
    500 continue
  end if
```
*  
end if
550  continue
*  
551  continue
*  
500  continue
  count = count + 1
*  
write(*,*) 'sig = '
write(*,97) (sig(i,i), i=1,n)
write(*,*)
go to 510
*  
else
  do 600 j = 1, n
    do 650 k = j+1, n
      eg(j, k) = sumgw(j, k) / tsumw
      vargw(j,k) = sumg2w2(j,k) / sumsize -
      & (eg(j,k)**2)*(wbar**2)
      & covgw(j,k) = sumgw2(j,k) /sumsize -
      & eg(j,k)*(wbar**2)
      s2(j,k) = 2*eg(j,k)*covgw(j,k)
      s3(j,k) = eg(j,k)*eg(j,k)*varw
      & sig2hat(j,k)= (vargw(j,k) - s2(j,k) +
      & s3(j,k))/(wbar**2)
      & sighat(j,k) = dsqrt(sig2hat(j,k)/ sumsize)
      sev(j,k) = sighat(j,k) / (d(j,k)**2)
      sev(k,j) = sev(j,k)
*  
do 655 j2 = istart(j), istart(j+1)-1
  if(wc(j2) .eq. k ) then
    wjk = wnz(j2)
    v(j,k) = -wjk/d(j,k) + eg(j,k)/ (d(j,k)**2)
    v(k,j) = v(j,k)
*  
go to 656
*  
endif
*  
655  continue
**  
v(j,k) = eg(j,k)/ (d(j,k)**2)
  v(k,j) = v(j,k)
* continue
* continue

eg(j, j) = sumg(j, j) / tsumw
vargw(j, j) = sumg2w2(j, j) / sumsize -
&
covgw(j, j) = sumg2w2(j, j) / sumsize - eg(j, j)*(wbar**2)
s2(j, j) = 2*eg(j, j)*covgw(j, j)
s3(j, j) = eg(j, j)*eg(j, j)*varw
sig2hat(j, j) = (vargw(j, j) - s2(j, j) + s3(j, j))/(wbar**2)
sighat(j, j) = dsqrt(sig2hat(j, j)/ sumsize)

* do 660 i6 = istart(j), istart(j+1)-1
  if(wc(i6) .eq. j) then
    wjj = wnz(i6)
    v(j, j) = 1.0d0/wjj + eg(j, j)/(wjj**2)
    sev(j, j) = sighat(j, j)/(wjj**2)
  * go to 661
* endif
* continue
* continue
* continue
* endif

*******************************************Printing*******************************************
write(*,*) 'Number of iterations ='
write(*,99) count + 1
write(*,*)
* do 1 i = 1, n
  do 2 j = 1, n
    if(v(i, j) .ne. 0.0d0) then
      write(*, 997) v(i, j)
      write(*, 98) sev(i, j)
    write(*,*)
endif

* 2 continue
1 continue
* *

***************Formatting*********************************************
*
97 format(ix, 10f10.4)
98 format(ix, f10.4)
99 format(ix, i10)
999 format(ix, 10i5)
997 format(ix, f10.4)
*

stop
end
*

******************Subroutines***************************************
*
Subroutine weight(u, n, nz, wc, wnz, sig, istart, wgt)
*
integer n, nz, i4, i7
integer ii
real*8 utwmsu
real*8 wgt
integer istart(n+1), wc(nz)
real*8 u(n), wnz(nz), sig(n,n), wms(nz)
*
real*8 dexp
*
*
utwmsu = 0.0d0
*
do 800 ii = 1, n
  do 810 i4 = istart(ii), istart(ii+1)-1
    if(wc(i4) .eq. ii) then
      wms(i4) = wnz(i4) - 1.0d0/sig(ii, ii)
    else
      wms(i4) = wnz(i4)
    endif
  810 continue
800 continue
*  
*  
do 820 ii = 1, n  
do 830 i7 = istart(ii), istart(ii+1)-1  
     utwmsu = utwmsu + wms(i7)*u(wc(i7))*u(ii)  
 830     continue  
 820     continue  
*  
wgt = dexp(-.5d0 * utwmsu)  
*  
*  
return  
end
APPENDIX B FORTRAN CODE FOR THE ADAPTIVE IMPORTANCE SAMPLING

MULTIVARIATE $t_3$ CASE

*******************************************************************
* Adaptive Importance Sampling: Sparse Format**********************
* ***
*******************************************************************
integer i, ifail, ii, istep, nu, n, splsize, sumsize
integer is, i1, i2, i5, i6, j, j1, j2, nzz, nz
integer s, nr, ic, count, iseed, k, l
parameter (n = 5, nr = (n+1)*(n+2) / 2, ic = n, is = n+1)
parameter (nzz = 13)

** Local Scalars **
real*8 xi2, nnu, wgt, giu, eps, seps
real*8 psumw, psumw2, tsumw, tsumw2
real*8 wbar, varw, wiotu
real*8 wiktu, wjk, wjktu, wi, wj, wij, wjk, x
real*8 epsilon, zalpha2
real*8 g05dhf

** Local Arrays **
integer wc(nzz), istart(is)
real*8 avg(n), r(nr), z(n), u(n)
real*8 sig(n,n), wnz(nzz)
real*8 gw(n, n), gw2(n,n), g2w2(n,n), sg(n,n)
real*8 sumgw(n,n), sumgw2(n,n), sumg2w2(n,n)
real*8 eg(n, n), sev(n, n), v(n, n), vargw(n, n)
real*8 covgw(n, n), sig2hat(n, n), sighat(n, n)
real*8 u2w(n), sumu2w(n)
real*8 s2(n, n), s3(n, n), d(n,n)
**Executable Statements**

*Reading the Basic Constants*

read(*,*) iseed
   call g05cbf(iseed)

read(*,*) splsize
   write(*,*) 'splsize = ' splsize
   write(*,99) splsize

read(*,*) nz
   write(*,*) 'number of nonzero elements of W = ' nz
   write(*,99) nz

read(*,*) istep

read(*,*) nnu

read(*,*) nu

read(*,*) epsilon
   write(*,*) 'epsilon = ' epsilon
   write(*,98) epsilon

read(*,*) zalpha2
   write(*,*) 'zalpha2 = ' zalpha2
   write(*,98) zalpha2

read(*,*) seps
   write(*,*) 'seps = ' seps
   write(*,98) seps

read(*,*)

ii = 0
1 = 1

***************Reading and Forming wnz, wc, and istart***************

   do 900 k = 1, nz
      read(*, *) i, j, x
      wnz(k) = x
      wc(k) = j
      if(ii .ne. i)then
         istart(1) = k
         ii = i
         l = l+ 1
      endif
   900 continue

   istart(n+1) = istart(n) + istep

   write(*,*) 'istart(n) = ',
   write(*, 999) istart(n)
   write(*,*)

   write(*,*) 'istart(n+1) = ',
   write(*, 999) istart(n+1)
   write(*,*)

   write(*,*) 'the nonzeros w = ',
   write(*,*)
   write(*,97) (wnz(i), i = 1, nz)
   write(*,*)

   write(*,*) ' wc = ',
   write(*,*)
   write(*,999) (wc(i), i = 1, nz)
   write(*,*)

   write(*,*) ' istart = ',
   write(*,*)
   write(*,999) (istart(i), i = 1, n+1)
   write(*,*)

***************Generating Sigma**********************
do 10 i = 1, n
  do 20 il = istart(i), istart(i+l)-l
    if(wc(il) .eq. i) then
      sig(i,i)= 1.0d0/wnz(il) + 1.0d0/(wnz(il)**2)
      go to 21
    endif
  20 continue
  21 continue
*
  avg(i) = 0.0d0
10 continue
*
*
write(*, *) 'sig = diagonal'
write(*,*)
write(*,97) (sig(i,i), i = 1,n)
write(*,*)

**********Initialization*********************************
*
ifail = 0
count = 0
eps = seps / n
*
sumsize = 0
tsumw  = 0.0d0
tsumw2 = 0.0d0
*
  do 25 i = 1, n
    do 27 j = 1, n
      sumgw(i, j)  = 0.0d0
      sumgw2(i, j) = 0.0d0
      sumg2w2(i, j) = 0.0d0
    27 continue
  25 continue
*
  sumu2w(i) = 0.0d0
25 continue
*
*
510 continue
psumw = 0.0d0
psumw2 = 0.0d0
*
  do 30 i = 1, n
    do 40 j = 1, n
gw(i,j) = 0.0d0
gw2(i,j) = 0.0d0
g2w2(i,j) = 0.0d0

continue

* u2w(i) = 0.0d0

* continue

*****Generating a random vector u from the Mvt(0, sigma)***********
*
*
call g05eaf(avg, n, sig, ic, eps, r, nr, ifail)
do 100 s = 1, splsize
  xi2 = g05dhf(nu, ifail)
call g05ezf(z, n, r, nr, ifail)
*
*
do 150 j = 1, n
  u(j) = z(j)*dsqrt(nnu) / dsqrt(xi2)
150 continue
*
*
***********Computing the weight of each u **************************
*

call weight(u,nnu, n, nz, wc, wnz, sig, istart, is, wgt)
*
*
*******Summing the weight and the square of the weight of each u***
*
  psumw = psumw + wgt
  psumw2 = psumw2 + wgt**2
*

***********Computing g for each i and each u ***********************
*
  do 50 i = 1, n
    wiotu = 0.0d0
    do 60 i2 = istart(i), istart(i+1)-1
      if(wc(i2) .ne. i)
        & wiotu = wiotu + wnz(i2)*u(wc(i2))
    60 continue
*
  giu = wiotu**2
\( \text{gw}(i, i) = \text{gw}(i, i) + g_{iu}w_{gt} \)
\( \text{gw}2(i, i) = \text{gw}2(i, i) + g_{iu}(w_{gt}^2) \)
\( g_{2w2}(i, i) = g_{2w2}(i, i) + (g_{iu}^2)(w_{gt}^2) \)

* 50 continue
*
*
**********Computing \( g \) for each \( i \) and each \( j \), \( g_{ij} \) of \( u \)**************
*
*
* do 70 i = 1, n
* do 80 j = i+1, n
* wiktu = 0.0d0
* wjktn = 0.0d0
* do 82 k = istart(j), istart(j+1)-1
  * if(wc(k) .ne. i .and. wc(k) .ne. j)
  * wjktn = wjktn + wnz(k) * \( u(wc(k)) \)
* continue
*
*
* do 84 l = istart(i), istart(i+1)-1
  * if(wc(l) .ne. i .and. wc(l) .ne. j)
  * wiktu = wiktu + wnz(l) * \( u(wc(l)) \)
* continue
*
*
* do 86 j1 = istart(j), istart(j+1)-1
  * if(wc(j1) .eq. j) then
  * wjj = wnz(j1)
  * go to 87
* endif
* 86 continue
* 87 continue
* 
* do 88 i2 = istart(i), istart(i+1)-1
  * if(wc(i2) .eq. i) then
  * wii = wnz(i2)
  * go to 89
* endif
* 88 continue
* 89 continue
* 
*
do 90 l = istart(i), istart(i+1)-1
   if(wc(l) .eq. j)then
     wij = wnz(l)
     *
     d(i, j) = wii*wjj - wij**2
     d(j, i) = d(i, j)
     *
     sg(i,j) = (wjj*wikt - wij*wjkt)*
     & (wii*wjkt - wij*wikt)
     sg(j,i) = sg(i,j)
     *
     go to 91
     *
   endif
90    continue
   *
   *
   sg(i,j) = wjj*wikt*wii*wjkt
   sg(j,i) = sg(i,j)
   *
   d(i, j) = wii*wjj
   d(j, i) = d(i, j)
   *
91    continue
   *
   *
   gw(i, j) = gw(i,j) + sg(i,j)*wgt
   gw2(i, j) = gw2(i,j) + sg(i,j)*(wgt**2)
   g2w2(i, j) = g2w2(i, j)+(sg(i,j)**2)*(wgt**2)
   gw(j,i) = gw(i,j)
   gw2(j,i) = gw2(i,j)
   g2w2(j,i) = g2w2(i,j)
   *
80    continue
70    continue
**
*  
**
* 100  continue
*
**********Total sum of gi and gisquare******************************
*
    do 300 i = 1, n
        do 350 j = i+1,n
            sumgw(i, j) = sumgw(i, j) + gw(i, j)
            sumgw2(i, j) = sumgw2(i, j) + gw2(i, j)
            sumg2w2(i, j) = sumg2w2(i, j) + g2w2(i, j)
        350 continue
*
        sumgw(i,i) = sumgw(i,i) + gw(i,i)
        sumu2w(i) = sumu2w(i) + u2w(i)
    300 continue
*
**********ToTal Sum of w and splsize******************************
*
    tsumw = tsumw + psumw
    tsumw2 = tsumw2 + psumw2

    sumsize = sumsize + splsize

    write(9, 99) sumsize
    write(*,*)
*
**********Computing wbar eind
*
    wbar = tsumw / sumsize
    varw = tsumw2 / sumsize - wbar**2
*
*
**********Stopping Rule and Variance of each estimate**************
*
    if( varw/(sumsize*(wbar**2)).gt.(epsilon/zalpha2)**2) then
        do 500 i = 1, n
            eg(i,i) = sumgw(i, i)/ tsumw
        do 550 i5 = istart(i), istart(i+1)-1
            if( wc(i5) .eq. i) then
                wii = wnz(i5)
                sig(i, i) = 1.0d0/wii + eg(i,i)/(wii**2)
        550 continue
    500 continue
*
    go to 551
* 
  endif
  continue
* 
  continue
*
  continue 
  count = count + 1
*
  write(*,*) 'sig = '
  write(*,97) (sig(i,i), i=1,n)
  write(*,*)
  go to 510
*
  else
    do 600 j = 1, n
      do 650 k = j+1, n
        eg(j, k) = sumgw(j, k) / tsumw
        vargw(j,k) = sumg2w2(j,k) / sumsize -
        & (eg(j,k)**2)*(wbar**2)
        & covgw(j,k) = sumg2w2(j,k) /sumsize -
        & eg(j,k)*wbar**2
        s2(j,k) = 2*eg(j,k)*covgw(j,k)
        s3(j,k) = eg(j,k)*eg(j,k)*varw
        sig2hat(j,k) = (vargw(j,k) - s2(j,k) +
        & s3(j,k))/(wbar**2)
        sighat(j,k) = dsqrt(sig2hat(j,k)/ sumsize)
        sev(j,k) = sighat(j,k) / (d(j,k)**2)
        sev(k,j) = sev(j,k)
*  do 655 j2 = istart(j), istart(j+1)-1
    if(wc(j2) .eq. k ) then
      wjk = wnz(j2)
      v(j,k) = -wjk/d(j,k) + eg(j,k)/ (d(j,k)**2)
      v(k,j) = v(j,k)
  *  go to 656
*  
  endif
*
  continue
**
  v(j,k) = eg(j,k)/ (d(j,k)**2)
  v(k,j) = v(j,k)
eg(j, j) = sumgw(j, j) / tsumw
vargw(j,j) = sumg2w2(j,j) / sumsize -
    (eg(j,j)**2)*(wbar**2)
&
covgw(j,j) = sumg2w2(j,j) / sumsize - eg(j,j)*(wbar**2)
s2(j,j)   = 2*eg(j,j)*covgw(j,j)
s3(j,j)   = eg(j,j)*eg(j,j)*varw
sig2hat(j,j)= (vargw(j,j)-s2(j,j)+ s3(j,j))/(wbar**2)
sighat(j,j) = dsqrt(sig2hat(j,j)/ sumsize)

do 660 i6 = istart(j), istart(j+1)-1
  if(wc(i6) .eq. j) then
    wjj = wnz(i6)
    v(j,j) = 1.0d0/wjj + eg(j,j)/(wjj**2)
    sev(j,j) = sighat(j,j)/(wjj**2)
  go to 661
end if

end do 660

endif

continue

continue

continue

continue

continue

continue

end if

end do 660

write(*,*) 'Number of iterations = ',
write(*,99) count + 1
write(*,*)

write(*,997) v(i,j)
write(*, 98) sev(i,j)
write(*,*)
endif

2     continue
1     continue
*
*
**************************Formatting******************************
*
97    format(1x, 10f10.4)
98    format(1x, f10.4)
99    format(1x, i10)
999   format(1x, 10i5)
997   format(1x, f10.4)
*
    stop
    end
*
*
**************************Subroutines******************************

Subroutine weight(u, nnu, n, nz, wc, wnz, sig, istart, is, wgt)
*
integer        n, nz, is, ii
real*8         utwu, utsinvu
real*8         s2, s3
real*8         nnu
real*8         wgt, wij
integer        istart(is), wc(nz)
real*8         u(n), wnz(nz), sig(n,n)
c
real*8         dexp
*
*
  utwu  = 0.0d0
  utsinvu = 0.0d0
*
*
  do 700 ii = 1, n
        do 710 i4 = istart(ii), istart(ii+1)-1
            wij = wnz(i4)
            utwu = utwu + wij*u(ii)*u(wc(i4))
            if(wc(i4) .eq. ii)  utsinvu = utsinvu
            & + 1.0d0/sig(ii,ii)*u(ii)*u(wc(i4))
        continue
700  continue

* if(utwu .gt. 500.0d0) then
   s2 = 0.0d0
else
   s2 = dexp(-utwu *0.5d0)
endif

* s3 = (1.0d0 + utsinvu / nnu)**((nnu + n) *0.5d0)
wgt= s2*s3

* return
end

********************************************************************
MULTIVARIATE $t_3$ CASE

********************************************************************
* Adaptive Importance Sampling: Sparse Format

integer i, ifail, icount, istep, n, spsize, sumsize
integer i1, i2, i5, i6, j, j1, j2, nzz, nz
integer s, nr, ic, count, iseed, k, l
parameter (n = 397, nr = (n+1)*(n+2) / 2, ic =n)
parameter (nzz = 1900)

* Local Scalars
real*8  wgt, giu, eps, seps
real*8  psumw, psumw2, tsumw, tsumw2
real*8  wbar, varw, wioutu
real*8  wiktu, wjktu, wii, wjj, wij, wjk, x

  real*8  epsilon, zalpha2


  real*8  epsilon, zalpha2

* Local Arrays

integer wc(nzz), istart(n+1)
real*8  avg(n), r(nr), u(n)
real*8  sig(n,n), wnz(nzz)
real*8  gw(n, n), gw2(n,n), g2w2(n,n), sg(n,n)
real*8  sumgw(n,n), sumgw2(n,n), sumg2w2(n,n)
real*8  eg(n, n), sev(n, n), v(n, n), vargw(n, n)
real*8  covgw(n, n), sig2hat(n, n), sighat(n, n)
real*8  u2w(n), sumu2w(n)
real*8  s2(n, n), s3(n, n), d(n,n)

***************Executable Statements***********************
*
***************Reading the Basic Constants***********************
*
read(*,*) iseed
call g05cbf(iseed)
*
read(*,*) splsize
write(*,*) 'splsize = ', splsize
write(*,*)
*
read(*,*) nz
write(*,*) 'number of nonzero elements of W = ', nz
write(*,*)
*
read(*,*) istep
*
*
read(*,*) epsilon
write(*,*) 'epsilon = ', epsilon
write(*,88) epsilon
write(*,*)
*
read(*,*) zalpha2
write(*,*) 'zalpha2 = ', zalpha2
write(*,98) zalpha2
write(*,*)
*
read(*,*) seps
write(*,*) 'seps = ', seps
write(*,98) seps
write(*,*)

***************Reading the Basic Constants***********************
*
read(*,*) iseed
call g05cbf(iseed)
*
read(*,*) splsize
write(*,*) 'splsize = ', splsize
write(*,*)
*
read(*,*) nz
write(*,*) 'number of nonzero elements of W = ', nz
write(*,*)
*
read(*,*) istep
*
*
read(*,*) epsilon
write(*,*) 'epsilon = ', epsilon
write(*,88) epsilon
write(*,*)
*
read(*,*) zalpha2
write(*,*) 'zalpha2 = ', zalpha2
write(*,98) zalpha2
write(*,*)
*
read(*,*) seps
write(*,*) 'seps = ', seps
write(*,98) seps
write(*,*)

***************Reading the Basic Constants***********************
*
*  
read(*,*)  
*  
icount = 0  
l = 1  
**************************************************  
Reading and Forming wnz, wc, and istart***************  
*  
do 900 k = 1, nz  
read(*,*) i, j, x  
wnz(k) = x  
w(c(k)) = j  
if(icount .ne. i)then  
istart(l) = k  
icount = i  
l = l+ 1  
endif  
900 continue  
*  
istart(n+1) = istart(n) + istep  
**************************************************  
Generating Sigma***********************  
*  
do 10 i = 1, n  
do 20 i1 = istart(i), istart(i+1)-1  
if(wc(i1) .eq. i) then  
sig(i,i) = 1.0d0/wnz(i1) + 1.0d0/(wnz(i1)**2)  
go to 21  
endif  
20 continue  
21 continue  
*  
avg(i) = 0.0d0  
10 continue  
*  
*  
write(*, *) 'sig = diagonal '  
write(*,*)  
write(*,97) (sig(i,i), i = 1,n)  
write(*,*)  
**********Initialization******************************************  
*  
ifail = 0  
count = 0
eps = seps / n
*
sumsize = 0
tsumw = 0.0d0
tsumw2 = 0.0d0
*
do 25 i = 1, n
    do 27 j = 1, n
        sumgw(i, j) = 0.0d0
        sumgw2(i, j) = 0.0d0
        sumg2w2(i, j) = 0.0d0
    27 continue
*
510 continue
psumw = 0.0d0
psumw2 = 0.0d0
*
do 30 i = 1, n
    do 40 j = 1, n
        gw(i, j) = 0.0d0
        gw2(i, j) = 0.0d0
        g2w2(i, j) = 0.0d0
    40 continue
*
u2w(i) = 0.0d0
*
30 continue
*
*****Generating a random vector u from the Mvt(0, sigma)***********
*
call g05eaf(avg, n, sig, ic, .00001d0, r, nr, ifail)
do 100 s = 1, splsize
    call g05ezf(u, n, r, nr, ifail)
*
*
*************Computing the weight of each u **********************
*
call weight(u, n, nz, wc, wnz, sig, istart, wgt)
*
**********Summing the weight and the square of the weight of each u***
*  
\[ psumw = psumw + wgt \]
\[ psumw^2 = psumw^2 + wgt^2 \]
*  
***********Computing \( g \) for each \( i \) and each \( u \) ***********
*  
\[ \text{do 50 i = 1, n} \]
\[ \text{wiotu} = 0.0d0 \]
\[ \text{do 60 i2 = istart(i), istart(i+1)-1} \]
\[ \quad \text{if(wc(i2) .ne. i) \&} \]
\[ \quad \quad \text{wiotu} = \text{wiotu} + wnz(i2)*u(wc(i2)) \]
\[ \text{60 continue} \]
*  
\[ \text{giu} = \text{wiotu}^2 \]
\[ \text{gw}(i,i) = \text{gw}(i,i) + \text{giu}*wgt \]
\[ \text{gw2}(i,i) = \text{gw2}(i,i) + \text{giu}*(\text{wgt}^2) \]
\[ \text{g2w2}(i,i) = \text{g2w2}(i,i) + (\text{giu}^2)*(\text{wgt}^2) \]
*  
\[ \text{50 continue} \]
*  
***********Computing \( g \) for each \( i \) and each \( j \), \( g_{ij} \) of \( u \)***********
*  
*  
\[ \text{do 70 i = 1, n} \]
\[ \text{do 80 j = i+1, n} \]
\[ \text{wiktu} = 0.0d0 \]
\[ \text{wjktu} = 0.0d0 \]
\[ \text{do 82 k = istart(j), istart(j+1)-1} \]
\[ \quad \text{if(wc(k) .ne. i .and. wc(k) .ne. j) \&} \]
\[ \quad \quad \text{wjktu} = \text{wjktu} + wnz(k)*u(wc(k)) \]
\[ \text{82 continue} \]
*  
*  
\[ \text{do 84 l = istart(i), istart(i+1)-1} \]
\[ \quad \text{if(wc(l) .ne. i .and. wc(l) .ne. j) \&} \]
\[ \quad \quad \text{wiktu} = \text{wiktu} + wnz(l)*u(wc(l)) \]
\[ \text{84 continue} \]
*  
*  
\[ \text{do 86 j1 = istart(j), istart(j+1)-1} \]
\[ \quad \text{if(wc(j1) .eq. j) then} \]
\[ \quad \quad \text{wj} = \text{wnz(j1)} \]
\[ \quad \quad \text{go to 87} \]
*
endif  
continue  
continue

*  
do 88  i2 = istart(i), istart(i+1)-1  
if(wc(i2) .eq. i) then  
    wii = wnz(i2)  
    go to 89  
endif  
continue  
continue

*  
do 90  l = istart(i), istart(i+1)-1  
if(wc(l) .eq. j)then  
    wij = wnz(l)  
*  
d(i, j) = wii*wjj - wij**2  
d(j, i) = d(i, j)  
*  
sg(i,j) = (wjj*wiktu - wij*wjktu)*  
&  
    (wii*wjktu - wij*wiktu)  
sg(j,i) = sg(i,j)  
*  
go to 91  
*  
endif  
continue  
*  
*  
sg(i,j) = wjj*wiktu*wii*wjktu  
sg(j,i) = sg(i,j)  
*  
d(i, j) = wii*wjj  
d(j, i) = d(i, j)  
*  
*  
91  continue  
*  
*  
gw(i, j) = gw(i,j) + sg(i,j)*wgt  
gw2(i, j) = gw2(i,j) + sg(i,j)*(wgt**2)  
g2w2(i, j) = g2w2(i, j)+(sg(i,j)**2)*(wgt**2)  
gw(j,i) = gw(i,j)
gw2(j,i) = gw2(i,j)  
g2w2(j,i) = g2w2(i,j)

*  
80     continue  
70     continue  
**  
**  
100    continue  
*  
**********Total sum of gi and gisquare**************************************************
*  
do 300 i = 1, n  
do 350 j = i+1,n  
    sumgw(i, j) = sumgw(i, j) + gw(i, j)  
    sumgw2(i, j) = sumgw2(i, j) + gw2(i, j)  
    sumg2w2(i, j) = sumg2w2(i, j) + g2w2(i, j)  
350    continue  
*  
    sumgw(i,i) = sumgw(i,i) + gw(i,i)  
    sumu2w(i) = sumu2w(i) + u2w(i)  
*  
300    continue  
*  
**********Total Sum of w and splsize******************************************************
*  
    tsumw  = tsumw + psumw  
    tsumw2 = tsumw2 + psumw2  

    sumsize = sumsize + splsize  

    write(*, *) 'sumsize = '  
    write(*, 99) sumsize  
    write(*,*)  
**********Computing wbar and varwbar******************************************************

    wbar = tsumw / sumsize  
    varw = tsumw2 / sumsize - wbar**2  
*  
*
********** Stopping Rule and Variance of each estimate **********

if (varw/(sumsize*(wbar**2)).gt.(epsilon/zalpha2)**2) then
   do 500 i = 1, n
      eg(i,i) = sumgw(i, i)/ tsumw
   do 550 i5 = istart(i), istart(i+1)-1
      if(wc(i5) .eq. i) then
         wii = wnz(i5)
         sig(i, i) = 1.0d0/wii + eg(i,i)/(wii**2)
      endif
      continue
   continue
   continue
   continue
   count = count + 1
   write(*,*) 'sig = ',
   write(*,97) (sig(i,i), i=1,n)
   write(*,*)
   go to 510
else
   do 600 j = 1, n
      do 650 k = j+1, n
         eg(j, k) = sumgw(j, k) / tsumw
         vargw(j,k) = sumg2w2(j,k) / sumsize -
                        (eg(j,k)**2)*(wbar**2)
         &
         covgw(j,k) = sumg2w2(j,k) /sumsize -
                        eg(j,k)*(wbar**2)
         &
         s2(j,k) = 2*eg(j,k)*covgw(j,k)
         s3(j,k) = eg(j,k)*eg(j,k)*varw
         &
         sig2hat(j,k)= (vargw(j,k) - s2(j,k) +
                        s3(j,k))/(wbar**2)
         &
         sighat(j,k) = dsqrt(sig2hat(j,k)/ sumsize)
         sev(j,k) = sighat(j,k) / (d(j,k)**2)
         sev(k,j) = sev(j,k)
      do 655 j2 = istart(j), istart(j+1)-1
         if(wc(j2) .eq. k ) then
            wjk = wnz(j2)
\[ v(j,k) = -\frac{wjk}{d(j,k)} + \frac{eg(j,k)}{(d(j,k)^2)} \]

\[ v(k,j) = v(j,k) \]

\* go to 656 \*

\* endif \*

655 continue

**

\[ v(j,k) = \frac{eg(j,k)}{(d(j,k)^2)} \]

\[ v(k,j) = v(j,k) \]

* 656 continue

* *

650 continue 

* eg(j, j) = sumgw(j, j) / tsumw
vargw(j,j) = sumg2w2(j,j) / sumsize - 
& covgw(j,j) = sumg2w2(j,j) / sumsize - eg(j,j)*(wbar**2)
s2(j,j) = 2*eg(j,j)*covgw(j,j)
s3(j,j) = eg(j,j)*eg(j,j)*varw
sig2hat(j,j) = (vargw(j,j)-s2(j,j)+ s3(j,j))/(wbar**2)
sighat(j,j) = dsqrt(sig2hat(j,j)/ sumsize)
* 

do 660 i6 = istart(j), istart(j+1)-1
if(wc(i6) .eq. j) then
 wjj = wnz(i6)
 v(j,j) = 1.0d0/wjj + eg(j,j)/(wjj**2)
 sev(j,j) = sighat(j,j)/(wjj**2)
* 

go to 661 *
* 
endif *
* 660 continue *
* 661 continue *
* 600 continue *
* endif 

*************Printing************
write(*,*) 'Number of iterations = '
write(*,99) count + 1
write(*,*)
*
do 1 i = 1, n
  do 2 j = 1, n
    if(v(i, j) .ne. 0.0d0) then
      write(*, 997) v(i,j)
      write(*, 98) sev(i,j)
      write(*,*)
    endif
 *
 2 continue
1 continue
*
*
***************Formatting**************
***************
97 format(ix, 10f10.4)
98 format(ix, f10.4)
99 format(ix, i10)
999 format(ix, 10i5)
997 format(ix, f10.4)
*
stop
end
*
*
***************Subroutines***************
***************
Subroutine weight(u, n, nz, wc, wnz, sig, istart, wgt)
*
integer n, nz, i4, i7
integer ii
real*8 utwmsu
real*8 wgt
integer istart(n+1), wc(nz)
real*8 u(n), wnz(nz), sig(n,n), wms(nz)
*
real*8 dexp
**
**
utwmsu = 0.0d0

* do 800 ii = 1, n
  do 810 i4 = istart(ii), istart(ii+1)-1
    if(wc(i4) .eq. ii) then
      wms(i4) = wnz(i4) - 1.0d0/sig(ii, ii)
    else
      wms(i4) = wnz(i4)
    endif
  810 continue
  800 continue
*
* do 820 ii = 1, n
  do 830 i7 = istart(ii), istart(ii+1)-1
    utwmsu = utwmsu + wms(i7)*u(wc(i7))*u(ii)
  830 continue
  820 continue
* wgt = dexp(-.5d0 * utwmsu)
***
  return
end

******************************************************************************
APPENDIX C PVM CODE FOR PARALLEL IMPLEMENTATION OF AIS

* Include Fortran PVM header file
  include 'fpvm3.h'
*
* parameter (NPROC = 4)
  integer mytid, me, numt, i
  integer tids(0:NPROC)
*
* Enroll in PVM
  call pvmfmtid(mytid)
*
* Find out if Processor is a Parent or Child
  call pvmfparent(tids(0))
  if(tids(0) .lt. 0) then
    tids(0) = mytid
    me = 0
  else
    * Receive tids Array and Set Me
    *
    call pvmfrec(tids(0), 0, info)
    call pvmunpack(INTEGER4, tids, NPROC, 1, info)
  do 10 i = 1, NPROC-1

10 continue
if(mytid .eq. tids(i)) me = i
10 continue
*
endif
endif

**************************************************************************************************************************
** At this point, all NPROC tasks are equal now */
** and can address each other by tids(0) thru tids(NPROC-1) */
** for each process me -> process number[0 :NPROC-1] */
**************************************************************************************************************************
*
call ais(me, tids, NPROC)
*
* Program Finished Exit PVM
*
call pvmfexit
stop
end
APPENDIX D TEMPLATE FOR PARALLEL IMPLEMENTATION OF AIS ON THE DEC8400

#! /bin/csh

# All lines below which start with
#$ -cwd

#»#############«####««#»#############»»############################
# Send error messages to test.stderr, normal output to
# test.stderr
#$ -e test.stderr
#$ -o test.stdout

#$ -N job
#-Mail messages to the following mail address when the job
# Begins and when the job Ends. Please insert YOUR mail address.
##$ -mu MAIL_ADDRESS@iastate.edu
#$ -mu bchitou@iastate.edu
#$ -mb
#$ -me

# The 4 below indicates to DQS that you will be using 4 processors.
# DQS will then wait until all 4 processors are available, and
# will reserve them for your exclusive use.
##
## When using KAP, you will ALSO have to set the environmental
## variable PARALLEL (as seen below) to actually get 4 processors.
## Only use 4 with the FARM-X queues, use 1 for all other queues.
#$ -G FARM-X 4

attach -n nag
# Set up environment for KAP
limit stacksize unlimited
setenv KMP_STACKSIZE 24000000
setenv KMP_SPINLOCKS off

# Compile and link with main program
kf90 -fxapargs='--conc -o naispars naispars.f -lnag
f90 -o naispars naispars.f -lnag

# Run the job with 1, 2, 3 and 4 processors to  
# show how performance scales with the number of processors.

setenv PARALLEL 4
echo "Env var PARALLEL set to $PARALLEL"
time pnaispars < sainv.dat > /dev/null

setenv PARALLEL 3
echo "Env var PARALLEL set to $PARALLEL"
time pnaispars < sainv.dat > /dev/null

setenv PARALLEL 2
echo "Env var PARALLEL set to $PARALLEL"
time pnaispars < sainv.dat > /dev/null

setenv PARALLEL 1
echo "Env var PARALLEL set to $PARALLEL"
time pnaispars < sainv.dat > /dev/null

# Serial
echo "Serial time"
time naispars < sainv.dat > /dev/null
BIBLIOGRAPHY


ACKNOWLEDGMENTS

First of all, I would like to express my sincere gratitude to my major professor Dr. Mervyn Marasinghe for advising me with kindness and patience.

I am very grateful to Dr. Harville for suggesting the problem and leading the way. I am also thankful to my committee members, Dr. Yasuo Amemiya, Dr. Kenneth Koehler, Dr. John Stufken, and Dr. Cornette for their constructive suggestions.

I am grateful to Dr. Jim Hoekstra and Dr. James Coley for assisting me on FORTRAN programming and parallel computing. My sincere thanks go to Dr. H.T. David for warm encouragement, inspiration, and new ideas.

My deepest gratitude goes to Mr. and Mrs. Simmons for providing me with an excellent spiritual direction, a mentorship and a path to freedom. You are my heroes.

To my host family, Mr. and Mrs. Whatoff, I am sincerely honored to have you as dad and mom. You have shown me that love is the only reason for living. I have learned with sincere desire. Thank you.

Most of all, I thank my parents, Raouf Chitou and Akuele da Silva for their invaluable support throughout my life.

Many thanks to my fiancee, Mrs. Raymonde Degbey for patience, support, courage, and love. You have always challenged me for constant improvement in all areas of life, I am deeply grateful.

My best wishes to all my friends who supported me emotionally and financially throughout my graduate school.