ABSTRACT

A SOFTWARE SYSTEM FOR LARGE-SCALE STRUCTURAL OPTIMIZATION

by
Keith MacBain

This work is driven by recent developments in mathematical programming, the state–of–the–art of structural optimization, the spectacular performance of linear programming algorithms, and computer hardware developments which imply that applications of structural optimization might be used commonly in engineering design. Currently, there are few general purpose optimization routines available to the structural engineer and much of the work has addressed specific classes of problems. Further, there is little widespread use of the available routines, partly due to the large amount of familiarity one must have with the specific details of both the problem and the optimization method. In response, it is the intention here to prototype a software system that implements a general approach for structural optimization using the latest in mathematical programming techniques.

This work develops a general system that can be used for a variety of structural optimization problems in a manner analogous to the finite element method for structural analysis. The most commonly used structural elements, truss and beam, are included as well as techniques for plate optimization. Consideration is given to the software requirements of a general purpose structural optimization system and the demands of large structural systems typically encountered in design practice.

This general approach is aimed at using classical methods taken directly from the area of mathematical programming, specifically linear programming, which has seen considerable change in the last ten years. Here, sequential linear programming (SLP)

techniques are shown to handle a wide variety of structural constraints including stress constraints, displacement constraints, buckling, and frequency constraints. It is the purpose of this thesis to bring the latest developments in linear programming to the field of structural optimization in the form of a general purpose, state–of–the–art structural optimization system. The model was tested for sample structures and it was shown to effect a reduction in total structure volume of up to 80%.

A SOFTWARE SYSTEM FOR LARGE-SCALE STRUCTURAL OPTIMIZATION

by
Keith M. MacBain

A Dissertation
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirement for the Degree of
Doctor of Philosophy in Civil Engineering

Department of Civil Engineering

May 2001

Blank Page

APPROVAL PAGE

A SOFTWARE SYSTEM FOR LARGE-SCALE STRUCTURAL OPTIMIZATION

Keith MacBain

Dr. William R. Spillers, Thesis Advisor — Date

Distinguished Professor of Civil and Environmental Engineering, NJIT

Dr. C. T. Thomas Hsu, Committee Member — Date

Professor of Civil and Environmental Engineering, NJIT

Dr. M. Ala Saadeghvaziri, Committee Member — Date

Associate Professor of Civil and Environmental Engineering, NJIT

Dr. John R. Schuring, Committee Member — Date

Professor of Civil and Environmental Engineering, NJIT

Dr. John Tavantzis, Committee Member — Date

Professor of Mathematics, NJIT

BIOGRAPHICAL SKETCH

Author: Keith MacBain

Degree: Doctor of Philosophy in Civil Engineering

Date: May 2001

Undergraduate and Graduate Education:

- Doctor of Philosophy in Civil Engineering,
  New Jersey Institute of Technology, Newark, NJ, 2001
- Master of Science in Structural Engineering
  New Jersey Institute of Technology, Newark, NJ, 1997
- Bachelor of Engineering in Civil Engineering
  New Jersey Institute of Technology, Newark, NJ, 1996

Major: Civil Engineering

Presentations and Publications:

MacBain, K., Saadeghvaziri, M. A., and Spillers, W. R., "Examples of Stress Hardening and Softening in Three–Dimensional Beams," Journal of Structural Engineering, Vol. 125, No. 8, September 1999

MacBain, K. and Saadeghvaziri, M. A., "Material Properties of Recycled Plastics and Analytical Modeling," Journal of Materials Engineering and Performance, ASM International, Vol 8 (3), June 1999

Saadeghvaziri, M. A. and MacBain, K., "Sound Barrier Applications of Recycled Plastics," Transportation Research Records, No. 1626, TRB, September, 1998

Stop, look, listen.

ACKNOWLEDGMENT

I would like to express my deepest gratitude to Dr. William Spillers who served as more than my advisor over the years. His wisdom is great and I consider myself fortunate to have worked with him. Dr. M. A. Saadeghvaziri has also been a continual source of inspiration and direction and it has been a pleasure to know him. Special thanks are also given to committee members Dr. C. T. Thomas Hsu, Dr. John R. Schuring, and Dr. John Tavantzis for their valuable support and guidance. I also wish to thank Dr. David Perel for his unfailing assistance and immeasurable expertise, both of which were instrumental in making this work possible. He is truly a gem, not often found. The financial support received through a Presidential Fellowship cannot be overlooked and is greatly appreciated. Additionally, the help and support from graduate students Selahattin Ersoy and Alireza Yazdani–Motlagh was most encouraging.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

Chapter **Page**

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES
(Continued)

Figure **Page**

CHAPTER 1
INTRODUCTION

This thesis is concerned with structural optimization, which is formulated as a mathematical programming problem as

$$\begin{aligned} &\text{Minimize } f(x) \\ &\text{subject to } g_i(x) \leq 0; \quad i = 1, \ldots, m \end{aligned} \tag{1}$$

Typically the objective function f represents the volume of material and the constraints g refer to the equations of structures and requirements concerning stresses and displacements although other objective functions and constraints are possible as the examples of this thesis indicate. It is a fact that no robust algorithm exists which will solve large–scale structural optimization problems. Were that not the case there would be no point to this thesis.

The overall goal of this thesis then is to design a software system which has the potential of mirroring the finite element packages now on the market. These packages are very robust (they solve a wide class of problems), they are inexpensive to use and therefore available to the general engineering profession for common applications, and they can handle large practical problems. The success of these finite element packages incidentally derives largely from our success with solving large systems of linear equations.

Optimization problems are generally more difficult than the problems typically dealt with using the finite element method. This is certainly clear of problems of linear stress analysis in which the solution of a single system of linear equations is involved. Generally linear mechanics problems have a unique solution which is of course not the case in problems of structural optimization. And in the past there have been difficulties

with finding optimization algorithms that work. That was certainly the case up to the 1990's when problem sizes numbered in the tens of equations.

During the 1990's there was a conjunction of activities that gave new hope to mathematical programming problems. The most obvious of these was the advances in computer hardware which decimated old concerns over problems of computer storage capacity. And of course computer speeds increased dramatically. The second development of the 1990's had to do with the so–called interior point methods which seem to have derived from the work of Karmarkar in 1984. Karmarkar showed that it could be more efficient to move through the interior of the feasible region when solving an optimization problem rather than staying on the boundary of it. His work had the added advantage of reducing linear programming problems to the solution of linear equations, where we have become very efficient.

Linear programming (LP) has always remained something of an enigma in the mathematical programming world. While general–purpose mathematical programming algorithms were dragging along having difficulties solving even small problems, linear programming algorithms were successively solving much larger problems. That remains the case today where LP problems with 12 million variables are cited [Bixby] and LP problems with thousands of variables have become routine. It is not surprising then that sequential linear programming (SLP) algorithms are proposed here for a general–purpose structural optimization system.

In the work that follows, a rather general description of structural optimization is first presented, then an SLP algorithm is proposed as a general purpose solver, next the outline of a general purpose software system for structural analysis is proposed, and

finally some typical applications are presented. In summary, this thesis presents a prototype of a general–purpose software system for structural optimization.

CHAPTER 2
STRUCTURAL OPTIMIZATION

The goal of structural optimization, as defined here, is to improve some initial structural system and ultimately identify a best design within a given set of constraints. It should be noted that although the equations of structures are often linear, the functional dependence of structural problems on design parameters is typically nonlinear, implying that most structural optimization problems are nonlinear programming problems. Developing a software system to solve structural optimization problems under a wide variety of constraints is the main focus of this thesis.

2.1 Previous Work and State-of-the-Art

Clearly, for structural optimization, an algorithm of some sort is required since the exhaustive search method is prohibitive for all but the most trivial problems. The total number of unique structures one can construct is an exponential function with the number of variables as the exponent and the number of possible values a variable can have as the base. Consider for example a simple structure with six variables (e.g., member sizes) and 10 possible values for each variable. In this case there are one million (10^6) distinct combinations, all of which must be evaluated for an exhaustive search to be carried out. Because this number increases exponentially with the number of variables, one can easily see that there should be a defined method to identify the optimum design.

In the most common formulation, the design variables are member sizes (i.e., cross sectional area) and the constraints are normally limits on member stresses or displacements. Often, weight or volume is minimized as this is a measure of the material used and thus the total cost. This basic problem is the focus of many optimization studies

due to its convenient formulation but there are many other factors that influence the total cost of a real structure. There has been work on more practical versions of structural optimization problems such as construction cost [Xu, 1995]. Other optimization problems commonly considered are shape [Pedersen, 1973] where the geometry of the structure is varied, and topology where the member connectivity is allowed to change. Additionally, specific dynamic properties of the structure such as frequency can be considered [Cassis, 1976; Xie, 1994].

In terms of existing software systems, some commercially available computer analysis programs such as ANSYS use sequential unconstrained minimization techniques (SUMT). These methods convert a constrained optimization problem into an unconstrained problem by introducing a penalty that adversely affects the objective function when constraints are violated. This allows for a very general formulation but is not however, entirely reliable and has been noted to be computationally expensive. This is because the approach overlooks the nature of the problem and computes the gradients numerically, a computationally expensive procedure. Furthermore, the number of independent variables is normally so limited that large–scale problems are not possible. Implementations such as ANSYS can strongly limit the number of variables one can use.

The most effective software now available uses what are called optimality criteria methods. These methods are typically heuristic and cannot be used in a general algorithm because they require the formulation to change as the objectives and constraints are varied. These methods are based on developing a condition that will be satisfied at the optimal point and then using this condition to construct an iterative scheme. Unfortunately, as the objective or constraints change, so does the optimality condition and thus a different optimality criteria is required. One detailed report [Rozvany, 1993]

expresses several formulations that can be used with different constraints but it is noted that each different set of constraints results in a different formulation. This does not lead to a single method applicable for a wide variety of problems.

The system MBB–LAGRANGE [Hornlein, 1993] uses a variety of methods including penalty, sequential linear programming, multiplier, optimality criteria, and reduced gradient methods. A numerical comparison of optimization methods was done [Schittkowski, 1994] with this package comparing the performance of several algorithms on 79 test problems including seven real–life problems ranging from one to 144 design variables and two to 2,136 degrees of freedom. In this work, no single algorithm was reported to solve all problems, further illustrating the need for a more robust approach. Recursive quadratic programming showed the best performance in this respect and was able to solve all but 13 problems within the specified accuracy and number of iterations. The authors did note, however, that there was no effort made to tune the control options and that an experienced user could obtain better results.

Another system, IDESIGN uses four state–of–the–art nonlinear optimization algorithms with efficient and reliable implementations [Arora, 1989]. This program allows the user to include subroutines that describe the specifics of a design problem and is structured to include a general–purpose interactive capability. This general–purpose optimization program has been under development since the late 1970's and has a wide range of options, including the ability to perform dynamic response optimization such as vibration isolation and designs for earthquake resistance among others.

Research with genetic algorithms (GA) has shown good results (e.g., Camp, 1998) and there are several advantages to using this approach. One of the strongest points is the ability to search the entire design space and thus possibly locate a global

minimum, even if it is a singular case (e.g., Sved, 1968). Another benefit is the ability to identify a number of good solutions rather than a single best solution. Furthermore, it is very easy to optimize with respect to a discontinuous set of design variables, a feature very useful for structural design where members are normally available in a finite number of sizes. Unfortunately, there is no guarantee that an optimal point will be obtained and it has been noted that GA can be computationally intensive. Additionally, there is no guarantee of feasibility or even progress at any given iteration and because of this, GA are not considered in this work.

It is not clear at this point what methods are most commonly used in practice although it is known that many real–life designs do not include any form of computer–based optimization effort. Often, the greatest effort in this regard is the engineering judgment of the designer. It has also been interpreted from the literature that most applications which do involve optimization use "in–house" programs that are written specifically for the needs of the user and are often neither freely available nor general purpose.

2.2 Structures as a Mathematical Programming Problem

Mathematically, the optimization problem can be expressed in the most general form as

$$\begin{aligned} &\text{Minimize } f(x) \\ &\text{subject to } g_i(x) \leq 0; \quad i = 1, \ldots, m \end{aligned} \tag{2}$$

where f is a scalar objective function of x, a vector of design variables and g_i represents one of the m constraints. In the above representation, both f and g are considered to be general nonlinear functions. In this work, the objective function f is taken as the total

structure weight. It is noted that although structural weight is an important part of the total cost of the structure, it clearly is not the only possible objective function to use. The set of constraints g is normally a vector but may be a scalar if there is only one constraint. In this context, a constraint is a upper or lower limit on a single structural response quantity and each additional constraint will add to the total number of constraints equations m. For example, one constraint might be an upper limit on the stress in a specific member. Given that there are normally several constraints on a structure, the dimension of g can be large.

In order to consider an optimization problem, there must exist more than one possible choice of designs, implying a statically indeterminate structure. Typically, there are a number of vectors, x that will satisfy the constraints, g. The collection of all such vectors is termed here the feasible space and the boundaries of this space will be referred to as the constraint surface. This space is not easily quantified in a general sense for all structures because it involves not only the constraints which are included but also the structure itself.

Some structural optimization problems have been shown to have isolated points, meaning here that the constraint surface is not connected. Specifically, in a case of two loading conditions [Spillers, 1975], as one approaches the optimal point with an arbitrarily small step, the objective is increased rather than decreased. In this case, it is because in the optimal solution, the cross sectional area of one member is zero which can be thought of as a discontinuity in the structure. Additionally, although there are specific cases of displacement related and frequency constraints that have been shown to be convex [Svanberg 1984], the solution space of a typical structural optimization problem is not convex.

2.3 Structural Equations

There are three different types of structures considered in this work, which also represent a majority of the commonly used structural analysis elements. These are the trusses, frames, and plates. Structural analysis is a well developed area and there are many texts [Wang, 1984] the reader can refer to for more information on the details and differences between them. For this work, the materials are assumed to be linear and elastic although non–linear geometry is also considered. For all of these structural elements, the following matrix equations apply.

$$\begin{aligned} N^T F &= P \\ F &= K\Delta \\ \Delta &= N\delta \end{aligned} \tag{3}$$

In the above, N is a transformation matrix that relates the local and global coordinate systems, F is a column vector of the member forces (internal forces), and P is a column vector of the nodal loads (applied loads). The matrix K is the primitive element stiffness matrix in the local element coordinate system, Δ is a column vector of the element displacements, and δ is a column vector of the nodal displacements. The first equation above represents force equilibrium at all joints (nodes), the second is a constitutive equation relating the element forces and the element displacements, and the third represents compatibility. These equations can be assembled to form the basic system of equations for linear elastic structures

$$\begin{aligned} N^T K N \delta &= P \\ K_E \delta &= P \\ K_E &\equiv N^T K N \end{aligned} \tag{4}$$

which defines the positive–definite elastic stiffness matrix, K_E.

Response quantities of the structure under some loading are termed here as state variables. These are variables that are dependent on the design variables. With these equations, one can now express the change in a state variable with respect to a change in a design variable. These are the terms that are needed to form a linear approximation of the constraints.

2.3.1 General Constraint Gradients

It will be seen later that it is necessary to express the constraints above in the Taylor series which requires the gradient of the constraints. This can be computed numerically through any number of numerical techniques but this procedure is normally computationally expensive and thus, best avoided. Fortunately, the gradients of structural equations can typically be expressed in a closed form and thus it is possible to obtain the gradients explicitly by simply differentiating equations (3).

Here, the general approach for explicitly obtaining the gradients is shown and specific equations are given in sections relating to each type of structure. Not all possible gradients are shown here, although the derivation is similar. When the member geometry is fixed, the three relations given in equations (3) above can be differentiated to obtain

$$\begin{aligned} N^T dF &= 0 \\ dF &= dK\Delta + K\,d\Delta \\ d\Delta &= N\,d\delta \end{aligned} \tag{5}$$

The above equations can be used to form any of the required gradients. For example, the change in joint displacement, $d\delta$ due to a change in member stiffness dK can be expressed using equations (3) and (5) as

$$
\begin{aligned}
N^T dF &= 0 \\
N^T(dK\,\Delta + K\,d\Delta) &= 0 \\
N^T dK\,\Delta &= -N^T K\,N\,d\delta \\
d\delta &= -K_E^{-1}\,N^T dK\,\Delta
\end{aligned}
\tag{6}
$$

Similar expressions will be shown for other quantities relative to their specific structure types as discussed later. Additionally, the derivative of the stiffness matrix, dK mentioned in equation (5) will also be shown as it specifically is related to each structural element.

2.3.2 Geometric Nonlinearities

Geometric nonlinearities are considered in this work through the addition of a geometric stiffness matrix, K_G. Several references are available for the formulation of this matrix, sometimes termed the stress–stiffening matrix. This will be shown later for each element type that it is used but here, the change in the basic system shown in equation (4) is described. Essentially K_G is added to the elastic stiffness matrix to form

$$(K_E + \lambda K_G)\delta = P \tag{7}$$

If the displacements are small, an eigenvalue analysis can be performed where λ represents a critical load factor.

2.3.3 Truss Structures

The truss element is the simplest of the structural elements considered here. It is a line element with only axial stiffness. In two dimensions, each unsupported node of a truss structure has two displacement degrees of freedom and in three dimensions, each node will have three degrees of freedom. It is often modeled as an axial spring and the element

has one degree of freedom in its local coordinate system, displacement along the axis of the element. The primitive stiffness matrix, K for the structure is diagonal and each row contains a term K_i corresponding to element i shown here as

$$K_i = \left[\frac{A_i E_i}{L_i} \right] \tag{8}$$

Element i contributes to the transformation matrix N a sub–matrix corresponding to the positive and negative ends of that element. Each sub–matrix has two terms determined from the member geometry.

$$N_i^+ = \begin{bmatrix} \cos(\alpha_i) \\ \sin(\alpha_i) \end{bmatrix}^T \quad N_i^- = \begin{bmatrix} -\cos(\alpha_i) \\ -\sin(\alpha_i) \end{bmatrix}^T \tag{9}$$

If the member geometry is fixed and the objective function is the total structure volume or weight, the independent variables are the member areas. The objective function f of equation (1) is then linear in these variables and the constant multiplying each variable is normally taken as the member lengths. If different material densities exist, one can include a factor reflecting the respective densities and easily get the total structure weight. As required by equation (5) the differential dK is shown here

$$dK = \frac{\partial K}{\partial A} dA \tag{10}$$

Thus the terms in the derivative of the member stiffness matrix K with respect to the independent variable, the member area A are shown as

$$\frac{\partial K_i}{\partial A_j} = \begin{Bmatrix} \frac{E_i}{L_i} & \text{for} \quad i = j \\ 0 & \text{for} \quad i \neq j \end{Bmatrix} \tag{11}$$

The derivative of an $m \times m$ matrix (e.g., the stiffness matrix K) with respect to an $m \times 1$ vector (e.g., the member areas A) is a set of m matrices, each with dimension $m \times m$. This set of matrices cannot be written conveniently but as seen in equation (6), it is normally multiplied by the vector of member displacements Δ. Thus, the terms of this product can be neatly collected in a single matrix. The result is a matrix with the number columns is equal to the number of free variables (e.g., members) and the number of rows equal to the number of members multiplied by the number of displacements an element can have, in this case one. Column j of this single matrix is shown here as

$$\Phi_j = dK_j \Delta \tag{12}$$

where dK_j is obtained from equation (11). This results in a highly sparse matrix, in fact for the truss element discussed here it is diagonal, so the computer implementation of this (mentioned later) is done with a vector, although in the following equations it should be thought of as a matrix with the dimensions noted above.

Joint displacement constraints are examined first. Here the joint displacement δ is a vector with one term for each degree of freedom and it is desired to constrain some or all of them to be within some upper and lower limits. With equations (6) and (12), the change in joint displacement $d\delta$ due to some change in member area (i.e., stiffness) dA can be expressed through

$$\begin{aligned} d\delta &= \frac{\partial \delta}{\partial A} dA \\ &= -K_E^{-1} N^T \phi \, dA \end{aligned} \tag{13}$$

Similarly, the change in member displacements $d\Delta$ is shown as

$$
\begin{aligned}
d\Delta &= \frac{\partial \Delta}{\partial A} dA \\
&= -N K_E^{-1} N^T \phi \, dA
\end{aligned}
\tag{14}
$$

Equations (13) and (14) describe the change in the entire set of displacements due to a change in member stiffness. Each of these shows the inverse of the structural stiffness matrix in the equation. This is the formulation used in this work, largely for the simplicity of implementation. However, it is known that matrix inversion is a costly procedure and for greater efficiency, an implementation that does not require the inverse is possible.

Allowable stress constraints can be handled in two ways. One can first note that for member *i*, the stress in that member is defined as

$$
\sigma_i = \frac{F_i}{A_i} \tag{15}
$$

A simple substitution is made for convience, such that S represents a diagonal matrix with the member areas along the diagonal and zeros elsewhere. With this, the change in member stress can be expressed by application of the chain rule as

$$
\begin{aligned}
S_i &= \frac{1}{A_i} \\
d\sigma &= S\, dF + dS\, F
\end{aligned}
\tag{16}
$$

Implementation of equation (16) has been shown to work in the examples discussed late but a second, more efficient method of dealing with stress constraints is to note that the stress in a member can be related to the displacement as shown

$$\begin{aligned} \Delta &= \frac{F}{A}\frac{L}{E} \\ \sigma &= \Delta\frac{E}{L} \end{aligned} \tag{17}$$

In other words, one can simply express allowable stress constraints in terms of allowable displacements multiplied by the constant E/L and the change in Δ is shown above in equation (14). Such a substitution can either be done by the user explicitly or handled in the software.

At times it may be desired to limit the force in a member to a certain value. This case is different from those previously mentioned because here it is possible to specify constraints that cannot be satisfied. To illustrate this, consider the case of a determinant bar with an axial load. As the area is changed, so is the member stress but clearly the force in the member is constant. Thus, it is possible to impose constraints on the level of forces on individual members in a structure but one should note that it is easily possible to create a problem that cannot be solved. When these are included, the change in member forces is expressed by

$$dF = (I - KNK_E^{-1}N^T)\Phi \tag{18}$$

where I represents an identity matrix.

Local member buckling is defined here for a truss as when the compressive force in a member exceeds the Euler buckling load $\pi^2 EI/L^2$. This requires the section constant I which is often not included in truss analysis. However, one can assume that the second moment of inertia of the cross section I is related to the section area by

$$I = c_0 A^{c_1} \tag{19}$$

Given this, one can proceed with buckling constraints in the same manner as force constraints although in this case, only a lower limit would be provided indicating a compressive force.

2.3.4 Frame Structures

The frame element is a line element similar to the truss but with axial, shear, and flexural resistance. It is often referred to as a beam element and both two and three dimensional frame elements are considered here. The main difference between these two is that for the latter, torsional resistance is included. Additionally, the three dimensional frame element has flexural terms corresponding to bending about two different axis, both perpendicular to the axis of the element whereas the two dimensional element only has terms corresponding to bending about one axis. Axial stiffness is included in both cases.

The two dimensional beam element has three degrees of freedom at each node. These are displacement along the axis of the element, displacement perpendicular to the axis, and rotation of the node. Element i contributes to the total primitive stiffness matrix K the following 3x3 sub–matrix located on the diagonal.

$$K_i = \begin{bmatrix} \frac{AE}{L} & 0 & 0 \\ 0 & \frac{4EI}{L} & \frac{2EI}{L} \\ 0 & \frac{2EI}{L} & \frac{4EI}{L} \end{bmatrix} \tag{20}$$

Similarly, the transformation matrices relating the local and global systems for the positive and negative ends of member i are given below.

$$N_i^+ = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \frac{1}{L_i} & -1 \\ 0 & \frac{1}{L_i} & 0 \end{bmatrix} \quad N_i^- = \begin{bmatrix} -1 & 0 & 0 \\ 0 & \frac{-1}{L_i} & 0 \\ 0 & \frac{-1}{L_i} & -1 \end{bmatrix} \tag{21}$$

The member displacement matrix Δ shown in equation (3) contains a single sub–matrix for each element. The sub–matrix corresponding to member i is

$$\Delta_i = \begin{bmatrix} dx \\ \alpha^+ \\ \alpha^- \end{bmatrix} \tag{22}$$

where dx is the elongation of a member along its axis and α is the angle of rotation at the positive and negative ends. This angle is measured between a line tangent to the deformed member shape at that end and a line connecting the two nodes. For frames, the member stiffness matrix is no longer a linear function of the cross sectional area. This is because the second moment of inertia of the cross section I also appears in the matrix. In this work, the second moment of inertia, I is assumed to be related to the area by the equation

$$I = c_0 A^{c_1} \tag{23}$$

where c_0 and c_1 are constants that can be determined for a given family of beams or columns.

The geometric stiffness matrix K_G used for this element models large displacements and small strains. Note here that it is depends only on the member force and length.

$$K_G = \frac{F}{30L}\begin{bmatrix} 0 & 0 & 0 \\ 0 & 36 & -3L \\ 0 & -3L & 4L^2 \end{bmatrix} \tag{24}$$

The first two terms of the change in member stiffness per a change in cross sectional for the 2D frame element can be expressed as

$$\frac{\partial K_i}{\partial A_i} = \frac{2Ec_0c_1}{L}\left\{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 2A^{(c_1-1)} & A^{(c_1-1)} \\ 0 & A^{(c_1-1)} & 2A^{(c_1-1)} \end{bmatrix} + (c_1-1)\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2A^{(c_1-2)} & A^{(c_1-2)} \\ 0 & A^{(c_1-2)} & 2A^{(c_1-2)} \end{bmatrix} + \ldots\right\} \tag{25}$$

Here, the second term is shown because it is possible to include the second order effects as will be discussed later. Furthermore, if $c_1 = 2$ as is commonly the case, using the first two terms is exact.

The three dimensional frame element is very similar to the 2D element just discussed. The 6 x 6 stiffness matrix for this element is essentially expanded to include the extra flexure, shear, and torsional stiffnesses.

$$K_i = \frac{1}{L}\begin{bmatrix} AE & 0 & 0 & 0 & 0 & 0 \\ 0 & I_xG & 0 & 0 & 0 & 0 \\ 0 & 0 & 4EI_y & 0 & 2EI_y & 0 \\ 0 & 0 & 0 & 4EI_z & 0 & 2EI_z \\ 0 & 0 & 2EI_y & 0 & 2EI_y & 0 \\ 0 & 0 & 0 & 2EI_z & 0 & 4EI_z \end{bmatrix} \tag{26}$$

Additionally, because there are more cross sectional properties, the number of equations that relate them to the cross sectional area should also be increased. The following equations are used

$$
\begin{aligned}
I_x &= c_0 A^{c_1} \\
I_y &= c_2 A^{c_3} \\
I_z &= c_4 A^{c_5}
\end{aligned}
\tag{27}
$$

where I_x is the torsional constant and I_y and I_z are the flexural constants. This provides for a more generalized approach because each section property can be defined by different functions if desired.

2.3.5 Buckling and Second Order Terms

If an optimal design is sought with respect to the buckling load, it is necessary to express the change in this factor with respect to a change in the design variables. This is because as the material is redistributed, the buckling load will likely change. However, the statement of the problem is normally such that the applied load is constant and the final design should simply support a that load without buckling, although it is implied that the critical load is near. The Rayleigh quotient for the eigenvalue problem in equation (7) gives the load factor λ

$$
\lambda = -\frac{\delta^T K_E \delta}{\delta^T K_G \delta} \tag{28}
$$

where in this case, δ represents the nodal displacements associated with the first mode shape.

As noted earlier, the member stiffnesses are not seen in the matrix K_G, which is a function of only the axial force and the member length. This is worth noting because although the member force does change with the member stiffness, this is ignored to

arrive at an approximation of the change in the load factor $d\lambda$ with respect to a change in the member stiffness.

$$d\lambda \approx \frac{-\Delta^T dK\,\Delta}{constant} \tag{29}$$

In equation (29), Δ is the set of member displacements associated with the first buckled mode shape, computed through the eigenvectors of equation (7).

It is trivial to ensure that a loading on an initial structure is at or below the critical load by simply scaling the structure. This will give an initial load factor that is greater than or equal to one and the change in this load factor should be greater than or equal to zero, indicating that the critical load has increased. Note that equation (29) contains the set of independent variables (e.g., member areas) through dK which was shown in equation (10) and thus resistance to global buckling can be expressed in a single constraint: the change in the load factor $d\lambda$ should be greater than or equal to zero.

Even though linear approximations are being used, it is possible to include the effects of second order terms. This is considered because the stiffness matrix for the frame element contains terms which are a second order function of the member areas. The approach used here is to approximate the second order terms but still have a linear expression by modifying the Taylor series approximation

$$\begin{aligned} f &\approx f_o + \nabla f\, dx + \frac{1}{2} dx_0^T \nabla^2 f\, dx \\ &\approx f_o + \left[\nabla f + \frac{1}{2} dx_0^T \nabla^2 f \right] dx \end{aligned} \tag{30}$$

where in this case, dx_0 is obtained iteratively to complete the linear expression in equation (30).

Clearly, the "exact" solution yields $dx_0 = dx$ so to approach this, one begins with the arbitrary assumption that $dx_0 = 0$, solves the optimization problem discussed later, and then sets $dx_0 = dx$. This forms a new expression from which a new dx is computed and completes one iteration. The function f being approximated here is the change in the member stiffness with respect to the member areas so Equation (25) gives expressions for ∇f and $\nabla^2 f$. It was initially thought that the extra iterations to compute dx_0 might be rewarded by the possibility of using a larger step size at each iteration. Unfortunately, this was not observed in the examples shown later and this procedure is not recommended.

2.3.6 Plate Structures

Plate elements are not explicitly considered in this work. Rather, an analogy [Timoshenko, 1940] was used to correlate a thin plate and a grid of beam elements. Interestingly, this analogy allows for the consideration of anisotropic plates. Summarized here, the familiar elasticity equations

$$\begin{aligned} \sigma_x &= \frac{E}{(1-\nu^2)}(\epsilon_x + \nu\epsilon_y) \\ \sigma_y &= \frac{E}{(1-\nu^2)}(\epsilon_y + \nu\epsilon_x) \\ \tau_{xy} &= G\gamma_{xy} \end{aligned} \tag{31}$$

can be expressed for an anisotropic plate as follows.

$$\begin{aligned} \sigma_x &= E'_x\epsilon_x + E''\epsilon_y \\ \sigma_y &= E'_y\epsilon_y + E''\epsilon_x \\ \tau_{xy} &= G\gamma_{xy} \end{aligned} \tag{32}$$

This shows four constants for the material properties which are chosen to coincide with the planes of symmetry with respect to the elastic properties. If one considers an isotropic plate, the four material properties can then all be expressed in terms of the modulus

$$
\begin{aligned}
E'_x = E'_y &= \frac{E}{(1-\nu^2)} \\
E'' &= \frac{\nu E}{(1-\nu^2)} \\
G &= \frac{E}{2(1-\nu)}
\end{aligned}
\tag{33}
$$

Similarly, the familiar plate constant D which depends on the material property E now has four terms in this analogy.

$$
\begin{aligned}
D_x &= \frac{E'_x h^3}{12} \\
D_y &= \frac{E'_y h^3}{12} \\
D_1 &= \frac{E'' h^3}{12} \\
D_{xy} &= \frac{G h^3}{12}
\end{aligned}
\tag{34}
$$

This leads to the beam element analogy and from this, Timoshenko shows that the differential equation of the deflection surface w is

$$
\frac{B_1}{b_1}\frac{\partial^4 w}{\partial x^4} + \left(\frac{C_1}{b_1} + \frac{C_2}{a_1}\right)\frac{\partial^4 w}{\partial x^2 \partial y^2} + \frac{B_2}{a_1}\frac{\partial^4 w}{\partial y^4} = q \tag{35}
$$

In the above, $B1$ and $B2$ are the flexural rigidities of elements oriented in the x and y directions while $C1$ and $C2$ are the torsional rigidities.

The direct conversion between the optimal grid and an equivalent plate is not done here but the steps above clearly outline the procedure. With the methods outlined

here, plate optimization can proceed in the same manner as outlined for three dimensional frames.

CHAPTER 3
A GENERAL PURPOSE SOLVER

The characteristics of a good algorithm for design optimization have been noted [Arora, 1989] to be generality, efficiency, and ease of use. It is thought here that such an approach would focus more on the formulation of the optimization problem than the specific method to solve it. A method that can handle a variety of problems well is often termed robust, and there has been much call for methods that are more robust [Thanedar, 1990]. A general–purpose optimization system must handle a variety of problems without the need to reformulate the problem or have intimate knowledge of the feasible space.

The objective function considered in this work is the amount of material used while the constraints can be displacements, stresses, and buckling. The most commonly encountered structural elements, specifically linear elastic truss and beam elements, will be used. A key point of this work is that it recognizes the interactive nature of the design process. With a general–purpose system, the engineer can specify particular design variables and objective function to obtain an initial design and then impose further constraints, change the design variables, or modify the objective to improve the design from that point. A general–purpose system would allow the designer to make these changes without additional consideration of the specific method. It is hoped that the work presented here will gain wider acceptance among structural engineers and provide a useful design tool which is easy to understand and implement without a tremendous degree of investment to the intimate details of a given problem.

3.1 Linear Programming

The branch of mathematics known as mathematical programming deals with optimization problems. There are several classes of problems, including dynamic, linear, quadratic, and geometric which are categorized based on how the objective and constraints are expressed. There are continual advances in mathematical programming including methods and algorithms that have brought a powerful collection of tools to the area of optimization and made it possible to solve large problems with greater reliability and speed. While it is sometimes possible to express specific structural optimization problems in one of these classes, (e.g., Arora, 1984) only linear programming will be considered in this work.

The selection of linear programming (LP) for this work is based on several factors. Primarily, there have been significant advances made in recent years in the field of linear programming. As a result, linear programming has much to offer as a component of a general purpose structural optimization solver. Linear programming has a rich history and has been used extensively in fields such as operations research for many years. There is a great deal of research that has been done in this area and new algorithms will continue to arise as this is a commonly used tool in many areas. Additionally, it is possible to express a wide range of problems in this format by making linear approximations of a general nonlinear function, as is done in this work. It is thought that the choice of linear programming is consistent with the goal of developing a single unified approach to general structural optimization.

3.1.1 Basic Linear Programming

For an optimization problem to be considered a linear program, both the constraints and the objective should be linear functions of the design variables. The reader is referred to any number of texts for a more complete description but some of the more important points will be outlined here. A standard LP is most commonly expressed in the form

$$\begin{aligned} &\text{minimize} \quad c^T x \\ &\text{subject to} \quad Ax = b, \quad x \geq 0 \end{aligned} \tag{36}$$

Here A is a matrix while b, c, and x are vectors. It should be noted that the minimum of a function is equivalent to the maximum of the negative of that same function. This means that the problem shown above can be stated equivalently as maximizing the negative of the objective and it is common to see an LP in either form. Further, one should note that inequality constraints can be written as equalities by introducing slack or surplus variables, thus the above is a reasonably general form. Equation (36) is often termed the primal problem and all linear programs have a related dual problem, which may be expressed as

$$\begin{aligned} &\text{maximize} \quad b^T y \\ &\text{subject to} \quad A^T y \leq c \end{aligned} \tag{37}$$

In the dual problem, there is no restriction on the sign of the dual variables, y and this will be returned to later. The primal – dual relationship gives that although the primal solution x and the dual solution y will normally be different, the value of the objective function for each is the same. In other words, the minimum of the primal problem in equation (36) and the maximum of the dual problem in equation (37) have the same value.

Linear programming problems were first solved by the simplex method which guarantees that if a solution exists, it will be found in a finite number of steps. This method locates the optimal point by moving along the perimeter of the feasible region until a point is found for which no movement along the perimeter will improve the objective function. There are many references available [Taha, 1992] regarding the specific details of how to solve linear programming problems with the simplex method.

It will be seen later that the non–negativity requirements of equation (36) must be released because in the formulation used here, the solution may be positive or negative. This is different than the standard LP which uses non–negative variables. There are different methods for working with unrestricted variables, and sometimes a procedure is implemented in the program. The most common method is to make a variable substitution for each free variable

$$x_f = x_p - x_n \tag{38}$$

Here x_f is the unrestricted variable and x_p and x_n are both positive, consistent with equation (36). A solution obtained in terms of these transformed variables can then be used to construct the unrestricted variables using equation (38).

Obviously this substitution doubles the number of variables if all are taken to be free, as is normally the case. A second approach, not often seen in the literature, is to recall that there is no such restriction on the dual variables mentioned before with equation (37). One can simply formulate the problem in terms of the dual variables and the solution will not need to be transformed back into the original problem variables. Additionally, there is no need to double the number of variables in the problem. This is the approach used here. It should be noted, however that the methods presented here are not dependent on this method and if one desired, the first method outlined above could

be used. There are already several LP solvers that include free variables explicitly so this is not a concern with that type of algorithm. As more sophisticated LP solvers become commonly available, it is thought that free variables will be routinely included in the solver.

3.1.2 Interior Point Methods

The majority of work in mathematical programming in the last ten years has been in the area of interior point algorithms. As the name implies, these methods search for an optimal point while always remaining interior to the feasible region. This is in notable contrast to the simplex method and its variations which improve a design by moving along the perimeter of the feasible region. Interior point methods have been used for both linear and nonlinear problems [Byrd, 1999; Boggs, 1996] including structural applications [Jarre, 1998].

Among the advances in the area of linear programming, interior point methods are at the forefront of recent developments, and have been noted for superior performance when dealing with large problems. There has been much focus on these methods since their introduction [Karmarkar, 1984] and it is not uncommon to see test problems with several thousand variables. Systems as large as 12 million variables [Bixby, 1991] have been solved with interior point methods. With this computing power, solving real structural problems is certainly possible.

3.1.3 Operation and Solvers

It is interesting to note that although the solution to an optimization problem lies on the boundary of the feasible region, interior point methods search for this point without

traversing the perimeter. This is perhaps the most notable difference from the simplex method and its variants. The search direction is normally obtained from the objective but to maximize progress, the feasible space is scaled. Karmarker's early work is termed a polynomial–time algorithm because it promises that the run time is, at worst, a polynomial function of the storage needed for the problem data.

Like the simplex method, there are a number of steps involved in approaching the optimal solution but interior point methods never exactly reach the optimal point. One can approach the solution within an arbitrarily closeness but not actually reach it exactly. Some researchers have proposed a combination of the two methods to overcome this but for most practical applications, the exact solution is not required. In fact, in this work it will be seen later that computation time can be decreased by forgoing the exact solution without any noticeable loss. This makes interior point methods even more attractive for this application.

There are several good LP solvers available at this time and based on the current level of interest and recent advancements, the future of LP is very promising. For this work, a single LP algorithm was used. This work does not use more advanced solvers (e.g., Vanderbei, 1989) because these are normally not free. The algorithm used here [Arbel, 1993] is freely available and solves the dual problem shown in equation (37). It is known as an affine–scaling dual algorithm and was chosen because it is an efficient interior point method as well as easy to follow and program. Simply outlined here, the dual algorithm is an iterative procedure which follows the steps outlined in Table 1.

Table 1 Affine – Scaling Dual Algorithm Steps

Step	Description	Equation
1	Select an initial feasible point, $z_0 > 0$ and y_0. Set the iteration counter $k = 0$	$A^T y_0 + z_0 = c$
2	Define the diagonal scaling matrix D for iteration k	$D_k = diag\left[\frac{1}{z_1^k}\frac{1}{z_2^k}\cdots,\frac{1}{z_n^k}\right]$
3	Find the step direction dy by solving	$[A D^2 A^T] dy = b$
4	Find the step direction dz by solving	$dz = -A^T dy$
5	Compute the step size α	$\alpha = min\left\{\frac{-z_i}{dz_i} : \forall\, dz_i < 0\right\}$
6	Update the solution through	$y_{k+1} = y_k + \rho\alpha\, dy$ $z_{k+1} = z_k + \rho\alpha\, dz$
7	Increment k and return to step 2 or terminate if stopping criteria are satisfied	

The full derivation of the steps in Table 1 can be found in Arbel's book but some points will be mentioned here. Note that in step 1, the inequality of equation (37) is changed to an equality by the addition of the slack variables z. These are strictly positive and thus so is the diagonal scaling matrix D defined in step 2. At each iteration, the linear system in step 3 must be solved, which is the most computationally intensive part. In step 6, the variable ρ is a step size parameter, typically taken to be 0.95 and always between zero and one.

This method is simple to code and implement. It does not perform any pre–solve analysis or employ other more sophisticated tools which are commonly found in other LP

solvers. Other methods could easily be incorporated to the procedures used here, although this is beyond the intention of this work. It should be noted however that as the field of LP advances, the methods presented herein could easily take advantage of more sophisticated solving routines.

3.2 Sequential Linear Programming

Although structures are normally considered linear, this linear relation is between the forces and the displacements. However, the member and joint displacements are not a linear function of the member stiffnesses. A general structural optimization problem to minimize the volume of material used subject to a given set of constraints is nonlinear. Specifically, the objective is linear but the constraints are nonlinear. The approach used in this work is to make a linear approximation of the constraints in the vicinity of an initial design and cast this as an LP. Solving the LP will give a step size and direction that should be near the constraint surface if the step size and the constraint nonlinearity are not too large.

Sequential linear programming is the method of optimization chosen for this work because it is possible to express a wide range of problems in this format. The basis of all formulations herein will be sequential linear programming (SLP) which can incorporate the state–of–the–art in linear programming algorithms. This approach involves making an approximation of the problem (1) and solving this approximate sub–problem at each iteration to obtain a better estimate of the solution.

A nonlinear function can be approximated with the first few terms of the Taylor series as

$$f = f_o + \nabla f\, dx + \frac{1}{2} dx^T \nabla^2 f\, dx + \epsilon \tag{39}$$

where ε represents higher order terms. Given an initial point f_0 one can search for a step dx that will improve the objective while not violating the constraints. If only the first two terms of (39) are considered, the problem is linear in the step dx and an associated LP can be formed and then solved in a finite number of steps by a number of optimization routines.

The solution to this linear sub–problem will give a step toward the optimal point and this sequence is repeated until some convergence criteria are satisfied or for a specified number of iterations. For each step, the following linear approximation is made which converts the general nonlinear optimization problem to a problem which is linear in the step, dx.

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & g_i(x) \le 0 \end{array} \quad \Rightarrow \quad \begin{array}{ll} \text{minimize} & \nabla f_0 dx \\ \text{subject to} & g_0 + \nabla g_0 dx \le 0 \end{array} \tag{40}$$

One can easily note that the linear approximation in equation (40) can be cast into the dual LP form shown in equation (37) with the following substitutions.

$$\begin{aligned} b^T &= \nabla f \\ A^T &= \nabla g \\ c &= -g_0 \end{aligned} \tag{41}$$

Thus, at each iteration to complete the linear approximation and set up the dual LP, one must compute the gradient of the objective ∇f_0, the matrix of constraint derivatives ∇g_0, and the negative of the constraints g_0. All of these are evaluated at this some initial point x_0. When the dual LP formed with equation (41) is solved, the dual variable solution y is

actually the step dx in equation (40) for iteration i. The system is then updated according to the following equation before the next iteration.

$$x_{i+1} = x_i + dx \tag{42}$$

The SLP method has been chosen here due to its widespread use in other areas and recent advances in linear programming resulting in a large number of efficient algorithms available to solve such problems. There exists a good deal of software available for this problem (e.g., More, 1993). The development of an LP solver is not the intention of this work as there is a great deal that has already been done in this area. It should be noted that with this approach, it does not matter if there are one or several constraints of a particular type as in the optimality criteria methods noted earlier.

It has been noted that SLP is not globally convergent, which means that the method will not necessarily converge to a unique global minimum from an arbitrary starting point. This is not thought to be a problem however, for two reasons. First, one cannot undermine progress that has been made solely because the solution is not guaranteed to be the absolute best. Certainly it is foolish to discard an improved design that satisfies a given set of constraints simply because there is no guarantee associated with it. Second, analogous to the way steepest descent methods will normally converge to a local minimum depending on the starting point, it is thought that such a flexibility might give the designer the possibility to direct the solution by using an appropriate starting point.

Because this is to be performed iteratively, it is not necessary to obtain the exact solution at any given iteration. This is only one of the most noticeable advantages of the interior point methods. Currently, the LP solver begins with a zero vector as the starting point. This is somewhat arbitrary however and it is possible that a different starting point

would lead to an earlier solution. It was noted that for some problems, the direction of the LP solution didn't change much from one iteration to the next. With this in mind, it is thought that using the previous solution as a starting point in the LP may be worthwhile.

The number of LP and SLP iterations are both controlled manually in this work but investigation into the LP solver and stopping criteria could be useful. Specifically, it was found in this work that one does normally not need the exact solution at a given iteration, thus possibly reducing the time to solve the LP. If some convergence criteria were used, both of these parameters could be selected automatically by the program. Although a parametric study would give more insight as to the optimal number, it was noted that the optimal number may change depending on the problem and thus is best determined by the program at run–time.

3.2.1 Step Size

For structural applications, it is necessary to specify an allowable step size (i.e., change in member stiffness) that can be made at any iteration. The most obvious reason for this is that without restrictions, a step outlined in equation (42) could possibly cause the result to be negative, which of course is not possible for member areas. Clearly this means the step size should be limited such that

$$\begin{aligned} dx_i &\leq \alpha x_i \\ dx_i &\geq -\alpha x_i \\ 0<\alpha<1 & \end{aligned} \tag{43}$$

where α is referred to later as the step size parameter.

A second reason, however to limit the step size is for stability. This is because the constraint surface is nonlinear and a step that may remain within the linear

approximation of this surface, could actually fall well outside, particularly if the constraints are highly nonlinear. Thus the limitation on the step is normally governed by how well the linear approximation fits the problem. For practical use, it was found that values up to 0.5 could be used for more well behaved problems but if the problem is more nonlinear, values as small as 0.05 were used.

3.2.2 Scaling

At each iteration, one should begin with a point just within the feasible region. This should not be confused with the feasible starting point interior point methods normally require to begin. Note that both the structure and the LP each have a feasible region. In the context of the LP, the feasible point can simply be taken as zero, analogous to a null hypothesis. Additionally, more sophisticated LP solvers will generate such a feasible point if needed. The structure however should be treated differently.

It is desirable to begin with a point near the constraint surface because the linear approximation of the surface is centered about this point. Obviously if it is near a constraint, the approximation should be better than if it is far away. Also, it is possible that the linear approximation of the previous step was poor and the structure currently violates some constraints. To scale the current design to be nearer the constraint surface, a scale factor ρ is found and the entire structure is scaled by this amount.

For most constraints such as displacement or stress, it is a trivial matter to obtain this scale factor that will give a feasible point. One simply performs a line search over the set of constraints to identify the largest violation. If no constraint violations exist, the constraint closest to its allowable value is selected. The scale factor ρ is thus computed as

the maximum ratio of the current value of a state variable to the allowable value over the entire set of constraints.

$$\rho = \max\left\{\frac{\text{value}_i}{\text{allowable}_i}\right\} \tag{44}$$

The entire set of member areas can then be multiplied by this ratio. Beam elements are slightly more complicated because such a linear scaling is not necessarily exact. This is because some terms in the stiffness matrices in equations (20) and (26) are not linear in the member areas. If a higher order quantity such as joint rotation or member end rotations represents an active constraint, scaling should proceed according to the computed value ρ raised to the power c_1 used in equation (23). This of course, is only exact for trivial problems but gives a much more useful approximation.

In practice, even if the exact scale factor is available, it is normally not advisable scale the structure to exactly that point. One reason for this is that computer implementations can fail when dealing with the exact value of real numbers due to machine round–off. By providing a point slightly inside the feasible region, one is guaranteed that all constraints are satisfied and that a pre–solve analysis to obtain a feasible point will not be required in the LP solver. Throughout this work, a second factor, ξ was used to ensure that the scaled structure was within the feasible region rather than exactly at the boundary. With this and equation (44), the update of the solution shown in equation (42) actually becomes

$$x_{i+1} = \frac{\rho}{\xi}(x_i + dx) \tag{45}$$

where ξ is always greater than or equal to one. To ensure that this new iterate is feasible before the next iteration, a line search is performed which scans the constraints for either the greatest violation or the nearest to a boundary.

The factor ξ can be thought of as an exploration factor, allowing one to travel away from the current point in a direction away from the zero vector into the feasible region. As this factor is increased, the design becomes be further away from the constraint surface and conceivably a different direction for the optimal could be found in the coming iteration. This would be expected if one were near a local rather than global optimum, but this effect was not found in the examples used. One should note however, that if this factor is increased, the step size should also be increased appropriately if one wishes to reach the constraint surface.

3.3 Software Considerations

This work relies largely on the implementation of the methods outlined above. There are several computer programs that were written to implement the procedures outlined earlier. All work was done in the UNIX environment on an UltraSparc computer system. Although many shell scripts were written to create and process data, the two most important aspects are considered to be the computational details and the graphical user interface (GUI), and the details of each will be discussed shortly.

Three main components of the conceptual software layout are seen in Figure 1. Note that it is not necessary to use the same language for all components and in fact, it is wise to exploit the advantages of different languages. The analysis components perform the finite element analysis required for the evaluation of structural quantities such as

displacements, stresses, and such. These results are needed as outlined above for the optimization component of the system. Thus, these two components are where the majority of the computational work is done and as such, should be linked together as closely as possible because if communication between them is slow (e.g., if data is written to the disk), the performance will suffer. The third component, the GUI, does not need to be intimately tied to the other two because its function is normally generation and post–processing, even though analysis capabilities were provided. Separating the components in this way allows for greater flexibility because other components can easily be added.

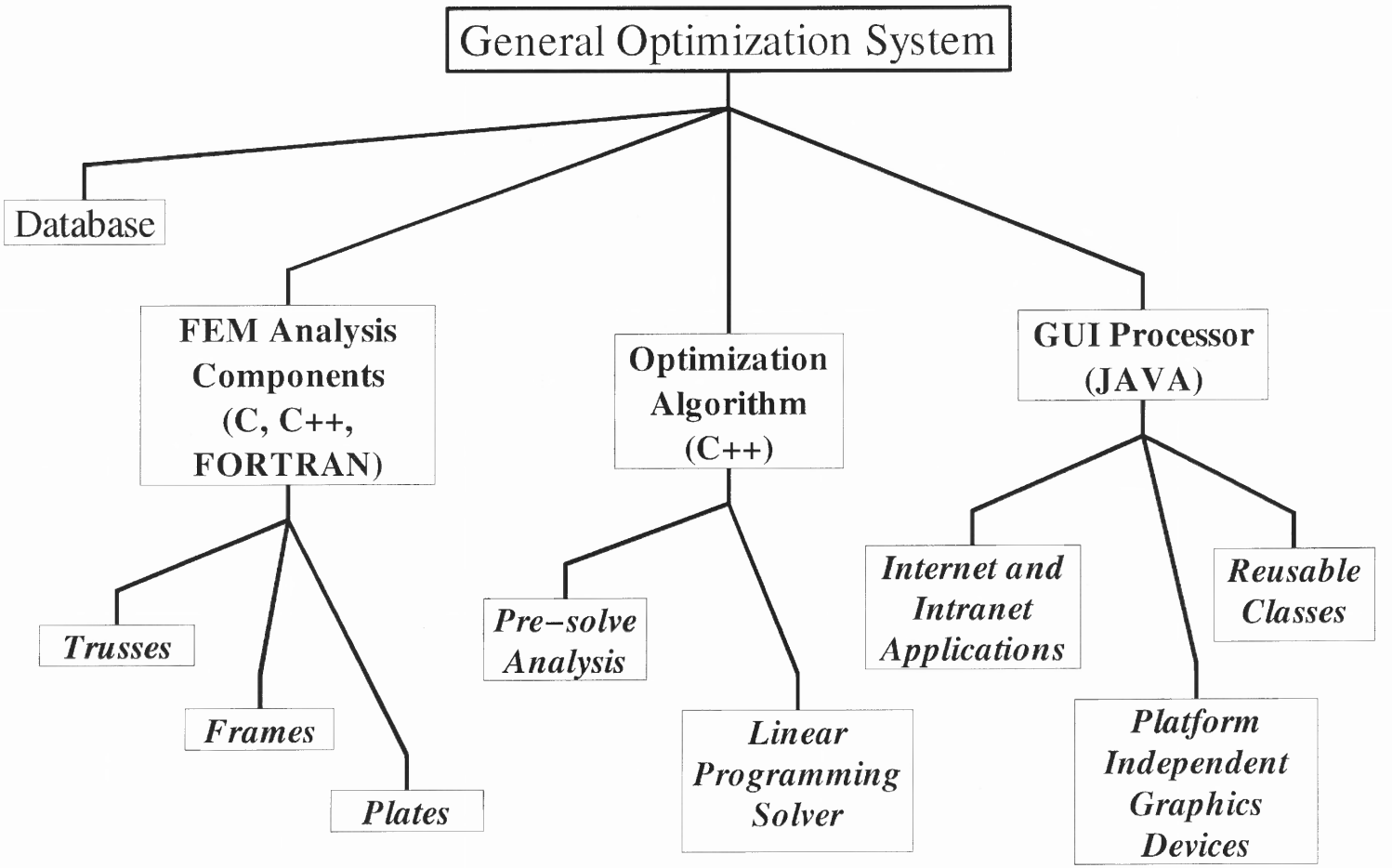

Figure 1 Conceptual Software Layout

Also shown in Figure 1 is a suggested point of entry for connection to a database, although this was not necessary here. It would be useful, however in real designs to obtain member sizes from a database because real member sizes belong to a discrete rather than continuous set. Even though the computed values may represent an optimal design, one needs to relate them to real values from a discrete set for any real structure. If these values were accessible through a database, the program could select among them. Such an implementation could possibly even favor less changes in member sizes, a real–world problem, by including that in the optimization problem.

3.3.1 Computational Details

The computer programming language C++ was used for all computational work and the code was compiled with the freely available GNU g++ compiler. C++ is a modern high–level object–oriented language. It supports inheritance, function and operator overloading, dynamic memory allocation, and several other features which are not used here. This is considered the best language for this application as it is common, well supported, easy to work with, and portable. Numerical precision can be greater than 96 bits, which is more than required here. Further, it is possible to incorporate routines written in other languages such as FORTRAN without loss of performance. The complete source code of the programs described here can be found in Appendix A.

One of the key features of the C++ programming language for this work is ability to allocate memory dynamically. As large systems are to be handled, large blocks of memory are often needed. However, it is desired not to request more memory than required for a given problem and there is no advantage in consuming large blocks of

memory for smaller problems. C++ provides a mechanism for exactly this task and all defined classes use it.

Similarly, overloaded functions can be written which have the same name but different parameter lists. This means that one can write the program once, overload the functions, and the appropriate one will be called based on the parameters. As an object–oriented language, user–defined classes can be constructed. A special class was created for each type structure investigated but as each can be derived from the same base class, object inheritance mentioned earlier means the main program doesn't need to be re–written based on the type of structure.

Although it is not strictly required for this work, a special class of object was created to simplify programming. This class was named 'matrix' and contains functions and overloaded operators to deal with the matrix equations. Overloading an operator allows the programmer to associate a symbol such as the plus sign with a function such as matrix addition. This simplifies code development because matrix equations can be written symbolically in the code rather than explicitly dealing with each individual element. Additionally, the strong type–checking system of C++ easily identifies erroneous statements. More importantly though, is the way memory can be handled when such a class is used.

The largest memory requirements of the program are in the matrices and it is important to deal with them efficiently. C++ supports two functions related to objects in general which are useful for this, specifically the constructor and destructor. In these, the programmer specifies what should be done when an object is created, destroyed, or its scope is lost. In this case, when a matrix is created, exactly the correct amount of space is requested for it. This means that memory is handled in an efficient manner because

specific requests for memory can be made at run–time when the exact requirements are known and memory can be released when it is no longer needed, even within a given function.

The details of the requests for memory are negotiated with the operating system and if a request is larger than the available RAM, space is allocated on the disk. Of course, disk access is much slower than RAM, but this is only done when no additional RAM is available. Fortunately the process is handled automatically and there is no need to write temporary files for large systems as was the norm several years ago. Thus, the size of a problem that the program can handle is not limited by the available RAM, which is normally considerable. For example, a structure with 1,000 degrees of freedom requires one million elements for the system matrix if it is stored densely. For double–precision variables which normally requires 8 bytes, the total storage is 8MB. This is a considerable amount but still well below the capacity of most modern PC's.

An important point to mention is that sparse matrix techniques were not used and only a few matrices noted earlier were not stored in dense fashion. This is admittedly a limitation but does not adversely affect the usefulness of this work. This was omitted for the sake of faster development of the algorithms used as well as smoother interaction between modules of the code. There are different methods for storing sparse matrices in memory and if any one is used, this must be accounted for in all functions that will access that block of memory.

The structural analysis follows traditional linear elastic analysis. The system matrix is assembled and Gaussian elimination is used to solve for the joint displacements. All routines were written by the author except the eigenvalue solver, which was adapted from the IBM Library subroutine, NROOT. Unfortunately, the eigenvalue solver not

only ignores sparsity, it proceeds to obtain all eigenvalues and vectors even though in this application, only the first is required.

The linear programming solver used in this work was mentioned earlier. It is very straight forward and there are few details to be mentioned. It is worth noting, however that this work began with different commercially available solvers illustrating the flexibility incorporated. By this it has been shown that a number of today's state of the art LP solvers can be used.

To begin an optimization run, a basic model and a set of constraints are required. The model is given to the program in the form of an input file which describes the initial geometry, loading, and member sizes. In this work, the input file was typically generated with scripts written for this purpose but this could easily be included in a GUI preprocessor as is commonly seen in FE programs. The program will look for a file named "basename.osi" where basename is a parameter that can be set by the user. At the end of the first successful iteration, a file named "basename.001.osi" is written and the three digit number is incremented for each successive iteration. This means that at the end of the optimization run, the working directory is populated with a collection of files, all similar to the input file but each containing data for that particular iteration. This was found to be very useful because it not only allows one to review the output of each iteration after the program has terminated, it also allows one to restart from any iteration, possibly with changes made to the design or the constraints.

It is important with any program to provide means to identify and handle possible problems that may occur. There are several reasons for abnormal program termination and in this work, error–handling techniques are accomplished through checking the return value of a function. This is more elegant than the FORTRAN method for

determining the success of a routine because it does not require extra variables in the parameter list and the return value can be ignored if desired. Each function is written to return a value, normally integer, and the value is checked before the program can proceed. For example, consider a function that returns the value zero after successful completion that would be called in the following statement

```
someFunction(parameter);
```

To ensure that the function is successful and terminate the program if a non–zero value is returned, the statement is written in the same line as

```
if(someFunction(parameters)) exit(0);
```

The program will terminate with a message if the constraints file has errors or unrecognized input. Linear scaling was mentioned earlier as a means to ensure that the current design is feasible but frame elements are not linear in the section areas. Because of this, after each scaling, a check is performed to ensure the structure is feasible. If it is not, two more attempts are made before the program exits with a message. In practice, it was found that no more than two attempts were required and only when the initial point was very far from feasible. If buckling is not among the constraints and a zero is found on the main diagonal while solving the system the program will exit and a message generated. All matrix operations that are dimensionally incorrect will cause immediate termination with a message.

Programs were written to accept a number of command line arguments to set the parameters at run–time rather than to recompile the source code when changes are made. These were found to be the most often changed parameters and it is thought that any optimization system should allow for these parameters to be user defined. Table 2 summarizes the case–sensitive command line arguments that can be set at run–time.

Table 2 Command Line Options

Flag	Description	Default Value
a	Step size	0.2
b	Basename	file
e	Output frequency	1
i	SLP iterations	1
I	LP iterations	10
r	restart	0
v	verbosity	0
V	LP solver verbosity	0
x	explore	1.01

3.3.2 Graphical User Interface

The iterative nature of this work naturally leads to the concept of a history, and it is often desirable to view the history of the optimization run. While it is possible to view this information numerically, it can be much more useful and efficient to review results qualitatively through a graphical representation of the entire structure. This is most easily understood by noting that although the objective function may decrease for a given iteration, it is often composed of positive and negative changes in the member sizes and it is difficult to determine the relative change without reviewing the entire set of members. The complete source code of the programs described here can be found in Appendix B.

The Java programming language was used for the post–processing programs. The freely available JDK 1.2 was used. This was selected primarily for its platform independence and ease of development when constructing a GUI. Although separate programs were developed and used for each type of structure, they differ only in the expected format of the input file and the specific details of the structural analysis. A

typical screenshot of one program is seen in Figure 2 which shows the menu items important for the GUI of an optimization system.

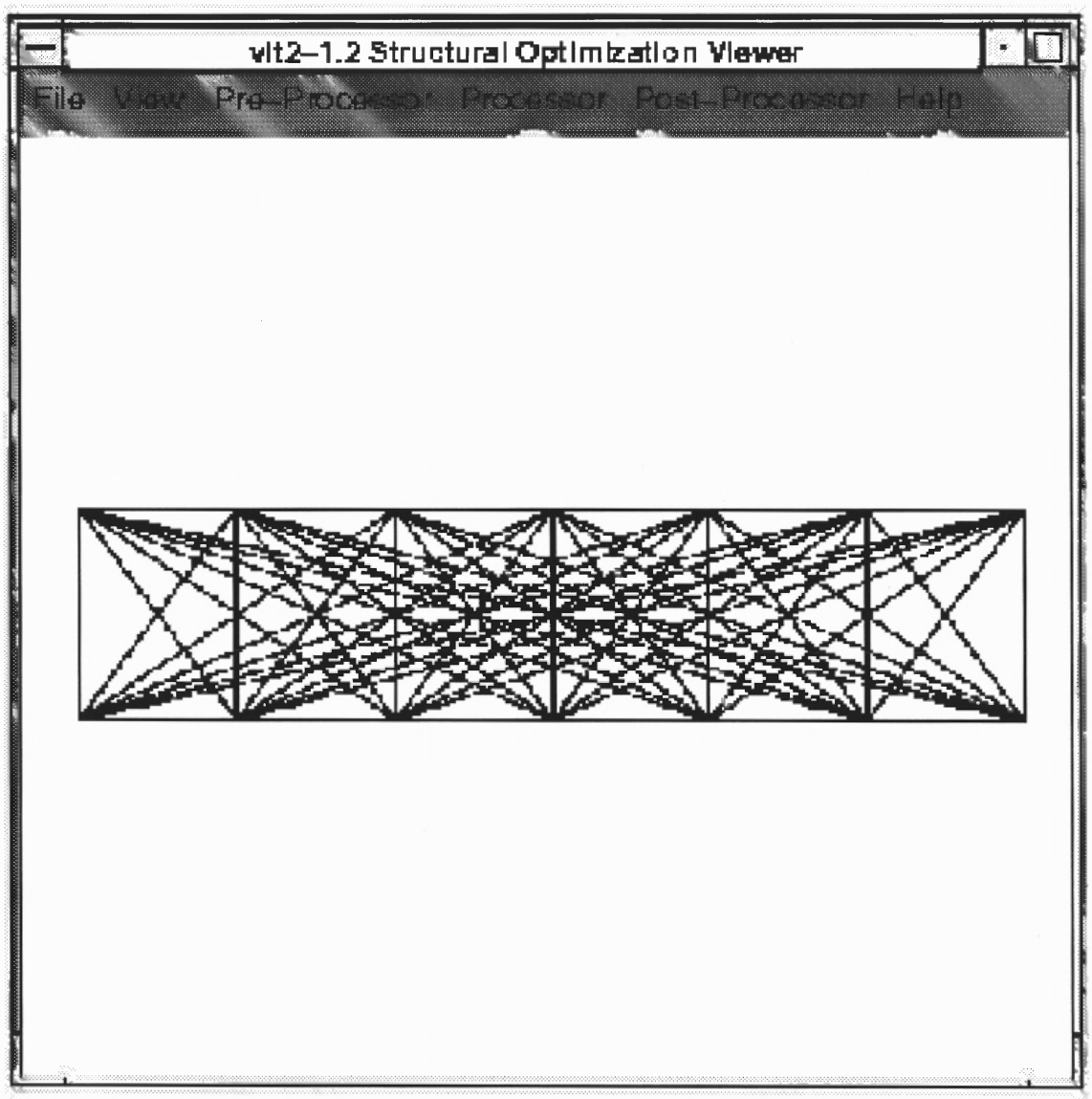

Figure 2 Screenshot of Graphical User Interface

As outlined, the GUI does not perform the optimization procedure mentioned above but rather reads and displays the data generated by the processor. It is possible however, to initiate the processor from the GUI giving the user the impression of seamless integration. Although the author prefers the non–graphical method because it is normally quicker and can be automated, this is thought to be an important part of a general optimization system. This is because nearly all modern applications are graphical in nature and many users are now unwilling to work with programs without a GUI. By separating the components as done here, the optimization system is flexible yet still accommodates the many users who prefer graphical interaction. Additionally, it is easier to add components or even connect each component to a central database.

Similar to C++, Java is an object–oriented language and supports classes but with Java, there is a rich library available allowing the programmer to derive custom classes based on existing classes. Because graphical components are often complicated, this makes Java a natural choice for a GUI. The most important parts of the program are derived from the JDK classes "Frame" and "Canvas". The first class provides the window and the second gives a drawing pane where all rendering of the structure is done. All Java programs use classes that were derived from these two.

An additional benefit of using Java in this application is that the language syntax is very similar to C, thus code written in C can often be inserted directly into a Java program. This was particularly true for the structural analysis aspect of the program. The post–processor has structural analysis capabilities, identical to those mentioned before. This feature along with the ability to identify maxima of structural response quantities is useful in the post–processor for determining which constraints are tight.

Graphically, an optimization system has all the requirements of a standard FE program as well as some additional requirements. The most important feature of a structural optimization GUI is the ability to interpret the member sizes graphically. In the programs developed here, it is possible to do this through setting the line color and thickness proportional to the member sizes. Each can be done independently so one can select color variations, thickness variations, both, or neither. The color scale was selected as a simple gradient from red, green and lastly blue in decreasing order. Thus, a member drawn in red would represent a large section and one drawn in blue would indicate a small size. This feature is not often seen in typical FE programs.

For simplicity, the GUI displays concentrated force loads as lines without arrows, and concentrated moments as well as supports are not shown at all. This is because these items are typically not editable in a post–processor and drawing them would require more programming effort than is warranted for a non–commercial program. Key–bindings are enabled so the user can advance through iterations quickly. An earlier version of the program also included a feature to automate the advancement after a short pause but this was rarely used and discarded in later versions. It is simple to include however, and thought that some users may find it useful thus recommended in a professional package.

Typically when viewing results for a given iteration, only a glance is sufficient and the numerical results are not needed. With this in mind, as each file is loaded, analysis is not performed unless the user requests it specifically through the menu. This allows the model to load much faster, particularly for large systems although details are available at any time.

Another special feature of the post–processor related specifically to structural optimization is the history plot. The program only stores the current model in memory but also keeps the volume of each model loaded. With this, a history plot can be seen instantly showing the volume versus the iteration. At any iteration, the display can be printed. This can be directed directly to the printer or saved as a postscript file.

Although there is a FE analysis tool included in the GUI, the pre–processor features are limited. Model generation was done outside of the system using other tools but ultimately this should be included in the main package. Currently, model generation is done by other programs written explicitly for that task but it is a trivial matter to include this in the main package. Additional features such as a display of the deformed shape might also be useful.

CHAPTER 4
EXAMPLES

In the following sections, several examples are presented. Results are shown graphically in the form of a volume versus iteration plot as well as a diagram of the final structure where lines are drawn with a thickness relative to the member size. The specific details such as the the structural dimensions and final results are found in Appendix C in the form of an input file along with a description of the input file format. The name of the file is constructed by concatenating the basename, the iteration number, and the extension "osi" all delimited with a dot. Units are not discussed in any of the following because these represent illustrative examples and thus the units are not significant. For all examples, the numerical value for Young's Modulus (E) is 29,000 and the initial design begins with some arbitrary value for the member sizes, normally 1.0 for the area.

4.1 Truss Examples

One approach to structural optimization is to begin with an array of nodes and provide a member connecting every pair of nodes. The array is typically rectangular and in the case of collinear nodes, a single member connects only the nearest two nodes. It is clear that for any support conditions and loading, such a structure has a high degree of redundancy and in fact, some of the members may not be useful. However, the intention of starting with this array of nodes and arbitrary collection of members is to allow the optimization algorithm to determine which members are useful. Members which are not considered useful are identified by their very small cross sectional areas relative to the other members. This approach is a simple test of a structural optimization algorithm but also

interesting in that it yields the possibility of arriving at a design that is not only optimal, but which the designer may not have initially considered.

The decision to discard members due to their insignificant contribution to the structure could easily be incorporated into the optimization system but this was not done in this work. This approach is shown here as only an example, thus for the relatively small number of uses, it was not considered important to include. In this work, when it is thought that members could be removed, a script was run which identifies the smallest member and then removes all within a specified range of that size, typically 5%. If this step were to be included in the optimization system, one would also need to check for structural stability, which has already been investigated by other researchers.

An example of this many–bar truss is seen in Figure 3. For this case, the boundary conditions are pin–supports for the three nodes at the right, thus the structure resembles a simple cantilever. There is a single concentrated load at the free end of the cantilever, at mid–height and the structure has 74 members and 15 nodes. The objective here is to minimize the total weight of the structure and the constraints require that the displacement of all members is within the range

$$-0.001 \leq \Delta \leq 0.001 \tag{46}$$

Because equation (46) shows an upper and lower limit on the displacement of each member, there are 148 structural constraints. For this model, the step size limits in equation (43) give an additional 148 constraints resulting in a total of 296 constraints for this model.

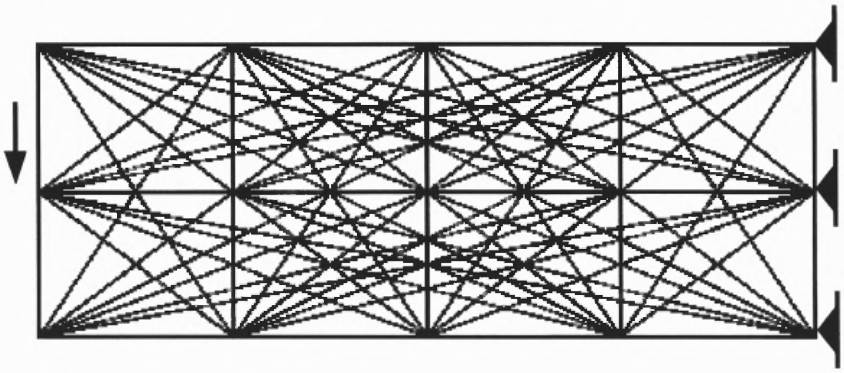

Figure 3 Multi–bar Truss

After 30 iterations, a simple scan of the member sizes was performed and all bars that were found with an area within 5% of the minimum bar area were removed from the structure. The resulting structure was optimized with the same constraints for an additional 40 iterations to yield the structure seen in Figure 4. The entire history of the volume versus the iteration can be seen for the entire procedure in Figure 5 where a jump in volume is seen between iteration 30 of the initial structure and iteration 1 of the reduced structure because of the elements removed.

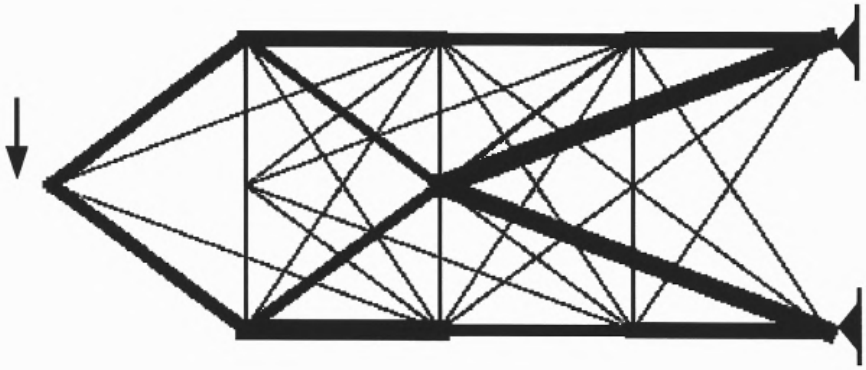

Figure 4 Optimal Solution t35s.040.osi

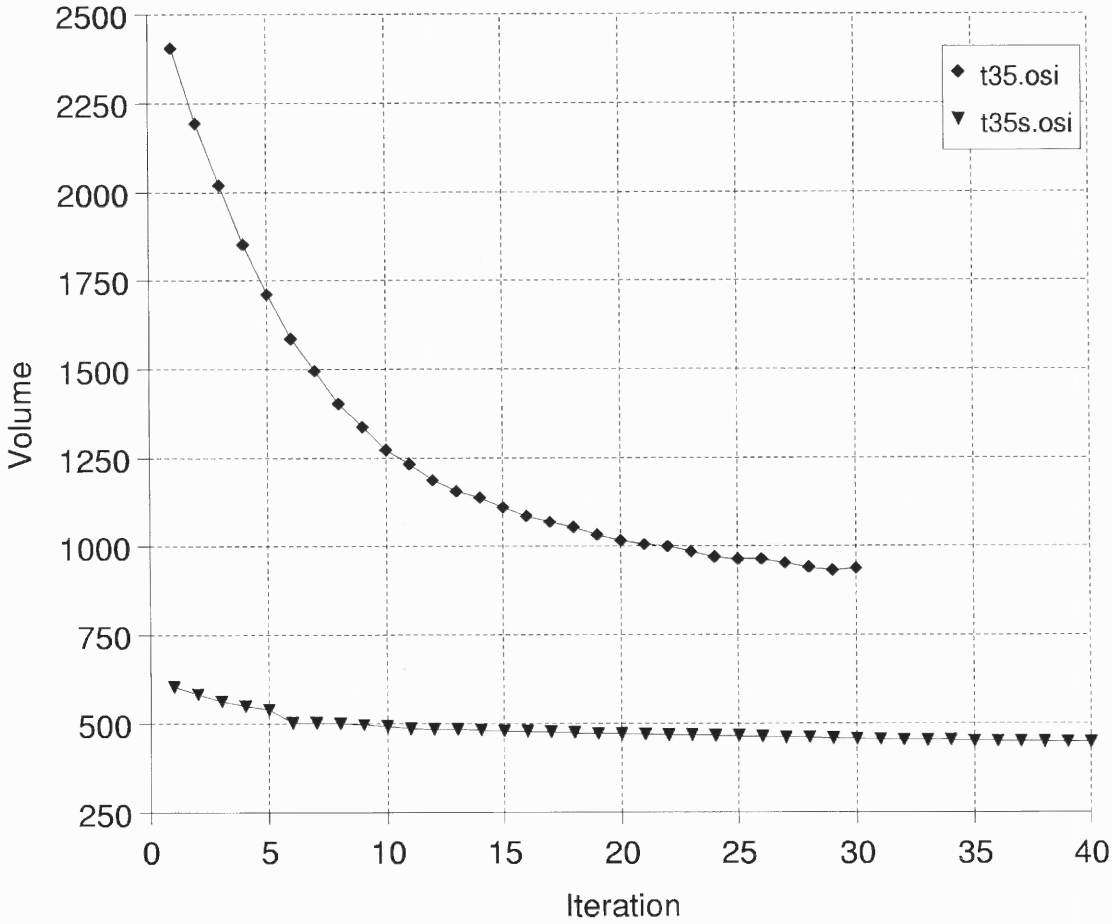

Figure 5 Iteration History for t35.osi

Another many–bar truss shown in Figure 6 is used to investigate the effect of changing the number of LP iterations. This example resembles a bridge in that it has two pin supports at the lower corners and a downward load mid–span. There are 61 bars and 14 nodes in this structure. The constraints are symmetric upper and lower limits on the set of member displacements. This is similar to equation (46) except the numerical value is changed to 0.488E–6. Because the number of bars is different, there are now 122 structural constraints and 122 step size constraints for a total of 244 constraints on the

system. This was selected over the previous example for this step not because there are fewer constraints but because the optimized structure after 30 iterations, seen in Figure 7, more clearly suggests the final result and is arrived at without the procedure of removing small bars noted earlier. By not removing the small bars, a more meaningful comparison can be made because the only variable is the number of LP iterations.

Obviously, fewer LP iterations are desired because this would provide faster execution times but one might then be concerned with the accuracy of the solution at a given step and the stability of the overall method. A note here should be made that the number of SLP iterations is different than the number of LP iterations and it is the latter that is of concern here. Figure 8 shows that for this example, there is little gain in using more than five LP iterations. This shows more clearly that, as mentioned earlier, the exact solution of the LP is not required again emphasizing the usefulness of interior point methods where one can prematurely halt the LP solver.

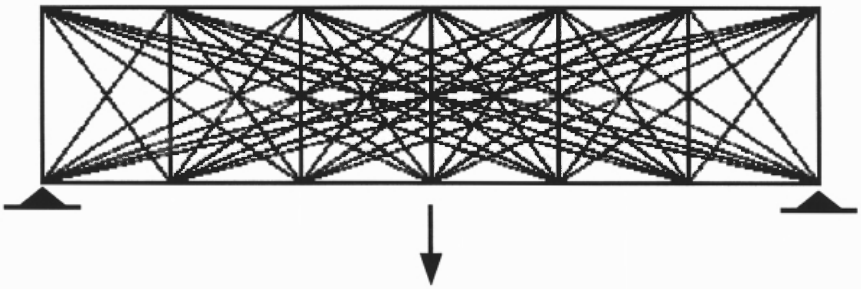

Figure 6 Multi–bar Truss t16.osi

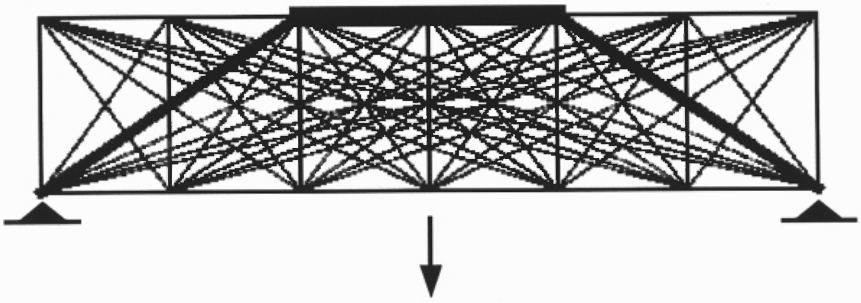

Figure 7 Optimal Solution t16.030.osi

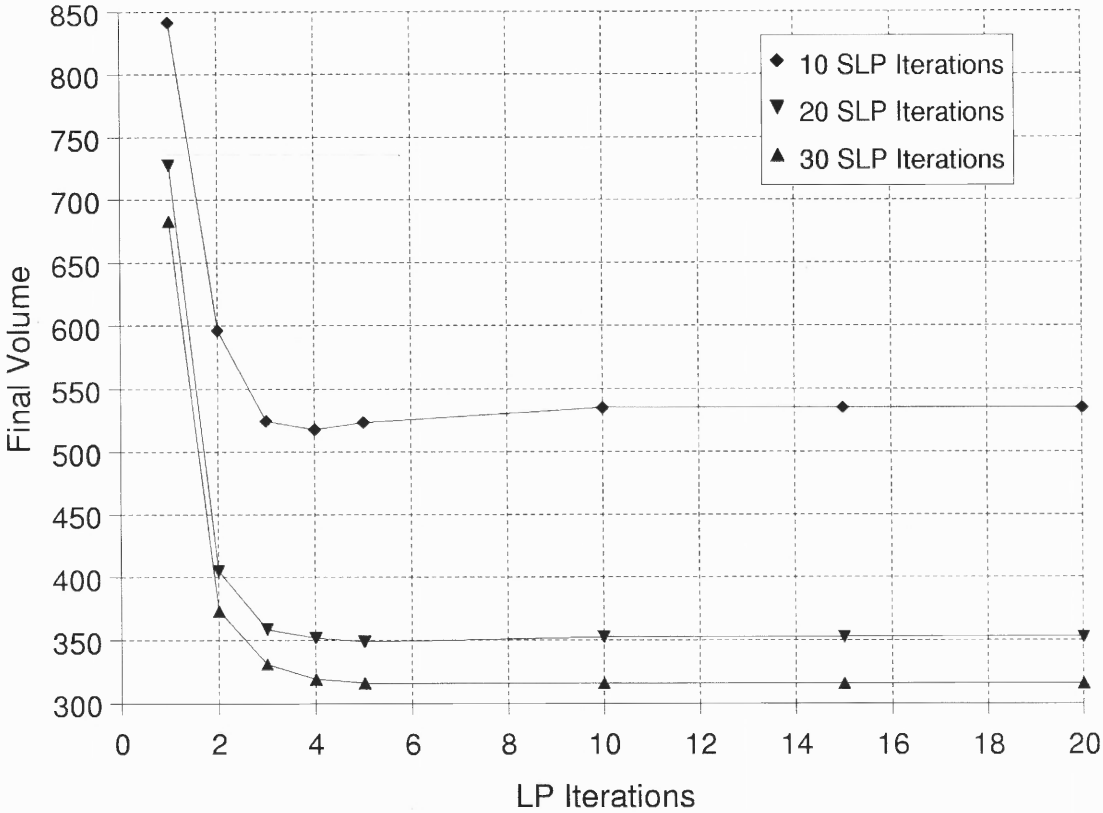

Figure 8 Comparison Using Different Number of LP Iterations

A different style of truss examined is an X–braced simply supported structure with 51 bars, 22 nodes, a pin support at each side, and a single concentrated load at mid–span. The optimal solution after 30 iterations using only the allowable member displacement constraints in equation (46), is seen in Figure 9 (xt210.030.osi). If the 102 allowable member displacement constraints are replaced by the same number of equivalent allowable member stress constraints, the same solution arises (xt213.030.osi). Equivalent replacement is done according to equation (16) and the optimal design is seen in Figure 10 which shows no apparent difference from the allowable displacement constraints. As noted earlier, the formulation for allowable member displacemtent is more compact. Because the solutions are equivalent, it is recommended to transform allowable stress constraints to allowable member displacement before the optimization begins.

It is interesting to note that if these 102 structural constraints are replaced by a single joint displacement constraint, a similar solution arises. The single joint displacement constraint requires that the vertical displacement at the point of loading is limited to the actual displacement of that same node in the previous allowable–displacement example (–0.0251 units). Numerically, there is little difference (xt211.030.osi) and the solution shown in Figure 11 appears very similar.

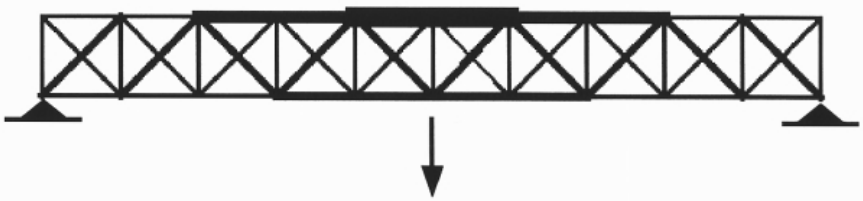

Figure 9 Optimal Solution xt210.030.osi

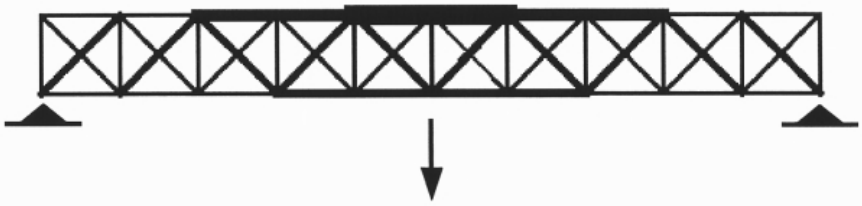

Figure 10 Optimal Solution xt213.030.osi

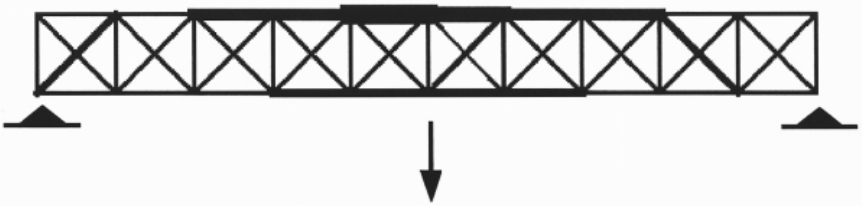

Figure 11 Optimal Solution xt211.030.osi

The structures in Figures 9 through 11 all show very similar results but a better comparison is seen in Figure 12, which shows the volume history for all three constraint condidtions. The difference between the allowable displacement and allowable stress constraints is so small that the graph shows nearly a single line for the two and the data points are nearly coincident. The allowable displacement condition shows convergence to the same value although it starts at a different point because of the initial scaling mentioned in equation (45).

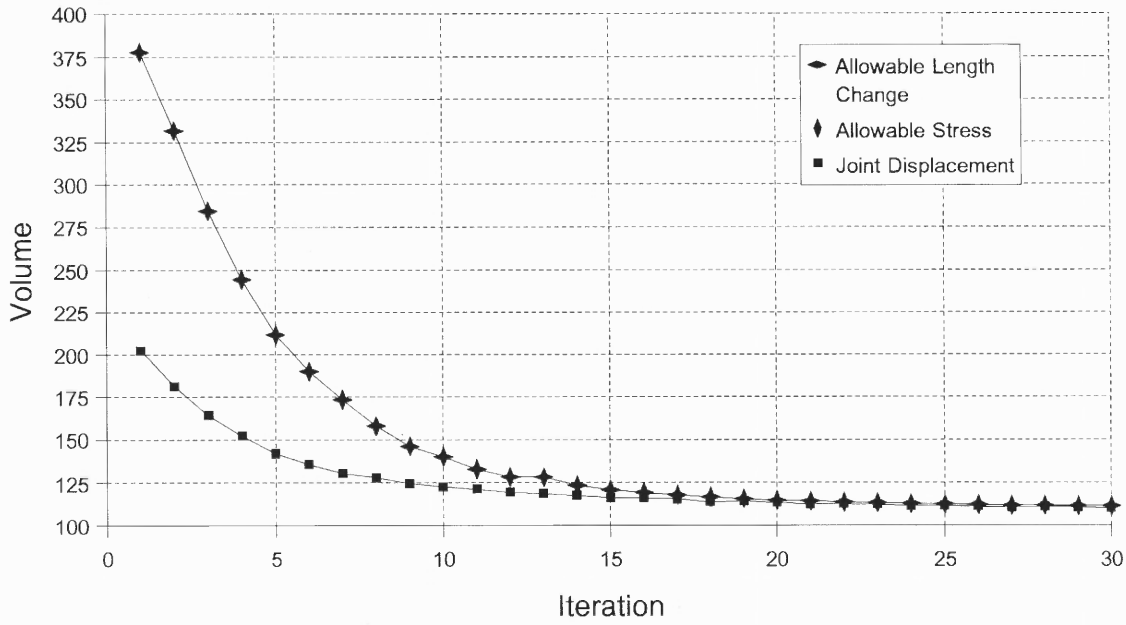

Figure 12 Comparison of Volume History Using Similar Constraints

The most interesting point of this problem however is as follows. One can note that in each panel, there is one diagonal member in tension an one in compression. It was initially thought that one could create a tension member preference by imposing more restrictive constraints on the compressive displacement (member shortening) or, conversely, applying less stringent constraints to the tension displacement (member elongation). For the latter, the symmetric constraints of equation (46) were changed to the unsymmetric case

$$-0.001 \leq \Delta \leq 0.1 \tag{47}$$

Interestingly, it was found that the structure cannot take advantage of the less stringent constraints on the tension members because the displacement of each tension member is related to that of a compression member in the same panel. In other words, if the displacement limit on a tension member were to be realized, the displacement limit on

the compression member in that same panel would be exceeded. This could have been determined before–hand through proper analysis but it is encouraging to note that the algorithm identifies such a behavior, possibly saving a designer much time.

The anticipated compression–only and tension–only solutions are seen in Figures 13 and 14 respectively. These structures were generated by manually removing all diagonals in compression or tension respectively, and then performing 30 iterations with the constraints in equation (46). All constraints are tight in the expected solution but it is interesting to note that both of these are less optimal than the initial solution (xt210.030.osi) in which all diagonals are present and some constraints are not tight. Figure 15 shows that the algorithm finds an optimal that is better with both tension and compression diagonal members, thus is performing correctly in not removing a diagonal member from each panel. This is a good example of how an optimization system can be used as a design tool because it can identify a design that is not only optimal but also may not be obvious.

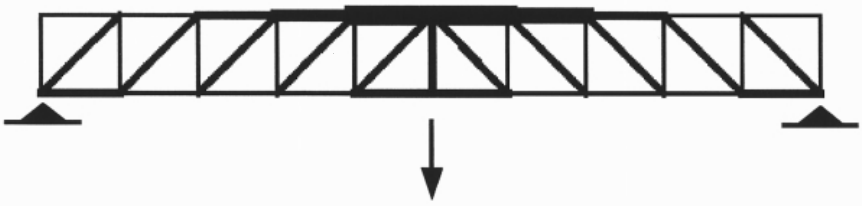

Figure 13 Expected Diagonal–Compression–Only Solution

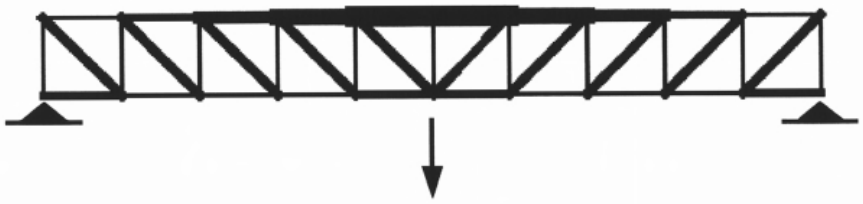

Figure 14 Expected Diagonal–Tension–Only Solution

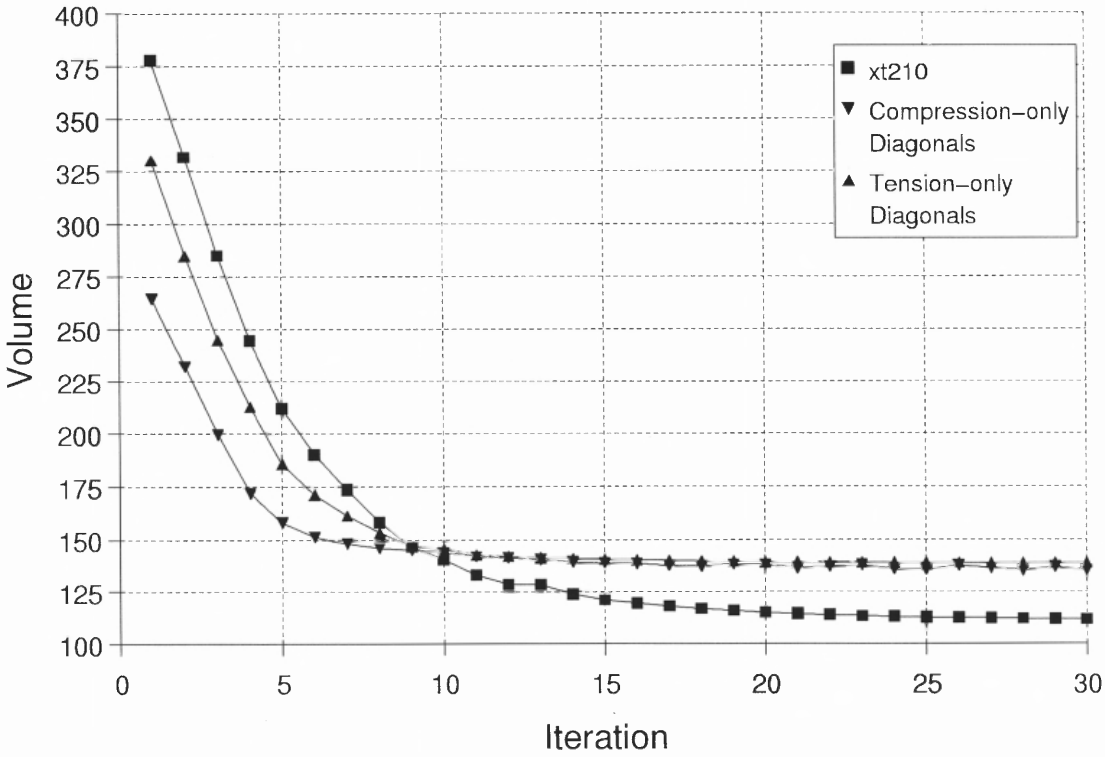

Figure 15 Optimal Solution with Compression and Tension Diagonals

4.2 Frame Examples

The simplest frame example discussed is a two–dimensional structure with two–elements of equal length, one beam and one column. There is one free node and a single force load applied in the horizontal direction. The only constraint is that the displacement of the free node in the direction of the load is limited to 0.1 units. This example is shown because it illustrates that the feasible design space can have local minima and, in this case, the algorithm succesfully locates the global minimum. The optimal solution (f02.020.osi) and the volume history are seen in Figures 16 and 17 respectively. For this case, all material is being distributed to the beam element and the load is being carried axially by the beam.

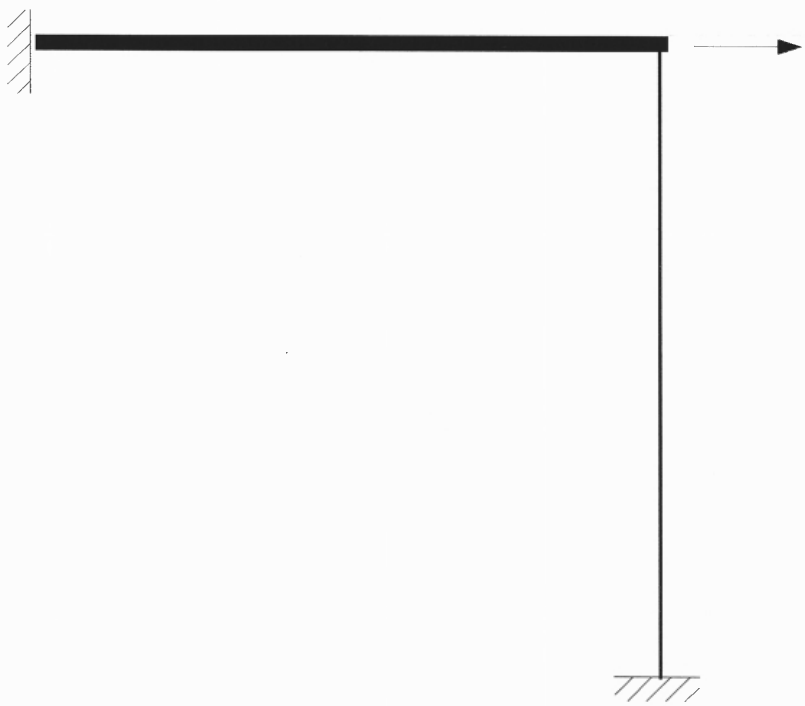

Figure 16 Two–element Frame – Beam Solution f02.020.osi

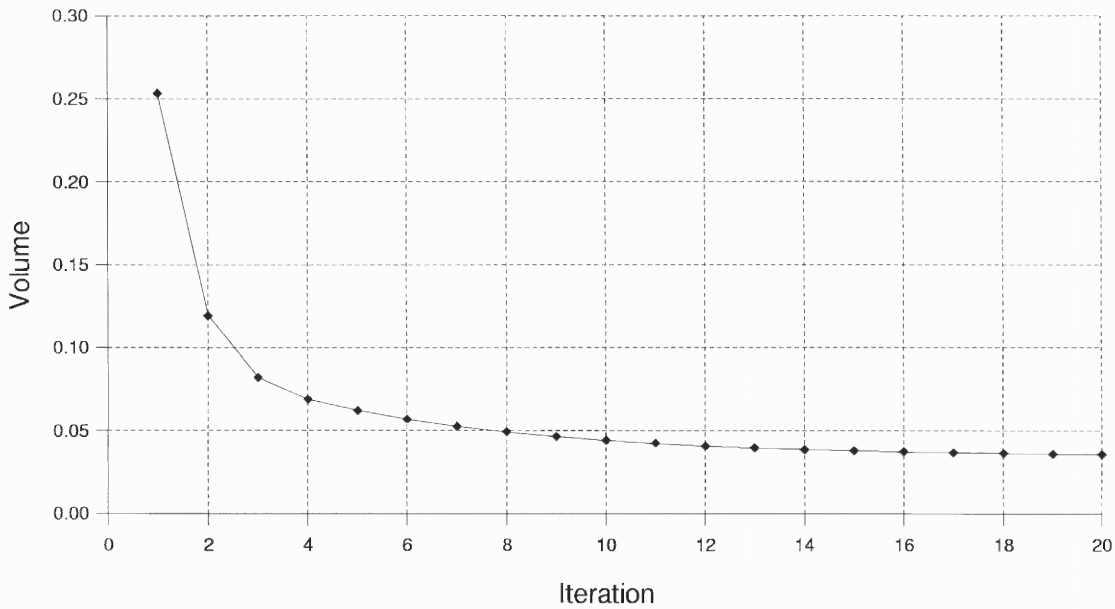

Figure 17 Volume History for Beam Solution

One may have expected that the beam–solution would be the optimal design based on common sense but technically this is only correct for typical values the constants c_0 and c_1 from equation (22). Academically, there are values of these constants that will cause the column–solution (i.e., all material is distributed to the column element) to be preferred and the load will be supported by a single element acting in flexure. Assuming $c_1 = 2$ for both cases, the exact value of c_0 for which the column–solution becomes preferred over the beam–solution can be found by setting the displacements equal for the individual cases and using the A to I relation given in equation (22).

$$\begin{aligned} \frac{PL^3}{EI} &= \frac{PL}{AE} \\ L^2 &= c_0 A \\ c_0 &= \frac{L^2}{A} \end{aligned} \tag{48}$$

The previous example used $c_0 = 5$ but to satisfy equation (48), c_0 must be greater than 10,000. Although this is an unrealistic value for most cases, it is a good test of the algorithm. Figure 18 shows that the expected solution is found but the history seen in Figure 19 shows an interesting occurrence. Occasionally, as in this problem, the volume history shows an initial increase before a later convergence. This is interesting because what is happening is that the algorithm is beginning in the vicinity of a local minimum but to reach the global minimum (the column–solution), the volume must first increase to get away from the local minimum.

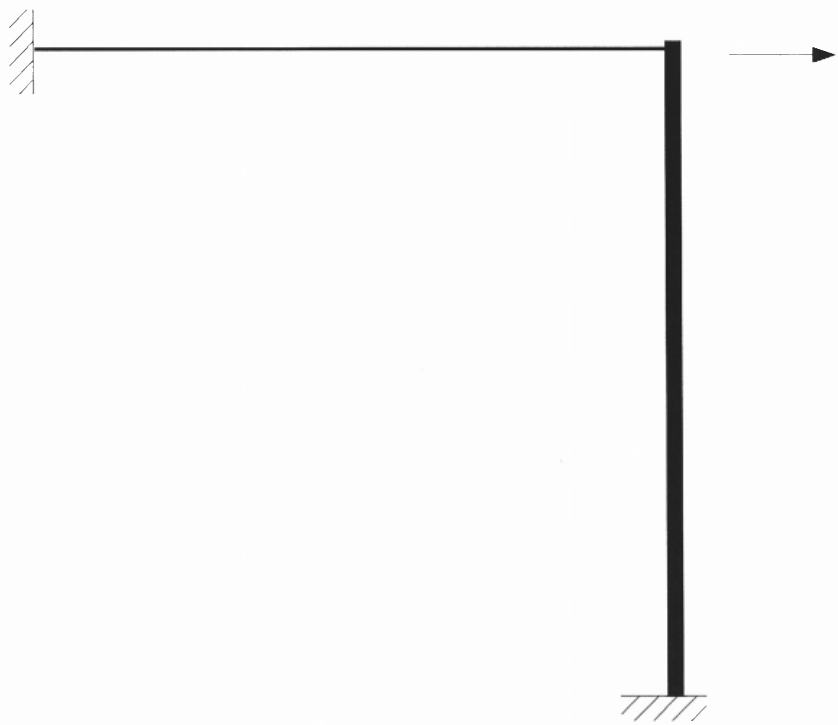

Figure 18 Two–element Frame – Column Solution f02a.030.osi

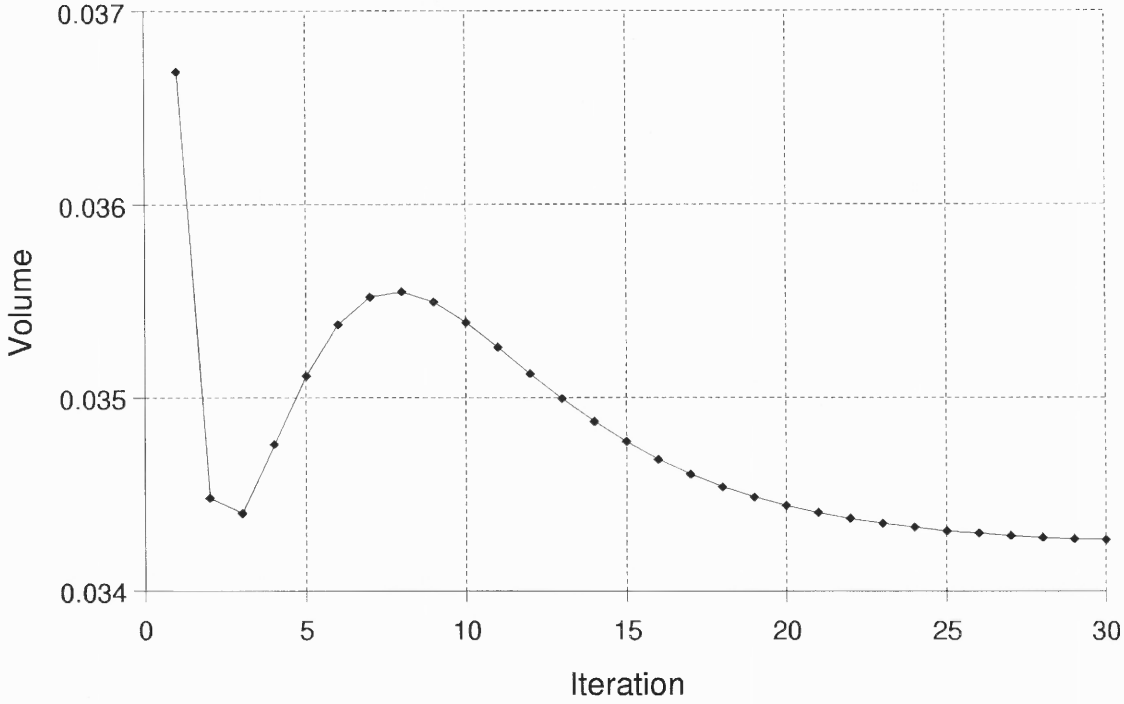

Figure 19 Volume History for Column Solution

This behavior is also seen more clearly in Figure 20 which shows the volume data from Figure 19 plotted on the ordinate versus the value of the beam element's area A_1 on the abscissa. Each point on this volume curve represents one iteration and because the change in area A_1 is monotonic, the points are in sequence. However, note that the first iteration is located with the largest A_1 and thus, the sequence moves to the left as the iteration number increases. Also seen on this graph is the feasible region for this problem which was generated by selecting a value for A_1, solving for the value A_2 that would satisfy the single displacement constraint, and then computing the volume.

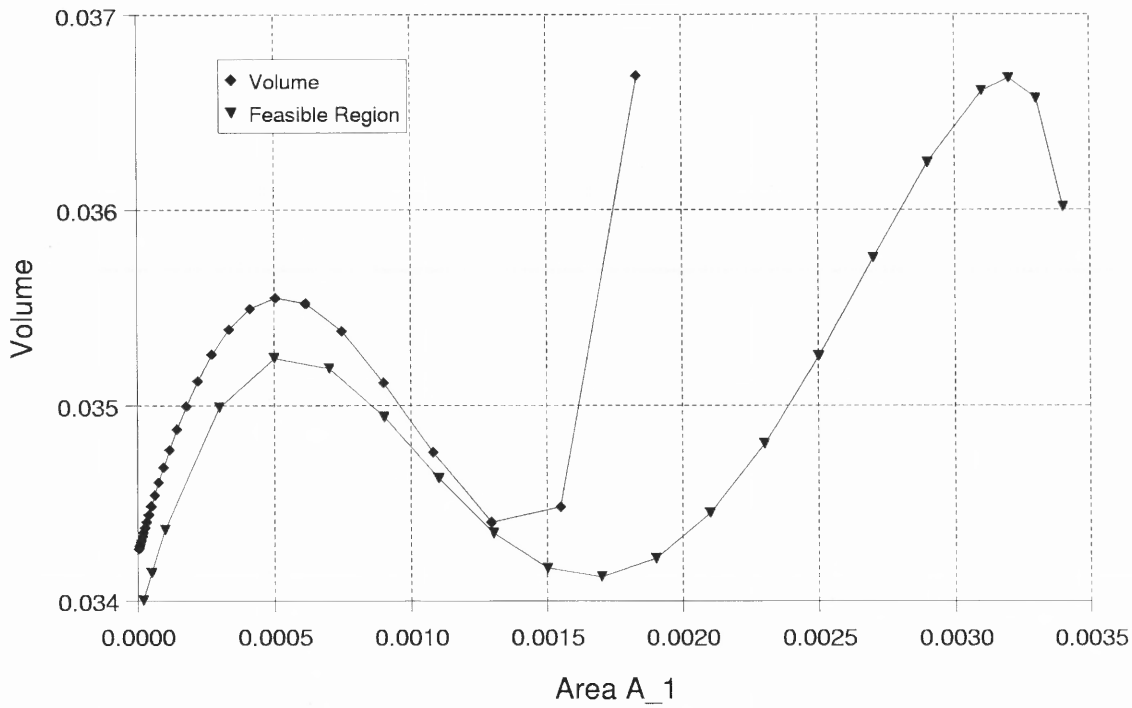

Figure 20 Location of Global Minimum

Note that in Figure 20, the volume curve has the same shape as in Figure 19 but is mirrored because the abscissa has changed. Also note that the volume curve is above the feasible region in accordance with equation (43). Even though this is a simple problem, it is worth noting because often the literature dismisses SLP because it doesn't guarantee that a global minimum will be found. Of course, this example still does not guarantee that a global minimum will be found in all cases but it does illustrate that it can occur. Further, it is well worth noting that even if a local rather than global minimum is found, it is still a form of optimal and likely better than the initial design. In short, an algorithm should not be dismissed solely because it doesn't guarantee a global minimum.

A more realistic frame example is a four–story, two–bay frame shown in Figure 21 which show the solution after 30 iterations. The applied loads are a concentrated

downward load at each node in addition to an applied moment at the exterior nodes representing fixed–end moments associated with a distributed load on a floor. The constraints are on the member displacement and rotation, all of which are found in the member displacement matrix described in equations (3) and (22). The values used are

$$\begin{bmatrix} -0.00001 \\ -0.0001 \\ -0.0001 \end{bmatrix} \leq \Delta_i \leq \begin{bmatrix} 0.00001 \\ 0.0001 \\ 0.0001 \end{bmatrix} \tag{49}$$

Similar to the truss example discussed earlier, the number of LP iterations was investigated for this example and again, it was found that fewer than expected LP iterations are required. This can be seen in Figure 22 which shows that for some cases, less iterations actually provide a slight improvement.

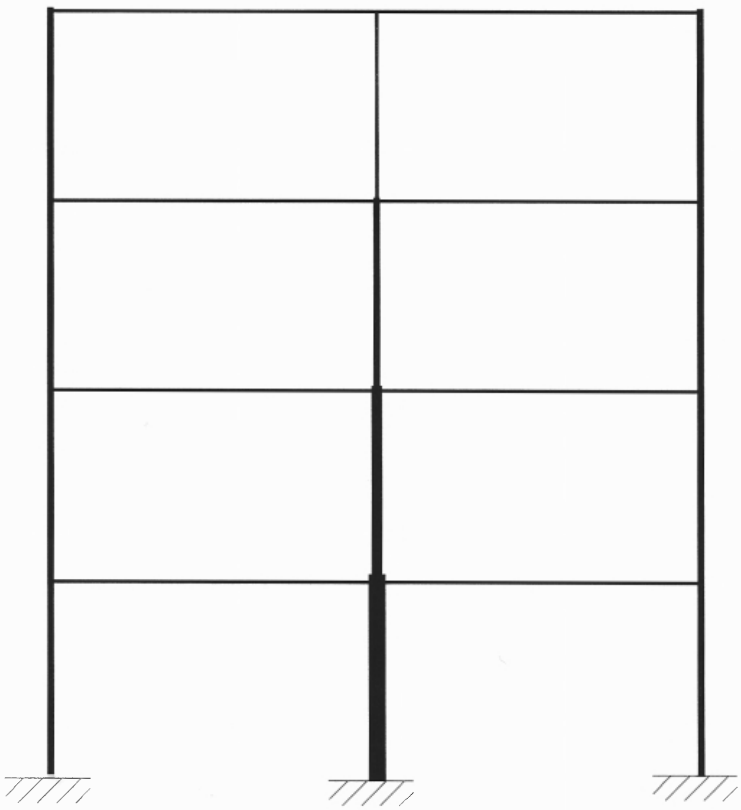

Figure 21 Optimal Solution of Two–bay Four–story Frame f242.030.osi

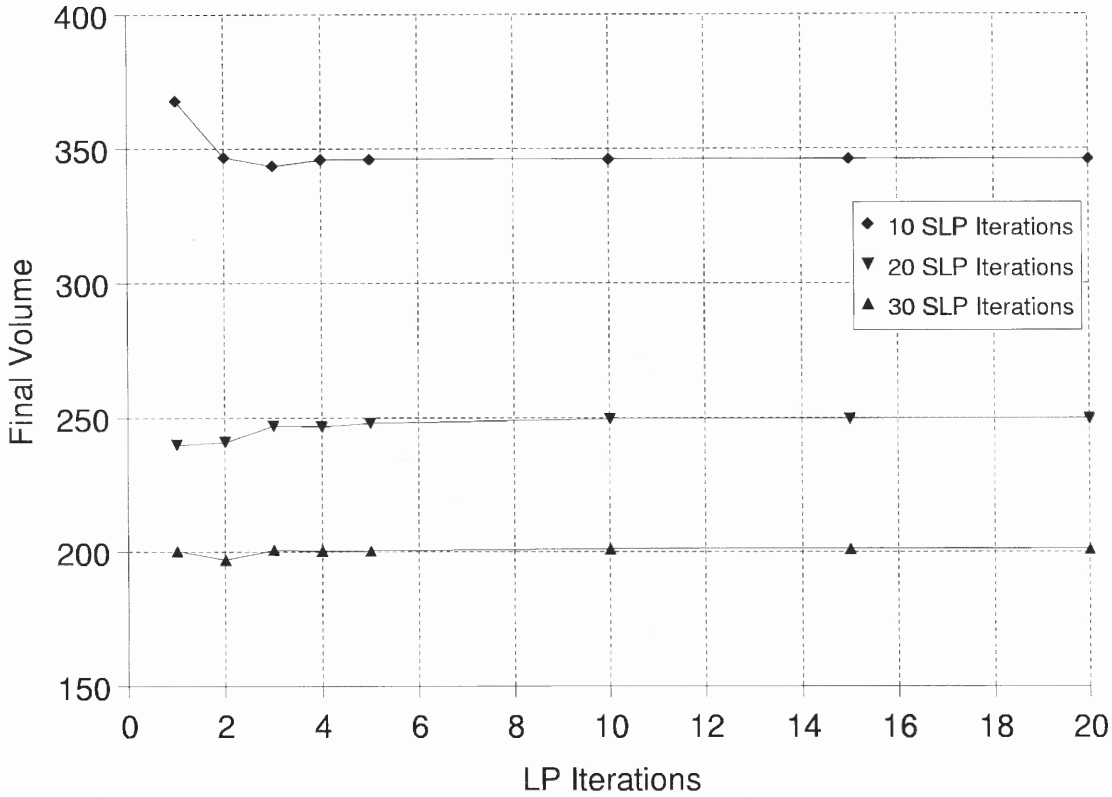

Figure 22 Comparison Using Different Number of LP Iterations

It was found that values of the step size parameter α in equation (43) greater than 0.3 caused variations from one iteration to the next and the resulting volume history is not monotonic. Figure 23 shows the volume history for this example using different values of α. From a practical point of view, there is little difference after 30 iterations but it is worth noting that the steps can vary greatly with large α and for frames, values less than 0.5 are recommended.

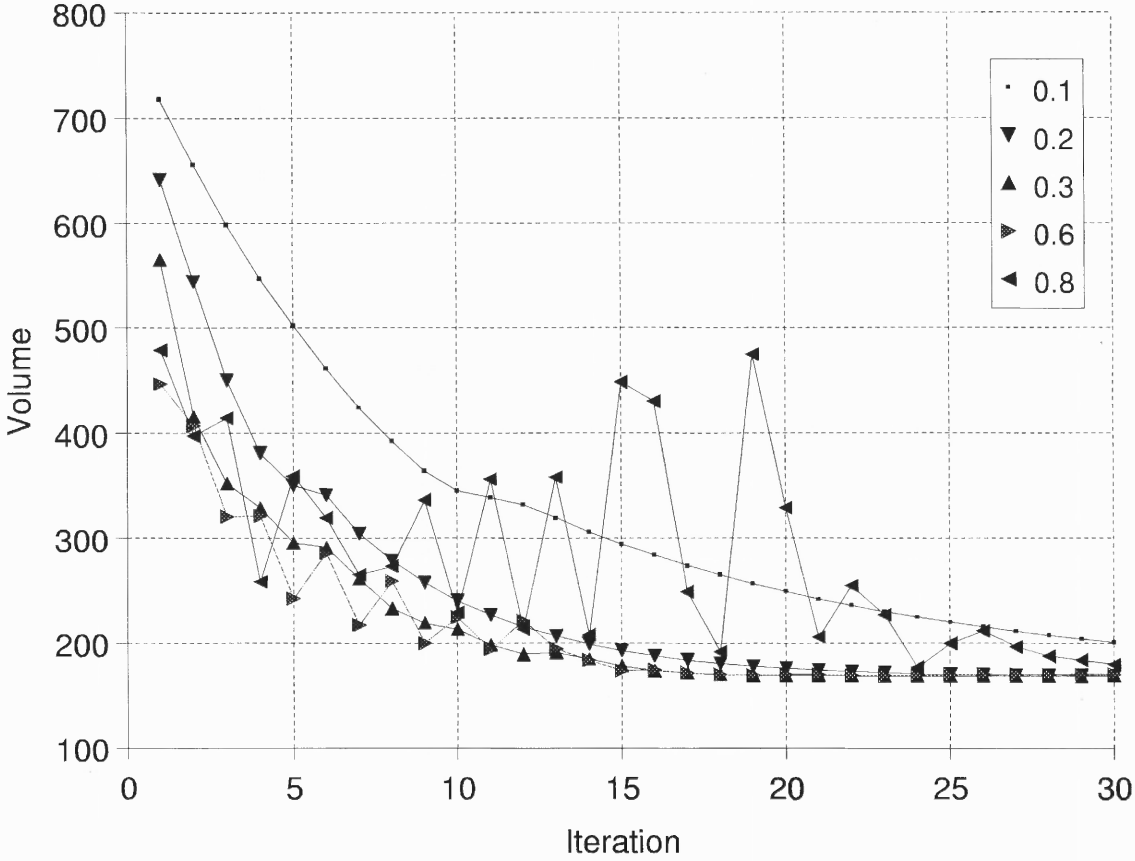

Figure 23 Comparison Using Different Step Size α

The nonlinear terms mentioned in equation (30) were included in this example but as seen in Figure 24 there is rarely an advantage and most times, the linear approximation is superior. Sometimes, improvement was noted for large step size parameter α, but was not seen consistently and this is not recommended. It is thought that this is because the degree of nonlinearity may be higher for the constraint functions, which were not approximated with this method, than for the stiffness matrix.

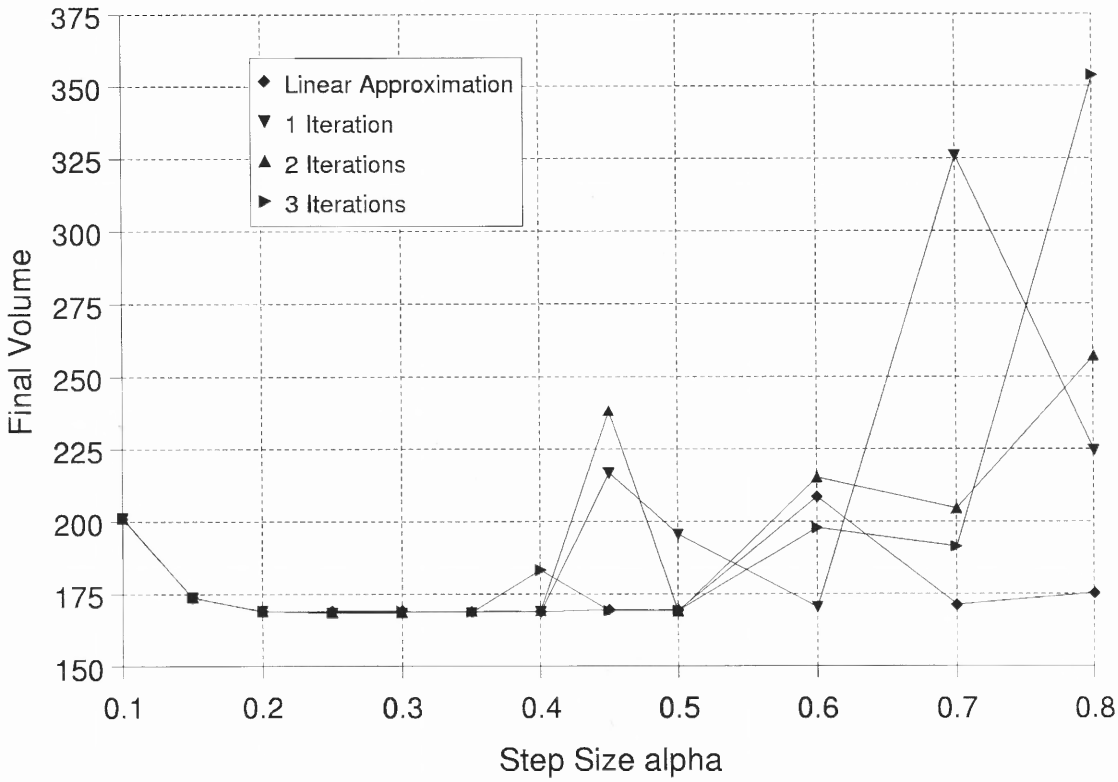

Figure 24 Comparison Using Nonlinear Terms

The first frame buckling problem to be discussed is the simple cantilever with a compressive axial load. In this case, 10 elements are used along the length of the cantilever and the problem is to find the optimal distribution of material that will support the load. The solution after 30 iterations is seen in Figure 25 and the area distribution is plotted in Figure 26. The volume history in Figure 27 shows an apparently good convergence but it does not show clearly that the relative distribution is still changing. This can be seen in Figure 28 which shows the area over the length at six different points in the iteration history. From this, one can see that the solution is very nearly approximated at iteration 20 but then continues to change, sometimes creating a shape with "pockets".

Figure 25 Optimal Solution for Cantilever with Axial Load col10.030.osi

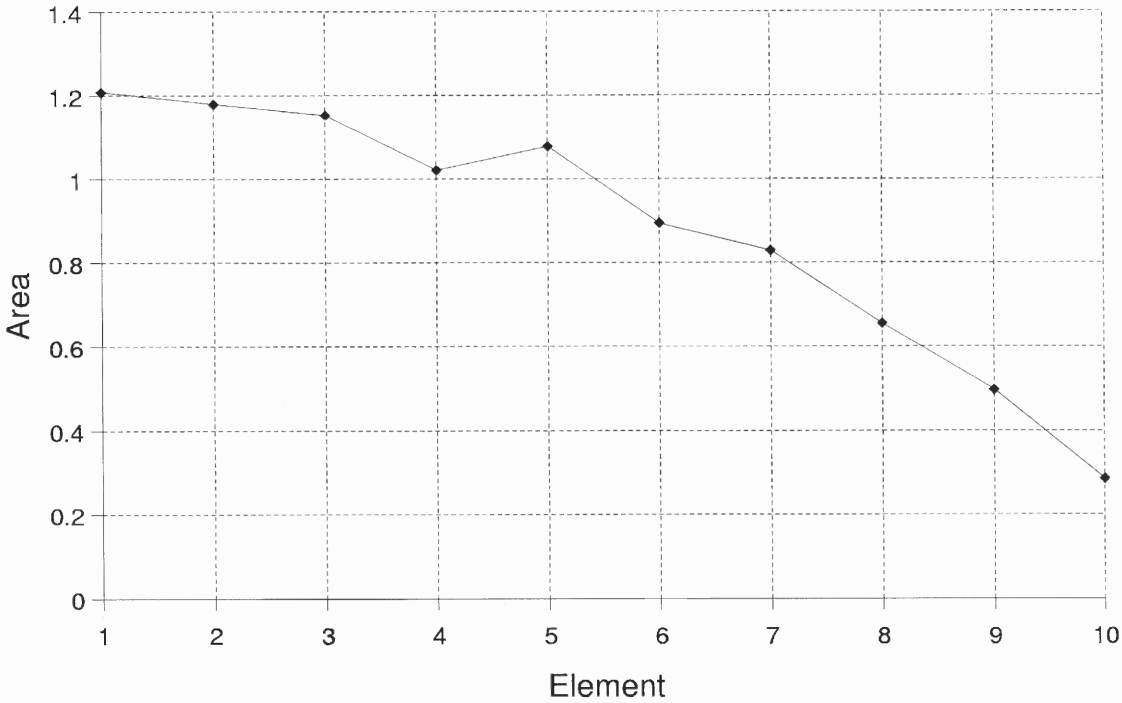

Figure 26 Area Distribution for col10.osi

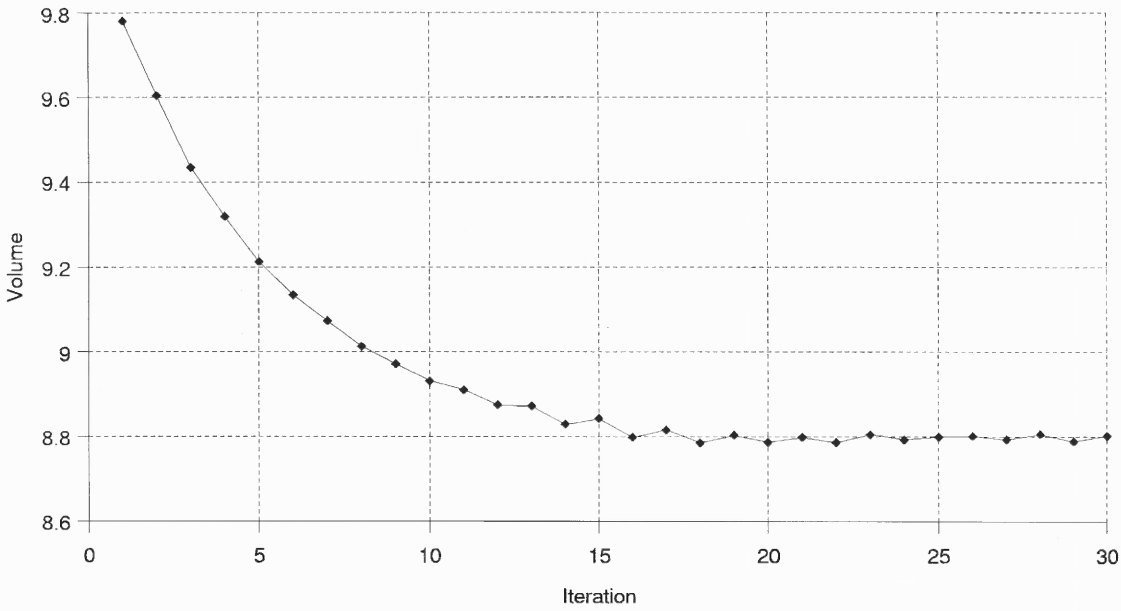

Figure 27 Volume History for col10.osi

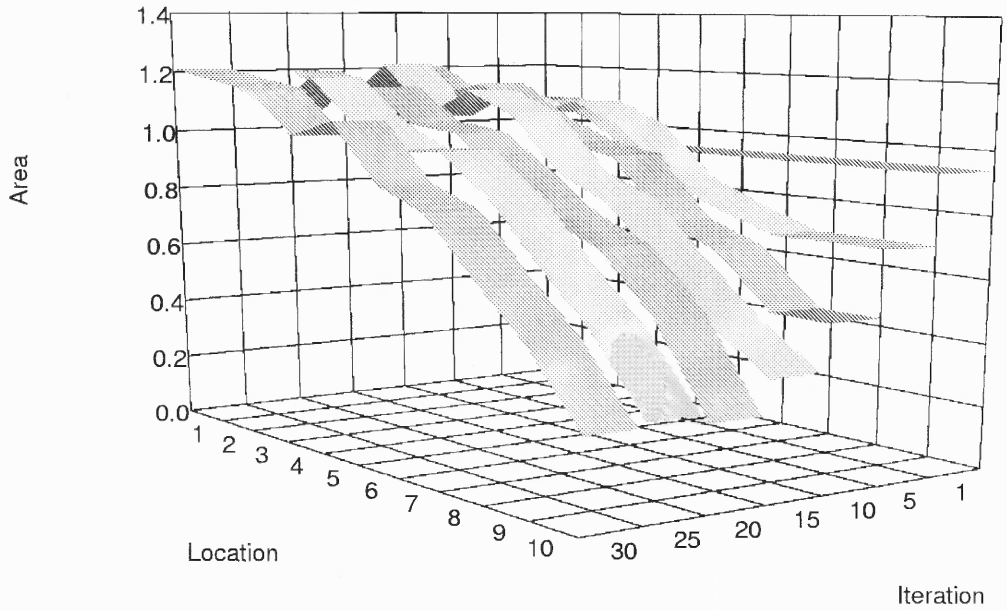

Figure 28 Area Distribution History for col10.osi

A one–story, one–bay frame was also used as a test problem for buckling. In this case, there is a downward axial load on each of the two free nodes and the columns are fixed at the base. It is interesting to note that the optimal solution in this case is dependent on the dimensions of the structure. A comparison is shown here for two structures, both with equal proportions and the same constants relating A to I in equation (23). The frame with the larger dimensions converges to the relatively equal member size solution seen in Figure 29 but the frame with the smaller dimensions converges to a column–solution seen in Figure 30 (col10s.030.osi). This transition from the distributed solution to the column solution was noted to be continious by examining a range of structures, all with the same proportions, between these two examples.

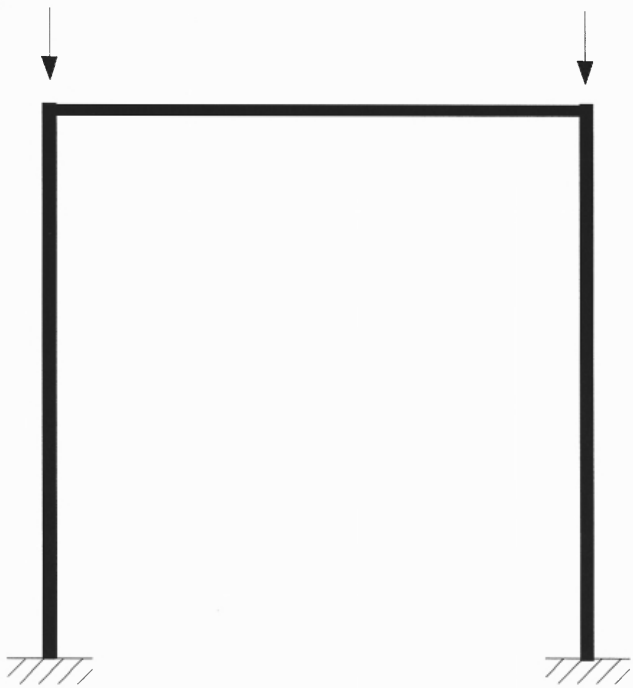

Figure 29 Frame Buckling Problem – Distributed Solution f11b.050.osi

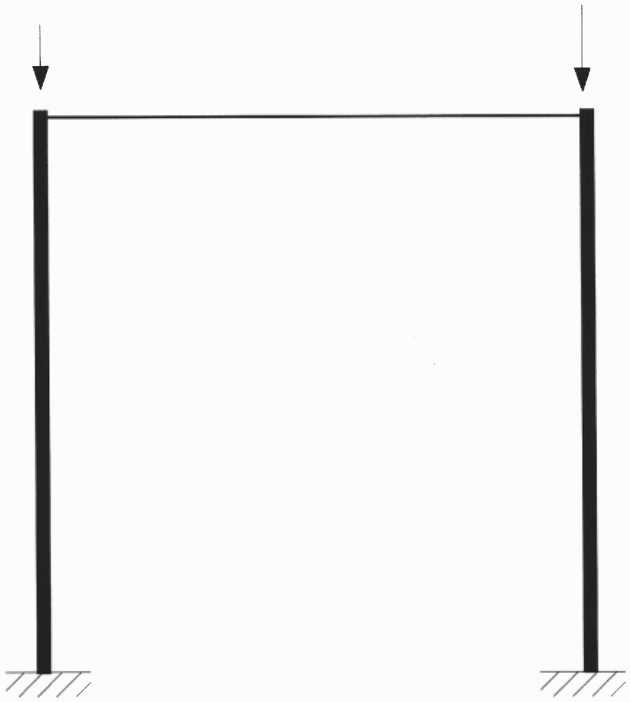

Figure 30 Frame Buckling Problem – Column Solution f11s.050.osi

It was commonly noticed in buckling problems such as this that there is not a monotonic approach to the optimal but rather a "zig–zag" as seen in the history Figure 31. This is due to the scaling mentioned in equation (45) and is accompanied by a change in the load factor seen in Figure 32. When dealing with buckling problems, scaling is done according to the load factor but this is not exact. This means that at different iterations, the load factor may be closer to one, although the program ensures that it is never less than one. For a given iteration, if the factor is closer to one, the volume will be less but it is not necessary to scale to exactly that point and this is the reason for the slight increase in volume seen at some iterations. Regardless, the general trend can clearly be seen in Figure 31 which shows an overall reduction in total structure volume.

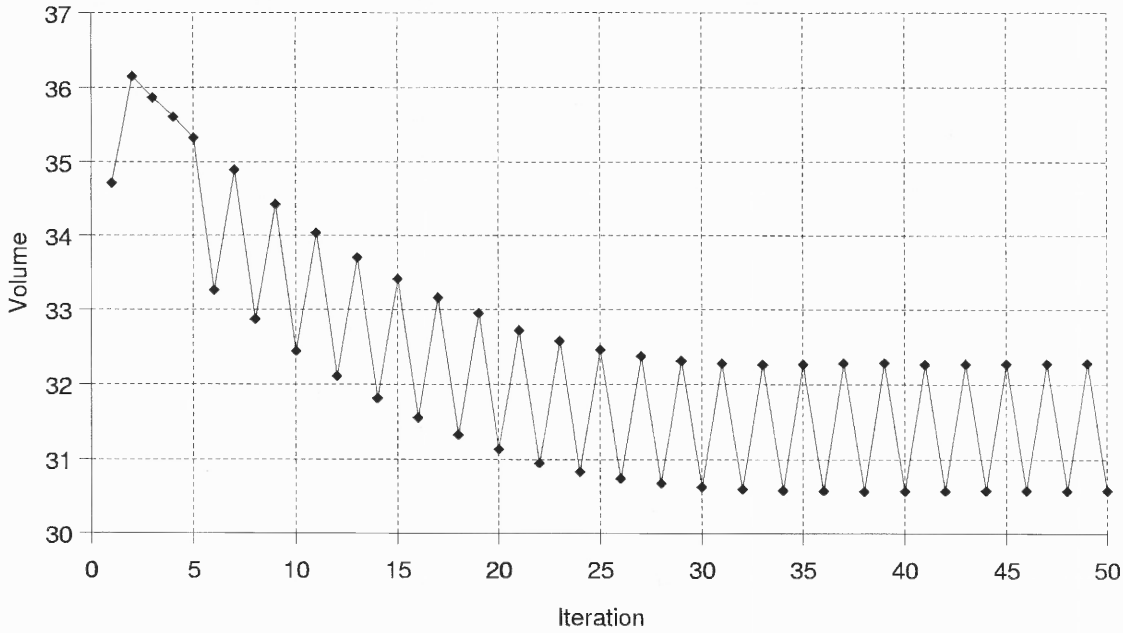

Figure 31 Volume History for Frame Buckling Problem

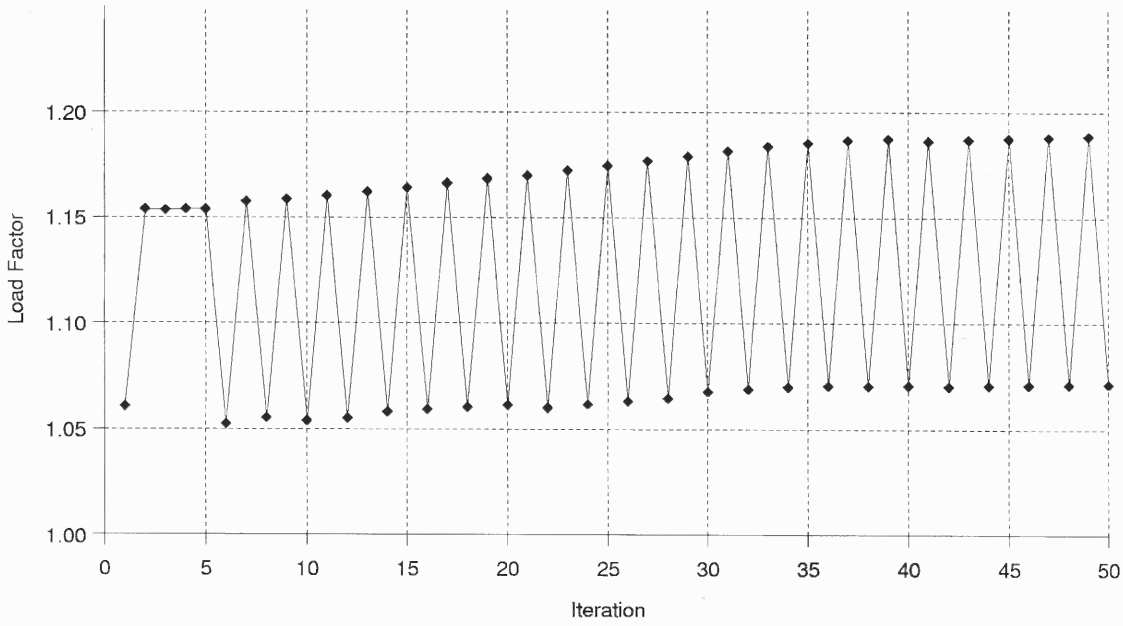

Figure 32 Load Factor History for Frame Buckling Problem

4.3 Plate Examples

As mentioned earlier, in this work plate structures are modeled with a grid of three–dimensional beam elements. For a plate that acts primarily in flexure, it is not uncommon to use beam elements with a 3x3 stiffness matrix, slightly modified from the two–dimensional beam element stiffness matrix in equation (20) by replacing the axial stiffness term with a torsional stiffness term. However, in this work it is desired to find the optimal distribution of material such that the plate will support some critical Euler–buckling load and thus it is necessary to include the axial terms as well. With this in mind, the three–dimensional beam elements are used instead of forming a custom element expressly for this purpose. The 6x6 element stiffness matrix was described in equation (26).

One of the more challenging problems in plate optimization is to find the optimal distribution of material that will support some critical buckling load. This problem has been addressed by several researchers [Prasad, 1989; Spillers, 1990] with varying results. In fact, one study concluded that the optimal solution is not even a smooth, continuous distribution of material but rather approaches a number of thin stiffeners.

Similar to the frame buckling example shown above, the optimal grid with respect to buckling was seen to be a function of the section properties. If one selects section properties to represent very high shear stiffness, the optimal material distribution is different than that seen if axial and flexural stiffness are favored. This can be seen in the two grids shown in Figures 33 and 34 where the first suggests very high shear stiffness. The volume history for these examples is shown in Figures 35 and 36 respectively.

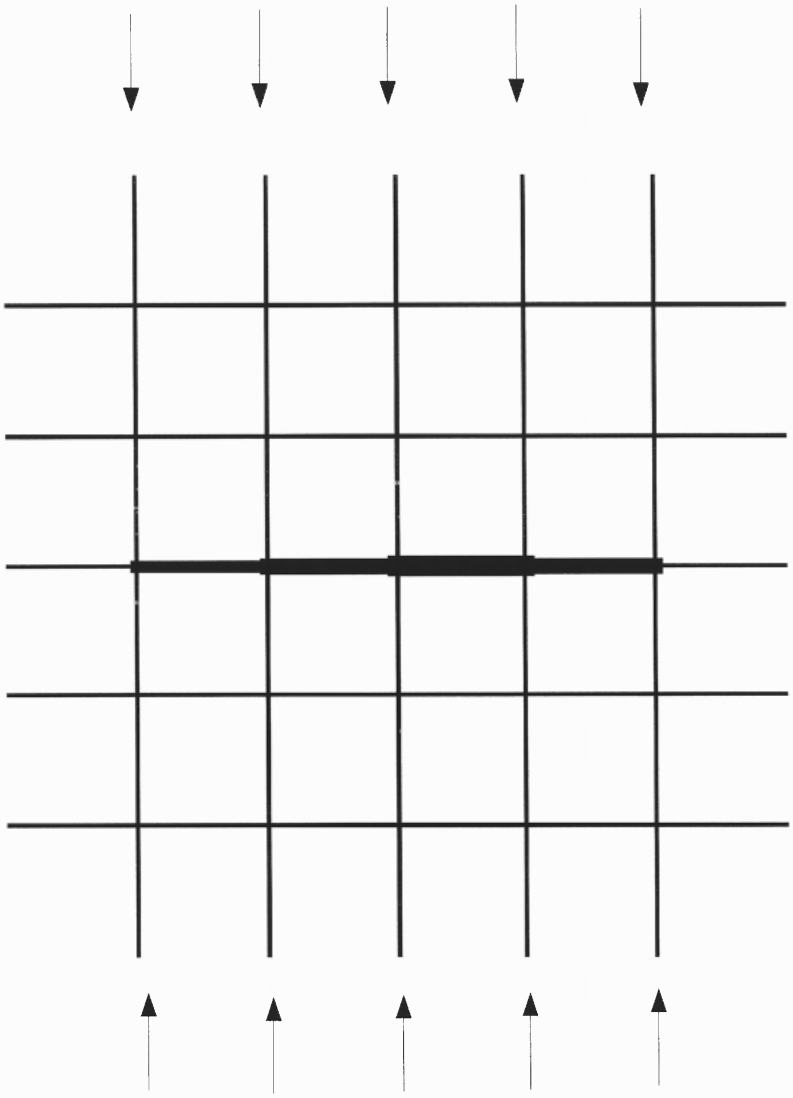

Figure 33 Optimal Solution for Grid Analogy g53.030.osi

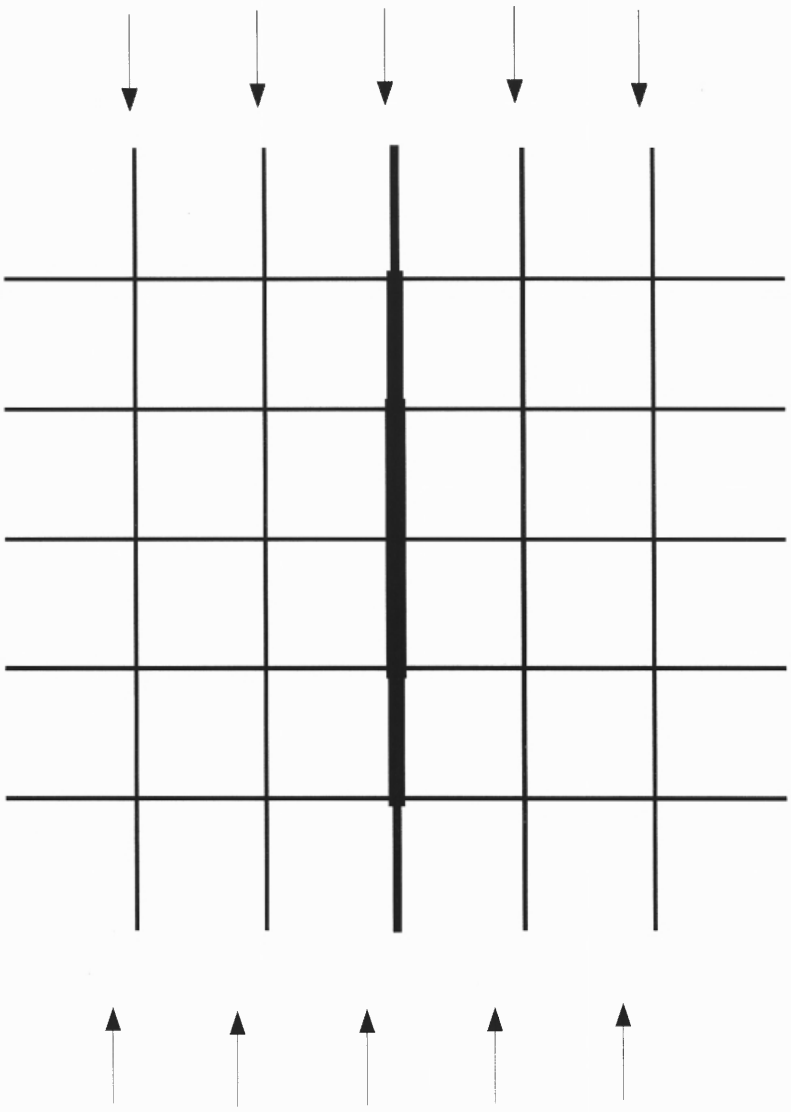

Figure 34 Optimal Solution for Grid Analogy g5.030.osi

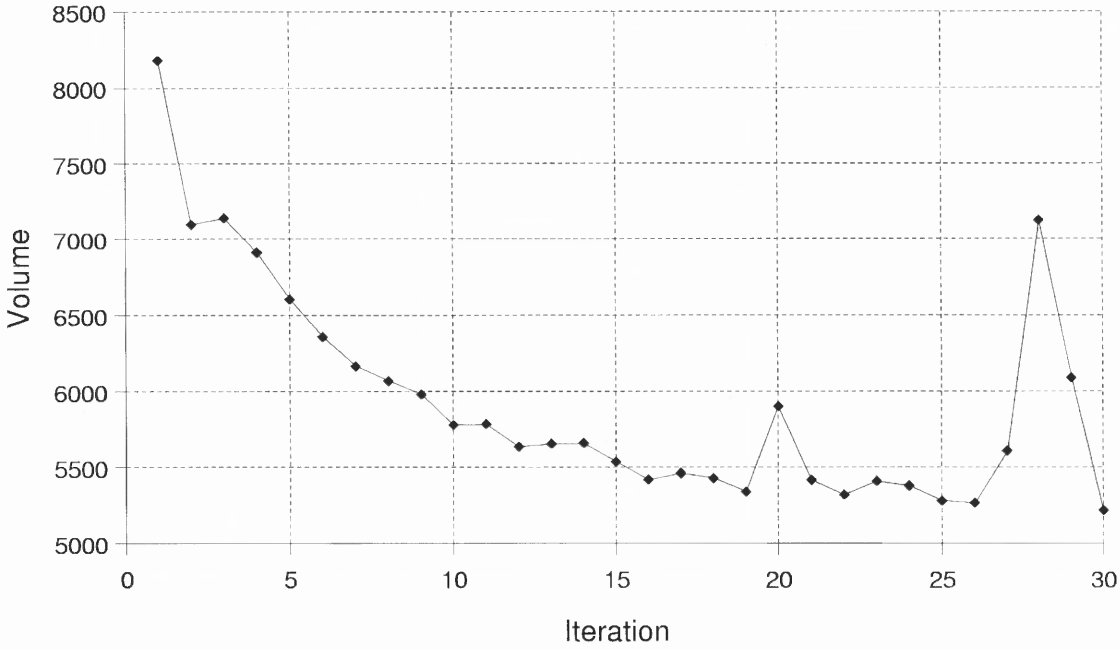

Figure 35 Volume History g53.osi

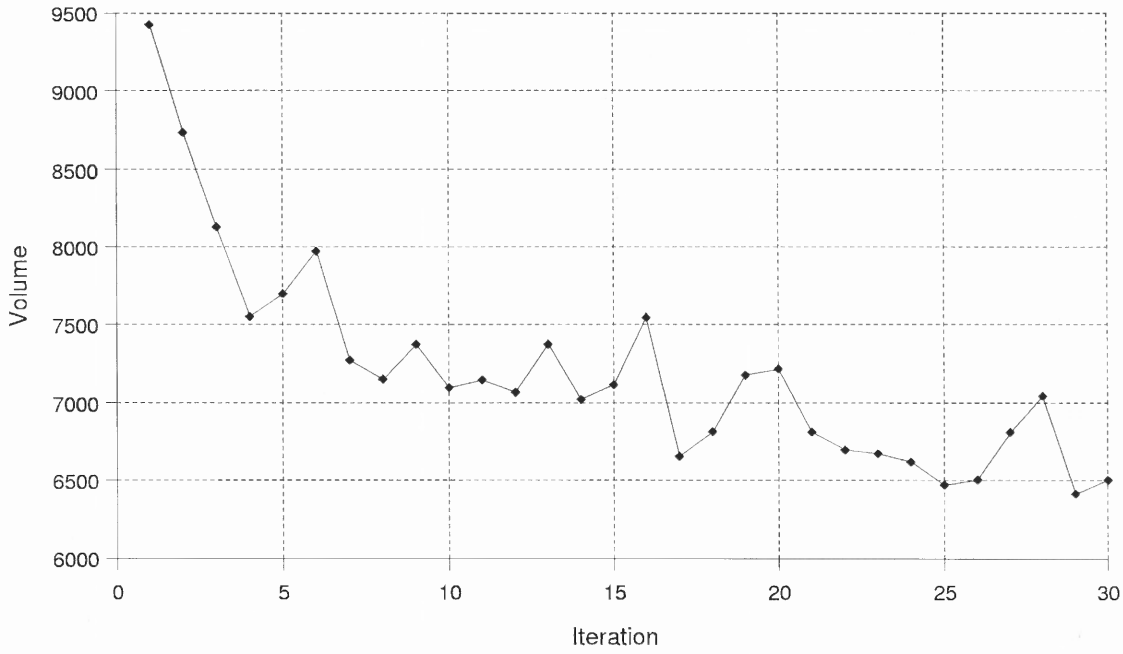

Figure 36 Volume History g5.osi

CHAPTER 5
SUMMARY AND CONCLUSIONS

This thesis has prototyped a computer system for structural optimization which uses the latest in linear programming techniques. It was shown that the developed system is able to handle various structure types as well as a wide variety of structural constraints. Further, sequential linear programming was shown to be a viable tool for a large-scale structural optimization system and the recent advances in linear programming are exploited for this purpose. The importance of a flexible, extensible approach was also shown through the development of a broad-based computer system.

The real test of the work done here involves its adoption by the engineering community to the extent that structural optimization can become as commonly used as the finite element method. In fact, the model for engineering design in the future has analysis augmented with optimization. Most of the analysis now done is concerned with designing something, analysis for its own sake is rare, and the underlying concern of engineering design is optimization. It is not difficult to predict this occurring in the near future. But a commercial version of a general engineering design (optimization) system will be both time-consuming and expensive.

5.1 Extensibility

There are many needed features not included in the software system discussed here: many element types, constraints, objective functions, etc. A practical approach to the development of a commercial software package would likely begin with a successful existing software system. The optimization would then sit on top of this system calling

the existing features as needed and supplying the new tools associated with optimization. In fact, some of this now exists in the finite element package ANSYS which has seriously limited optimization capabilities and is not written with structural engineering design in mind. ANSYS underscores the basic software need of shifting emphasis from analysis to design. The good part of this is the fact that hardware capabilities are improving by the minute.

5.2 Future Work

The most immediate work to be done has to do with a commercial working environment. We have contacted the engineering firm of Weidlinger Associates in the hope of working with them to develop a practical engineering tool but that has yet to develop. That practical tool will involve special features to help the engineering designer, probably in the form of a graphical user interface. Developing features like this is a labor–intensive process.

The most interesting technical issue to be resolved is the optimization routine. It is clear that the potential exists for a large–scale SLP solver to be used in optimization but there is as yet no experience with routines of this type in structural optimization. There is probably profit to be made in putting together structural designers and applied mathematicians to work on this task. The nice part of this is that in a properly designed software system, replacing the optimization routine should not be difficult.

In summary, extensions of this thesis will include:

- A focus on the features needed by the engineering designer in a practical working environment.

- Studies to determine the best optimization scheme to be used. (It is likely that problems of structural optimization have special properties that may be helpful in the optimization process. That is certainly the case in analysis where the system matrix is positive definite and sparse.) That is also the case of optimality criteria methods, which have unfortunately proven to have restricted generality.)
- Working with some finite element package like ANSYS or LARSA, which would provide a basis from which to develop a general–purpose software system.
- Human factor studies to assist in determining features appropriate to a general–purpose software system.

APPENDIX A
C++ PROGRAMS

This appendix contains a complete listing of the source code for the programs written in C++ used in this work.

File: args.h

```
// args.h header file for general functions

#if !defined(PARSE_ARGS_INCLUDED)
#define PARSE_ARGS_INCLUDED

class args
{
public:
// constructors
args();
args(char * defaults);

// general functions
int parseArg(char *);
int showAll();
int showHelp();
char * num2str(int num);

// JAVA-style getXXX methods
int getIntegerFlag(char);
double  getDoubleFlag(char);
int getCharsFlag(char c, char * str);
int isDefined(char);

// member variables
int numArgs;
char argFlags[40], argChars[40][40];

};


#endif // PARSE_ARGS_INCLUDED
```

File: complex.h

```
// complex.h: interface for the complex class.
//
//////////////////////////////////////////////////////////////

#if !defined(COMPLEX_INCLUDED)
#define COMPLEX_INCLUDED


#include <iostream.h>
#include <fstream.h>
#include <stdio.h>


class complex
{
   public:
/* members of the class */
double real, imag;

/* constructor / destructor */
complex();
complex(double re, double im);
complex(complex &v);
   ~complex();

/* overloaded operators */
complex operator+ (const complex & v) const;
complex operator- (const complex & v) const;
complex operator* (const complex & v) const;
complex operator/ (const complex & v) const;

complex operator- (void) const;// negation

complex operator* (double d) const;
complex operator- (double d) const;
friend complex operator- (double d, const complex & v);
friend complex operator* (double d, const complex & v);
friend complex operator/ (double d, const complex & v);
const complex&  operator= (const complex & v);
const complex&  operator= (const double & d);

// double& operator() (int i, int j=0) const;
```

```
// double& operator[] (int d) const;

friend ostream& operator<<(ostream &out, const complex & v);
friend ofstream& operator<<(ofstream &out, const complex &v);


/* functions useful for the complex class */
int setVal(double re, double im);
double magnitude() const;
complex xroot(int n=2, int k=0); // default is first square root
complex xexp();


};


#endif // !defined(COMPLEX_INCLUDED)
```

File: constraint.h

```
// constraint.h header file for constraint and objective functions

#if !defined(CONSTRAINT_INCLUDED)
#define CONSTRAINT_INCLUDED

// This is the max number of unique 'items' that can possibly
// be constrained (e.g., LengthChange, X-displacement, ...)
#define MAX_ITEMS 10

class constraint
{
   public:
constraint();
constraint(int, int);
virtual ~constraint();

// members of the 'constraint' class
int number; /* total count */
int isObjective;
char existingConstraint[MAX_ITEMS];
char *item; /* item to be constrained, e.g., L */
int *loc;/* location of constraint, e.g., 2  */
int *type;/* type of constraint, e.g., U  */
double*val;/* value of constraint, e.g., 0.003 */

// general functions
char * num2str(int num);
int isDefined(char ch);
int findItem(char ch);
int setDefined(char ch);
int showValues();
int parseConstraintString(char * str, int i);
int parseObjectiveString(char * str, int i);

static int countNumber(char*, char *);
static int numberConstraints(char*);
static int numberObjectives(char*);
static int readConstraints(char * fileName, constraint * f, con-
straint * g);
};

#endif  // CONSTRAINT_INCLUDED
```

```
File: fr2d.h
/* fr2d.h: interface for the fr2d class.
 *
 * This version is a nonlinear, elastic 2D frame
 *
 */

#if !defined(FR2D_INCLUDED)
#define FR2D_INCLUDED

#include "constraint.h"
#include "mymath.h"
#include "matrix.h"


class fr2d
{
public:
fr2d();
fr2d(int *);
fr2d(int *, char *);
fr2d(int b, int n, int s);
virtual ~fr2d();

/* methods that return int */
int analyze();
int analyze(int nLoadSteps, int nIterations);
int assembleKe();
int assembleKg();
int beamColumnStiffness(int, double &, double &);
int echoJointDisplacements();
int echoMemberDisplacements();
int echoMemberForces();
int echoStructure();
int initialize();
int insertElement(matrix &, double[3][3], int row, int col);
int isViolated(constraint * g);
int mkF(matrix & c, constraint * f);
int mkG(matrix & A, matrix & b, constraint * g, int f_num, double
* pda=NULL);
int mkKei(double ske[3][3], int bar);
int mkKgi(double skg[3][3], int bar);
int mkKgi3(double kg11[3][3], double kg12[3][3], double
kg22[3][3], int bar);
int mkProps(int bar);
int mkRN(double NRP[3][3], double NRM[3][3], int bar);
int mkRotationMatrix(double R[3][3], int bar);
int nonlinearNdelta(matrix & D, matrix d, char * flag="");
int optimizeOC(double * low, double * upp);
int prod3T(double A[3][3], double B[3][3], double C[3][3], double
D[3][3]);
// int prod3k(double A[3][3], double B[3][3], double C[3][3], dou-
ble D[3][3]);
int readFile(char*);
int rmSmallBars(double);

int writeFile(char*);
int writeResults(char*);
int writeResults();
int zeroDisplacements(void);
int zeroForces(void);

static int * getDimensions(char*);

/* methods that return double */
double a2i(int bar);
double getUnbalancedForce(matrix & P);
double scale(constraint * g, double factor);
double volume();


/*
 ****************************************
 *
 * variables (members) of the frame class
 *
 ****************************************
 */


/* Ke the global stiffness matrix
 * Kgthe geometric stiffness matrix
 * Kinv the inverse of Ke
 * N_sys transformation matrix
 * dKdAderivative of Ke with respect to A
```

```
 * K_sys system member stiffness matrix; local coordinate system
 * nonlinear determines if Kg is computed and considered
 *
 */

int NB, NN, NS, dof, freeNodes;
int nonlinear;
matrix Ke, Kg, Kinv, K_sys, N_sys, modeShapes, frequencies;

/* Each member of a frame has these variables (properties).
 * The array, A2I, is used to relate the area and the section
 * modulus.
 */

struct frameMember {
          int NP;/* positive (+) node */
          int NM;/* negative (-) node */
          double area;/* cross sectional area */
          double section_I;/* section modulus */
  double A2I[2];/* relate area to section_I */
          double theta;/* radians in global coord. sys. */
          double length;
          double E;
          double Fx;/* axial force */
          double Mp;/* moment at + end (node) */
          double Mn;/* moment at - end (node) */
          double sigAxial;/* axial stress */
          double sigFlex;/* flexural stress */
          double dl;/* change in length */
          double rp;/* rotation at + end (node) */
          double rn;/* rotation at - end (node) */
} *bars;

struct node {
  double x, y;/* coordinates */
  double dx, dy, rz;/* translations */
  double Px, Py, Mz;/* loading */
} *nodes;

};

#endif // FR2D_INCLUDED
```

File: fr3d.h

```
/* fr3d.h: interface for the fr3d class.
 *
 * This version is a nonlinear, elastic 3D frame
 *
 */

#if !defined(FR3D_INCLUDED)
#define FR3D_INCLUDED

#include "constraint.h"
#include "mymath.h"
#include "matrix.h"

class fr3d
{
public:
fr3d();
fr3d(int *);
fr3d(int *, char *);
fr3d(int b, int n, int s);
virtual ~fr3d();

/* methods that return int */
int analyze();
int analyze(int nLoadSteps, int nIterations);
int assembleKe();
int assembleKg();
int beamColumnStiffness(int, double &, double &);
int echoJointDisplacements();
int echoMemberDisplacements();
int echoMemberForces();
int echoStructure();
int firstDofIndex(int n);
int getDofNumber(int node, int index);
int initialize();
int insertElement(matrix &, double[6][6], int row, int col);
int insertToNsys(matrix &, double[6][6], int row, int col);
int isViolated(constraint * g);
int mkF(matrix & c, constraint * f);
int mkG(matrix & A, matrix & b, constraint * g, int f_num, double
* pda=NULL);
int mkKei(double ske[6][6], int bar);
```

```
int mkKgi(double skg[6][6], int bar);
int mkKgi3(double kg11[6][6], double kg12[6][6], double
kg22[6][6], int bar);
int mkProps(int bar);
int mkRN(double NRP[6][6], double NRM[6][6], int bar);
int mkRotationMatrix(double R[6][6], int bar);
int nonlinearNdelta(matrix & D, matrix d, char * flag="");
int numberDofAtNode(int n);
int prod6T(double A[6][6], double B[6][6], double C[6][6], double
D[6][6]);
int readFile(char*);
int rmSmallBars(double);

int writeFile(char*);
int writeResults();
int zeroDisplacements(void);
int zeroForces(void);

static int * getDimensions(char*);

/* methods that return double */
double a2ix(int bar);
double a2iy(int bar);
double a2iz(int bar);
double getUnbalancedForce(matrix & P);
double scale(constraint * g, double factor);
double volume();

/*
 ****************************************
 *
 * variables (members) of the frame class
 *
 ****************************************
 */

/* Ke the global stiffness matrix
 * Kgthe geometric stiffness matrix
 * Kinv the inverse of Ke
 * N_sys transformation matrix
 * dKdAderivative of Ke with respect to A
 * K_sys system member stiffness matrix; local coordinate system
 * nonlinear determines if Kg is computed and considered
 *
 */

int NB, NN, NS, dof, freeNodes;
int nonlinear;
matrix Ke, Kg, Kinv, N_sys, modeShapes, frequencies;

/* Each member of a frame has these variables (properties).
 * The array, A2I, is used to relate the area and the section
 * modulus.
 */

struct frameMember {
          int NP;/* positive (+) node */
          int NM;/* negative (-) node */
          double area;/* cross sectional area */
          double Ix, Iy, Iz;/* section modulus */
  double A2Ix[2], A2Iy[2], A2Iz[2];/* relate area to section_I */
          double thx,thy,thz;/* radians in global coord. sys. */
          double length;
          double E, G;
          double Fx;/* axial force */
  double Torque;/* torque */
          double Mpy, Mpz;/* moment at + end (node) */
          double Mny, Mnz;/* moment at - end (node) */
          double dl;/* change in length */
          double twist;/* rotation about local x-axis */
          double rpy, rpz;/* rotation at + end (node) */
          double rny, rnz;/* rotation at - end (node) */
} *bars;

struct node {
  int fix[6];
  double x,  y,  z;/* coordinates */
  double dx, dy, dz;/* translations */
  double rx, ry, rz;/* rotations */
  double Px, Py, Pz;/* loading */
  double Mx, My, Mz;/* loading */
} *nodes;

};

#endif // FR3D_INCLUDED
```

File: iplp.h

```
/* file: iplp.h
 *
 * Methods and class members of the iplp class.
 *
 */


#if !defined(IPDUAL_INCLUDED)
#define IPDUAL_INCLUDED

#include "matrix.h"

class iplp
{
   public:
/* constructor / destructor */
iplp();
iplp(int * dim);
iplp(int r, int c);
virtual ~iplp();

/* member functions */
static int * getDimensions(char * fName);
int printInfeasibility();
int printLP();
int printObjective();
int readLP(char * fName, char * type);

int startBigM();
int startBill();
int startFix1(int i, double val);
int startY0(double fixVal=0.0000001);
int solvePrimal();
int solveDual();
int solvePD();

/* class members */
int itermax, // number of iterations
row, // number of rows in the LP matrix (A)
col, // number of columns in LP matrix (A)
verbose;// flag for level of output
double rho, // step size limit
obj, // primal objective value
dobj;// dual objective value
class matrix A, // the LP matrix
b, // dual objective function
c, // primal objective function
x, // primal variables
y, // dual variables
z;// dual slack vector

};

#endif // !defined(IPDUAL_INCLUDED)
```

```
File: matrix.h
// matrix.h: interface for the matrix class.
//
//////////////////////////////////////////////////////////////////
/////

#if !defined(MATRIX_INCLUDED)
#define MATRIX_INCLUDED


#include <iostream.h>
#include <fstream.h>
#include <stdio.h>


class matrix
{
   public:
/* members of the class */
int row, col;
double *m;

/* constructor / destructor */
matrix(int r, int c=1);
matrix(matrix &v);
   ~matrix();

void del() {delete [] m;};

/* overloaded operators */
matrix operator+ (const matrix & v) const;
matrix operator- (const matrix & v) const;
matrix operator* (const matrix & v) const;
matrix operator* (double d) const;
matrix operator- (void) const;
friend matrix operator* (double d, const matrix & v);
const matrix&  operator= (const matrix & v);
const matrix&  operator= (const double & d);

double& operator() (int i, int j=0) const;
double& operator[] (int d) const;

friend ostream& operator<<(ostream &out, const matrix & v);
friend ofstream& operator<<(ofstream &out, const matrix &v);


/* member functions: linear algebra */
matrix invert() const;
matrix transpose() const;
int setZero();
int showSparse() const;
int showSparse(double lim) const;


/* functions useful for the matrix class */
// Method 'eigen' is a helper function for 'nroot' which finds
// the eigenvectors and eigenvalues.
static int eigen(double *a, double *r__, int dim_1, int *mv);
static int nroot(matrix a, matrix b, matrix & val, matrix & vec);
static int solve(matrix a, matrix b, matrix & x, int verbose=0);
static double minv(matrix & a);

};


#endif // !defined(MATRIX_INCLUDED)
```

File: mymath.h

```
// mymath.h header file for general functions

#if !defined(MYMATH_INCLUDED)
#define MYMATH_INCLUDED


#define SIZE1 200

// general functions


int sign(double);
int invert(double A[SIZE1][SIZE1], double Ainv[SIZE1][SIZE1], int
dim);
int product(
double A[SIZE1][SIZE1],
double B[SIZE1][SIZE1],
double C[SIZE1][SIZE1],
int Arows,
int Acols,
int Bcols);
int showSparse(double A[SIZE1][SIZE1], int rows, int cols);
int transpose(
double A[SIZE1][SIZE1],
double Atrans[SIZE1][SIZE1],
int Arows,
int Acols);

#endif // MYMATH_INCLUDED
```

File: tr2d.h

```
// tr2d.h: interface for the tr2d class.
//
//////////////////////////////////////////////////////////////////
/////

#if !defined(TR2D_INCLUDED)
#define TR2D_INCLUDED

#include "constraint.h"
#include "mymath.h"
#include "matrix.h"
// #include "utility.h"


class tr2d
{
public:

tr2d();
tr2d(int * dim);
tr2d(int b, int n, int s);
virtual ~tr2d();
int analyze();
int assemble();
int assembleSimple();
int insertElement(int bar);
int makeUnitVector(int bar);
int mkF(matrix & c, constraint *f);
int mkG(matrix & A, matrix & b,constraint *g, int f_num);
int optimizeOC(tr2d, tr2d);
int readFile(char*);
static int * getDimensions(char*);
int readConstraints(char*, char*);
int rmSmallBars(double);
int unitVector(int, int, double&, double[2]);

int writeFile(char*);
int writeResults(char*);
int writeResults();

int echoStructure();
int echoJointDisplacements();
```

```
int echoMemberDisplacements();
int echoMemberResults();

double scale(constraint *g, double factor);
double volume();

/* properties (members) of the truss class */
struct trussMember {
          int NP;/* positive (+) node */
          int NM;/* negative (-) node */
          double area;/* bar area */
          double length;/* bar length */
          double force;/* bar force */
          double stress;/* bar stress */
          double dl;/* change in length */
          double UV[2];/* unit vector direction */
          double E;/* Young's modulus */
}  * bars;

struct trussNode {
  int    n; /* node number */
  double x, y;/* coordinate */
  double dx, dy;/* translation */
  double Px, Py;/* loading */
} * nodes;

/* the following are
 * NB number of bars
 * NNnumber of nodes
 * NSnumber of supports
 * dofdegrees of freedom
 * Keglobal stiffness matrix
 * Kinv inverse of global stiffness matrix
 * N_systransformation matrix (lcs to gcs)
 */

int NB, NN, NS, dof, freeNodes;
class matrix Ke, Kinv, N_sys, K_sys;

};

#endif // !defined(TR2D_INCLUDED)
```

File: utility.h

```
// utility.h header file for general functions

#if !defined(UTILITY_INCLUDED)
#define UTILITY_INCLUDED


// general functions

void killme(char * msg, int code=0);
void killme(char * msg, char * problem, int code=0);
void showTime(char * msg="", int num=0);

int freeNodes(char *);
int setFileNames(char* jobName, char* structureName, char* con-
straintsName);
int setFileName(char* jobName, char* outputName, int iter);

char * num2str(int num);


#endif // UTILITY_INCLUDED
```

```
File: args.cpp
// args.cpp:
//
// Class to handle command line args. This class was developed in
// the spirit of java-style class-driven programming which is more
// class focused than method of function oriented.
//
// Flags (a single character) are matched with a value (in char
array format)
// and can be accessed using the getXXX methods which return the
value of
// that char array as integer or double.
//
//
//////////////////////////////////////////////////////////////////
/////


#include <stdlib.h>
#include <fstream.h> // contains fstream
#include <iostream.h>                          // cin, cout, <<,
>>
#include <iomanip.h> // formatted I/O
#include <stdio.h>

#include <ctype.h> // isalnum(int ch)
#include <string.h>

#include "utility.h"
#include "args.h"


/* constructor */
args::args() {
int i;
numArgs = 0;
for(i=0; i<40; i++) {
  argFlags[i] = ' ';
  strcpy(argChars[i], "");
}
}
```

```
args::args(char * dummy) {
// Flag 'dummy' causes these initializations
int i;

// printf ("args::args set defaults \n");
char flags[]   = {'a',   'e', 'r', 'i', 'I',  'v', 'V', 'x'};
char values[][10] = {"0.1", "1", "0", "1", "10", "0", "0",
"1.01"};
numArgs = 8;

for(i=0; i<numArgs; i++) {
  // printf("args:: flag[%d] = '%c'\n", i, flags[i]);
  argFlags[i] = flags[i];
  strcpy(argChars[i], values[i]);
}
}



int args::parseArg(char * arg) {
/* The elements in the character array 'arg' should be (by index):
 * 0 a minus sign (-) indicating that a flag follows
 *1a single letter indicating the type of flag
 *2-19characters that make up the flag
 */

int i, j;

if(arg[0] != '-') {
cout << "\n ***Error::args::parseArg arguments should "
   << "begin with '-f' where 'f' is a valid flag" << endl;
return -1;
}

// Scan current flags for an existing setting and overwrite if
found.
i = 0;
while( (argFlags[i] != arg[1]) && (i < numArgs) ) {
  // printf("\targs::parseArg flag '%c' against '%c' \n", arg[1],
argFlags[i]);
  // if(argFlags[i] == arg[1]) printf("\tfound match\n");
  i++;
}
```

```
if(argFlags[i] == arg[1]) {
  // This is an existing flag, overwrite the current value.
  for(j=2; j<=strlen(arg); j++) argChars[i][j-2] = arg[j];
  // printf("args::parseArg change '-%c' to '%s' \n", arg[1], arg-
Chars[i]);
} else {
  // This is a new flag, add it to the list.
  argFlags[numArgs]=arg[1];
  for(i=2; i<=strlen(arg); i++) argChars[numArgs][i-2] = arg[i];
  // printf("args::parseArg new val '-%c' is %s \n", arg[1], arg-
Chars[numArgs]);
  numArgs++;
}

// get the value for this flag as a string
// for(i=2; i<=strlen(arg); i++) argChars[numArgs][i-2] = arg[i];
// printf("\tflag: -%c \tvalue: %s\n", arg[1], argChars[numArgs]);


return 0;
}




int args::getCharsFlag(char c, char * str) {
/* Locate the sequence of characters associated with the command
 * line arg 'c'. This would normally be used to return a string
 * such as a filename in the variable 'str'.
 */

int i=0;
// char * val = new char[dim];

while( (argFlags[i] != c) && (i < numArgs) ) {
  // printf("\tcompare flag %d, val '%c' against '%c' \n", i, arg-
Flags[i], c);
  i++;
}

if(argFlags[i] == c) {
  strcpy(str, argChars[i]);
  // printf("args::getCharsFlag flag '%c' found, val %s \n", c,
str);
} else {
  printf("args::getCharsFlag flag '%c' not found\n", c);
  killme("***Error::args::getCharsFlag undefined argument");
}

return 0;
}




double args::getDoubleFlag(char c) {
/* Locate and return the value associated with flag 'c'. The value
 * is returned as a double after it is converted from its 'native'
 * character type.
 */

int i=0;

while(i < numArgs) {
  if(argFlags[i] == c) return atof(argChars[i]);
  // printf("\tcompare flag %d, val '%c' against '%c' \n", i, arg-
Flags[i], c);
  i++;
}

printf("args::getDoubleFlag: argument '%c' not found\n", c);
killme("***Error::args::getDoubleFlag undefined argument");

return 0.0;
}



int args::getIntegerFlag(char c) {
/* Locate and return the value associated with flag 'c'. The value
 * is returned as an integer after it is converted from its
'native'
 * character type.
 */
```

```
int i=0, val=0;

while( (argFlags[i] != c) && (i < numArgs) ) {
  // printf("\tcompare flag %d, val '%c' against '%c' \n", i, arg-
Flags[i], c);
  i++;
}

if(argFlags[i] == c) {
  val = atoi(argChars[i]);
  // printf("args::getIntegerFlag flag '%c' found, val %d \n", c,
val);
} else {
  printf("args::getIntegerFlag flag '%c' not found\n", c);
  killme("***Error::args::getIntegerFlag undefined argument");
}

return val;
}



int args::isDefined(char c) {
// Determine if an argument flag 'c' is currently defined and
// return 1 if so, 0 otherwise.

int i;
for(i=0; i<numArgs; i++) if(argFlags[i] == c) return 1;
return 0;
}



int args::showAll() {
/* Print the currently defined command line args in a format that
 * can be used in a later command (i.e., -Fvalue, where 'F' is the
 * character flag and 'value' is the string value
 */
int i;
for(i=0; i<numArgs; i++) printf(" -%c%s", argFlags[i], arg-
Chars[i]);
printf("\n");

return 0;
}



int args::showHelp() {
printf("\nFlags:\n");
printf("\t-aVALUE\t\tSet alpha to VALUE\n");
printf("\t-bNAME\t\tSet jobName to NAME\n");
printf("\t-d\t\tExplicitly set defaults\n");
printf("\t-eNUM\t\tPrint every NUM iterations\n");
printf("\t-h\t\tShow this help\n");
printf("\t-iNUM\t\tSet NUM iterations in main program\n");
printf("\t-Inum\t\tSet num iterations in LP solver\n");
printf("\t-rNUM\t\tRestart from iteration NUM\n");
printf("\t-vLEVEL\t\tSet LEVEL of verbose in main program\n");
printf("\t   LEVEL = -1 \tshow LP matrix at each iteration\n");
printf("\t   LEVEL = -2 \tshow constraints\n");
printf("\t   LEVEL = -3 \tshow Ke and N_sys\n");
printf("\t   LEVEL = -4 \tshow time\n");
printf("\t-Vlevel\t\tSet level of verbose in LP solver\n");

return 0;
}
```

```
File: complex.cpp
// complex.cpp: implementation of the complex class.
//
////////////////////////////////////////////////////////////////
/////

#include <math.h>
#include "complex.h"

#define PI 3.141592653

////////////////////////////////////////////////////////////////
/////
// Construction/Destruction
////////////////////////////////////////////////////////////////
/////


/* basic constructor */
complex::complex() {
/* initalize to zero */
real = 0.0;
imag = 0.0;
}


/* initialization constructor */
complex::complex(double re, double im) {
real = re;
imag = im;
}


/* copy constructor */
complex::complex(complex &v) {
real = v.real;
imag = v.imag;
}



/* destructor; free memory */
complex::~complex() {
// printf("destructor\n");
}



/* ------------------------------------
 *
 * basic complex operations (+, -, *, =)
 *
 * ------------------------------------
 */

/* addition */
complex complex::operator+ (const complex & v) const {
complex temp;
temp.real = v.real + real;
temp.imag = v.imag + imag;

return temp;
}



/* subtraction */
complex complex::operator- (const complex & v) const {
complex temp;
temp.real = real - v.real;
temp.imag = imag - v.imag;

return temp;
}


/* subtraction, complex minus a double */
complex complex::operator- (const double d) const {
complex temp;
temp.real = real - d;
temp.imag = imag;

return temp;
}
```

```
/* subtraction, double minus a complex */
complex operator- (double d, const complex & v) {
complex temp;
temp.real = d - v.real;
temp.imag = -v.imag;

return temp;
}


/* negation */
complex complex::operator- () const {
complex temp;
temp.real = -real;
temp.imag = -imag;

return temp;
}


/* multiplication, two complex numbers */
complex complex::operator* (const complex & v) const {
complex temp;
temp.real = real*v.real - imag*v.imag;
temp.imag = real*v.imag + imag*v.real;

return temp;
}

/* multiplication, complex and a double */
complex complex::operator* (double d) const {
complex temp;
temp.real = real * d;
temp.imag = imag * d;

return temp;
}


/* multiplication, double and a complex */
complex operator* (double d, const complex & v) {
complex temp;
temp.real = v.real * d;
temp.imag = v.imag * d;

return temp;
}


/* division, two complex numbers */
complex complex::operator/ (const complex & v) const {
if(v.magnitude() <= 0.0) {
  printf("***Error::complex::division v.magnitude <= 0\n");
  exit(-1);
}

complex temp;
temp.real = (real*v.real + imag*v.imag)/v.magnitude();
temp.imag = (imag*v.real - real*v.imag)/v.magnitude();

return temp;
}


/* division, double divided by complex */
complex operator/ (double d, const complex & v) {
// Construct a temporary complex object, 'd_' with a value
// of zero for the imaginary part and value of 'd' for the
// real part. Then perform normal complex division.

complex temp, d_(d, 0.0);
temp = d_ / v;

return temp;
}

/* assignment, two complex numbers */
const complex& complex::operator= (const complex & v) {
real = v.real;
imag = v.imag;

return *this;
}


/* assignment, initialize real part of complex with a double */
```

```
const complex& complex::operator= (const double & d) {
real = d;
imag = 0.0;

return *this;
}

/* ------------------------------------
 *
 * general math operations
 *
 * ------------------------------------
 */

double complex::magnitude() const {
return (real*real + imag*imag);
}

int complex::setVal(double re, double im) {
real = re;
imag = im;
return 0;
}

complex complex::xroot(int n=2, int k=0) {
/* Return the k-th of the n-roots of complex number. That
 * is, a complex number 'z' has 'n' roots to z^(1/n) and this
 * will return the k-th one of them. Here the variable 'arg' is
 * the angle from the real axis and 'r' is the magnitude. Default
 * values will give the first (i.e., the zero-th) square root
 * similar to sqrt() for real numbers.
 */

double arg, r;
complex temp;
if((n < 1) || (k>n) ) {
  printf("***Error::complex::xroot bad dimension\n");
  exit(-1);
}

r = sqrt(real*real + imag*imag);
if((real == 0) && (imag != 0)) arg = PI / 2;
else arg = atan2(imag, real);

temp.real = pow(r,1.0/n)*cos((arg+2*k*PI)/n);
temp.imag = pow(r,1.0/n)*sin((arg+2*k*PI)/n);

return temp;
}

complex complex::xexp() {
/* Return the complex number e^z where z is complex. */

complex temp;
temp.real = exp(real)*cos(imag);
temp.imag = exp(real)*sin(imag);
return temp;
}

/* ------------------------------------
 *
 * output operations
 *
 * ------------------------------------
 */

/* basic cout */
ostream& operator<<(ostream & out, const  complex &v){
  if(v.imag >= 0.0)
    out << v.real << " + " << v.imag << "i";
  else
    out << v.real << " - " << fabs(v.imag) << "i";

  return out;
}

/* file output */
ofstream& operator<<(ofstream & out, const complex &v){
  if(v.imag >= 0.0)
    out << v.real << " + " << v.imag << "i";
  else
    out << v.real << " - " << fabs(v.imag) << "i";

  return out;
}
```

```
File: constraint.cpp
// constraint.cpp: routines for the 'constraint' class
//
//
//////////////////////////////////////////////////////////////////
/////


#include <stdlib.h>
#include <fstream.h> // contains fstream
#include <iostream.h>                          // cin, cout, <<,
>>
#include <iomanip.h> // formatted I/O
#include <stdio.h>
// #include <math.h>

#include <string.h>
#include <ctype.h> // isalnum(int ch)
#include "utility.h"
#include "constraint.h"

constraint::constraint() {

}

constraint::constraint(int dim, int obj) {
int i;

// printf("begin constructing constraint, dim %d \n", dim);
number = dim;
isObjective = obj;

if(!obj) {
  item = new char[dim];
  type = new int[dim];
}
loc = new int[dim];
val = new double[dim];

for(i=0; i<MAX_ITEMS; i++) existingConstraint[i] = '\0';

}

constraint::~constraint() {

}


int constraint::countNumber(char* fileName, char * countType) {
/*
 * This method reads the file 'fileName' and searches for the
character
 * sequence 'countType' at the beginning of a line. On any line
that has
 * this sequence, all digits that follow are interpreted as a sin-
gle
 * number and this is added to the 'sum'. This variable is then
returned.
 */

char str[160], *p;
int sum;
FILE *infile;

if( (infile = fopen(fileName, "r")) == NULL ) {
  printf(" ***Error::countNumber: '%s' not opened\n", fileName);
  return -1;
}

sum = 0;
do {
  fgets(str, 159, infile);
  p = strtok(str, " \t");
  // printf("countNumber: p = %s strcmp = %d \n", p, strcmp(p,
countType) );
  // printf(" countNumber found: %s", str);
  if(!strcmp(p, countType)) {
  p = strtok('\0', " \t");
sum += atoi(p);
// printf("countNumber: adding sum %d\n", sum);
  }
} while ( !feof(infile) );
fclose(infile);

// printf("constraint::countNumber: returning sum %d\n", sum);
return sum;
```

```
}


int constraint::findItem(char ch) {
// Determine if an item with the tag 'ch' has been constrained.
//
int i;
for(i=0; i<number; i++)
  if(ch == item[i]) return 1;

// printf("constraint::findItem: item '%c' not found\n", ch);
return 0;
}


int constraint::isDefined(char ch) {
// Determine if an item with the tag 'ch' has been constrained.
// This method is quicker than method 'findItem' because it works
with
// the much shorter list 'existingConstraint' which should have
been filled
// during 'readConstraints' with one entry for each unique 'item'
// that is constrained.

int i = 0;

// printf("constraint::isDefined: look for '%c'\n", ch);
while(existingConstraint[i] != '\0' && i < MAX_ITEMS) {
  // printf("constraint::isDefined check i = %d\n", i);
  if(existingConstraint[i] == ch) {
    // printf("constraint::isDefined: item '%c' found \n", ch);
    return 1;
  }
  i++;
}

// printf("constraint::isDefined: item '%c' not found\n", ch);
return 0;
}
```

```
int constraint::readConstraints( char* fileName, constraint * f,
constraint * g) {

// reads only the 'non-comment' data from file into
// objective, 'f' and constraints, 'g'
//
// The 'constraint' structure is:
// membercontentsexampledata type
//
// item quantity dl, dx, ...char
// location member / node 1, 34, ...int
// typetype UPPER / LOWERint (+/- 1)
// val limit0.05, -120double


char str[160], *p;
int i, fi, gi, num;

FILE *infile;

fi = 0;
gi = 0;

if( (infile = fopen(fileName, "r")) == NULL ) {
  printf(" ***Error::readConstraints: '%s' not opened\n", file-
Name);
  return 1;
}

// printf("readConstraints: begin\n");
do {
  fgets(str, 159, infile);
  if(feof(infile)) break;
  // printf(" line: %s", str);
  p = strtok(str, " \t");
  if(!strcmp(p, "*CONSTRAINTS")) {
    p = strtok('\0', " \t");
    num = atoi(p);
    for(i=0; i<num; i++) {
fgets(str, 159, infile);
g->parseConstraintString(str, gi);
// printf(" Constraint %d:", gi);
// printf("item: %s, location: %d, type: %d, value: %f \n",
```

```
//  g->item[gi], g->loc[gi], g->type[gi], g->val[gi]);
gi++;
    }
  } else if(!strcmp(p, "*OBJECTIVE")) {
    p = strtok('\0', " \t");
    num = atoi(p);
    for(i=0; i<num; i++) {
fgets(str, 159, infile);
f->parseObjectiveString(str, fi);
// printf(" Objective %d:", fi);
// printf("item: %s location: %d type: %d value: %f \n",
//f->item[fi], f->loc[fi], f->type[fi], f->val[fi]);
fi++;
    }
  } else if(!strcmp(p, "%")) {
  } else if(!strcmp(p, "\n")) {
  } else if(p[0] == '%') {
  } else if(p[0] == '\0') {
  } else {
fclose(infile);
killme("readConstraints bad entry\n", p, -1);
  }

} while ( !feof(infile) );

fclose(infile);

// check to see if the number read is as expected
if(f->number != fi || g->number != gi)killme("readConstraints: bad
count");

return 0;
}


int constraint::parseConstraintString(char * str, int i) {
// This method fills constraint entry 'i' with it's appropriate
// items (a char array, two ints and one double) from the char
// array 'str'.
char * p;

// printf("begin parseConstraintString, str: '%s'\n", str);
p = strtok(str, " \t");
if(p[0] == 'G') {
  // strcpy(item[i], "Global-buckling");
  setDefined('G');
  item[i] = 'G';
  loc[i] = 0;
  type[i] = 0;
  val[i] = 0;
  return 0;
}

// strcpy(item[i], p);
item[i] = p[0];
setDefined(item[i]);
p = strtok('\0', " \t");
loc[i] = atoi(p);
p = strtok('\0', " \t");
if(!strcmp(p, "UPPER")) { type[i] =  1; }
else if(!strcmp(p, "LOWER")){ type[i] = -1; }
else if(!strcmp(p, "VOLUME")){ type[i] =  1; }
else if(!strcmp(p, "WEIGHT")){ type[i] =  1; }
else {  killme("parseString: bad 'TYPE'\n", -1); }

p = strtok('\0', " \t");
val[i] = atof(p);

return 0;
}

int constraint::parseObjectiveString(char * str, int i) {
// This method fills objective entry 'i' with it's appropriate
// items (one int and one double) from the char
// array 'str'.
char * p;

// printf("begin parseObjectiveString, str: '%s'\n", str);
p = strtok(str, " \t");
loc[i] = atoi(p);
p = strtok('\0', " \t");
val[i] = atof(p);

return 0;
}
```

```
int constraint::setDefined(char ch) {
// Set an element in the array 'existingConstraints' to the
// character 'ch'. The order is not important, just the existence
// so that this array can be scanned later to check if this 'item'
// is constrained. The constructor fills the array with '\0' so
// we'll scan through until either 'ch' is found (return with no
// change) or the first '\0' is found and it will be changed to
// this 'ch'.

int i = 0;

// printf("constraint::setDefined: check '%c'\n", ch);
while(existingConstraint[i] != '\0' && i < MAX_ITEMS) {
  // printf("constraint::setDefined: check '%c' at i=%d\n", ch,
i);
  if(existingConstraint[i] == ch) return 1;
  i++;
}

existingConstraint[i] = ch;
// printf("constraint::setDefined: item '%c' now set\n", ch);
return 0;
}


int constraint::showValues() {

int i, j;
// printf("constraint::showConstraint\n");
if(isObjective) {
  printf(" i \tloc \tval\n");
  for(i=0; i<number; i++)
    printf("%d \t%d \t%f\n", i, loc[i], val[i]);
} else {
  printf(" i \titem \tloc \ttype \tval\n");
  for(i=0; i<number; i++)
    printf("%d \t%c \t%d \t%d \t%f\n", i, item[i], loc[i],
type[i], val[i]);
}

return 0;
}
```

File: fr2d.cpp

```
/*************************************************
 * fr2d.cpp: implementation of the fr2d class.
 *
 * two-dimensional nonlinear, elastic frame
 *
 */

#include <fstream.h> // contains fstream
#include <iomanip.h> // formatted I/O
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>

#include "constraint.h"
#include "complex.h"// for beam column stiffness
#include "fr2d.h"
#include "matrix.h"
#include "utility.h"

#define SMALL 0.00000001


/*************************************
 *
 *  Construction/Destruction
 *
 */

/*
fr2d::fr2d()
{

}
*/


fr2d::fr2d(int * dim) : Ke( 3*(dim[1]-dim[2]), 3*(dim[1]-dim[2])
),
Kg(1, 1),
modeShapes(1, 1),
frequencies(1, 1),
```

```
Kinv( 3*(dim[1]-dim[2]), 3*(dim[1]-dim[2]) ),
K_sys( 3*dim[0], 3*dim[0] ) ,
N_sys(3*dim[0], 3*(dim[1]-dim[2]) )
{
/* Variable 'dim' has three elements from a "quick peek" at the
 * file. The elements, in order are NB, NN, and NS (number of bars,
 * nodes, and supports). This gives enough to allocate space for
 * the entire structural analysis.
 *
 * The text above after the colon must be there to allocate space
 * for the matrix objects.
 */

// printf("dim: %d, %d, %d\n", dim[0], dim[1], dim[2] );
NB = dim[0];
NN = dim[1];
NS = dim[2];

freeNodes = NN - NS;/* number of free nodes */
dof = 3*freeNodes;/* total degrees of freedom */
nonlinear = false;/* default value is false */

/* nodes and bars are one-based (i.e., first element is index 1,
not 0) */
nodes = new node[NN+1];
bars  = new frameMember[NB+1];

}

fr2d::fr2d(int * dim, char * nlflag) : Ke( 3*(dim[1]-dim[2]),
3*(dim[1]-dim[2]) ),
Kg( 3*(dim[1]-dim[2]), 3*(dim[1]-dim[2]) ),
modeShapes( 3*(dim[1]-dim[2]), 3*(dim[1]-dim[2]) ),
frequencies( 3*(dim[1]-dim[2]), 1),
Kinv( 3*(dim[1]-dim[2]), 3*(dim[1]-dim[2]) ),
K_sys( 3*dim[0], 3*dim[0] ) ,
N_sys(3*dim[0], 3*(dim[1]-dim[2]) )
{
/* Variable 'dim' has three elements from a "quick peek" at the
 * file. The elements, in order are NB, NN, and NS (number of bars,
 * nodes, and supports). This gives enough to allocate space for
 * the entire structural analysis. Variable 'nlflag' is not used
but
 * required to determine if space should be allocated fo Kg->
 *
 * The text above after the colon must be there to allocate space
 * for the matrix objects.
 */


// printf("dim: %d, %d, %d\n", dim[0], dim[1], dim[2] );
NB = dim[0];
NN = dim[1];
NS = dim[2];

freeNodes = NN - NS;/* number of free nodes */
dof = 3*freeNodes;/* total degrees of freedom */
nonlinear = true;

/* nodes and bars are one-based (i.e., first element is index 1,
not 0) */
nodes = new node[NN+1];
bars  = new frameMember[NB+1];

}

fr2d::~fr2d()
{
}


/************************************
 *
 * class methods -- methods that return int
 *
 */

int fr2d::analyze() {

/* Basic linear elastic analysis */
int i, j, k, ii;
double SK[3][3];
matrix P(dof, 1), delta(dof, 1), Delta(3*NB, 1);

/* set up system matrix and inverse */
assembleKe();
```

```
/* set up load vector */
for (i=1; i<=freeNodes; i++) {
  P[3*i-3] = nodes[i].Px;
  P[3*i-2] = nodes[i].Py;
  P[3*i-1] = nodes[i].Mz;
}

/* solve system */
// cout << "Ke\n" << Ke << endl;
matrix::solve(Ke, P, delta);

/* set displacements from solution */
for (i=1; i<=freeNodes; i++) {
  nodes[i].dx = delta[3*i-3];
  nodes[i].dy = delta[3*i-2];
  nodes[i].rz = delta[3*i-1];
}

/*  compute member forces and displacements */
Delta = N_sys * delta;
// cout << " Delta is \n" << Delta << endl;
// cout << " delta is \n" << delta << endl;
for (i=1; i<=NB; i++) {
  bars[i].dl = Delta[3*i-3];
  bars[i].rp = Delta[3*i-2];
  bars[i].rn = Delta[3*i-1];
  mkKei(SK, i);
  bars[i].Fx = bars[i].dl*SK[0][0];
  bars[i].Mp = bars[i].rp*SK[1][1] + bars[i].rn*SK[1][2];
  bars[i].Mn = bars[i].rp*SK[2][1] + bars[i].rn*SK[2][2];
  bars[i].sigAxial = bars[i].Fx / bars[i].area;
}

Kinv = Ke.invert();

// This single-step nonlinear analysis is used only to form the
// matrix Kg and check for buckling. The displacements are com-
puted
// from the linear-elastic analysis above and not changed here.
if(nonlinear) {
  matrix KeKg(dof, dof);
  assembleKg();
  KeKg = Ke + Kg;
  // matrix::solve(KeKg, P, delta);
  if(matrix::solve(KeKg, P, delta)) {
    // printf(" --- fr2d::analyze() negative term on main diago-
nal\n\n");
    // free memory before exit
    delta.del();
    Delta.del();
    KeKg.del();
    return 1;
  }
  // no buckling but still want to free memory
  KeKg.del();
}

/* free memory */
delta.del();
Delta.del();

return 0;
}


int fr2d::analyze(int nLoadSteps, int nIterations) {
/* When the analyze method is called with two integer arguments,
 * it is taken to be a nonlinear analysis. This will apply the
 * total load in 'nLoadSteps' and each load step will be solved
 * with 'nIterations' for equilibrium.
 *
 * Method returns zero if all loadsteps and iterations are com-
pleted
 * sucessfully, returns number of loadsteps if solver fails (non-
positive
 * term on main diagonal).
 */

// For zero or one load steps, the analysis is nonlinear
// but the displacements are small thus call 'analyze()'.
if(nLoadSteps <= 1) return analyze();

int i, j, k, ii, iter, lstp;
double SK[3][3];
double error, loadFactor, theta_[NB+1];
```

```
matrix P(dof, 1), delta(dof, 1), Delta(3*NB, 1), KeKg(dof, dof);

// During the nonlinear analysis the member orientation, 'theta'
// will be updated to reflect the deformed geometry. Save the
// initial values here and restore at the end of the analysis.
// printf("fr2d::analyze(int, int): begin\n");
for(i=1; i<=NB; i++)  theta_[i] = bars[i].theta;
zeroDisplacements();
zeroForces();

//////////////////////////////////////////////////
//
// iterations
//
for(lstp=1; lstp <= nLoadSteps; lstp++) {
// printf("fr2d::analyze(int, int): loadstep %d\n", lstp);
for(iter=0; iter < nIterations; iter++) {
// printf("fr2d::analyze(int, int): loadstep %d, iter %d\n", lstp,
iter);

/* Set up system matrices Ke, Kg, and Kinv */
assembleKe();
assembleKg();

/* set up load vector */
loadFactor = ((double)lstp) / ((double)nLoadSteps);
for (i=1; i<=freeNodes; i++) {
  P[3*i-3] = loadFactor * nodes[i].Px;
  P[3*i-2] = loadFactor * nodes[i].Py;
  P[3*i-1] = loadFactor * nodes[i].Mz;
}
// Apply only the out-of-balance load
// P = P - N_sys * F
// which will yield an incremental displacement
//
error = getUnbalancedForce(P);
// printf(" fr2d::analyze: step %d, iter %d, error %f\n", lstp,
iter, error);
// cout << "applied load\n" << P << endl;

/* solve system */
KeKg = Ke + Kg;
// cout << "in analyze, Ke: \n" << Ke << " Kg: \n" << Kg << endl;
if(matrix::solve(KeKg, P, delta)) {
  // printf(" --- fr2d::analyze(int, int) negative term on main
diagonal\n\n");
  // restore values and free memory before exit
  for(i=1; i<=NB; i++) bars[i].theta = theta_[i];
  delta.del();
  Delta.del();
  P.del();
  KeKg.del();
  return lstp;
}

/* update displacements from this increment */
for (i=1; i<=freeNodes; i++) {
  nodes[i].dx += delta[3*i-3];
  nodes[i].dy += delta[3*i-2];
  nodes[i].rz += delta[3*i-1];
}

/*  compute member forces and incremental displacements */
nonlinearNdelta(Delta, delta, "THETA");
// cout << "analyze: joint displacement increment: \n" << delta <<
endl;
// cout << "analyze: member displacement increment: \n" << Delta
<< endl;
for (i=1; i<=NB; i++) {
  mkKei(SK, i);
  bars[i].dl += Delta[3*i-3];
  bars[i].rp += Delta[3*i-2];
  bars[i].rn += Delta[3*i-1];
  bars[i].Fx += Delta[3*i-3]*SK[0][0];
  bars[i].Mp += Delta[3*i-2]*SK[1][1] + Delta[3*i-1]*SK[1][2];
  bars[i].Mn += Delta[3*i-2]*SK[2][1] + Delta[3*i-1]*SK[2][2];
}
// printf("analyze: current member forces");
// echoMemberForces();
// printf("\n\nend this iteration\n\n");


}// close iteration loop
}// close loadstep loop
//
// end iterations
```

```
//
//////////////////////////////////////////////////

// Kinv = Ke.invert();

// restore values and free memory before exit
for(i=1; i<=NB; i++) bars[i].theta = theta_[i];
delta.del();
Delta.del();
P.del();
KeKg.del();

return 0;
}


int fr2d::assembleKe() {
/*
 * This method assembles the stiffness matrix, Ke by creating
 * the sub-matrix for each element and then inserting it.
 *
 * local variables (matrices):
 * ke_i contribution to Ke of a single member
 * kg_i contribution to Kg of a single member
 * NRP transformation matrix for + node of a member
 * rotated to global coordinate system
 * NRM transformation matrix for - node of a member
 * rotated to global coordinate system
 * SK primitive member stiffness matrix for a single bar
 * dK derivative of SK for a single bar
 */


int i, ii, j, row;

double ke_i[3][3], kg_i[3][3], temp1[3][3];
double NRP[3][3], NRM[3][3], SK[3][3], RT[3][3], R[3][3];
double kgi11[3][3], kgi12[3][3], kgi22[3][3];

Ke.setZero();
N_sys.setZero();

for(i=1; i<=NB; i++) {/* begin assembly of Ke*/
  /* create matrices for this element */
  mkRN(NRP, NRM, i);
  mkKei(SK, i);

  if(bars[i].NP <= freeNodes) {
    // printf(" element %d effects node %d\n", i, bars[i].NP);
    /* transform from local to global coordinate system */
    prod3T(NRP, SK, NRP, ke_i);
    // cout << "before inserting, ke_i: \n";
    insertElement(Ke, ke_i, bars[i].NP, bars[i].NP);
    insertElement (N_sys, NRP,  i, bars[i].NP);

    if(bars[i].NM <= freeNodes) {
/* these are the cross terms (both nodes are free). It
 * does not affect the transformation matrix, N_sys
 */
    // printf(" el %d, nodes %d %d \n",i, bars[i].NM, bars[i].NP);
prod3T(NRP, SK, NRM, ke_i);
insertElement(Ke, ke_i, bars[i].NP, bars[i].NM);
for(ii=0; ii<3; ii++) for(j=0; j<3; j++) temp1[ii][j] =
ke_i[j][ii];
insertElement(Ke, temp1, bars[i].NM, bars[i].NP);
    }
  }

  if(bars[i].NM <= freeNodes) {
    // printf(" element %d effects node %d\n", i, bars[i].NM);
    prod3T(NRM, SK, NRM, ke_i);
    insertElement(Ke, ke_i, bars[i].NM, bars[i].NM);
    insertElement(N_sys, NRM, i, bars[i].NM);
  }

}/* get next bar for Ke*/

// printf("invert Ke\n");
Kinv = Ke.invert();
// cout << "in assemble, Ke and Kinv\n" << Ke << Kinv << "\ndone
assemble\n";
// printf("fr2d::assemble; done\n");

return 0;
}
```

```
int fr2d::assembleKg() {
/*
 * This method assembles the geometric stiffness matrix, Kg by
creating
 * the sub-matrix for each element and then inserting it.
 *
 * local variables (matrices):
 * ke_i contribution to Ke of a single member
 * kg_i contribution to Kg of a single member
 * NRP transformation matrix for + node of a member
 * rotated to global coordinate system
 * NRM transformation matrix for - node of a member
 * rotated to global coordinate system
 * SK primitive member stiffness matrix for a single bar
 * dK derivative of SK for a single bar
 */


int i, ii, j, row;

double ke_i[3][3], kg_i[3][3], temp1[3][3];
double NRP[3][3], NRM[3][3], SK[3][3], RT[3][3], R[3][3];
double kgi11[3][3], kgi12[3][3], kgi22[3][3];

if(!nonlinear) killme("***fr2d::assembleKg request for Kg in lin-
ear analysis");
// printf("fr2d::assemble begin Kg\n");
Kg.setZero();
for(i=0; i<3; i++) {
  for(j=0; j<3; j++) {
    kg_i[i][j] = 0.0;
    kgi11[i][j] = 0.0;
    kgi12[i][j] = 0.0;
    kgi22[i][j] = 0.0;
  }
}

for(i=1; i<=NB; i++) {
  // mkKgi(kg_i, i);
  mkRotationMatrix(R, i);
  mkKgi3(kgi11, kgi12, kgi22, i);
  // printf("fr2d::assemble kg, bar %d\n", i);
  prod3T(R, kgi11, R, kg_i);
  if(bars[i].NP <= freeNodes) {
insertElement(Kg, kg_i, bars[i].NP, bars[i].NP);

if(bars[i].NM <= freeNodes) {
/*
  // Create  the negative of kg_i before inserting, then
  // return to its original state.
  // Bill's version
  for(ii=0; ii<3; ii++) for(j=0; j<3; j++) kg_i[ii][j] = -
kg_i[ii][j];
  insertElement(Kg, kg_i, bars[i].NP, bars[i].NM);
  for(ii=0; ii<3; ii++) for(j=0; j<3; j++) temp1[ii][j] =
kg_i[j][ii];
  insertElement(Kg, temp1, bars[i].NM, bars[i].NP);
  for(ii=0; ii<3; ii++) for(j=0; j<3; j++) kg_i[ii][j] = -
kg_i[ii][j];

if(bars[i].NM <= freeNodes)
  // Create  the negative of kg_i before inserting, then
  // return to its original state.
  // non-symmetric version
  insertElement(Kg, kg_i, bars[i].NP, bars[i].NM);
  insertElement(Kg, kg_i, bars[i].NM, bars[i].NP);
*/
prod3T(R, kgi12, R, kg_i);
insertElement(Kg, kg_i, bars[i].NP, bars[i].NM);
for(ii=0; ii<3; ii++) for(j=0; j<3; j++) temp1[ii][j] =
kg_i[j][ii];
insertElement(Kg, temp1, bars[i].NM, bars[i].NP);
}
  }
  if(bars[i].NM <= freeNodes) {
    // printf(" element %d effects node %d\n", i, bars[i].NM);
    prod3T(R, kgi22, R, kg_i);
    insertElement(Kg, kg_i, bars[i].NM, bars[i].NM);
  }
}/* get next bar for Kg*/

// printf("fr2d::assembleKg; done\n");
```

```
return 0;
}

int fr2d::beamColumnStiffness(int i, double & dist, double & cof)
{
// Form beam column stiffness factors for member 'i'. This uses
// the complex number class.

complex AK,C1,C2,EKL,EMKL,AKL;
double EI;

EI = bars[i].E * bars[i].section_I;

if(EI == 0.0) {
  cof  = 0.0;
  dist = 0.0;
  return 0;
}

// if load is very far from critical use the regular 4EI/L
if(fabs(bars[i].Fx * bars[i].length * bars[i].length / EI) <
0.0001) {
  // printf("beamColumnStiffness: bar %d is far from critical\n",
i);
  cof  = 2.0 * EI / bars[i].length;
  dist = 2.0 * cof;
  return 0;
}

AK = bars[i].Fx / EI;
AK = AK.xroot();
AKL = AK * bars[i].length;
EKL = (AK*bars[i].length).xexp();
EMKL = 1.0 / EKL;

C2 = -1.0/(AK*(1.0-EMKL-(EKL-1.0)*(AKL-1.0+EMKL)/(EKL-1.0-AKL)));
C1 = -1.0*C2*(EMKL-1.0+AKL)/(EKL-1.0-AKL);
cof = (AK*AK*(C1+C2)).real * EI;
dist = (-1.0 * (AK*AK*(C1*EKL+C2*EMKL)).real ) * EI;

return 0;
}
```

```
int fr2d::echoJointDisplacements() {// echo joint displacements

printf("\n\n i\tdx \t\tdy \t\trz \n");
for (int i=1; i<=NN; i++)
  printf(" %d \t%e \t%e \t%e\n",
    i, nodes[i].dx , nodes[i].dy, nodes[i].rz );

return 0;
}

int fr2d::echoMemberDisplacements() {// echo member displacements

printf("\n\n i\tdl \t\trp \t\trn \n");
for (int i=1; i<=NB; i++)
  printf(" %d \t%e \t%e \t%e\n",
    i, bars[i].dl , bars[i].rp, bars[i].rn );

return 0;
}

int fr2d::echoMemberForces() {// echo member forces

printf("\n\n i\tFx \t\tMp \t\tMn \n");
for (int i=1; i<=NB; i++)
  printf(" %d \t%e \t%e \t%e\n",
    i, bars[i].Fx , bars[i].Mp, bars[i].Mn );

return 0;
}

int fr2d::echoStructure() {
/*
 * This shows just the variables that define the structure,
 * i.e., not the results.
 *
 */

int i;

printf(" %d bars \n %d nodes \n %d supports\n", NB, NN, NS);

printf("Nodes:\n i\t\tx\t\ty\t\tPx\t\tPy\t\tMz\n");
for (i=1; i<=NN; i++)// nodal properties
```

```
  printf(" %d \t%10.5f \t%10.5f \t%10.5f \t%10.5f  \t%10.5f \n",
     i, nodes[i].x, nodes[i].y, nodes[i].Px, nodes[i].Py,
nodes[i].Mz);

printf("Members:\n
i\tNM\tNP\t\tarea\t\tI\t\tlength\t\ttheta\t\tE\n");
for (i=1; i<=NB; i++)// member properties
  printf(" %d \t%d \t%d \t%10.5f \t%10.5f  \t%10.5f \t%10.5f
\t%10.5f \n",
    i, bars[i].NM, bars[i].NP, bars[i].area, bars[i].section_I,
bars[i].length, bars[i].theta, bars[i].E);

return 0;
}


int* fr2d::getDimensions(char* fileName)
{
/* This method only reads the first line of the model
 * and returns the dimensions so they can be used for
 * the constructor which allocates the space for the model
 */

int * dim = new int[3];

fstream inPut;

// fileName="file.dat";
inPut.open(fileName, ios::in);  // open the file
if(inPut.fail())  {
  cout << "***Error::getDimensions file: "
<< fileName << " not opened. \n";
  exit(-1);
}

inPut >> dim[0] >> dim[1] >> dim[2];
inPut.close();  // close the file

return dim;
}


int fr2d::initialize() {
/*
 * This should probably be near the constructor but it is
 * here for now. It calls routines that are normally done
 * only once for the structure; i.e., not at each iteration
 * and probably called from the main program.
 *
 */

int i;

for (i=1; i<=NB; i++) mkProps(i);// set basic member props.

return 0;
}

int fr2d::insertElement(matrix & MAT, double m[3][3], int row, int
col) {
/*
 * Insert a 3x3 matrix contribution to a larger matrix (e.g., Ke,
Kg,
 * or N_sys). Variables 'row' and 'col' are the one-based indices
of
 * the location to insert this 3x3 matrix, 'm' into the largeer
matrix,
 * 'MAT'. Thus the indices are one-based but the matrices 'm' and
'MAT' are
 * zero-based. Also, the indices treat the 3x3 matrix as a single
element
 * thus (row,col) indices (2,2) will put the elements from 'm'
into 'MAT'
 * beginning at MAT(3,3)=m[0][0].
 */

int i, j, r, c;

r = 3*(row-1);
c = 3*(col-1);

for(i=r; i<r+3; i++)
  for(j=c; j<c+3; j++)
    MAT(i,j) += m[i-r][j-c];

// printf("insert (%d, %d) value %f\n", i, j, m[i-r][j-c]);
```

```
return 0;
}


int fr2d::isViolated(constraint * g) {

int i, ii, b;
double temp = 0.0, maxRatio = 0.0;

/* Check to see if any constraints are violated. This will check
 * for the occasional case where a large initial scaling is made
 * that is too non-linear to give a good prediction. If any con-
straints
 * are voilated, a 1 will be returned; zero otherwise.
 */

for(i=0; i < g->number; i++) {
  b = g->loc[i];
  // printf("fr2d::isViolated: constraint %d, item %c\n", i, g-
>item[i]);
  /* member length change */
  if(g->item[i] == 'L') {
    if (sign(g->val[i]) == sign(bars[b].dl) )
temp = bars[b].dl / g->val[i];
  /* member end rotations (positive end) */
  } else if(g->item[i] == 'P' ) {
    if (sign(g->val[i]) == sign(bars[b].rp) )
temp = pow( (bars[b].rp / g->val[i]), 1/bars[b].A2I[1]);
  /* member end rotations (negative end) */
  } else if(g->item[i] == 'N') {
    if (sign(g->val[i]) == sign(bars[b].rn) )
temp = pow( (bars[b].rn / g->val[i]), 1/bars[b].A2I[1]);
  /* joint displacement (dx) */
  } else if(g->item[i] == 'X') {
    if (sign(g->val[i]) == sign(nodes[b].dx) )
temp = nodes[b].dx / g->val[i];
  /* joint displacement (dy) */
  } else if(g->item[i] == 'Y') {
    if (sign(g->val[i]) == sign(nodes[b].dy) )
temp = nodes[b].dy / g->val[i];
  /* member size (M) */
  } else if(g->item[i] == 'M') {
    if (g->type[i] == -1)
    temp = g->val[i] / bars[b].area;
  /* default */
  } else {
    killme("fr2d::isViolated unknown constraint");
  }
  // maxRatio = temp > maxRatio ? temp : maxRatio;
  if(temp > maxRatio) {
maxRatio = temp;
ii = i;
  }
}

// printf("fr2d::isViolated, maxRatio = %f \n", maxRatio);
if(maxRatio > 1) {
  printf("\t isViolated: maxRatio = %10.5f \n", maxRatio);
  printf("\t\t location: %d\n", ii);
  return 1;
} else return 0;
}


int fr2d::mkF(matrix & LP_c, constraint * f) {

// construct the vector, 'c' in the problem
// minimize c^T x subject to Ax <= b
//

int i, j;

for(i=0; i < f->number; i++) {
  j = i+1;
  LP_c[i]   =  bars[j].length;
  // printf(" in mkF c[%d] = %lf \n", i, LP_c[i]);
}

return 0;
}


int fr2d::mkG(matrix & LP_A, matrix & LP_b, constraint * g, int
n_var, double * pda) {
```

```
/* construct the A and b in
 * minimize c^T x subject to Ax <= b
 *
 * members of the constraint structure are:
 * variablecontentsexampledata type
 * ---------------------------------
 * item quantity dl, dx, ...char
 * location member / node 1, 34, ..int
 * typetype UPPER / LOWERint (+/- 1)
 * val limit0.05, -120double
 *
 * Variable 'pda' is the previous DA increment and should be used
 * in an iterative solution to approximate the second order terms.
 * This is done by using an estimate of DA (given in 'pda') for one
 * of the two terms in the second order part, '1/2 d2K/da2 da^2'.
 *
 */

int b, i, ii, ikn, j, k, n_con;
double D1, D2, D3, dK[3][3], tempd;
matrix NKN(3*NB, 3*NB), KN(dof, 3*NB), phi(3*NB), Del(3*NB),
delta(dof);
/* 'phi' is a column vector representation of the sparse matrix,
 * dK/dA * Del. Each column of this sparse matrix has three ele-
ments
 * corresponding to dK/dA_i * Del.
 */

// printf(" begin mkG \n");
/* assemble:
 * -[N Kinv N~]
 * for general constraints and put this entire product into 'NKN'
 */
        if( g->isDefined('L') || g->isDefined('P') || g->isDe-
fined('N') )
          NKN = -N_sys * Kinv * N_sys.transpose();
        if( g->isDefined('X') || g->isDefined('Y') )
          KN = -Kinv * N_sys.transpose();
n_con = g->number;

for(i=0; i<3; i++)
  for(j=0; j<3; j++)
dK[i][j] = 0;

// if(pda != NULL) printf("fr2d::mkG using second order terms\n");
// else printf("fr2d::mkG using first order terms\n");

phi.setZero();
/* construct 'phi' */
for(i=0; i<NB; i++) {
  b = i+1;
  /* derivative of the primitive stiffness matrix for each bar */
  dK[0][0] =bars[b].E / bars[b].length;
  dK[1][1] =4 * bars[b].E * bars[b].A2I[0] * bars[b].A2I[1] *
pow(bars[b].area, bars[b].A2I[1]-1) / bars[b].length;
  if(pda != NULL) {
    // the second order term is "1/2 d2/dA2 K * dA^2" but using
linear
    // approximation so the previous dA, 'pda' is being used to
make
    // up the other part of dA^2.
    dK[1][1] += 2*bars[b].E * bars[b].A2I[0] * bars[b].A2I[1] *
(bars[b].A2I[1]-1) *
  pow(bars[b].area, (bars[b].A2I[1]-2)) / bars[b].length * pda[b-
1];
  }
  dK[2][2] =dK[1][1];
  dK[1][2] =dK[1][1] * 0.5 ;
  dK[2][1] =dK[1][2];

  for(ii=0; ii<3; ii++) {
    phi[3*i+ii] = dK[ii][0] * bars[b].dl
     + dK[ii][1] * bars[b].rp;
     + dK[ii][2] * bars[b].rn;
  }
}

/* fill in the LP matrix */
LP_A.setZero();
for(i=0; i<n_con; i++) {
  /* b is the location (e.g., bar number) */
  b = g->loc[i];
  /* member length change constraints (dl) */
  if(g->item[i] == 'L') {
    LP_b[i] = g->type[i] * (g->val[i] - bars[b].dl);
    for(j=0; j<n_var; j++) {
```

```
      /* Here k is the first element of 3 in a block corresponding
to
       * variable j. This requires that the variables are ordered
and
       * inclusive because k is a function of j.
       */
      for(ii=0; ii<3; ii++) {
        k = 3*j + ii;
LP_A(i,j) += NKN(3*(b-1),k) * phi[k];
      }
      LP_A(i,j) = LP_A(i,j) *  g->type[i];
    }
  /* member end rotation constraints (positive end) */
  } else if(g->item[i] == 'P') {
    LP_b[i] = g->type[i] * (g->val[i] - bars[b].rp);
    for(j=0; j<n_var; j++) {
      for(ii=0; ii<3; ii++) {
        k = 3*j + ii;
LP_A(i,j) += NKN(3*(b-1)+1,k) * phi[k];
      }
      LP_A(i,j) = LP_A(i,j) * g->type[i];
    }
  /* member end rotation constraints (negative end) */
  } else if(g->item[i] == 'N') {
    LP_b[i] = g->type[i] * (g->val[i] - bars[b].rn);
    for(j=0; j<n_var; j++) {
      for(ii=0; ii<3; ii++) {
        k = 3*j + ii;
LP_A(i,j) += NKN(3*(b-1)+2,k) * phi[k];
      }
      LP_A(i,j) = LP_A(i,j) * g->type[i];
    }
          /* joint displacement constraints (dx) */
  } else if(g->item[i] == 'X') {
    ikn = 3*(b-1);                    // index in KN is dof, not
node number
    LP_b[i] = g->type[i] * (g->val[i] - nodes[b].dx);
    for(j=0; j<n_var; j++) {
      for(ii=0; ii<3; ii++) {
        k = 3*j + ii;
LP_A(i,j) += KN(ikn,k) * phi[k];
      }
      LP_A(i,j) = LP_A(i,j) * g->type[i];
    }
          /* joint displacement constraints (dy) */
  } else if(g->item[i] == 'Y') {
    ikn = 3*(b-1)+1;                  // index in KN is dof, not
node number
    LP_b[i] = g->type[i] * (g->val[i] - nodes[b].dx);
    for(j=0; j<n_var; j++) {
      for(ii=0; ii<3; ii++) {
        k = 3*j + ii;
LP_A(i,j) += KN(ikn,k) * phi[k];
      }
      LP_A(i,j) = LP_A(i,j) * g->type[i];
    }

  /* member size constraints (M) */
  } else if(g->item[i] == 'M' ) {
    LP_b[i] = g->type[i] * (g->val[i] - bars[b].area);
    // printf("i %d, Lii %d,  b[i] %f \n", i, Lii, b[Lii]);
    // for(j=0; j<n_var; j++) LP_A(i,j) = 0.0;
    // this assumes the variables are inclusive and ordered
    LP_A(i,b-1) = g->type[i] * 1.0;

  /* global buckling */
  } else if(g->item[i] == 'G') {
    // A nonlinear analysis must have been performed and the first
    // mode shape computed. This mode shape is used as the defomed
    // shape and the corresponding member displacements are com-
puted.
    // Here the first mode shape is in the last column.
    //
    // cout << "mkG:: modeShapes\n" << modeShapes << endl;
    // cout << "mkG:: delta\n" << delta << endl;
    // When using matrix::nroot(), the modeShapes are really the
    // transpose; i.e., vectors stored in rows, not columns.
    for(ii=0; ii<dof; ii++) delta[ii] = modeShapes(dof-1, ii);
    nonlinearNdelta(Del, delta);
    // cout << "mkG:: delta (mode shape)\n" << delta << endl;
    // cout << "mkG:: Del\n" << Del << endl;
    LP_b[i] = 0.0;
    for(j=0; j<n_var; j++) {
      k = 3*j;
      b = j+1;
      tempd = Del[k]*Del[k]
```

```
        + 4*bars[b].A2I[0]*bars[b].A2I[1]*pow(bars[b].area,
bars[b].A2I[1]-1)
* (Del[k+1]*Del[k+1] + Del[k+1]*Del[k+2] + Del[k+2]*Del[k+2]);
      if(pda != NULL) {
        tempd += 2*bars[b].A2I[0]*bars[b].A2I[1]*(bars[b].A2I[1]-
1)
        * pow(bars[b].area, bars[b].A2I[1]-2)
* (Del[k+1]*Del[k+1] + Del[k+1]*Del[k+2] + Del[k+2]*Del[k+2]);
      }
      // cout << i << " " << j << " " << tempd << " " << k << " "
      //  << bars[b].A2I[0] << " " << bars[b].A2I[1] << endl;
      LP_A(i,j) = -tempd * bars[b].E / bars[b].length;
    }
  } else {
    killme("fr2d::mkG, bad constraint item");
  }
}

// free memory
NKN.del();
phi.del();

return 0;
}


int fr2d::mkKei(double SK[3][3], int bar) {
// Construct the primitive stiffness matrix for bar 'b'
int i, j;
double dist, cof;

for(i=0; i<3; i++)
  for(j=0; j<3; j++)
    SK[i][j] = 0.0;

if(nonlinear) {
  beamColumnStiffness(bar, dist, cof);
} else {
  dist = 4*bars[bar].E * bars[bar].section_I / bars[bar].length;
  cof  = dist / 2.0;
}
SK[0][0] = bars[bar].E * bars[bar].area / bars[bar].length;
SK[1][1] = dist;
SK[2][2] = SK[1][1];
SK[1][2] = cof;
SK[2][1] = SK[1][2];

return 0;
}


int fr2d::mkKgi3(double kg11[3][3], double kg12[3][3], double
kg22[3][3], int b) {
// Construct the three submatrices for Kg in the element lcs.
double c1;

c1 = bars[b].Fx / 30.0;
kg11[1][1] =  c1 * 36.0 / bars[b].length;
kg11[1][2] = -c1 * 3.0;
kg11[2][1] =  kg11[1][2];
kg11[2][2] =  c1 * 4.0 * bars[b].length;

kg12[1][1] = -kg11[1][1];
kg12[1][2] =  kg11[1][2];
kg12[2][1] = -kg11[2][1];
kg12[2][2] = -c1 * bars[b].length;

kg22[1][1] =  kg11[1][1];
kg22[1][2] = -kg11[1][2];
kg22[2][1] = -kg11[2][1];
kg22[2][2] =  kg11[2][2];

return 0;
}


int fr2d::mkKgi(double SKG[3][3], int b) {
// SUBROUTINE GSTIFF(T,AMP,AMM,AL,R,A)
// Construct the contribution of bar 'b' to Kg

double T, V, ANX,ANY;

T = bars[b].Fx;
V = -(bars[b].Mp + bars[b].Mn) / bars[b].length;
ANX = cos(bars[b].theta);
ANY = sin(bars[b].theta);
```

```
/*
SKG[0][0] = (T*(1.0-ANX*ANX) + V*2.0*ANX*ANY) / bars[b].length;
SKG[0][1] = (-T*ANX*ANY - V*(1.0-2.0*ANY*ANY)) / bars[b].length;
SKG[1][0] = SKG[0][1];
SKG[1][1] = (T*(1.-ANY*ANY) - V*2.0*ANX*ANY) / bars[b].length;
*/
// try this (not symmetric)
SKG[0][0] = (T*(1.0-ANX*ANX) + V*ANX*ANY) / bars[b].length;
SKG[0][1] = (-T*ANX*ANY - V*(1.0-ANY*ANY)) / bars[b].length;
SKG[1][0] = (-T*ANX*ANY + V*(1.0-ANX*ANX)) / bars[b].length;
SKG[1][1] = (T*(1.0-ANY*ANY) - V*ANX*ANY) / bars[b].length;

return 0;
}


int fr2d::mkProps(int bar) {
/*
 * Compute the geometric properties of element number 'bar'.
 * Properties computed are
 * angle of the bar's alignment (in radians!!)
 * the bar's length
 * constants c0 and c1 that relate bar area and
 * its section modulus according to
 * I = c0 * A ^ c1
 */

int n1, n2;
double dx, dy;

n1 = bars[bar].NM;
n2 = bars[bar].NP;

bars[bar].length = 0.0;
dx = nodes[n2].x - nodes[n1].x;// delta x
dy = nodes[n2].y - nodes[n1].y;
bars[bar].length = dx*dx + dy*dy;
bars[bar].length = sqrt(bars[bar].length);  // length

if(bars[bar].length == 0) {
  printf(" bar %d, n1: %d, n2: %d \n", bar, n1, n2);
  killme("fr2d::mkProps, zero length bar", bar);
}

bars[bar].theta = atan2(dy, dx);

/* Right now, using c1 = 2 (quadratic) for area <=> section_I */
bars[bar].A2I[1] = 2.0;
bars[bar].A2I[0] = bars[bar].section_I/pow(bars[bar].area,
bars[bar].A2I[1]);

// printf(" bar %d, c0 %f, c1 %f\n", bar, bars[bar].A2I[0],
bars[bar].A2I[1]);
// printf(" bar %d, dx %f, dy %f, theta %f\n", bar, dx, dy,
bars[bar].theta);

return 0;
}


int fr2d::mkRN(double NRP[3][3], double NRM[3][3], int bar) {
/*
 * Create the rotated transformation matrices NRP and NRM for
 * the 'bar' member and the rotation matrix 'R'.
 *
 */

double R[3][3], SNP[3][3], SNM[3][3];
int i, j, k;

for(i=0; i<3; i++) {
  for(j=0; j<3; j++) {
SNP[i][j] = 0.0;
SNM[i][j] = 0.0;
  }
}

mkRotationMatrix(R, bar);

/* transformation matrix, N, for (+) node */
SNP[0][0] = 1.0;
SNP[1][2] = 1.0;
SNP[1][1] =-1.0 / bars[bar].length;
SNP[2][1] =SNP[1][1];

/* transformation matrix, N, for (-) node */
```

```
SNM[0][0] = -1.0;
SNM[2][2] = 1.0;
SNM[1][1] =1.0 / bars[bar].length;
SNM[2][1] =SNM[1][1];

/* transform from local coord. sys. to global coord. sys. */
for(i=0; i<3 ; i++) {
  for(j=0; j<3 ; j++) {
  NRP[i][j]=0.0;
  NRM[i][j]=0.0;
    for(k=0; k<3 ; k++) {
NRP[i][j] += SNP[i][k]*R[k][j];
NRM[i][j] += SNM[i][k]*R[k][j];
    }
  }
}

return 0;
}


int fr2d::mkRotationMatrix(double R[3][3], int bar) {
/*
 * Create the rotation matrix 'R' for member
 * number 'bar'.
 *
 */

int i, j;

for(i=0; i<3; i++) for(j=0; j<3; j++) R[i][j] = 0.0;

/* rotation matrix */
R[0][0] =  cos( bars[bar].theta );
R[1][1] =  R[0][0];
R[0][1] =  sin( bars[bar].theta );
R[1][0] = -R[0][1];
R[2][2] =  1.0;

return 0;
}
```

```
int fr2d::nonlinearNdelta(matrix & DDD, matrix ddd, char* update-
Flag) {
// SUBROUTINE RESET(P,R,NP,MI,TH,FAC,NNS,AL,DL,ALPHP,ALPHM,PI)
// This provides the 'exact' version of equation
// Delta = N * delta
// relating the member displacements to the joint displacements.
Also
// the member alignment 'theta' is updated if variable 'update-
Flag'
// is equal to "ROTATION". At this point, length is not updated
(i.e.,
// small strains) but if added later, this flag would indicate
when to do
// so.
//
double DX,DY,ROT,DAL,DXR,DYR;
int b, nm, np;

for(b=1; b<=NB; b++) {
DX = 0.0;
DY = 0.0;
DDD[3*b-2] = 0.0;
DDD[3*b-1] = 0.0;
np = bars[b].NP;
nm = bars[b].NM;

if(np <= freeNodes) {
  DX = ddd[3*np-3];
  DY = ddd[3*np-2];
  DDD[3*b-2] = ddd[3*np-1];
}

if(nm <= freeNodes) {
  DX -= ddd[3*nm-3];
  DY -= ddd[3*nm-2];
  DDD[3*b-1] = ddd[3*nm-1];
}

DXR =  cos(bars[b].theta)*DX + sin(bars[b].theta)*DY;
DYR = -sin(bars[b].theta)*DX + cos(bars[b].theta)*DY;
DAL = bars[b].length;
```

```
// cout << "nonlinearNdelta: b " << b << " dxyr " << DXR << " " <<
DYR << endl;
ROT = atan2(DYR, DAL+DXR);
DDD[3*b-3] = sqrt((DAL+DXR)*(DAL+DXR) + DYR*DYR) - DAL;
DDD[3*b-2] -= ROT;
DDD[3*b-1] -= ROT;
// TH += ROT/PI;
if(!strcmp(updateFlag, "THETA")) {
  // printf("fr2d::nonlinearNdelta: updating theta\n");
  bars[b].theta += ROT;
}

// printf("nonlinNdel %d, %f \t%f \t%f\n", b, bars[b].dl,
bars[b].rp, bars[b].rn);
}

return 0;
}


int fr2d::optimizeOC(double * lowLim, double * uppLim) {

double allowable = 0.00001;

/*
**********************
for(int i=1; i<=NB; i++)
{
  if(bars[i].dl > 0)  {
bars[i].area =
fabs(bars[i].force) * bars[i].length /
(bars[i].E * conUpp.bars[i].dl);
  } else { bars[i].area =
fabs(bars[i].force) * bars[i].length /
(bars[i].E * fabs(conLow.bars[i].dl) );
  }
}
************************
*/
return 0;
}
```

```
int fr2d::prod3T(double A[3][3], double B[3][3], double C[3][3],
double D[3][3]) {

// multiply A^T x B x C and return in D

int i, ii, j, k, L;

for(i=0; i<3 ; i++) { // form product
  for(j=0; j<3 ; j++) {
  D[i][j]=0.0;
    for(k=0; k<3 ; k++)
for(L=0; L<3; L++)
D[i][j] = D[i][j] + A[k][i]*B[k][L]*C[L][j];
  }
}

return 0;
}


int fr2d::readFile(char* fileName)
{
int i;
fstream inPut;

// fileName="file.dat";
inPut.open(fileName, ios::in);  // open the file
if(inPut.fail())  {
  cout << "***Error::readFile file: '"
<< fileName << "' not opened. \n";
  exit(-1);
}

inPut >> NB >> NN >> NS;
for (i=1; i<=NN; i++) // nodal coordinates and loads
  inPut >> nodes[i].x >> nodes[i].y >>
nodes[i].Px >> nodes[i].Py >> nodes[i].Mz ;

for (i=1; i<=NB; i++) // member properties
  inPut >> bars[i].NM >> bars[i].NP >>
bars[i].area >> bars[i].section_I >> bars[i].E;

inPut.close();  // close the file
```

```
// cout << "done reading file: " << fileName <<  "\n";

initialize();

return 0;
}


int fr2d::rmSmallBars(double min) {

int i, j, remove;
remove = 0;
for(i=1; i<=NB; i++)
{
  if (bars[i].area < min) {
    // Should check if resulting structure will still be
    // stable before removing this bar.
    for(j=i; j<=NB; j++) { // shift properties of previous
bars[j].E    = bars[j+1].E;
bars[j].NM   = bars[j+1].NM;
bars[j].NP   = bars[j+1].NP;
bars[j].area = bars[j+1].area;
    }
    NB--;
    remove++;
  }
}

return remove;

}


int fr2d::writeFile(char* fileName) {
/*
 * Write file in a format that matches the input file
 * format.
 *
 */

int i;
FILE *outfile;
if( (outfile = fopen(fileName, "w")) == NULL ) {
  printf(" ***Error::writeFile: '%s' not opened\n", fileName);
  return 1;
}

fprintf(outfile, "%5d%5d%5d\n", NB, NN, NS);

/* nodal coordinates and loads */
for (i=1; i<=NN; i++)
fprintf(outfile, "%14.5f %14.5f %14.5f %14.5f %14.5f\n",
nodes[i].x, nodes[i].y,
nodes[i].Px, nodes[i].Py, nodes[i].Mz);

/* member properties */
for (i=1; i<=NB; i++)
fprintf(outfile, "%5d%5d %14.5e %14.5e %19.5e\n",
bars[i].NM, bars[i].NP,
bars[i].area, bars[i].section_I, bars[i].E );

fclose(outfile);
// printf("done writing file: %s\n", fileName);

return 0;
}


int fr2d::writeResults(char* fileName) {

int i;
fstream outPut;

// fileName="file.rst";
outPut.open(fileName, ios::out);  // open the file

outPut  << "bars: \t"  << setw(5) << NB << "\n"
<< "nodes: \t" << setw(5) << NN << "\n"
<< "suppt: \t" << setw(5) << NS << "\n";

outPut  << setprecision(3)
<< setw(15)
<< setiosflags(ios::showpoint | ios::fixed);

outPut  << "\n\n coordinates \t\t loads \t\t displacements\n";
```

```
for (i=1; i<=NN; i++)// nodal coordinates and loads
{
outPut  << setw(15) << nodes[i].x
<< setw(15) << nodes[i].y
<< setw(15) << nodes[i].Px
<< setw(15) << nodes[i].Py
<< setw(15) << nodes[i].Mz
<< setw(15) << nodes[i].dx
<< setw(15) << nodes[i].dy
<< setw(15) << nodes[i].rz << "\n";
}

outPut  << "\n\n NM \t NP \t A t I \t L \t force \t stress-ax \t dl
\n";
for (i=1; i<=NB; i++)// member properties
{
outPut  << setprecision(3)
<< setw(5)  << bars[i].NM
<< setw(5)  << bars[i].NP
<< setw(10) << bars[i].area
<< setw(10) << bars[i].section_I
<< setprecision(2)
<< setw(6)  << bars[i].length
<< setw(9)  << bars[i].Fx
<< setw(9)  << bars[i].sigAxial
<< setprecision(5)
<< setw(9)  << bars[i].dl << "\n";
}

outPut.close();  // close the file
// cout << "done writing file: " << fileName <<"\n";

return 0;
}

int fr2d::writeResults(void) {
/*
 * Write the nodal and member displacements
 *
 */

echoJointDisplacements();
echoMemberDisplacements();
echoMemberForces();

return 0;
}

int fr2d::zeroDisplacements(void) {
int i;

for (i=1; i<=NN; i++) {
  nodes[i].dx = 0.0;
  nodes[i].dy = 0.0;
  nodes[i].rz = 0.0;
}

for (i=1; i<=NB; i++) {
  bars[i].dl = 0.0;
  bars[i].rp = 0.0;
  bars[i].rn = 0.0;
}

return 0;
}

int fr2d::zeroForces(void) {
int i;

for (i=1; i<=NB; i++) {
  bars[i].Fx = 0.0;
  bars[i].Mp = 0.0;
  bars[i].Mn = 0.0;
}

return 0;
}

/************************************
 *
 * class methods -- methods that return double and double *
 *
 */
```

```
double fr2d::a2i(int bar) {
/* Given the bar number, 'bar', return the section modulus, I
 * according to the relation
 * I = c0 * A^c1
 * where c0 and c1 are variables in the array 'A2I' of the
 * frameMember structure. These constants should be set with
 * a call to 'mkProps'.
 */

double I;

I = bars[bar].A2I[0] * pow(bars[bar].area, bars[bar].A2I[1]);
// printf("A^2 %f\n", pow(bars[bar].area, bars[bar].A2I[1]) );

return I;
}

double fr2d::getUnbalancedForce(matrix & p) {
// Compute the out of balance forces for the applied load, 'p' and
// return this vector in 'p'. Used for nonlinear analysis.
int b, i, j;
double sum;
matrix f(3*NB);

for(b=1; b<=NB; b++) {
  f[3*b-3] = bars[b].Fx;
  f[3*b-2] = bars[b].Mp;
  f[3*b-1] = bars[b].Mn;
}
// cout << "getUnbalancedForce: f = \n" << f << endl;
p = p - N_sys.transpose() * f;

sum = 0.0;
for(i=0; i<dof; i++) sum += p[i]*p[i];
sum = sqrt(sum);

f.del();
return sum;
}

double fr2d::scale(constraint * g, double factor) {

int i, b, govern;
double scale = SMALL;
// double temp = 1.0;
double temp = SMALL;

/* Linear scaling will change magnitude but not sign so
 * perform scaling only if delta and delta_allowable
 * have the same sign. Variable 'factor' should be equal
 * to 1.0 for "at limit" condition and greater than 1.0
 * for "below limit" condition.
 */

/* If global buckling, scale for only by 'factor' at this point.
Scaling
 * to be at the buckling load is done iteratively in the main pro-
gram
 * using the eigenvalue.
 */
if(g->findItem('G')) {
  // Make sure it's a nonlinear analysis in case this hasn't been
set
  // and scale by 'factor', then return.
  if(!nonlinear) killme("fr2d::scale buckling analysis should be
nonlinear\n");
  for(i=1; i <= NB; i++) {
    bars[i].area *= factor;
    bars[i].section_I = a2i(i);
  }
  return factor;
}

/* It is assumed here that member end rotations and joint dis-
placements
 * are not linearly related to the area. This is because 'fr2d'
elements
 * support flexure, which is a function of 'I_mod' and this is
related to
 * 'area' by the exponent 'A2I[1]'. The member length change is
assumed
 * to be linearly related to the area however.
 */
for(i=0; i < g->number; i++) {
  b = g->loc[i];
  // printf(" fr2d::scale, constraint %d \n", i);
```

```
  /* member length change (dl) */
  if(g->item[i] == 'L') {
    if (sign(g->val[i]) == sign(bars[b].dl) ) {
// printf("scale::constr %d on dl; same sign\n", i);
temp = bars[b].dl / g->val[i];
    }
  /* member end rotations (positive end) */
  } else if(g->item[i] == 'P') {
    if (sign(g->val[i]) == sign(bars[b].rp) ) {
// printf("scale::constr %d on rp; same sign\n", i);
temp = bars[b].rp / g->val[i];
// temp = sqrt(temp);
temp = pow(temp, 1.0/bars[b].A2I[1]);
    }
  /* member end rotations (negative end) */
  } else if(g->item[i] == 'N') {
    if (sign(g->val[i]) == sign(bars[b].rn) ) {
// printf("scale::constr %d on rn; same sign\n", i);
temp = bars[b].rn / g->val[i];
temp = pow(temp, 1.0/bars[b].A2I[1]);
    }
  /* joint displacement (dx) */
  } else if(g->item[i] == 'X') {
    if (sign(g->val[i]) == sign(nodes[b].dx) ) {
// printf("scale::constr %d on dx; same sign\n", i);
temp = nodes[b].dx / g->val[i];
temp = pow(temp, 1.0/bars[b].A2I[1]);
    }
  /* joint displacement (dy) */
  } else if(g->item[i] == 'Y') {
    if (sign(g->val[i]) == sign(nodes[b].dy) ) {
// printf("scale::constr %d on dl; same sign\n", i);
temp = nodes[b].dy / g->val[i];
temp = pow(temp, 1.0/bars[b].A2I[1]);
    }
  } else if(g->item[i] == 'M') {
    // This should be handled carefully because there could be
    // conflicts here. Minimum is easy but maximum may cause an
    // infeasible problem. Ignore upper limits for now.
    if(g->type[i] == -1) temp = g->val[i] / bars[b].area;
  /* default */
  } else {
    // killme("fr2d::scale unknown constraint", i+1);
    killme("fr2d::scale unknown constraint", g->item[i], i+1);
  }

  if(temp > scale) {
    scale = temp;
    govern = i;
  }
  // printf("\tc: %d item: '%c' ratio: %8.5f scale: %8.5f\n", i,
g->item[i], temp, scale);
}

// printf("fr2d::scale: constraint %d governs\n", govern);
// printf("\t scale: \t%10.5f x fact = %10.5f\n", scale, fac-
tor*scale);

for(i=1; i <= NB; i++) {
  bars[i].area *= scale*factor;
  bars[i].section_I = a2i(i);
  // printf("fr2d::scale bar %d, A %f\n", i, bars[i].area);
}

return scale*factor;
}


double fr2d::volume() {

double vol = 0.0;

for(int i=1; i<=NB; i++)
  vol += bars[i].area * bars[i].length;

return vol;
}
```

```
File: fr3d.cpp
/**************************************************
 * fr3d.cpp: implementation of the fr3d class.
 *
 * two-dimensional nonlinear, elastic frame
 *
 */

#include <fstream.h> // contains fstream
#include <iomanip.h> // formatted I/O
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>

#include "constraint.h"
#include "complex.h"// for beam column stiffness
#include "fr3d.h"
#include "matrix.h"
#include "utility.h"

#define SMALL 0.00000001
#define PI 3.14159265358979
#define TIMING 0


/***********************************
 *
 *  Construction/Destruction
 *
 */

/*
fr3d::fr3d()
{

}
*/


fr3d::fr3d(int * dim) : Ke( dim[2], dim[2] ),
Kg(1, 1),
modeShapes(1, 1),
frequencies(1, 1),
Kinv( dim[2], dim[2] ),
N_sys(6*dim[0], dim[2] )
{
/* Variable 'dim' has three elements from a "quick peek" at the
 * file. The elements, in order are NB, NN, and NS (number of bars,
 * nodes, and supports). This gives enough to allocate space for
 * the entire structural analysis.
 *
 * The text above after the colon must be there to allocate space
 * for the matrix objects.
 */


// printf("dim: %d, %d, %d\n", dim[0], dim[1], dim[2] );
NB = dim[0];
NN = dim[1];
// NS = dim[2];
dof = dim[2];

freeNodes = NN - NS;/* number of free nodes */
// dof = 6*freeNodes;/* total degrees of freedom */
nonlinear = false;/* default value is false */

/* nodes and bars are one-based (i.e., first element is index 1,
not 0) */
nodes = new node[NN+1];
bars  = new frameMember[NB+1];

}


fr3d::fr3d(int * dim, char * nlflag) : Ke( dim[2], dim[2] ),
Kg( dim[2], dim[2] ),
modeShapes(dim[2], dim[2] ),
frequencies(dim[2], 1 ),
Kinv( dim[2], dim[2] ),
N_sys(6*dim[0], dim[2] )

{
/* Variable 'dim' has three elements from a "quick peek" at the
 * file. The elements, in order are NB, NN, and NS (number of bars,
 * nodes, and supports). This gives enough to allocate space for
```

```
 * the entire structural analysis. Variable 'nlflag' is not used
but
 * required to determine if space should be allocated fo Kg->
 *
 * The text above after the colon must be there to allocate space
 * for the matrix objects.
 */


// printf("dim: %d, %d, %d\n", dim[0], dim[1], dim[2] );
NB = dim[0];
NN = dim[1];
// NS = dim[2];
dof = dim[2];

freeNodes = NN - NS;/* number of free nodes */
// dof = 6*freeNodes;/* total degrees of freedom */
nonlinear = true;

/* nodes and bars are one-based (i.e., first element is index 1,
not 0) */
nodes = new node[NN+1];
bars  = new frameMember[NB+1];

}


fr3d::~fr3d()
{

}


/************************************
 *
 * class methods -- methods that return int
 *
 */

int fr3d::analyze() {

/* Basic linear elastic analysis */
int i, j, k, ii, status;
double SK[6][6];
matrix P(dof, 1), delta(dof, 1), Delta(6*NB, 1);

/* set up system matrix and inverse */
if(TIMING) showTime("fr3d::analyze begin");
assembleKe();
if(TIMING) showTime("fr3d::analyze done forming Ke");

/* set up load vector */
i = 1;
k = 0;
while (k < dof) {
  if(!nodes[i].fix[0]) P[k++] = nodes[i].Px;
  if(!nodes[i].fix[1]) P[k++] = nodes[i].Py;
  if(!nodes[i].fix[2]) P[k++] = nodes[i].Pz;
  if(!nodes[i].fix[3]) P[k++] = nodes[i].Mx;
  if(!nodes[i].fix[4]) P[k++] = nodes[i].My;
  if(!nodes[i].fix[5]) P[k++] = nodes[i].Mz;
  i++;
}
if(TIMING) showTime("fr3d::analyze done forming load vector");

/* solve system */
// cout << "fr3d::analyze Ke\n" << Ke << endl;
// cout << "fr3d::analyze P\n" << P << endl;
status = matrix::solve(Ke, P, delta);
if(TIMING) showTime("fr3d::analyze done solving system");
// cout << "fr3d::analyze delta\n" << delta << endl;

/* set displacements from solution */
i = 1;
k = 0;
while (k < dof) {
  if(!nodes[i].fix[0]) nodes[i].dx = delta[k++];
  if(!nodes[i].fix[1]) nodes[i].dy = delta[k++];
  if(!nodes[i].fix[2]) nodes[i].dz = delta[k++];
  if(!nodes[i].fix[3]) nodes[i].rx = delta[k++];
  if(!nodes[i].fix[4]) nodes[i].ry = delta[k++];
  if(!nodes[i].fix[5]) nodes[i].rz = delta[k++];
  i++;
}
if(TIMING) showTime("fr3d::analyze done set displacements");
```

```
/*  compute member forces and displacements */
Delta = N_sys * delta;
if(TIMING) showTime("fr3d::analyze done 'D = N d'");
// cout << " N_sys is \n" << N_sys << endl;
// cout << " Delta is \n" << Delta << endl;
// cout << " delta is \n" << delta << endl;
for (i=1; i<=NB; i++) {
  bars[i].dl     = Delta[6*i-6];
  bars[i].twist = Delta[6*i-5];
  bars[i].rpy    = Delta[6*i-4];
  bars[i].rpz    = Delta[6*i-3];
  bars[i].rny    = Delta[6*i-2];
  bars[i].rnz    = Delta[6*i-1];
  mkKei(SK, i);
  bars[i].Fx = bars[i].dl*SK[0][0];
  bars[i].Torque = bars[i].twist*SK[1][1];
  bars[i].Mpy = bars[i].rpy*SK[2][2] + bars[i].rny*SK[2][4];
  bars[i].Mny = bars[i].rpy*SK[4][2] + bars[i].rny*SK[4][4];
  bars[i].Mpz = bars[i].rpz*SK[3][3] + bars[i].rnz*SK[3][5];
  bars[i].Mnz = bars[i].rpz*SK[5][3] + bars[i].rnz*SK[5][5];
  // bars[i].sigAxial = bars[i].Fx / bars[i].area;
}
if(TIMING) showTime("fr3d::analyze done member forces ");

// This single-step nonlinear analysis is used only to form the
// matrix Kg and check for buckling. The displacements are com-
puted
// from the linear-elastic analysis above and not changed here.
if(nonlinear) {
  matrix KeKg(dof, dof);
  assembleKg();
  if(TIMING) showTime("fr3d::analyze done forming Kg ");
  KeKg = Ke + Kg;
  status = matrix::solve(KeKg, P, delta);
  if(TIMING) showTime("fr3d::analyze done solving system ");
  if(status) {
    // printf(" --- fr3d::analyze() negative term on main diago-
nal\n\n");
    // free memory before exit
    delta.del();
    Delta.del();
    KeKg.del();
    return status;
  }
  // no buckling but still want to free memory
  KeKg.del();
}

/* free memory */
P.del();
delta.del();
Delta.del();

return status;
}


int fr3d::analyze(int nLoadSteps, int nIterations) {
/* When the analyze method is called with two integer arguments,
 * it is taken to be a nonlinear analysis. This will apply the
 * total load in 'nLoadSteps' and each load step will be solved
 * with 'nIterations' for equilibrium.
 *
 * Method returns zero if all loadsteps and iterations are com-
pleted
 * sucessfully, returns number of loadsteps if solver fails (non-
positive
 * term on main diagonal).
 */

// For zero or one load steps, the analysis is nonlinear
// but the displacements are small thus call 'analyze()'.
if(nLoadSteps <= 1) return analyze();

int i, j, k, ii, iter, lstp;
double SK[6][6];
double error, loadFactor, thy_[NB+1];
matrix P(dof, 1), delta(dof, 1), Delta(6*NB, 1), KeKg(dof, dof);

// During the nonlinear analysis the member orientation, 'thy'
// will be updated to reflect the deformed geometry. Save the
// initial values here and restore at the end of the analysis.
// printf("fr3d::analyze(int, int): begin\n");
for(i=1; i<=NB; i++)  thy_[i] = bars[i].thy;
zeroDisplacements();
zeroForces();
```

```
////////////////////////////////////////////////////
//
// iterations
//
for(lstp=1; lstp <= nLoadSteps; lstp++) {
// printf("fr3d::analyze(int, int): loadstep %d\n", lstp);
for(iter=0; iter < nIterations; iter++) {
// printf("fr3d::analyze(int, int): loadstep %d, iter %d\n", lstp,
iter);

/* Set up system matrices Ke, Kg, and Kinv */
assembleKe();
assembleKg();

/* set up load vector */
i = 1;
k = 0;
loadFactor = ((double)lstp) / ((double)nLoadSteps);
while (k < dof) {
  if(!nodes[i].fix[0]) P[k++] = loadFactor * nodes[i].Px;
  if(!nodes[i].fix[1]) P[k++] = loadFactor * nodes[i].Py;
  if(!nodes[i].fix[2]) P[k++] = loadFactor * nodes[i].Pz;
  if(!nodes[i].fix[3]) P[k++] = loadFactor * nodes[i].Mx;
  if(!nodes[i].fix[4]) P[k++] = loadFactor * nodes[i].My;
  if(!nodes[i].fix[5]) P[k++] = loadFactor * nodes[i].Mz;
  i++;
}

// Apply only the out-of-balance load
// P = P - N_sys * F
// which will yield an incremental displacement
//
error = getUnbalancedForce(P);
// printf(" fr3d::analyze: step %d, iter %d, error %f\n", lstp,
iter, error);
// cout << "applied load\n" << P << endl;

/* solve system */
KeKg = Ke + Kg;
// cout << "in analyze, Ke: \n" << Ke << " Kg: \n" << Kg << endl;
if(matrix::solve(KeKg, P, delta)) {
  // printf(" --- fr3d::analyze(int, int) negative term on main
diagonal\n\n");
  // restore values and free memory before exit
  for(i=1; i<=NB; i++) bars[i].thy = thy_[i];
  delta.del();
  Delta.del();
  P.del();
  KeKg.del();
  return lstp;
}

/* set displacements from solution */
i = 1;
k = 0;
while (k < dof) {
  if(!nodes[i].fix[0]) nodes[i].dx += delta[k++];
  if(!nodes[i].fix[1]) nodes[i].dy += delta[k++];
  if(!nodes[i].fix[2]) nodes[i].dz += delta[k++];
  if(!nodes[i].fix[3]) nodes[i].rx += delta[k++];
  if(!nodes[i].fix[4]) nodes[i].ry += delta[k++];
  if(!nodes[i].fix[5]) nodes[i].rz += delta[k++];
  i++;
}

/*  compute member forces and incremental displacements */
// nonlinearNdelta(Delta, delta, "THETA");
Delta = N_sys * delta;
// cout << "analyze: joint displacement increment: \n" << delta <<
endl;
// cout << "analyze: member displacement increment: \n" << Delta
<< endl;
for (i=1; i<=NB; i++) {
  bars[i].dl    += Delta[6*i-6];
  bars[i].twist += Delta[6*i-5];
  bars[i].rpy   += Delta[6*i-4];
  bars[i].rpz   += Delta[6*i-3];
  bars[i].rny   += Delta[6*i-2];
  bars[i].rnz   += Delta[6*i-1];
  mkKei(SK, i);
  bars[i].Fx += bars[i].dl*SK[0][0];
  bars[i].Torque += bars[i].twist*SK[1][1];
  bars[i].Mpy += bars[i].rpy*SK[2][2] + bars[i].rny*SK[2][4];
  bars[i].Mny += bars[i].rpy*SK[4][2] + bars[i].rny*SK[4][4];
  bars[i].Mpz += bars[i].rpz*SK[3][3] + bars[i].rnz*SK[3][5];
  bars[i].Mnz += bars[i].rpz*SK[5][3] + bars[i].rnz*SK[5][5];
```

```
  // bars[i].sigAxial = bars[i].Fx / bars[i].area;
}

// printf("analyze: current member forces");
// echoMemberForces();
// printf("\n\nend this iteration\n\n");


}// close iteration loop
}// close loadstep loop
//
// end iterations
//
//////////////////////////////////////////////////

// Kinv = Ke.invert();

// restore values and free memory before exit
for(i=1; i<=NB; i++) bars[i].thy = thy_[i];
delta.del();
Delta.del();
P.del();
KeKg.del();

return 0;
}

int fr3d::assembleKe() {
/*
 * This method assembles the stiffness matrix, Ke by creating
 * the sub-matrix for each element and then inserting it.
 *
 * local variables (matrices):
 * ke_i contribution to Ke of a single member
 * kg_i contribution to Kg of a single member
 * NRP transformation matrix for + node of a member
 * rotated to global coordinate system
 * NRM transformation matrix for - node of a member
 * rotated to global coordinate system
 * SK primitive member stiffness matrix for a single bar
 * dK derivative of SK for a single bar
 */
```

```
int i, ii, j, row;

double ke_i[6][6], temp1[6][6];
double NRP[6][6], NRM[6][6], SK[6][6], R[6][6];
double kgi11[6][6], kgi12[6][6], kgi22[6][6];

Ke.setZero();
N_sys.setZero();

for(i=1; i<=NB; i++) {/* begin assembly of Ke*/
  /* create matrices for this element */
  mkRN(NRP, NRM, i);
  mkKei(SK, i);
  // printf("fr3d::assembleKe have SK for bar %d\n", i);

  if( numberDofAtNode(bars[i].NP) ) {
    // printf(" element %d effects node %d\n", i, bars[i].NP);
    /* transform from local to global coordinate system */
    prod6T(NRP, SK, NRP, ke_i);
    insertElement(Ke, ke_i, bars[i].NP, bars[i].NP);
    insertToNsys (N_sys, NRP,  i, bars[i].NP);

    if( numberDofAtNode(bars[i].NM) ) {
/* these are the cross terms (both nodes are free). It
 * does not affect the transformation matrix, N_sys
 */
    // printf(" el %d, nodes %d %d \n",i, bars[i].NM, bars[i].NP);
prod6T(NRP, SK, NRM, ke_i);
insertElement(Ke, ke_i, bars[i].NP, bars[i].NM);
for(ii=0; ii<6; ii++) for(j=0; j<6; j++) temp1[ii][j] =
ke_i[j][ii];
insertElement(Ke, temp1, bars[i].NM, bars[i].NP);
    }
  }

  if( numberDofAtNode(bars[i].NM) ) {
    // printf(" element %d effects node %d\n", i, bars[i].NM);
    prod6T(NRM, SK, NRM, ke_i);
// printf("ke_i for nm\n");
// for(ii=0; ii<6; ii++) for(j=0; j<6; j++) printf("%d %d %f\n",
ii, j, ke_i[ii][j]);
    insertElement(Ke, ke_i, bars[i].NM, bars[i].NM);
```

```
    insertToNsys (N_sys, NRM, i, bars[i].NM);
  }

}/* get next bar for Ke*/

// printf("invert Ke\n");
Kinv = Ke.invert();
// cout << "fr3d::assembleKe, Ke \n" << Ke << endl;
// cout << "fr3d::assembleKe, Kinv\n" << Kinv << endl;
// cout << "fr3d::assemble, N_sys\n" << N_sys << endl;
// printf("fr3d::assembleKe; done\n");

return 0;
}


int fr3d::assembleKg() {
/*
 * This method assembles the geometric stiffness matrix, Kg by
creating
 * the sub-matrix for each element and then inserting it.
 *
 */


int i, ii, j;

double kg_i[6][6], temp1[6][6], R[6][6];
double kgi11[6][6], kgi12[6][6], kgi22[6][6];

if(!nonlinear) killme("***fr3d::assembleKg request for Kg in lin-
ear analysis");
// printf("fr3d::assemble begin Kg\n");
Kg.setZero();
for(i=0; i<6; i++) {
  for(j=0; j<6; j++) {
    kg_i[i][j] = 0.0;
    kgi11[i][j] = 0.0;
    kgi12[i][j] = 0.0;
    kgi22[i][j] = 0.0;
  }
}
for(i=1; i<=NB; i++) {
  // mkKgi(kg_i, i);
  mkRotationMatrix(R, i);

  mkKgi3(kgi11, kgi12, kgi22, i);
  // printf("fr3d::assemble kg, bar %d\n", i);
  if( numberDofAtNode(bars[i].NP) ) {
prod6T(R, kgi11, R, kg_i);
insertElement(Kg, kg_i, bars[i].NP, bars[i].NP);
/*
  // Bill's version
  for(ii=0; ii<3; ii++) for(j=0; j<3; j++) kg_i[ii][j] = -
kg_i[ii][j];
  insertElement(Kg, kg_i, bars[i].NP, bars[i].NM);
  for(ii=0; ii<3; ii++) for(j=0; j<3; j++) temp1[ii][j] =
kg_i[j][ii];
  insertElement(Kg, temp1, bars[i].NM, bars[i].NP);
  for(ii=0; ii<3; ii++) for(j=0; j<3; j++) kg_i[ii][j] = -
kg_i[ii][j];
*/
if( numberDofAtNode(bars[i].NM) ) {
  prod6T(R, kgi12, R, kg_i);
  insertElement(Kg, kg_i, bars[i].NP, bars[i].NM);
  for(ii=0; ii<6; ii++) for(j=0; j<6; j++) temp1[ii][j] =
kg_i[j][ii];
  insertElement(Kg, temp1, bars[i].NM, bars[i].NP);
}
  }
  if( numberDofAtNode(bars[i].NM) ) {
    // printf(" element %d effects node %d\n", i, bars[i].NM);
    prod6T(R, kgi22, R, kg_i);
    insertElement(Kg, kg_i, bars[i].NM, bars[i].NM);
  }
}/* get next bar for Kg*/

// printf("fr3d::assembleKg; done\n");

return 0;
}

int fr3d::beamColumnStiffness(int i, double & dist, double & cof)
{
// Form beam column stiffness factors for member 'i'. This uses
```

```
// the complex number class.

complex AK,C1,C2,EKL,EMKL,AKL;
double EI;

EI = bars[i].E * bars[i].Iy;

if(EI == 0.0) {
  cof  = 0.0;
  dist = 0.0;
  return 0;
}

// if load is very far from critical use the regular 4EI/L
if(fabs(bars[i].Fx * bars[i].length * bars[i].length / EI) <
0.0001) {
  // printf("beamColumnStiffness: bar %d is far from critical\n",
i);
  cof  = 2.0 * EI / bars[i].length;
  dist = 2.0 * cof;
  return 0;
}

AK = bars[i].Fx / EI;
AK = AK.xroot();
AKL = AK * bars[i].length;
EKL = (AK*bars[i].length).xexp();
EMKL = 1.0 / EKL;

C2 = -1.0/(AK*(1.0-EMKL-(EKL-1.0)*(AKL-1.0+EMKL)/(EKL-1.0-AKL)));
C1 = -1.0*C2*(EMKL-1.0+AKL)/(EKL-1.0-AKL);
cof = (AK*AK*(C1+C2)).real * EI;
dist = (-1.0 * (AK*AK*(C1*EKL+C2*EMKL)).real ) * EI;
// printf("beamColumnStiffness: bar %d, Fx %f, dist %f, cof %f\n",
//  i, bars[i].Fx, dist, cof);

return 0;
}


int fr3d::echoJointDisplacements() {// echo joint displacements

printf("\n\n i\tdx \t\tdy \t\tdz \t\trx \t\try \t\trz \n");
for (int i=1; i<=NN; i++)
  printf(" %d \t%e \t%e \t%e \t%e \t%e \t%e\n",
    i, nodes[i].dx, nodes[i].dy, nodes[i].dz,
    nodes[i].rx, nodes[i].ry, nodes[i].rz );

return 0;
}

int fr3d::echoMemberDisplacements() {// echo member displacements

printf("\n\n i\tdl \t\ttwist \t\trpy \t\trpz \t\trny \t\trnz \n");
for (int i=1; i<=NB; i++)
  printf(" %d \t%e \t%e \t%e \t%e \t%e \t%e\n",
    i, bars[i].dl, bars[i].twist, bars[i].rpy, bars[i].rpz,
    bars[i].rny, bars[i].rnz );

return 0;
}


int fr3d::echoMemberForces() {// echo member forces

printf("\n\n i\tFx \t\tTorque \t\tMpy \t\tMny \t\tMpz \t\tMnz
\n");
for (int i=1; i<=NB; i++)
  printf(" %d \t%e \t%e \t%e \t%e \t%e \t%e\n",
    i, bars[i].Fx, bars[i].Torque, bars[i].Mpy, bars[i].Mny,
    bars[i].Mpz, bars[i].Mnz );

return 0;
}


int fr3d::echoStructure() {
/*
 * This shows just the variables that define the structure,
 * i.e., not the results.
 *
 */

int i;

printf(" %d bars \n %d nodes \n %d dof\n", NB, NN, dof);
```

```
printf("Nodes:\n i\tx\ty\tz\t\tPx\tPy\tPz\t\tMx\tMy\tMz\n");
for (i=1; i<=NN; i++)// nodal properties
  printf(" %d %10.5f %10.5f %10.5f %10.5f %10.5f %10.5f %10.5f
%10.5f %10.5f \n",
     i, nodes[i].x, nodes[i].y, nodes[i].z,
     nodes[i].Px, nodes[i].Py, nodes[i].Pz,
     nodes[i].Mx, nodes[i].My, nodes[i].Mz);

printf("Members:\n i\tNM NP
area\tIx\tIy\tIz\t\tlength\tthx\tthy\tthz\tE\tG\n");
for (i=1; i<=NB; i++)// member properties
  printf(" %d \t%d %d %10.5f %10.5f %10.5f %10.5f %10.5f %10.5f
%10.5f %10.5f %10.1f %10.1f \n",
    i, bars[i].NM, bars[i].NP, bars[i].area,
    bars[i].Ix, bars[i].Iy, bars[i].Iz, bars[i].length,
    bars[i].thx*180/PI, bars[i].thy*180/PI, bars[i].thz*180/PI,
bars[i].E, bars[i].G);

printf("Nodal Fixity:\n i dx dy dz rx ry rz\n");
for (i=1; i<=NN; i++)// nodal properties
  printf(" %d  %d  %d  %d  %d  %d  %d\n",
     i, nodes[i].fix[0], nodes[i].fix[1], nodes[i].fix[2],
nodes[i].fix[3], nodes[i].fix[4], nodes[i].fix[5]);

return 0;
}


int fr3d::firstDofIndex(int n) {
// Return the index (zero-based) of the first free dof of
// node 'n'.
int i, j, sum;

sum = 0;
for(i=1; i<n; i++)
  for(j=0; j<6; j++)
if(!nodes[i].fix[j]) sum++;

// printf("fr3d::firstDofIndex node %d, first dof: %d\n", n, sum);
return sum;
}
```

```
int* fr3d::getDimensions(char* fileName)
{
/* This method only reads the first line of the model
 * and returns the dimensions so they can be used for
 * the constructor which allocates the space for the model
 */

int * dim = new int[3];

fstream inPut;

// fileName="file.dat";
inPut.open(fileName, ios::in);  // open the file
if(inPut.fail())  {
  cout << "***Error::getDimensions file: "
<< fileName << " not opened. \n";
  exit(-1);
}

inPut >> dim[0] >> dim[1] >> dim[2];
inPut.close();  // close the file

return dim;
}

int fr3d::getDofNumber(int node, int index) {
// Any node can have 6 dof but may have less than that. If there
are
// less, it is necessary to get the index of terms in the rotated
element
// stiffness matrix so only the terms corresponding to free nodes
will be
// inserted in the global matrices (e.g., Ke). For example, the
2,2 term
// is the 4EIy/L flexural stiffness) but this may be the first
free dof at
// this node. Thus all elements that tie into this node should
begin
// with terms at the 2,2 index.
//
// This method returns the index 'i' in the rotated 6x6 element
stiffness
```

```
// matrix of the 'index' free dof. For the example here, given
index=0
// then i=2 is returned. In this example, the array 'fix' might
have the
// form {1 1 0 1 1 0} where here it is significant that the first
two
// terms are not zero.

int i, ind;
ind = -1;

for(i=0; i<6; i++) {
  if(!nodes[node].fix[i]) {
    // printf("free dof %d\n", i);
    ind++;
  }
  if(ind == index) {
    // printf(" at i=%d, found %d == %d\n", i, ind, index);
    return i;
  }
}
if(ind == -1) killme("fr3d::getDofNumber failed");
return 0;
}

int fr3d::initialize() {
/*
 * This should probably be near the constructor but it is
 * here for now. It calls routines that are normally done
 * only once for the structure; i.e., not at each iteration
 * and probably called from the main program.
 *
 */

int i, sum;
sum = 0;

for(i=1; i<=NB; i++) mkProps(i);// set basic member props.
for(i=1; i<=NN; i++) {
  sum += numberDofAtNode(i);
  // printf("fr3d::initialize n %d dof %d sum %d\n", i, numberD-
ofAtNode(i), sum);
}
if(sum != dof) {
  printf("fr3d::initialize sum=%d, dof=%d\n", sum, dof);
  killme("fr3d::initialize, dof != sum");
}

return 0;
}

int fr3d::insertElement(matrix & MAT, double m[6][6], int n1, int
n2) {
/*
 * Insert a 6x6 matrix contribution to a larger matrix (e.g., Ke,
Kg,
 * or N_sys). Variables 'n1' and 'n2' are the one-based indices of
 * the location to insert this 6x6 matrix, 'm' into the largeer
matrix,
 * 'MAT'. Thus the indices are one-based but the matrices 'm' and
'MAT' are
 * zero-based. Also, the indices treat the 6x6 matrix as a single
element
 * thus (n1,n2) indices (2,2) will put the elements from 'm' into
'MAT'
 * beginning at MAT(6,6)=m[0][0].
 */

int i, j, K, L, r, c;

r = firstDofIndex(n1);
c = firstDofIndex(n2);

for(i=r; i<r+numberDofAtNode(n1); i++) {
  for(j=c; j<c+numberDofAtNode(n2); j++) {
    K = getDofNumber(n1, i-r);
    L = getDofNumber(n2, j-c);
    if( !(nodes[n1].fix[K]) && !(nodes[n2].fix[L]) ) {
      // printf("fr3d:insertElement Ke(%d,%d) from m(%d,%d)\n", i,
j, K, L);
      MAT(i,j) += m[K][L];
    }
  }
}
```

```
// printf("fr3d::insertElement (%d, %d) value %f\n", i, j, m[i-
r][j-c]);

return 0;
}

int fr3d::insertToNsys(matrix & MAT, double m[6][6], int n1, int
n2) {
/*
 * Insert a 6x6 matrix contribution to a larger matrix (e.g., Ke,
Kg,
 * or N_sys). Variables 'n1' and 'n2' are the one-based indices of
 * the location to insert this 6x6 matrix, 'm' into the largeer
matrix,
 * 'MAT'. Thus the indices are one-based but the matrices 'm' and
'MAT' are
 * zero-based. Also, the indices treat the 6x6 matrix as a single
element
 * thus (n1,n2) indices (2,2) will put the elements from 'm' into
'MAT'
 * beginning at MAT(6,6)=m[0][0].
 */

int i, j, K, L, r, c;

r = 6*(n1-1);
c = firstDofIndex(n2);

for(i=r; i<r+6; i++) {
  for(j=c; j<c+numberDofAtNode(n2); j++) {
    // printf("fr3d::insertToNsys i %d j %d\n", i, j);
    K = i-r;
    L = getDofNumber(n2, j-c);
    if( !(nodes[n2].fix[L]) ) {
      MAT(i,j) += m[K][L];
    }
  }
}

// printf("fr3d::insertToNsys (%d, %d) value %f\n", i, j, m[i-
r][j-c]);

return 0;
}

int fr3d::isViolated(constraint * g) {

int i, b;
double temp = 0.0, maxRatio = 0.0;

/* Check to see if any constraints are violated. This will check
 * for the occasional case where a large initial scaling is made
 * that is too non-linear to give a good prediction. If any con-
straints
 * are voilated, a 1 will be returned; zero otherwise.
 */

for(i=0; i < g->number; i++) {
  b = g->loc[i];
  // printf("fr3d::isViolated: constraint %d, item %c\n", i, g-
>item[i]);
  /* member length change */
  if(g->item[i] == 'L') {
    if (sign(g->val[i]) == sign(bars[b].dl) )
temp = bars[b].dl / g->val[i];
  /* member end rotations (positive end) */
  } else if(g->item[i] == 'P' ) {
    if (sign(g->val[i]) == sign(bars[b].rpy) )
temp = pow( (bars[b].rpy / g->val[i]), 1/bars[b].A2Ix[1]);
  /* member end rotations (negative end) */
  } else if(g->item[i] == 'N') {
    if (sign(g->val[i]) == sign(bars[b].rny) )
temp = pow( (bars[b].rny / g->val[i]), 1/bars[b].A2Ix[1]);
  /* joint displacement (dx) */
  } else if(g->item[i] == 'X') {
    if (sign(g->val[i]) == sign(nodes[b].dx) )
temp = nodes[b].dx / g->val[i];
  /* joint displacement (dy) */
  } else if(g->item[i] == 'Y') {
    if (sign(g->val[i]) == sign(nodes[b].dy) )
temp = nodes[b].dy / g->val[i];
  /* member size (M) */
  } else if(g->item[i] == 'M') {
    if (g->type[i] == -1)
    temp = g->val[i] / bars[b].area;
  /* default */
```

```
  } else {
    killme("fr3d::isViolated unknown constraint");
  }
  maxRatio = temp > maxRatio ? temp : maxRatio;
}

// printf("fr3d::isViolated, maxRatio = %f \n", maxRatio);
if(maxRatio > 1) {
  printf("\t isViolated: maxRatio = %10.5f \n", maxRatio);
  return 1;
} else return 0;
}

int fr3d::mkF(matrix & LP_c, constraint * f) {

// construct the vector, 'c' in the problem
// minimize c^T x subject to Ax <= b
//

int i, j;

for(i=0; i < f->number; i++) {
  j = i+1;
  LP_c[i]   =  bars[j].length;
  // printf(" in mkF c[%d] = %lf \n", i, LP_c[i]);
}
return 0;
}


int fr3d::mkG(matrix & LP_A, matrix & LP_b, constraint * g, int
n_var, double * pda) {

/* construct the A and b in
 * minimize c^T x subject to Ax <= b
 *
 * members of the constraint structure are:
 * variablecontentsexampledata type
 * -------------------------------
 * item quantity dl, dx, ...char
 * location member / node 1, 34, ..int
 * typetype UPPER / LOWERint (+/- 1)
 * val limit0.05, -120double
 *
 * Variable 'pda' is the previous DA increment and should be used
 * in an iterative solution to approximate the second order terms.
 * This is done by using an estimate of DA (given in 'pda') for one
 * of the two terms in the second order part, '1/2 d2K/da2 da^2'.
 *
 */

int b, i, ii, ikn, j, k, n_con;
double dK[6][6], tempd, t1, t2, t3;
matrix NKN(6*NB, 6*NB), KN(dof, 6*NB), phi(6*NB);
/* 'phi' is a column vector representation of the sparse matrix,
 * dK/dA * Del. Each column of this sparse matrix has three ele-
ments
 * corresponding to dK/dA_i * Del.
 */

// printf(" begin mkG \n");
/* assemble:
 * -[N Kinv N~]
 * for general constraints and put this entire product into 'NKN'
 */
n_con = g->number;
if(!(g->item[0] == 'G' && n_con == 1)) {

        if( g->isDefined('L') || g->isDefined('P') || g->isDe-
fined('N') )
          NKN = -N_sys * Kinv * N_sys.transpose();
        if( g->isDefined('X') || g->isDefined('Y') )
          KN = -Kinv * N_sys.transpose();

for(i=0; i<6; i++)
  for(j=0; j<6; j++)
dK[i][j] = 0;

// if(pda != NULL) printf("fr3d::mkG using second order terms\n");
// else printf("fr3d::mkG using first order terms\n");

phi.setZero();
/* construct 'phi' */
for(i=0; i<NB; i++) {
  b = i+1;
  /* derivative of the primitive stiffness matrix for each bar */
```

```
  dK[0][0] = bars[b].E / bars[b].length;
  dK[1][1] = bars[b].G * bars[b].A2Ix[0] * bars[b].A2Ix[1] *
pow(bars[b].area, bars[b].A2Ix[1]-1) / bars[b].length;
  dK[2][2] = 4 * bars[b].E * bars[b].A2Iy[0] * bars[b].A2Iy[1] *
pow(bars[b].area, bars[b].A2Iy[1]-1) / bars[b].length;
  dK[3][3] = 4 * bars[b].E * bars[b].A2Iz[0] * bars[b].A2Iz[1] *
pow(bars[b].area, bars[b].A2Iz[1]-1) / bars[b].length;
  if(pda != NULL) {
    killme("***fr3d::mkG pda not ready\n");
    dK[1][1] +=
2*bars[b].E*bars[b].A2Ix[0]*bars[b].A2Ix[1]*(bars[b].A2Ix[1]-1) *
  pow(bars[b].area, (bars[b].A2Ix[1]-2)) / bars[b].length *
pda[i];
  }
  dK[2][2] = dK[1][1];
  dK[1][2] = dK[1][1] * 0.5 ;
  dK[2][1] = dK[1][2];

  for(ii=0; ii<6; ii++) {
    phi[6*i+ii] = dK[ii][0] * bars[b].dl
     + dK[ii][1] * bars[b].twist
     + dK[ii][2] * bars[b].rpy
     + dK[ii][3] * bars[b].rpz
     + dK[ii][4] * bars[b].rny
     + dK[ii][5] * bars[b].rnz;
  }
}

} // close if(!(g->item[0] == 'G' && g->num == 1))

/* fill in the LP matrix */
LP_A.setZero();
for(i=0; i<n_con; i++) {
  /* b is the location (e.g., bar number) */
  b = g->loc[i];
  /* member length change constraints (dl) */
  if(g->item[i] == 'L') {
    LP_b[i] = g->type[i] * (g->val[i] - bars[b].dl);
    for(j=0; j<n_var; j++) {
      /* Here k is the first element of 3 in a block corresponding
to
       * variable j. This requires that the variables are ordered
and
       * inclusive because k is a function of j.
       */
      for(ii=0; ii<3; ii++) {
        k = 3*j + ii;
LP_A(i,j) += NKN(3*(b-1),k) * phi[k];
      }
      LP_A(i,j) = LP_A(i,j) *  g->type[i];
    }
  /* member end rotation constraints (positive end) */
  } else if(g->item[i] == 'P') {
    LP_b[i] = g->type[i] * (g->val[i] - bars[b].rpy);
    for(j=0; j<n_var; j++) {
      for(ii=0; ii<3; ii++) {
        k = 3*j + ii;
LP_A(i,j) += NKN(3*(b-1)+1,k) * phi[k];
      }
      LP_A(i,j) = LP_A(i,j) * g->type[i];
    }
  /* member end rotation constraints (negative end) */
  } else if(g->item[i] == 'N') {
    LP_b[i] = g->type[i] * (g->val[i] - bars[b].rny);
    for(j=0; j<n_var; j++) {
      for(ii=0; ii<3; ii++) {
        k = 3*j + ii;
LP_A(i,j) += NKN(3*(b-1)+2,k) * phi[k];
      }
      LP_A(i,j) = LP_A(i,j) * g->type[i];
    }
          /* joint displacement constraints (dx) */
  } else if(g->item[i] == 'X') {
    ikn = 3*(b-1);                    // index in KN is dof, not
node number
    LP_b[i] = g->type[i] * (g->val[i] - nodes[b].dx);
    for(j=0; j<n_var; j++) {
      for(ii=0; ii<3; ii++) {
        k = 3*j + ii;
LP_A(i,j) += KN(ikn,k) * phi[k];
      }
      LP_A(i,j) = LP_A(i,j) * g->type[i];
    }
          /* joint displacement constraints (dy) */
  } else if(g->item[i] == 'Y') {
```

```
    ikn = 3*(b-1)+1;                    // index in KN is dof, not
node number
    LP_b[i] = g->type[i] * (g->val[i] - nodes[b].dx);
    for(j=0; j<n_var; j++) {
      for(ii=0; ii<3; ii++) {
        k = 3*j + ii;
LP_A(i,j) += KN(ikn,k) * phi[k];
      }
      LP_A(i,j) = LP_A(i,j) * g->type[i];
    }

  /* member size constraints (M) */
  } else if(g->item[i] == 'M' ) {
    LP_b[i] = g->type[i] * (g->val[i] - bars[b].area);
    // printf("i %d, Lii %d,  b[i] %f \n", i, Lii, b[Lii]);
    // for(j=0; j<n_var; j++) LP_A(i,j) = 0.0;
    // this assumes the variables are inclusive and ordered
    LP_A(i,b-1) = g->type[i] * 1.0;

  /* global buckling */
  } else if(g->item[i] == 'G') {
    matrix Del(6*NB), delta(dof);
    // A nonlinear analysis must have been performed and the first
    // mode shape computed. This mode shape is used as the defomed
    // shape and the corresponding member displacements are com-
puted.
    // Here the first mode shape is in the last column.
    //
    // cout << "mkG:: modeShapes\n" << modeShapes << endl;
    // cout << "mkG:: delta\n" << delta << endl;
    // When using matrix::nroot(), the modeShapes are really the
    // transpose; i.e., vectors stored in rows, not columns.
    for(ii=0; ii<dof; ii++) delta[ii] = modeShapes(dof-1, ii);
    // nonlinearNdelta(Del, delta);
    Del = N_sys * delta;
    // cout << "mkG:: delta (mode shape)\n" << delta << endl;
    // cout << "mkG:: Del\n" << Del << endl;
    LP_b[i] = 0.0;
    for(j=0; j<n_var; j++) {
      // Here 'b' is the bar number and 'k' is the index of the
      // first displacement 'dl' for bar 'b'.
      k = 6*j;
      b = j+1;
      t1 = bars[b].G / bars[b].length * bars[b].A2Ix[0] *
bars[b].A2Ix[1]
      * pow(bars[b].area,(bars[b].A2Ix[1]-1));
      t2 = 4 * bars[b].E / bars[b].length * bars[b].A2Iy[0] *
bars[b].A2Iy[1]
      * pow(bars[b].area,(bars[b].A2Iy[1]-1));
      t3 = 4 * bars[b].E / bars[b].length * bars[b].A2Iz[0] *
bars[b].A2Iz[1]
      * pow(bars[b].area,(bars[b].A2Iz[1]-1));

      tempd = bars[b].E / bars[b].length * Del[k]*Del[k]
+ t1 * Del[k+1]*Del[k+1]
+ t2 * Del[k+2]*Del[k+2]
+ t3 * Del[k+3]*Del[k+3]
+ t2 * Del[k+4]*Del[k+4]
+ t3 * Del[k+5]*Del[k+5]
+ t2 * Del[k+2]*Del[k+4]
+ t3 * Del[k+3]*Del[k+5];

      if(pda != NULL) {
        tempd +=
2*bars[b].A2Ix[0]*bars[b].A2Ix[1]*(bars[b].A2Ix[1]-1)
        * pow(bars[b].area, bars[b].A2Ix[1]-2)
* (Del[k+1]*Del[k+1] + Del[k+1]*Del[k+2] + Del[k+2]*Del[k+2]);
      }
      // cout << i << " " << j << " " << tempd << " " << k << " "
      //  << bars[b].A2Ix[0] << " " << bars[b].A2Ix[1] << endl;
      LP_A(i,j) = -tempd;

      // free memory
      delta.del();
      Del.del();
    }
  } else {
    killme("fr3d::mkG, bad constraint item");
  }
}

// free memory
NKN.del();
KN.del();
phi.del();
```

```
return 0;
}

int fr3d::mkKei(double SK[6][6], int bar) {
// Construct the primitive stiffness matrix for bar 'b'
int i, j;
double disty, cofy, distz, cofz;

for(i=0; i<6; i++)
  for(j=0; j<6; j++)
    SK[i][j] = 0.0;

if(nonlinear) {
  beamColumnStiffness(bar, disty, cofy);
  distz = disty * bars[bar].Iz / bars[bar].Iy;
  cofz  = cofy  * bars[bar].Iz / bars[bar].Iy;
} else {
  disty = 4*bars[bar].E * bars[bar].Iy / bars[bar].length;
  distz = 4*bars[bar].E * bars[bar].Iz / bars[bar].length;
  cofy  = disty / 2.0;
  cofz  = distz / 2.0;
}
SK[0][0] = bars[bar].E * bars[bar].area / bars[bar].length;
SK[1][1] = bars[bar].G * bars[bar].Ix / bars[bar].length;
SK[2][2] = disty;
SK[4][4] = SK[2][2];
SK[2][4] = cofy;
SK[4][2] = cofy;
SK[3][3] = distz;
SK[5][5] = SK[3][3];
SK[3][5] = cofz;
SK[5][3] = cofz;

return 0;
}

int fr3d::mkKgi3(double kg11[6][6], double kg12[6][6], double
kg22[6][6], int b) {
// Construct the three submatrices for Kg in the element lcs.
double c1;

c1 = bars[b].Fx / 30.0;
kg11[1][1] =  c1 * 36.0 / bars[b].length;
kg11[2][2] =  kg11[1][1];
kg11[1][5] = -c1 * 3.0;
kg11[2][4] =  kg11[1][5];
kg11[4][2] =  kg11[1][5];
kg11[5][1] =  kg11[1][5];
kg11[4][4] =  c1 * 4.0 * bars[b].length;
kg11[5][5] =  kg11[4][4];

// try first changing only the NP term because the first model
// has only this node free. This change is consistent with the
change
// in sign in the stiffness matrix.
kg11[2][4] =  -kg11[1][5];
kg11[4][2] =  -kg11[1][5];

kg12[1][1] = -kg11[1][1];
kg12[2][2] = -kg11[2][2];
kg12[1][5] =  kg11[1][5];
kg12[2][4] =  kg11[2][4];
kg12[4][2] = -kg11[4][2];
kg12[5][1] = -kg11[5][1];
kg12[4][4] = -c1 * bars[b].length;
kg12[5][5] =  kg12[4][4];

kg22[1][1] =  kg11[1][1];
kg22[2][2] =  kg11[2][2];
kg22[1][5] = -kg11[1][5];
kg22[2][4] = -kg11[2][4];
kg22[4][2] = -kg11[4][2];
kg22[5][1] = -kg11[5][1];
kg22[4][4] =  kg11[4][4];
kg22[5][5] =  kg11[5][5];

return 0;
}

int fr3d::mkKgi(double SKG[6][6], int b) {
// Construct the contribution of bar 'b' to Kg

double T, V, ANX,ANY;

T = bars[b].Fx;
```

```
V = -(bars[b].Mpy + bars[b].Mny) / bars[b].length;
ANX = cos(bars[b].thy);
ANY = sin(bars[b].thy);

SKG[0][0] = (T*(1.0-ANX*ANX) + V*2.0*ANX*ANY) / bars[b].length;
SKG[0][1] = (-T*ANX*ANY - V*(1.0-2.0*ANY*ANY)) / bars[b].length;
SKG[1][0] = SKG[0][1];
SKG[1][1] = (T*(1.-ANY*ANY) - V*2.0*ANX*ANY) / bars[b].length;

return 0;
}


int fr3d::mkProps(int bar) {
/*
 * Compute the geometric properties of element number 'bar'.
 * Properties computed are
 * angle of the bar's alignment (in radians!!)
 * the bar's length
 * constants c0 and c1 that relate bar area and
 * its section modulus according to
 * I = c0 * A ^ c1
 */

int n1, n2;
double dx, dy, dz;
double th1, th2, th3;

n1 = bars[bar].NM;
n2 = bars[bar].NP;

bars[bar].length = 0.0;
dx = nodes[n2].x - nodes[n1].x;// delta x
dy = nodes[n2].y - nodes[n1].y;
dz = nodes[n2].z - nodes[n1].z;
bars[bar].length = dx*dx + dy*dy + dz*dz;
bars[bar].length = sqrt(bars[bar].length);  // length

if(bars[bar].length == 0) {
  printf(" bar %d, n1: %d, n2: %d \n", bar, n1, n2);
  killme("fr3d::mkProps, zero length bar", bar);
}

/*
th1 = atan2(dz,dy) * 180 / PI;
th2 = atan2(-dz,dx) * 180 / PI;
th3 = atan2(dy,dx) * 180 / PI;
printf("bar %d %f %f %f | %f %f %f\n", bar, th1, th2, th3,
 bars[bar].thx*180.0/PI, bars[bar].thy*180.0/PI,
bars[bar].thz*180.0/PI);
*/

/* Right now, using c1 = 2 (quadratic) for area <=> Ix */
bars[bar].A2Ix[1] = 2.0;
bars[bar].A2Iy[1] = 2.0;
bars[bar].A2Iz[1] = 2.0;
bars[bar].A2Ix[0] = bars[bar].Ix/pow(bars[bar].area,
bars[bar].A2Ix[1]);
bars[bar].A2Iy[0] = bars[bar].Iy/pow(bars[bar].area,
bars[bar].A2Iy[1]);
bars[bar].A2Iz[0] = bars[bar].Iz/pow(bars[bar].area,
bars[bar].A2Iz[1]);

// printf(" bar %d, c0 %f, c1 %f\n", bar, bars[bar].A2Ix[0],
bars[bar].A2Ix[1]);
// printf(" bar %d, dx %f, dy %f, thy %f\n", bar, dx, dy,
bars[bar].thy);

return 0;
}


int fr3d::mkRN(double NRP[6][6], double NRM[6][6], int bar) {
/*
 * Create the rotated transformation matrices NRP and NRM for
 * the 'bar' member and the rotation matrix 'R'.
 *
 */

double R[6][6], SNP[6][6], SNM[6][6], linv;
int i, j, k;

for(i=0; i<6; i++) {
  for(j=0; j<6; j++) {
SNP[i][j] = 0.0;
SNM[i][j] = 0.0;
```

```
  }
}

mkRotationMatrix(R, bar);
linv = 1.0 / bars[bar].length;

/* transformation matrix, N, for (+) node */
SNP[0][0] = 1.0;
SNP[1][3] = 1.0;
SNP[2][4] = 1.0;
SNP[3][5] = 1.0;
SNP[2][2] = linv;
SNP[3][1] =-linv;
SNP[4][2] = linv;
SNP[5][1] =-linv;

/* transformation matrix, N, for (-) node */
SNM[0][0] = -1.0;
SNM[1][3] = -1.0;
SNM[4][4] =  1.0;
SNM[5][5] =  1.0;
SNM[2][2] =-linv;
SNM[3][1] = linv;
SNM[4][2] =-linv;
SNM[5][1] = linv;

/* transform from local coord. sys. to global coord. sys. */
for(i=0; i<6 ; i++) {
  for(j=0; j<6 ; j++) {
    NRP[i][j]=0.0;
    NRM[i][j]=0.0;
    for(k=0; k<6 ; k++) {
NRP[i][j] += SNP[i][k]*R[k][j];
NRM[i][j] += SNM[i][k]*R[k][j];
    }
  }
}

return 0;
}

int fr3d::mkRotationMatrix(double R[6][6], int bar) {
/*
 * Create the rotation matrix 'R' for member
 * number 'bar'.
 *
 */

int i, j, k;
double r[3][3], s[3][3], t[3][3];

for(i=0; i<6; i++) for(j=0; j<6; j++) R[i][j] = 0.0;
for(i=0; i<3; i++) for(j=0; j<3; j++) t[i][j] = 0.0;

/* about x-axis */
t[0][0] =  1.0;
t[1][1] =  cos( bars[bar].thx );
t[2][2] =  t[1][1];
t[1][2] =  sin( bars[bar].thx );
t[2][1] = -t[1][2];
for(i=0; i<3; i++) for(j=0; j<3; j++) r[i][j] = t[i][j];

/* about y-axis */
for(i=0; i<3; i++) for(j=0; j<3; j++) t[i][j] = 0.0;
t[0][0] =  cos( bars[bar].thy );
t[1][1] =  1.0;
t[2][2] =  t[0][0];
t[0][2] = -sin( bars[bar].thy );
t[2][0] = -t[0][2];
for(i=0; i<3; i++)
  for(j=0; j<3; j++) {
    s[i][j] = 0.0;
    for(k=0; k<3; k++) s[i][j] += t[i][k] * r[k][j];
  }
for(i=0; i<3; i++) for(j=0; j<3; j++) r[i][j] = s[i][j];

/* about z-axis */
for(i=0; i<3; i++) for(j=0; j<3; j++) t[i][j] = 0.0;
t[0][0] =  cos( bars[bar].thz );
t[1][1] =  t[0][0];
t[2][2] =  1.0;
t[0][1] =  sin( bars[bar].thz );
t[1][0] = -t[0][1];
for(i=0; i<3; i++)
  for(j=0; j<3; j++) {
    s[i][j] = 0.0;
```

```
    for(k=0; k<3; k++) s[i][j] += t[i][k] * r[k][j];
  }
for(i=0; i<3; i++) for(j=0; j<3; j++) r[i][j] = s[i][j];

for(i=0; i<3; i++) {
  for(j=0; j<3; j++) {
    R[i][j] = r[i][j];
    R[i+3][j+3] = r[i][j];
  }
}

return 0;
}


int fr3d::nonlinearNdelta(matrix & DDD, matrix ddd, char* update-
Flag) {
// SUBROUTINE RESET(P,R,NP,MI,TH,FAC,NNS,AL,DL,ALPHP,ALPHM,PI)
// This provides the 'exact' version of equation
// Delta = N * delta
// relating the member displacements to the joint displacements.
Also
// the member alignment 'thy' is updated if variable 'updateFlag'
// is equal to "ROTATION". At this point, length is not updated
(i.e.,
// small strains) but if added later, this flag would indicate
when to do
// so.
//
double DX,DY,ROT,DAL,DXR,DYR;
int b, nm, np;

for(b=1; b<=NB; b++) {
DX = 0.0;
DY = 0.0;
DDD[3*b-2] = 0.0;
DDD[3*b-1] = 0.0;
np = bars[b].NP;
nm = bars[b].NM;

if(np <= freeNodes) {
  DX = ddd[3*np-3];
  DY = ddd[3*np-2];
  DDD[3*b-2] = ddd[3*np-1];
}

if(nm <= freeNodes) {
  DX -= ddd[3*nm-3];
  DY -= ddd[3*nm-2];
  DDD[3*b-1] = ddd[3*nm-1];
}

DXR =  cos(bars[b].thy)*DX + sin(bars[b].thy)*DY;
DYR = -sin(bars[b].thy)*DX + cos(bars[b].thy)*DY;
DAL = bars[b].length;

// cout << "nonlinearNdelta: b " << b << " dxyr " << DXR << " " <<
DYR << endl;
ROT = atan2(DYR, DAL+DXR);
DDD[3*b-3] = sqrt((DAL+DXR)*(DAL+DXR) + DYR*DYR) - DAL;
DDD[3*b-2] -= ROT;
DDD[3*b-1] -= ROT;
// TH += ROT/PI;
if(!strcmp(updateFlag, "THETA")) {
  // printf("fr3d::nonlinearNdelta: updating thy\n");
  bars[b].thy += ROT;
}

// printf("nonlinNdel %d, %f \t%f \t%f\n", b, bars[b].dl,
bars[b].rpy, bars[b].rny);
}

return 0;
}


int fr3d::numberDofAtNode(int n) {
// Return the number of degrees of freedom at node 'n' based
// on the array 'fix'.
int i, sum;

sum = 0;
for(i=0; i<6; i++) if(!nodes[n].fix[i]) sum++;

return sum;
}
```

```
int fr3d::prod6T(double A[6][6], double B[6][6], double C[6][6],
double D[6][6]) {

// multiply A^T x B x C and return in D

int i, j, k;
double temp[6][6];

for(i=0; i<6 ; i++) { // form product
  for(j=0; j<6 ; j++) {
    temp[i][j]=0.0;
    for(k=0; k<6 ; k++) temp[i][j] += A[k][i]*B[k][j];
  }
}

for(i=0; i<6 ; i++) { // form product
  for(j=0; j<6 ; j++) {
    D[i][j]=0.0;
    for(k=0; k<6 ; k++) D[i][j] += temp[i][k]*C[k][j];
  }
}

return 0;
}


int fr3d::readFile(char* fileName)
{
int i, j;
fstream inPut;

// fileName="file.dat";
inPut.open(fileName, ios::in);  // open the file
if(inPut.fail())  {
  cout << "***Error::readFile file: '"
<< fileName << "' not opened. \n";
  exit(-1);
}

inPut >> NB >> NN >> dof;
for (i=1; i<=NN; i++) // nodal coordinates and loads
  inPut >> nodes[i].x >> nodes[i].y >> nodes[i].z
>> nodes[i].Px >> nodes[i].Py >> nodes[i].Pz
>> nodes[i].Mx >> nodes[i].My >> nodes[i].Mz ;

for (i=1; i<=NB; i++) {// member properties
  inPut >> bars[i].NM >> bars[i].NP >> bars[i].area
  >> bars[i].Ix >> bars[i].Iy >> bars[i].Iz
>> bars[i].E >> bars[i].G >> bars[i].thx >> bars[i].thy >>
bars[i].thz;
  bars[i].thx *= PI / 180.0;
  bars[i].thy *= PI / 180.0;
  bars[i].thz *= PI / 180.0;
}

for (i=1; i<=NN; i++) // nodal fixity
  for(j=0; j<6; j++)
    inPut >> nodes[i].fix[j];

inPut.close();  // close the file
// cout << "done reading file: " << fileName <<  "\n";

initialize();

return 0;
}


int fr3d::rmSmallBars(double min) {

int i, j, remove;

remove = 0;
for(i=1; i<=NB; i++)
{
  if (bars[i].area < min) {
    // Should check if resulting structure will still be
    // stable before removing this bar.
    for(j=i; j<=NB; j++) { // shift properties of previous
bars[j].E    = bars[j+1].E;
bars[j].NM   = bars[j+1].NM;
bars[j].NP   = bars[j+1].NP;
bars[j].area = bars[j+1].area;
    }
    NB--;
    remove++;
```

```
  }
}
return remove;
}

int fr3d::writeFile(char* fileName) {
/*
 * Write file in a format that matches the input file
 * format.
 *
 */

int i;
FILE *outfile;

if( (outfile = fopen(fileName, "w")) == NULL ) {
  printf(" ***Error::writeFile: '%s' not opened\n", fileName);
  return 1;
}

fprintf(outfile, "%5d%5d%5d\n", NB, NN, dof);

/* nodal coordinates and loads */
for (i=1; i<=NN; i++)
fprintf(outfile, "%12.5e %12.5e %12.5e %12.5e %12.5e %12.5e %12.5e
%12.5e %12.5e\n",
nodes[i].x, nodes[i].y, nodes[i].z,
nodes[i].Px, nodes[i].Py,  nodes[i].Pz,
nodes[i].Mx, nodes[i].My, nodes[i].Mz );

/* member properties */
for (i=1; i<=NB; i++)
fprintf(outfile, " %4d %4d %12.5e %12.5e %12.5e %12.5e %8.2e %8.2e
%8.1f %8.1f %8.1f \n",
bars[i].NM, bars[i].NP, bars[i].area,
bars[i].Ix, bars[i].Iy, bars[i].Iz,
bars[i].E, bars[i].G, bars[i].thx*180.0/PI,
bars[i].thy*180.0/PI, bars[i].thz*180.0/PI );

for (i=1; i<=NN; i++)
fprintf(outfile, "%d %d %d %d %d %d\n",
nodes[i].fix[0], nodes[i].fix[1], nodes[i].fix[2],
nodes[i].fix[3], nodes[i].fix[4], nodes[i].fix[5] );

fclose(outfile);
// printf("done writing file: %s\n", fileName);

return 0;
}

int fr3d::writeResults(void) {
/*
 * Write the nodal and member displacements
 *
 */

echoJointDisplacements();
echoMemberDisplacements();
echoMemberForces();

return 0;
}

int fr3d::zeroDisplacements(void) {
int i;

for (i=1; i<=NN; i++) {
  nodes[i].dx = 0.0;
  nodes[i].dy = 0.0;
  nodes[i].dz = 0.0;
  nodes[i].rx = 0.0;
  nodes[i].ry = 0.0;
  nodes[i].rz = 0.0;
}

for (i=1; i<=NB; i++) {
  bars[i].dl = 0.0;
  bars[i].twist = 0.0;
  bars[i].rpy = 0.0;
  bars[i].rny = 0.0;
  bars[i].rpz = 0.0;
  bars[i].rnz = 0.0;
}

return 0;
}
```

```
int fr3d::zeroForces(void) {
int i;

for (i=1; i<=NB; i++) {
  bars[i].Fx = 0.0;
  bars[i].Torque = 0.0;
  bars[i].Mpy = 0.0;
  bars[i].Mny = 0.0;
  bars[i].Mpy = 0.0;
  bars[i].Mny = 0.0;
}

return 0;
}


/************************************
 *
 * class methods -- methods that return double and double *
 *
 */



double fr3d::a2ix(int bar) {
/* Given the bar number, 'bar', return the section modulus, I
 * according to the relation
 * I = c0 * A^c1
 * where c0 and c1 are variables in the array 'A2Ix' of the
 * frameMember structure. These constants should be set with
 * a call to 'mkProps'.
 */

double I;

I = bars[bar].A2Ix[0] * pow(bars[bar].area, bars[bar].A2Ix[1]);
// printf("A^2 %f\n", pow(bars[bar].area, bars[bar].A2Ix[1]) );

return I;
}

double fr3d::a2iy(int bar) {
double I;
I = bars[bar].A2Iy[0] * pow(bars[bar].area, bars[bar].A2Iy[1]);
return I;
}


double fr3d::a2iz(int bar) {
double I;
I = bars[bar].A2Iz[0] * pow(bars[bar].area, bars[bar].A2Iz[1]);
return I;
}


double fr3d::getUnbalancedForce(matrix & p) {
// Compute the out of balance forces for the applied load, 'p' and
// return this vector in 'p'. Used for nonlinear analysis.
int b, i, j;
double sum;
matrix f(6*NB);

for(b=1; b<=NB; b++) {
  f[6*b-6] = bars[b].Fx;
  f[6*b-5] = bars[b].Torque;
  f[6*b-4] = bars[b].Mpy;
  f[6*b-3] = bars[b].Mpz;
  f[6*b-2] = bars[b].Mny;
  f[6*b-1] = bars[b].Mnz;
}
// cout << "getUnbalancedForce: f = \n" << f << endl;
p = p - N_sys.transpose() * f;

sum = 0.0;
for(i=0; i<dof; i++) sum += p[i]*p[i];
sum = sqrt(sum);

f.del();
return sum;
}


double fr3d::scale(constraint * g, double factor) {

int i, b, governy;
```

```
double scale = SMALL;
// double temp = 1.0;
double temp = SMALL;

/* Linear scaling will change magnitude but not sign so
 * perform scaling only if delta and delta_allowable
 * have the same sign. Variable 'factor' should be equal
 * to 1.0 for "at limit" condition and greater than 1.0
 * for "below limit" condition.
 */

/* If global buckling, scale for only by 'factor' at this point.
Scaling
 * to be at the buckling load is done iteratively in the main pro-
gram
 * using the eigenvalue.
 */
if(g->findItem('G')) {
  // Make sure it's a nonlinear analysis in case this hasn't been
set
  // and scale by 'factor', then return.
  if(!nonlinear) killme("fr3d::scale buckling analysis should be
nonlinear\n");
  for(i=1; i <= NB; i++) {
    bars[i].area *= factor;
    bars[i].Ix = a2ix(i);
    bars[i].Iy = a2iy(i);
    bars[i].Iz = a2iz(i);
  }
  return factor;
}

/* It is assumed here that member end rotations and joint dis-
placements
 * are not linearly related to the area. This is because 'fr3d'
elements
 * support flexure, which is a function of 'I_mod' and this is
related to
 * 'area' by the exponent 'A2Ix[1]'. The member length change is
assumed
 * to be linearly related to the area however.
 */
for(i=0; i < g->number; i++) {
  b = g->loc[i];
  // printf(" fr3d::scale, constraint %d \n", i);
  /* member length change (dl) */
  if(g->item[i] == 'L') {
    if (sign(g->val[i]) == sign(bars[b].dl) ) {
// printf("scale::constr %d on dl; same sign\n", i);
temp = bars[b].dl / g->val[i];
    }
  /* member end rotations (positive end) */
  } else if(g->item[i] == 'P') {
    if (sign(g->val[i]) == sign(bars[b].rpy) ) {
// printf("scale::constr %d on rpy; same sign\n", i);
temp = bars[b].rpy / g->val[i];
// temp = sqrt(temp);
temp = pow(temp, 1/bars[b].A2Ix[1]);
    }
  /* member end rotations (negative end) */
  } else if(g->item[i] == 'N') {
    if (sign(g->val[i]) == sign(bars[b].rny) ) {
// printf("scale::constr %d on rny; same sign\n", i);
temp = bars[b].rpy / g->val[i];
temp = pow(temp, 1/bars[b].A2Ix[1]);
    }
  /* joint displacement (dx) */
  } else if(g->item[i] == 'X') {
    if (sign(g->val[i]) == sign(nodes[b].dx) ) {
// printf("scale::constr %d on dx; same sign\n", i);
temp = nodes[b].dx / g->val[i];
temp = pow(temp, 1/bars[b].A2Ix[1]);
    }
  /* joint displacement (dy) */
  } else if(g->item[i] == 'Y') {
    if (sign(g->val[i]) == sign(nodes[b].dy) ) {
// printf("scale::constr %d on dl; same sign\n", i);
temp = nodes[b].dy / g->val[i];
temp = pow(temp, 1/bars[b].A2Ix[1]);
    }
  } else if(g->item[i] == 'M') {
    // This should be handled carefully because there could be
    // conflicts here. Minimum is easy but maximum may cause an
    // infeasible problem. Ignore upper limits for now.
    if(g->type[i] == -1) temp = g->val[i] / bars[b].area;
  /* default */
```

```
  } else {
    // killme("fr3d::scale unknown constraint", i+1);
    killme("fr3d::scale unknown constraint", g->item[i], i+1);
  }

  if(temp > scale) {
    scale = temp;
    governy = i;
  }
  // printf("\t\tc: %d ratio: %8.5f scale: %8.5f\n", i, temp,
scale);
}

// printf("scale:: constraint %d governys\n", governy);
// printf("\t scale: \t%10.5f x fact = %10.5f\n", scale, fac-
tor*scale);

for(i=1; i <= NB; i++) {
  bars[i].area *= scale*factor;
  bars[i].Ix = a2ix(i);
  bars[i].Iy = a2iy(i);
  bars[i].Iy = a2iy(i);
}

return scale*factor;
}


double fr3d::volume() {

double vol = 0.0;

for(int i=1; i<=NB; i++)
  vol += bars[i].area * bars[i].length;

return vol;
}
```

File: iplp.cpp

```
/* file iplp.cpp
 *
 * Implementation of the 'iplp' class which describes a
 * primal linear program and provides an interior point method
 * to solve it.
 *
 */

#include <math.h>
#include "iplp.h"

/* ---------------------------------
 *
 * constructor / destructor
 *
 * ---------------------------------
 */
iplp::iplp (int r, int c) : A(r, c), b(r, 1),
c(c, 1), x(c, 1),
y(r, 1), z(c, 1)
{
row  = r; // number of rows
col  = c;// number of columns
dobj = 0.0;// dual objective value
obj  = 0.0;// primal objective value
rho  = 0.95;// step size limit
verbose = 0;// intermediate output
}

iplp::iplp (int * dim) : A(dim[0], dim[1]), b(dim[0], 1),
c(dim[1], 1), x(dim[1], 1),
y(dim[0], 1), z(dim[1], 1)
{
row  = dim[0]; // number of rows
col  = dim[1];// number of columns
dobj = 0.0;// dual objective value
obj  = 0.0;// primal objective value
rho  = 0.95;// step size limit
verbose = 0;// intermediate output
}

iplp::~iplp() { }
```

```
/* ----------------------------------
 *
 * member functions
 *
 * ----------------------------------
 */

int iplp::startBigM() {
/* Find an initial feasible interior point.
 */
int i;
for(i=0; i<col; i++) {
  // if(c[i] <= 0) return i+1;
  // y[i] = 0.0;
  // z[i] = c[i];
}
return 0;
}

int iplp::startFix1(int iii, double val) {
/* This method will attempt to recover from 'startY0' when there
is
 * one c_i <= 0, i.e., the 'iii' term. The logic here is to set the
 * vector (matrix) 'y' to a small positive or negative value that
will
 * satisfy the equation for this 'c[iii]' and then solve for the
 * remaining 'z' based on this. The variable 'val' indicates what
small
 * is and has a default value in the header file.
 *
 * This is intended for the case where there is one term in 'c'
that is
 * equal to zero but may work for other cases. If successful, a
zero is
 * returned, otherwise the index+1 of the first non-positive term
in 'z'
 * is returned.
 */

int i, j;
double sum;

if(verbose > 4) printf("iplp::startFix1 try y = %f\n", val);
y = val;

// Set 'z[iii]' for the problem row based on this small value
// of y. If this gives a non-positive z[iii], then try a negative
// value for 'y'.
//
z[iii] = 0.0;
for(i=0; i<row; i++) z[iii] += A(i,iii) * y[i];
if(z[iii] <= 0) {
  // try negative
  if(verbose > 4) printf("iplp::startFix1 try y = %f\n", -val);
  y = -val;
  z[iii] = 0.0;
  for(i=0; i<row; i++) z[iii] += A(i,iii) * y[i];
}

// Now compute remaining z based on this small positive or
// negative 'y'.
//
for(j=0; j<col; j++) {
  if(j != iii) {
    sum = 0.0;
    for(i=0; i<row; i++) sum += A(i,iii) * y[i];
    z[j] = c[j] - sum;
  }
  if(z[j] <= 0) return j+1;
}
if(verbose > 4) printf("iplp::startFix1 found solution\n");

return 0;
}

int iplp::startY0(double fixVal) {
/* Set vector y0 to zero and z0 to c. Zero is returned if
 * all z_i > 0 (required for feasible starting solution).
 * Otherwise, the index+1 of the first c_i <= 0 is returned.
 * This return value should be checked because a non-zero value
 * indicates an infeasible starting solution has been generated
 * and a different method may be needed.
 *
 * The call to 'startFix1' will attempt to recover from the first
 * occurence of a non-positive term in 'c' by using y = +/- 'fix-
Val'
```

```
 * which should be a small number. A default value for 'fixVal' is
 * provided in the header file.
 */
int i;
if(verbose > 8) printf(" begin iplp::startY0 \n");

for(i=0; i<row; i++) y[i] = 0.0;
for(i=0; i<col; i++) {
  if(c[i] <= 0) return startFix1(i, fixVal);
  z[i] = c[i];
}

return 0;
}

int * iplp::getDimensions(char * fName) {
/* read a file that contains the LP problem and return the
 * size of the LP problem in array 'dim'
 */
int * dim = new int[2];
fstream inPut;

// fileName="dual.dat";
inPut.open(fName, ios::in);  // open the file
if(inPut.fail()) {
  cout << "***Error::iplp::getDimensions, '" << fName << "' not
opened \n";
  exit(-1);
}
inPut >> dim[0] >> dim[1];// read the file
inPut.close();  // close the file

return dim;
}


int iplp::printLP() {
/* show current values in 'iplp' class results */
cout << "A: \n" << A << endl;
cout << "b: \n" << b << endl;
cout << "c: \n" << c << endl;
cout << "x: \n" << x << endl;
cout << "y: \n" << y << endl;
cout << "z: \n" << z << endl;

return 0;
}


int iplp::printObjective() {
/* show current values in 'iplp' class results */
cout << "Primal Objective: " << obj << endl;
cout << "Dual Objective: " << dobj << endl;

return 0;
}

int iplp::readLP(char * fName, char * type) {
/* Read a file that contains the LP problem. The problem
 * should also have an initial point which determines the 'type'
 * of solution that can be used. Types are;
 *
 *typeaction solution
 *------------------
 * primal set x0 (primal algorithm)
 * dualset y0 and z0 (dual algorithm)
 * both set x0, y0, and z0 (primal-dual algorithm)
 *
 * If 'type' is not one of listed above, an initial point should
 * be set elsewhere.
 *
 */

int i, j, Rows, Cols;
fstream inPut;

// fileName="dual.dat";
inPut.open(fName, ios::in);  // open the file
if(inPut.fail()) {
  cout << "***Error::iplp::readPrimal, '" << fName << "' not
opened \n";
  exit(-1);
}
inPut >> Rows >> Cols;// read the file
if(Rows != row || Cols != col) exit(-1);
for (i=0; i < Rows; i++) {
```

```
  for (j=0; j < Cols; j++) inPut >> A(i,j);
  inPut >> b[i];
}
for (i=0; i < Cols; i++) inPut >> c[i];

if(!strcmp("primal", type) ) {
  cout << " set initial point for primal\n";
  for (i=0; i < Cols; i++) inPut >> x[i];
} else if(!strcmp("dual", type) ) {
  cout << " set initial point for dual\n";
  for (i=0; i < Rows; i++) inPut >> y[i];
  for (i=0; i < Cols; i++) inPut >> z[i];
} else if(!strcmp("both", type) ) {
  cout << " set initial point for primal-dual\n";
  for (i=0; i < Cols; i++) inPut >> x[i];
  for (i=0; i < Rows; i++) inPut >> y[i];
  for (i=0; i < Cols; i++) inPut >> z[i];
} else if(!strcmp("general", type) ) {
  /* general LP is defined by only A, b, and c*/
} else {
  cout << " ***Warning::iplp::readLP, initial point not
defined\n";
}
inPut.close();  // close the file

return 0;
}


int iplp::solvePD() {
   /* Interior point method: Ami Arbel's primal-dual algorithm
    *
    * Problem
    * primal: min c^T x | A x = b, x >= 0
    * dual:max b^T y | A^T y + z = c, z >= 0
    *
    * Initial point x0, y0, and z0 should be defined.
    *
    * Local variables:
    * mubar barrier parameter
    * vauxauxillary vector
    * dx step direction for the primal vector
    * dy step direction for the dual vector
    * dz step direction for the reduced-cost vector
    */

   int i, j, iter;
   double alpha_p, alpha_d, mubar;
   matrix D2(col, col), Zi(col, 1), ada(row, row), azv(row, 1),
   dx(col, 1), dy(row, 1), dz(col, 1), vaux(col, 1);

   for(iter=1; iter <= itermax; iter++) {

mubar = 0.0;
for(i=0; i<col; i++) mubar += x[i] * z[i];
mubar *= (0.1 / col);
for(i=0; i<col; i++) {
  D2(i,i) = x[i] / z[i];
  Zi(i,i) = 1 / z[i];
  vaux[i] = mubar - x[i] * z[i];
}

ada = A * D2 * A.transpose();
azv = A * Zi * vaux;

matrix::solve(ada, azv, y);
dz = -A.transpose() * dy;
dx = (Zi - D2 * A.transpose() * ada * A * Zi) * vaux;

/* ratio test */
alpha_p=0.0;
for(i=0; i < col; i++)
  if(dx[i] < 0)
    alpha_p = alpha_p > fabs(dx[i]/x[i]) ? alpha_p : fabs(dx[i]/
x[i]);
alpha_d=0.0;
for(i=0; i < col; i++)
  if(dx[i] < 0)
    alpha_d = alpha_d > fabs(dx[i]/x[i]) ? alpha_d : fabs(dx[i]/
x[i]);

/* solution */
x = x + dx * (rho/alpha_p);
y = y + dy * (rho/alpha_d);
z = z + dz * (rho/alpha_d);
```

```
/* compute objectives at each iteration */
obj  = 0.0;
dobj = 0.0;
for(i=0; i < col; i++) obj  += c[i] * x[i];
for(i=0; i < row; i++) dobj += b[i] * y[i];

/* show results of each iteration */
if(verbose) {
  printf("\n solve PD iteration: %d \n", iter);
  cout << "alpha_p: " << alpha_p << endl;
  cout << "alpha_d: " << alpha_d << endl;
  cout << "dx: \n" << dx << endl;
  cout << "dy: \n" << dy << endl;
  cout << "dz: \n" << dz << endl;
  if(verbose > 10) printLP();
}
   }

   /* free memory after solution */
   D2.del();
   Zi.del();
   dx.del();
   dy.del();
   dz.del();
   ada.del();
   azv.del();

   return 0;
}

int iplp::solvePrimal() {
   /* Interior point method: Ami Arbel's primal algorithm
    *
    * Problem
    * primal: min c^T x | A x = b, x >= 0
    * dual:max b^T y | A^T y + z = c, z >= 0
    *
    * Initial point x0 should be defined.
    *
    */

   int i, j, iter;
   double alpha;
   matrix D(col, col), dx(col, 1), ada(row, row), adc(row, 1);

   for(iter=1; iter <= itermax; iter++) {

for(i=0; i<col; i++) D(i,i) = pow(x(i), 2);
ada = A * D * A.transpose();
adc = A * D * c;
matrix::solve(ada, adc, y);
z = c - A.transpose() * y;
dx = -D * z;

/* ratio test */
alpha=0.0;
for(i=0; i < col; i++)
  if(dx[i] < 0)
    alpha = alpha > fabs(dx[i]/x[i]) ? alpha : fabs(dx[i]/x[i]);

/* solution */
x = x + dx * (rho/alpha);

/* compute objectives at each iteration */
obj  = 0.0;
dobj = 0.0;
for(i=0; i < col; i++) obj  += c[i] * x[i];
for(i=0; i < row; i++) dobj += b[i] * y[i];

/* show results of each iteration */
if(verbose) {
  printf("\n solvePrimal iteration: %d \n", iter);
  cout << "alpha: " << alpha << endl;
  cout << "dx: \n" << dx << endl;
  if(verbose > 10) printLP();
}
   }

   /* free memory after solution */
   D.del();
   dx.del();
   ada.del();
   adc.del();

   return 0;
}
```

```
int iplp::solveDual() {
   /* Interior point method: Ami Arbel's dual algorithm
    *
    * Problem
    * primal: min c^T x | A x = b, x >= 0
    * dual:max b^T y | A^T y + z = c, z > 0
    *
    * Initial point y0, and z0 should be defined.
    *
    */

   int i, j, k, iter;
   double alpha, rhoOverAlpha;
   // matrix D(col, col)
   matrix dY(row, 1), dZ(col, 1), ada(row, row);
   matrix D(col,1);

   if(verbose) printf("begin solveDual\n");
   for(iter=1; iter <= itermax; iter++) {
ada.setZero();
dZ.setZero();

for(i=0; i<col; i++) D.m[i] = 1.0 / pow(z(i), 2);
// Even though these are matrix objects, use the "old-fashioned"
// method because when these are large, the extra memory consumed
// is significant.
//
// ada = A * D * A.transpose();
for(i=0; i<row; i++)
  for(j=0; j<row; j++)
    for(k=0; k<col; k++)
      ada(i,j) += A(i,k) * D.m[k] * A(j,k);

matrix::solve(ada, b, dY);
// dZ = -A.transpose() * dY;
for(i=0; i<col; i++)
  for(j=0; j<row; j++)
    dZ.m[i] += -A(j,i) * dY.m[j];

/* ratio test */
alpha=0.0;
for(i=0; i < col; i++)
  if(dZ[i] < 0)
    alpha = alpha > fabs(dZ[i]/z[i]) ? alpha : fabs(dZ[i]/z[i]);

/* solution */
// z = z + dZ * (rho/alpha);
rhoOverAlpha = rho / alpha;
for(i=0; i<col; i++)
  z.m[i] += dZ.m[i] * rhoOverAlpha;

// y = y + dY * (rho/alpha);
for(i=0; i<row; i++)
  y.m[i] = y.m[i] + dY.m[i] * rhoOverAlpha;

// x = -D * dZ;
for(i=0; i<col; i++)
  x.m[i] = -D.m[i] * dZ.m[i];

/* compute objectives at each iteration */
obj  = 0.0;
dobj = 0.0;
for(i=0; i < col; i++) obj  += c.m[i] * x.m[i];
for(i=0; i < row; i++) dobj += b.m[i] * y.m[i];

/* show results of each iteration */
if(verbose > 2) {
  printf("\n solveDual iteration: %d \n", iter);
  cout << "dY: \n" << dY << endl;
  cout << "dZ: \n" << dZ << endl;
  printObjective();
  if(verbose > 10) printLP();
}
   }

   /* free memory after solution */
   D.del();
   dY.del();
   dZ.del();
   ada.del();

   return 0;
}
```

```
File: matrix.cpp
// matrix.cpp: implementation of the matrix class.
//
//////////////////////////////////////////////////////////////

#include "matrix.h"

//////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////

/* basic constructor */
matrix::matrix(int r, int c) : row(r), col(c), m(new dou-
ble[row*col]) {
/* initalize to zero */
for (int i=0; i<(row*col); i++) m[i] = 0;
}

/* copy constructor */
matrix::matrix(matrix &v) : row(v.row), col(v.col), m(new dou-
ble[row*col]) {
for (int i=0; i<(row*col); i++) m[i]=v.m[i];
}

/* destructor; free memory */
matrix::~matrix() {
// printf("destructor\n");
delete [] m;
}

/* ------------------------------------
 *
 * basic matrix operations (+, -, *, =)
 *
 * ------------------------------------
 */

/* addition */
matrix  matrix::operator+ (const matrix & v) const {
/* check dimensions */
if (v.row != row || v.col != col) {
   printf ("***Error::matrix::operator+, bad dimensions\n\n");
  exit(-1);
}
matrix temp(row,col);
for (int i=0; i<(row*col); i++) temp.m[i] = v.m[i] + m[i];
return temp;
}

/* subtraction */
matrix  matrix::operator- (const matrix & v) const {
/* check dimensions */
if (v.row != row || v.col != col) {
   printf ("***Error::matrix::operator-, bad dimensions\n\n");
  exit(-1);
}
matrix temp(row,col);
for (int i=0; i<(row*col);i++) temp[i] = m[i] - v.m[i];

return temp;
}

/* negation */
matrix  matrix::operator- () const {
matrix temp(row,col);
for (int i=0; i<(row*col);i++) temp[i] = -m[i];

return temp;
}


/* multiplication, two matrices */
matrix  matrix::operator* (const matrix & v) const {
  /* check dimensions */
  if (v.row != col) {
   printf ("***Error::matrix::operator*, bad dimensions: ");
   printf (" (%d cols) != (%d rows) \n\n", col, v.row);
    exit(-1);
  }
  matrix temp(row,v.col);

  double sum=0;
  int ik,kj,ij;

  for (int i=0; i<row;i++){
for (int j=0; j<v.col;j++){
```

```
  for (int k=0;k<col;k++){
ik=i*col+k;
kj=k*v.col+j;
sum += m[ik] * v.m[kj];
  }

  ij=i*v.col+j;
  temp[ij]=sum;
  sum=0;
}
  }
  return temp;
}


/* multiplication, matrix and a double */
matrix  matrix::operator* (double d) const {
matrix temp(row,col);

for (int i=0; i<(row*col);i++) temp[i] = m[i] * d;

return temp;
}

/* multiplication, double and a matrix */
matrix operator* (double d, const matrix & v) {
matrix temp(v.row,v.col);

for(int i=0; i<(v.row*v.col); i++) temp[i] = v.m[i] * d;

return temp;
}


/* assignment, matrix = matrix */
const matrix&  matrix::operator= (const matrix & v) {
/* check dimensions */
if (v.row != row || v.col != col) {
  printf ("***Error::matrix::operator= bad dimensions: ");
    if(v.row != row) printf(" (%d rows) != (%d rows)\n\n", v.row,
row);
    else printf(" (%d cols) != (%d cols)\n\n", v.col, col);
  exit(-1);
}

for (int i=0; i<(row*col);i++) m[i] = v.m[i];
return *this;
}

/* assignment, matrix = double */
const matrix&  matrix::operator= (const double & d) {
for (int i=0; i<(row*col);i++) m[i] = d;
return *this;
}

/* ------------------------------------
 *
 * linear algebra operations
 *
 * ------------------------------------
 */


   int matrix::solve (matrix a, matrix b, matrix & x, int ver-
bose){
   /*
    * Solve Ax = b using Gaussian elimination
    *
    * This is the simplest form of solving simultaneous equations.
No
    * checks for singularity are performed. Pivoting is also not
done.
    *
    */

int i, ii, j, rows, cols;
double lead;

/* check dimensions */
if (a.row != b.row || b.row != x.row || b.col != 1
  || a.row != a.col || b.col != x.col) {
    printf ("***Error::matrix::solve, bad dimensions\n\n");
    exit(-1);
}

rows = a.row;
```

```
cols = a.col;
x = b;

/* create upper triangular matrix (zeros below diagonal) */
for (i = 0; i < rows; i++) {
    // no zeros on main diagonal
    if(a(i,i) <= 0.0) {
      if(verbose) printf(" --- matrix::solve negative on main
diagonal\n");
      return i+1;
    }
    // zero column i in all rows below this one
    for (ii = i+1; ii < rows; ii++) {
x.m[ii] =  x.m[ii] - a(ii, i) / a(i, i) * x.m[i];
// save the lead term of this row for row operation
lead = a(ii, i);
for(j = i; j < cols; j++) {
  a(ii, j) =  a(ii, j) - lead / a(i, i) * a(i, j);
    }
    }
}

/* now zero upper part (above diagonal), start from bottom */
for (i=rows-1; i>=0; i--) {
    // no zeros on main diagonal
    if(a(i,i) <= 0.0) {
      if(verbose) printf("*** matrix::solve negative on main diag-
onal\n");
      return i+1;
    }
    // zero column i in all rows above this one
    for (ii = 0; ii < i; ii++) {
// printf("ii: %d\n", ii);
    // no need to explicitly put the zeros in matrix a, only do x
x.m[ii] -=  a(ii, i) / a(i, i) * x.m[i];
    }
    x.m[i] /= a(i,i);
    // printf(" \t inside solve, soln: %d, %f\n", i, x.m[i]);
}

a.del();
b.del();
return 0;
```

```
   }


/* Return the transpose of a matrix */
matrix matrix::transpose() const {
matrix temp(col, row);

for (int i=0; i<row; i++)
  for (int j=0; j<col; j++)
temp(j,i) = m[i*col + j];

return temp;


}

   /*
    * Return the inverse of a matrix. The initial matrix is
unchanged
    *
    */
   matrix matrix::invert() const {
int i;
double det=0;
matrix temp(row, col);

for(i=0; i<(row*col); i++) temp.m[i] = m[i];

det = matrix::minv(temp);

return temp;

   }


/* ********************************************** */


   /* Use the minv routine for matrix inversion. This inverts the
argument
    * and returns the value of the determinant.
    */
```

```
   double matrix::minv(matrix & aa)  {
    if(aa.row != aa.col) {
       printf("***Error::matrix::minv, matrix not square\n");
       exit(-1);
    }
    /* Simulate parameter list of the original minv call:
     * int minv(double * a, int n_, double & d__);
     */
    double *a = aa.m;
    double d__ = 0;
    int n_ = aa.row;
    aa = aa.transpose();

    /* System generated locals */
    int i__1, i__2, i__3;
    double r__1;

    /* Local variables */
    static double biga, hold;
    static int i__, j, k, ij, ki, ji, kk, jk, ik, nk, jp, kj, jq,
jr, iz;

    int *l, *m, ll[n_], mm[n_];
    l = ll;
    m = mm;

/*        SEARCH FOR LARGEST ELEMENT */

    /* Parameter adjustments */
    --m;
    --l;
    --a;

    /* Function Body */
    /* *d__ = (float)1.; */
    d__ = 1.0;
    nk = -(n_);
    i__1 = n_;
    for (k = 1; k <= i__1; ++k) {
nk += n_;
l[k] = k;
m[k] = k;
kk = nk + k;
biga = a[kk];
i__2 = n_;
for (j = k; j <= i__2; ++j) {
    iz = n_ * (j - 1);
    i__3 = n_;
    for (i__ = k; i__ <= i__3; ++i__) {
ij = iz + i__;
/* L10: */
if (fabs(biga) - (r__1 = a[ij], fabs(r__1)) >= (float)0.) {
    goto L20;
} else {
    goto L15;
}
L15:
biga = a[ij];
l[k] = i__;
m[k] = j;
L20:
;
    }
}

/*        INTERCHANGE ROWS */

j = l[k];
if (j - k <= 0) {
    goto L35;
} else {
    goto L25;
}
L25:
ki = k - n_;
i__3 = n_;
for (i__ = 1; i__ <= i__3; ++i__) {
    ki += n_;
    hold = -a[ki];
    ji = ki - k + j;
    a[ki] = a[ji];
/* L30: */
    a[ji] = hold;
}

/*        INTERCHANGE COLUMNS */
```

```
L35:
i__ = m[k];
if (i__ - k <= 0) {
    goto L45;
} else {
    goto L38;
}
L38:
jp = n_ * (i__ - 1);
i__3 = n_;
for (j = 1; j <= i__3; ++j) {
    jk = nk + j;
    ji = jp + j;
    hold = -a[jk];
    a[jk] = a[ji];
/* L40: */
    a[ji] = hold;
}

/*        DIVIDE COLUMN BY MINUS PIVOT (VALUE OF PIVOT ELEMENT IS
*/
/*        CONTAINED IN BIGA) */

L45:
if (biga != (float)0.) {
    goto L48;
} else {
    goto L46;
}
L46:
/* *d__ = (float)0.; */
d__ = 0.0;
return 0;
L48:
i__3 = n_;
for (i__ = 1; i__ <= i__3; ++i__) {
    if (i__ - k != 0) {
goto L50;
    } else {
goto L55;
    }
L50:
    ik = nk + i__;
    a[ik] /= -biga;
L55:
    ;
}

/*        REDUCE MATRIX */

i__3 = n_;
for (i__ = 1; i__ <= i__3; ++i__) {
    ik = nk + i__;
    hold = a[ik];
    ij = i__ - n_;
    i__2 = n_;
    for (j = 1; j <= i__2; ++j) {
ij += n_;
if (i__ - k != 0) {
    goto L60;
} else {
    goto L65;
}
L60:
if (j - k != 0) {
    goto L62;
} else {
    goto L65;
}
L62:
kj = ij - i__ + k;
a[ij] = hold * a[kj] + a[ij];
L65:
;
    }
}

/*        DIVIDE ROW BY PIVOT */

kj = k - n_;
i__2 = n_;
for (j = 1; j <= i__2; ++j) {
    kj += n_;
    if (j - k != 0) {
goto L70;
```

```
    } else {
goto L75;
    }
L70:
    a[kj] /= biga;
L75:
    ;
}

/*        PRODUCT OF PIVOTS */
d__ *= biga;

/*        REPLACE PIVOT BY RECIPROCAL */
a[kk] = (float)1. / biga;
/* L80: */
    }

/*        FINAL ROW AND COLUMN INTERCHANGE */
    k = n_;
L100:
    --k;
    if (k <= 0) {
goto L150;
    } else {
goto L105;
    }
L105:
    i__ = l[k];
    if (i__ - k <= 0) {
goto L120;
    } else {
goto L108;
    }
L108:
    jq = n_ * (k - 1);
    jr = n_ * (i__ - 1);
    i__1 = n_;
    for (j = 1; j <= i__1; ++j) {
jk = jq + j;
hold = a[jk];
ji = jr + j;
a[jk] = -a[ji];
/* L110: */
a[ji] = hold;
    }
L120:
    j = m[k];
    if (j - k <= 0) {
goto L100;
    } else {
goto L125;
    }
L125:
    ki = k - n_;
    i__1 = n_;
    for (i__ = 1; i__ <= i__1; ++i__) {
ki += n_;
hold = a[ki];
ji = ki - k + j;
a[ki] = -a[ji];
/* L130: */
a[ji] = hold;
    }
    goto L100;
L150:

aa = aa.transpose();

return d__;


} /* minv */


/* ------------------------------------
 *
 * miscellaneous operations
 *
 * ------------------------------------
 */

/* single subscript indexing (classical style) */
double&  matrix::operator[] (int d) const {
return m[d];
}
```

```
/* two dimensional array indexing */
double&  matrix::operator() (int i, int j) const {
/* check dimensions */
if (i > row) {
    printf ("***Error::matrix row index %d out of bounds \n\n",
i);
    exit(-1);
}
if (j > col) {
    printf ("***Error::matrix col index %d out of bounds \n\n",
j);
    exit(-1);
}

return m[i*col+j];
}

int matrix::setZero() {
  for(int i=0; i < row; i++)
    for(int j=0; j<col; j++)
      m[i*col+j]=0;
  return 0;
}


/* ------------------------------------
 *
 * output operations
 *
 * ------------------------------------
 */

/* basic cout */
ostream& operator<<(ostream & out, const  matrix &v1){
  if(v1.col > 1) {
    for (int i=0; i<v1.row;i++)
      for (int j=0; j<v1.col;j++)
// out << i << " " << j << "\t" << v1.m[i*v1.col+j] << endl;
out << i << " " << j << "\t" << v1(i,j) << endl;
  } else {
    for (int i=0; i<v1.row;i++)
out << i << "\t" << v1.m[i] << endl;
  }
  return out;
}

/* file output */
ofstream& operator<<(ofstream & out, const matrix &v1){
  for (int i=0; i<v1.row;i++)
    for (int j=0; j<v1.col;j++)
out << i << " " << j << "\t" << v1(i,j) << endl;
  return out;
}

/* Show the sparsity of a matrix schematically by printing
 * an 'X' for elements not equal to zero and a space otherwise.
 */
int matrix::showSparse() const {
  int i, j;
  for(i=0; i < row; i++) {
    for(j=0; j < col; j++) {
if(m[(i*col + j)] != 0) printf(" X");
else printf("  ");
    }
    printf("\n");
  }
  return 0;
}

// Same as above but allow for tolerance 'lim' rather that exact
zero
int matrix::showSparse(double lim) const {
  int i, j;
  for(i=0; i < row; i++) {
    for(j=0; j < col; j++) {
if(fabs(m[(i*col + j)]) > lim) printf(" X");
else printf("  ");
    }
    printf("\n");
  }
  return 0;
}
```

```
/* This was converted from fortram using f2c and 'polished' manu-
ally.  */

/* ****************************SSP246      NROOT */
/*
.............................................................
. */
/*        SUBROUTINE NROOT */

/*        PURPOSE */
/*          COMPUTE EIGENVALUES AND EIGENVECTORS OF A REAL NONSYM-
METRIC */
/*          MATRIX OF THE FORM B-INVERSE TIMES A.  THIS SUBROUTINE
IS */
/*          NORMALLY CALLED BY SUBROUTINE CANOR IN PERFORMING A */
/*           CANONICAL CORRELATION ANALYSIS. */

/*        USAGE */
/*           CALL NROOT (M,A,B,XL,X) */

/*        DESCRIPTION OF PARAMETERS */
/*           M  - ORDER OF SQUARE MATRICES A, B, AND X. */
/*           A  - INPUT MATRIX (M X M). */
/*           B  - INPUT MATRIX (M X M). */
/*          XL - OUTPUT VECTOR OF LENGTH M CONTAINING EIGENVALUES
OF */
/*                B-INVERSE TIMES A. */
/*           X  - OUTPUT MATRIX (M X M) CONTAINING EIGENVECTORS
COLUMN- */
/*                WISE. */

/*        REMARKS */
/*           NONE */

/*        SUBROUTINES AND FUNCTION SUBPROGRAMS REQUIRED */
/*           EIGEN */

/*        METHOD */
/*          REFER TO W. W. COOLEY AND P. R. LOHNES, 'MULTIVARIATE
PRO- */
/*           CEDURES FOR THE BEHAVIORAL SCIENCES', JOHN WILEY AND
SONS, */
/*           1962, CHAPTER 3. */
/*
.............................................................
. */

// int nroot(int dim, double *a, double *b, double *xl, double *x)
int matrix::nroot(matrix aa, matrix bb, matrix & vals, matrix &
vec) {
    /* Check matrix dimensions */
    if(aa.row != aa.col || bb.row != bb.col || vec.row != vec.col)
{
printf ("***Error::matrix::nroot matrices not square\n\n");
exit(-1);
    } else if(bb.row != aa.row || vec.row != aa.row
    || vals.row != aa.row || vals.col != 1) {
printf ("***Error::matrix::nroot bad dimensions\n\n");
exit(-1);
    }

    /* simulate original C call */
    int dim = aa.row;
    double *a = aa.m;
    double *b = bb.m;
    double *xl = vals.m;
    double *x = vec.m;

    /* System generated locals */
    int i__1, i__2, i__3;
    double r__1;

    /* Local variables */
    static double sumv;
    static int i__, j, k, l;
    static int n1, n2, mv;


/*     COMPUTE EIGENVALUES AND EIGENVECTORS OF B */

    /* Parameter adjustments */
    --x;
    --xl;
    --b;
    --a;
```

```
    /* Function Body */
    k = 1;
    i__1 = dim;
    for (j = 2; j <= i__1; ++j) {
l = dim * (j - 1);
i__2 = j;
for (i__ = 1; i__ <= i__2; ++i__) {
    ++l;
    ++k;
    b[k] = b[l];
}
    }

/*        THE MATRIX B IS A REAL SYMMETRIC MATRIX. */

    mv = 0;
    matrix::eigen(&b[1], &x[1], dim, &mv);

/*    FORM RECIPROCALS OF SQUARE ROOT OF EIGENVALUES.  THE RESULTS
*/
/*     ARE PREMULTIPLIED BY THE ASSOCIATED EIGENVECTORS. */

    l = 0;
    i__2 = dim;
    for (j = 1; j <= i__2; ++j) {
l += j;
xl[j] = 1.0 / sqrt( fabs(b[l]) );
    }
    k = 0;
    i__2 = dim;
    for (j = 1; j <= i__2; ++j) {
i__1 = dim;
for (i__ = 1; i__ <= i__1; ++i__) {
    ++k;
    b[k] = x[k] * xl[j];
}
    }

/*     FORM (B**(-1/2))PRIME * A * (B**(-1/2)) */

    i__1 = dim;
    for (i__ = 1; i__ <= i__1; ++i__) {
n2 = 0;
i__2 = dim;
for (j = 1; j <= i__2; ++j) {
    n1 = dim * (i__ - 1);
    l = dim * (j - 1) + i__;
    x[l] = (double)0.0;
    i__3 = dim;
    for (k = 1; k <= i__3; ++k) {
++n1;
++n2;
x[l] += b[n1] * a[n2];
    }
}
    }
    l = 0;
    i__3 = dim;
    for (j = 1; j <= i__3; ++j) {
i__2 = j;
for (i__ = 1; i__ <= i__2; ++i__) {
    n1 = i__ - dim;
    n2 = dim * (j - 1);
    ++l;
    a[l] = (double)0.0;
    i__1 = dim;
    for (k = 1; k <= i__1; ++k) {
n1 += dim;
++n2;
a[l] += x[n1] * b[n2];
    }
}
    }

/*     COMPUTE EIGENVALUES AND EIGENVECTORS OF A */

    matrix::eigen(&a[1], &x[1], dim, &mv);
    l = 0;
    i__1 = dim;
    for (i__ = 1; i__ <= i__1; ++i__) {
l += i__;
xl[i__] = a[l];
    }

/*     COMPUTE THE NORMALIZED EIGENVECTORS */
```

```
    i__1 = dim;
    for (i__ = 1; i__ <= i__1; ++i__) {
n2 = 0;
i__2 = dim;
for (j = 1; j <= i__2; ++j) {
    n1 = i__ - dim;
    l = dim * (j - 1) + i__;
    a[l] = (double)0.0;
    i__3 = dim;
    for (k = 1; k <= i__3; ++k) {
n1 += dim;
++n2;
a[l] += b[n1] * x[n2];
    }
}
    }
    l = 0;
    k = 0;
    i__3 = dim;
    for (j = 1; j <= i__3; ++j) {
sumv = (double)0.0;
i__2 = dim;
for (i__ = 1; i__ <= i__2; ++i__) {
    ++l;
    sumv += a[l] * a[l];
}
sumv = sqrt(sumv);
i__2 = dim;
for (i__ = 1; i__ <= i__2; ++i__) {
    ++k;
    x[k] = a[k] / sumv;
}
    }
    return 0;
} /* nroot_ */

/* ****************************SSP167      EIGEN */
/*        SUBROUTINE EIGEN */
/*        USAGE */
/*           CALL EIGEN(A,R,N,MV) */

/*        DESCRIPTION OF PARAMETERS */
/*           A - ORIGINAL MATRIX (SYMMETRIC), DESTROYED IN COMPU-
TATION. */
/*              RESULTANT EIGENVALUES ARE DEVELOPED IN DIAGONAL OF
*/
/*                MATRIX A IN DESCENDING ORDER. */
/*           R - RESULTANT MATRIX OF EIGENVECTORS (STORED COLUMN-
WISE, */
/*                IN SAME SEQUENCE AS EIGENVALUES) */
/*           N - ORDER OF MATRICES A AND R */
/*           MV- INPUT CODE */
/*                   0   COMPUTE EIGENVALUES AND EIGENVECTORS */
/*                  1   COMPUTE EIGENVALUES ONLY (R NEED NOT BE */
/*                      DIMENSIONED BUT MUST STILL APPEAR IN CALL-
ING */
/*                       SEQUENCE) */

/*        REMARKS */
/*           ORIGINAL MATRIX A MUST BE REAL SYMMETRIC (STORAGE
MODE=1) */
/*          MATRIX A CANNOT BE IN THE SAME LOCATION AS MATRIX R */

/*        METHOD */
/*           DIAGONALIZATION METHOD ORIGINATED BY JACOBI AND
ADAPTED */
/*           BY VON NEUMANN FOR LARGE COMPUTERS AS FOUND IN 'MATH-
EMATICAL */
/*           METHODS FOR DIGITAL COMPUTERS', EDITED BY A. RALSTON
AND */
/*           H.S. WILF, JOHN WILEY AND SONS, NEW YORK, 1962, CHAP-
TER 7 */
/*
.............................................................
. */

int matrix::eigen(double *a, double *r__, int dim_1, int *mv)
{
    /* System generated locals */
    int i__1, i__2, i__3;
    double r__1;

    /* Builtin functions */
    // double sqrt();
```

```
    /* Local variables */
    static double cosx, sinx, cosx2, sinx2;
    static int i__, j, k, l, m;
    static double x, y, range, anorm, sincs, anrmx;
    static int ia, ij, il, im, ll, lm, iq, mm, jq, lq, mq, ind,
ilq, imq,
    ilr, imr;
    static double thr;


/*        GENERATE IDENTITY MATRIX */
    /* Parameter adjustments */
    --r__;
    --a;

    /* Function Body */
    // range = (double)1e-12;
    range = 1e-12;
    if (*mv - 1 != 0) {
      iq = -(dim_1);
      i__1 = dim_1;
      for (j = 1; j <= i__1; ++j) {
iq += dim_1;
i__2 = dim_1;
for (i__ = 1; i__ <= i__2; ++i__) {
    ij = iq + i__;
    r__[ij] = 0.0;
    if (i__ - j != 0) {
// continue (goto L20;)
    } else {
    r__[ij] = 1.0;
    }
}
      }
    } else {
// continue (goto L25;)
    }


/*        COMPUTE INITIAL AND FINAL NORMS (ANORM AND ANORMX) */
    anorm = (double)0.;
    i__2 = dim_1;
    for (i__ = 1; i__ <= i__2; ++i__) {
i__1 = dim_1;
for (j = i__; j <= i__1; ++j) {
    if (i__ - j != 0) {
      ia = i__ + (j * j - j) / 2;
      anorm += a[ia] * a[ia];
    } else {
// continue (goto L35;)
    }
}
    }
    if (anorm <= (double)0.) {
goto L165;
    } else {
goto L40;
    }
L40:
    // anorm = sqrt(anorm) * (double)1.414;
    // use more sig figs
    anorm = sqrt(anorm) * (double)1.41421356237309;
    anrmx = anorm * range / (double) (dim_1);

/*        INITIALIZE INDICATORS AND COMPUTE THRESHOLD, THR */

    ind = 0;
    thr = anorm;
L45:
    thr /= (double) (dim_1);
L50:
    l = 1;
L55:
    m = l + 1;

/*        COMPUTE SIN AND COS */
L60:
    mq = (m * m - m) / 2;
    lq = (l * l - l) / 2;
    lm = l + mq;
/* L62: */
    if (fabs(a[lm]) - thr >= 0.0) {
goto L65;
    } else {
goto L130;
    }
```

```
L65:
    ind = 1;
    ll = l + lq;
    mm = m + mq;
    x = (a[ll] - a[mm]) * 0.5;
/* L68: */
    y = -a[lm] / sqrt(a[lm] * a[lm] + x * x);
    if (x >= 0.0) {
goto L75;
    } else {
goto L70;
    }
L70:
    y = -y;
L75:
//   75 SINX=Y/ SQRT(2.0*(1.0+( SQRT(1.0-Y*Y))))
    sinx = y / sqrt(2.0*(1.0+sqrt(1.0-y*y) ) );
    sinx2 = sinx * sinx;
/* L78: */
    cosx = sqrt(1.0 - sinx2);
    cosx2 = cosx * cosx;
    sincs = sinx * cosx;

/*        ROTATE L AND M COLUMNS */

    ilq = dim_1 * (l - 1);
    imq = dim_1 * (m - 1);
    i__1 = dim_1;
    for (i__ = 1; i__ <= i__1; ++i__) {
iq = (i__ * i__ - i__) / 2;
if (i__ - l != 0) {
    goto L80;
} else {
    goto L115;
}
L80:
if ((i__2 = i__ - m) < 0) {
    goto L85;
} else if (i__2 == 0) {
    goto L115;
} else {
    goto L90;
}
L85:
im = i__ + mq;
goto L95;
L90:
im = m + iq;
L95:
if (i__ - l >= 0) {
   il = l + iq;
} else {
   il = i__ + lq;
}

x = a[il] * cosx - a[im] * sinx;
a[im] = a[il] * sinx + a[im] * cosx;
a[il] = x;
L115:
if (*mv - 1 != 0) {
    goto L120;
} else {
    goto L125;
}
L120:
ilr = ilq + i__;
imr = imq + i__;
x = r__[ilr] * cosx - r__[imr] * sinx;
r__[imr] = r__[ilr] * sinx + r__[imr] * cosx;
r__[ilr] = x;
L125:
;
    }
    x = a[lm] * 2.0 * sincs;
    y = a[ll] * cosx2 + a[mm] * sinx2 - x;
    x = a[ll] * sinx2 + a[mm] * cosx2 + x;
    a[lm] = (a[ll] - a[mm]) * sincs + a[lm] * (cosx2 - sinx2);
    a[ll] = y;
    a[mm] = x;

/*        TESTS FOR COMPLETION */

/*        TEST FOR M = LAST COLUMN */

L130:
    if (m - dim_1 != 0) {
```

```
goto L135;
    } else {
goto L140;
    }
L135:
    ++m;
    goto L60;

/*        TEST FOR L = SECOND FROM LAST COLUMN */

L140:
    if (l - (dim_1 - 1) != 0) {
goto L145;
    } else {
goto L150;
    }
L145:
    ++l;
    goto L55;
L150:
    if (ind - 1 != 0) {
goto L160;
    } else {
goto L155;
    }
L155:
    ind = 0;
    goto L50;

/*        COMPARE THRESHOLD WITH FINAL NORM */

L160:
    if (thr - anrmx <= 0.0) {
goto L165;
    } else {
goto L45;
    }

/*        SORT EIGENVALUES AND EIGENVECTORS */

L165:
    iq = -(dim_1);
    i__1 = dim_1;
    for (i__ = 1; i__ <= i__1; ++i__) {
iq += dim_1;
ll = i__ + (i__ * i__ - i__) / 2;
jq = dim_1 * (i__ - 2);
i__2 = dim_1;
for (j = i__; j <= i__2; ++j) {
    jq += dim_1;
    mm = j + (j * j - j) / 2;
    if (a[ll] - a[mm] >= (double)0.0) {
goto L185;
    } else {
goto L170;
    }
L170:
    x = a[ll];
    a[ll] = a[mm];
    a[mm] = x;
    if (*mv - 1 != 0) {
goto L175;
    } else {
goto L185;
    }
L175:
    i__3 = dim_1;
    for (k = 1; k <= i__3; ++k) {
ilr = iq + k;
imr = jq + k;
x = r__[ilr];
r__[ilr] = r__[imr];
/* L180: */
r__[imr] = x;
    }
L185:
    ;
}
    }
    return 0;
} /* eigen */
```

```
File: mymath.cpp
// mymath.cpp:
//
//////////////////////////////////////////////////////////

#include <stdlib.h>
#include <fstream.h> // contains fstream
#include <iostream.h>                // cin, cout, <<, >>
#include <iomanip.h> // formatted I/O
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "mymath.h"

#define SIZE1 200
#define SIZE1_2 400

int sign(double val)
{
if(val < 0) {
  return -1;
} else {
  return  1;
}
}

int invert(double noUse[SIZE1][SIZE1], double Ainv[SIZE1][SIZE1],
int dim)
{
int i, ii, j, jj;
double A[SIZE1][SIZE1_2], temp;

for(i=0; i<dim; i++) { // initialize augmented matrix
  for(j=0; j<dim; j++) {
A[i][j+dim] = 0.0;
A[i][j] = noUse[i][j];
  }
  A[i][i+dim] = 1.0;
}

// create upper triangular with 1.0 on diagonal
for(i=0; i<dim; i++) {
  // set leading 1 in this row
  temp = A[i][i];
  for(j=i; j<2*dim; j++)  A[i][j] = A[i][j] / temp;
  for(ii=i+1; ii<dim; ii++) { // set leading 0 in other rows
    temp = A[ii][i];
    for(jj=i; jj<2*dim; jj++) {
A[ii][jj] = A[ii][jj] - temp * A[i][jj];
    }
  }
}

// now create I from upper triangular; zero the upper part
for(j=1; j<dim; j++) {

  // ii is the row with 1.0 on diagonal in the (i+1) column

  for(ii=0; ii<j; ii++) { // set 0 in all other rows
    temp = A[ii][j];
    for(jj=j; jj<2*dim; jj++) {
A[ii][jj] = A[ii][jj] - temp * A[j][jj];
    }
  }
}

for(i=0; i<dim; i++)
  for(j=0; j<dim; j++)
Ainv[i][j] = A[i][j+dim];

return 0;
}

int showSparse(double A[SIZE1][SIZE1], int rows, int cols)
{
for(int i=0; i < rows; i++) {
  for(int j=0; j < cols; j++) {
if(A[i][j] != 0) {
  printf(" X");
} else {
  printf("  ");
}
  }
  printf("\n");
}
```

```
return 0;
}

int transpose(double A[SIZE1][SIZE1], double Atrans[SIZE1][SIZE1],
int Arows, int Acols)
{
for(int i=0; i < Arows; i++)
  for(int j=0; j < Acols; j++)
Atrans[j][i] = A[i][j];

return 0;
}

int product(double A[SIZE1][SIZE1], double B[SIZE1][SIZE1], double
C[SIZE1][SIZE1], int Arows, int Acols, int Bcols)
{
// multiply A x B and return in C
// use a temporary array for the product so that
// this can be called for A = A * B

int i, ii, j, jj;
double temp[SIZE1][SIZE1];

for(int i=0; i < Arows; i++)// zero temp
  for(int j=0; j < Bcols; j++)
temp[i][j] = 0.0;

for(int i=0; i < Arows; i++) {// form product
  for(int j=0; j < Bcols; j++) {
for(int k=0; k < Acols; k++)
  temp[i][j] += A[i][k] * B[k][j];
  }
}

for(int i=0; i < Arows; i++)// copy temp to C
  for(int j=0; j < Bcols; j++)
C[i][j] = temp[i][j];

return 0;
}
```

File: opfr2.cpp

```
// opfr2.cpp
//
// Program for 2D frame optimization.
//

#include <stdlib.h>
#include <fstream.h> // contains fstream
#include <iostream.h>                          // cin, cout, <<, >>
#include <iomanip.h>                             // setiosflags()
                                                 // setprecision()
#include <math.h>                                // sqrt()
#include <stdio.h>
#include <string.h>
#include <time.h>

#include "args.h"
#include "constraint.h"
#include "fr2d.h"
#include "iplp.h"
#include "mymath.h"
#include "utility.h"

#define SIZE2 300
#define LARGE 1000000000

int analyzeAndScale(fr2d * f, constraint * g, args a);

int main(int argc, char * argv[])
{
char constraintsFileName[80],structureFileName[80],
outFileName[20], jobName[20];
int i, j, k, ii, iii, lpStatus, tempi, iter, iterStart, iterStop;
int itermax, lpIter, lpVerb, printEvery, restart, timing, verbose;
int gNonLinearIter;
double alpha, explore, scale, tempd;
FILE *structureFile, *constraintsFile, *outFile, *errorFile;

args arg("default");
constraint *f, *g;
fr2d *frame;
iplp *lp;
```

```
// process command line args
// strcpy(jobName, "file");/* default jobName */
if(argc == 1) killme("\tOptimization of nonlinear 2D frame\n");
// printf(" read command line args ...\n");
for(i=1; i<argc; i++)
  if( arg.parseArg(argv[i]) ) killme("parseArg, bad arguments\n",
-1);
if (arg.getIntegerFlag('e') == 0) printEvery = LARGE;

// These should be defined in arg("default") but may have been
changed
// by the command line arguments.
itermax= arg.getIntegerFlag('i');
lpIter= arg.getIntegerFlag('I');
lpVerb = arg.getIntegerFlag('V');
restart = arg.getIntegerFlag('r');
verbose = arg.getIntegerFlag('v');
printEvery = arg.getIntegerFlag('e');
alpha = arg.getDoubleFlag('a');// step size limit
explore = arg.getDoubleFlag('x');// scale-back into feasible
arg.getCharsFlag('b', jobName);

// Arguments that may not exist on the command line
// and may not have been assigned a value.
timing = arg.isDefined('t');
gNonLinearIter = (arg.isDefined('g') ? arg.getIntegerFlag('g') :
0);
if(arg.isDefined('g') && arg.getIntegerFlag('g') == 0) gNonLin-
earIter = 1;

if(verbose) printf("\n");
if(timing) showTime("begin \t");
if(verbose) arg.showAll();

// set filenames
if(verbose > 2) printf(" set filenames ...\n");
strcpy(constraintsFileName, jobName);
strcat(constraintsFileName, ".osc");
strcpy(structureFileName, jobName);
if(restart) strcat(structureFileName, num2str(restart));
strcat(structureFileName, ".osi");

// read constraints for the model
if(verbose > 2) printf(" read constraints ...\n");
f=new constraint(constraint::countNumber(constraintsFile-
Name,"*OBJECTIVE"), 1);
g=new constraint(constraint::countNumber(constraintsFile-
Name,"*CONSTRAINTS"), 0);
if(verbose > 2) printf(" read constraints variable ...\n");
if( constraint::readConstraints(constraintsFileName, f, g) )
  killme("\n ***Error::main readConstraints failed \n ", 1);
if(timing) showTime("done read constraints ");

if (verbose > 0) printf("num vars: %d\nnum cons: %d\n", f->number,
g->number);
if (verbose == -2) {
  // printf("main:: f.showValues():\n");
  f->showValues();
  g->showValues();
}

// read initial model design
if(verbose > 2) printf(" read structure file ...");
if(g->findItem('G'))
  frame = new fr2d( fr2d::getDimensions(structureFileName), "non-
linear" );
else frame = new fr2d( fr2d::getDimensions(structureFileName) );
frame->readFile(structureFileName);
if(verbose > 2) printf("  done.\n");
if(timing) showTime("done read model \t");
frame->initialize();
frame->analyze();
if (verbose > 6) {
  frame->echoStructure();
  frame->echoJointDisplacements();
  frame->echoMemberDisplacements();
}
if(timing) showTime("done analyze model \t");


// check problem dimensions
if(f->number > SIZE2) killme("***main: f->number must be less than
SIZE2");
```

```
if(g->number > SIZE2) killme("***main: g->number must be less than
SIZE2");
if(verbose > 10)
  printf("constraint dimensions \n\tvariables: %d\n\tconstraints:
%d\n\t ..OK\n",
    f->number, g->number);

// allocate space for the LP
// lp = new iplp(g->number, f->number);
lp = new iplp(f->number, g->number+2*f->number);
lp->verbose = lpVerb;
lp->itermax = lpIter;
if(verbose > 10) printf("done allocate space for LP\n");

//////////////////////////////////////////////////////
//
// Set up an initial solution before starting iterations.
//
if(verbose > 10) printf("set up initial solution\n");
analyzeAndScale(frame, g, arg);
if(timing) showTime("done analyzeAndScale \t");

matrix lastDA(frame->NB);

/*************************************************************
 *
 * iterate
 *
 */
iterStart = restart+1;/* default 'restart' is 0 */
iterStop  = iterStart + itermax;

// if(verbose == 0) printf("begin iterations\n");
for (iter=iterStart; iter<iterStop; iter++)
{
if(verbose > 0) printf("ITERATION: %d\n", iter);
if(timing) showTime("begin iteration \t", iter);
setFileName(jobName, outFileName, iter);
// printf(" outfile name: %s \n", outFileName);


// Assemble c, A, and b for LP problem: min c^T x s.t. Ax <= b
//
// Note: using dual formulation
//  maximize b^T y s.t. A^T y <= c,
//
// printf("main: set up LP problem\n");
if (verbose > 2) printf("\t frame->mkF \n");
frame->mkF(lp->b, f);
lp->b = -lp->b;

   // when using nonlinear terms in LP, need to solve for second
order
   // terms iteratively. Start with 'lastDA' = 0, then solve for dA
'lp->y'
   // and then use this value to solve for a new dA.
   lastDA.setZero();
   iii = 0;
   do {
// Create a temporary matrix for A because using
// the dual formulation here and need A^T.
//
if (verbose > 2) printf("\t frame->mkG \n");
matrix lp_A_temp(lp->A.col, lp->A.row);
if(gNonLinearIter) frame->mkG(lp_A_temp, lp->c, g, f->number,
lastDA.m);
else frame->mkG(lp_A_temp, lp->c, g, f->number);
lp->A = lp_A_temp.transpose();
lp_A_temp.del();

/* limits on variables (step size) */
for (i=0; i < f->number; i++) {
  /* NOTE: THIS ASSUMES CONSTRAINTS ARE ORDERED */
  /* b = f.loc[i];// bar number */
  j = i+1;// bar number
  lp->c[g->number + 2*i]   = alpha * frame->bars[j].area;
  lp->c[g->number + 2*i+1] = alpha * frame->bars[j].area;
  lp->A(i, g->number + 2*i)   =  1.0;
  lp->A(i, g->number + 2*i+1) = -1.0;
  if (verbose > 6) printf("limit %d: +/- %lf\n", j, alpha*frame-
>bars[j].area);
}

/* starting point */
// lp->printLP();
lpStatus = lp->startY0();
```

```
if(lpStatus) killme("dualStart failed (nonpositive 'c')", lpSta-
tus);

// echo lp details
if(verbose == -1) lp->printLP();

/* solve LP problem */
if(timing) showTime("done forming LP \t");
if(verbose > 6) printf("main: solve LP problem\n");
lpStatus = lp->solveDual();


if(gNonLinearIter && ((verbose > 6) || (verbose > 4 && iii ==
gNonLinearIter)) )
  cout << "using nonlinear terms, iter " << iii << ", error \n"
    << lp->y - lastDA << endl;
lastDA = lp->y;

if(lpStatus) {
  printf("\n ***Warning:main LP solver failed, lpStatus: %d\n",
lpStatus);
  // exit(0);
}
if(timing) showTime("done solving LP \t");
// printf(" iii %d, nonliniter %d \n", iii, gNonLinearIter);
   } while(iii++ < gNonLinearIter);
/* show solution */
if(verbose > 2) cout << " LP solved\n lpStatus: " << lpStatus
<< " \n vars: \n" << lp->y << endl;

// update variables
for(i=1; i<=frame->NB; i++) {
  frame->bars[i].area += lp->y[i-1];
  frame->bars[i].section_I = frame->a2i(i);
}
if(verbose > 20) {
  printf("\n\t new step vvvv\n");
  frame->echoStructure();
}
if(timing) showTime("done update variables ");


// scale the solution and write this iteration
analyzeAndScale(frame, g, arg);
if(timing) showTime("done analyzeAndScale \n");
if( iter == iterStart || iter == (iterStop-1) || !(iter%printEv-
ery) ) {
  // create 'exact' solution for writing
  // analyzeAndScale(frame, g, arg);
  if (verbose > 1) printf("\t end vol: \t%15.5f \n", frame->vol-
ume() );
  if (verbose > 4) {
    // printf("\n\t vvvv scaled step vvvv\n");
    frame->echoStructure();
    frame->echoMemberDisplacements();
  }
  if(verbose > 2 ) printf("\t end %d: write %s \n", iter, outFile-
Name);
  frame->writeFile(outFileName);
}

// scale to remain interior for next iteration
// analyzeAndScale(frame, g, arg);
if(timing) showTime("end iteration \t");

}

/*
 * Close iteration loop
 *
 ***************************************************************
 *
 * Perform final scaling and show final solution
 *
 */

if(verbose > 0) printf("\nDone iterations\n");
if(timing) showTime("end main \t");
// tempd = frame->volume();
// analyzeAndScale(frame, g, arg);
// if(verbose > 3) frame->echoStructure();
// if(verbose > 0) frame->echoMemberDisplacements();

printf("\n OK, exit\n");

return 0;
```

```
}

/*
 * END MAIN PROGRAM
 *
 ***********************************************************/

/***********************************************************
 *
 * Helper function to scale the structure
 *
 */

int analyzeAndScale(fr2d * fr, constraint * g, args a) {
int i, j, iterMax, numLoadSteps, vbs;
double tempd, fac1, ex;

numLoadSteps = (a.isDefined('L') ? a.getIntegerFlag('L') : 1);
if(a.isDefined('v')) vbs = a.getIntegerFlag('v');
ex = a.getDoubleFlag('x');
vbs   = (a.isDefined('v') ? a.getIntegerFlag('v')-1 : 0);
// vbs = 5;
iterMax = 5;

if(vbs > 0) printf("\n\nvvvvvvvvvvvvv\nanalyzeAndScale: begin\n");
// A nonlinear analysis implies that we want to be near the
// buckling load. Scale accordingly.
//
if(fr->nonlinear) {
  if(vbs > 3) printf("first nonlinear analysis\n");
  if(vbs > 3 && a.isDefined('L')) printf("using %d load-
steps\n",numLoadSteps);
  j = fr->analyze(numLoadSteps,5);

  // matrix::nroot(fr->Kg, fr->Ke, fr->frequencies, fr->mode-
Shapes);
  // cout << " analyzeAndScale: before scale: frequencies\n"
  //   << fr->frequencies << endl;
  // cout << " analyzeAndScale: before scale: modeShapes\n"
  //   << fr->modeShapes << endl;
  if(numLoadSteps > 1) {
    // scaling by number of loadsteps
    fac1 = (j == 0 ? 0.9 : (double)numLoadSteps / (double)j );
  } else {
    // scaling by eigenvalue
    if(j) matrix::nroot(fr->Kg, fr->Ke, fr->frequencies, fr->mode-
Shapes);
    fac1 = (j == 0 ? 0.9 : fabs(fr->frequencies[fr->dof - 1]) );
  }
  fac1 = sqrt(fac1);
  if(vbs > 2)
    if(j) printf("analyzeAndScale: buckle at iter %d, \n", j);
    else printf("analyzeAndScale: buckling load not reached \n");

  // If the solution is in a number of loadsteps, attempt to scale
  // structure so that it buckles at the last loadstep 'numLoad-
Steps'.
  // Use a maximum of 'iterMax' attempts.
  i = 0;

  do {
    if(vbs > 2 && numLoadSteps > 1) printf(" iter %d of %d ", i,
iterMax);
    if(vbs > 2) printf(" scale by %f\n", fac1);
    fr->scale(g, fac1);
    j = fr->analyze(numLoadSteps,5);
    // matrix::nroot(fr->Kg, fr->Ke, fr->frequencies, fr->mode-
Shapes);
    // fac1 = fabs(fr->frequencies[fr->dof - 1]);
    fac1 = (j == 0 ? 0.9 : (double)numLoadSteps / (double)j );
    fac1 = sqrt(fac1);
    if(vbs > 2)
  if(j) printf(" analyzeAndScale: buckle at iter %d\n", j);
  else printf(" analyzeAndScale: buckling load not reached \n");
  } while(numLoadSteps > 1 && j != numLoadSteps && i++ < iterMax);

  if(i > iterMax) {
    printf("\n\n***analyzeAndScale: buckling convergence problem
\n");
    printf("\t*** calling 'matrix::nroot()' may fail....? \n");
  }
  matrix::nroot(fr->Kg, fr->Ke, fr->frequencies, fr->modeShapes);

  // printf(" lambda: %f \n", fabs(fr->frequencies[fr->dof - 1])
);
```

```
  // cout << " analyzeAndScale: frequencies\n" << fr->frequencies
<< endl;
  // cout << " analyzeAndScale: modeShapes\n" << fr->modeShapes <<
endl;
} else {
  /* Linear analysis */
  fr->analyze();
  if (vbs > 4) {
    printf("\t initial soln:\n");
    fr->echoMemberDisplacements();
  }
  i = 0;
  // Try using 'ex' for explore factor the first time but if more
  // iterations are needed, use the amount of violation, 'tempd'
  // found from the previous scaling.
  tempd = ex;
  do {
    if(i) printf(" Re-scaling: %d of 3\n", i);
    // use 1.1 to ensure feasibility
    // if(tempd < 1.0) tempd = ex;
    // printf("analyzeAndScale: call fr->scale with explore %f
\n", tempd);
    // tempd = fr->scale(g, tempd);
    tempd = fr->scale(g, ex);
    // printf("analyzeAndScale: done fr->scale \n");
    // printf("analyzeAndScale: fr->scale factor %f \n", tempd);
    // printf("analyzeAndScale: call fr->analyze ...\n");
    fr->analyze();
    if (vbs > 3) {
printf("\n\t scale factor (by alpha = %f): %f\n", ex, tempd);
printf("\n\t initial scaled feasible soln:\n");
fr->echoMemberDisplacements();
    }
  } while(fr->isViolated(g) && i++ < 3);
  // printf("\n\nanalyzeAndScale: outside do block, i = %d\n", i);
  if(i >= 3) killme("analyzeAndScale: rescale failed");
}

if(vbs > 0) printf("\nanalyzeAndScale: end\n^^^^^^^^^^^^^\n");
return 0;

}
```

File: opfr3.cpp

```
// opfr3.cpp
//
// Program for 2D frame optimization.
//

#include <stdlib.h>
#include <fstream.h> // contains fstream
#include <iostream.h>                          // cin, cout, <<, >>
#include <iomanip.h>                             // setiosflags()
                                                 // setprecision()
#include <math.h>                                // sqrt()
#include <stdio.h>
#include <string.h>
#include <time.h>

#include "args.h"
#include "constraint.h"
#include "fr3d.h"
#include "iplp.h"
#include "mymath.h"
#include "utility.h"

#define DIM1 250
#define LARGE 1000000000

int analyzeAndScale(fr3d * f, constraint * g, args a);

int main(int argc, char * argv[])
{
char constraintsFileName[80],structureFileName[80],
outFileName[20], jobName[20];
int i, j, k, ii, iii, lpStatus, tempi, iter, iterStart, iterStop;
int itermax, lpIter, lpVerb, printEvery, restart, timing, verbose;
int gNonLinearIter;
double alpha, explore, scale, tempd;
FILE *structureFile, *constraintsFile, *outFile, *errorFile;

args arg;
constraint *f, *g;
fr3d *frame;
iplp *lp;
```

```
// process command line args
// strcpy(jobName, "file");/* default jobName */
if(argc == 1) killme("\tOptimization of (non)linear 3D frame\n");
// printf(" read command line args ...\n");
for(i=1; i<argc; i++)
  if( arg.parseArg(argv[i]) ) killme("parseArg, bad arguments\n",
-1);

// Set defaults of args that weren't found on
// the command line.
if(!arg.isDefined('i')) arg.parseArg("-i1");
if(!arg.isDefined('I')) arg.parseArg("-I10");
if(!arg.isDefined('V')) arg.parseArg("-V0");
if(!arg.isDefined('r')) arg.parseArg("-r0");
if(!arg.isDefined('v')) arg.parseArg("-v0");
if(!arg.isDefined('e')) arg.parseArg("-e1");
if(!arg.isDefined('a')) arg.parseArg("-a0.2");
if(!arg.isDefined('x')) arg.parseArg("-x1.1");

itermax= arg.getIntegerFlag('i');
lpIter= arg.getIntegerFlag('I');
lpVerb = arg.getIntegerFlag('V');
restart = arg.getIntegerFlag('r');
verbose = arg.getIntegerFlag('v');
printEvery = arg.getIntegerFlag('e');
alpha = arg.getDoubleFlag('a');
explore = arg.getDoubleFlag('x');

arg.getCharsFlag('b', jobName);
if(arg.getIntegerFlag('e') == 0) printEvery = LARGE;

// Arguments that may not exist on the command line
// and may not have been assigned a value.
timing = arg.isDefined('t');
gNonLinearIter = (arg.isDefined('g') ? arg.getIntegerFlag('g') :
0);
if(arg.isDefined('g') && arg.getIntegerFlag('g') == 0) gNonLin-
earIter = 1;

if(verbose > 0) printf("\n");
if(timing) showTime("begin \t");
if(verbose) arg.showAll();
```

```
// set filenames
if(verbose > 2) printf(" set filenames ...\n");
strcpy(constraintsFileName, jobName);
strcat(constraintsFileName, ".osc");
strcpy(structureFileName, jobName);
if(restart) strcat(structureFileName, num2str(restart));
strcat(structureFileName, ".osi");


// read constraints for the model
if(verbose > 2) printf(" read constraints ...\n");
f=new constraint(constraint::countNumber(constraintsFile-
Name,"*OBJECTIVE"), 1);
g=new constraint(constraint::countNumber(constraintsFile-
Name,"*CONSTRAINTS"), 0);
if(verbose > 2) printf(" read constraints variable ...\n");
if( constraint::readConstraints(constraintsFileName, f, g) )
  killme("\n ***Error::main readConstraints failed \n ", 1);
if(timing) showTime("done read constraints ");

if (verbose > 0) printf("num vars: %d\nnum cons: %d\n", f->number,
g->number);
if (verbose == -2) {
  // printf("main:: f.showValues():\n");
  f->showValues();
  g->showValues();
}

// read initial model design
if(verbose > 2) printf(" read structure file ...");
if(g->findItem('G'))
  frame = new fr3d( fr3d::getDimensions(structureFileName), "non-
linear" );
else frame = new fr3d( fr3d::getDimensions(structureFileName) );
frame->readFile(structureFileName);
if(verbose > 2) printf("  done.\n");
if(timing) showTime("done read model \t");
frame->initialize();
frame->analyze();
if (verbose > 6) {
  frame->echoStructure();
  frame->echoJointDisplacements();
```

```
  frame->echoMemberDisplacements();
}
if(timing) showTime("done analyze model \t");


// check problem dimensions
if(f->number != frame->NB) killme("***main: f->number must be
equal 'NB'");
if(f->number > DIM1) killme("***main: f->number must be less than
DIM1");
if(g->number > DIM1) killme("***main: g->number must be less than
DIM1");
if(verbose > 10)
  printf("constraint dimensions \n\tvariables: %d\n\tconstraints:
%d\n\t ..OK\n",
    f->number, g->number);

// allocate space for the LP
// lp = new iplp(g->number, f->number);
lp = new iplp(f->number, g->number+2*f->number);
lp->verbose = lpVerb;
lp->itermax = lpIter;
if(verbose > 10) printf("done allocate space for LP\n");

//////////////////////////////////////////////////////////
//
// Set up an initial solution before starting iterations.
//
if(verbose > 10) printf("set up initial solution\n");
analyzeAndScale(frame, g, arg);
if(timing) showTime("done analyzeAndScale \t");

matrix lastDA(frame->NB);

/*************************************************************
 *
 * iterate
 *
 */
iterStart = restart+1;/* default 'restart' is 0 */
iterStop  = iterStart + itermax;

// if(verbose == 0) printf("begin iterations\n");
for (iter=iterStart; iter<iterStop; iter++)
{
if(verbose > 0) printf("ITERATION: %d\n", iter);
if(timing) showTime("begin iteration \t", iter);
setFileName(jobName, outFileName, iter);
// printf(" outfile name: %s \n", outFileName);


// Assemble c, A, and b for LP problem: min c^T x s.t. Ax <= b
//
// Note: using dual formulation
//  maximize b^T y s.t. A^T y <= c,
//
// printf("main: set up LP problem\n");
if (verbose > 2) printf("\t frame->mkF \n");
frame->mkF(lp->b, f);
lp->b = -lp->b;

   // when using nonlinear terms in LP, need to solve for second
order
   // terms iteratively. Start with 'lastDA' = 0, then solve for dA
'lp->y'
   // and then use this value to solve for a new dA.
   lastDA.setZero();
   iii = 0;
   do {
// Create a temporary matrix for A because using
// the dual formulation here and need A^T.
//
if (verbose > 2) printf("\t frame->mkG \n");
matrix lp_A_temp(lp->A.col, lp->A.row);
if(gNonLinearIter) frame->mkG(lp_A_temp, lp->c, g, f->number,
lastDA.m);
else frame->mkG(lp_A_temp, lp->c, g, f->number);
lp->A = lp_A_temp.transpose();
lp_A_temp.del();

/* limits on variables (step size) */
for (i=0; i < f->number; i++) {
  /* NOTE: THIS ASSUMES CONSTRAINTS ARE ORDERED */
  /* b = f.loc[i];// bar number */
  j = i+1;// bar number
  lp->c[g->number + 2*i]   = alpha * frame->bars[j].area;
```

```
  lp->c[g->number + 2*i+1] = alpha * frame->bars[j].area;
  lp->A(i, g->number + 2*i)    =  1.0;
  lp->A(i, g->number + 2*i+1) = -1.0;
  if (verbose > 6) printf("limit %d: +/- %lf\n", j, alpha*frame-
>bars[j].area);
}

/* starting point */
// lp->printLP();
lpStatus = lp->startY0();
if(lpStatus) killme("dualStart failed (nonpositive 'c')", lpSta-
tus);

// echo lp details
if(verbose == -1) lp->printLP();

/* solve LP problem */
if(timing) showTime("done forming LP \t");
if(verbose > 6) printf("main: solve LP problem\n");
lpStatus = lp->solveDual();


if(gNonLinearIter && ((verbose > 6) || (verbose > 4 && iii ==
gNonLinearIter)) )
  cout << "using nonlinear terms, iter " << iii << ", error \n"
    << lp->y - lastDA << endl;
lastDA = lp->y;

if(lpStatus) {
  printf("\n ***Warning:main LP solver failed, lpStatus: %d\n",
lpStatus);
  // exit(0);
}
if(timing) showTime("done solving LP \t");
// printf(" iii %d, nonliniter %d \n", iii, gNonLinearIter);
   } while(iii++ < gNonLinearIter);
/* show solution */
if(verbose > 2) cout << " LP solved\n lpStatus: " << lpStatus
<< " \n vars: \n" << lp->y << endl;

// update variables
for(i=1; i<=frame->NB; i++) {
  frame->bars[i].area += lp->y[i-1];
  frame->bars[i].Ix = frame->a2ix(i);
  frame->bars[i].Iy = frame->a2iy(i);
  frame->bars[i].Iz = frame->a2iz(i);
}
if(verbose > 20) {
  printf("\n\t new step vvvv\n");
  frame->echoStructure();
}
if(timing) showTime("done update variables ");

// scale the solution and write this iteration
analyzeAndScale(frame, g, arg);
if(timing) showTime("done analyzeAndScale ");
if( iter == iterStart || iter == (iterStop-1) || !(iter%printEv-
ery) ) {
  // create 'exact' solution for writing
  // analyzeAndScale(frame, g, arg);
  if (verbose > 1) printf("\t end vol: \t%15.5f \n", frame->vol-
ume() );
  if (verbose > 4) {
    // printf("\n\t vvvv scaled step vvvv\n");
    frame->echoStructure();
    frame->echoMemberDisplacements();
  }
  if(verbose > 2 ) printf("\t end %d: write %s \n", iter, outFile-
Name);
  frame->writeFile(outFileName);
}

// scale to remain interior for next iteration
// analyzeAndScale(frame, g, arg);
if(timing) showTime("end iteration \t");

}

/*
 * Close iteration loop
 *
 ***************************************************************
 *
 * Perform final scaling and show final solution
 *
 */
```

```
if(verbose > 0) printf("\nDone iterations\n");
if(timing) showTime("end main \t");
// tempd = frame->volume();
// analyzeAndScale(frame, g, arg);
// if(verbose > 3) frame->echoStructure();
// if(verbose > 0) frame->echoMemberDisplacements();

printf("\n OK, exit\n");

return 0;
}

/*
 *
 * END MAIN PROGRAM
 *
 ************************************************************/


/************************************************************
 *
 * Helper function to scale the structure
 *
 */

int analyzeAndScale(fr3d * fr, constraint * g, args a) {

int i, j, iterMax, numLoadSteps, timing, vbs;
double tempd, fac1, ex;

numLoadSteps = (a.isDefined('L') ? a.getIntegerFlag('L') : 1);
ex = a.getDoubleFlag('x');
vbs = a.getIntegerFlag('v')-1;
// vbs = 5;
iterMax = 5;
timing = a.isDefined('t');

if(vbs > 0) printf("\n\nvvvvvvvvvvvv\nanalyzeAndScale: begin\n");
// A nonlinear analysis implies that we want to be near the
// buckling load. Scale accordingly.
//
if(fr->nonlinear) {
  if(vbs > 3) printf("first nonlinear analysis\n");
  if(vbs > 3 && a.isDefined('L')) printf("using %d load-
steps\n",numLoadSteps);
  j = fr->analyze(numLoadSteps,5);
  if(timing) showTime("analyzeAndScale:: done analyze");
  // printf("analyzeAndScale: steps %d, status %d\n", numLoad-
Steps, j);

  if(numLoadSteps > 1) {
    // scaling by number of loadsteps
    fac1 = (j == 0 ? 0.9 : (double)numLoadSteps / (double)j );
  } else {
    // scaling by eigenvalue
    if(j && vbs > 2) printf("analyzeAndScale: call nroot for scale
factor\n");
    if(j) matrix::nroot(fr->Kg, fr->Ke, fr->frequencies, fr->mode-
Shapes);
    if(timing) showTime("analyzeAndScale:: done nroot for fac1");
    fac1 = (j == 0 ? 0.9 : fabs(fr->frequencies[fr->dof - 1]) );
  }
  fac1 = sqrt(fac1);

  // If the solution is in a number of loadsteps, attempt to scale
  // structure so that it buckles at the last loadstep 'numLoad-
Steps'.
  // Use a maximum of 'iterMax' attempts.
  i = 0;
  do {
    if(vbs > 2 && numLoadSteps > 1) printf(" iter %d of %d ", i,
iterMax);
    if(vbs > 2) printf(" scale by %f\n", fac1);
    fr->scale(g, fac1);
    j = fr->analyze(numLoadSteps,5);
    if(timing) showTime("analyzeAndScale:: done analyze iter", i);
    fac1 = (j == 0 ? 0.9 : (double)numLoadSteps / (double)j );
    fac1 = sqrt(fac1);
    if(vbs > 2)
  if(j) printf(" analyzeAndScale: buckle at iter %d\n", j);
  else printf(" analyzeAndScale: buckling load not reached \n");
  } while(numLoadSteps > 1 && j != numLoadSteps && i++ < iterMax);

  if(i > iterMax) {
```

```
    printf("\n\n***analyzeAndScale: buckling convergence problem
\n");
    printf("\t*** calling 'matrix::nroot()' may fail....? \n");
  }
  matrix::nroot(fr->Kg, fr->Ke, fr->frequencies, fr->modeShapes);
  if(timing) showTime("analyzeAndScale:: done nroot final");

  // printf(" lambda: %f \n", fabs(fr->frequencies[fr->dof - 1])
);
} else {
  /* Linear analysis */
  fr->analyze();
  if (vbs > 4) {
    printf("\t initial soln:\n");
    fr->echoMemberDisplacements();
  }
  i = 0;
  do {
    if(i) printf(" Re-scaling: %d of 3\n", i);
    // use 1.1 to ensure feasibility
    tempd = fr->scale(g, ex);
    fr->analyze();
    if (vbs > 3) {
printf("\n\t scale factor (by alpha = %f): %f\n", ex, tempd);
printf("\n\t initial scaled feasible soln:\n");
fr->echoMemberDisplacements();
    }
  } while(fr->isViolated(g) && i++ < 3);
}

if(vbs > 0) printf("\nanalyzeAndScale: end\n^^^^^^^^^^^^^^\n");
return 0;

}
```

File: optr2.cpp

```
// optr2.cpp: implementation of the main class.
//

#include <stdlib.h>
#include <fstream.h> // contains fstream
#include <iostream.h>                          // cin, cout, <<, >>
#include <iomanip.h>                            // setiosflags(),
setprecision()
#include <math.h>                               // sqrt()
#include <stdio.h>
#include <string.h>

#include "args.h"
#include "constraint.h"
#include "iplp.h"
#include "matrix.h"
#include "mymath.h"
#include "tr2d.h"
#include "utility.h"


#define SIZE2 1000
#define LARGE 1000000000
#define SMALL 0.0000000001


int main(int argc, char **argv)
{
char constraintsFileName[80],structureFileName[80],
outFileName[20], jobName[20];
int itermax, lpIter, lpVerb, printEvery, restart, verbose;
int b, i, j, k, ii, lpStatus, iter, iterStart, iterStop,
tempi, timing;
double  scale, tempd, alpha, explore;

FILE *structureFile, *constraintsFile, *outFile, *errorFile;

args arg;
constraint *f, *g;
tr2d * tr;
iplp * lp;
```

```
// process command line args
if ( argc==1 ) killme("\tOptimization of 2D truss\n");
for(i=1; i<argc; i++)
  if( arg.parseArg(argv[i]) ) killme("parseArg, bad arguments\n",
-1);

// Set defaults of args that weren't found on
// the command line.
if(!arg.isDefined('i')) arg.parseArg("-i1");
if(!arg.isDefined('I')) arg.parseArg("-I10");
if(!arg.isDefined('V')) arg.parseArg("-V0");
if(!arg.isDefined('r')) arg.parseArg("-r0");
if(!arg.isDefined('v')) arg.parseArg("-v0");
if(!arg.isDefined('e')) arg.parseArg("-e1");
if(!arg.isDefined('a')) arg.parseArg("-a0.2");
if(!arg.isDefined('x')) arg.parseArg("-x1.05");
if( arg.getIntegerFlag('e') == 0) printEvery = LARGE;

itermax= arg.getIntegerFlag('i');
lpIter= arg.getIntegerFlag('I');
lpVerb = arg.getIntegerFlag('V');
restart = arg.getIntegerFlag('r');
verbose = arg.getIntegerFlag('v');
printEvery = arg.getIntegerFlag('e');
alpha = arg.getDoubleFlag('a');// step size limit
explore = arg.getDoubleFlag('x');// scale-back into feasible
arg.getCharsFlag('b', jobName);

// Optional command line args that don't have default values
timing = arg.isDefined('t');
if(verbose) arg.showAll();

// set filenames and read initial model design
strcpy(constraintsFileName, jobName);
strcat(constraintsFileName, ".osc");
strcpy(structureFileName, jobName);
if(restart) strcat(structureFileName, num2str(restart));
strcat(structureFileName, ".osi");

if(timing) showTime(" begin ");
tr = new tr2d( tr2d::getDimensions(structureFileName) );
if(verbose > 1) printf("read file: %s\n", structureFileName);
tr->readFile(structureFileName);
if(verbose > 5) {
  tr->echoStructure();
  cout << "analyze ... \n";
}
tr->analyze();
if(timing) showTime("done read and analyze");

/* show system matrix, inverse, and N_sys */
if (verbose == -3) {
  cout << "System matrix Ke\n" << tr->Ke << endl
  << "Inverse system matrix Kinv\n" << tr->Kinv << endl
  << "Transformation matrix N_sys\n" << tr->N_sys << endl;
}


// read constraints for the model
f=new constraint(constraint::countNumber(constraintsFile-
Name,"*OBJECTIVE"), 1);
g=new constraint(constraint::countNumber(constraintsFile-
Name,"*CONSTRAINTS"), 0);
if(verbose > 4) printf("read constraints\n");
if( constraint::readConstraints(constraintsFileName, f, g) )
  killme("\n ***Error::main readConstraints failed \n ", 1);
if(timing) showTime("done read constraints ");

if (verbose == -2 || verbose > 8) {
  printf("done reading constraints, %d vars and %d constraints\n",
    f->number, g->number);
  f->showValues();
  g->showValues();
}

// check problem dimensions and for bad parameters
if(verbose > 5) printf("checking problem dimensions\n");
if(explore <= 1.0) killme(" parameter 'explore' must be > 1");
if(f->number > SIZE2) killme(" f->number > SIZE1\n", f->number);
if(g->number > SIZE2) killme(" g->number over > SIZE2\n", g->num-
ber);


// allocate space and set up the lp parameters
/* g->number imposed constraints and 2*f->number step size con-
straints */
```

```
// lp = new iplp(g->number+2*f->number, f->number);
if(verbose > 5) printf("allocate LP space\n");
lp = new iplp(f->number, g->number+2*f->number);

lp->verbose = lpVerb;
lp->itermax = lpIter;


/* prepare for first iteration */
if(verbose > 5) printf("perform initial analysis\n");
tr->analyze();
scale = tr->scale(g, explore);
if(verbose > 6) printf("initial scale factor %f\n", scale);
tr->analyze();
// tr->writeResults();
if(verbose > 5) printf("analyze, scale, analyze, echo\n");
if(verbose > 5) tr->echoMemberResults();

if(verbose > 2) printf("begin iterations ... \n");


/*************************************************************
 *
 * iterate
 *
 */
iterStart = restart+1;/* default 'restart' is 0 */
iterStop  = iterStart + itermax;

printf("\n");
for (iter=iterStart; iter<iterStop; iter++)
{
if(verbose) printf("\tITERATION: %d\n", iter);
if(timing) showTime("begin iteration ", iter);

setFileName(jobName, outFileName, iter);


// assemble LP problem
/*
 * using dual formulation
 *maximize b^T y
 *s.t. A^T y + z = c
 *
 * This means the objective is in 'lp->b' and the r.h.s. is
 * in 'lp->c'. Also, the matrix 'lp->A' should be transposed.
 */

if (verbose > 2) printf("\t tr->mkF... ");
tr->mkF(lp->b, f);
lp->b = -lp->b;

if (verbose > 2) printf("tr->mkG\n");
/* Create a temporary matrix for A because using
 * the dual formulation here.
 */
matrix lp_A_temp(lp->A.col, lp->A.row);
tr->mkG(lp_A_temp, lp->c, g, f->number);
lp->A = lp_A_temp.transpose();
lp_A_temp.del();
if(verbose > 12) lp->printLP();

/* limits on variables (step size) */
if(verbose > 9) printf("add step size limits\n");
for (i=0; i < f->number; i++) {
  /* NOTE: THIS ASSUMES CONSTRAINTS ARE ORDERED */
  /* b = f->loc[i];// bar number */
  b = i+1;// bar number
  // tempd = tr->bars[b].area * tr->bars[b].E / tr-
>bars[b].length;
  tempd = tr->bars[b].area;
  lp->c[g->number + 2*i]   = alpha * tempd;
  lp->c[g->number + 2*i+1] = alpha * tempd;
  lp->A(i, g->number + 2*i)   =  1.0;
  lp->A(i, g->number + 2*i+1) = -1.0;
  if (verbose > 4) printf("limit %d: +/- %lf\n", b, alpha*tempd);
}

/* starting point: set dual vectors, y0 and z0 */
// consider using previous solution
// for (i=0; i < lp->col; i++)  lp->x[i] = 0;
lpStatus = lp->startY0();
if(lpStatus) killme("startY0 failed", lpStatus);

if(timing) showTime("done forming LP");
if(verbose == -1 || verbose > 9) lp->printLP();
```

```
/* solve LP problem */
if (verbose > 2) printf("main: solve LP problem\n");
lpStatus = lp->solveDual();
if (lpStatus) {
  printf("\n ***Warning:main LP solver failed, lpStatus: %d\n",
lpStatus);
  // exit(0);
}
if(timing) showTime("done solving LP");

/* show solution */
if (verbose > 2) {
  printf("lpStatus: %d \n LP solution (dual): \n", lpStatus);
  for (j=0; j < lp->row; j++)
    printf(" %d %12.6e \n", j, lp->y[j]);
}

// update variables
if (verbose > 3) printf(" update vars\n");
for(i=1; i <= tr->NB; i++) {
  // cout << " bar: " << i << endl;
  // tr->bars[i].area += lp->y[i-1] / tr->bars[i].E * tr-
>bars[i].length;
  if(verbose > 5) printf("%d %f %f \n", i, tr->bars[i].area, lp-
>y[i-1]);
  tr->bars[i].area += lp->y[i-1];
  if(tr->bars[i].area <= 0) killme(" bar area <= 0 ", i);
  if( !(alpha > 0) ) {
    if( tr->bars[i].area < 0 )  tr->bars[i].area = SMALL;
    if (verbose > 2)  printf("\t bar %d is now small\n", i);
  }
}
if(timing) showTime("done update variables");


// analyze, scale, and possibly write the solution of this itera-
tion
tr->analyze();
if (verbose > 18) {
  printf("\t initial unscaled step:\n");
  tr->echoMemberResults();
  tr->echoJointDisplacements();
}

if( iter == iterStart || iter == (iterStop-1) || !(iter%printEv-
ery) ) {
  tr->scale(g, 1.0);
  tr->analyze();
  if(verbose > 2 ) printf("\t end %d: write %s \n", iter, outFile-
Name);
  tr->writeFile(outFileName);
}

/* use 1.1 to ensure feasibility for next iteration */
scale = tr->scale(g, explore);
if(verbose > 6) printf("scale by %f\n", scale);
tr->analyze();

if(verbose > 4) {
  printf("\t scaled feasible soln:\n");
  // tr->echoBars();
  tr->echoJointDisplacements();
}
if(verbose > 6) tr->echoMemberResults();
if(verbose > 8) tr->echoStructure();
if(verbose > 1) printf("\t end volume: \t%15.5f \n", tr->volume()
);
if(timing) showTime("done scaling, end iteration");


}

/*
 * close iteration loop
 *
 *
 ***************************************************************
 */

if(verbose > 2) printf("\nDone iterations\n");

/* free memory used to store the problem */

if (timing) showTime("end");
```

```
/* show final */
// tempd = tr->volume();
// tr->analyze();
tr->scale(g, 1.0);
tr->analyze();
if(verbose > 2) tr->echoMemberResults();

printf("\n OK, exit\n");

return 0;
}
```

File: solve.cpp

```
// solve.cpp: analyze a (non)linear 2D frame and put the results
to stdout.
//

#include <stdlib.h>
#include <fstream.h> // contains fstream
#include <iostream.h>                          // cin, cout, <<, >>
#include <math.h>                                // sqrt()
#include <stdio.h>
#include <string.h>

#include "args.h"// handle command line args
#include "fr2d.h"
#include "utility.h"// killme()


int main(int argc, char **argv)
{
int i, j, k;
int iter = 5, ldstp = 10, verbose = 0;

args arg;
fr2d *frame;

// process command line args and read model
if (argc == 1) killme("\tanalyze 2D frame\n");
for(i=2; i<argc; i++) {
  // printf(" parse arg %d of %d '%s'\n", i, argc, argv[i]);
  if( arg.parseArg(argv[i]) ) killme("parseArg, bad arguments\n",
-1);
}
if(arg.isDefined('v')) verbose = arg.getIntegerFlag('v');
if(verbose) printf("verbose: %d\n", verbose);

if(arg.isDefined('n') || arg.isDefined('f') || arg.isDefined('F'))
{
  frame = new fr2d( fr2d::getDimensions(argv[1]), "nonlin");
  if(verbose) printf("nonlinear analysis\n");
} else {
  frame = new fr2d( fr2d::getDimensions(argv[1]) );
  if(verbose) printf("linear analysis\n");
}
```

```
frame->readFile(argv[1]);
if(verbose > 1) frame->echoStructure();

// This is different than the default level of verbose = 0 because
// the flag is defined but a value is not given (i.e., default
zero).
if(arg.isDefined('v') && arg.getIntegerFlag('v') == 0) {
  printf("%f\n", frame->volume() );
  exit(0);
}

if(frame->nonlinear) {
  iter  = (arg.isDefined('I') ? arg.getIntegerFlag('I') : 5);
  ldstp = (arg.isDefined('L') ? arg.getIntegerFlag('L') : 1);
  i = frame->analyze(ldstp, iter);
  if(i) printf(" done analyze, buckle at loadstep %d\n", i);
  else printf(" done analyze, no buckling \n");
  if(arg.isDefined('b')) exit(0);
  if(arg.isDefined('F')) {
    matrix::nroot(frame->Kg, frame->Ke, frame->frequencies, frame-
>modeShapes);
    cout << " all frequencies\n" << frame->frequencies << endl;
    cout << " all modes\n" << frame->modeShapes << endl;
  } else if(arg.isDefined('f')) {
    matrix::nroot(frame->Kg, frame->Ke, frame->frequencies, frame-
>modeShapes);
    cout << " first frequency\n" << frame->frequencies(frame->dof-
1) << endl;
    // cout << " first modeShape\n";
    // for(i=0; i<frame->dof; i++)
    //   cout << frame->modeShapes(frame->dof-1, i) << endl;
  }
} else frame->analyze();

/* show sparsity */
if(verbose > 100)  {
  printf("sparsity of Ke (%d x %d) \n", frame->dof, frame->dof);
  frame->Ke.showSparse(0.00001);
}

if(verbose > 100)  {
  printf("sparsity of N_sys (%d x %d) \n", 3*frame->NB, frame-
>dof);
  frame->N_sys.showSparse();
}

/* show system matrices */
if(verbose > 40)  {
  cout << "\nmain:: Ke\n" << frame->Ke << endl;
  if(frame->nonlinear) cout << "\nmain:: Kg\n" << frame->Kg <<
endl;
}
// << "\nmain:: Kinv\n" << frame->Kinv
// << "\nmain:: N_sys\n" << frame->N_sys << endl;

/* show final */
if(verbose > 1) frame->writeResults();
if(verbose > 0) {
  printf(" volume: %lf\n", frame->volume() );
  printf("\n end of: %s %s\n", argv[0], argv[1]);
}
if(arg.isDefined('d')) frame->echoJointDisplacements();
if(arg.isDefined('D')) frame->echoMemberDisplacements();
if(arg.isDefined('m')) frame->echoMemberForces();
if(arg.isDefined('r')) frame->writeResults();
return 0;
}
```

```
File: tr2d.cpp
// tr2d.cpp: implementation of the tr2d class.
//
//////////////////////////////////////////////////////////////

#include <fstream.h> // contains fstream
#include <iomanip.h> // formatted I/O
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include "tr2d.h"
#include "utility.h"

#define SMALL_BAR 0.000001

//////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////


tr2d::tr2d(int * dim) : Ke( 2*(dim[1]-dim[2]), 2*(dim[1]-dim[2])
),
Kinv( 2*(dim[1]-dim[2]), 2*(dim[1]-dim[2]) ),
K_sys( dim[0], dim[0] ) ,
N_sys(dim[0], 2*(dim[1]-dim[2]) )
{
/* Variable 'dim' has three elements from a "quick peek" at the
 * file. The elements, in order are NB, NN, and NS (number of bars,
 * nodes, and supports). This gives enough to allocate space for
 * the entire structural analysis.
 *
 * The text above after the colon must be there to allocate space
 * for the matrix objects.
 */

// printf("dim: %d, %d, %d\n", dim[0], dim[1], dim[2] );
NB = dim[0];
NN = dim[1];
NS = dim[2];

freeNodes = NN - NS;
dof = 2 * freeNodes;

/* allocate space for the structure */
// K_sys = new double[NB];

/* nodes and bars are one-based (i.e., first element is index 1,
not 0) */
nodes = new trussNode[NN+1];
bars  = new trussMember[NB+1];

}

tr2d::~tr2d()
{
delete nodes;

}

int tr2d::assembleSimple()
{
/* This method assembles the stiffness matrix, Ke. It is not
 * as efficient as the method, 'assemble()' but is easier
 * to code and understand. This method uses the full matrices
 * that make up the equation Ke = N' K N, where N is the
 * transformation matrix and K is a diagonal matrix with the
 * member stiffness.
 */
int i, j;

for (i=1; i<=NB; i++)
  unitVector(bars[i].NM, bars[i].NP, bars[i].length, bars[i].UV);

for (i=0; i<NB; i++) { // set up N and K matrix
  // printf(" %d %f %f\n", i, bars[i+1].area, bars[i+1].E);
  K_sys(i,i) = bars[i+1].area * bars[i+1].E / bars[i+1].length;
  if(bars[i+1].NM <= freeNodes) {
j = bars[i+1].NM;
N_sys(i, 2*(j-1))   = -bars[i+1].UV[0];
N_sys(i, 2*(j-1)+1) = -bars[i+1].UV[1];
  }
  if(bars[i+1].NP <= freeNodes) {
j = bars[i+1].NP;
N_sys(i, 2*(j-1))   = bars[i+1].UV[0];
N_sys(i, 2*(j-1)+1) = bars[i+1].UV[1];
```

```
  }
}

Ke = N_sys.transpose() * K_sys * N_sys;
Kinv = Ke.invert();
// cout << " in assemble, Ke:\n" << Ke << "Kinv\n" << Kinv << endl;

return 0;

}

int tr2d::assemble()
{
/* This method assembles the stiffness matrix, Ke faster than
 * the 'assmeble' method because it uses the 'insertElement'
 * method rather than the entire matrix multiplication.
 */

int i, j;

/* set up system matrix */
Ke.setZero();

for (i=1; i<=NB; i++) {// set up Ke
  makeUnitVector(i);
  insertElement(i);
}

for (i=0; i<NB; i++) { // set up N matrix
  j = bars[i+1].NM;
  K_sys(i,i) = bars[i+1].area * bars[i+1].E / bars[i+1].length;
  if(2*j <= dof) {
N_sys(i, 2*(j-1))   = -bars[i+1].UV[0];
N_sys(i, 2*(j-1)+1) = -bars[i+1].UV[1];
  }
  j = bars[i+1].NP;
  if(2*j <= dof) {
N_sys(i, 2*(j-1))   = bars[i+1].UV[0];
N_sys(i, 2*(j-1)+1) = bars[i+1].UV[1];
  }
}

Kinv = Ke.invert();
// cout << " in assembleFast, Ke and Kinv:\n" << Ke << Kinv <<
endl;

}

int tr2d::analyze()
{
int i, ii, j;
double FAC;

matrix x(dof, 1);
matrix b(dof, 1);

/* assemble system matrix before analysis */
// assembleSimple();
assemble();

for (i=0; i<freeNodes; i++) {// set rhs vector
  b(2*i,   0) = nodes[i+1].Px;
  b(2*i+1, 0) = nodes[i+1].Py;
}

matrix::solve(Ke, b, x);
// cout << " solution \n" << x << endl;

for (i=0; i<freeNodes; i++){/* set displacements */
  nodes[i+1].dx = x(2*i,   0);
  nodes[i+1].dy = x(2*i+1, 0);
}

/*  compute member forces and displacements from joint displace-
ments */
for(i=1; i<=NB; i++) {
  ii = bars[i].NM;
  bars[i].dl=0.0;
  FAC= -1.0;
  for(j=1; j<=2; j++) {
if(ii <= freeNodes) {
  bars[i].dl += FAC * (
nodes[ii].dx*bars[i].UV[0] +
nodes[ii].dy*bars[i].UV[1] );
}
FAC=1.0;
```

```
ii = bars[i].NP;
  }
  bars[i].force = bars[i].dl*bars[i].E*bars[i].area/
bars[i].length;
  bars[i].stress= bars[i].force/bars[i].area;
}

return 0;
}

int tr2d::makeUnitVector(int b)
{
double dx, dy, L;

// Using zero based array for unit vector!!
dx = nodes[bars[b].NP].x - nodes[bars[b].NM].x;// delta x
dy = nodes[bars[b].NP].y - nodes[bars[b].NM].y;
L = dx*dx + dy*dy;
L = sqrt(L);  // length

// System.out.println("i, dx, dy, L " + b + " " + dx + " " + dy +
" " + L);
bars[b].UV[0] = dx / L;
bars[b].UV[1] = dy / L;
bars[b].length = L;

return 0;
}

int tr2d::unitVector(int node1, int node2, double& C1, double
UVEC[2])
{
int I,K,M;

// Using zero based array for unit vector!!
C1=0.0;
UVEC[0]=nodes[node2].x - nodes[node1].x;// delta x
UVEC[1]=nodes[node2].y - nodes[node1].y;
C1=UVEC[0]*UVEC[0]+UVEC[1]*UVEC[1];
C1=sqrt(C1);  // length
UVEC[0]=UVEC[0]/C1;
UVEC[1]=UVEC[1]/C1;
return 0;
}

int tr2d::echoMemberDisplacements() {// echo results to screen

int i;

cout  << "\n\n  i \t\t  A \t\t dl \n";
for (i=1; i<=NB; i++)// member properties
{
cout  << setiosflags(ios::showpoint | ios::fixed)
<< setw(5)  << i << "\t"
<< setprecision(10) << setw(15)
<< bars[i].area << "\t"
<< setprecision(10) << setw(15)
<< bars[i].dl
<< "\n";
}

return 0;
}

int tr2d::echoMemberResults() {// echo results to screen
int i;
printf("\n\n i \t\t A \t\t dl\t\t force \t\tstress\n");
for(i=1; i<=NB; i++)
  printf("%d \t%f \t%f \t%f \t%f\n", i, bars[i].area, bars[i].dl,
    bars[i].force, bars[i].stress);
return 0;
}


int tr2d::echoJointDisplacements() {// echo results to screen

int i;

cout  << "\n\n  i \t\t  dx \t\t dy \n";
for (i=1; i<=NN; i++)// member properties
{
cout  << setiosflags(ios::showpoint | ios::fixed)
<< setw(5)  << i << "\t"
<< setprecision(10) << setw(15)
<< nodes[i].dx << "\t"
<< setprecision(10) << setw(15)
```

```
<< nodes[i].dy
<< "\n";
}

return 0;
}

int tr2d::echoStructure() {// echo structure to screen
int i;

printf(" bars: \t%d\n nodes: \t%d\n suppt: %d\n\n", NB, NN, NS);

printf("Nodes:\n N\t\tx\t\ty\t\tPx\t\tPy\n");
for (i=1; i<=NN; i++)// nodal properties
printf(" %d \t%10.5f \t%10.5f \t%10.5f \t%10.5f \n",
i, nodes[i].x, nodes[i].y, nodes[i].Px, nodes[i].Py);

printf("Members:\n N\tNM\tNP\t\tarea\t\tE\n");
for (i=1; i<=NB; i++)// member properties
printf(" %d \t%d \t%d \t%10.5f \t%10.5f \n",
i, bars[i].NM, bars[i].NP, bars[i].area, bars[i].E);

return 0;
}

int tr2d::insertElement(int b)
{
/* Insert the contribution of bar b to the global stiffness
matrix. Each
 * element contributes nkn to Ke at at least one node (unless the
member
 * has no free nodes) and may contribute at two nodes if both nodes
are
 * free. In this case, there are also 'cross terms' that 'connect'
node1
 * to node2 and the contribution here is simply -nkn. The approach
here
 * is to form the 2x2 matrix nkn for this element and then place it
in the
 * correct locations (if the bar is connected to any free nodes).
 */

int i, j, n1, n2;
double K_i;
double nkn[2][2]; // = new double[2][2];

K_i = bars[b].area * bars[b].E / bars[b].length;
nkn[0][0] = K_i * bars[b].UV[0] * bars[b].UV[0];
nkn[0][1] = K_i * bars[b].UV[1] * bars[b].UV[0];
nkn[1][1] = K_i * bars[b].UV[1] * bars[b].UV[1];
nkn[1][0] = nkn[0][1];

n1 = 2*(bars[b].NM - 1);// get first dof for (-) end of this bar
n2 = 2*(bars[b].NP - 1);// get first dof for (+) end of this bar
if(n1 < dof) {  // NM is a free node
  for(i = n1; i < n1+2; i++)
    for(j = n1; j < n1+2; j++)
Ke(i,j) += nkn[i-n1][j-n1];
  if(n2 < dof) {// both NM and NP are free nodes
    for(i = n1; i < n1+2; i++)
      for(j = n2; j < n2+2; j++) {
Ke(i,j) += -nkn[i-n1][j-n2];
Ke(j,i) += -nkn[i-n1][j-n2];
      }
  }
}

if(n2 < dof) {// NP is a free node
  for(i = n2; i < n2+2; i++)
    for(j = n2; j < n2+2; j++)
Ke(i,j) += nkn[i-n2][j-n2];
}
return 0;
}


int tr2d::mkF(matrix &c, constraint *f) {

// Construct the objective vector, 'c' in
// minimize c^T x subject to Ax <= b
//

int i;

for(i=0; i < f->number; i++) {
  // c[i]   =  bars[j].length * bars[j].length / bars[j].E;
```

```
  c[i] = bars[i+1].length;
}

return 0;
}

int tr2d::mkG(matrix & A, matrix & b, constraint *g, int n_var) {

// construct the A and b in
// minimize c^T x subject to Ax <= b
//
// Constraint variables are:
// variablecontentsexampledata type
// --------------------------------
// g.item quantity dl, dx, ...char
// g.location member / node 1, 34, ..int
// g.typetype upper / lowerint (+/- 1)
// g.val limit0.05, -120double

int i, Lii, ikn, j;
        matrix NKN(NB, NB), KN(dof, NB), phi(NB);

        /* 'phi' is a column vector representation of the sparse
matrix,
         * dK/dA * Del. Each column of this sparse matrix has one
element
         * corresponding to dK/dA_i * Del (i.e., only one element
because
         * Del is a scalar for a truss member).
         */

// Assemble -[N Kinv N~] for allowable member displacement
// Sometime later consolidate this so two matrices aren't needed.
// For now, space is allocated for each of these but the matrix is
// actually assembled only if needed (based on constraint exist-
ence).
//
if( g->isDefined('L') || g->isDefined('S') || g->isDefined('F') )
  NKN = -N_sys * Kinv * N_sys.transpose();
if( g->isDefined('X') || g->isDefined('Y') )
  KN = -Kinv * N_sys.transpose();

// cout << "\n in mkG: \n";
// cout << "Ke \n"<< Ke << endl;
// cout << "Kinv \n"<< Kinv << endl;
// cout << "N_sys \n"<< N_sys << endl;
// cout << "NKN \n"<< NKN << endl;
// NKND.showSparse();

A.setZero();
        /* construct 'phi' */
       for(i=0; i<NB; i++) phi[i] = bars[i+1].E / bars[i+1].length
* bars[i+1].dl;

/* fill in the LP matrix */
for(i=0; i< g->number; i++) {
  /* Lii is the location (e.g., bar number, node number, ...) */
  Lii = g->loc[i];
  // printf("mkG: constraint %d\n", i);
  /* member length change constraints (dl) */
  if(g->item[i] == 'L' ) {
    b[i] = g->type[i] * (g->val[i] - bars[Lii].dl);
    // printf("i %d, Lii %d,  b[i] %f \n", i, Lii, b[Lii]);
    for(j=0; j<n_var; j++) A(i,j) = g->type[i] * NKN(Lii-1, j) *
phi[j];

  /* member stress constraints (sigma) */
  } else if(g->item[i] == 'S' ) {
    // printf("tr2d::mkG constraint %d is stress\n", i);
    b[i] = g->type[i] * (g->val[i] - bars[Lii].stress);
    // printf("i %d, Lii %d,  b[i] %f \n", i, Lii, b[Lii]);
    // If S=stress and s=1/A, then S = s*F. The derivative follows
    // the chain rule:
    // dS = ds*F + s*dF
    // first form the dF as below for dF only. Then multiply by s
    // and then add ds*F terms.
    for(j=0; j<n_var; j++) // form (-K N Kinv N^T)
      A(i,j) = NKN(Lii-1, j) * (bars[Lii].area*bars[Lii].E/
bars[Lii].length);
    A(i,Lii-1) += 1.0;// add identity matrix
    for(j=0; j<n_var; j++) A(i,j) *= phi[j]; // multiply by phi

    for(j=0; j<n_var; j++) A(i,j) /= bars[Lii].area; // multiply
by s
    // add ds*F term. Add only the diagonal element because
    // the matrix is diagonal.
```

```
    A(i,Lii-1) -= bars[Lii].force / bars[Lii].area /
bars[Lii].area;
    for(j=0; j<n_var; j++) A(i,j) *= g->type[i];

  /* member size constraints (M) */
  } else if(g->item[i] == 'M' ) {
    b[i] = g->type[i] * (g->val[i] - bars[Lii].area);
    // printf("i %d, Lii %d,  b[i] %f \n", i, Lii, b[Lii]);
    for(j=0; j<n_var; j++) A(i,j) = 0.0;
    // this assumes the variables are inclusive and ordered
    A(i,Lii-1) = g->type[i] * 1.0;

  /* joint displacement constraints (dx) */
  } else if(g->item[i] == 'X') {
    ikn = 2*(Lii-1);// index in KN is not the node number
    b[i] = g->type[i] * (g->val[i] - nodes[Lii].dx);
    for(j=0; j<n_var; j++) A(i,j) = g->type[i] * KN(ikn, j) *
phi[j];

  /* joint displacement constraints (dy) */
  } else if(g->item[i] == 'Y') {
    ikn = 2*(Lii-1)+1;// index in KN is not the node number
    b[i] = g->type[i] * (g->val[i] - nodes[Lii].dy);
    for(j=0; j<n_var; j++) A(i,j) = g->type[i] * KN(ikn, j) *
phi[j];

  /* member force constraints (F) */
  } else if(g->item[i] == 'F') {
    // printf("tr2d::mkG constraint %d is force\n", i);
    b[i] = g->type[i] * (g->val[i] - bars[Lii].force);

    for(j=0; j<n_var; j++) // form (-K N Kinv N^T)
      A(i,j) = NKN(Lii-1, j) * (bars[Lii].area*bars[Lii].E/
bars[Lii].length);
    A(i,Lii-1) += 1.0;// add identity matrix
    for(j=0; j<n_var; j++) A(i,j) *= g->type[i]*phi[j]; // multi-
ply by phi

  } else killme("tr2d::mkG, unknown constraint item", i);
}

NKN.del();
KN.del();
phi.del();

return 0;
}


int tr2d::optimizeOC(tr2d conLow, tr2d conUpp)
{

double allowable = 0.00001;

for(int i=1; i<=NB; i++)
{
  if(bars[i].dl > 0)  {
bars[i].area =
fabs(bars[i].force) * bars[i].length /
(bars[i].E * conUpp.bars[i].dl);
  } else { bars[i].area =
fabs(bars[i].force) * bars[i].length /
(bars[i].E * fabs(conLow.bars[i].dl) );
  }
}
return 0;
}


int tr2d::readFile(char* fileName)
{
int i;
fstream inPut;

// fileName="file.dat";
inPut.open(fileName, ios::in);  // open the file
if(inPut.fail())  killme("readFile file not opened \n");

inPut >> NB >> NN >> NS;
for (i=1; i<=NN; i++)// nodal coordinates and loads
{
  inPut >> nodes[i].x >> nodes[i].y >> nodes[i].Px >> nodes[i].Py;
}

for (i=1; i<=NB; i++)// member properties
{
```

```
  inPut >> bars[i].NM >> bars[i].NP >> bars[i].area >> bars[i].E;
  // bars[i].E = 29000000;
}

inPut.close();  // close the file
// cout << "done reading file: " << fileName <<  "\n";

return 0;
}

int* tr2d::getDimensions(char* fileName)
{
/* This method only reads the first line of the model
 * and returns the dimensions so they can be used for
 * the constructor which allocates the space for the model
 */

int * dim = new int[3];

fstream inPut;

// fileName="file.dat";
inPut.open(fileName, ios::in);  // open the file
if(inPut.fail())  {
  cout << "***Error::readFile file: "
<< fileName << " not opened. \n";
  exit(-1);
}

inPut >> dim[0] >> dim[1] >> dim[2];
inPut.close();  // close the file

return dim;
}


int tr2d::rmSmallBars(double min) {

int i, j, remove;

remove = 0;
for(i=1; i<=NB; i++)
{
  if (bars[i].area < min) {
    for(j=i; j<=NB; j++) {
bars[j].E    = bars[j+1].E;
bars[j].NM   = bars[j+1].NM;
bars[j].NP   = bars[j+1].NP;
bars[j].area = bars[j+1].area;
    }
    NB--;
    remove++;
    // should check if resulting structure is still stable
  }
}

return remove;
}


double tr2d::scale(constraint *g, double factor) {

int i, j, b, v;
double scale, temp;
int *forceViolated;

forceViolated = new int[NB+1];

/* Linear scaling will change magnitude but not sign:
 * perform scaling only if delta and delta_allowable
 * have the same sign. There may be several types of constraints
 * so must check all types and scale according to the maximum
 * violation.
 */

// First check for member force constraints which can't be handled
// by linear scaling. If a force constraint is violated, try to
shift
// the forces away from that member by decreasing its area.
j = 0;
do {
  v = 0;
  // if(j) printf("tr2d::scale: rescan constraint force viola-
tions\n");
  for(i=1; i <= NB; i++) forceViolated[i] = 0;
  for(i=0; i < g->number; i++) {
```

```
    b = g->loc[i];
    if( (g->item[i] == 'F') && (sign(g->val[i]) ==
sign(bars[b].force))
      && (fabs(g->val[i]) < fabs(bars[b].force)) ) {
// Force in this bar has the same sign and exceeds the allowable
// try to decrease the area of this bar to shift the forces to
// elsewhere in the structure.
bars[b].area *= 0.8;// when 3 iter, 0.8^3 = 0.512
forceViolated[b] = 1;
v = 1;
printf("\ttr2d::scale decrease bar %d, iter %d of 5 for
force\n",b,j+1);
break;
    }
  }
  for(i=1; i <= NB; i++) if(!forceViolated[i]) bars[i].area *=
1.5;
  analyze();
} while(j++ < 5 && v);

if(v) {
  printf("tr2d::scale force constraint violated, final
attempt...\n");
  for(i=0; i < g->number; i++) {
    b = g->loc[i];
    if( (g->item[i] == 'F') && (sign(g->val[i]) ==
sign(bars[b].force))
      && (fabs(g->val[i]) < fabs(bars[b].force)) ) {
printf("tr2d::scale bar %d is now small\n", b);
bars[b].area = SMALL_BAR;
    }
  }
  // killme("tr2d::scale force constraint violated");
}

scale = 0.0001;
/* checking all constraints */
for(i=0; i < g->number; i++) {
  b = g->loc[i];
  /* member length change */
  if(g->item[i] =='L') {
    if (sign(g->val[i]) == sign(bars[b].dl) ) {
      temp = bars[b].dl / g->val[i];
    }
  /* member stress */
  } else if(g->item[i] =='S') {
    if (sign(g->val[i]) == sign(bars[b].stress) ) {
      temp = bars[b].stress / g->val[i];
    }
  /* member size */
  } else if(g->item[i] =='M') {
    // This should be handled carefully because there could be
    // conflicts here. Minimum is easy but maximum may cause an
    // infeasible problem. Ignore upper limits for now.
    if(g->type[i] == -1) temp = g->val[i] / bars[b].area;
  /* joint displacement (dx) */
  } else if(g->item[i] =='X') {
    if (sign(g->val[i]) == sign(nodes[b].dx) ) {
      temp = nodes[b].dx / g->val[i];
    }
  /* joint displacement (dy) */
  } else if(g->item[i] == 'Y') {
    if (sign(g->val[i]) == sign(nodes[b].dy) ) {
      temp = nodes[b].dy / g->val[i];
      // printf("\t\t ratio: %15.10f scale: %15.10f\n", temp,
scale);
    }
  /* member force (F) this is meaningless if it is the only con-
straint */
  } else if(g->item[i] == 'F') {
    // do nothing here, it is handled above
  } else killme("tr2d::scale, unknown constraint item", i+1);
  if(temp > scale) scale = temp;
}

// printf("\t scale: \t%10.5f x fact = %10.5f\n", scale, fac-
tor*scale);
for(i=1; i <= NB; i++) bars[i].area *= scale*factor;
// analyze();


return scale*factor;
}

double tr2d::volume()
{
```

```
double vol = 0.0;

for(int i=1; i<=NB; i++)
  vol += bars[i].area * bars[i].length;

return vol;
}

int tr2d::writeFile(char* fileName)
{
int i;
FILE *outfile;

// fileName="file.out";

if( (outfile = fopen(fileName, "w")) == NULL ) {
  printf(" ***Error::writeFile: '%s' not opened\n", fileName);
  return 1;
}

fprintf(outfile, "%5d%5d%5d\n", NB, NN, NS);

/* nodal coordinates and loads */
for (i=1; i<=NN; i++)
fprintf(outfile, "%14.5f %14.5f %14.5f %14.5f\n",
nodes[i].x, nodes[i].y, nodes[i].Px, nodes[i].Py);

/* member properties */
for (i=1; i<=NB; i++)
fprintf(outfile, "%5d%5d %14.5e %19.5e\n",
bars[i].NM, bars[i].NP, bars[i].area, bars[i].E );

// printf("done writing file: %s\n", fileName);
fclose(outfile);
return 0;
}


int tr2d::writeResults(char* fileName)
{
int i;
fstream outPut;
// fileName="file.rst";
outPut.open(fileName, ios::out);  // open the file

outPut  << "bars: \t"  << setw(5) << NB << "\n"
<< "nodes: \t" << setw(5) << NN << "\n"
<< "suppt: \t" << setw(5) << NS << "\n";

outPut  << setprecision(3)
<< setw(15)
<< setiosflags(ios::showpoint | ios::fixed);

outPut  << "\n\n coordinates \t\t loads \t\t displacements\n";
for (i=1; i<=NN; i++)// nodal coordinates and loads
{
outPut  << setw(15) << nodes[i].x
<< setw(15) << nodes[i].y
<< setw(15) << nodes[i].Px
<< setw(15) << nodes[i].Py
<< setw(15) << nodes[i].dx
<< setw(15) << nodes[i].dy << "\n";
}

outPut  << "\n\n NM \t NP \t A \t L \t force \t stress \t dl \n";
for (i=1; i<=NB; i++)// member properties
{
outPut  << setprecision(3)
<< setw(5)  << bars[i].NM
<< setw(5)  << bars[i].NP
<< setw(10) << bars[i].area
<< setprecision(2)
<< setw(6)  << bars[i].length
<< setw(9)  << bars[i].force
<< setw(9)  << bars[i].stress
<< setprecision(5)
<< setw(9)  << bars[i].dl << "\n";
}

outPut.close();  // close the file
// cout << "done writing file: " << fileName <<"\n";

return 0;
}
```

```
int tr2d::writeResults(void)
{
int i;

cout  << "bars: \t"  << setw(5) << NB << "\n"
<< "nodes: \t" << setw(5) << NN << "\n"
<< "suppt: \t" << setw(5) << NS << "\n";

cout  << setprecision(3)
<< setw(15)
<< setiosflags(ios::showpoint | ios::fixed);

cout  << "\n\n\t Coordinates \t\t Loads \t\t\t\t Displacements\n";
for (i=1; i<=NN; i++)// nodal coordinates and loads
{
cout  << setprecision(3) << setw(15)
<< nodes[i].x << "\t" << nodes[i].y << "\t"
<< nodes[i].Px << "\t" << nodes[i].Py << "\t"
<< setprecision(9)
<< setw(15)
<< nodes[i].dx << "\t" << nodes[i].dy << "\n";
}

cout  << " \n\n i \t NM \t NP \t A \t L \t force \t stress \t dl
\n";
for (i=1; i<=NB; i++)// member properties
{
cout  << setprecision(3)
<< setw(5)
<< i << "\t" << bars[i].NM << "\t" << bars[i].NP
<< setw(10) << bars[i].area
<< setprecision(2)
<< setw(10)  << bars[i].length
<< setw(10)  << bars[i].force
<< setw(10)  << bars[i].stress
<< setprecision(8)
<< setw(15)  << bars[i].dl << "\n";
}

return 0;
}
```

```
File: utility.cpp
// utility.cpp: general helper functions
//
//
//////////////////////////////////////////////////////////


#include <ctype.h> // isalnum(int ch)
#include <fstream.h> // contains fstream
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <time.h>
#include "utility.h"

int freeNodes(char* fileName) {
/* Function to determine the number of free nodes in
 * the file 'fname' by reading the first line and then
 * closing the file.
 */

int num, nb, nn, ns;
FILE *fp;

num = 0;
if( (fp = fopen(fileName, "r")) == NULL ) {
  printf(" ***Error::freeNodes: '%s' not opened\n", fileName);
  return 1;
}

fscanf(fp, "%d %d %d", &nb, &nn, &ns);
fclose(fp);
num = nn - ns;

return num;
}

int setFileNames(char* jobName, char* structureFileName, char*
constraintsFileName)
{

char t_char[40];
```

```
strcpy(t_char, jobName);

strcat(t_char, ".osi");
strcpy(structureFileName, t_char);

strcpy(t_char, jobName);
strcat(t_char, ".osc");
strcpy(constraintsFileName, t_char);
return 0;
}

int setFileName(char* jobName, char* outputName, int iterNum) {

char t_char[40], temp1[4];

strcpy(t_char, jobName);
strcat(t_char, num2str(iterNum) );
strcat(t_char, ".osi");
strcpy(outputName, t_char);
// printf(" iter %d outputName is '%s' \n", iterNum, outputName);
return 0;
}

char * num2str(int num) {
/* Convert the number 'num' to a four character string '.xxx'
 * and return a pointer to it.
 */
char temp1[5], * t_str;

strcpy(temp1, "xxxx");
if((num>=0) && (num<10)) {
  temp1[0] = '.';
  temp1[1] = '0';
  temp1[2] = '0';
  temp1[3] = 48 + num;
} else if(num<100) {
  temp1[0] = '.';
  temp1[1] = '0';
  temp1[2] = 48 + num/10;
  temp1[3] = 48 + num%10;
} else if(num<1000) {
  temp1[0] = '.';
  temp1[1] = 48 + num/100;
  temp1[2] = 48 + (num - 100 * (num/100)) / 10;
  temp1[3] = 48 + num%10;
}
t_str = temp1;
return t_str;
}

void killme(char * msg, int code) {

if(code == 0) {
  printf("%s \n", msg);
} else {
  printf("***Error::%s \n", msg);
  printf("\texit code: %d \n", code);
}
exit(code);
}

void killme(char * msg, char * problem, int code) {

if(code == 0) {
  printf("%s \n", msg);
} else {
  printf("***Error::%s \n\tproblem: %s\n", msg, problem);
  printf("\texit code: %d \n", code);
}
exit(code);
}

void showTime(char * msg, int num) {
time_t t;
struct tm *systime;

t = time (NULL);
systime = localtime(&t);
if(num)
  printf("\n showTime: %s %d \t%.2d:%.2d:%.2d\n", msg, num,
    systime->tm_hour, systime->tm_min, systime->tm_sec );
else
  printf("\n showTime: %s \t%.2d:%.2d:%.2d\n", msg,
    systime->tm_hour, systime->tm_min, systime->tm_sec );
}
```

APPENDIX B
JAVA PROGRAMS

This appendix contains a complete listing of the source code for the programs written in Java used in this work.

```
File: DataPlotFrame.java
import java.awt.*;
import java.applet.Applet;
import java.lang.String;
import java.lang.Integer;
import java.text.*; // DecimalFormat
import java.awt.event.*;

// DataPlotFrame.java
//
// This class is to define a general use frame for plotting lines
and text. The
// basic elements of the class are point coordinates. The entire
object is scaled
// to fit into the size of the frame.

class DataPlotFrame extends Frame implements KeyListener,
MouseListener,
MouseMotionListener, WindowListener, ActionListener {

/* Each line is defined by two points, 'X[i]' and 'Y[i]'. There
 * are a total of 'nPoints' and thus (nPoints -1) lines. Lines are
drawn
 * in order.
 */

// members of the 'DataPlotFrame' class
public boolean showPointNumber, showLineNumber;
public int    nPoints;
public double X[], Y[];
public String labels[];
public Color lineColor, backColor, pointNumberColor, lineNumber-
Color;

private Button dismissButton;
private double XSHIFT, YSHIFT, SCALE_X, SCALE_Y, Ymax, Xmax, Ymin,
Xmin;
private double AMAXX, AMAYY;// usable plotting area

private int frameWidth, frameHeight, BUF;
private int lastMouseX, lastMouseY;

/* basic constructor sets default values */
public DataPlotFrame(int width, int height) {
int i, j;
int dim = 100;
X = new double [dim];
Y = new double [dim];
labels = new String[dim];

showLineNumber  = false;
showPointNumber = true;
backColor = Color.black;
lineColor = Color.black;
lineNumberColor = Color.red;
pointNumberColor   = Color.blue;

nPoints = 0;
BUF = 40;
frameWidth = width;
frameHeight = height;

AMAXX = frameWidth - 2*BUF;// usable plotting area
AMAYY = frameHeight - 2*BUF;

dismissButton = new Button("Dismiss");
dismissButton.addActionListener(this);

setLayout(new BorderLayout() );
add("South", dismissButton);

setTitle("Data Plot");

addKeyListener(this);
addWindowListener(this);
addMouseListener(this);
addMouseMotionListener(this);

setSize(frameWidth, frameHeight);
}

public void paint(Graphics g) {
int i;
double x1, y1, x2, y2;

g.setColor(lineColor);
```

```
setRange();

// Plot the lines
for(i = 1; i < nPoints; i++) {
// System.out.println("plot line " + i);
x1 = (X[i-1] / SCALE_X) - XSHIFT;
y1 = frameHeight - ((Y[i-1] / SCALE_Y) - YSHIFT);
x2 = (X[i] / SCALE_X) - XSHIFT;
y2 = frameHeight - ((Y[i] / SCALE_Y) - YSHIFT);
// System.out.println(" "+x1+" "+y1+" "+x2+" "+y2);
g.drawLine( (int)x1, (int)y1, (int)x2, (int)y2);
}

// Show data points
g.setColor(pointNumberColor);
for(i=0; i < nPoints; i++) {
x1 = (X[i] / SCALE_X) - XSHIFT;
y1 = frameHeight - ((Y[i] / SCALE_Y) - YSHIFT);
g.drawOval((int)x1-5, (int)y1-5, 10, 10);
}
g.setColor(lineColor);

// Show coordinates
if(showPointNumber) {
  DecimalFormat fmt;
  if(Ymax < 0.001) fmt = new DecimalFormat("0.0000000");
  else if(Ymax < 0.1) fmt = new DecimalFormat("0.0000");
  else if(Ymax < 100) fmt = new DecimalFormat("0.00");
  else fmt = new DecimalFormat("0");

  g.setColor(pointNumberColor);
  i=nPoints-1;
x1 = (X[i] / SCALE_X) - XSHIFT;
y1 = frameHeight - ((Y[i] / SCALE_Y) - YSHIFT);
g.drawString( "vol: " +fmt.format(Y[i]), (int)x1, (int)y1);
  g.setColor(lineColor);
}
}

public void setRange() {
// This sets the scale factors for the object.
double x, y, Xrange, Yrange;
double LARGE = 1000000;
int i;

Xmax = 0;
Xmin = LARGE;
Ymax = 0;
Ymin = LARGE;

for(i=0; i < nPoints; i++) {
  x = X[i];
  y = Y[i];
  if(x > Xmax) Xmax = x;
  if(x < Xmin) Xmin = x;
  if(y > Ymax) Ymax = y;
  if(y < Ymin) Ymin = y;
}

Xrange = Xmax - Xmin;
Yrange = Ymax - Ymin;

// Scale so everything fits. X and Y need not be proportional
SCALE_X = Xrange/AMAXX;
SCALE_Y = Yrange/AMAYY;

// shift in pixels to align centroid of 2D object
XSHIFT = (Xmax+Xmin)/2/SCALE_X - frameWidth/2;
YSHIFT = (Ymax+Ymin)/2/SCALE_Y - frameHeight/2;
}

// required for KeyListener
public void keyPressed(KeyEvent e) { }
public void keyReleased(KeyEvent e) { }
public void keyTyped(KeyEvent e) {
// System.out.println(" Key typed" + e.getKeyChar());
if(e.getKeyChar() == KeyEvent.VK_ENTER) {
  // System.out.println(" 'Enter' typed");
  showPointNumber = !showPointNumber;
  repaint();
} else if(e.getKeyChar() == KeyEvent.VK_ESCAPE) {
  setVisible(false);
  dispose();
}
}
```

```
public void windowClosed(WindowEvent event) { }
public void windowDeiconified(WindowEvent event) { }
public void windowIconified(WindowEvent event) { }
public void windowActivated(WindowEvent event) { }
public void windowDeactivated(WindowEvent event) { }
public void windowOpened(WindowEvent event) { }
public void windowClosing(WindowEvent event) {
setVisible(false);
dispose();
}

public void actionPerformed(ActionEvent e) {
if (e.getSource() == dismissButton) {
setVisible(false);
dispose();
}
}

/* required for MouseListener */
public void mouseClicked( MouseEvent e)  {  }
public void mouseReleased( MouseEvent e) {  }
public void mouseEntered( MouseEvent e)  {  }
public void mouseExited( MouseEvent e)   { }
public void mouseMoved( MouseEvent e)    {  }

public void mousePressed( MouseEvent e) {
lastMouseX = e.getX();
lastMouseY = e.getY();
}

public void mouseDragged( MouseEvent e) {
int dx, dy;
double angle;

dx = e.getX() - lastMouseX;
dy = e.getY() - lastMouseY;
lastMouseX = e.getX();
lastMouseY = e.getY();

repaint();
}
}
```

```
File: Fr2d.java
import java.awt.*;
import java.util.*;// StringTokenizer
import java.lang.String;
import java.lang.Integer;
import java.text.*;// format numbers

//////////////////////////////////////////////////////
//
// begin the 'Fr2d' class

public class Fr2d {

//////////////////////////////////////////////////////
//
// member variables for this class
//
//
int NB, NS, NN, dof, freeNodes;
double Ke[][], Kinv[][];
FrameNode nodes[];
FrameMemberbars[];

/* basic constructor */
public Fr2d(int nb, int nn, int ns) {
int i;

NB = nb;
NN = nn;
NS = ns;
freeNodes = NN - NS;
dof = 3 * freeNodes;

Ke = new double [dof][dof];
Kinv = new double [dof][dof];

/* nodes and bars are one-based (i.e., first element is index 1,
not 0) */
nodes = new FrameNode[nn+1];
bars = new FrameMember[nb+1];
for(i=1; i <= nn; i++) nodes[i] = new FrameNode();
for(i=1; i <= nb; i++) bars[i]  = new FrameMember();
}
```

```
public Fr2d(String s) {
int i;
StringTokenizer t = new StringTokenizer(s);

NB = Integer.parseInt( t.nextToken() );
NN = Integer.parseInt( t.nextToken() );
NS = Integer.parseInt( t.nextToken() );
freeNodes = NN - NS;
dof = 3 * freeNodes;

Ke = new double [dof][dof];
Kinv = new double [dof][dof];

/* nodes and bars are one-based (i.e., first element is index 1,
not 0) */
nodes = new FrameNode[NN+1];
bars = new FrameMember[NB+1];
for(i=1; i <= NN; i++) {
nodes[i] = new FrameNode();
nodes[i].x = Double.valueOf( t.nextToken() ).doubleValue();
nodes[i].y = Double.valueOf( t.nextToken() ).doubleValue();
nodes[i].Px = Double.valueOf( t.nextToken() ).doubleValue();
nodes[i].Py = Double.valueOf( t.nextToken() ).doubleValue();
nodes[i].Mz = Double.valueOf( t.nextToken() ).doubleValue();
}

for(i=1; i <= NB; i++) {
bars[i]  = new FrameMember();
bars[i].NM = Integer.parseInt( t.nextToken() );
bars[i].NP = Integer.parseInt( t.nextToken() );
bars[i].area = Double.valueOf( t.nextToken() ).doubleValue();
bars[i].section_I = Double.valueOf( t.nextToken() ).doubleValue();
bars[i].E = Double.valueOf( t.nextToken() ).doubleValue();
}
}


/************************************
 *
 * class methods -- methods that return int
 *
 */
```

```
public int analyze() {

/* Basic linear elastic analysis */
int i, j, k, ii;
double SK[][] = new double[3][3];
// matrix P(dof, 1), delta(dof, 1), Delta(3*NB, 1);

double x[] = new double[dof];
double b[] = new double[dof];

/* set up system matrix */
assembleKe();

/* set up load vector */
for (i=1; i<=freeNodes; i++) {
  b[3*i-3] = nodes[i].Px;
  b[3*i-2] = nodes[i].Py;
  b[3*i-1] = nodes[i].Mz;
  // System.out.println(" "+i+" "+b[3*i-3]+" "+b[3*i-2]+" "+b[3*i-
1]);
}

// solve the system
MyMath.solve(Ke, b, x);

/* set displacements from solution */
// System.out.println("Fr2d::analyze displacements");
for (i=1; i<=freeNodes; i++) {
  nodes[i].dx = x[3*i-3];
  nodes[i].dy = x[3*i-2];
  nodes[i].rz = x[3*i-1];
  // System.out.println(" "+i+" "+x[3*i-3]+" "+x[3*i-2]+" "+x[3*i-
1]);
}

/* compute member forces and displacements */
mkMemberDisplacements();

return 0;
}
```

```
public int assembleKe() {
/*
 * This method assembles the stiffness matrix, Ke by creating
 * the sub-matrix for each element and then inserting it.
 *
 * local variables (matrices):
 * ke_i contribution to Ke of a Math.single member
 * kg_i contribution to Kg of a Math.single member
 * NRP transformation matrix for + node of a member
 * rotated to global coordinate system
 * NRM transformation matrix for - node of a member
 * rotated to global coordinate system
 * SK primitive member stiffness matrix for a Math.single bar
 * dK derivative of SK for a Math.single bar
 */


int i, ii, j, row, nm, np;

double ke_i[][] = new double[3][3];
double temp1[][] = new double[3][3];
double NRP[][] = new double[3][3];
double NRM[][] = new double[3][3];
double SK[][] = new double[3][3];

for (i=0; i<dof; i++)  // zero Ke
  for (j=0; j<dof; j++)
  Ke[i][j] = 0.0;

for(i=1; i<=NB; i++) {/* begin assembly of Ke*/
  // System.out.println(" bar " + i);
  /* create matrices for this element */
  mkProps(i);
  mkRN(NRP, NRM, i);
  mkKei(SK, i);

  np = bars[i].NP;
  nm = bars[i].NM;

  if(bars[i].NP <= freeNodes) {
    // System.out.println(" element %d effects node %d\n", i,
bars[i].NP);
    /* transform from local to global coordinate system */
    prod3T(NRP, SK, NRP, ke_i);
    insertElement(ke_i, np, np);

    if(bars[i].NM <= freeNodes) {
/* these are the cross terms (both nodes are free). It
 * does not affect the transformation matrix, N_sys
 */
    // System.out.println(" el %d, node %d %d \n",i, bars[i].NM,
bars[i].NP);
prod3T(NRP, SK, NRM, ke_i);
insertElement(ke_i, np, nm);
for(ii=0; ii<3; ii++) for(j=0; j<3; j++) temp1[ii][j] =
ke_i[j][ii];
insertElement(temp1, nm, np);
    }
  }

  if(bars[i].NM <= freeNodes) {
    // System.out.println(" element %d effects node %d\n", i,
bars[i].NM);
    prod3T(NRM, SK, NRM, ke_i);
    insertElement(ke_i, nm, nm);
  }

}/* get next bar for Ke*/

return 0;
}

public String toString() {
String s;
String tab = "\t";
String nl = "\n";
DecimalFormat fmt = new DecimalFormat("0.0000");

s = NB + tab + NN + tab + NS + nl;
for(int i=1; i<=NN; i++)
  s += nodes[i].x + tab + nodes[i].y + tab
+ nodes[i].Px + tab + nodes[i].Py + tab + nodes[i].Mz + nl;
for(int i=1; i<=NB; i++)
  s += bars[i].NM + tab + bars[i].NP + tab
+ fmt.format(bars[i].area) + tab
+ fmt.format(bars[i].section_I) + tab + bars[i].E + nl;
```

```
return s;
}

public String JointDisplacements() {
String s;
String tab = "\t";
String nl = "\n";
int i, j;
double d[]   = new double[3];
double max[] = new double[3];
double min[] = new double[3];
DecimalFormat fx_6 = new DecimalFormat("0.000000");

s = "Joint \t dx \t\t dy \t\t rz\n";
for(i=1; i<=NN; i++) {
  s += " " + i + tab + fx_6.format(nodes[i].dx) + tab
    + fx_6.format(nodes[i].dy) + tab
    + fx_6.format(nodes[i].rz) + nl;
  d[0] = nodes[i].dx;
  d[1] = nodes[i].dy;
  d[2] = nodes[i].rz;
  for(j=0; j<3; j++) {
    max[j] = (d[j] > max[j] ? d[j] : max[j]);
    min[j] = (d[j] < min[j] ? d[j] : min[j]);
  }
}
s += "\nMax:\t" + fx_6.format(max[0]) + tab + fx_6.format(max[1])
+ tab + fx_6.format(max[2]);
s += "\nMin:\t" + fx_6.format(min[0]) + tab + fx_6.format(min[1])
+ tab + fx_6.format(min[2]) + nl;

return s;
}

public String MemberDisplacements() {
String s;
String tab = "\t";
String nl = "\n";
int i, j;
double d[]   = new double[3];
double max[] = new double[3];
double min[] = new double[3];
DecimalFormat fx_6 = new DecimalFormat("0.000000");
DecimalFormat fx_3 = new DecimalFormat("0.000");

s = "Bar\t dl \t rp \t rn \n";
for(i=1; i<=NB; i++) {
  s += " " + i + tab + fx_6.format(bars[i].dl) + tab
    + fx_6.format(bars[i].rp) + tab
    + fx_6.format(bars[i].rn) + nl;
  d[0] = bars[i].dl;
  d[1] = bars[i].rp;
  d[2] = bars[i].rn;
  for(j=0; j<3; j++) {
    max[j] = (d[j] > max[j] ? d[j] : max[j]);
    min[j] = (d[j] < min[j] ? d[j] : min[j]);
  }
}
s += "\nMax:\t" + fx_6.format(max[0]) + tab + fx_6.format(max[1])
+ tab + fx_6.format(max[2]);
s += "\nMin:\t" + fx_6.format(min[0]) + tab + fx_6.format(min[1])
+ tab + fx_6.format(min[2]) + nl;

return s;
}

public String MemberForces() {
String s;
String tab = "\t";
String nl = "\n";
int i, j;
double d[]   = new double[3];
double max[] = new double[3];
double min[] = new double[3];
DecimalFormat fx_6 = new DecimalFormat("0.000000");
DecimalFormat fx_3 = new DecimalFormat("0.000");

s = "Bar\t Fx \t Mp \t Mn \n";
for(i=1; i<=NB; i++) {
  s += " " + i + tab + fx_3.format(bars[i].Fx) + tab
    + fx_3.format(bars[i].Mp) + tab
    + fx_3.format(bars[i].Mn) + nl;
  d[0] = bars[i].Fx;
  d[1] = bars[i].Mp;
  d[2] = bars[i].Mn;
```

```
  for(j=0; j<3; j++) {
    max[j] = (d[j] > max[j] ? d[j] : max[j]);
    min[j] = (d[j] < min[j] ? d[j] : min[j]);
  }
}
s += "\nMax:\t" + fx_3.format(max[0]) + tab + fx_3.format(max[1])
+ tab + fx_3.format(max[2]);
s += "\nMin:\t" + fx_3.format(min[0]) + tab + fx_3.format(min[1])
+ tab + fx_3.format(min[2]) + nl;

return s;
}

public int initialize() {
/*
 * This should probably be near the constructor but it is
 * here for now. It calls routines that are normally done
 * only once for the structure; i.e., not at each iteration
 * and probably called from the main program.
 *
 */
int i;
for (i=1; i<=NB; i++) mkProps(i);// set basic member props.
return 0;
}

public int insertElement(double m[][], int row, int col) {
/*
 * Insert a 3x3 matrix contribution to a larger matrix (e.g., Ke,
Kg,
 * or N_sys). Variables 'row' and 'col' are the one-based indices
of
 * the location to insert this 3x3 matrix, 'm' into the largeer
matrix,
 * 'MAT'. Thus the indices are one-based but the matrices 'm' and
'MAT' are
 * zero-based. Also, the indices treat the 3x3 matrix as a
Math.single element
 * thus (row,col) indices (2,2) will put the elements from 'm'
into 'MAT'
 * beginning at MAT(3,3)=m[0][0].
 */
int i, j, r, c;

r = 3*(row-1);
c = 3*(col-1);

for(i=r; i<r+3; i++)
  for(j=c; j<c+3; j++)
    Ke[i][j] += m[i-r][j-c];

// System.out.println("insert (%d, %d) value %f\n", i, j, m[i-
r][j-c]);

return 0;
}

public int mkKei(double SK[][], int bar) {
// Construct the primitive stiffness matrix for bar 'b'
int i, j;
double dist, cof;

for(i=0; i<3; i++)
  for(j=0; j<3; j++)
    SK[i][j] = 0.0;

dist = 4*bars[bar].E * bars[bar].section_I / bars[bar].length;
cof  = dist / 2.0;

SK[0][0] = bars[bar].E * bars[bar].area / bars[bar].length;
SK[1][1] = dist;
SK[2][2] = SK[1][1];
SK[1][2] = cof;
SK[2][1] = SK[1][2];

return 0;
}

public int mkMemberDisplacements() {
/*
 * Compute the member displacements from the already computed
 * joint displacements
 */
```

```
int b, i, ii, j, row, nm, np;

double NRP[][] = new double[3][3];
double NRM[][] = new double[3][3];
double SK[][] = new double[3][3];
double d[] = new double[3];
double D[] = new double[3];
double F[] = new double[3];

for(b=1; b<=NB; b++) {
  // System.out.println("Fr2d::mkDisplacements bar " + b);
  /* create matrices for this element */
  mkRN(NRP, NRM, b);
  np = bars[b].NP;
  nm = bars[b].NM;

  for(i=0; i<3; i++) D[i] = 0.0;
  if(bars[b].NP <= freeNodes) {
    d[0] = nodes[np].dx;
    d[1] = nodes[np].dy;
    d[2] = nodes[np].rz;
    for(i=0; i<3; i++)
      for(j=0; j<3; j++)
        D[i] += NRP[i][j] * d[j];
  }
  if(bars[b].NM <= freeNodes) {
    d[0] = nodes[nm].dx;
    d[1] = nodes[nm].dy;
    d[2] = nodes[nm].rz;
    for(i=0; i<3; i++)
      for(j=0; j<3; j++)
        D[i] += NRM[i][j] * d[j];
  }
  // F = K * Delta
  mkKei(SK, b);
  for(i=0; i<3; i++) {
    F[i] = 0.0;
    for(j=0; j<3; j++) {
      F[i] += SK[i][j] * D[j];
    }
  }
  bars[b].dl = D[0];
  bars[b].rp = D[1];
  bars[b].rn = D[2];
  bars[b].Fx = F[0];
  bars[b].Mp = F[1];
  bars[b].Mn = F[2];

}/* get next bar */

return 0;
}

public int mkProps(int bar) {
/*
 * Compute the geometric properties of element number 'bar'.
 * Properties computed are
 * angle of the bar's alignment (in radians!!)
 * the bar's length
 * constants c0 and c1 that relate bar area and
 * its section modulus according to
 * I = c0 * A ^ c1
 */

int n1, n2;
double dx, dy;

// System.out.println("mkProps bar "+bar);
n1 = bars[bar].NM;
n2 = bars[bar].NP;

bars[bar].length = 0.0;
dx = nodes[n2].x - nodes[n1].x;// delta x
dy = nodes[n2].y - nodes[n1].y;
bars[bar].length = dx*dx + dy*dy;
bars[bar].length = Math.sqrt(bars[bar].length);  // length

if(bars[bar].length == 0) {
  // System.out.println(" bar %d, n1: %d, n2: %d \n", bar, n1,
n2);
  // killme("fr2d::mkProps, zero length bar", bar);
}

bars[bar].theta = Math.atan2(dy, dx);
```

```
/* Right now, uMath.sing c1 = 2 (quadratic) for area <=> section_I
*/
bars[bar].A2I[1] = 2.0;
bars[bar].A2I[0] = bars[bar].section_I/Math.pow(bars[bar].area,
bars[bar].A2I[1]);

return 0;
}

public int mkRN(double NRP[][], double NRM[][], int bar) {
/*
 * Create the rotated transformation matrices NRP and NRM for
 * the 'bar' member and the rotation matrix 'R'.
 *
 */

double R[][] =  new double[3][3];
double SNP[][] = new double[3][3];
double SNM[][] = new double[3][3];
int i, j, k;

for(i=0; i<3; i++) {
  for(j=0; j<3; j++) {
SNP[i][j] = 0.0;
SNM[i][j] = 0.0;
  }
}

mkRotationMatrix(R, bar);

/* transformation matrix, N, for (+) node */
SNP[0][0] = 1.0;
SNP[1][2] = 1.0;
SNP[1][1] =-1.0 / bars[bar].length;
SNP[2][1] =SNP[1][1];

/* transformation matrix, N, for (-) node */
SNM[0][0] = -1.0;
SNM[2][2] = 1.0;
SNM[1][1] =1.0 / bars[bar].length;
SNM[2][1] =SNM[1][1];

/* transform from local coord. sys. to global coord. sys. */
for(i=0; i<3 ; i++) {
  for(j=0; j<3 ; j++) {
  NRP[i][j]=0.0;
  NRM[i][j]=0.0;
    for(k=0; k<3 ; k++) {
NRP[i][j] += SNP[i][k]*R[k][j];
NRM[i][j] += SNM[i][k]*R[k][j];
    }
  }
}

return 0;
}

public int mkRotationMatrix(double R[][], int bar) {
/*
 * Create the rotation matrix 'R' for member
 * number 'bar'.
 *
 */

int i, j;

for(i=0; i<3; i++) for(j=0; j<3; j++) R[i][j] = 0.0;

/* rotation matrix */
R[0][0] =  Math.cos( bars[bar].theta );
R[1][1] =  R[0][0];
R[0][1] =  Math.sin( bars[bar].theta );
R[1][0] = -R[0][1];
R[2][2] =  1.0;

return 0;
}

public int prod3T(double A[][], double B[][], double C[][], double
D[][]) {
// multiply A^T x B x C and return in D
int i, ii, j, k, L;

for(i=0; i<3 ; i++) { // form product
  for(j=0; j<3 ; j++) {
  D[i][j]=0.0;
```

```
    for(k=0; k<3 ; k++)
for(L=0; L<3; L++)
D[i][j] = D[i][j] + A[k][i]*B[k][L]*C[L][j];
  }
}

return 0;
}

public int writeResults() {
/* Write the nodal and member displacements */
echoJointDisplacements();
echoMemberDisplacements();
echoMemberForces();
return 0;
}

public int zeroJointDisplacements() {
int i;

for (i=1; i<=NN; i++) {
  nodes[i].dx = 0.0;
  nodes[i].dy = 0.0;
  nodes[i].rz = 0.0;
}
return 0;
}

public int zeroMemberDisplacements() {
int i;

for (i=1; i<=NB; i++) {
  bars[i].dl = 0.0;
  bars[i].rp = 0.0;
  bars[i].rn = 0.0;
}
return 0;
}

public int zeroMemberForces() {
int i;

for (i=1; i<=NB; i++) {
  bars[i].Fx = 0.0;
  bars[i].Mp = 0.0;
  bars[i].Mn = 0.0;
}
return 0;
}


/************************************
 *
 * class methods -- methods that return double and double *
 *
 */

public double a2i(int bar) {
/* Given the bar number, 'bar', return the section modulus, I
 * according to the relation
 * I = c0 * A^c1
 * where c0 and c1 are variables in the array 'A2I' of the
 * frameMember structure. These constants should be set with
 * a call to 'mkProps'.
 */

double I;

I = bars[bar].A2I[0] * Math.pow(bars[bar].area, bars[bar].A2I[1]);
// // System.out.println("A^2 %f\n", Math.pow(bars[bar].area,
bars[bar].A2I[1]) );

return I;
}

public double volume() {

double vol = 0.0;

for(int i=1; i<=NB; i++)
  vol += bars[i].area * bars[i].length;
return vol;
}
```

```
}
//
// end of 'Fr2d' class definition
//
////////////////////////////////////////////////////////////////


////////////////////////////////////////////////////////////////
//
//The Fr2d class uses the 'FrameNode' class
//

class FrameNode {
int    n;                        /* node number */
 double x, y;                     /* coordinate */
double dx, dy, rz;                  /* translation */
double Px, Py, Mz;                  /* loading */

        public FrameNode() {
        }
}

//
// end of 'FrameNode' class definition
//
////////////////////////////////////////////////////////////////



////////////////////////////////////////////////////////////////
//
//The Fr2d class uses the 'FrameMember' class
//

class FrameMember {
        int NP;                 /* positive (+) node */
        int NM;                 /* negative (-) node */
        double area;            /* bar area */
        double section_I;       /* section modulus */
        double length;          /* bar length */
        double theta;           /* bar orientation */
        double Fx;              /* axial bar force */
        double Mp;              /* moment at (+) end */
        double Mn;              /* moment at (-) end */
        double stress;          /* bar stress */
        double dl;              /* change in length */
        double rp;              /* rotation at (+) end */
        double rn;              /* rotation at (-) end */
        double UV[];            /* unit vector direction */
        double A2I[];            /* unit vector direction */
        double E;               /* Young's modulus */


        public FrameMember() {
        NP = 0;
        NM = 0;
                UV = new double[2];
A2I = new double [2];

        }
}
//
// end of 'FrameMember' class definition
//
////////////////////////////////////////////////////////////////
```

```
File: Fr3d.java
import java.awt.*;
import java.util.*;// StringTokenizer
// import java.applet.Applet;
import java.lang.String;
import java.lang.Integer;
import java.text.*;// format numbers

//////////////////////////////////////////////////////
//
//
// begin the 'Fr3d' class


public class Fr3d {

//////////////////////////////////////////////////////
//
// member variables for this class
//
//
int NB, NS, NN, dof, freeNodes;
double Ke[][], Kinv[][];
FrameNode nodes[];
FrameMemberbars[];


/* basic constructor */
public Fr3d(int nb, int nn, int df) {
int i;

NB = nb;
NN = nn;
dof = df;

Ke = new double [dof][dof];
Kinv = new double [dof][dof];

/* nodes and bars are one-based (i.e., first element is index 1,
not 0) */
nodes = new FrameNode[nn+1];
bars = new FrameMember[nb+1];
for(i=1; i <= nn; i++) nodes[i] = new FrameNode();
for(i=1; i <= nb; i++) bars[i]  = new FrameMember();
}

public Fr3d(String s) {
int i, j;
StringTokenizer t = new StringTokenizer(s);

// System.out.println("begin Fr3d(String s) constructor");

NB = Integer.parseInt( t.nextToken() );
NN = Integer.parseInt( t.nextToken() );
dof = Integer.parseInt( t.nextToken() );

Ke = new double [dof][dof];
Kinv = new double [dof][dof];

/* nodes and bars are one-based (i.e., first element is index 1,
not 0) */
nodes = new FrameNode[NN+1];
bars = new FrameMember[NB+1];
for(i=1; i <= NN; i++) {
nodes[i] = new FrameNode();
nodes[i].x = Double.valueOf( t.nextToken() ).doubleValue();
nodes[i].y = Double.valueOf( t.nextToken() ).doubleValue();
nodes[i].z = Double.valueOf( t.nextToken() ).doubleValue();
nodes[i].Px = Double.valueOf( t.nextToken() ).doubleValue();
nodes[i].Py = Double.valueOf( t.nextToken() ).doubleValue();
nodes[i].Pz = Double.valueOf( t.nextToken() ).doubleValue();
nodes[i].Mx = Double.valueOf( t.nextToken() ).doubleValue();
nodes[i].My = Double.valueOf( t.nextToken() ).doubleValue();
nodes[i].Mz = Double.valueOf( t.nextToken() ).doubleValue();
}

for(i=1; i <= NB; i++) {
bars[i]  = new FrameMember();
bars[i].NM = Integer.parseInt( t.nextToken() );
bars[i].NP = Integer.parseInt( t.nextToken() );
bars[i].area = Double.valueOf( t.nextToken() ).doubleValue();
bars[i].Ix = Double.valueOf( t.nextToken() ).doubleValue();
bars[i].Iy = Double.valueOf( t.nextToken() ).doubleValue();
bars[i].Iz = Double.valueOf( t.nextToken() ).doubleValue();
bars[i].E = Double.valueOf( t.nextToken() ).doubleValue();
bars[i].G = Double.valueOf( t.nextToken() ).doubleValue();
```

```
bars[i].thx = Double.valueOf( t.nextToken() ).doubleValue() *
Math.PI / 180.0;
bars[i].thy = Double.valueOf( t.nextToken() ).doubleValue() *
Math.PI / 180.0;
bars[i].thz = Double.valueOf( t.nextToken() ).doubleValue() *
Math.PI / 180.0;
}

for(i=1; i <= NN; i++) {
  for(j=0; j<6; j++)
nodes[i].fix[j] = Integer.parseInt( t.nextToken() );
}
}


/************************************
 *
 * class methods -- methods that return int
 *
 */

public int analyze() {

/* Basic linear elastic analysis */
int i, j, k, ii;
double SK[][] = new double[6][6];
// matrix P(dof, 1), delta(dof, 1), Delta(6*NB, 1);

double delta[] = new double[dof];
double P[] = new double[dof];

/* set up system matrix */
assembleKe();
if(1 == 1) {
  for(i=1; i<=NB; i++) mkProps(i);
  return 0;
}

/* set up load vector */
 i = 1;
 k = 0;
 while (k < dof) {
   if(nodes[i].fix[0] == 0) P[k++] = nodes[i].Px;
   if(nodes[i].fix[1] == 0) P[k++] = nodes[i].Py;
   if(nodes[i].fix[2] == 0) P[k++] = nodes[i].Pz;
   if(nodes[i].fix[3] == 0) P[k++] = nodes[i].Mx;
   if(nodes[i].fix[4] == 0) P[k++] = nodes[i].My;
   if(nodes[i].fix[5] == 0) P[k++] = nodes[i].Mz;
   i++;
}

// solve the system
MyMath.solve(Ke, P, delta);

/* set displacements from solution */
// System.out.println("Fr3d::analyze displacements");
 i = 1;
 k = 0;
 while (k < dof) {
   if(nodes[i].fix[0] == 0) nodes[i].dx = delta[k++];
   if(nodes[i].fix[1] == 0) nodes[i].dy = delta[k++];
   if(nodes[i].fix[2] == 0) nodes[i].dz = delta[k++];
   if(nodes[i].fix[3] == 0) nodes[i].rx = delta[k++];
   if(nodes[i].fix[4] == 0) nodes[i].ry = delta[k++];
   if(nodes[i].fix[5] == 0) nodes[i].rz = delta[k++];
   i++;
}

/* compute member forces and displacements */
mkMemberDisplacements();

return 0;
}

public int assembleKe() {
/*
 * This method assembles the stiffness matrix, Ke by creating
 * the sub-matrix for each element and then inserting it.
 *
 * local variables (matrices):
 * ke_i contribution to Ke of a Math.single member
 * kg_i contribution to Kg of a Math.single member
 * NRP transformation matrix for + node of a member
 * rotated to global coordinate system
 * NRM transformation matrix for - node of a member
 * rotated to global coordinate system
```

```
 * SK primitive member stiffness matrix for a Math.single bar
 * dK derivative of SK for a Math.single bar
 */

int i, ii, j, row, nm, np;

double ke_i[][] = new double[6][6];
double temp1[][] = new double[6][6];
double NRP[][] = new double[6][6];
double NRM[][] = new double[6][6];
double SK[][] = new double[6][6];

for (i=0; i<dof; i++)  // zero Ke
  for (j=0; j<dof; j++)
  Ke[i][j] = 0.0;

for(i=1; i<=NB; i++) {/* begin assembly of Ke*/
  // System.out.println(" bar " + i);
  /* create matrices for this element */
  mkProps(i);
  mkRN(NRP, NRM, i);
  mkKei(SK, i);

  np = bars[i].NP;
  nm = bars[i].NM;

   if( numberDofAtNode(bars[i].NP) > 0) {
    // System.out.println(" element %d effects node %d\n", i,
bars[i].NP);
    /* transform from local to global coordinate system */
    prod3T(NRP, SK, NRP, ke_i);
    insertElement(ke_i, np, np);
     insertToNsys (NRP,  i, bars[i].NP);

     if( numberDofAtNode(bars[i].NM) > 0) {
/* these are the cross terms (both nodes are free). It
 * does not affect the transformation matrix, N_sys
 */
    // System.out.println(" el %d, node %d %d \n",i, bars[i].NM,
bars[i].NP);
prod3T(NRP, SK, NRM, ke_i);
insertElement(ke_i, np, nm);
for(ii=0; ii<6; ii++) for(j=0; j<6; j++) temp1[ii][j] =
ke_i[j][ii];
insertElement(temp1, nm, np);
    }
  }

   if( numberDofAtNode(bars[i].NM) > 0) {
    // System.out.println(" element %d effects node %d\n", i,
bars[i].NM);
    prod3T(NRM, SK, NRM, ke_i);
    insertElement(ke_i, nm, nm);
     insertToNsys (NRM, i, bars[i].NM);
  }

}/* get next bar for Ke*/

return 0;
}


public int getDofNumber(int node, int index) {
// Any node can have 6 dof but may have less than that. If there
are
// less, it is necessary to get the index of terms in the rotated
element
// stiffness matrix so only the terms corresponding to free nodes
will be
// inserted in the global matrices (e.g., Ke). For example, the
2,2 term
// is the 4EIy/L flexural stiffness) but this may be the first
free dof at
// this node. Thus all elements that tie into this node should
begin
// with terms at the 2,2 index.
//
// This method returns the index 'i' in the rotated 6x6 element
stiffness
// matrix of the 'index' free dof. For the example here, given
index=0
// then i=2 is returned. In this example, the array 'fix' might
have the
// form {1 1 0 1 1 0} where here it is significant that the first
two
```

```
// terms are not zero.

int i, ind;
ind = -1;

for(i=0; i<6; i++) {
  if(nodes[node].fix[i] == 0) {
    // printf("free dof %d\n", i);
    ind++;
  }
  if(ind == index) {
    // printf(" at i=%d, found %d == %d\n", i, ind, index);
    return i;
  }
}
// if(ind == -1) killme("fr3d::getDofNumber failed");
return 0;
}

public int firstDofIndex(int n) {
// Return the index (zero-based) of the first free dof of
// node 'n'.
int i, j, sum;

sum = 0;
for(i=1; i<n; i++)
  for(j=0; j<6; j++)
if(nodes[i].fix[j] == 0) sum++;

// printf("fr3d::firstDofIndex node %d, first dof: %d\n", n, sum);
return sum;
}

public String toString() {
String s;
String tab = "\t";
String nl = "\n";
DecimalFormat fmt = new DecimalFormat("0.0000");

s = NB + tab + NN + tab + dof + nl;
for(int i=1; i<=NN; i++)
  s += nodes[i].x + tab + nodes[i].y + tab + nodes[i].z + tab
+ nodes[i].Px + tab + nodes[i].Py + tab + nodes[i].Pz
+ nodes[i].Mx + tab + nodes[i].My + tab + nodes[i].Mz + nl;
for(int i=1; i<=NB; i++)
  s += bars[i].NM + tab + bars[i].NP + tab
+ fmt.format(bars[i].area) + tab
+ fmt.format(bars[i].Ix) + tab + fmt.format(bars[i].Iy)
+ tab + fmt.format(bars[i].Iz) + tab + bars[i].E + tab + bars[i].G
+ tab + (bars[i].thx * 180.0 / Math.PI)
+ tab + (bars[i].thy * 180.0 / Math.PI)
+ tab + (bars[i].thz * 180.0 / Math.PI) + nl;

return s;
}

public String JointDisplacements() {
String s;
String tab = "\t";
String nl = "\n";
int i, j;
double d[]   = new double[6];
double max[] = new double[6];
double min[] = new double[6];
DecimalFormat fx_6 = new DecimalFormat("0.000000");

s = "Joint \t dx \t\t dy \t\t dz \t\t rx \t\t ry \t\t rz\n";
for(i=1; i<=NN; i++) {
  s += " " + i + tab + fx_6.format(nodes[i].dx) + tab
    + fx_6.format(nodes[i].dy) + tab + fx_6.format(nodes[i].dz) +
tab
    + fx_6.format(nodes[i].rx) + tab + fx_6.format(nodes[i].ry) +
tab
    + fx_6.format(nodes[i].rz) + nl;
  d[0] = nodes[i].dx;
  d[1] = nodes[i].dy;
  d[2] = nodes[i].dz;
  d[3] = nodes[i].rx;
  d[4] = nodes[i].ry;
  d[5] = nodes[i].rz;
  for(j=0; j<6; j++) {
    max[j] = (d[j] > max[j] ? d[j] : max[j]);
    min[j] = (d[j] < min[j] ? d[j] : min[j]);
  }
}
s += "\nMax:\t" + fx_6.format(max[0]) + tab + fx_6.format(max[1])
```

```
+ tab + fx_6.format(max[2]) + tab + fx_6.format(max[3])
+ tab + fx_6.format(max[4]) + tab + fx_6.format(max[5]);
s += "\nMin:\t" + fx_6.format(min[0]) + tab + fx_6.format(min[1])
+ tab + fx_6.format(min[2]) + tab + fx_6.format(min[3])
+ tab + fx_6.format(min[4]) + tab + fx_6.format(min[5]) + nl;

return s;
}

public String MemberDisplacements() {
String s;
String tab = "\t";
String nl = "\n";
int i, j;
double d[]   = new double[6];
double max[] = new double[6];
double min[] = new double[6];
DecimalFormat fx_6 = new DecimalFormat("0.000000");
DecimalFormat fx_3 = new DecimalFormat("0.000");

s = "Bar\t dl \t twist \t rpy \t rpz \t rny \t rnz \n";
for(i=1; i<=NB; i++) {
  s += " " + i + tab +  fx_6.format(bars[i].dl) + tab
    + fx_6.format(bars[i].twist) + tab
    + fx_6.format(bars[i].rpy) + tab + fx_6.format(bars[i].rpz) +
tab
    + fx_6.format(bars[i].rny) + tab + fx_6.format(bars[i].rnz) +
nl;
  d[0] = bars[i].dl;
  d[1] = bars[i].twist;
  d[2] = bars[i].rpy;
  d[3] = bars[i].rpz;
  d[4] = bars[i].rny;
  d[5] = bars[i].rnz;
  for(j=0; j<6; j++) {
    max[j] = (d[j] > max[j] ? d[j] : max[j]);
    min[j] = (d[j] < min[j] ? d[j] : min[j]);
  }
}
s += "\nMax:\t" + fx_6.format(max[0]) + tab + fx_6.format(max[1])
+ tab + fx_6.format(max[2]) + tab + fx_6.format(max[3])
+ tab + fx_6.format(max[4]) + tab + fx_6.format(max[5]);
s += "\nMin:\t" + fx_6.format(min[0]) + tab + fx_6.format(min[1])
+ tab + fx_6.format(min[2]) + tab + fx_6.format(min[3])
+ tab + fx_6.format(min[4]) + tab + fx_6.format(min[5]) + nl;

return s;
}

public String MemberForces() {
String s;
String tab = "\t";
String nl = "\n";
int i, j;
double d[]   = new double[6];
double max[] = new double[6];
double min[] = new double[6];
DecimalFormat fx_6 = new DecimalFormat("0.000000");
DecimalFormat fx_3 = new DecimalFormat("0.000");

s = "Bar\t Fx \t Torque \t Mpy \t Mpz \t Mny \t Mpz \n";
for(i=1; i<=NB; i++) {
  s += " " + i + tab + fx_3.format(bars[i].Fx) + tab
    + fx_3.format(bars[i].Torque) + tab
    + fx_3.format(bars[i].Mpy) + tab + fx_3.format(bars[i].Mpz) +
tab
    + fx_3.format(bars[i].Mny) + tab + fx_3.format(bars[i].Mnz) +
nl;
  d[0] = bars[i].Fx;
  d[1] = bars[i].Torque;
  d[2] = bars[i].Mpy;
  d[2] = bars[i].Mpz;
  d[2] = bars[i].Mny;
  d[2] = bars[i].Mnz;
  for(j=0; j<6; j++) {
    max[j] = (d[j] > max[j] ? d[j] : max[j]);
    min[j] = (d[j] < min[j] ? d[j] : min[j]);
  }
}
s += "\nMax:\t" + fx_3.format(max[0]) + tab + fx_3.format(max[1])
+ tab + fx_3.format(max[2]) + tab + fx_3.format(max[3])
+ tab + fx_3.format(max[4]) + tab + fx_3.format(max[5]);
s += "\nMin:\t" + fx_3.format(min[0]) + tab + fx_3.format(min[1])
+ tab + fx_3.format(min[2]) + tab + fx_3.format(min[3])
+ tab + fx_3.format(min[4]) + tab + fx_3.format(min[5]) + nl;
```

```
return s;
}

public int initialize() {
/*
 * This should probably be near the constructor but it is
 * here for now. It calls routines that are normally done
 * only once for the structure; i.e., not at each iteration
 * and probably called from the main program.
 *
 */

int i, sum;

sum = 0;
for (i=1; i<=NB; i++) mkProps(i);// set basic member props.
 for(i=1; i<=NN; i++) sum += numberDofAtNode(i);

 if(sum != dof) {
   SimpleMessage m = new SimpleMessage("Fr3d::initialize, dof .ne.
sum", "Input Error");
}
return 0;
}


public int insertElement(double m[][], int n1, int n2) {
/*
 * Insert a 3x3 matrix contribution to a larger matrix (e.g., Ke,
Kg,
 * or N_sys). Variables 'row' and 'col' are the one-based indices
of
 * the location to insert this 3x3 matrix, 'm' into the largeer
matrix,
 * 'MAT'. Thus the indices are one-based but the matrices 'm' and
'MAT' are
 * zero-based. Also, the indices treat the 3x3 matrix as a
Math.single element
 * thus (row,col) indices (2,2) will put the elements from 'm'
into 'MAT'
 * beginning at MAT(3,3)=m[0][0].
 */

int i, j, K, L, r, c;

 r = firstDofIndex(n1);
 c = firstDofIndex(n2);
for(i=r; i<r+numberDofAtNode(n1); i++) {
  for(j=c; j<c+numberDofAtNode(n2); j++) {
    K = getDofNumber(n1, i-r);
    L = getDofNumber(n2, j-c);
    if( (nodes[n1].fix[K] == 0) && (nodes[n2].fix[L] == 0) ) {
     // printf("fr3d:insertElement Ke(%d,%d) from m(%d,%d)\n", i,
j, K, L);
      Ke[i][j] += m[K][L];
    }
  }
}

// System.out.println("insert (%d, %d) value %f\n", i, j, m[i-
r][j-c]);

return 0;
}

public int insertToNsys(double m[][], int n1, int n2) {
/*
 * Insert a 6x6 matrix contribution to a larger matrix (e.g., Ke,
Kg,
 * or N_sys). Variables 'n1' and 'n2' are the one-based indices of
 * the location to insert this 6x6 matrix, 'm' into the largeer
matrix,
 * 'MAT'. Thus the indices are one-based but the matrices 'm' and
'MAT' are
 * zero-based. Also, the indices treat the 6x6 matrix as a single
element
 * thus (n1,n2) indices (2,2) will put the elements from 'm' into
'MAT'
 * beginning at MAT(6,6)=m[0][0].
 */

int i, j, K, L, r, c;

r = 6*(n1-1);
c = firstDofIndex(n2);
```

```
for(i=r; i<r+6; i++) {
  for(j=c; j<c+numberDofAtNode(n2); j++) {
    // printf("fr3d::insertToNsys i %d j %d\n", i, j);
    K = i-r;
    L = getDofNumber(n2, j-c);
    if( nodes[n2].fix[L] == 0 ) {
      // MAT(i,j) += m[K][L];
    }
  }
}

// printf("fr3d::insertToNsys (%d, %d) value %f\n", i, j, m[i-
r][j-c]);

return 0;
}

public int numberDofAtNode(int n) {
// Return the number of degrees of freedom at node 'n' based
// on the array 'fix'.
int i, sum;

sum = 0;
for(i=0; i<6; i++) if(nodes[n].fix[i] == 0) sum++;

return sum;
}

public int mkKei(double SK[][], int bar) {
// Construct the primitive stiffness matrix for bar 'b'
int i, j;
double disty, distz, cofy, cofz;

for(i=0; i<6; i++)
  for(j=0; j<6; j++)
    SK[i][j] = 0.0;

disty = 4*bars[bar].E * bars[bar].Iy / bars[bar].length;
distz = 4*bars[bar].E * bars[bar].Iz / bars[bar].length;
cofy  = disty / 2.0;
cofz  = distz / 2.0;

SK[0][0] = bars[bar].E * bars[bar].area / bars[bar].length;
SK[1][1] = bars[bar].G * bars[bar].Ix / bars[bar].length;
SK[2][2] = disty;
SK[4][4] = SK[2][2];
SK[2][4] = cofy;
SK[4][2] = cofy;
SK[3][3] = distz;
SK[5][5] = SK[3][3];
SK[3][5] = cofz;
SK[5][3] = cofz;

return 0;
}

public int mkMemberDisplacements() {
/*
 * Compute the member displacements from the already computed
 * joint displacements
 */

int b, i, ii, j, row, nm, np;

double NRP[][] = new double[6][6];
double NRM[][] = new double[6][6];
double SK[][] = new double[6][6];
double d[] = new double[6];
double D[] = new double[6];
double F[] = new double[6];

for(b=1; b<=NB; b++) {
  // System.out.println("Fr3d::mkDisplacements bar " + b);
  /* create matrices for this element */
  mkRN(NRP, NRM, b);
  np = bars[b].NP;
  nm = bars[b].NM;

  for(i=0; i<6; i++) D[i] = 0.0;
  if(bars[b].NP <= freeNodes) {
    d[0] = nodes[np].dx;
    d[1] = nodes[np].dy;
    d[2] = nodes[np].dz;
    d[3] = nodes[np].rx;
    d[4] = nodes[np].ry;
    d[5] = nodes[np].rz;
```

```
    for(i=0; i<6; i++)
      for(j=0; j<6; j++)
        D[i] += NRP[i][j] * d[j];
  }
  if(bars[b].NM <= freeNodes) {
    d[0] = nodes[nm].dx;
    d[1] = nodes[nm].dy;
    d[2] = nodes[nm].dz;
    d[3] = nodes[nm].rx;
    d[4] = nodes[nm].ry;
    d[5] = nodes[nm].rz;
    for(i=0; i<6; i++)
      for(j=0; j<6; j++)
        D[i] += NRM[i][j] * d[j];
  }

  // F = K * Delta
  mkKei(SK, b);
  for(i=0; i<6; i++) {
    F[i] = 0.0;
    for(j=0; j<6; j++) {
      F[i] += SK[i][j] * D[j];
    }
  }
  bars[b].dl = D[0];
  bars[b].twist = D[1];
  bars[b].rpy = D[2];
  bars[b].rpz = D[3];
  bars[b].rny = D[4];
  bars[b].rnz = D[5];

  bars[b].Fx = F[0];
  bars[b].Torque = F[1];
  bars[b].Mpy = F[2];
  bars[b].Mpz = F[3];
  bars[b].Mny = F[4];
  bars[b].Mnz = F[5];

}/* get next bar */

return 0;
}
```

```
public int mkProps(int bar) {
/*
 * Compute the geometric properties of element number 'bar'.
 * Properties computed are
 * angle of the bar's alignment (in radians!!)
 * the bar's length
 * constants c0 and c1 that relate bar area and
 * its section modulus according to I = c0 * A ^ c1
 */

int n1, n2;
double dx, dy, dz;

n1 = bars[bar].NM;
n2 = bars[bar].NP;
bars[bar].length = 0.0;
dx = nodes[n2].x - nodes[n1].x;// delta x
dy = nodes[n2].y - nodes[n1].y;
dz = nodes[n2].z - nodes[n1].z;
bars[bar].length = dx*dx + dy*dy + dz*dz;
bars[bar].length = Math.sqrt(bars[bar].length);  // length

if(bars[bar].length == 0) {
  // System.out.println(" bar %d, n1: %d, n2: %d \n", bar, n1,
n2);
  // killme("fr2d::mkProps, zero length bar", bar);
}

// bars[bar].theta = Math.atan2(dy, dx);

/* Right now, uMath.sing c1 = 2 (quadratic) for area <=> Ix */
bars[bar].A2Ix[1] = 2.0;
bars[bar].A2Ix[0] = bars[bar].Ix/Math.pow(bars[bar].area,
bars[bar].A2Ix[1]);

// System.out.println(" bar %d, c0 %f, c1 %f\n", bar,
bars[bar].A2Ix[0], bars[bar].A2Ix[1]);
// System.out.println(" bar %d, dx %f, dy %f, theta %f\n", bar,
dx, dy, bars[bar].theta);

return 0;
}
```

```
public int mkRN(double NRP[][], double NRM[][], int bar) {
/*
 * Create the rotated transformation matrices NRP and NRM for
 * the 'bar' member and the rotation matrix 'R'.
 *
 */

double R[][] =  new double[6][6];
double SNP[][] = new double[6][6];
double SNM[][] = new double[6][6];
double linv;
int i, j, k;

for(i=0; i<6; i++) {
  for(j=0; j<6; j++) {
SNP[i][j] = 0.0;
SNM[i][j] = 0.0;
  }
}

mkRotationMatrix(R, bar);
linv = 1.0 / bars[bar].length;

/* transformation matrix, N, for (+) node */
SNP[0][0] = 1.0;
SNP[1][3] = 1.0;
SNP[2][4] = 1.0;
SNP[3][5] = 1.0;
SNP[2][2] = linv;
SNP[3][1] =-linv;
SNP[4][2] = linv;
SNP[5][1] =-linv;

/* transformation matrix, N, for (-) node */
SNM[0][0] = -1.0;
SNM[1][3] = -1.0;
SNM[4][4] =  1.0;
SNM[5][5] =  1.0;
SNM[2][2] =-linv;
SNM[3][1] = linv;
SNM[4][2] =-linv;
SNM[5][1] = linv;

/* transform from local coord. sys. to global coord. sys. */
for(i=0; i<6 ; i++) {
  for(j=0; j<6 ; j++) {
  NRP[i][j]=0.0;
  NRM[i][j]=0.0;
    for(k=0; k<6 ; k++) {
NRP[i][j] += SNP[i][k]*R[k][j];
NRM[i][j] += SNM[i][k]*R[k][j];
    }
  }
}

return 0;
}


public int mkRotationMatrix(double R[][], int bar) {
/*
 * Create the rotation matrix 'R' for member
 * number 'bar'.
 *
 */

int i, j, k;
double r[][] = new double[3][3];
double s[][] = new double[3][3];
double t[][] = new double[3][3];

for(i=0; i<6; i++) for(j=0; j<6; j++) R[i][j] = 0.0;
for(i=0; i<3; i++) for(j=0; j<3; j++) t[i][j] = 0.0;

/* about x-axis */
t[0][0] =  1.0;
t[1][1] =  Math.cos( bars[bar].thx );
t[2][2] =  t[1][1];
t[1][2] =  Math.sin( bars[bar].thx );
t[2][1] = -t[1][2];
for(i=0; i<3; i++) for(j=0; j<3; j++) r[i][j] = t[i][j];

/* about y-axis */
for(i=0; i<3; i++) for(j=0; j<3; j++) t[i][j] = 0.0;
t[0][0] =  Math.cos( bars[bar].thy );
t[1][1] =  1.0;
```

```
t[2][2] =  t[0][0];
t[0][2] = -Math.sin( bars[bar].thy );
t[2][0] = -t[0][2];
for(i=0; i<3; i++)
  for(j=0; j<3; j++) {
    s[i][j] = 0.0;
    for(k=0; k<3; k++) s[i][j] += t[i][k] * r[k][j];
  }
for(i=0; i<3; i++) for(j=0; j<3; j++) r[i][j] = s[i][j];

/* about z-axis */
for(i=0; i<3; i++) for(j=0; j<3; j++) t[i][j] = 0.0;
t[0][0] =  Math.cos( bars[bar].thz );
t[1][1] =  t[0][0];
t[2][2] =  1.0;
t[0][1] =  Math.sin( bars[bar].thz );
t[1][0] = -t[0][1];
for(i=0; i<3; i++)
  for(j=0; j<3; j++) {
    s[i][j] = 0.0;
    for(k=0; k<3; k++) s[i][j] += t[i][k] * r[k][j];
  }
for(i=0; i<3; i++) for(j=0; j<3; j++) r[i][j] = s[i][j];

for(i=0; i<3; i++) {
  for(j=0; j<3; j++) {
    R[i][j] = r[i][j];
    R[i+3][j+3] = r[i][j];
  }
}

return 0;
}

public int prod3T(double A[][], double B[][], double C[][], double
D[][]) {

// multiply A^T x B x C and return in D

int i, j, k;
double temp[][] = new double[6][6];

for(i=0; i<6 ; i++) { // form product
  for(j=0; j<6 ; j++) {
    temp[i][j]=0.0;
    for(k=0; k<6 ; k++) temp[i][j] += A[k][i]*B[k][j];
  }
}

for(i=0; i<6 ; i++) { // form product
  for(j=0; j<6 ; j++) {
    D[i][j]=0.0;
    for(k=0; k<6 ; k++) D[i][j] += temp[i][k]*C[k][j];
  }
}
return 0;
}

public int writeResults() {
/*
 * Write the nodal and member displacements
 *
 */
echoJointDisplacements();
echoMemberDisplacements();
echoMemberForces();
return 0;
}

/************************************
 *
 * class methods -- methods that return double and double *
 *
 */

public double a2i(int bar) {
/* Given the bar number, 'bar', return the section modulus, I
 * according to the relation
 * I = c0 * A^c1
 * where c0 and c1 are variables in the array 'A2I' of the
 * frameMember structure. These constants should be set with
 * a call to 'mkProps'.
 */

double I;
```

```
I = bars[bar].A2Ix[0] * Math.pow(bars[bar].area,
bars[bar].A2Ix[1]);
bars[bar].A2Ix[1]) );

return I;
}

public double volume() {
double vol = 0.0;
for(int i=1; i<=NB; i++)
  vol += bars[i].area * bars[i].length;
return vol;
}

}
//
// end of 'Fr3d' class definition
//
////////////////////////////////////////////////////////////////


////////////////////////////////////////////////////////////////
//
// The Fr3d class uses the 'FrameNode' class
//

class FrameNode {
int    n;                    /* node number */
        int fix[];/* flag for fixity */
 double x, y, z;                  /* coordinate */
double dx, dy, dz;                /* translation */
double rx, ry, rz;                /* translation */
double Px, Py, Pz;                /* loading */
double Mx, My, Mz;                /* loading */

        public FrameNode() {
fix  = new int[6];
        }
}

//
// end of 'FrameNode' class definition
//
```

```
////////////////////////////////////////////////////////////////


////////////////////////////////////////////////////////////////
//
//The Fr3d class uses the 'FrameMember' class
//

class FrameMember {
        int NP;                 /* positive (+) node */
        int NM;                 /* negative (-) node */
        double area;            /* bar area */
        double Ix,Iy,Iz;        /* section modulus */
        double length;          /* bar length */
        double thx,thy,thz;     /* bar orientation */
        double Fx;              /* axial bar force */
        double Torque;              /* torque */
        double Mpy,Mpz;              /* moment at (+) end */
        double Mny,Mnz;              /* moment at (-) end */
        double stress;          /* bar stress */
        double dl;              /* change in length */
        double twist;           /* twist */
        double rpy,rpz;         /* rotation at (+) end */
        double rny,rnz;         /* rotation at (-) end */
        double A2Ix[],A2Iy[],A2Iz[];          /* relate a to I */
        double E;               /* Young's modulus */
        double G;               /* shear modulus */


        public FrameMember() {
        NP = 0;
        NM = 0;
A2Ix = new double[2];
A2Iy = new double[2];
A2Iz = new double[2];

        }
}
//
// end of 'FrameMember' class definition
//
////////////////////////////////////////////////////////////////
```

```
File: HelpFrame.java
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;

//////////////////////////////////////////////////////////
//
// begin class HelpFrame
//

public class HelpFrame extends Frame implements WindowListener,
ActionListener {

private TextAreahelpArea;
private Button dismissButton;

// constructor
public HelpFrame(int visibleRows, int visibleColumns) {

helpArea = new TextArea(visibleRows, visibleColumns);
dismissButton = new Button("Dismiss");
dismissButton.addActionListener(this);

// helpArea.setText(helpMsg);

GridBagConstraints constraints = new GridBagConstraints();
GridBagLayout gridbag = new GridBagLayout();
setLayout(gridbag);

constraints.gridx  = 0;
constraints.gridy  = 0;
constraints.gridwidth  = 1;
constraints.gridheight = 1;
constraints.weightx  = 1.0;
constraints.weighty  = 1.0;
constraints.fill       = GridBagConstraints.HORIZONTAL;
constraints.anchor     = GridBagConstraints.NORTH;
gridbag.setConstraints(helpArea, constraints);
add(helpArea);

constraints.gridy++;
constraints.weightx  = 1.0;
constraints.weighty  = 0.0;
constraints.anchor     = GridBagConstraints.SOUTH;
gridbag.setConstraints(dismissButton, constraints);
add(dismissButton);

// setLayout(new BorderLayout() );
setTitle("Help");

addWindowListener(this);
}

public void setMessage(String s) {
helpArea.setText(s);
}

public void windowClosed(WindowEvent event) { }
public void windowDeiconified(WindowEvent event) { }
public void windowIconified(WindowEvent event) { }
public void windowActivated(WindowEvent event) { }
public void windowDeactivated(WindowEvent event) { }
public void windowOpened(WindowEvent event) { }
public void windowClosing(WindowEvent event) {
setVisible(false);
dispose();
}

public void actionPerformed(ActionEvent e) {
if (e.getSource() == dismissButton) {
setVisible(false);
dispose();
}
}

}


//
//end class HelpFrame
//
//////////////////////////////////////////////////////////
```

```
File: L3PCanvas.java
import java.awt.*;
import java.applet.Applet;
import java.lang.String;
import java.lang.Integer;
import java.awt.event.*;

// L3PCanvas.java
//
// This class is to define a general use canvas for plotting lines
and text. The
// basic elements of the class are point coordinates and member
(i.e., line)
// connectivity. Members of this class include background color,
text color, and
// line color. The entire object is scaled to fit into the size of
the canvas.


class L3PCanvas extends Canvas implements KeyListener, MouseLis-
tener, MouseMotionListener {
/* Each line is defined by two points, 'Point1[i]' and
'Point2[i]'. There
 * are a total of 'nPoints', each of which have coordinates 'X',
'Y',
 * and 'Z' defined in the object's global coordinate system. These
points
 * are used to draw the 'nLines' of the object.
 * Additionally, there may be 'nAux' auxillary lines which have a
previously
 * defined point at one end 'auxP1' which is the integer index of
an
 * existing point in X, Y, Z and a another point 'auxP2' for which
the
 * coordinates must be defined in elements after 'nPoints'. The
coordinates
 * for this second point are defined in the global coordinate sys-
tem and are
 * scaled with 'scaleAuxLines(factor)'.
 */

// members of the L3PCanvas class
// XY is coordinates5,0 and 5,1 are x,y for 5th pt
// NP is connectivity5,0 and 5,1 are +,- nodes for 5th member

boolean showPointNumber, showLineNumber, XYplot, middleButton-
Click,
showAuxLines;

public int    Point1[], Point2[], nLines, nPoints, nAux;
public int  auxP1[], auxP2[];
public double X[], Y[], Z[], lineThickness[];
public String labels[];
public Color lineColor, backColor, generalTextColor, auxLineColor,
pointNumberColor, lineNumberColor;
public boolean varyLineColors, varyLineThickness;
public Frame parentFrame;

private double rXYZ[][], R[][][], thetaX, thetaY, thetaZ;
private double XSHIFT, YSHIFT, SCALE, vAux[][];
private double AMAXX, AMAYY;// usable plotting area

private int canvasWidth, canvasHeight;
private int lastMouseX, lastMouseY;
private boolean         canRotate = true; // respond to mouse drag
private String tlString;

final private int MAX_LINE_THICKNESS = 10;
final private int MAX_DIMENSION = 1000;

/* basic constructor sets default values */
public L3PCanvas(int height, int width) {
// System.out.println(" begin L3PCanvas basic constructor \n");
X = new double [MAX_DIMENSION];
Y = new double [MAX_DIMENSION];
Z = new double [MAX_DIMENSION];
vAux = new double [MAX_DIMENSION][3];
rXYZ = new double [MAX_DIMENSION][3];
Point1 = new int [MAX_DIMENSION];
Point2 = new int [MAX_DIMENSION];
auxP1 = new int [MAX_DIMENSION];
auxP2 = new int [MAX_DIMENSION];
lineThickness = new double [MAX_DIMENSION];
labels = new String[MAX_DIMENSION];
R = new double[3][3][3];
canvasWidth = width;
```

```
canvasHeight = height;

addKeyListener(this);
addMouseListener(this);
addMouseMotionListener(this);

setSize(canvasWidth, canvasHeight);
setDefaults();
}

/* Constructor with a reference to the parent object*/
public L3PCanvas(int height, int width, Frame f) {
parentFrame = f;
// System.out.println(" L3PCanvas constructor with object refer-
ence \n");
X = new double [MAX_DIMENSION];
Y = new double [MAX_DIMENSION];
Z = new double [MAX_DIMENSION];
vAux = new double [MAX_DIMENSION][3];
rXYZ = new double [MAX_DIMENSION][3];
Point1 = new int [MAX_DIMENSION];
Point2 = new int [MAX_DIMENSION];
auxP1 = new int [MAX_DIMENSION];
auxP2 = new int [MAX_DIMENSION];
lineThickness = new double [MAX_DIMENSION];
labels = new String[MAX_DIMENSION];
R = new double[3][3][3];
canvasWidth = width;
canvasHeight = height;

addKeyListener(this);
addMouseListener(this);
addMouseMotionListener(this);

setSize(canvasWidth, canvasHeight);
setDefaults();
}

public void setDefaults() {
int i, j;
showAuxLines     = true;
showLineNumber   = false;
showPointNumber = false;
varyLineColors   = true;
varyLineThickness = false;

auxLineColor = Color.white;
backColor = Color.black;
generalTextColor = Color.red;
lineColor = Color.green;
lineNumberColor = Color.yellow;
pointNumberColor    = Color.red;

nLines = 0;
nAux = 0;
nPoints = 0;

AMAXX = canvasWidth - 80;// usable plotting area
AMAYY = canvasHeight - 80;

for(i=0; i<3; i++) // start with identity matrices
  for(j=0; j<3; j++)
    R[i][i][j] = 1.0;
}

private Color makeRgb(double val) {
// Construct three values, rgb from a single real value,
// 'val' which should be between 0 and 1 for proper scaling.

int i, rgb[] = new int[3];

if(val < 0) val = 0.0;
if(val > 1) val = 1.0;

if(val < 0.5) {
  // linear scaling between blue (val=0) and green (val=0.5)
  rgb[2] = (int)(2*255.0*(0.5-val));
  rgb[1] = 255 - rgb[2];
  rgb[0] = 0;
} else {
  // linear scaling between green (val=0.5) and red (val=1.0)
  rgb[2] = 0;
  rgb[1] = (int)(2*255.0*(1.0-val));
  rgb[0] = 255 - rgb[1];
}
```

```
// OK, so I'm slightly neurotic.
for(i=0; i<3; i++) {
  if(rgb[i] < 0) rgb[i] = 0;
  if(rgb[i] > 255) rgb[i] = 255;
}

// System.out.println(val + " " + rgb[0] + " " + rgb[1] + " " +
rgb[2]);

Color c = new Color(rgb[0], rgb[1], rgb[2]);
return c;
}

private int makeThickness(double val) {
// Return a positive integer 't' indicating the relative
// thickness (e.g., of a line) based on a double
// 'val' in the range of 0.0 to 1.0. This is based on
// the minimum value of 1 and a max of 'MAX_LINE_THICKNESS'.

int t;
if(val < 0) t = 1;
if(val > 1) t = MAX_LINE_THICKNESS;

t = (int)(val * MAX_LINE_THICKNESS);
if(t <= 0) t = 1;

return t;
}

public void paint(Graphics g) {
drawObject( (Graphics2D)g );
}

public void drawObject(Graphics2D g, int xsize, int ysize) {
// Force the object to be drawn in the size specified
// as in for printing to a page.

int oldWidth = canvasWidth;
int oldHeight = canvasHeight;

resetSize(xsize, ysize);
drawObject(g);
resetSize(oldWidth, oldHeight);
}

public void drawObject(Graphics2D g) {
int i;
int c[] = new int[3];
double x1, y1, x2, y2;

// Graphics2D g2 = new Graphics2D();
setBackground(backColor);
g.setColor(lineColor);
if(!varyLineColors && backColor == Color.white) g.set-
Color(Color.black);
// g.setFont();

// Plot the mesh
for(i = 1; i <= nLines; i++) {
// System.out.println("set color " + i);
if(varyLineColors) g.setColor(makeRgb(lineThickness[i]));
if(varyLineThickness)
  g.setStroke(new BasicStroke(makeThickness(lineThickness[i])));
int K = Point1[i];
int M = Point2[i];
x1 = (rXYZ[K][0] / SCALE) - XSHIFT;
y1 = (rXYZ[K][1] / SCALE) - YSHIFT;
x2 = (rXYZ[M][0] / SCALE) - XSHIFT;
y2 = (rXYZ[M][1] / SCALE) - YSHIFT;
g.drawLine( (int)x1, (int)y1, (int)x2, (int)y2);
}

// Show point numbers
if(showPointNumber) {
  g.setColor(pointNumberColor);
  for(i = 1; i <= nPoints; i++) {
x1 = (rXYZ[i][0] / SCALE) - XSHIFT;
y1 = (rXYZ[i][1] / SCALE) - YSHIFT;
g.drawString( " "+i,(int)x1, (int)y1);
  }
  g.setColor(lineColor);
}

// Show line numbers
if(showLineNumber) {
  g.setColor(lineNumberColor);
```

```
  for(i = 1; i <= nLines; i++) {
int K = Point1[i];
int M = Point2[i];
x1 = (rXYZ[K][0] / SCALE) - XSHIFT;
y1 = (rXYZ[K][1] / SCALE) - YSHIFT;
x2 = (rXYZ[M][0] / SCALE) - XSHIFT;
y2 = (rXYZ[M][1] / SCALE) - YSHIFT;
x1 = (x1 + x2)/2;
y1 = (y1 + y2)/2;
g.drawString( " "+i,(int)x1, (int)y1);
  }
  g.setColor(lineColor);
}

// Draw the auxillary lines
if(showAuxLines) {
  for(i=1; i<=nAux; i++){
// g.setColor(makeRgb(lineThickness[i+nLines]));
g.setColor(auxLineColor);
int K = auxP1[i];
int M = auxP2[i];
x1 = (rXYZ[K][0] / SCALE) - XSHIFT;
y1 = (rXYZ[K][1] / SCALE) - YSHIFT;
x2 = (rXYZ[M][0] / SCALE) - XSHIFT;
y2 = (rXYZ[M][1] / SCALE) - YSHIFT;
  // System.out.println("aux "+i+" ("+K+", "+M+")" );
  // System.out.println("x1 - y2 "+ x1+" "+y1+" "+x2+" "+y2 );
g.drawLine( (int)x1, (int)y1, (int)x2, (int)y2);
  }
}

// Show color scale legend
if(varyLineColors) {
int w=20, h=5, y=10, x=canvasWidth-40;
double val;

for(i=0; i<15; i++) {
  val = 1.0 - i/15.0;
  g.setColor( makeRgb(val) );
  // g.fillRect(10, y, w, h);
  g.fillRect(x, y, w, h);
  y += h;
}
}
// Show the filename tag at the top left to the right of the
if(tlString != null) {
  if(backColor == Color.black) g.setColor(Color.white);
  else g.setColor(Color.black);
  g.drawString(tlString,10, 20);
}

}

// required for KeyListener
// These will simply pass all KeyEvents to the parentFrame where
// they will be handled optionally
public void keyPressed(KeyEvent e) {
if(parentFrame != null) parentFrame.dispatchEvent(e);
}
public void keyReleased(KeyEvent e) {
if(parentFrame != null) parentFrame.dispatchEvent(e);
}
public void keyTyped(KeyEvent e) {
if(parentFrame != null) parentFrame.dispatchEvent(e);
}

/* required for MouseListener */
public void mouseClicked( MouseEvent e)  { }
public void mouseReleased( MouseEvent e) { }
public void mouseEntered( MouseEvent e)  { }
public void mouseExited( MouseEvent e)   { }
public void mouseMoved( MouseEvent e)    { }

public void mousePressed( MouseEvent e) {
lastMouseX = e.getX();
lastMouseY = e.getY();
middleButtonClick = false;
if ( e.isAltDown() ) {
  // System.out.println("click with middle");
  middleButtonClick = true;
}
}

public void mouseDragged( MouseEvent e) {
// System.out.println(" drag at x = " + e.getX() + ", y = " +
e.getY() );
```

```
int dx, dy;
double angle;

dx = e.getX() - lastMouseX;
dy = e.getY() - lastMouseY;
lastMouseX = e.getX();
lastMouseY = e.getY();

if(canRotate) {
if(middleButtonClick) {
  // if(e.getX() > canvasWidth / 2)
  // Later form the angle of rotation here from dx, dy and location
  // but for now assume the click was at the lower center of the
canvas
  // and simply use dx.
  angle = dx / 100.0;
  addRotation(angle);
} else {
  addRotation(dx, dy);
}
}

repaint();
}

private void mkRotationMatrix() {

R[0][0][0] = 1.0;
R[1][1][0] = Math.cos(thetaX);
R[1][2][0] = Math.sin(thetaX);
R[2][1][0] = -Math.sin(thetaX);
R[2][2][0] = Math.cos(thetaX);

R[0][0][1] = Math.cos(thetaY);
R[0][2][1] = -Math.sin(thetaY);
R[1][1][1] = 1.0;
R[2][0][1] = Math.sin(thetaY);
R[2][2][1] = Math.cos(thetaY);

R[0][0][2] = Math.cos(thetaZ);
R[0][1][2] = Math.sin(thetaZ);
R[1][0][2] = -Math.sin(thetaZ);
R[1][1][2] = Math.cos(thetaZ);
R[2][2][2] = 1.0;

}

private void rotate() {
/* When looking at the screen, consider it to be in a local xy
plane
 * and the viewer is looking along the z-axis. This will allow
only
 * two types of rotations due to a mouse drag: about x-axis and
about
 * y-axis. The dy increment indicates about the x-axis while a dx
 * increment calls for rotation about the y-axis.
 */
int i, j, k;
double temp[] = new double[3];

for(i=1; i<=nPoints+nAux; i++) {
rXYZ[i][0] = X[i];
rXYZ[i][1] = Y[i];
rXYZ[i][2] = Z[i];
}

for(i=1; i<=nPoints+nAux; i++) {
  for(k=0; k<3; k++) {// do rotation about x, y, and z axis
    for(j=0; j<3; j++) temp[j] = R[j][0][k]*rXYZ[i][0]
+ R[j][1][k]*rXYZ[i][1] + R[j][2][k]*rXYZ[i][2];
    for(j=0; j<3; j++) rXYZ[i][j] = temp[j];
  }
}

setRange();
}

private void addRotation(int dx, int dy) {
// System.out.println(" drag dx = " + dx + ", dy = " + dy );
// 0.17 rad ~ 10 degrees
thetaX += -dy / 100.0;
thetaY += dx / 100.0;
mkRotationMatrix();
rotate();
}
```

```
private void addRotation(double angle) {
// Rotation about z-axis
thetaZ += angle;
mkRotationMatrix();
rotate();
}

public void resetSize(int w, int h) {
// Reset the size of this object. Should be called explicitly
// when the container is resized. Not applicable for applets.
canvasWidth = w;
canvasHeight = h;
AMAXX = canvasWidth - 80;// usable plotting area
AMAYY = canvasHeight - 80;
setRange();
repaint();
}

private void setRange() {
// This sets the scale factors for the object, not the
// auxLines.

double x, y, Xmax, Xmin, Ymax, Ymin, Xrange, Yrange;
int i;

Xmax = 0;
Xmin = 0;
Ymax = 0;
Ymin = 0;

// for(i=1; i <= nPoints+nAux; i++) {
for(i=1; i <= nPoints; i++) {
  x = rXYZ[i][0];
  y = rXYZ[i][1];
  if(x > Xmax) Xmax = x;
  if(x < Xmin) Xmin = x;
  if(y > Ymax) Ymax = y;
  if(y < Ymin) Ymin = y;
}

Xrange = Xmax - Xmin;
Yrange = Ymax - Ymin;

// Scale so everything fits but x and y are proportional
if((Xrange/AMAXX) > (Yrange/AMAYY)) SCALE = Xrange/AMAXX;
if((Xrange/AMAXX) < (Yrange/AMAYY)) SCALE = Yrange/AMAYY;

// shift in pixels to align centroid of 2D object
XSHIFT = (Xmax+Xmin)/2/SCALE - canvasWidth/2;
YSHIFT = (Ymax+Ymin)/2/SCALE - canvasHeight/2;

}


//////////////////////////////////////////////////////////////
//
// public methods including get and set for private variables
//
public void scaleLineThickness() {
int i;
double max = 0.0;

for(i=1; i <= nAux; i++) lineThickness[nLines+i] = 0.5;
for(i=1; i <= nLines; i++)
  max = (lineThickness[i] > max ? lineThickness[i] : max);
for(i=1; i <= nLines; i++) {
  lineThickness[i] /= max;
  // System.out.println("scaleLine: " + lineThickness[i]);
}
}

public double[] getRotation() {
// Return the current rotation angles in degrees
double r[] = new double[3];
r[0] = thetaX * 180.0 / Math.PI;
r[1] = thetaY * 180.0 / Math.PI;
r[2] = thetaZ * 180.0 / Math.PI;
return r;
}

public void setRotation(double x, double y, double z) {
// Set the view to the specified rotation
// angles in degrees

thetaX = x * Math.PI / 180;
```

```
thetaY = y * Math.PI / 180;
thetaZ = z * Math.PI / 180;
mkRotationMatrix();

rotate();
return;
}

public void invertBackground() {
// This method causes black background to go to
// white and any other color to go to black.

if(backColor == Color.black) backColor = Color.white;
else backColor = Color.black;
}

public void setCanRotate(boolean val) {
canRotate = val;
}

public void setTopLeftString(String f) {
tlString = f;
}

public String getTopLeftString() {
return tlString;
}

public void refresh() {
rotate();
}

public Dimension getPreferredSize() {

// return new Dimension(canvasWidth, canvasHeight);
return new Dimension(100, 100);

}

}
```

```
File: MyFiles.java
import java.io.*;
import java.awt.*;

class MyFiles extends FileDialog {

String fileName;

public MyFiles( Frame f, int type) {
super(f, (type == FileDialog.LOAD ? "Open " : "Save " ) + "File",
type);
fileName = "";
File SelectedFileObject;
setSize(400, 400);
show();

if(getFile() != null) {
fileName = getFile();
SelectedFileObject = new File(fileName);
// System.out.println("\n Selected file name is " + fileName);
// if(SelectedFileObject.exists()) System.out.println("  File
Exists ");
} else {
// System.out.println("\n no file selected ");
}
}

}
```

```
File: MyMath.java
import java.awt.*;
import java.applet.Applet;
import java.lang.String;
import java.lang.Integer;
import java.text.*;// format numbers

public class MyMath {

static int sign(double d) {
if(d==0) return 0;
else if(d<0) return -1;
return 1;
}

static double[][] minv (double a[][]) {
   /*
    * Return the inverse of a matrix. This is the simplest form and
doesn't check
    * for singularities nor include pivoting. The procedure here
is to start with
    * an augmented matrix, 'A' which contains the original matrix,
'a' on the
    * left and an identity matrix on the right. The augmented
matrix is then
    * reduced so there is an identity matrix on the left and the
inverse of the
    * original matrix will remain on the right.
    *
    */
int i, ii, j, jj, dim;
double temp;

dim = a.length;
double Ainv[][] = new double[dim][dim];// this will be the inverse
double A[][] = new double[dim][2*dim];// augmented matrix

for(i=0; i<dim; i++) { // initialize augmented matrix
  for(j=0; j<dim; j++) {
  A[i][j] = a[i][j];
  }
  A[i][i+dim] = 1.0;
}
// System.out.println(" MyMath::minv, dim="+dim);

// create upper triangular with 1.0 on diagonal
for(i=0; i<dim; i++) {

  // set leading 1 in this row
  temp = A[i][i];
  for(j=i; j<2*dim; j++)  A[i][j] = A[i][j] / temp;

  for(ii=i+1; ii<dim; ii++) { // set leading 0 in other rows
    temp = A[ii][i];
    for(jj=i; jj<2*dim; jj++) {
A[ii][jj] = A[ii][jj] - temp * A[i][jj];
    }
  }
}

// now create I from upper triangular; zero the upper part
for(j=1; j<dim; j++) {

  // ii is the row with 1.0 on diagonal in the (i+1) column
  for(ii=0; ii<j; ii++) { // set 0 in all other rows
    temp = A[ii][j];
    for(jj=j; jj<2*dim; jj++) {
A[ii][jj] = A[ii][jj] - temp * A[j][jj];
    }
  }
}

for(i=0; i<dim; i++)
  for(j=0; j<dim; j++)
Ainv[i][j] = A[i][j+dim];

return Ainv;
}

static int mmult (double a[][], double b[][], double c[][]) {
//
// Perform c = a * b
//
int i, j, k;

// double temp[][] = new double[a.length][b[0].length];
```

```
// a.length is the number of rows and a[0].length is the
// number of columns
if( (a[0].length != b.length) || (a.length != c.length)
 || (c[0].length != b[0].length) ) {
  System.out.println("***Error::MyMath::mmult dimensions");
  System.out.println("*** c=a*b, "
    + "a("+a.length+" x "+a[0].length+"), "
    + "b("+b.length+" x "+b[0].length+"), "
    + "c("+c.length+" x "+c[0].length+")" );
  return 1;
}
for(i=0; i<a.length; i++) {
  for(j=0; j<b[0].length; j++) {
    c[i][j] = 0.0;
    for(k=0; k<a[0].length; k++) c[i][j] += a[i][k]*b[k][j];
  }
}

return 0;
}

static int mmult (double a[][], double b[][], double c[][], double
d[][]) {
//
// Perform d = a * b * c
//
int i, j, k;

// a.length is the number of rows and a[0].length is the
// number of columns
if( (a[0].length != b.length) || (b[0].length != c.length)
 || (d.length != a.length) || (d[0].length != c[0].length) ) {
  System.out.println("***Error::MyMath::mmult dimensions");
  System.out.println("*** d=a*b*c, "
    + "a("+a.length+" x "+a[0].length+"), "
    + "b("+b.length+" x "+b[0].length+"), "
    + "c("+c.length+" x "+c[0].length+"), "
    + "d("+d.length+" x "+d[0].length+")" );
  return 1;
}
double temp[][] = new double[a.length][b[0].length];
for(i=0; i<d.length; i++) for(j=0; j<d[0].length; j++) d[i][j] =
0.0;

for(i=0; i<a.length; i++)
  for(j=0; j<b[0].length; j++)
    for(k=0; k<a[0].length; k++) temp[i][j] += a[i][k]*b[k][j];

for(i=0; i<a.length; i++)
  for(j=0; j<c[0].length; j++)
    for(k=0; k<a[0].length; k++) d[i][j] += temp[i][k]*c[k][j];

return 0;
}

static void scalarMult (double a[][], double b) {
int i, j;

// a.length is the number of rows and a[0].length is the
// number of columns
for(i=0; i<a.length; i++)
  for(j=0; j<a[0].length; j++)
    a[i][j] *= b;
}

static double[] vmult (double a[][], double b[]) {
int i, j;
double temp[] = new double[a.length];

// a.length is the number of rows and a[0].length is the
// number of columns
if(a[0].length != b.length) {
  System.out.println("***Error::MyMath::vmult dimensions");
  return temp;
}
for(i=0; i<a.length; i++) {
    for(j=0; j<a[0].length; j++) temp[i] += a[i][j]*b[j];
}
return temp;
}

static double[][] transpose (double a[][]) {
int i, j;
double temp[][] = new double[a[0].length][a.length];
```

```
for(i=0; i<a.length; i++) {
  for(j=0; j<a[0].length; j++) {
  temp[j][i] = a[i][j];
  }
}

return temp;
}

static int solve (double a[][], double b[], double x[]) {
   /*
    * Solve Ax = b using Gaussian elimination
    *
    * This is the simplest form of solving simultaneous equations.
No
    * checks for singularity are performed. Pivoting is also not
done.
    * Matrix 'a' is destroyed in this method but resrored from
'Asave'.
    *
    */

int i, ii, j, rows, cols;
double lead;

// check dimensions
if (a.length != b.length || b.length != x.length
  || a.length != a[0].length ) {
    System.out.println ("***Error::MyMath::solve, bad dimen-
sions\n\n");
    // exit(-1);
}

rows = a.length;
cols = a[0].length;

double Asave[][] = new double[rows][rows];
for(i=0; i<rows; i++)
  for(j=0; j<rows; j++)
    Asave[i][j] = a[i][j];

for (i = 0; i < rows; i++) x[i] = b[i];

/* create upper triangular matrix (zeros below diagonal) */
for (i = 0; i < rows; i++) {
    // assert (a[i][i] != 0);// no zeros on main diagonal
    if(a[i][i]==0) System.out.println("***Error::MyMath::solve
zero on main diag.");
    // zero column i in all rows below this one
    for (ii = i+1; ii < rows; ii++) {
x[ii] =  x[ii] - a[ii][i] / a[i][i] * x[i];
// save the lead term of this row for row operation
lead = a[ii][i];
for(j = i; j < cols; j++) {
  a[ii][j] =  a[ii][j] - lead / a[i][i] * a[i][j];
    }
    }
}

/* now zero upper part (above diagonal), start from bottom */
for (i=rows-1; i>=0; i--) {
    // assert (a[i][i] != 0);// no zeros on main diagonal
    // zero column i in all rows above this one
    if(a[i][i]==0) System.out.println("***Error::MyMath::solve
zero on main diag.");
    for (ii = 0; ii < i; ii++) {
// printf("ii: %d\n", ii);
    // no need to explicitly put the zeros in matrix a, only do x
x[ii] -=  a[ii][i] / a[i][i] * x[i];
    }
    x[i] /= a[i][i];
    // printf(" \t inside solve, soln: %d, %f\n", i, x.m[i]);
}

for(i=0; i<rows; i++)
  for(j=0; j<rows; j++)
    a[i][j] = Asave[i][j];
return 0;
   }
}
//
//end class 'MyMath'
//
//////////////////////////////////////////////////////
```

```
File: OptSetFrame.java
import java.awt.*;
import java.applet.Applet;
import java.lang.String;
import java.lang.Integer;
import java.awt.event.*;

//////////////////////////////////////////////////////////////
//
// begin class OptSetFrame
//
// Dialog to get a few variables that pertain to the optimization
// algorithm. These variables are public and need to be accesed in
// this class.
//

public class OptSetFrame extends Frame implements WindowListener,
ActionListener {

public int totalIterMax, solverIterMax, viewPause;
public doublealpha;
private Label labels[];
private TextField textFields[];
private Button setButton, dismissButton;

//////////////////////////////////////////////////////////
//
// constructor
//
public OptSetFrame() {

int i, numLabels;
GridBagConstraints constraints = new GridBagConstraints();
GridBagLayout gridbag = new GridBagLayout();
setLayout(gridbag);

//////////////////////////////////////////////////////////
//
// initialize the items that will be placed on the frame
//
//
numLabels = 4;
labels = new Label[numLabels];
textFields = new TextField[numLabels];
i = 0;
labels[i++] = new Label("SLP Iterations");
labels[i++] = new Label("Solver Iterations");
labels[i++] = new Label("SLP step (alpha)");
labels[i++] = new Label("View Pause (ms)");

for(i=0; i < numLabels; i++) textFields[i] = new TextField(6);

setButton = new Button("Set");
dismissButton = new Button("Dismiss");

setButton.addActionListener(this);
dismissButton.addActionListener(this);

totalIterMax = 5;
solverIterMax = 5;
alpha = 0.5;
viewPause = 20;

//////////////////////////////////////////////////////////
//
// place these items on the frame
//
//
constraints.fill       = GridBagConstraints.HORIZONTAL;
constraints.gridwidth  = 1;
constraints.gridheight = 1;

// Labels and TextFields
for(i=0; i < numLabels; i++) {
  constraints.gridx  = 0;
  constraints.gridy  = i;
  constraints.anchor = GridBagConstraints.EAST;
  gridbag.setConstraints(labels[i], constraints);
  add(labels[i]);
  constraints.gridx = 1;
  constraints.anchor = GridBagConstraints.WEST;
  gridbag.setConstraints(textFields[i], constraints);
  add(textFields[i]);
}

textFields[0].setText( Integer.toString(totalIterMax) );
```

```
textFields[1].setText( Integer.toString(solverIterMax) );
textFields[2].setText( Double.toString(alpha) );
textFields[3].setText( Integer.toString(viewPause) );

// buttons
constraints.gridx  = 0;
constraints.gridy++;
constraints.gridwidth = 1;
constraints.anchor = GridBagConstraints.SOUTH;
constraints.weighty = 1.0;
gridbag.setConstraints(setButton, constraints);
add(setButton);
constraints.gridx++;
gridbag.setConstraints(dismissButton, constraints);
add(dismissButton);

setTitle("Optimization Settings" );

addWindowListener(this);
}

public void windowClosed(WindowEvent event) { }
public void windowDeiconified(WindowEvent event) { }
public void windowIconified(WindowEvent event) { }
public void windowActivated(WindowEvent event) { }
public void windowDeactivated(WindowEvent event) { }
public void windowOpened(WindowEvent event) { }
public void windowClosing(WindowEvent event) {
// if (inAnApplet) {
setVisible(false);
dispose();
// } else { System.exit(0); }
}

public String toString() {
return "three variables";
}

public void setString(String s) {
}

public void actionPerformed(ActionEvent e) {
if(e.getSource() == dismissButton) {
  setVisible(false);
  dispose();
}else if(e.getSource() == setButton) {
  // the layout above defines the order
  try {
totalIterMax = Integer.parseInt( textFields[0].getText() );
solverIterMax = Integer.parseInt( textFields[1].getText() );
alpha = Double.valueOf( textFields[2].getText() ).doubleValue();
viewPause = Integer.parseInt( textFields[3].getText() );

// check that the values are reasonable
if(totalIterMax > 50) totalIterMax = 50;
if(solverIterMax > 50) solverIterMax = 50;
if(alpha > 0.9) alpha = 0.9;
if(viewPause > 3000) viewPause = 3000;
if(totalIterMax < 1) totalIterMax = 1;
if(solverIterMax < 1) solverIterMax = 1;
if(alpha < 0.01) alpha = 0.01;
if(viewPause < 0) viewPause = 0;
  } catch(NumberFormatException nfe) {
// System.out.println("OptSetFrame: bad input values\n"+ nfe);
// set default values
totalIterMax = 5;
solverIterMax = 5;
alpha = 0.5;
viewPause = 20;
  }
}
}
}

//
// end class OptSetFrame
//
//////////////////////////////////////////////////
```

```
File: SimpleMessage.java
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;

//////////////////////////////////////////////////////////////
//
// begin class SimpleMessage
//
// This class describes a simple frame with a message.

public class SimpleMessage extends Frame implements WindowLis-
tener, ActionListener {

private Button dismissButton;
private Label msgLabel;

// constructors
public SimpleMessage(String msg) {
dismissButton = new Button("Dismiss");
dismissButton.addActionListener(this);
msgLabel = new Label(msg);
GridBagConstraints constraints = new GridBagConstraints();
GridBagLayout gridbag = new GridBagLayout();
setLayout(gridbag);
constraints.anchor = GridBagConstraints.CENTER;
constraints.weighty = 1.0;
gridbag.setConstraints(msgLabel, constraints);
add(msgLabel);
constraints.weighty = 0.0;
constraints.gridy = 1;
constraints.anchor = GridBagConstraints.SOUTH;
constraints.fill = GridBagConstraints.HORIZONTAL;
gridbag.setConstraints(dismissButton, constraints);
add(dismissButton);
addWindowListener(this);
setSize(300, 100);
setVisible(true);
dismissButton.requestFocus();
}

public SimpleMessage(String msg, String title) {
dismissButton = new Button("Dismiss");
dismissButton.addActionListener(this);
msgLabel = new Label(msg);
GridBagConstraints constraints = new GridBagConstraints();
GridBagLayout gridbag = new GridBagLayout();
setLayout(gridbag);
constraints.anchor = GridBagConstraints.CENTER;
constraints.weighty = 1.0;
gridbag.setConstraints(msgLabel, constraints);
add(msgLabel);
constraints.weighty = 0.0;
constraints.gridy = 1;
constraints.anchor = GridBagConstraints.SOUTH;
constraints.fill = GridBagConstraints.HORIZONTAL;
gridbag.setConstraints(dismissButton, constraints);
add(dismissButton);
addWindowListener(this);
setTitle(title);
setSize(300, 100);
setVisible(true);
dismissButton.requestFocus();
}

public void setMessage(String m) {  msgLabel.setText(m); }

public void windowClosed(WindowEvent event) { }
public void windowDeiconified(WindowEvent event) { }
public void windowIconified(WindowEvent event) { }
public void windowActivated(WindowEvent event) { }
public void windowDeactivated(WindowEvent event) { }
public void windowOpened(WindowEvent event) { }
public void windowClosing(WindowEvent event) {
setVisible(false);
dispose();
}

public void actionPerformed(ActionEvent e) {
if (e.getSource() == dismissButton) {
setVisible(false);
dispose();
}
}

}
```

```
File: TextFileFrame.java
import java.awt.*;
import java.applet.Applet;
import java.lang.String;
import java.lang.Integer;
import java.awt.event.*;

//////////////////////////////////////////////////////////////
//
// begin class TextFileFrame
//
// Dialog to simulate 'text file input' by using a text area. The
// intent is that other classes will access the 'textString' mem-
ber
// of this class as if it were a text file that had been read in as
// one long string. The 'setString(Sring s)' must be called to
change
// contents of the static variable 'textString' but it can be
accessed
// any time by other classes.
//

public class TextFileFrame extends Frame implements WindowLis-
tener, ActionListener {

public  TextAreatextFileContents;
public  String textString;
private Label infoLabel;
private Button setButton, dismissButton;

// constructor with dismiss button only
public TextFileFrame(int visRows, int visCols, String title, bool-
ean setable) {

// setFont(new Font("TimesRoman", Font.PLAIN, 20));
setFont(new Font("Helvetica", Font.PLAIN, 16));
GridBagConstraints constraints = new GridBagConstraints();
GridBagLayout gridbag = new GridBagLayout();
setLayout(gridbag);

//////////////////////////////////////////////////////
//
// initialize the items that will be placed on the frame
//
textFileContents = new TextArea(visRows, visCols);
infoLabel = new Label("Copy-Paste OK in this field");
setButton = new Button("Set");
dismissButton = new Button("Dismiss");

setButton.addActionListener(this);
dismissButton.addActionListener(this);

//////////////////////////////////////////////////////
//
// place these items on the frame
//
// text area
constraints.gridx  = 0;
constraints.gridy  = 0;
constraints.gridwidth  = 2;
constraints.weightx  = 1.0;
constraints.weighty  = 1.0;
constraints.gridheight = 1;
constraints.fill       = GridBagConstraints.HORIZONTAL;
constraints.anchor     = GridBagConstraints.NORTH;
gridbag.setConstraints(textFileContents, constraints);
add(textFileContents);

// description label
constraints.gridy++;
constraints.weightx  = 0.0;
constraints.weighty  = 0.0;
constraints.anchor = GridBagConstraints.WEST;
constraints.fill   = GridBagConstraints.NONE;
gridbag.setConstraints(infoLabel, constraints);
// add(infoLabel);

// buttons
constraints.gridy++;
constraints.gridwidth = 1;
if(setable) {
  constraints.anchor    = GridBagConstraints.EAST;
  gridbag.setConstraints(setButton, constraints);
  add(setButton);
}
constraints.anchor    = GridBagConstraints.EAST;
```

```
constraints.gridx++;
gridbag.setConstraints(dismissButton, constraints);
add(dismissButton);
setTitle(title);
addWindowListener(this);
}

public void windowClosed(WindowEvent event) { }
public void windowDeiconified(WindowEvent event) { }
public void windowIconified(WindowEvent event) { }
public void windowActivated(WindowEvent event) { }
public void windowDeactivated(WindowEvent event) { }
public void windowOpened(WindowEvent event) { }
public void windowClosing(WindowEvent event) {
setVisible(false);
dispose();
}

public String toString() {
return textString;
}

public void setString(String s) {
textString = s;
textFileContents.setText(s);
}

public void actionPerformed(ActionEvent e) {
if(e.getSource() == dismissButton) {
setVisible(false);
dispose();
}else if(e.getSource() == setButton) {
textString = textFileContents.getText();
}
}
}
//
// end class TextFileFrame
//
////////////////////////////////////////////////////////////////
```

```
File: Tr2d.java
import java.awt.*;
import java.util.*;// StringTokenizer
import java.lang.String;
import java.lang.Integer;
import java.text.*;// format numbers
import java.io.*;

// begin the 'Tr2d' class
public class Tr2d {

// member variables for this class
//
//
int NB, NS, NN, dof, freeNodes;
double Ke[][], Kinv[][], N_sys[][];
TrussNode nodes[];
TrussMemberbars[];

/* basic constructor */
public Tr2d(int nb, int nn, int ns) {
int i;

NB = nb;
NN = nn;
NS = ns;
freeNodes = NN - NS;
dof = 2 * freeNodes;

Ke = new double [dof][dof];
Kinv = new double [dof][dof];
N_sys = new double [nb][dof];

/* nodes and bars are one-based (i.e., first element is index 1,
not 0) */
nodes = new TrussNode[nn+1];
bars = new TrussMember[nb+1];
for(i=1; i <= nn; i++) nodes[i] = new TrussNode();
for(i=1; i <= nb; i++) bars[i]  = new TrussMember();
// System.out.println("end Tr2d constructor");
}
```

```
public Tr2d(String s) {
int i;
StringTokenizer t = new StringTokenizer(s);

// System.out.println("begin Tr2d(String s) constructor");

NB = Integer.parseInt( t.nextToken() );
NN = Integer.parseInt( t.nextToken() );
NS = Integer.parseInt( t.nextToken() );
freeNodes = NN - NS;
dof = 2 * freeNodes;

Ke = new double [dof][dof];
Kinv = new double [dof][dof];
N_sys = new double [NB][dof];

/* nodes and bars are one-based (i.e., first element is index 1,
not 0) */
nodes = new TrussNode[NN+1];
bars = new TrussMember[NB+1];
for(i=1; i <= NN; i++) {
nodes[i] = new TrussNode();
nodes[i].x = Double.valueOf( t.nextToken() ).doubleValue();
nodes[i].y = Double.valueOf( t.nextToken() ).doubleValue();
nodes[i].Px = Double.valueOf( t.nextToken() ).doubleValue();
nodes[i].Py = Double.valueOf( t.nextToken() ).doubleValue();
}

for(i=1; i <= NB; i++) {
bars[i]  = new TrussMember();
bars[i].NM = Integer.parseInt( t.nextToken() );
bars[i].NP = Integer.parseInt( t.nextToken() );
bars[i].area = Double.valueOf( t.nextToken() ).doubleValue();
bars[i].E = Double.valueOf( t.nextToken() ).doubleValue();
}

}

public int assemble()
{
/* This method assembles the stiffness matrix, Ke. */

int i, j, ii;

// System.out.println("begin assemble");

for (i=0; i<dof; i++)  // zero Ke
  for (j=0; j<dof; j++)
  Ke[i][j] = 0.0;

for (i=1; i<=NB; i++) {// set up Ke
  makeUnitVector(i);
  insertElement(i);
}

for (i=0; i<NB; i++) { // set up N matrix
  j = bars[i+1].NM;
  if(2*j <= dof) {
N_sys[i][2*(j-1)]   = -bars[i+1].UV[0];
N_sys[i][2*(j-1)+1] = -bars[i+1].UV[1];
  }
  j = bars[i+1].NP;
  if(2*j <= dof) {
N_sys[i][2*(j-1)]   = bars[i+1].UV[0];
N_sys[i][2*(j-1)+1] = bars[i+1].UV[1];
  }
}

/*System.out.println("in assemble, Ke:");
for(i=0; i<dof; i++)
  for(j=0; j<dof; j++)
System.out.println(" " + i + " " + j + " " + Ke[i][j]);
*/
return 0;

}

public int analyze()
{
int i, ii, j;
double D1, FAC;

double x[], b[];
x = new double[dof];
b = new double[dof];
```

```
/* assemble system matrix before analysis */
assemble();

for (i=0; i<freeNodes; i++) {// set rhs vector
  b[2*i] = nodes[i+1].Px;
  b[2*i+1] = nodes[i+1].Py;
}

// solve the system
MyMath.solve(Ke, b, x);
Kinv = MyMath.minv(Ke);

// System.out.println("analyze:solution:");
// for(i=0; i<dof; i++) System.out.println(i + " " + x[i]);

for (i=0; i<freeNodes; i++){/* set displacements */
  nodes[i+1].dx = x[2*i];
  nodes[i+1].dy = x[2*i+1];
}

/*  compute member forces and displacements from joint displace-
ments */
for(i=1; i<=NB; i++) {
  ii = bars[i].NM;
  bars[i].dl=0.0;
  FAC= -1.0;
  for(j=1; j<=2; j++) {
if(ii <= freeNodes) {
  bars[i].dl += FAC * (
nodes[ii].dx*bars[i].UV[0] +
nodes[ii].dy*bars[i].UV[1] );
}
FAC=1.0;
ii = bars[i].NP;
  }
  bars[i].force = bars[i].dl*bars[i].E*bars[i].area/
bars[i].length;
  // bars[i].force = 3.14159;
  bars[i].stress= bars[i].force/bars[i].area;
}

return 0;
}
```

```
int makeUnitVector(int b)
{
double dx, dy, L;

// Using zero based array for unit vector!!
dx = nodes[bars[b].NP].x - nodes[bars[b].NM].x;// delta x
dy = nodes[bars[b].NP].y - nodes[bars[b].NM].y;
L = dx*dx + dy*dy;
L = Math.sqrt(L);  // length

bars[b].UV[0] = dx / L;
bars[b].UV[1] = dy / L;
bars[b].length = L;
return 0;
}

public String toString() {
String s;
String tab = "\t";
String nl = "\n";
DecimalFormat fmt = new DecimalFormat("0.0000");

s = NB + tab + NN + tab + NS + nl;
for(int i=1; i<=NN; i++)
  s += nodes[i].x + tab + nodes[i].y + tab
+ nodes[i].Px + tab + nodes[i].Py + nl;
for(int i=1; i<=NB; i++)
  s += bars[i].NM + tab + bars[i].NP + tab
+ fmt.format(bars[i].area) + tab + bars[i].E + nl;
return s;
}

public String echoStructure() {
String s;
DecimalFormat fmt = new DecimalFormat("0.0000");

s = NB + " bars\n" + NN + " nodes\n" + NS + " supports\n";
s += "\nnode\t x \t y \t Px \t Py \n";
for(int i=1; i<=NN; i++)
  s += " " + i + " \t " + nodes[i].x + " \t " + nodes[i].y + "\t"
+ nodes[i].Px + " \t " + nodes[i].Py + "\n";
s += "\nBar\t N- \t N+ \t Area \t E \n";
for(int i=1; i<=NB; i++)
```

```
  s += " " + i + " \t " + bars[i].NM + " \t " + bars[i].NP + "\t"
+ fmt.format(bars[i].area) + " \t " + bars[i].E + "\n";

return s;
}

public double volume() {
double vol = 0.0;
String s;
DecimalFormat fmt = new DecimalFormat("0.000");

for(int i=1; i<=NB; i++) vol += bars[i].area * bars[i].length;
// s = "Volume: \t " + fmt.format(vol) + "\n";

return vol;
}

public String JointDisplacements() {
String s;
DecimalFormat fx_6 = new DecimalFormat("0.000000");

s = "Joint \t dx \t\t dy \n";
for(int i=1; i<=NN; i++)
  s += " " + i + " \t " + fx_6.format(nodes[i].dx) + " \t "
    + fx_6.format(nodes[i].dy) + "\n";

return s;
}

public String MemberDisplacements() {
String s;
DecimalFormat fx_6 = new DecimalFormat("0.000000");
DecimalFormat fx_3 = new DecimalFormat("0.000");

s = "Bar\t Force \t Stress \t dl \n";
for(int i=1; i<=NB; i++)
  s += " " + i + " \t " + fx_3.format(bars[i].force)
    + " \t " + fx_3.format(bars[i].stress)
    + " \t " + fx_6.format(bars[i].dl) + "\n";

return s;
}
int echoBars() {// echo results to screen
return 0;
}

int insertElement(int b)
{
/* Insert the contribution of bar b to the global stiffness
matrix. Each
 * element contributes nkn to Ke at at least one node (unless the
member
 * has no free nodes) and may contribute at two nodes if both nodes
are
 * free. In this case, there are also 'cross terms' that 'connect'
node1
 * to node2 and the contribution here is simply -nkn. The approach
here
 * is to form the 2x2 matrix nkn for this element and then place it
in the
 * correct locations (if the bar is connected to any free nodes).
 */

int i, j, n1, n2;
double K_i;
double nkn[][] = new double[2][2];

K_i = bars[b].area * bars[b].E / bars[b].length;
nkn[0][0] = K_i * bars[b].UV[0] * bars[b].UV[0];
nkn[0][1] = K_i * bars[b].UV[1] * bars[b].UV[0];
nkn[1][1] = K_i * bars[b].UV[1] * bars[b].UV[1];
nkn[1][0] = nkn[0][1];


n1 = 2*(bars[b].NM - 1);// get first dof for (-) end of this bar
n2 = 2*(bars[b].NP - 1);// get first dof for (+) end of this bar
if(n1 < dof) {  // NM is a free node
  for(i = n1; i < n1+2; i++)
    for(j = n1; j < n1+2; j++)
Ke[i][j] += nkn[i-n1][j-n1];
  if(n2 < dof) {// both NM and NP are free nodes
    for(i = n1; i < n1+2; i++)
      for(j = n2; j < n2+2; j++) {
Ke[i][j] += -nkn[i-n1][j-n2];
Ke[j][i] += -nkn[i-n1][j-n2];
      }
```

```
  }
}

if(n2 < dof) {// NP is a free node
  for(i = n2; i < n2+2; i++)
    for(j = n2; j < n2+2; j++)
Ke[i][j] += nkn[i-n2][j-n2];
}
return 0;
}

boolean readFile(String fileName) {
BufferedReader b = null;
StringTokenizer t;
String s = “”;
int i;

try {
b = new BufferedReader ( new FileReader(fileName) );
} catch(IOException e) {
System.err.println(“Input file not opened properly \n”
  + e.toString() );
SimpleMessage msg = new SimpleMessage(“file open failed”
  + e, “File Not Opened”);
b = null;
return false;
// System.exit(1);
}

// Read each line as a String and tokenize to get the entries.
// System.out.println(“readFile(“ + fileName + “)”);
if(b != null) {
  try {
s = b.readLine();
for(i=1; i <= NN; i++) {
  s = b.readLine();
  t = new StringTokenizer(s);
  nodes[i] = new TrussNode();
  nodes[i].x = Double.valueOf( t.nextToken() ).doubleValue();
  nodes[i].y = Double.valueOf( t.nextToken() ).doubleValue();
  nodes[i].Px = Double.valueOf( t.nextToken() ).doubleValue();
  nodes[i].Py = Double.valueOf( t.nextToken() ).doubleValue();
}
```

```
for(i=1; i <= NB; i++) {
  s = b.readLine();
  t = new StringTokenizer(s);
  bars[i]  = new TrussMember();
  bars[i].NM = Integer.parseInt( t.nextToken() );
  bars[i].NP = Integer.parseInt( t.nextToken() );
  bars[i].area = Double.valueOf( t.nextToken() ).doubleValue();
  bars[i].E = Double.valueOf( t.nextToken() ).doubleValue();
}

b.close();
return true;

  } catch(EOFException e) {
System.out.println(“Error -- end of file reached”);
return false;
  } catch(IOException e) {
System.err.println(“Error during read file \n” + e.toString() );
// System.exit(1);
return false;
  }
}
return false;

}

}
//
// end of ‘Tr2d’ class definition
//
//////////////////////////////////////////////////////////////////



//////////////////////////////////////////////////////////////////
//
//The Tr2d class uses the ‘TrussNode’ class
//

class TrussNode {
int    n;                      /* node number */
 double x, y;                   /* coordinate */
double dx, dy;                 /* translation */
```

```
double Px, Py;                  /* loading */

        public TrussNode() {
        }
}

//
// end of 'TrussNode' class definition
//
/////////////////////////////////////////////////////////////////


/////////////////////////////////////////////////////////////////
//
//The Tr2d class uses the 'TrussMember' class
//

class TrussMember {
        int NP;                 /* positive (+) node */
        int NM;                 /* negative (-) node */
        double area;            /* bar area */
        double length;          /* bar length */
        double force;           /* bar force */
        double stress;          /* bar stress */
        double dl;              /* change in length */
        double UV[];            /* unit vector direction */
        double E;               /* Young's modulus */


        public TrussMember() {
// System.out.println("\tbegin TrussMember constructor");
        NP = 0;
        NM = 0;
                UV = new double[2];

        }
}
//
// end of 'TrussMember' class definition
//
/////////////////////////////////////////////////////////
```

```
File: Vif2.java
import java.awt.*;
import java.io.*;
import java.lang.String;
import java.lang.Integer;
import java.text.*; // DecimalFormat
import java.awt.event.*;
import java.awt.print.PrinterJob;
import java.awt.print.*;

/////////////////////////////////////////////////////////
//
// begin class Vif2
//

public class Vif2 extends Frame implements ActionListener, Compo-
nentListener,
KeyListener, Printable, WindowListener {

// These are the labels that will be used for the
// buttons in the ButtonStack objects to be created.
//
private String preProLabs[] = {"Input File Editor", "Constraint
File Editor",
"Optimization Settings", "Apply Random Loading", "Reset Default
Values"};
private String mainProLabs[] = {"Start Processor", "Stop Proces-
sor"};
private String postProLabs[] = {"Joint Displacements", "Member
Displacements",
"Member Forces", "All Results", "Plot History"};

private String inputErrorMsg, helpMsg;
private String baseName, currentFileName;
private int currentIteration = 0;

// Thread processorThread = new Thread(this, "SLP Iteration
Thread");
private Thread processorThread;

private MenuBar bar;
private Menu fileMenu, viewMenu, preProMenu, mainProMenu,
postProMenu, helpMenu;
```

```
private PageFormat pgFormat = new PageFormat();
private Paper pgPaper = new Paper();

// special non-java objects
private L3PCanvas drawingArea;
private Fr2d myObject;
private OptSetFrame optVars;
private DataPlotFramedataFrame;
private HelpFrame helpFrame;
private TextFileFrameinfileFrame;
private TextFileFrameconstraintFrame;
private TextFileFrameresultsFrame;

public static void main(String args[]) {
Vif2 f = new Vif2();

// only using the first arg as a filename for now
if(args.length > 0) {
  // System.out.println("found arg '" + args[0] + "'");
  f.readNewFile(args[0]);
} else {
  f.setDemo();
}
}

// Constructor
public Vif2() {
super("Structural Optimization Viewer - vif2 - 1.2.0");

int i;
int width  = 600;
int height = 600;

                // create menus
fileMenu   = new Menu("File");
viewMenu = new Menu("View");
preProMenu = new Menu("Pre-Processor");
mainProMenu   = new Menu("Processor");
postProMenu   = new Menu("Post-Processor");
helpMenu   = new Menu("Help");

                // add items to the menus
fileMenu.add( "Open File");
fileMenu.add( "Save File");
fileMenu.add( new MenuItem("-"));
fileMenu.add( "Print as Image");
fileMenu.add( "Print as Page");
fileMenu.add( new MenuItem("-"));
fileMenu.add( "Exit");

viewMenu.add("Node Numbers");
viewMenu.add("Member Numbers");
viewMenu.add( new MenuItem("-"));
viewMenu.add("Color Lines");
viewMenu.add("Thick Lines");
viewMenu.add("Show Loads");
viewMenu.add( new MenuItem("-"));
viewMenu.add("Invert Background");

for(i=0; i<preProLabs.length; i++) preProMenu.add(preProLabs[i]);
for(i=0; i<mainProLabs.length; i++) mainProMenu.add(mainPro-
Labs[i]);
for(i=0; i<postProLabs.length; i++) postProMenu.add(postPro-
Labs[i]);
postProMenu.add( new MenuItem("-"));
postProMenu.add("First Iteration");
postProMenu.add("Next Iteration");
postProMenu.add("Previous Iteration");
helpMenu.add("Help");

                // add menus to menu bar
bar = new MenuBar();
                bar.add(fileMenu);
bar.add(viewMenu);
bar.add(preProMenu);
bar.add(mainProMenu);
bar.add(postProMenu);
bar.add(helpMenu);
setMenuBar(bar);

dataFrame = new DataPlotFrame(400, 400);
drawingArea= new L3PCanvas(height-50, width, this);
infileFrame = new TextFileFrame(40, 70, "Input File Editor",
true);
```

```
constraintFrame=new TextFileFrame(40,70, "Constraint File Editor",
true);
resultsFrame = new TextFileFrame(40, 70, "Results Window", false);
helpFrame = new HelpFrame(40, 70);
optVars = new OptSetFrame();
setSize(width, height);

// things to listen to
fileMenu.addActionListener(this);
viewMenu.addActionListener(this);
preProMenu.addActionListener(this);
mainProMenu.addActionListener(this);
postProMenu.addActionListener(this);
helpMenu.addActionListener(this);

addComponentListener(this);
addKeyListener(this);
addWindowListener(this);

// Setup the form
add(drawingArea);

drawingArea.setBackground(drawingArea.backColor);
drawingArea.setCanRotate(false);
helpFrame.setSize (600,400);
constraintFrame.setSize (500,400);
infileFrame.setSize (500,500);
resultsFrame.setSize (600,500);
optVars.setSize (200,300);

initialize();
setVisible(true);
}

// required for ComponentListener
public void componentHidden(ComponentEvent e) {
// System.out.println("componentHidden event from "
//  + e.getComponent().getClass().getName());
}
public void componentMoved(ComponentEvent e) {
// System.out.println("componentMoved event");
}
public void componentResized(ComponentEvent e) {
// System.out.println("componentResized event");
// System.out.println("new width: " + this.getSize().width);
// System.out.println("new height: " + this.getSize().height);
drawingArea.resetSize(this.getSize().width, this.get-
Size().height-50 );
}
public void componentShown(ComponentEvent e) {
// System.out.println("componentShown event");
}

// required for KeyListener
public void keyPressed(KeyEvent e) {
}
public void keyReleased(KeyEvent e) {
}
public void keyTyped(KeyEvent e) {
// if(e.getKeyChar() == KeyEvent.VK_O) { System.out.println(" O
typed");}
if('m' == e.getKeyChar()) {
  drawingArea.showLineNumber = !drawingArea.showLineNumber;
  drawingArea.repaint();
} else if('n' == e.getKeyChar()) {
  drawingArea.showPointNumber = !drawingArea.showPointNumber;
  drawingArea.repaint();
} else if('a' == e.getKeyChar()) {
  fileMenu.dispatchEvent( new ActionEvent(fileMenu,
  ActionEvent.ACTION_PERFORMED, "All Results") );
} else if('c' == e.getKeyChar()) {
  fileMenu.dispatchEvent( new ActionEvent(fileMenu,
  ActionEvent.ACTION_PERFORMED, "Color Lines") );
} else if('d' == e.getKeyChar()) {
  fileMenu.dispatchEvent( new ActionEvent(fileMenu,
  ActionEvent.ACTION_PERFORMED, "Joint Displacements") );
} else if('D' == e.getKeyChar()) {
  fileMenu.dispatchEvent( new ActionEvent(fileMenu,
  ActionEvent.ACTION_PERFORMED, "Member Displacements") );
} else if('f' == e.getKeyChar()) {
  fileMenu.dispatchEvent( new ActionEvent(fileMenu,
  ActionEvent.ACTION_PERFORMED, "Member Forces") );
} else if('h' == e.getKeyChar()) {
  fileMenu.dispatchEvent( new ActionEvent(fileMenu,
  ActionEvent.ACTION_PERFORMED, "Plot History") );
} else if('i' == e.getKeyChar()) {
```

```
  fileMenu.dispatchEvent( new ActionEvent(fileMenu,
  ActionEvent.ACTION_PERFORMED, "Invert Background") );
} else if('j' == e.getKeyChar()) {
  fileMenu.dispatchEvent( new ActionEvent(fileMenu,
  ActionEvent.ACTION_PERFORMED, "Next Iteration") );
} else if('k' == e.getKeyChar()) {
  fileMenu.dispatchEvent( new ActionEvent(fileMenu,
  ActionEvent.ACTION_PERFORMED, "Previous Iteration") );
} else if('K' == e.getKeyChar()) {
  fileMenu.dispatchEvent( new ActionEvent(fileMenu,
  ActionEvent.ACTION_PERFORMED, "First Iteration") );
} else if('l' == e.getKeyChar()) {
  fileMenu.dispatchEvent( new ActionEvent(fileMenu,
  ActionEvent.ACTION_PERFORMED, "Show Loads") );
} else if('o' == e.getKeyChar()) {
  fileMenu.dispatchEvent( new ActionEvent(fileMenu,
  ActionEvent.ACTION_PERFORMED, "Open File") );
} else if('P' == e.getKeyChar()) {
  fileMenu.dispatchEvent( new ActionEvent(fileMenu,
  ActionEvent.ACTION_PERFORMED, "Print as Page") );
} else if('p' == e.getKeyChar()) {
  fileMenu.dispatchEvent( new ActionEvent(fileMenu,
  ActionEvent.ACTION_PERFORMED, "Print as Image") );
} else if('q' == e.getKeyChar()) {
  fileMenu.dispatchEvent( new ActionEvent(fileMenu,
  ActionEvent.ACTION_PERFORMED, "Exit") );
} else if('s' == e.getKeyChar()) {
  fileMenu.dispatchEvent( new ActionEvent(fileMenu,
  ActionEvent.ACTION_PERFORMED, "Save File") );
} else if('t' == e.getKeyChar()) {
  fileMenu.dispatchEvent( new ActionEvent(fileMenu,
  ActionEvent.ACTION_PERFORMED, "Thick Lines") );
} else if('v' == e.getKeyChar()) {
  SimpleMessage msg = new SimpleMessage("Volume: " +
  String.valueOf(myObject.volume() ), "Volume" );

}
// System.out.println("\tchar of key typed: '" + e.getKeyChar() +
"'");
// System.out.println("\tcode of key typed: '" + e.getKeyCode() +
"'");
}
```

```
// required for WindowListener
public void windowClosed(WindowEvent event) { }
public void windowDeiconified(WindowEvent event) { }
public void windowIconified(WindowEvent event) { }
public void windowActivated(WindowEvent event) { }
public void windowDeactivated(WindowEvent event) { }
public void windowOpened(WindowEvent event) { }
public void windowClosing(WindowEvent e) { System.exit(0); }

public void actionPerformed(ActionEvent e) {

// Menu Items
if (e.getSource() instanceof MenuItem) {
  String cmd = e.getActionCommand();
  // File menu items
  if (cmd == "Open File") {
MyFiles m = new MyFiles(this, FileDialog.LOAD);
if(m.fileName != "") {
// System.out.println("Reading " + m.fileName);
readNewFile(m.fileName);
}
  } else if (cmd == "Save File") {
MyFiles m = new MyFiles(this, FileDialog.SAVE);
if(m.fileName != "") {
System.out.println("Writing " + m.fileName);
writeFile(m.fileName);
}
  } else if (cmd == "Print as Page") {
  printImage(0);
  } else if (cmd == "Print as Image") {
  // printImage(1);
  SimpleMessage
  m = new SimpleMessage("'Print as Image' not enabled",
  "Function Not Enabled");

  } else if (cmd == "Exit") {
// setVisible(false);
// dispose();
System.exit(0);

  // View menu items
  } else if (cmd == "Node Numbers") {
```

```
drawingArea.showPointNumber = !drawingArea.showPointNumber;
  } else if (cmd == "Member Numbers") {
drawingArea.showLineNumber = !drawingArea.showLineNumber;
  } else if (cmd == "Color Lines") {
drawingArea.varyLineColors = !drawingArea.varyLineColors;
  } else if (cmd == "Thick Lines") {
drawingArea.varyLineThickness = !drawingArea.varyLineThickness;
  } else if (cmd == "Show Loads") {
drawingArea.showAuxLines = !drawingArea.showAuxLines;
  } else if (cmd == "Invert Background") {
drawingArea.invertBackground();

  // Pre - Processor menu items
  } else if (cmd == "Input File Editor") {
infileFrame.setVisible(true);
  } else if (cmd == "Constraint File Editor") {
constraintFrame.setVisible(true);
  } else if (cmd == "Optimization Settings") {
optVars.setVisible(true);
  } else if (cmd == "Reset Default Values") {
setDemo();

  // Main Processor menu items
  } else if (cmd == "Start Processor") {
myObject.analyze();

  // Post - Processor menu items
  } else if (cmd == "Joint Displacements") {
resultsFrame.setString( myObject.JointDisplacements() );
resultsFrame.setVisible(true);
  } else if (cmd == "Member Displacements") {
resultsFrame.setString( myObject.MemberDisplacements() );
resultsFrame.setVisible(true);
  } else if (cmd == "Member Forces") {
resultsFrame.setString( myObject.MemberForces() );
resultsFrame.setVisible(true);
  } else if (cmd == "All Results") {
String s = myObject.toString();
// s += "\nOptimization Variables";
// s += "\n\tSLP Iterations: \t" + optVars.totalIterMax;
s += "\n" + myObject.JointDisplacements();
s += "\n" + myObject.MemberDisplacements();
s += "\n" + myObject.MemberForces();
s += "\n" + "Volume: \t" + myObject.volume();
resultsFrame.setString( s );
resultsFrame.setVisible(true);
  } else if (cmd == "Plot History") {
dataFrame.setVisible(true);
  } else if (cmd == "First Iteration") {
changeModel(0);
  } else if (cmd == "Next Iteration") {
changeModel(1);
  } else if (cmd == "Previous Iteration") {
changeModel(-1);

  // Help menu
  } else if (cmd == "Help") {
helpFrame.setVisible(true);
  }
}
drawingArea.repaint();
}

public int printImage(int flag) {
// This local method will set the 'Paper' to simulate
// either full-page (8.5 x 11) or "image" which is
// defined here. For 'flag' = 0, full-page is used.

// pgPaper = new Paper();
if(flag == 1) {
  // System.out.println("printImage:: flag = 1");
  // setSize(double width, double height)
  double w = 5.0;
  double h = 5.0;
  double margin = 0.5;
  pgPaper.setSize( w*72.0, h*72.0);
  pgPaper.setImageableArea( margin*72.0, margin*72.0,
    (w-2*margin)*72.0, (h-2*margin)*72.0);
}

PrinterJob pj = PrinterJob.getPrinterJob();
// pgFormat.setOrientation(PageFormat.LANDSCAPE);
pgFormat.setPaper(pgPaper);
pgFormat = pj.validatePage(pgFormat);
// pj.setPrintable(this);
pj.setPrintable(this, pgFormat );
```

```
if (pj.printDialog()) {
  try {
    pj.print();
  } catch (Exception ex) {
    ex.printStackTrace();
  }
}
return 0;
}

    public int print(Graphics g, PageFormat pf, int pi) throws
PrinterException {
        if (pi >= 1) {
            return Printable.NO_SUCH_PAGE;
}

pf = pgFormat;
pf.setPaper(pgPaper);

// Shift and scale this object to fit on a page.
// System.out.println(" pf img " + pf.getImageableHeight()+" "
// +pf.getImageableWidth() );
// System.out.println(" pf w x h " + pf.getWidth()+" "+ pf.getH-
eight() );
// System.out.println(" paper w x h " + pgPaper.getWidth()+" "
// + pgPaper.getHeight() );
g.translate( (int)pf.getImageableX(), (int)pf.getImageableY() );

if(pf.getOrientation() == PageFormat.LANDSCAPE) {
  double r[] = new double[3];
  r = drawingArea.getRotation();
  r[2] -= 45.0;
  drawingArea.setRotation(r[0], r[1], r[2]);
  drawingArea.drawObject((Graphics2D) g, (int)pf.getImageable-
Width(),
    (int)pf.getImageableHeight() );
  r[2] += 45.0;
  drawingArea.setRotation(r[0], r[1], r[2]);
} else {
  drawingArea.drawObject((Graphics2D) g, (int)pf.getImageable-
Width(),
    (int)pf.getImageableHeight() );
}

        return Printable.PAGE_EXISTS;
    }

public void changeModel(int inc) {
// Get a list of files in this directory with the current base-
name.
// Locate the 'currentFileName' in this list and then read the
next
// one if it exists. This will allow for non-consecutive iteration
// numbers (e.g., if the analysis prints only every 5th itera-
tion).

boolean found;
DecimalFormat fmt = new DecimalFormat("000");
File f;
String fileList[];
int i;

// System.out.println("changeModel: current: '" + currentFileName
+ "'");
// System.out.println("changeModel: baseName: '" + baseName +
"'");
// System.out.println("changeModel: inc: '" + inc + "'");

if(currentIteration == 0 || inc == 0) {
  // System.out.println("starting from zero");
  if(readFile(baseName + ".001.osi") ) {
  currentFileName = baseName + ".001.osi";
  // myObject.analyze();
  updateDrawing();
  currentIteration = 1;
  dataFrame.X[0] = currentIteration;
  dataFrame.Y[0] = myObject.volume();
  dataFrame.nPoints = 1;
  return;
  }
}

f = new File(currentFileName);
f = new File(f.getAbsolutePath());
f = new File(f.getParent());
```

```
// if(f.exists()) System.out.println(" f.getParent() " + f.getPar-
ent() );
fileList = f.list(); // don't know how to create a filter
sortStringArray(fileList);

// Search the entire list for the 'currentFileName'. If there
exists
// a file in this family after (before) it, read this file.
//
for(i=0; i<fileList.length; i++) {
  // System.out.println("compare "+fileList[i]+" and "+current-
FileName);
  if(fileList[i].equals(currentFileName) ) {
    // Found the current file, now see if the increment (+/-) is
    // also an "osi" file with the same 'baseName' and is not the
    // original model itself ('getIteration' returns a zero).
    //
    // System.out.println("\tfound a match");
    if(fileList[i+inc].startsWith(baseName)
      && fileList[i+inc].endsWith(".osi")
      && getIteration(fileList[i+inc]) != 0) {
readFile(fileList[i+inc]);
currentFileName = fileList[i+inc];
updateDrawing();
currentIteration = getIteration(currentFileName);
// myObject.analyze();

// Update the history plot by scanning the entire array
// for existence of this iteration. If not found, increment
// the number of points 'nPoints' and add the y-value.
found = false;
for(i=0; i<dataFrame.nPoints; i++) {
  if(dataFrame.X[i] == currentIteration) {
    dataFrame.Y[i] = myObject.volume();
    found = true;
  }
}
if(!found) {
  dataFrame.X[dataFrame.nPoints] = currentIteration;
  dataFrame.Y[dataFrame.nPoints] = myObject.volume();
  dataFrame.nPoints++;
}
    } else {
SimpleMessage msg = new SimpleMessage("This is the "
 + (inc > 0 ? "last" : "first") + " iteration", "BaseName: "
 + baseName);
    }
    break;
  }
}
}

public void sortStringArray(String s[]) {
// This method will sort the string 's' and return it with the
// same number of elements in a lexicographic sorted order.
// It's hard to believe that there's no method provided in java
// to do this but I conldn't find one. Then again, it's not perl.
//
int i;
String temp;
boolean switched;

do {
 switched = false;
 for(i=0; i<s.length-1; i++) {
 if(s[i].compareTo(s[i+1]) > 0) {
   temp = s[i];
   s[i] = s[i+1];
   s[i+1] = temp;
   switched = true;
  }
 }
} while(switched);

// System.out.println("sortStringArray: sorted list");
// for(i=0; i<s.length; i++) System.out.println(" " + i+" "+s[i]);
}

public String removeExtension(String fullName) {
// Given a file name 'fullName' this method returns the
// equivalent 'baseName'. This is defined here as all
// characters before the last 4, which are assumed here
// to be the extension '.osi'. If the string doesn't end
// with this, simply return the original string

if(fullName.endsWith(".osi"))
```

```
  return fullName.substring(0, fullName.length()-4);
else
  return fullName;
}

public int getIteration(String fullName) {
// Given a file name 'fullName' this method returns the embedded
// iteration number. This number is defined here as the three
// characters after the 'baseName' but before the extension.

int start, end, num;
num = 0;

// System.out.println("getIteration: string: " + fullName);
start = baseName.length() + 1;
end = start + 3;
try {
  num = Integer.parseInt( fullName.substring(start, end) );
} catch(NumberFormatException e) {
  // System.out.println("getIteration: exception " + e);
  num = 0;
}
// System.out.println("getIteration: num: " + num);

return num;
}


public void readNewFile(String fileName) {
// Reset some values if the call to 'readFile' is successful

if(readFile(fileName)) {
  baseName = removeExtension(fileName);
  currentFileName = fileName;
  currentIteration = 0;
  dataFrame.nPoints = 0;
  updateDrawing();
}
}

public boolean readFile(String fileName) {// OPEN command
BufferedReader b = null;
String s = "", t="";
```

```
try {
b = new BufferedReader( new FileReader(fileName) );
} catch(IOException e) {
// System.err.println("Input file not opened properly \n"
//   + e.toString() );
SimpleMessage msg = new SimpleMessage("file open failed"
  + e, "File Not Opened");
b = null;
return false;
// System.exit(1);
}

// Read the file into a String 's' and (if successful) construct
// a new 'myObject' with this data.
// System.out.println("readFile(" + fileName + ")");
if(b != null) {
  try {
// System.out.println("Vif2::readFile begin with 'readByte'");
t = b.readLine();
while(t != null) {
  // s += (char)input.readByte();
  s += t + "\n";
  t = b.readLine();
}
b.close();

// System.out.println("Vif2::readFile s = '"+s+"'");
infileFrame.setString(s);
myObject = new Fr2d(s);
// myObject.analyze();
myObject.initialize();
return true;
  } catch(EOFException e) {
    System.err.println("Vif2::readFile EOFExcep-
tion\n"+e.toString() );
  } catch(IOException e) {
   System.err.println("Vif2::readFile IOException\n"+e.toString()
);
  }
}
return false;
}
```

```
public void writeFile(String fileName) {// SAVE command
DataOutputStream output = null;
try {// Open the file
output = new DataOutputStream ( new FileOutputStream(fileName) );
} catch(IOException e) {
System.err.println("File not opened properly \n" + e.toString() );
// System.exit(1);
}

if(output != null) {
  try {// Write the file
output.writeBytes( myObject.toString() );
output.close();
  } catch(IOException e) {
System.err.println("Error during write to file \n" + e.toString()
);
// System.exit(1);
  }
}
}

public void updateDrawing() {
int i, n1, n2;
double al;

for(i=1; i<= myObject.NN; i++) {
drawingArea.X[i] = myObject.nodes[i].x;
drawingArea.Y[i] = myObject.nodes[i].y;
}
for(i=1; i<= myObject.NB; i++) {
drawingArea.Point1[i] = myObject.bars[i].NM;
drawingArea.Point2[i] = myObject.bars[i].NP;
drawingArea.lineThickness[i] = myObject.bars[i].area;
}

                al = scaleLoadsToObject();
               // Set the auxillary lines (i.e., the applied loads)
// System.out.println("\n updateDrawing: n1   n2  nAux");
n2=myObject.NN;
drawingArea.nAux = 0;
                for(i=1; i <= myObject.freeNodes; i++) {
                    if( (myObject.nodes[i].Px != 0) || (myOb-
ject.nodes[i].Py != 0) ) {
                        n1 = i;
n2++;
drawingArea.nAux++;
                        drawingArea.auxP1[drawingArea.nAux] = n1;
                        drawingArea.auxP2[drawingArea.nAux] = n2;
                        drawingArea.X[n2] = drawingArea.X[n1] -
al*myObject.nodes[i].Px;
                        drawingArea.Y[n2] = drawingArea.Y[n1] -
al*myObject.nodes[i].Py;
// System.out.println(n1+" "+n2+" "+drawingArea.nAux);
                    }
                }

drawingArea.nPoints = myObject.NN;
drawingArea.nLines  = myObject.NB;
drawingArea.scaleLineThickness();
drawingArea.setTopLeftString(currentFileName);
drawingArea.refresh();
drawingArea.setRotation(180.0, 0.0, 0.0);
drawingArea.repaint();
}

        public double scaleLoadsToObject() {
               // Compute and return a scale factor, 'alpha' that
should be
                // used to multiply the loads by so that the max
load shows
                // up in a drawing as a percentage of the maximum
object dimension.
                int i;
                double alpha, ratio, maxRange, maxPrange;
                double max[] = new double[2];
                double min[] = new double[2];
                double Pmax[] = new double[2];
                double Pmin[] = new double[2];
                double range[] = new double[2];
                double Prange[] = new double[2];

                // Find the extreme values
                for(i=1; i<= myObject.NN; i++) {
```

```
                min[0] = myObject.nodes[i].x > min[0] ? min[0] :
myObject.nodes[i].x;
                min[1] = myObject.nodes[i].y > min[1] ? min[1] :
myObject.nodes[i].y;
                max[0] = myObject.nodes[i].x < max[0] ? max[0] :
myObject.nodes[i].x;
                max[1] = myObject.nodes[i].y < max[1] ? max[1] :
myObject.nodes[i].y;

                 Pmin[0]= myObject.nodes[i].Px>Pmin[0] ? Pmin[0]
: myObject.nodes[i].Px;
                 Pmin[1]= myObject.nodes[i].Py>Pmin[1] ? Pmin[1]
: myObject.nodes[i].Py;
                 Pmax[0]= myObject.nodes[i].Px<Pmax[0] ? Pmax[0]
: myObject.nodes[i].Px;
                 Pmax[1]= myObject.nodes[i].Py<Pmax[1] ? Pmax[1]
: myObject.nodes[i].Py;
               }
               maxRange = 0;
               maxPrange = 0;
               for(i=0; i<2; i++) {
                 range[i] = max[i] - min[i];
                 Prange[i] = Pmax[i] - Pmin[i];
                 maxRange = range[i] > maxRange ? range[i] :
maxRange;
                 maxPrange = Prange[i] > maxPrange ? Prange[i] :
maxPrange;
                 // System.out.println(i+" "+range[i]+"
"+Prange[i]);
               }
               // alpha = 1000000;
               alpha = maxPrange != 0 ? Math.abs(maxRange/max-
Prange) : 1.0;
              alpha *= 0.2;                    // 20% is done here
               return alpha;
        }


        public int createRandomLoad(double density) {
               int i, j;
               double MinVal, MaxVal, newMax;
               double Load[] = new double[2];
               String s;
               DecimalFormat fmt;

               MaxVal = 0.0;
               newMax = 0.0;

              // First identify the existing maximum value. This
may not be the
               // most efficient but that really doesn't matter
because this method
               // probably won't be called often and when it is,
it will be a 'low
              // - demand' time (i.e., nothing else is happening).
               for(i=1; i <= myObject.NN; i++) {
                if(Math.abs(myObject.nodes[i].Px) > MaxVal)
                       MaxVal = Math.abs(myObject.nodes[i].Px);
                if(Math.abs(myObject.nodes[i].Py) > MaxVal)
                       MaxVal = Math.abs(myObject.nodes[i].Py);
               }

               // MaxVal *= 1.5;
               MinVal = 0.1 * MaxVal;
               if(MaxVal < 10) fmt = new DecimalFormat("0.000");
               else fmt = new DecimalFormat("0.0");

// Gererate a random load and (not) apply it to a node.
               for(i=1; i <= myObject.freeNodes; i++) {
                for(j=0; j<2; j++) {
                Load[j] = MinVal + Math.random() * (MaxVal - Min-
Val);
                  if(Math.random() > 0.5) Load[j] *= -1.0;
                }
 myObject.nodes[i].Px = (Math.random() < density ? Load[0] : 0.0);
 myObject.nodes[i].Py = (Math.random() < density ? Load[1] : 0.0);
               }

// Ensure that at least one of the new values is equal in
// magnitude to the previous 'MaxVal'.
               for(i=1; i <= myObject.NN; i++) {
  if(Math.abs(myObject.nodes[i].Px) > newMax)
    newMax = Math.abs(myObject.nodes[i].Px);
  if(Math.abs(myObject.nodes[i].Py) > newMax)
    newMax = Math.abs(myObject.nodes[i].Py);
               }
```

```
                // System.out.println("randomLoad: maxval "+Max-
Val+", newmax "+newMax);
if(newMax == 0) {
  // System.out.println("found zero" );
  myObject.nodes[1].Px = MaxVal;
} else if(newMax != MaxVal) {
  // System.out.println("scaling by " + (MaxVal/newMax) );
                  for(i=1; i <= myObject.NN; i++) {
    myObject.nodes[i].Px *= MaxVal / newMax;
    myObject.nodes[i].Py *= MaxVal / newMax;
  }
}

// Reset all bar areas in the input file to 1.0 because one
                // optimization job may have been run by now and
the bar areas
                // will have different values.
                s = myObject.NB + "\t" + myObject.NN + "\t" + myO-
bject.NS + "\n";
                for(i=1; i <= myObject.NN; i++)
                 s += myObject.nodes[i].x
                   + "\t" + myObject.nodes[i].y
                   + "\t" + fmt.format(myObject.nodes[i].Px)
                   + "\t" + fmt.format(myObject.nodes[i].Py)
                   + "\n";
                for(i=1; i <= myObject.NB; i++)
                 s += myObject.bars[i].NM + "\t" + myOb-
ject.bars[i].NP + "\t 1.0 \t"
                   + myObject.bars[i].E + "\n";
                infileFrame.setString(s);
                myObject = new Fr2d(s);
                // myObject.analyze();

               // The loads have been changed so we need to update
the drawing.
                updateDrawing();

                return 0;
        }


public void initialize() {
   helpMsg = "Help::General"
   + "\n\tThis text area should define the object. The"
+ "\n\tfirst line should have  NB, NN, NS. "
+ "\n\tThe next block should have NN lines of nodal data. \n"
+ "\nHelp::Input"
+ "\n\tdata and the following line should have NRINGS "
+ "\n\tThe buttons at the right clear this frame (but "
+ "\n\tnot the data), read data from this frame, set \n"
+ "\nHelp::Output"
+ "\n\tYou can get the values"
+ "\n\tfor any given iteration by simply viewing that iteration
and"
+ "\n\tviewing the Input File Editor.\n"
+ "\nHelp::Print"
+ "\n\tBe sure to click on the Invert Colors checkbox before
printing"
+ "\n\tunless you really want the background to be black (which
uses "
+ "\n\t*lots* of toner). ";
   inputErrorMsg = "*** INPUT ERROR ***"
+ " \n There was a problem reading your input. This "
+ " \n may be due to not enough data or text (other"
+ " \n than e or E for exponential) in the data \n"
+ " \n You can delete these lines, make your correction"
+ " \n and try again. \n\n"
+ " \n A NumberFormatException was thrown with message:\n";
   helpFrame.setMessage(helpMsg);
}


public int charsToNextStr(String s, int i) {
// Beginning at character i in String s, return the number
// of characters to the next non-whitespace character.

int start = i;
while( (i < s.length()) && (
  (s.charAt(i)=='\t') || (s.charAt(i)=='\n') || (s.charAt(i)==' ')
) )
i++;

return i-start;
}
```

```
public void setDemo() {
String s;
// System.out.println(" setting ");

/*myObject   = new Fr2d(4, 5, 4);
double x[] = { 0.0, -10.0,  10.0,   3.0,  -3.0 };
double y[] = { 0.0, -10.0, -10.0, -10.0, -10.0 };
int NM[]   = {1, 1, 1, 1};
int NP[]   = {2, 3, 4, 5};
myObject.nodes[1].Px = 10.0;
myObject.nodes[1].Py = 5.0;
*/
myObject   = new Fr2d(11, 7, 2);
double x[] = { 2.5, 7.5, 12.5, 5.0, 10.0, 0.0, 15.0};
double y[] = { 4.0, 5.0,  4.0, 0.0, 0.0,  0.0, 0.0};
int NM[]   = {1,2,1, 1, 4, 2, 5, 3,   6, 4, 5};
int NP[]   = {2,3,6, 4, 2, 5, 3, 7,   4, 5, 7};

                for(int i = 1; i <= myObject.NN; i++) {
                  myObject.nodes[i].x = x[i-1];
                  myObject.nodes[i].y = y[i-1];
                  myObject.nodes[i].Px = 0;
                  myObject.nodes[i].Py = 0;
                  myObject.nodes[i].Mz = 0;
                }
myObject.nodes[1].Py = -1000.0;
myObject.nodes[2].Py = -1000.0;

                for(int i = 1; i <= myObject.NB; i++) {
                  myObject.bars[i].NM = NM[i-1];
                  myObject.bars[i].NP = NP[i-1];
                  myObject.bars[i].area = 1.0;
                  myObject.bars[i].section_I = 20.0;
                  myObject.bars[i].E = 29000;
                }
infileFrame.setString(myObject.toString() );

s = "% The constraints file can have comments on a \n% line that"
  + " begins with a percent sign\n%\n\n";
s += "*CONSTRAINTS \t " + (2*myObject.NB)+ "\n";
for(int i=1; i<=myObject.NB; i++)
  s += "dl \t" + i + "\tUPPER \t 0.1 \ndl \t" + i + "\tLOWER \t -
0.1\n";
s += "\n\n*OBJECTIVE \t "+myObject.NB+"\n";
for(int i=1; i<=myObject.NB; i++)
  s += "obj \t" + i + "\tVOLUME \t 1.0 \n";
constraintFrame.setString(s);
updateDrawing();
baseName = "";
currentFileName = "";
}

}

//
// end class Vif2
//
////////////////////////////////////////////////////////////////
```

```
File: Vif3.java
import java.awt.*;
import java.io.*;
import java.lang.String;
import java.lang.Integer;
import java.text.*; // DecimalFormat
import java.awt.event.*;
import java.awt.print.PrinterJob;
import java.awt.print.*;

//////////////////////////////////////////////////////////
//
// begin class Vif3
//

public class Vif3 extends Frame implements ActionListener, Compo-
nentListener,
KeyListener, Printable, WindowListener {

// These are the labels that will be used for the
// buttons in the ButtonStack objects to be created.
//
private String preProLabs[] = {"Input File Editor", "Constraint
File Editor",
"Optimization Settings", "Reset Default Values"};
private String mainProLabs[] = {"Start Processor", "Stop Proces-
sor"};
private String postProLabs[] = {"Joint Displacements", "Member
Displacements",
"Member Forces", "All Results", "Plot History"};

private String inputErrorMsg, helpMsg;
private String baseName, currentFileName;
private int currentIteration = 0;
private boolean DEBUG = false;

private MenuBar bar;
private Menu fileMenu, viewMenu, preProMenu, mainProMenu,
postProMenu, helpMenu;
private PageFormat pgFormat = new PageFormat();
private Paper pgPaper = new Paper();

// special non-java objects
private L3PCanvas drawingArea;
private Fr3d myObject;
private OptSetFrame optVars;
private DataPlotFramedataFrame;
private HelpFrame helpFrame;
private TextFileFrameinfileFrame;
private TextFileFrameconstraintFrame;
private TextFileFrameresultsFrame;

public static void main(String args[]) {
Vif3 f = new Vif3();

// only using the first arg as a filename for now
if(args.length > 0) {
  // System.out.println("found arg '" + args[0] + "'");
  f.readNewFile(args[0]);
} else {
  f.setDemo();
}
}

// Constructor
public Vif3() {
super("Structural Optimization Viewer vif3 1.2.0");

int i;
int width  = 600;
int height = 600;

                // create menus
fileMenu   = new Menu("File");
viewMenu = new Menu("View");
preProMenu = new Menu("Pre-Processor");
mainProMenu   = new Menu("Processor");
postProMenu   = new Menu("Post-Processor");
helpMenu   = new Menu("Help");

                // add items to the menus
fileMenu.add( "Open File");
fileMenu.add( "Save File");
fileMenu.add( new MenuItem("-"));
fileMenu.add( "Print as Image");
fileMenu.add( "Print as Page");
```

```
fileMenu.add( new MenuItem("-"));
fileMenu.add( "Exit");

viewMenu.add("Node Numbers");
viewMenu.add("Member Numbers");
viewMenu.add( new MenuItem("-"));
viewMenu.add("Color Lines");
viewMenu.add("Thick Lines");
viewMenu.add("Show Loads");
viewMenu.add( new MenuItem("-"));
viewMenu.add("Invert Background");

for(i=0; i<preProLabs.length; i++) preProMenu.add(preProLabs[i]);
for(i=0; i<mainProLabs.length; i++) mainProMenu.add(mainPro-
Labs[i]);
for(i=0; i<postProLabs.length; i++) postProMenu.add(postPro-
Labs[i]);
postProMenu.add( new MenuItem("-"));
postProMenu.add("First Iteration");
postProMenu.add("Next Iteration");
postProMenu.add("Previous Iteration");
helpMenu.add("Help");

                // add menus to menu bar
bar = new MenuBar();
                bar.add(fileMenu);
bar.add(viewMenu);
bar.add(preProMenu);
bar.add(mainProMenu);
bar.add(postProMenu);
bar.add(helpMenu);
setMenuBar(bar);

dataFrame = new DataPlotFrame(400, 400);
drawingArea= new L3PCanvas(height-50, width, this);
infileFrame = new TextFileFrame(40, 70, "Input File Editor",
true);
constraintFrame=new TextFileFrame(40,70, "Constraint File Editor",
true);
resultsFrame = new TextFileFrame(40, 70, "Results Window", false);
helpFrame = new HelpFrame(40, 70);
optVars = new OptSetFrame();
setSize(width, height);

// things to listen to
fileMenu.addActionListener(this);
viewMenu.addActionListener(this);
preProMenu.addActionListener(this);
mainProMenu.addActionListener(this);
postProMenu.addActionListener(this);
helpMenu.addActionListener(this);

addComponentListener(this);
addKeyListener(this);
addWindowListener(this);

// Setup the form
add(drawingArea);

drawingArea.setBackground(drawingArea.backColor);
drawingArea.setCanRotate(true);
helpFrame.setSize (600,400);
constraintFrame.setSize (500,400);
infileFrame.setSize (500,500);
resultsFrame.setSize (600,500);
optVars.setSize (200,300);

initialize();
setVisible(true);
}

// required for ComponentListener
public void componentHidden(ComponentEvent e) { }
public void componentMoved(ComponentEvent e) { }
public void componentResized(ComponentEvent e) {
drawingArea.resetSize(this.getWidth(), this.getHeight()-50 );
}
public void componentShown(ComponentEvent e) {
// System.out.println("componentShown event");
}


// required for KeyListener
public void keyPressed(KeyEvent e) { }
public void keyReleased(KeyEvent e) { }
public void keyTyped(KeyEvent e) {
```

```
// if(e.getKeyChar() == KeyEvent.VK_O) { System.out.println(" O
typed");}
if('m' == e.getKeyChar()) {
  drawingArea.showLineNumber = !drawingArea.showLineNumber;
  drawingArea.repaint();
} else if('n' == e.getKeyChar()) {
  drawingArea.showPointNumber = !drawingArea.showPointNumber;
  drawingArea.repaint();
} else if('a' == e.getKeyChar()) {
  fileMenu.dispatchEvent( new ActionEvent(fileMenu,
  ActionEvent.ACTION_PERFORMED, "All Results") );
} else if('c' == e.getKeyChar()) {
  fileMenu.dispatchEvent( new ActionEvent(fileMenu,
  ActionEvent.ACTION_PERFORMED, "Color Lines") );
} else if('d' == e.getKeyChar()) {
  fileMenu.dispatchEvent( new ActionEvent(fileMenu,
  ActionEvent.ACTION_PERFORMED, "Joint Displacements") );
} else if('D' == e.getKeyChar()) {
  fileMenu.dispatchEvent( new ActionEvent(fileMenu,
  ActionEvent.ACTION_PERFORMED, "Member Displacements") );
} else if('f' == e.getKeyChar()) {
  fileMenu.dispatchEvent( new ActionEvent(fileMenu,
  ActionEvent.ACTION_PERFORMED, "Member Forces") );
} else if('g' == e.getKeyChar()) {
  DEBUG = !DEBUG;
  System.out.println("DEBUG is " + DEBUG);
} else if('h' == e.getKeyChar()) {
  fileMenu.dispatchEvent( new ActionEvent(fileMenu,
  ActionEvent.ACTION_PERFORMED, "Plot History") );
} else if('i' == e.getKeyChar()) {
  fileMenu.dispatchEvent( new ActionEvent(fileMenu,
  ActionEvent.ACTION_PERFORMED, "Invert Background") );
} else if('j' == e.getKeyChar()) {
  fileMenu.dispatchEvent( new ActionEvent(fileMenu,
  ActionEvent.ACTION_PERFORMED, "Next Iteration") );
} else if('k' == e.getKeyChar()) {
  fileMenu.dispatchEvent( new ActionEvent(fileMenu,
  ActionEvent.ACTION_PERFORMED, "Previous Iteration") );
} else if('K' == e.getKeyChar()) {
  fileMenu.dispatchEvent( new ActionEvent(fileMenu,
  ActionEvent.ACTION_PERFORMED, "First Iteration") );
} else if('l' == e.getKeyChar()) {
  fileMenu.dispatchEvent( new ActionEvent(fileMenu,
  ActionEvent.ACTION_PERFORMED, "Show Loads") );
} else if('o' == e.getKeyChar()) {
  fileMenu.dispatchEvent( new ActionEvent(fileMenu,
  ActionEvent.ACTION_PERFORMED, "Open File") );
} else if('P' == e.getKeyChar()) {
  fileMenu.dispatchEvent( new ActionEvent(fileMenu,
  ActionEvent.ACTION_PERFORMED, "Print as Page") );
} else if('p' == e.getKeyChar()) {
  fileMenu.dispatchEvent( new ActionEvent(fileMenu,
  ActionEvent.ACTION_PERFORMED, "Print as Image") );
} else if('q' == e.getKeyChar()) {
  fileMenu.dispatchEvent( new ActionEvent(fileMenu,
  ActionEvent.ACTION_PERFORMED, "Exit") );
} else if('s' == e.getKeyChar()) {
  fileMenu.dispatchEvent( new ActionEvent(fileMenu,
  ActionEvent.ACTION_PERFORMED, "Save File") );
} else if('t' == e.getKeyChar()) {
  fileMenu.dispatchEvent( new ActionEvent(fileMenu,
  ActionEvent.ACTION_PERFORMED, "Thick Lines") );
} else if('v' == e.getKeyChar()) {
  SimpleMessage msg = new SimpleMessage("Volume: " +
  String.valueOf(myObject.volume() ), "Volume" );
}
}


// required for WindowListener
public void windowClosed(WindowEvent event) { }
public void windowDeiconified(WindowEvent event) { }
public void windowIconified(WindowEvent event) { }
public void windowActivated(WindowEvent event) { }
public void windowDeactivated(WindowEvent event) { }
public void windowOpened(WindowEvent event) { }
public void windowClosing(WindowEvent e) { System.exit(0); }

public void actionPerformed(ActionEvent e) {

// Menu Items
if (e.getSource() instanceof MenuItem) {
  String cmd = e.getActionCommand();
  // File menu items
  if (cmd == "Open File") {
MyFiles m = new MyFiles(this, FileDialog.LOAD);
```

```
if(m.fileName != "") {
// System.out.println("Reading " + m.fileName);
readNewFile(m.fileName);
}
  } else if (cmd == "Save File") {
MyFiles m = new MyFiles(this, FileDialog.SAVE);
if(m.fileName != "") {
System.out.println("Writing " + m.fileName);
writeFile(m.fileName);
}
  } else if (cmd == "Print as Page") {
  printImage(0);
  } else if (cmd == "Print as Image") {
  // printImage(1);
  SimpleMessage
  m = new SimpleMessage("'Print as Image' doesn't work",
  "Function Not Enabled");

  } else if (cmd == "Exit") {
// setVisible(false);
// dispose();
System.exit(0);

  // View menu items
  } else if (cmd == "Node Numbers") {
drawingArea.showPointNumber = !drawingArea.showPointNumber;
  } else if (cmd == "Member Numbers") {
drawingArea.showLineNumber = !drawingArea.showLineNumber;
  } else if (cmd == "Color Lines") {
drawingArea.varyLineColors = !drawingArea.varyLineColors;
  } else if (cmd == "Thick Lines") {
drawingArea.varyLineThickness = !drawingArea.varyLineThickness;
  } else if (cmd == "Show Loads") {
drawingArea.showAuxLines = !drawingArea.showAuxLines;
  } else if (cmd == "Invert Background") {
drawingArea.invertBackground();

  // Pre - Processor menu items
  } else if (cmd == "Input File Editor") {
infileFrame.setVisible(true);
  } else if (cmd == "Constraint File Editor") {
constraintFrame.setVisible(true);
  } else if (cmd == "Optimization Settings") {
optVars.setVisible(true);
  } else if (cmd == "Reset Default Values") {
setDemo();

  // Main Processor menu items
  } else if (cmd == "Start Processor") {
myObject.analyze();

  // Post - Processor menu items
  } else if (cmd == "Joint Displacements") {
resultsFrame.setString( myObject.JointDisplacements() );
resultsFrame.setVisible(true);
  } else if (cmd == "Member Displacements") {
resultsFrame.setString( myObject.MemberDisplacements() );
resultsFrame.setVisible(true);
  } else if (cmd == "Member Forces") {
resultsFrame.setString( myObject.MemberForces() );
resultsFrame.setVisible(true);
  } else if (cmd == "All Results") {
String s = myObject.toString();
// s += "\nOptimization Variables";
// s += "\n\tSLP Iterations: \t" + optVars.totalIterMax;
s += "\n" + myObject.JointDisplacements();
s += "\n" + myObject.MemberDisplacements();
s += "\n" + myObject.MemberForces();
s += "\n" + "Volume: \t" + myObject.volume();
resultsFrame.setString( s );
resultsFrame.setVisible(true);
  } else if (cmd == "Plot History") {
dataFrame.setVisible(true);
  } else if (cmd == "First Iteration") {
changeModel(0);
  } else if (cmd == "Next Iteration") {
changeModel(1);
  } else if (cmd == "Previous Iteration") {
changeModel(-1);

  // Help menu
  } else if (cmd == "Help") {
helpFrame.setVisible(true);
  }
}
drawingArea.repaint();
```

```
}


public int printImage(int flag) {
// This local method will set the 'Paper' to simulate
// either full-page (8.5 x 11) or "image" which is
// defined here. For 'flag' = 0, full-page is used.

// pgPaper = new Paper();
if(flag == 1) {
  // System.out.println("printImage:: flag = 1");
  // setSize(double width, double height)
  double w = 5.0;
  double h = 5.0;
  double margin = 0.5;
  pgPaper.setSize( w*72.0, h*72.0);
  pgPaper.setImageableArea( margin*72.0, margin*72.0,
    (w-2*margin)*72.0, (h-2*margin)*72.0);
}

PrinterJob pj = PrinterJob.getPrinterJob();
// pgFormat.setOrientation(PageFormat.LANDSCAPE);
pgFormat.setPaper(pgPaper);
pgFormat = pj.validatePage(pgFormat);
// pj.setPrintable(this);
pj.setPrintable(this, pgFormat );

if (pj.printDialog()) {
  try {
    pj.print();
  } catch (Exception ex) {
    ex.printStackTrace();
  }
}
return 0;
}

    public int print(Graphics g, PageFormat pf, int pi) throws
PrinterException {
        if (pi >= 1) {
            return Printable.NO_SUCH_PAGE;
}
pf = pgFormat;
pf.setPaper(pgPaper);

// Shift and scale this object to fit on a page.
// System.out.println(" pf img " + pf.getImageableHeight()+" "
// +pf.getImageableWidth() );
// System.out.println(" pf w x h " + pf.getWidth()+" "+ pf.getH-
eight() );
// System.out.println(" paper w x h " + pgPaper.getWidth()+" "
// + pgPaper.getHeight() );
g.translate( (int)pf.getImageableX(), (int)pf.getImageableY() );

if(pf.getOrientation() == PageFormat.LANDSCAPE) {
  double r[] = new double[3];
  r = drawingArea.getRotation();
  r[2] -= 45.0;
  drawingArea.setRotation(r[0], r[1], r[2]);
  drawingArea.drawObject((Graphics2D) g, (int)pf.getImageable-
Width(),
    (int)pf.getImageableHeight() );
  r[2] += 45.0;
  drawingArea.setRotation(r[0], r[1], r[2]);
} else {
  drawingArea.drawObject((Graphics2D) g, (int)pf.getImageable-
Width(),
    (int)pf.getImageableHeight() );
}
        return Printable.PAGE_EXISTS;
    }

public void changeModel(int inc) {
// Get a list of files in this directory with the current base-
name.
// Locate the 'currentFileName' in this list and then read the
next
// one if it exists. This will allow for non-consecutive iteration
// numbers (e.g., if the analysis prints only every 5th itera-
tion).

boolean found;
DecimalFormat fmt = new DecimalFormat("000");
File f;
String fileList[];
```

```
int i;

// System.out.println("changeModel: current: '" + currentFileName
+ "'");
// System.out.println("changeModel: baseName: '" + baseName +
"'");
// System.out.println("changeModel: inc: '" + inc + "'");

if(inc == 0) currentIteration = 0;
if(currentIteration == 0) {
  // System.out.println("starting from zero");
  if(readFile(baseName + ".001.osi") ) {
    currentFileName = baseName + ".001.osi";
    updateDrawing();
    // myObject.analyze();
    myObject.initialize();
    currentIteration = 1;
    dataFrame.X[0] = currentIteration;
    dataFrame.Y[0] = myObject.volume();
    if(dataFrame.nPoints == 0) dataFrame.nPoints = 1;
  } else {
    SimpleMessage msg = new SimpleMessage("Not found "+base-
Name+".001.osi",
      "BaseName: " + baseName);
  }
  return;
}

f = new File(currentFileName);
f = new File(f.getAbsolutePath());
f = new File(f.getParent());
// if(f.exists()) System.out.println(" f.getParent() " + f.getPar-
ent() );
fileList = f.list(); // don't know how to create a filter
sortStringArray(fileList);

// Search the entire list for the 'currentFileName'. If there
exists
// a file in this family after (before) it, read this file.
//
for(i=0; i<fileList.length; i++) {
  // System.out.println("compare "+fileList[i]+" and "+current-
FileName);
  if(fileList[i].equals(currentFileName) ) {
    // Found the current file, now see if the increment (+/-) is
    // also an "osi" file with the same 'baseName' and is not the
    // original model itself ('getIteration' returns a zero).
    //
    // System.out.println("\tfound a match");
    if(fileList[i+inc].startsWith(baseName)
      && fileList[i+inc].endsWith(".osi")
      && getIteration(fileList[i+inc]) != 0) {
readFile(fileList[i+inc]);
currentFileName = fileList[i+inc];
updateDrawing();
currentIteration = getIteration(currentFileName);
// myObject.analyze();
myObject.initialize();

// Update the history plot by scanning the entire array
// for existence of this iteration. If not found, increment
// the number of points 'nPoints' and add the y-value.
found = false;
for(i=0; i<dataFrame.nPoints; i++) {
  if(dataFrame.X[i] == currentIteration) {
    dataFrame.Y[i] = myObject.volume();
    found = true;
  }
}
if(!found) {
  dataFrame.X[dataFrame.nPoints] = currentIteration;
  dataFrame.Y[dataFrame.nPoints] = myObject.volume();
  dataFrame.nPoints++;
}
    } else {
SimpleMessage msg = new SimpleMessage("This is the "
 + (inc > 0 ? "last" : "first") + " iteration", "BaseName: "
 + baseName);
    }
    break;
  }
}
}


public void sortStringArray(String s[]) {
```

```
// This method will sort the string 's' and return it with the
// same number of elements in a lexicographic sorted order.
// It's hard to believe that there's no method provided in java
// to do this but I conldn't find one. Then again, it's not perl.
//
int i;
String temp;
boolean switched;

do {
 switched = false;
 for(i=0; i<s.length-1; i++) {
 if(s[i].compareTo(s[i+1]) > 0) {
   temp = s[i];
   s[i] = s[i+1];
   s[i+1] = temp;
   switched = true;
  }
 }
} while(switched);

// System.out.println("sortStringArray: sorted list");
// for(i=0; i<s.length; i++) System.out.println(" " + i+" "+s[i]);
}

public String removeExtension(String fullName) {
// Given a file name 'fullName' this method returns the
// equivalent 'baseName'. This is defined here as all
// characters before the last 4, which are assumed here
// to be the extension '.osi'. If the string doesn't end
// with this, simply return the original string

if(fullName.endsWith(".osi"))
  return fullName.substring(0, fullName.length()-4);
else
  return fullName;
}


public int getIteration(String fullName) {
// Given a file name 'fullName' this method returns the embedded
// iteration number. This number is defined here as the three
// characters after the 'baseName' but before the extension.

int start, end, num;
num = 0;

// System.out.println("getIteration: string: " + fullName);
start = baseName.length() + 1;
end = start + 3;
try {
  num = Integer.parseInt( fullName.substring(start, end) );
} catch(NumberFormatException e) {
  // System.out.println("getIteration: exception " + e);
  num = 0;
}
// System.out.println("getIteration: num: " + num);

return num;
}

public void readNewFile(String fileName) {
// Reset some values if the call to 'readFile' is successful

if(readFile(fileName)) {
  baseName = removeExtension(fileName);
  currentFileName = fileName;
  currentIteration = 0;
  dataFrame.nPoints = 0;
  updateDrawing();
  drawingArea.setRotation(180.0, 0.0, 0.0);
  // System.out.println("readNewFile: zero rotation");
}
}

public boolean readFile(String fileName) {// OPEN command
BufferedReader b = null;
String s = "", t="";

try {
b = new BufferedReader( new FileReader(fileName) );
} catch(IOException e) {
// System.err.println("Input file not opened properly \n"
//   + e.toString() );
SimpleMessage msg = new SimpleMessage("file open failed"
  + e, "File Not Opened");
```

```
b = null;
return false;
// System.exit(1);
}

// Read the file into a String 's' and (if successful) construct
// a new 'myObject' with this data.
// System.out.println("readFile(" + fileName + ")");
if(b != null) {
  try {
// System.out.println("Vif3::readFile begin with 'readByte'");
t = b.readLine();
while(t != null) {
  // s += (char)input.readByte();
  s += t + "\n";
  t = b.readLine();
}
b.close();

// System.out.println("Vif3::readFile s = '"+s+"'");
infileFrame.setString(s);
myObject = new Fr3d(s);
// myObject.analyze();
myObject.initialize();
return true;
  } catch(EOFException e) {
    System.err.println("Vif3::readFile EOFExcep-
tion\n"+e.toString() );
  } catch(IOException e) {
   System.err.println("Vif3::readFile IOException\n"+e.toString()
);
  }
}
return false;
}

public void writeFile(String fileName) {// SAVE command
DataOutputStream output = null;
try {// Open the file
output = new DataOutputStream( new FileOutputStream(fileName) );
} catch(IOException e) {
System.err.println("File not opened \n" + e.toString() );
// System.exit(1);
}

if(output != null) {
  try {// Write the file
output.writeBytes( myObject.toString() );
output.close();
  } catch(IOException e) {
System.err.println("Error during writeFile \n" + e.toString() );
// System.exit(1);
  }
}
}

public void updateDrawing() {
int i, n1, n2;
double al;
String s;

for(i=1; i<= myObject.NN; i++) {
drawingArea.X[i] = myObject.nodes[i].x;
drawingArea.Y[i] = myObject.nodes[i].y;
drawingArea.Z[i] = myObject.nodes[i].z;
}
for(i=1; i<= myObject.NB; i++) {
drawingArea.Point1[i] = myObject.bars[i].NM;
drawingArea.Point2[i] = myObject.bars[i].NP;
drawingArea.lineThickness[i] = myObject.bars[i].area;
}

                al = scaleLoadsToObject();
               // Set the auxillary lines (i.e., the applied loads)
// System.out.println("\n updateDrawing: n1   n2  nAux");
n2=myObject.NN;
drawingArea.nAux = 0;
                for(i=1; i <= myObject.freeNodes; i++) {
                  if( (myObject.nodes[i].Px != 0) || (myOb-
ject.nodes[i].Py != 0)
                    || (myObject.nodes[i].Pz != 0) ) {
                        n1 = i;
n2++;
drawingArea.nAux++;
                        drawingArea.auxP1[drawingArea.nAux] = n1;
                        drawingArea.auxP2[drawingArea.nAux] = n2;
```

```
                        drawingArea.X[n2] = drawingArea.X[n1] -
al*myObject.nodes[i].Px;
                        drawingArea.Y[n2] = drawingArea.Y[n1] -
al*myObject.nodes[i].Py;
drawingArea.Z[n2] = drawingArea.Z[n1] - al*myObject.nodes[i].Pz;
// System.out.println(n1+" "+n2+" "+drawingArea.nAux);
                }
            }

drawingArea.nPoints = myObject.NN;
drawingArea.nLines  = myObject.NB;
drawingArea.setTopLeftString("File: " + currentFileName +" vol: "
 + MyMath.simpleFormat(myObject.volume()) );
drawingArea.scaleLineThickness();
drawingArea.refresh();
drawingArea.repaint();
}


        public double scaleLoadsToObject() {
            // Compute and return a scale factor, 'alpha' that
should be
            // used to multiply the loads by so that the max
load shows
            // up in a drawing as a percentage of the maximum
object dimension.
            int i;
            double alpha, ratio, maxRange, maxPrange;
            double max[] = new double[2];
            double min[] = new double[2];
            double Pmax[] = new double[2];
            double Pmin[] = new double[2];
            double range[] = new double[2];
            double Prange[] = new double[2];

            // Find the extreme values
            for(i=1; i<= myObject.NN; i++) {
             min[0] = myObject.nodes[i].x > min[0] ? min[0] :
myObject.nodes[i].x;
             min[1] = myObject.nodes[i].y > min[1] ? min[1] :
myObject.nodes[i].y;
             max[0] = myObject.nodes[i].x < max[0] ? max[0] :
myObject.nodes[i].x;
             max[1] = myObject.nodes[i].y < max[1] ? max[1] :
myObject.nodes[i].y;

             Pmin[0]= myObject.nodes[i].Px>Pmin[0] ? Pmin[0]
: myObject.nodes[i].Px;
             Pmin[1]= myObject.nodes[i].Py>Pmin[1] ? Pmin[1]
: myObject.nodes[i].Py;
             Pmax[0]= myObject.nodes[i].Px<Pmax[0] ? Pmax[0]
: myObject.nodes[i].Px;
             Pmax[1]= myObject.nodes[i].Py<Pmax[1] ? Pmax[1]
: myObject.nodes[i].Py;
            }
            maxRange = 0;
            maxPrange = 0;
            for(i=0; i<2; i++) {
              range[i] = max[i] - min[i];
              Prange[i] = Pmax[i] - Pmin[i];
              maxRange = range[i] > maxRange ? range[i] :
maxRange;
              maxPrange = Prange[i] > maxPrange ? Prange[i] :
maxPrange;
              // System.out.println(i+" "+range[i]+"
"+Prange[i]);
            }
            // alpha = 1000000;
            alpha = maxPrange != 0 ? Math.abs(maxRange/max-
Prange) : 1.0;
            alpha *= 0.2;                    // 20% is done here
            return alpha;
        }

public void initialize() {
   helpMsg = "Help::General"
   + "\n\tThis text area should define the object. The"
+ "\n\tfirst line should have  NB, NN, NS. "
+ "\n\tThe next block should have NN lines of nodal data. \n"
+ "\nHelp::mnemonic"
+ "\n\tThe following 'hot-keys' are recognized"
+ "\n\t a \t all results"
+ "\n\t c \t toggle line color variation"
+ "\n\t d \t joint displacements"
+ "\n\t D \t member displacements"
+ "\n\t f \t member forces"
```

```
+ "\n\t h \t plot history"
+ "\n\t i \t invert background"
+ "\n\t j \t next iteration"
+ "\n\t k \t previous iteration"
+ "\n\t K \t first iteration"
+ "\n\t m \t member numbers"
+ "\n\t n \t node numbers"
+ "\n\t o \t open"
+ "\n\t q \t quit"
+ "\n\t s \t save"
+ "\n\t t \t toggle line thickness variation"
+ "\n\t v \t volume"
+ "\nHelp::Input"
+ "\n\tThe buttons at the right clear this frame (but "
+ "\n\tnot the data), read data from this frame, set \n"
+ "\nHelp::Output"
+ "\n\tYou can get the values"
+ "\n\tfor any given iteration by simply viewing that iteration
and"
+ "\n\tviewing the Input File Editor.\n"
+ "\nHelp::Print"
+ "\n\tBe sure to click on the Invert Colors checkbox before
printing"
+ "\n\tunless you really want the background to be black (which
uses "
+ "\n\t*lots* of toner). ";
   inputErrorMsg = "*** INPUT ERROR ***"
+ " \n There was a problem reading your input. This "
+ " \n may be due to not enough data or text (other"
+ " \n than e or E for exponential) in the data \n"
+ " \n You can delete these lines, make your correction"
+ " \n and try again. \n\n"
+ " \n A NumberFormatException was thrown with message:\n";
   helpFrame.setMessage(helpMsg);
}

public int charsToNextStr(String s, int i) {
// Beginning at character i in String s, return the number
// of characters to the next non-whitespace character.

int start = i;
while( (i < s.length()) && (
  (s.charAt(i)=='\t') || (s.charAt(i)=='\n') || (s.charAt(i)==' ')
) )
i++;

return i-start;
}

public void setDemo() {
String s;
// System.out.println(" setting ");

myObject   = new Fr3d(8, 8, 4);
double x[] = {0.0,   200.0, 200.0,   0.0,   0.0, 200.0, 200.0,
0.0};
double y[] = {100.0, 100.0, 100.0, 100.0,   0.0,   0.0,   0.0,
0.0};
double z[] = {0.0,     0.0, 150.0, 150.0,   0.0,   0.0, 150.0,
150.0};
double tx[] = { 0, 0,   0,   0,        0,   0,   0,   0};
double ty[] = { 0, 0, -90, -90,        0,   0,   0,   0};
double tz[] = { 0, 0,   0,   0,      -90, -90, -90, -90};
int NM[]   = {1,4,1,2, 1,4,2,3};
int NP[]   = {2,3,4,3, 5,8,6,7};

                for(int i = 1; i <= myObject.NN; i++) {
                  myObject.nodes[i].x = x[i-1];
                  myObject.nodes[i].y = y[i-1];
                  myObject.nodes[i].z = z[i-1];
                  myObject.nodes[i].Px = 0;
                  myObject.nodes[i].Py = 0;
                  myObject.nodes[i].Pz = 0;
                  myObject.nodes[i].Mx = 0;
                  myObject.nodes[i].My = 0;
                  myObject.nodes[i].Mz = 0;
                }
myObject.nodes[1].Py = -1000.0;
myObject.nodes[2].Py = -1000.0;

                for(int i = 1; i <= myObject.NB; i++) {
                  myObject.bars[i].NM = NM[i-1];
                  myObject.bars[i].NP = NP[i-1];
                  myObject.bars[i].area = 1.0;
                  myObject.bars[i].Ix = 20.0;
```

```
                    myObject.bars[i].Iy = 20.0;
                    myObject.bars[i].Iz = 20.0;
                    myObject.bars[i].G = 20.0;
                    myObject.bars[i].E = 29000;
                    myObject.bars[i].thx = tx[i-1];
                    myObject.bars[i].thy = ty[i-1];
                    myObject.bars[i].thz = tz[i-1];
                  }
infileFrame.setString(myObject.toString() );

s = "% The constraints file can have comments on a \n% line that"
  + " begins with a percent sign\n%\n\n";
s += "*CONSTRAINTS \t " + (2*myObject.NB)+ "\n";
for(int i=1; i<=myObject.NB; i++)
  s += "dl \t" + i + "\tUPPER \t 0.1 \ndl \t" + i + "\tLOWER \t -
0.1\n";
s += "\n\n*OBJECTIVE \t "+myObject.NB+"\n";
for(int i=1; i<=myObject.NB; i++)
  s += "obj \t" + i + "\tVOLUME \t 1.0 \n";
constraintFrame.setString(s);
updateDrawing();
}

}

//
// end class Vif3
//
//////////////////////////////////////////////////////////////
```

```
File: Vit2.java
import java.awt.*;
import java.io.*;
import java.lang.String;
import java.lang.Integer;
import java.text.*; // DecimalFormat
import java.util.*;// StringTokenizer
import java.awt.event.*;
import java.awt.print.PrinterJob;
import java.awt.print.*;

//////////////////////////////////////////////////////////////
//
// begin class Vit2
//

public class Vit2 extends Frame implements ActionListener, Compo-
nentListener,
KeyListener, Printable, WindowListener {

// These are the labels that will be used for the
// buttons in the ButtonStack objects to be created.
//
private String preProLabs[] = {"Input File Editor", "Constraint
File Editor",
"Optimization Settings", "Apply Random Loading", "Reset Default
Values"};
private String mainProLabs[] = {"Start Processor", "Stop Proces-
sor"};
private String postProLabs[] = {"Joint Displacements", "Member
Displacements",
"All Results", "Plot History"};

private String inputErrorMsg, helpMsg;
private String baseName, currentFileName;
private int currentIteration = 0;

private MenuBar bar;
private Menu fileMenu, viewMenu, preProMenu, mainProMenu,
postProMenu, helpMenu;
private PageFormat pgFormat = new PageFormat();
private Paper pgPaper = new Paper();
```

```
// special non-java objects
private L3PCanvas drawingArea;
private Tr2d myObject;
private OptSetFrame optVars;
private DataPlotFramedataFrame;
private HelpFrame helpFrame;
private TextFileFrameinfileFrame;
private TextFileFrameconstraintFrame;
private TextFileFrameresultsFrame;

public static void main(String args[]) {
Vit2 f = new Vit2();

// only using the first arg as a filename for now
if(args.length > 0) {
  // System.out.println("found arg '" + args[0] + "'");
  f.readNewFile(args[0]);
} else {
  f.setDemo();
}
}

// Constructor
public Vit2() {
super("vit2-1.2  Structural Optimization Viewer");

int i;
int width  = 600;
int height = 600;

// setFont(new Font("TimesRoman", Font.PLAIN, 20));

                // create menus
fileMenu   = new Menu("File");
viewMenu = new Menu("View");
preProMenu = new Menu("Pre-Processor");
mainProMenu   = new Menu("Processor");
postProMenu   = new Menu("Post-Processor");
helpMenu   = new Menu("Help");

                // add items to the menus
fileMenu.add( "Open File");
fileMenu.add( "Save File");
fileMenu.add( new MenuItem("-"));
fileMenu.add( "Print as Image");
fileMenu.add( "Print as Page");
fileMenu.add( new MenuItem("-"));
fileMenu.add( "Exit");

viewMenu.add("Node Numbers");
viewMenu.add("Member Numbers");
viewMenu.add( new MenuItem("-"));
viewMenu.add("Color Lines");
viewMenu.add("Thick Lines");
viewMenu.add("Show Loads");
viewMenu.add( new MenuItem("-"));
viewMenu.add("Invert Background");

for(i=0; i<preProLabs.length; i++) preProMenu.add(preProLabs[i]);
for(i=0; i<mainProLabs.length; i++) mainProMenu.add(mainPro-
Labs[i]);
for(i=0; i<postProLabs.length; i++) postProMenu.add(postPro-
Labs[i]);
postProMenu.add( new MenuItem("-"));
postProMenu.add("First Iteration");
postProMenu.add("Next Iteration");
postProMenu.add("Previous Iteration");
helpMenu.add("Help");

                // add menus to menu bar
bar = new MenuBar();
                bar.add(fileMenu);
bar.add(viewMenu);
bar.add(preProMenu);
bar.add(mainProMenu);
bar.add(postProMenu);
bar.add(helpMenu);
setMenuBar(bar);

dataFrame = new DataPlotFrame(400, 400);
drawingArea= new L3PCanvas(height-50, width, this);
infileFrame = new TextFileFrame(40, 70, "Input File Editor",
true);
constraintFrame=new TextFileFrame(40,70, "Constraint File Editor",
true);
resultsFrame = new TextFileFrame(40, 70, "Results Window", false);
```

```
helpFrame = new HelpFrame(40, 70);
optVars = new OptSetFrame();
setSize(width, height);

// things to listen to
fileMenu.addActionListener(this);
viewMenu.addActionListener(this);
preProMenu.addActionListener(this);
mainProMenu.addActionListener(this);
postProMenu.addActionListener(this);
helpMenu.addActionListener(this);

addComponentListener(this);
addKeyListener(this);
addWindowListener(this);

// Setup the form
add(drawingArea);

drawingArea.setBackground(drawingArea.backColor);
drawingArea.setCanRotate(false);
helpFrame.setSize (600,400);
constraintFrame.setSize (500,400);
infileFrame.setSize (500,500);
resultsFrame.setSize (600,500);
optVars.setSize (200,300);

initialize();
setVisible(true);
}

// required for ComponentListener
public void componentHidden(ComponentEvent e) {
// System.out.println("componentHidden event from "
//  + e.getComponent().getClass().getName());
}
public void componentMoved(ComponentEvent e) {
// System.out.println("componentMoved event");
}
public void componentResized(ComponentEvent e) {
// System.out.println("componentResized event");
// System.out.println("new width: " + this.getSize().width);
// System.out.println("new height: " + this.getSize().height);
drawingArea.resetSize(this.getSize().width, this.get-
Size().height-50 );
}
public void componentShown(ComponentEvent e) {
// System.out.println("componentShown event");
}

// required for KeyListener
public void keyPressed(KeyEvent e) { }
public void keyReleased(KeyEvent e) { }
public void keyTyped(KeyEvent e) {
if('m' == e.getKeyChar()) {
  drawingArea.showLineNumber = !drawingArea.showLineNumber;
  drawingArea.repaint();
} else if('n' == e.getKeyChar()) {
  drawingArea.showPointNumber = !drawingArea.showPointNumber;
  drawingArea.repaint();

} else if('a' == e.getKeyChar()) {
  fileMenu.dispatchEvent( new ActionEvent(fileMenu,
  ActionEvent.ACTION_PERFORMED, "All Results") );
} else if('c' == e.getKeyChar()) {
  fileMenu.dispatchEvent( new ActionEvent(fileMenu,
  ActionEvent.ACTION_PERFORMED, "Color Lines") );
} else if('d' == e.getKeyChar()) {
  fileMenu.dispatchEvent( new ActionEvent(fileMenu,
  ActionEvent.ACTION_PERFORMED, "Joint Displacements") );
} else if('D' == e.getKeyChar()) {
  fileMenu.dispatchEvent( new ActionEvent(fileMenu,
  ActionEvent.ACTION_PERFORMED, "Member Displacements") );
} else if('f' == e.getKeyChar()) {
  fileMenu.dispatchEvent( new ActionEvent(fileMenu,
  ActionEvent.ACTION_PERFORMED, "Member Forces") );
} else if('h' == e.getKeyChar()) {
  fileMenu.dispatchEvent( new ActionEvent(fileMenu,
  ActionEvent.ACTION_PERFORMED, "Plot History") );
} else if('i' == e.getKeyChar()) {
  fileMenu.dispatchEvent( new ActionEvent(fileMenu,
  ActionEvent.ACTION_PERFORMED, "Invert Background") );
} else if('j' == e.getKeyChar()) {
  fileMenu.dispatchEvent( new ActionEvent(fileMenu,
  ActionEvent.ACTION_PERFORMED, "Next Iteration") );
} else if('k' == e.getKeyChar()) {
```

```
  fileMenu.dispatchEvent( new ActionEvent(fileMenu,
  ActionEvent.ACTION_PERFORMED, "Previous Iteration") );
} else if('K' == e.getKeyChar()) {
  fileMenu.dispatchEvent( new ActionEvent(fileMenu,
  ActionEvent.ACTION_PERFORMED, "First Iteration") );
} else if('l' == e.getKeyChar()) {
  fileMenu.dispatchEvent( new ActionEvent(fileMenu,
  ActionEvent.ACTION_PERFORMED, "Show Loads") );
} else if('o' == e.getKeyChar()) {
  fileMenu.dispatchEvent( new ActionEvent(fileMenu,
  ActionEvent.ACTION_PERFORMED, "Open File") );
} else if('P' == e.getKeyChar()) {
  fileMenu.dispatchEvent( new ActionEvent(fileMenu,
  ActionEvent.ACTION_PERFORMED, "Print as Page") );
} else if('p' == e.getKeyChar()) {
  fileMenu.dispatchEvent( new ActionEvent(fileMenu,
  ActionEvent.ACTION_PERFORMED, "Print as Image") );
} else if('q' == e.getKeyChar()) {
  fileMenu.dispatchEvent( new ActionEvent(fileMenu,
  ActionEvent.ACTION_PERFORMED, "Exit") );
} else if('r' == e.getKeyChar()) {
  fileMenu.dispatchEvent( new ActionEvent(fileMenu,
  ActionEvent.ACTION_PERFORMED, "Apply Random Loading") );
} else if('s' == e.getKeyChar()) {
  fileMenu.dispatchEvent( new ActionEvent(fileMenu,
  ActionEvent.ACTION_PERFORMED, "Save File") );
} else if('t' == e.getKeyChar()) {
  fileMenu.dispatchEvent( new ActionEvent(fileMenu,
  ActionEvent.ACTION_PERFORMED, "Thick Lines") );
} else if('v' == e.getKeyChar()) {
  SimpleMessage msg = new SimpleMessage("Volume: " +
  String.valueOf(myObject.volume() ), "Volume" );

}
}

// required for WindowListener
public void windowClosed(WindowEvent event) { }
public void windowDeiconified(WindowEvent event) { }
public void windowIconified(WindowEvent event) { }
public void windowActivated(WindowEvent event) { }
public void windowDeactivated(WindowEvent event) { }
public void windowOpened(WindowEvent event) { }
```

```
public void windowClosing(WindowEvent e) { System.exit(0); }


public void actionPerformed(ActionEvent e) {

// Menu Items
if (e.getSource() instanceof MenuItem) {
  // File menu items
  String cmd = e.getActionCommand();
  // if (cmd == readInputFile) {
  if (cmd == "Open File") {
MyFiles m = new MyFiles(this, FileDialog.LOAD);
if(m.fileName != "") {
// System.out.println("Reading " + m.fileName);
readNewFile(m.fileName);
}
//parseinfileString(infileFrame.textString);
  } else if (cmd == "Save File") {
MyFiles m = new MyFiles(this, FileDialog.SAVE);
if(m.fileName != "") {
System.out.println("Writing " + m.fileName);
writeFile(m.fileName);
}
  } else if (cmd == "Print as Page") {
  printImage(0);
  } else if (cmd == "Print as Image") {
  // printImage(1);
  SimpleMessage
  m = new SimpleMessage("'Print as Image' doesn't work",
  "Function Not Enabled");
  } else if (cmd == "Exit") {
System.exit(0);

  // View menu items
  } else if (cmd == "Node Numbers") {
drawingArea.showPointNumber = !drawingArea.showPointNumber;
  } else if (cmd == "Member Numbers") {
drawingArea.showLineNumber = !drawingArea.showLineNumber;
  } else if (cmd == "Color Lines") {
drawingArea.varyLineColors = !drawingArea.varyLineColors;
  } else if (cmd == "Thick Lines") {
drawingArea.varyLineThickness = !drawingArea.varyLineThickness;
  } else if (cmd == "Show Loads") {
```

```
drawingArea.showAuxLines = !drawingArea.showAuxLines;
  } else if (cmd == "Invert Background") {
drawingArea.invertBackground();

  // Pre - Processor menu items
  } else if (cmd == "Input File Editor") {
infileFrame.setVisible(true);
  } else if (cmd == "Constraint File Editor") {
constraintFrame.setVisible(true);
  } else if (cmd == "Optimization Settings") {
optVars.setVisible(true);
  } else if (cmd == "Apply Random Loading") {
createRandomLoad(0.5);
  } else if (cmd == "Reset Default Values") {
setDemo();

  // Main Processor menu items
  } else if (cmd == "Start Processor") {
myObject.analyze();
SimpleMessage m=new SimpleMessage("Analysis Complete",
"Analysis Done");
  } else if (cmd == "Stop Processor") {
// myObject.assemble();
// myObject.analyze();

  // Post - Processor menu items
  } else if (cmd == "Joint Displacements") {
resultsFrame.setString( myObject.JointDisplacements() );
resultsFrame.setVisible(true);
  } else if (cmd == "Member Displacements") {
resultsFrame.setString( myObject.MemberDisplacements() );
resultsFrame.setVisible(true);
  } else if (cmd == "All Results") {
String s = myObject.echoStructure();
// s += "\nOptimization Variables";
// s += "\n\tSLP Iterations: \t" + optVars.totalIterMax;
s += "\n" + myObject.JointDisplacements();
s += "\n" + myObject.MemberDisplacements();
s += "\n" + "Volume: \t" + myObject.volume();
resultsFrame.setString( s );
resultsFrame.setVisible(true);
  } else if (cmd == "Plot History") {
dataFrame.setVisible(true);
  } else if (cmd == "First Iteration") {
changeModel(0);
  } else if (cmd == "Next Iteration") {
changeModel(1);
  } else if (cmd == "Previous Iteration") {
changeModel(-1);

  // Help menu
  } else if (cmd == "Help") {
helpFrame.setVisible(true);
  }
}
// drawingArea.showPointNumber = displayNodeBox.getState();
// drawingArea.showLineNumber  = displayMembBox.getState();
drawingArea.repaint();
}


public int printImage(int flag) {
// This local method will set the 'Paper' to simulate
// either full-page (8.5 x 11) or "image" which is
// defined here. For 'flag' = 0, full-page is used.

// pgPaper = new Paper();
if(flag == 1) {
  // System.out.println("printImage:: flag = 1");
  // setSize(double width, double height)
  double w = 5.0;
  double h = 5.0;
  double margin = 0.5;
  pgPaper.setSize( w*72.0, h*72.0);
  pgPaper.setImageableArea( margin*72.0, margin*72.0,
    (w-2*margin)*72.0, (h-2*margin)*72.0);
}

PrinterJob pj = PrinterJob.getPrinterJob();
// pgFormat.setOrientation(PageFormat.LANDSCAPE);
pgFormat.setPaper(pgPaper);
pgFormat = pj.validatePage(pgFormat);
// pj.setPrintable(this);
pj.setPrintable(this, pgFormat );

if (pj.printDialog()) {
```

```
  try {
    pj.print();
  } catch (Exception ex) {
    ex.printStackTrace();
  }
}
return 0;
}

    public int print(Graphics g, PageFormat pf, int pi) throws
PrinterException {
        if (pi >= 1) {
            return Printable.NO_SUCH_PAGE;
}
pf = pgFormat;
pf.setPaper(pgPaper);

// Shift and scale this object to fit on a page.
// System.out.println(" pf img " + pf.getImageableHeight()+" "
// +pf.getImageableWidth() );
// System.out.println(" pf w x h " + pf.getWidth()+" "+ pf.getH-
eight() );
// System.out.println(" paper w x h " + pgPaper.getWidth()+" "
// + pgPaper.getHeight() );
g.translate( (int)pf.getImageableX(), (int)pf.getImageableY() );

if(pf.getOrientation() == PageFormat.LANDSCAPE) {
  double r[] = new double[3];
  r = drawingArea.getRotation();
  r[2] -= 45.0;
  drawingArea.setRotation(r[0], r[1], r[2]);
  drawingArea.drawObject((Graphics2D) g, (int)pf.getImageable-
Width(),
    (int)pf.getImageableHeight() );
  r[2] += 45.0;
  drawingArea.setRotation(r[0], r[1], r[2]);
} else {
  drawingArea.drawObject((Graphics2D) g, (int)pf.getImageable-
Width(),
    (int)pf.getImageableHeight() );
}
        return Printable.PAGE_EXISTS;
    }
```

```
public void changeModel(int inc) {
// Get a list of files in this directory with the current base-
name.
// Locate the 'currentFileName' in this list and then read the
next
// one if it exists. This will allow for non-consecutive iteration
// numbers (e.g., if the analysis prints only every 5th itera-
tion).

boolean found;
DecimalFormat fmt = new DecimalFormat("000");
File f;
String fileList[];
int i;

// System.out.println("changeModel: current: '" + currentFileName
+ "'");
// System.out.println("changeModel: baseName: '" + baseName +
"'");
// System.out.println("changeModel: inc: '" + inc + "'");

if(currentIteration == 0 || inc == 0) {
  // System.out.println("starting from zero");
  if(readFile(baseName + ".001.osi") ) {
    currentFileName = baseName + ".001.osi";
    myObject.analyze();
    updateDrawing();
    currentIteration = 1;
    dataFrame.X[0] = currentIteration;
    dataFrame.Y[0] = myObject.volume();
    dataFrame.nPoints = 1;
    return;
  } else {
                    SimpleMessage m = new SimpleMessage("Not found
"+baseName+".001.osi",
                        "BaseName: " + baseName);
                    }
}

f = new File(currentFileName);
f = new File(f.getAbsolutePath());
```

```
f = new File(f.getParent());
// if(f.exists()) System.out.println(" f.getParent() " + f.getPar-
ent() );
fileList = f.list(); // don't know how to create a filter
sortStringArray(fileList);

// Search the entire list for the 'currentFileName'. If there
exists
// a file in this family after (before) it, read this file.
//
for(i=0; i<fileList.length; i++) {
  // System.out.println("compare "+fileList[i]+" and "+current-
FileName);
  if(fileList[i].equals(currentFileName) ) {
    // Found the current file, now see if the increment (+/-) is
    // also an "osi" file with the same 'baseName' and is not the
    // original model itself ('getIteration' returns a zero).
    //
    // System.out.println("\tfound a match");
    if(fileList[i+inc].startsWith(baseName)
      && fileList[i+inc].endsWith(".osi")
      && getIteration(fileList[i+inc]) != 0) {
readFile(fileList[i+inc]);
currentFileName = fileList[i+inc];
updateDrawing();
currentIteration = getIteration(currentFileName);
myObject.analyze();

// Update the history plot by scanning the entire array
// for existence of this iteration. If not found, increment
// the number of points 'nPoints' and add the y-value.
found = false;
for(i=0; i<dataFrame.nPoints; i++) {
  if(dataFrame.X[i] == currentIteration) {
    dataFrame.Y[i] = myObject.volume();
    found = true;
  }
}
if(!found) {
  dataFrame.X[dataFrame.nPoints] = currentIteration;
  dataFrame.Y[dataFrame.nPoints] = myObject.volume();
  dataFrame.nPoints++;
}
```

```
    } else {
SimpleMessage msg = new SimpleMessage("This is the "
 + (inc > 0 ? "last" : "first") + " iteration", "BaseName: "
 + baseName);
    }
    break;
  }
}
}

public void sortStringArray(String s[]) {
// This method will sort the string 's' and return it with the
// same number of elements in a lexicographic sorted order.
// It's hard to believe that there's no method provided in java
// to do this but I conldn't find one. Then again, it's not perl.
//
int i;
String temp;
boolean switched;

do {
 switched = false;
 for(i=0; i<s.length-1; i++) {
 if(s[i].compareTo(s[i+1]) > 0) {
   temp = s[i];
   s[i] = s[i+1];
   s[i+1] = temp;
   switched = true;
  }
 }
} while(switched);

}


public String removeExtension(String fullName) {
// Given a file name 'fullName' this method returns the
// equivalent 'baseName'. This is defined here as all
// characters before the last 4, which are assumed here
// to be the extension '.osi'. If the string doesn't end
// with this, simply return the original string

if(fullName.endsWith(".osi"))
```

```
  return fullName.substring(0, fullName.length()-4);
else
  return fullName;
}

public int getIteration(String fullName) {
// Given a file name 'fullName' this method returns the embedded
// iteration number. This number is defined here as the three
// characters after the 'baseName' but before the extension.

int start, end, num;
num = 0;

// System.out.println("getIteration: string: " + fullName);
start = baseName.length() + 1;
end = start + 3;
try {
  num = Integer.parseInt( fullName.substring(start, end) );
} catch(NumberFormatException e) {
  // System.out.println("getIteration: exception " + e);
  num = 0;
}
// System.out.println("getIteration: num: " + num);

return num;
}


public void readNewFile(String fileName) {
// Reset some values if the call to 'readFile' is successful

if(readFile(fileName)) {
  baseName = removeExtension(fileName);
  currentFileName = fileName;
  currentIteration = 0;
  dataFrame.nPoints = 0;
  updateDrawing();
}
}

public boolean readFile(String fileName) {// OPEN command
BufferedReader b = null;
StringTokenizer t;
String s = "";
int i;

try {
b = new BufferedReader ( new FileReader(fileName) );
} catch(IOException e) {
SimpleMessage m = new SimpleMessage("file open failed: '"
  + fileName + "'", "File Not Opened");
b = null;
return false;
// System.exit(1);
}

// Read each line as a String and tokenize to get the entries.
// System.out.println("readFile(" + fileName + ")");
if(b != null) {
try {
int nn, nb, ns;
s = b.readLine();
b.close();
t = new StringTokenizer(s);
nb = Integer.parseInt( t.nextToken() );
nn = Integer.parseInt( t.nextToken() );
ns = Integer.parseInt( t.nextToken() );
myObject = new Tr2d(nb, nn, ns);
myObject.readFile(fileName);
infileFrame.setString( myObject.toString() );
return true;
} catch(Exception e) {
System.err.println("Input file not opened properly \n"
  + e.toString() );
SimpleMessage msg = new SimpleMessage("file open failed"
  + e, "File Not Opened");
// System.exit(1);
}
}

return false;
}

public boolean readFileToString(String fileName) {// OPEN command
DataInputStream input = null;
String s = "";
```

```
try {
input = new DataInputStream ( new FileInputStream(fileName) );
} catch(IOException e) {
System.err.println("Input file not opened properly \n"
  + e.toString() );
SimpleMessage msg = new SimpleMessage("file open failed"
  + e, "File Not Opened");
input = null;
// System.exit(1);
}

// Read the file into a String 's' and (if successful) construct
// a new 'myObject' with this data.
// System.out.println("readFile(" + fileName + ")");
if(input != null) {
  try {
// there should be a better way to do this
while(true) {
  s += (char)input.readByte();
}
  } catch(EOFException e) {
try {
input.close();
// System.out.println("found data '" + s + "' in file");
// System.out.println("End of file reached");
infileFrame.setString(s);
myObject = new Tr2d(s);
myObject.analyze();
return true;
} catch(IOException ex) {
System.out.println("Error closing file\n" + ex);
}
  } catch(IOException e) {
System.err.println("Error during read file \n" + e.toString() );
// System.exit(1);
  }
}
return false;
}

public void writeFile(String fileName) {// SAVE command
DataOutputStream output = null;
try {// Open the file
output = new DataOutputStream ( new FileOutputStream(fileName) );
} catch(IOException e) {
System.err.println("File not opened properly \n" + e.toString() );
// System.exit(1);
}

if(output != null) {
  try {// Write the file
output.writeBytes( myObject.toString() );
output.close();
  } catch(IOException e) {
System.err.println("Error during write to file \n" + e.toString()
);
// System.exit(1);
  }
}
}

public void updateDrawing() {
int i, n1, n2;
double al;

for(i=1; i<= myObject.NN; i++) {
drawingArea.X[i] = myObject.nodes[i].x;
drawingArea.Y[i] = myObject.nodes[i].y;
}
for(i=1; i<= myObject.NB; i++) {
drawingArea.Point1[i] = myObject.bars[i].NM;
drawingArea.Point2[i] = myObject.bars[i].NP;
drawingArea.lineThickness[i] = myObject.bars[i].area;
}

                al = scaleLoadsToObject();
              // Set the auxillary lines (i.e., the applied loads)
// System.out.println("\n updateDrawing: n1   n2  nAux");
n2=myObject.NN;
drawingArea.nAux = 0;
                for(i=1; i <= myObject.freeNodes; i++) {
                  if( (myObject.nodes[i].Px != 0) || (myOb-
ject.nodes[i].Py != 0) ) {
                        n1 = i;
n2++;
```

```
drawingArea.nAux++;
                        drawingArea.auxP1[drawingArea.nAux] = n1;
                        drawingArea.auxP2[drawingArea.nAux] = n2;
                        drawingArea.X[n2] = drawingArea.X[n1] -
al*myObject.nodes[i].Px;
                        drawingArea.Y[n2] = drawingArea.Y[n1] -
al*myObject.nodes[i].Py;
// System.out.println(n1+" "+n2+" "+drawingArea.nAux);
                  }
                }

drawingArea.nPoints = myObject.NN;
drawingArea.nLines  = myObject.NB;
drawingArea.scaleLineThickness();
drawingArea.setTopLeftString(currentFileName);
drawingArea.refresh();
drawingArea.setRotation(180.0, 0.0, 0.0);
drawingArea.repaint();
}

        public double scaleLoadsToObject() {
                // Compute and return a scale factor, 'alpha' that
should be
                // used to multiply the loads by so that the max
load shows
                // up in a drawing as a percentage of the maximum
object dimension.
                int i;
                double alpha, ratio, maxRange, maxPrange;
                double max[] = new double[2];
                double min[] = new double[2];
                double Pmax[] = new double[2];
                double Pmin[] = new double[2];
                double range[] = new double[2];
                double Prange[] = new double[2];

                // Find the extreme values
                for(i=1; i<= myObject.NN; i++) {
                 min[0] = myObject.nodes[i].x > min[0] ? min[0] :
myObject.nodes[i].x;
                 min[1] = myObject.nodes[i].y > min[1] ? min[1] :
myObject.nodes[i].y;
                 max[0] = myObject.nodes[i].x < max[0] ? max[0] :
myObject.nodes[i].x;
                 max[1] = myObject.nodes[i].y < max[1] ? max[1] :
myObject.nodes[i].y;

                 Pmin[0]= myObject.nodes[i].Px>Pmin[0] ? Pmin[0]
: myObject.nodes[i].Px;
                 Pmin[1]= myObject.nodes[i].Py>Pmin[1] ? Pmin[1]
: myObject.nodes[i].Py;
                 Pmax[0]= myObject.nodes[i].Px<Pmax[0] ? Pmax[0]
: myObject.nodes[i].Px;
                 Pmax[1]= myObject.nodes[i].Py<Pmax[1] ? Pmax[1]
: myObject.nodes[i].Py;
                }
                maxRange = 0;
                maxPrange = 0;
                for(i=0; i<2; i++) {
                  range[i] = max[i] - min[i];
                  Prange[i] = Pmax[i] - Pmin[i];
                  maxRange = range[i] > maxRange ? range[i] :
maxRange;
                  maxPrange = Prange[i] > maxPrange ? Prange[i] :
maxPrange;
                  // System.out.println(i+" "+range[i]+"
"+Prange[i]);
                }
                // alpha = 1000000;
                alpha = maxPrange != 0 ? Math.abs(maxRange/max-
Prange) : 1.0;
                alpha *= 0.2;                    // 20% is done here
                return alpha;
        }


        public int createRandomLoad(double density) {
                int i, j;
                double MinVal, MaxVal, newMax;
                double Load[] = new double[2];
                String s;
                DecimalFormat fmt;

                MaxVal = 0.0;
                newMax = 0.0;
```

```
                // First identify the existing maximum value. This
may not be the
                // most efficient but that really doesn't matter
because this method
                // probably won't be called often and when it is,
it will be a 'low
               // - demand' time (i.e., nothing else is happening).
                 for(i=1; i <= myObject.NN; i++) {
                  if(Math.abs(myObject.nodes[i].Px) > MaxVal)
                         MaxVal = Math.abs(myObject.nodes[i].Px);
                  if(Math.abs(myObject.nodes[i].Py) > MaxVal)
                         MaxVal = Math.abs(myObject.nodes[i].Py);
                 }

                 // MaxVal *= 1.5;
                 MinVal = 0.1 * MaxVal;
                 if(MaxVal < 10) fmt = new DecimalFormat("0.000");
                 else fmt = new DecimalFormat("0.0");

// Gererate a random load and (not) apply it to a node.
                 for(i=1; i <= myObject.freeNodes; i++) {
                  for(j=0; j<2; j++) {
                  Load[j] = MinVal + Math.random() * (MaxVal - Min-
Val);
                    if(Math.random() > 0.5) Load[j] *= -1.0;
                  }
 myObject.nodes[i].Px = (Math.random() < density ? Load[0] : 0.0);
 myObject.nodes[i].Py = (Math.random() < density ? Load[1] : 0.0);
                 }

// Ensure that at least one of the new values is equal in
// magnitude to the previous 'MaxVal'.
                 for(i=1; i <= myObject.NN; i++) {
  if(Math.abs(myObject.nodes[i].Px) > newMax)
    newMax = Math.abs(myObject.nodes[i].Px);
  if(Math.abs(myObject.nodes[i].Py) > newMax)
    newMax = Math.abs(myObject.nodes[i].Py);
                 }
                 // System.out.println("randomLoad: maxval "+Max-
Val+", newmax "+newMax);
if(newMax == 0) {
  // System.out.println("found zero" );
  myObject.nodes[1].Px = MaxVal;
} else if(newMax != MaxVal) {
  // System.out.println("scaling by " + (MaxVal/newMax) );
                  for(i=1; i <= myObject.NN; i++) {
    myObject.nodes[i].Px *= MaxVal / newMax;
    myObject.nodes[i].Py *= MaxVal / newMax;
  }
}

// Reset all bar areas in the input file to 1.0 because one
                // optimization job may have been run by now and
the bar areas
                // will have different values.
                s = myObject.NB + "\t" + myObject.NN + "\t" + myO-
bject.NS + "\n";
                for(i=1; i <= myObject.NN; i++)
                 s += myObject.nodes[i].x
                   + "\t" + myObject.nodes[i].y
                   + "\t" + fmt.format(myObject.nodes[i].Px)
                   + "\t" + fmt.format(myObject.nodes[i].Py)
                   + "\n";
                for(i=1; i <= myObject.NB; i++)
                 s += myObject.bars[i].NM + "\t" + myOb-
ject.bars[i].NP + "\t 1.0 \t"
                   + myObject.bars[i].E + "\n";
                infileFrame.setString( myObject.toString() );
                myObject.analyze();

               // The loads have been changed so we need to update
the drawing.
                updateDrawing();

                return 0;
        }

public void initialize() {
   helpMsg = "Help::General"
   + "\n\tThis text area should define the object. The"
+ "\n\tfirst line should have  NB, NN, NS. "
+ "\n\tThe next block should have NN lines of nodal data. \n"
+ "\nHelp::Input"
+ "\n\tdata and the following line should have NRINGS "
+ "\n\tThe buttons at the right clear this frame (but "
```

```
+ "\n\tnot the data), read data from this frame, set \n"
+ "\nHelp::Output"
+ "\n\tYou can get the values"
+ "\n\tfor any given iteration by simply viewing that iteration
and"
+ "\n\tviewing the Input File Editor.\n"
+ "\nHelp::Print"
+ "\n\tBe sure to click on the Invert Colors checkbox before
printing"
+ "\n\tunless you really want the background to be black (which
uses "
+ "\n\t*lots* of toner). ";
   inputErrorMsg = "*** INPUT ERROR ***"
+ " \n There was a problem reading your input. This "
+ " \n may be due to not enough data or text (other"
+ " \n than e or E for exponential) in the data \n"
+ " \n You can delete these lines, make your correction"
+ " \n and try again. \n\n"
+ " \n A NumberFormatException was thrown with message:\n";
   helpFrame.setMessage(helpMsg);
}

public void setDemo() {
String s;

/*myObject   = new Tr2d(4, 5, 4);
double x[] = { 0.0, -10.0,  10.0,   3.0,  -3.0 };
double y[] = { 0.0, -10.0, -10.0, -10.0, -10.0 };
int NM[]   = {1, 1, 1, 1};
int NP[]   = {2, 3, 4, 5};
myObject.nodes[1].Px = 10.0;
myObject.nodes[1].Py = 5.0;
*/
myObject   = new Tr2d(11, 7, 2);
double x[] = { 2.5, 7.5, 12.5, 5.0, 10.0, 0.0, 15.0};
double y[] = { 4.0, 5.0,  4.0, 0.0, 0.0,  0.0, 0.0};
int NM[]   = {1,2,1, 1, 4, 2, 5, 3,   6, 4, 5};
int NP[]   = {2,3,6, 4, 2, 5, 3, 7,   4, 5, 7};

                for(int i = 1; i <= myObject.NN; i++) {
                  myObject.nodes[i].x = x[i-1];
                  myObject.nodes[i].y = y[i-1];
                  myObject.nodes[i].Px = 0;
                  myObject.nodes[i].Py = 0;
                }
myObject.nodes[4].Py = -1000.0;
myObject.nodes[5].Py = -1000.0;

                for(int i = 1; i <= myObject.NB; i++) {
                  myObject.bars[i].NM = NM[i-1];
                  myObject.bars[i].NP = NP[i-1];
                  myObject.bars[i].area = 1.0;
                  myObject.bars[i].E = 29000;
                }

s = myObject.NB + "\t" + myObject.NN + "\t" + myObject.NS + "\n";
for(int i = 1; i <= myObject.NN; i++)
s += myObject.nodes[i].x + "\t" + myObject.nodes[i].y + "\t"
  + myObject.nodes[i].Px + "\t" + myObject.nodes[i].Py + "\n";
for(int i = 1; i <= myObject.NB; i++)
s += myObject.bars[i].NM + "\t" + myObject.bars[i].NP + "\t"
  + myObject.bars[i].area + "\t" + myObject.bars[i].E + "\n";
infileFrame.setString(s);

s = "% The constraints file can have comments on a \n% line that"
  + " begins with a percent sign\n%\n\n";
s += "*CONSTRAINTS \t " + (2*myObject.NB)+ "\n";
for(int i=1; i<=myObject.NB; i++)
  s += "dl \t" + i + "\tUPPER \t 0.1 \ndl \t" + i + "\tLOWER \t -
0.1\n";
s += "\n\n*OBJECTIVE \t "+myObject.NB+"\n";
for(int i=1; i<=myObject.NB; i++)
  s += "obj \t" + i + "\tVOLUME \t 1.0 \n";
constraintFrame.setString(s);
updateDrawing();
baseName = "";
currentFileName = "";

}

}
//
// end class Vit2
//
////////////////////////////////////////////////////////////////
```

APPENDIX C
INPUT FILE FORMAT

There are three very similar input file formats used in this work. All have the structural dimensions on the first line, followed by a number of blocks that describe the nodal and member details. Each block is a group of lines that all have the same number of terms and the lines in each block are ordered such that line *i* in a block corresponds to item *i* in the structure. The number of terms on each line varies according to the type of block and the terms are listed in the order described here.

The first line of the truss input file contains three integers which are the number of members, nodes and supports in the structure. Following this line are two blocks of data. The first block contains the nodal data and thus has one line for each node. Each data line in this block has four terms, which are the x coordinate, y coordinate, applied force in the x–direction, and the applied force the y–direction for a given node. The second block describes the member data and connectivity. Each line in this block also has four terms but these are the negative node number, positive node number, cross sectional area, and Young's Modulus for that member.

There are only two differences between the two–dimensional frame input file and the truss file just described. The first difference is that in the first block, there is one extra term required for the value of a concentrated moment at each node thus the total number of terms on each line in this block is five. The second difference is that there are more terms used to describe the member. Each line in the second block has, in order, the negative node number, positive node number, cross sectional area, second moment of the area (also known as the section constant I), and Young's Modulus for a given member.

The third type of input file is also very similar but there are three blocks of data after the first line. The first line has three integers that determine the number of nodes,

the number of nodes, and the total number of degrees of freedom. The first block describes the nodal data thus there is one line for each node. Each line in this block contains nine terms which are the x, y and z coordinates, the force loads in the x, y, and z directions, and the concentrated moments about the global x, y, and z axes.

The second block of data contains information about the members and there is one line for each member. Each line in this block contains 11 terms, and the first two are the negative node number and the positive node number. The following six terms describe the section and material constants for the member. These are the area, the torsional constant, the second moment of the area about the y and z axis, Young's modulus (E), and the shear modulus (G). The final three terms in each line of the second block describe the angles required to rotate the element into the global coordinate system. In order, these angles represent rotations about the x, y, and z axis.

The third block of data contains information about the degrees of freedom. There is one line for each node and each line has six integer values. The integers are used as flags and each can have a value of either zero or one. Each flag, in order, corresponds to one of the six possible degrees of freedom dx, dy, dz, θ_x, θ_y, and θ_z. A zero indicates the direction is free while a one indicates that direction is fixed.

File: t35s.040.osi

```
36   12    2
   60.00000        0.00000        0.00000        0.00000
    0.00000       15.00000        0.00000        1.00000
   60.00000       15.00000        0.00000        0.00000
   20.00000        0.00000        0.00000        0.00000
   20.00000       15.00000        0.00000        0.00000
   20.00000       30.00000        0.00000        0.00000
   40.00000        0.00000        0.00000        0.00000
   40.00000       15.00000        0.00000        0.00000
   40.00000       30.00000        0.00000        0.00000
   60.00000       30.00000        0.00000        0.00000
   80.00000        0.00000        0.00000        0.00000
   80.00000       30.00000        0.00000        0.00000
 2    4    9.43572e-01        2.90000e+04
 2    6    9.84339e-01        2.90000e+04
 2    7    2.00017e-01        2.90000e+04
 2    9    2.14182e-01        2.90000e+04
 4    5    2.97167e-02        2.90000e+04
 4    7    1.35420e+00        2.90000e+04
 4    8    7.77491e-01        2.90000e+04
 4    9    7.52808e-02        2.90000e+04
 5    6    3.53663e-02        2.90000e+04
 5    7    7.26506e-02        2.90000e+04
 5    9    7.15347e-02        2.90000e+04
 5    1    2.14060e-01        2.90000e+04
 5   10    2.16548e-01        2.90000e+04
 6    7    1.10423e-01        2.90000e+04
 6    8    7.02931e-01        2.90000e+04
 6    9    1.30401e+00        2.90000e+04
 7    8    1.87267e-02        2.90000e+04
 7    1    9.34022e-01        2.90000e+04
 7    3    1.64269e-01        2.90000e+04
 7   10    3.77428e-02        2.90000e+04
 8    9    1.43676e-02        2.90000e+04
 8    1    2.38321e-01        2.90000e+04
 8   10    3.25801e-01        2.90000e+04
 8   11    1.44943e+00        2.90000e+04
 8   12    1.49628e+00        2.90000e+04
 9    1    7.25405e-02        2.90000e+04
 9    3    1.04249e-01        2.90000e+04
 9   10    8.89136e-01        2.90000e+04
 1    3    6.01753e-02        2.90000e+04
 1   11    1.27214e+00        2.90000e+04
 1   12    1.77019e-01        2.90000e+04
 3   10    9.34501e-02        2.90000e+04
 3   11    2.26410e-01        2.90000e+04
 3   12    2.76145e-01        2.90000e+04
10   11    1.74762e-01        2.90000e+04
10   12    1.15597e+00        2.90000e+04
```

File: t16.030.osi

```
61   14    2
   45.00000        10.00000         0.00000         0.00000
    0.00000        10.00000         0.00000         0.00000
    7.50000         0.00000         0.00000        -1.00000
    7.50000        10.00000         0.00000         0.00000
   15.00000         0.00000         0.00000        -1.00000
   15.00000        10.00000         0.00000         0.00000
   22.50000         0.00000         0.00000        -1.00000
   22.50000        10.00000         0.00000         0.00000
   30.00000         0.00000         0.00000        -1.00000
   30.00000        10.00000         0.00000         0.00000
   37.50000         0.00000         0.00000        -1.00000
   37.50000        10.00000         0.00000         0.00000
    0.00000         0.00000         0.00000         0.00000
   45.00000         0.00000         0.00000         0.00000
13     2    9.02516e-03         2.90000e+07
13     3    1.35377e-02         2.90000e+07
13     4    1.09956e+00         2.90000e+07
13     6    3.39628e+00         2.90000e+07
13     8    5.84487e-02         2.90000e+07
13    10    1.23352e-02         2.90000e+07
13    12    9.02516e-03         2.90000e+07
13     1    9.02516e-03         2.90000e+07
 2     3    9.02516e-03         2.90000e+07
 2     4    1.60568e-02         2.90000e+07
 2     5    9.02516e-03         2.90000e+07
 2     7    9.02516e-03         2.90000e+07
 2     9    9.02516e-03         2.90000e+07
 2    11    9.02516e-03         2.90000e+07
 2    14    9.02516e-03         2.90000e+07
 3     4    6.97931e-01         2.90000e+07
 3     5    9.02516e-03         2.90000e+07
 3     6    9.02516e-03         2.90000e+07
 3     8    9.02516e-03         2.90000e+07
 3    10    9.02516e-03         2.90000e+07
 3    12    9.02516e-03         2.90000e+07
 3     1    9.02516e-03         2.90000e+07
 4     5    9.02516e-03         2.90000e+07
 4     6    1.42492e+00         2.90000e+07
 4     7    9.02516e-03         2.90000e+07
 4     9    9.02516e-03         2.90000e+07
 4    11    9.02516e-03         2.90000e+07
 4    14    9.02516e-03         2.90000e+07
 5     6    6.95187e-01         2.90000e+07
 5     7    9.02516e-03         2.90000e+07
 5     8    9.02516e-03         2.90000e+07
 5    10    9.02516e-03         2.90000e+07
 5    12    9.02516e-03         2.90000e+07
 5     1    9.02516e-03         2.90000e+07
 6     7    6.98128e-01         2.90000e+07
 6     8    5.85782e+00         2.90000e+07
 6     9    9.02516e-03         2.90000e+07
 6    11    9.02516e-03         2.90000e+07
 6    14    1.23352e-02         2.90000e+07
 7     8    1.05329e-02         2.90000e+07
 7     9    9.02516e-03         2.90000e+07
 7    10    6.83324e-01         2.90000e+07
 7    12    9.02516e-03         2.90000e+07
 7     1    9.02516e-03         2.90000e+07
 8     9    9.02516e-03         2.90000e+07
 8    10    5.85614e+00         2.90000e+07
 8    11    9.02516e-03         2.90000e+07
 8    14    6.01953e-02         2.90000e+07
 9    10    6.95210e-01         2.90000e+07
 9    11    9.02516e-03         2.90000e+07
 9    12    9.02516e-03         2.90000e+07
 9     1    9.02516e-03         2.90000e+07
10    11    9.02516e-03         2.90000e+07
10    12    1.42588e+00         2.90000e+07
10    14    3.39475e+00         2.90000e+07
11    12    6.97963e-01         2.90000e+07
11    14    1.35377e-02         2.90000e+07
11     1    9.02516e-03         2.90000e+07
12    14    1.09977e+00         2.90000e+07
12     1    1.65055e-02         2.90000e+07
14     1    9.02516e-03         2.90000e+07
```

File: xt210.030.osi

```
51   22    2
  100.00000        10.00000         0.00000         0.00000
    0.00000        10.00000         0.00000         0.00000
   10.00000         0.00000         0.00000         0.00000
   10.00000        10.00000         0.00000         0.00000
   20.00000         0.00000         0.00000         0.00000
   20.00000        10.00000         0.00000         0.00000
   30.00000         0.00000         0.00000         0.00000
   30.00000        10.00000         0.00000         0.00000
   40.00000         0.00000         0.00000         0.00000
   40.00000        10.00000         0.00000         0.00000
   50.00000         0.00000         0.00000        -1.00000
   50.00000        10.00000         0.00000         0.00000
   60.00000         0.00000         0.00000         0.00000
   60.00000        10.00000         0.00000         0.00000
   70.00000         0.00000         0.00000         0.00000
   70.00000        10.00000         0.00000         0.00000
   80.00000         0.00000         0.00000         0.00000
   80.00000        10.00000         0.00000         0.00000
   90.00000         0.00000         0.00000         0.00000
   90.00000        10.00000         0.00000         0.00000
    0.00000         0.00000         0.00000         0.00000
  100.00000         0.00000         0.00000         0.00000
21    2    5.39600e-03         2.90000e+04
 3    4    1.68307e-01         2.90000e+04
 5    6    5.39707e-03         2.90000e+04
 7    8    5.39564e-03         2.90000e+04
 9   10    5.39566e-03         2.90000e+04
11   12    8.09473e-03         2.90000e+04
13   14    5.39566e-03         2.90000e+04
15   16    5.39564e-03         2.90000e+04
17   18    5.39707e-03         2.90000e+04
19   20    1.68307e-01         2.90000e+04
22    1    5.39600e-03         2.90000e+04
21    3    1.81429e-01         2.90000e+04
 2    4    5.39600e-03         2.90000e+04
 3    5    1.17421e-02         2.90000e+04
 4    6    1.74941e-01         2.90000e+04
 5    7    5.39557e-03         2.90000e+04
 6    8    5.10625e-01         2.90000e+04
 7    9    3.32901e-01         2.90000e+04
 8   10    5.21733e-01         2.90000e+04
 9   11    3.39742e-01         2.90000e+04
10   12    8.59720e-01         2.90000e+04
11   13    3.39742e-01         2.90000e+04
12   14    8.59720e-01         2.90000e+04
13   15    3.32901e-01         2.90000e+04
14   16    5.21733e-01         2.90000e+04
15   17    5.39557e-03         2.90000e+04
16   18    5.10625e-01         2.90000e+04
17   19    1.17421e-02         2.90000e+04
18   20    1.74941e-01         2.90000e+04
19   22    1.81429e-01         2.90000e+04
20    1    5.39600e-03         2.90000e+04
21    4    3.41667e-01         2.90000e+04
 3    6    3.39774e-01         2.90000e+04
 5    8    1.32342e-02         2.90000e+04
 7   10    3.35844e-01         2.90000e+04
 9   12    5.39718e-03         2.90000e+04
11   14    3.40130e-01         2.90000e+04
13   16    9.00426e-03         2.90000e+04
15   18    3.31594e-01         2.90000e+04
17   20    5.39892e-03         2.90000e+04
19    1    5.39564e-03         2.90000e+04
 2    3    5.39564e-03         2.90000e+04
 4    5    5.39892e-03         2.90000e+04
 6    7    3.31594e-01         2.90000e+04
 8    9    9.00426e-03         2.90000e+04
10   11    3.40130e-01         2.90000e+04
12   13    5.39718e-03         2.90000e+04
14   15    3.35844e-01         2.90000e+04
16   17    1.32342e-02         2.90000e+04
18   19    3.39774e-01         2.90000e+04
20   22    3.41667e-01         2.90000e+04
```

File: xt213.030.osi

```
51   22    2
  100.00000        10.00000         0.00000         0.00000
    0.00000        10.00000         0.00000         0.00000
   10.00000         0.00000         0.00000         0.00000
   10.00000        10.00000         0.00000         0.00000
   20.00000         0.00000         0.00000         0.00000
   20.00000        10.00000         0.00000         0.00000
   30.00000         0.00000         0.00000         0.00000
   30.00000        10.00000         0.00000         0.00000
   40.00000         0.00000         0.00000         0.00000
   40.00000        10.00000         0.00000         0.00000
   50.00000         0.00000         0.00000        -1.00000
   50.00000        10.00000         0.00000         0.00000
   60.00000         0.00000         0.00000         0.00000
   60.00000        10.00000         0.00000         0.00000
   70.00000         0.00000         0.00000         0.00000
   70.00000        10.00000         0.00000         0.00000
   80.00000         0.00000         0.00000         0.00000
   80.00000        10.00000         0.00000         0.00000
   90.00000         0.00000         0.00000         0.00000
   90.00000        10.00000         0.00000         0.00000
    0.00000         0.00000         0.00000         0.00000
  100.00000         0.00000         0.00000         0.00000
21    2    5.39601e-03         2.90000e+04
 3    4    1.68307e-01         2.90000e+04
 5    6    5.39709e-03         2.90000e+04
 7    8    5.39565e-03         2.90000e+04
 9   10    5.39567e-03         2.90000e+04
11   12    8.09476e-03         2.90000e+04
13   14    5.39567e-03         2.90000e+04
15   16    5.39565e-03         2.90000e+04
17   18    5.39709e-03         2.90000e+04
19   20    1.68307e-01         2.90000e+04
22    1    5.39601e-03         2.90000e+04
21    3    1.81429e-01         2.90000e+04
 2    4    5.39601e-03         2.90000e+04
 3    5    1.17421e-02         2.90000e+04
 4    6    1.74941e-01         2.90000e+04
 5    7    5.39559e-03         2.90000e+04
 6    8    5.10625e-01         2.90000e+04
 7    9    3.32901e-01         2.90000e+04
 8   10    5.21733e-01         2.90000e+04
 9   11    3.39742e-01         2.90000e+04
10   12    8.59720e-01         2.90000e+04
11   13    3.39742e-01         2.90000e+04
12   14    8.59720e-01         2.90000e+04
13   15    3.32901e-01         2.90000e+04
14   16    5.21733e-01         2.90000e+04
15   17    5.39559e-03         2.90000e+04
16   18    5.10625e-01         2.90000e+04
17   19    1.17421e-02         2.90000e+04
18   20    1.74941e-01         2.90000e+04
19   22    1.81429e-01         2.90000e+04
20    1    5.39601e-03         2.90000e+04
21    4    3.41669e-01         2.90000e+04
 3    6    3.39776e-01         2.90000e+04
 5    8    1.32343e-02         2.90000e+04
 7   10    3.35846e-01         2.90000e+04
 9   12    5.39719e-03         2.90000e+04
11   14    3.40131e-01         2.90000e+04
13   16    9.00430e-03         2.90000e+04
15   18    3.31596e-01         2.90000e+04
17   20    5.39894e-03         2.90000e+04
19    1    5.39566e-03         2.90000e+04
 2    3    5.39566e-03         2.90000e+04
 4    5    5.39894e-03         2.90000e+04
 6    7    3.31596e-01         2.90000e+04
 8    9    9.00430e-03         2.90000e+04
10   11    3.40131e-01         2.90000e+04
12   13    5.39719e-03         2.90000e+04
14   15    3.35846e-01         2.90000e+04
16   17    1.32343e-02         2.90000e+04
18   19    3.39776e-01         2.90000e+04
20   22    3.41669e-01         2.90000e+04
```

File: xt211.030.osi

```
51   22    2
  100.00000        10.00000         0.00000         0.00000
    0.00000        10.00000         0.00000         0.00000
   10.00000         0.00000         0.00000         0.00000
   10.00000        10.00000         0.00000         0.00000
   20.00000         0.00000         0.00000         0.00000
   20.00000        10.00000         0.00000         0.00000
   30.00000         0.00000         0.00000         0.00000
   30.00000        10.00000         0.00000         0.00000
   40.00000         0.00000         0.00000         0.00000
   40.00000        10.00000         0.00000         0.00000
   50.00000         0.00000         0.00000        -1.00000
   50.00000        10.00000         0.00000         0.00000
   60.00000         0.00000         0.00000         0.00000
   60.00000        10.00000         0.00000         0.00000
   70.00000         0.00000         0.00000         0.00000
   70.00000        10.00000         0.00000         0.00000
   80.00000         0.00000         0.00000         0.00000
   80.00000        10.00000         0.00000         0.00000
   90.00000         0.00000         0.00000         0.00000
   90.00000        10.00000         0.00000         0.00000
    0.00000         0.00000         0.00000         0.00000
  100.00000         0.00000         0.00000         0.00000
21    2    6.11339e-03         2.90000e+04
 3    4    1.56631e-01         2.90000e+04
 5    6    6.14381e-03         2.90000e+04
 7    8    6.11305e-03         2.90000e+04
 9   10    6.12538e-03         2.90000e+04
11   12    3.10301e-02         2.90000e+04
13   14    9.18257e-03         2.90000e+04
15   16    6.11630e-03         2.90000e+04
17   18    9.17406e-03         2.90000e+04
19   20    1.56447e-01         2.90000e+04
22    1    6.11261e-03         2.90000e+04
21    3    2.53660e-01         2.90000e+04
 2    4    6.11339e-03         2.90000e+04
 3    5    3.12982e-02         2.90000e+04
 4    6    2.45366e-01         2.90000e+04
 5    7    6.11149e-03         2.90000e+04
 6    8    5.28261e-01         2.90000e+04
 7    9    3.09163e-01         2.90000e+04
 8   10    5.64471e-01         2.90000e+04
 9   11    4.67080e-01         2.90000e+04
10   12    9.59881e-01         2.90000e+04
11   13    4.32706e-01         2.90000e+04
12   14    8.29860e-01         2.90000e+04
13   15    3.52117e-01         2.90000e+04
14   16    5.63833e-01         2.90000e+04
15   17    6.11127e-03         2.90000e+04
16   18    5.95966e-01         2.90000e+04
17   19    3.11005e-02         2.90000e+04
18   20    2.34957e-01         2.90000e+04
19   22    2.30133e-01         2.90000e+04
20    1    6.11261e-03         2.90000e+04
21    4    2.99226e-01         2.90000e+04
 3    6    2.49890e-01         2.90000e+04
 5    8    3.09373e-02         2.90000e+04
 7   10    2.34655e-01         2.90000e+04
 9   12    2.05620e-02         2.90000e+04
11   14    2.46646e-01         2.90000e+04
13   16    2.41871e-02         2.90000e+04
15   18    2.53857e-01         2.90000e+04
17   20    2.06100e-02         2.90000e+04
19    1    6.11159e-03         2.90000e+04
 2    3    6.11169e-03         2.90000e+04
 4    5    2.06159e-02         2.90000e+04
 6    7    2.34460e-01         2.90000e+04
 8    9    3.10265e-02         2.90000e+04
10   11    2.75304e-01         2.90000e+04
12   13    1.37797e-02         2.90000e+04
14   15    2.54490e-01         2.90000e+04
16   17    3.69977e-02         2.90000e+04
18   19    3.16729e-01         2.90000e+04
20   22    2.35348e-01         2.90000e+04
```

File: xt218.030.osi

```
41   22    2
  100.00000        10.00000         0.00000         0.00000
    0.00000        10.00000         0.00000         0.00000
   10.00000         0.00000         0.00000         0.00000
   10.00000        10.00000         0.00000         0.00000
   20.00000         0.00000         0.00000         0.00000
   20.00000        10.00000         0.00000         0.00000
   30.00000         0.00000         0.00000         0.00000
   30.00000        10.00000         0.00000         0.00000
   40.00000         0.00000         0.00000         0.00000
   40.00000        10.00000         0.00000         0.00000
   50.00000         0.00000         0.00000        -1.00000
   50.00000        10.00000         0.00000         0.00000
   60.00000         0.00000         0.00000         0.00000
   60.00000        10.00000         0.00000         0.00000
   70.00000         0.00000         0.00000         0.00000
   70.00000        10.00000         0.00000         0.00000
   80.00000         0.00000         0.00000         0.00000
   80.00000        10.00000         0.00000         0.00000
   90.00000         0.00000         0.00000         0.00000
   90.00000        10.00000         0.00000         0.00000
    0.00000         0.00000         0.00000         0.00000
  100.00000         0.00000         0.00000         0.00000
21    2    8.31711e-03         2.90000e+04
 3    4    1.76747e-01         2.90000e+04
 5    6    1.76750e-01         2.90000e+04
 7    8    1.76750e-01         2.90000e+04
 9   10    1.76750e-01         2.90000e+04
11   12    4.05129e-01         2.90000e+04
13   14    1.76752e-01         2.90000e+04
15   16    1.76753e-01         2.90000e+04
17   18    1.76758e-01         2.90000e+04
19   20    1.76748e-01         2.90000e+04
22    1    8.31711e-03         2.90000e+04
21    3    4.01492e-01         2.90000e+04
 2    4    8.31711e-03         2.90000e+04
 3    5    1.70997e-01         2.90000e+04
 4    6    1.76749e-01         2.90000e+04
 5    7    8.31709e-03         2.90000e+04
 6    8    4.04357e-01         2.90000e+04
 7    9    2.52202e-01         2.90000e+04
 8   10    5.29326e-01         2.90000e+04
 9   11    4.20445e-01         2.90000e+04
10   12    8.94166e-01         2.90000e+04
11   13    4.20225e-01         2.90000e+04
12   14    8.94185e-01         2.90000e+04
13   15    2.52193e-01         2.90000e+04
14   16    6.93126e-01         2.90000e+04
15   17    8.31709e-03         2.90000e+04
16   18    4.03985e-01         2.90000e+04
17   19    1.70996e-01         2.90000e+04
18   20    1.76748e-01         2.90000e+04
19   22    4.01509e-01         2.90000e+04
20    1    8.31711e-03         2.90000e+04
21    4    3.52288e-01         2.90000e+04
 3    6    3.52288e-01         2.90000e+04
 5    8    3.52288e-01         2.90000e+04
 7   10    3.52288e-01         2.90000e+04
 9   12    3.52288e-01         2.90000e+04
12   13    3.52288e-01         2.90000e+04
14   15    3.52288e-01         2.90000e+04
16   17    3.52288e-01         2.90000e+04
18   19    3.52288e-01         2.90000e+04
20   22    3.52288e-01         2.90000e+04
```

File: xt219.030.osi

```
41   22    2
  100.00000        10.00000         0.00000         0.00000
    0.00000        10.00000         0.00000         0.00000
   10.00000         0.00000         0.00000         0.00000
   10.00000        10.00000         0.00000         0.00000
   20.00000         0.00000         0.00000         0.00000
   20.00000        10.00000         0.00000         0.00000
   30.00000         0.00000         0.00000         0.00000
   30.00000        10.00000         0.00000         0.00000
   40.00000         0.00000         0.00000         0.00000
   40.00000        10.00000         0.00000         0.00000
   50.00000         0.00000         0.00000        -1.00000
   50.00000        10.00000         0.00000         0.00000
   60.00000         0.00000         0.00000         0.00000
   60.00000        10.00000         0.00000         0.00000
   70.00000         0.00000         0.00000         0.00000
   70.00000        10.00000         0.00000         0.00000
   80.00000         0.00000         0.00000         0.00000
   80.00000        10.00000         0.00000         0.00000
   90.00000         0.00000         0.00000         0.00000
   90.00000        10.00000         0.00000         0.00000
    0.00000         0.00000         0.00000         0.00000
  100.00000         0.00000         0.00000         0.00000
21     2    1.72414e-01         2.90000e+04
 3     4    1.72414e-01         2.90000e+04
 5     6    1.72414e-01         2.90000e+04
 7     8    1.72414e-01         2.90000e+04
 9    10    1.72414e-01         2.90000e+04
11    12    5.20898e-03         2.90000e+04
13    14    1.72414e-01         2.90000e+04
15    16    1.72414e-01         2.90000e+04
17    18    1.72414e-01         2.90000e+04
19    20    1.72414e-01         2.90000e+04
22     1    1.72414e-01         2.90000e+04
21     3    3.44828e-01         2.90000e+04
 2     4    1.72414e-01         2.90000e+04
 3     5    1.72414e-01         2.90000e+04
 4     6    3.44828e-01         2.90000e+04
 5     7    5.20898e-03         2.90000e+04
 6     8    5.17241e-01         2.90000e+04
 7     9    1.72414e-01         2.90000e+04
 8    10    6.89655e-01         2.90000e+04
 9    11    3.44828e-01         2.90000e+04
10    12    8.62069e-01         2.90000e+04
11    13    3.44828e-01         2.90000e+04
12    14    8.62069e-01         2.90000e+04
13    15    1.72414e-01         2.90000e+04
14    16    6.89655e-01         2.90000e+04
15    17    5.20898e-03         2.90000e+04
16    18    5.17241e-01         2.90000e+04
17    19    1.72414e-01         2.90000e+04
18    20    3.44828e-01         2.90000e+04
19    22    3.44828e-01         2.90000e+04
20     1    1.72414e-01         2.90000e+04
11    14    3.44828e-01         2.90000e+04
13    16    3.44828e-01         2.90000e+04
15    18    3.44828e-01         2.90000e+04
17    20    3.44828e-01         2.90000e+04
19     1    3.44828e-01         2.90000e+04
 2     3    3.44828e-01         2.90000e+04
 4     5    3.44828e-01         2.90000e+04
 6     7    3.44828e-01         2.90000e+04
 8     9    3.44828e-01         2.90000e+04
10    11    3.44828e-01         2.90000e+04
```

File: f02.020.osi

```
 2    3    2
   10.00000        10.00000         1.00000         0.00000         0.00000
    0.00000        10.00000         0.00000         0.00000         0.00000
   10.00000         0.00000         0.00000         0.00000         0.00000
 1    2    3.48259e-03    1.21284e-05         2.90000e+04
 1    3    1.12995e-04    1.27678e-08         2.90000e+04
```

File: f02a.030.osi

```
 2    3    2
   10.00000        10.00000         1.00000         0.00000         0.00000
    0.00000        10.00000         0.00000         0.00000         0.00000
   10.00000         0.00000         0.00000         0.00000         0.00000
 1    2    4.82522e-06    2.32828e-07         2.90000e+04
 1    3    3.42184e-03    1.17090e-01         2.90000e+04
```

File: f242.030.osi

```
20   15    3
    0.00000       480.00000         0.00000        -0.10000        -0.20000
  200.00000       480.00000         0.00000        -0.20000         0.00000
  400.00000       480.00000         0.00000        -0.10000         0.20000
    0.00000       360.00000         0.00000        -1.00000        -1.00000
  200.00000       360.00000         0.00000        -2.00000         0.00000
  400.00000       360.00000         0.00000        -1.00000         1.00000
    0.00000       240.00000         0.00000        -1.00000        -1.00000
  200.00000       240.00000         0.00000        -2.00000         0.00000
  400.00000       240.00000         0.00000        -1.00000         1.00000
    0.00000       120.00000         0.00000        -1.00000        -1.00000
  200.00000       120.00000         0.00000        -2.00000         0.00000
  400.00000       120.00000         0.00000        -1.00000         1.00000
    0.00000         0.00000         0.00000         0.00000         0.00000
  200.00000         0.00000         0.00000         0.00000         0.00000
  400.00000         0.00000         0.00000         0.00000         0.00000
 1    4    1.08239e-01    4.68634e-02         2.90000e+04
 2    5    2.02631e-02    1.64238e-03         2.90000e+04
 3    6    1.07683e-01    4.63830e-02         2.90000e+04
 4    7    9.99397e-02    3.99518e-02         2.90000e+04
 5    8    9.26555e-02    3.43401e-02         2.90000e+04
 6    9    1.00612e-01    4.04906e-02         2.90000e+04
 7   10    8.84410e-02    3.12873e-02         2.90000e+04
 8   11    1.76888e-01    1.25159e-01         2.90000e+04
 9   12    8.84414e-02    3.12875e-02         2.90000e+04
10   13    1.30556e-01    6.81788e-02         2.90000e+04
11   14    2.61121e-01    2.72735e-01         2.90000e+04
12   15    1.30556e-01    6.81800e-02         2.90000e+04
 1    2    2.02630e-02    4.10592e-04         2.90000e+04
 2    3    2.02630e-02    4.10590e-04         2.90000e+04
 4    5    2.02630e-02    4.10592e-04         2.90000e+04
 5    6    2.02631e-02    4.10593e-04         2.90000e+04
 7    8    2.02630e-02    4.10592e-04         2.90000e+04
 8    9    2.02631e-02    4.10593e-04         2.90000e+04
10   11    2.02630e-02    4.10592e-04         2.90000e+04
11   12    2.02631e-02    4.10592e-04         2.90000e+04
```

File: col10.030.osi

```
10   11    1
   10.00000        0.00000    -3600.00000        0.00000        0.00000
    9.00000        0.00000        0.00000        0.00000        0.00000
    8.00000        0.00000        0.00000        0.00000        0.00000
    7.00000        0.00000        0.00000        0.00000        0.00000
    6.00000        0.00000        0.00000        0.00000        0.00000
    5.00000        0.00000        0.00000        0.00000        0.00000
    4.00000        0.00000        0.00000        0.00000        0.00000
    3.00000        0.00000        0.00000        0.00000        0.00000
    2.00000        0.00000        0.00000        0.00000        0.00000
    1.00000        0.00000        0.00000        0.00000        0.00000
    0.00000        0.00000        0.00000        0.00000        0.00000
 1    2    2.84003e-01    4.03290e-01        2.90000e+04
 2    3    4.97639e-01    1.23822e+00        2.90000e+04
 3    4    6.55528e-01    2.14859e+00        2.90000e+04
 4    5    8.29598e-01    3.44117e+00        2.90000e+04
 5    6    8.95083e-01    4.00587e+00        2.90000e+04
 6    7    1.07745e+00    5.80449e+00        2.90000e+04
 7    8    1.02099e+00    5.21209e+00        2.90000e+04
 8    9    1.15374e+00    6.65562e+00        2.90000e+04
 9   10    1.17987e+00    6.96044e+00        2.90000e+04
10   11    1.20922e+00    7.31117e+00        2.90000e+04
```

File: f11b.050.osi

```
 3    4    2
    0.00000     1000.00000        0.00000       -4.30000        0.00000
 1000.00000     1000.00000        0.00000       -4.30000        0.00000
    0.00000        0.00000        0.00000        0.00000        0.00000
 1000.00000        0.00000        0.00000        0.00000        0.00000
 1    3    1.10116e+00    2.42509e+01        2.90000e+04
 2    4    1.09937e+00    2.41724e+01        2.90000e+04
 1    2    9.66268e-01    1.86735e+01        2.90000e+04
```

File: f11b.050.osi

```
 3    4    2
    0.00000       10.00000        0.00000   -30000.00000        0.00000
   10.00000       10.00000        0.00000   -30000.00000        0.00000
    0.00000        0.00000        0.00000        0.00000        0.00000
   10.00000        0.00000        0.00000        0.00000        0.00000
 1    3    1.34824e+00    3.63549e+01        2.90000e+04
 2    4    1.31828e+00    3.47572e+01        2.90000e+04
 1    2    3.91912e-01    3.07189e+00        2.90000e+04
```

File: g53.030.osi

```
   60   45  180
 1.66667e+02  1.66667e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 1.66667e+02  3.33333e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 1.66667e+02  5.00000e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 1.66667e+02  6.66667e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 1.66667e+02  8.33333e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 3.33333e+02  1.66667e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 3.33333e+02  3.33333e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 3.33333e+02  5.00000e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 3.33333e+02  6.66667e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 3.33333e+02  8.33333e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 5.00000e+02  1.66667e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 5.00000e+02  3.33333e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 5.00000e+02  5.00000e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 5.00000e+02  6.66667e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 5.00000e+02  8.33333e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 6.66667e+02  1.66667e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 6.66667e+02  3.33333e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 6.66667e+02  5.00000e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 6.66667e+02  6.66667e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 6.66667e+02  8.33333e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 8.33333e+02  1.66667e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 8.33333e+02  3.33333e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 8.33333e+02  5.00000e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 8.33333e+02  6.66667e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 8.33333e+02  8.33333e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 0.00e+00  1.66667e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 0.00e+00  3.33333e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 0.00e+00  5.00000e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 0.00e+00  6.66667e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 0.00e+00  8.33333e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 1.00000e+03  1.66667e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 1.00000e+03  3.33333e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 1.00000e+03  5.00000e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 1.00000e+03  6.66667e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 1.00000e+03  8.33333e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 1.66667e+02 0.0e+00 0.0e+00 0.0e+00  3.20e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 3.33333e+02 0.0e+00 0.0e+00 0.0e+00  3.20e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 5.00000e+02 0.0e+00 0.0e+00 0.0e+00  3.20e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 6.66667e+02 0.0e+00 0.0e+00 0.0e+00  3.20e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 8.33333e+02 0.0e+00 0.0e+00 0.0e+00  3.20e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 1.66667e+02  1.00000e+03 0.0e+00 0.0e+00 -3.20e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 3.33333e+02  1.00000e+03 0.0e+00 0.0e+00 -3.20e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 5.00000e+02  1.00000e+03 0.0e+00 0.0e+00 -3.20e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 6.66667e+02  1.00000e+03 0.0e+00 0.0e+00 -3.20e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 8.33333e+02  1.00000e+03 0.0e+00 0.0e+00 -3.20e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
1    2  3.06604e-01  1.88011e-01  4.70030e-01  8.46051e+06 2.90e+04 1.20e+04 0.0 0.0 90.0
2    3  9.90027e-02  1.96032e-02  4.90078e-02  8.82140e+05 2.90e+04 1.20e+04 0.0 0.0 90.0
3    4  9.90030e-02  1.96033e-02  4.90083e-02  8.82147e+05 2.90e+04 1.20e+04 0.0 0.0 90.0
4    5  5.01212e-01  5.02428e-01  1.25607e+00  2.26092e+07 2.90e+04 1.20e+04 0.0 0.0 90.0
6    7  7.51824e-01  1.13047e+00  2.82617e+00  5.08710e+07 2.90e+04 1.20e+04 0.0 0.0 90.0
7    8  3.34141e-01  2.23300e-01  5.58253e-01  1.00485e+07 2.90e+04 1.20e+04 0.0 0.0 90.0
8    9  3.49880e-01  2.44833e-01  6.12082e-01  1.10175e+07 2.90e+04 1.20e+04 0.0 0.0 90.0
9   10  6.75791e-01  9.13389e-01  2.28347e+00  4.11024e+07 2.90e+04 1.20e+04 0.0 0.0 90.0
11   12  8.30955e-01  1.38097e+00  3.45243e+00  6.21437e+07 2.90e+04 1.20e+04 0.0 0.0 90.0
12   13  3.34142e-01  2.23301e-01  5.58254e-01  1.00486e+07 2.90e+04 1.20e+04 0.0 0.0 90.0
13   14  3.34142e-01  2.23300e-01  5.58252e-01  1.00486e+07 2.90e+04 1.20e+04 0.0 0.0 90.0
14   15  9.56163e-01  1.82850e+00  4.57123e+00  8.22819e+07 2.90e+04 1.20e+04 0.0 0.0 90.0
16   17  1.12774e+00  2.54355e+00  6.35890e+00  1.14460e+08 2.90e+04 1.20e+04 0.0 0.0 90.0
17   18  3.34141e-01  2.23300e-01  5.58253e-01  1.00485e+07 2.90e+04 1.20e+04 0.0 0.0 90.0
18   19  5.01212e-01  5.02426e-01  1.25607e+00  2.26092e+07 2.90e+04 1.20e+04 0.0 0.0 90.0
19   20  1.12774e+00  2.54355e+00  6.35889e+00  1.14460e+08 2.90e+04 1.20e+04 0.0 0.0 90.0
21   22  4.80590e-02  4.61931e-03  1.15482e-02  2.07869e+05 2.90e+04 1.20e+04 0.0 0.0 90.0
22   23  1.30369e-02  3.39921e-04  8.49804e-04  1.52964e+04 2.90e+04 1.20e+04 0.0 0.0 90.0
23   24  1.30369e-02  3.39921e-04  8.49804e-04  1.52964e+04 2.90e+04 1.20e+04 0.0 0.0 90.0
24   25  9.90044e-02  1.96037e-02  4.90092e-02  8.82166e+05 2.90e+04 1.20e+04 0.0 0.0 90.0
1    6  3.34141e-01  2.23299e-01  5.58249e-01  1.00485e+07 2.90e+04 1.20e+04 0.0 0.0 0.0
2    7  9.90039e-02  1.96037e-02  4.90092e-02  8.82162e+05 2.90e+04 1.20e+04 0.0 0.0 0.0
3    8  1.90303e+00  7.24303e+00  1.81076e+01  3.25936e+08 2.90e+04 1.20e+04 0.0 0.0 0.0
4    9  2.22761e-01  9.92445e-02  2.48111e-01  4.46598e+06 2.90e+04 1.20e+04 0.0 0.0 0.0
5   10  9.90042e-02  1.96036e-02  4.90089e-02  8.82164e+05 2.90e+04 1.20e+04 0.0 0.0 0.0
6   11  1.30369e-02  3.39921e-04  8.49804e-04  1.52964e+04 2.90e+04 1.20e+04 0.0 0.0 0.0
7   12  1.95558e-02  7.64853e-04  1.91215e-03  3.44186e+04 2.90e+04 1.20e+04 0.0 0.0 0.0
8   13  2.57243e+00  1.32349e+01  3.30870e+01  5.95566e+08 2.90e+04 1.20e+04 0.0 0.0 0.0
9   14  2.93342e-02  1.72098e-03  4.30244e-03  7.74439e+04 2.90e+04 1.20e+04 0.0 0.0 0.0
10   15  1.30369e-02  3.39921e-04  8.49804e-04  1.52964e+04 2.90e+04 1.20e+04 0.0 0.0 0.0
11   16  1.30369e-02  3.39921e-04  8.49804e-04  1.52964e+04 2.90e+04 1.20e+04 0.0 0.0 0.0
12   17  1.95558e-02  7.64853e-04  1.91215e-03  3.44186e+04 2.90e+04 1.20e+04 0.0 0.0 0.0
13   18  2.84659e+00  1.62062e+01  4.05155e+01  7.29278e+08 2.90e+04 1.20e+04 0.0 0.0 0.0
14   19  2.93342e-02  1.72098e-03  4.30244e-03  7.74439e+04 2.90e+04 1.20e+04 0.0 0.0 0.0
15   20  1.30369e-02  3.39921e-04  8.49804e-04  1.52964e+04 2.90e+04 1.20e+04 0.0 0.0 0.0
16   21  5.01212e-01  5.02427e-01  1.25607e+00  2.26093e+07 2.90e+04 1.20e+04 0.0 0.0 0.0
17   22  1.95558e-02  7.64853e-04  1.91215e-03  3.44186e+04 2.90e+04 1.20e+04 0.0 0.0 0.0
```

```
18    23  2.00900e+00  8.07209e+00  2.01803e+01  3.63246e+08 2.90e+04 1.20e+04 0.0 0.0 0.0
19    24  4.40016e-02  3.87228e-03  9.68072e-03  1.74252e+05 2.90e+04 1.20e+04 0.0 0.0 0.0
20    25  5.01212e-01  5.02427e-01  1.25607e+00  2.26091e+07 2.90e+04 1.20e+04 0.0 0.0 0.0
26     1  4.36148e-01  3.80450e-01  9.51126e-01  1.71202e+07 2.90e+04 1.20e+04 0.0 0.0 0.0
27     2  1.29118e-01  3.33426e-02  8.33562e-02  1.50041e+06 2.90e+04 1.20e+04 0.0 0.0 0.0
28     3  1.12774e+00  2.54356e+00  6.35890e+00  1.14460e+08 2.90e+04 1.20e+04 0.0 0.0 0.0
29     4  2.20348e-01  9.71061e-02  2.42763e-01  4.36975e+06 2.90e+04 1.20e+04 0.0 0.0 0.0
30     5  2.22761e-01  9.92445e-02  2.48110e-01  4.46598e+06 2.90e+04 1.20e+04 0.0 0.0 0.0
21    31  5.01213e-01  5.02427e-01  1.25607e+00  2.26093e+07 2.90e+04 1.20e+04 0.0 0.0 0.0
22    32  1.30369e-02  3.39921e-04  8.49804e-04  1.52964e+04 2.90e+04 1.20e+04 0.0 0.0 0.0
23    33  1.12774e+00  2.54355e+00  6.35890e+00  1.14460e+08 2.90e+04 1.20e+04 0.0 0.0 0.0
24    34  4.40016e-02  3.87226e-03  9.68064e-03  1.74252e+05 2.90e+04 1.20e+04 0.0 0.0 0.0
25    35  6.08536e-01  7.40631e-01  1.85158e+00  3.33284e+07 2.90e+04 1.20e+04 0.0 0.0 0.0
36     1  5.01213e-01  5.02427e-01  1.25607e+00  2.26093e+07 2.90e+04 1.20e+04 0.0 0.0 90.0
37     6  7.51824e-01  1.13047e+00  2.82617e+00  5.08710e+07 2.90e+04 1.20e+04 0.0 0.0 90.0
38    11  6.44595e-01  8.31007e-01  2.07752e+00  3.73953e+07 2.90e+04 1.20e+04 0.0 0.0 90.0
39    16  8.80699e-01  1.55127e+00  3.87815e+00  6.98068e+07 2.90e+04 1.20e+04 0.0 0.0 90.0
40    21  3.34142e-01  2.23301e-01  5.58254e-01  1.00486e+07 2.90e+04 1.20e+04 0.0 0.0 90.0
5    41  6.34482e-01  8.05135e-01  2.01284e+00  3.62311e+07 2.90e+04 1.20e+04 0.0 0.0 90.0
10    42  7.51821e-01  1.13047e+00  2.82617e+00  5.08712e+07 2.90e+04 1.20e+04 0.0 0.0 90.0
15    43  7.99307e-01  1.27778e+00  3.19445e+00  5.75002e+07 2.90e+04 1.20e+04 0.0 0.0 90.0
20    44  1.12774e+00  2.54355e+00  6.35889e+00  1.14460e+08 2.90e+04 1.20e+04 0.0 0.0 90.0
25    45  3.34142e-01  2.23301e-01  5.58253e-01  1.00486e+07 2.90e+04 1.20e+04 0.0 0.0 90.0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
1 1 1 1 0 1
1 1 1 1 0 1
1 1 1 1 0 1
1 1 1 1 0 1
1 1 1 1 0 1
1 1 1 1 0 1
1 1 1 1 0 1
1 1 1 1 0 1
1 1 1 1 0 1
1 1 1 1 0 1
1 0 1 0 1 1
1 0 1 0 1 1
1 0 1 0 1 1
1 0 1 0 1 1
1 0 1 0 1 1
1 0 1 0 1 1
1 0 1 0 1 1
1 0 1 0 1 1
1 0 1 0 1 1
1 0 1 0 1 1
```

File: g5.030.osi

```
   60   45  180
 1.66667e+02  1.66667e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 1.66667e+02  3.33333e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 1.66667e+02  5.00000e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 1.66667e+02  6.66667e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 1.66667e+02  8.33333e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 3.33333e+02  1.66667e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 3.33333e+02  3.33333e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 3.33333e+02  5.00000e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 3.33333e+02  6.66667e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 3.33333e+02  8.33333e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 5.00000e+02  1.66667e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 5.00000e+02  3.33333e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 5.00000e+02  5.00000e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 5.00000e+02  6.66667e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 5.00000e+02  8.33333e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 6.66667e+02  1.66667e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 6.66667e+02  3.33333e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 6.66667e+02  5.00000e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 6.66667e+02  6.66667e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 6.66667e+02  8.33333e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 8.33333e+02  1.66667e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 8.33333e+02  3.33333e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 8.33333e+02  5.00000e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 8.33333e+02  6.66667e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 8.33333e+02  8.33333e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 0.00e+00  1.66667e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 0.00e+00  3.33333e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 0.00e+00  5.00000e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 0.00e+00  6.66667e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 0.00e+00  8.33333e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 1.00000e+03  1.66667e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 1.00000e+03  3.33333e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 1.00000e+03  5.00000e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 1.00000e+03  6.66667e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 1.00000e+03  8.33333e+02 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 1.66667e+02 0.0e+00 0.0e+00 0.0e+00  3.20e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 3.33333e+02 0.0e+00 0.0e+00 0.0e+00  3.20e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 5.00000e+02 0.0e+00 0.0e+00 0.0e+00  3.20e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 6.66667e+02 0.0e+00 0.0e+00 0.0e+00  3.20e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 8.33333e+02 0.0e+00 0.0e+00 0.0e+00  3.20e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 1.66667e+02  1.00000e+03 0.0e+00 0.0e+00 -3.20e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 3.33333e+02  1.00000e+03 0.0e+00 0.0e+00 -3.20e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 5.00000e+02  1.00000e+03 0.0e+00 0.0e+00 -3.20e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 6.66667e+02  1.00000e+03 0.0e+00 0.0e+00 -3.20e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
 8.33333e+02  1.00000e+03 0.0e+00 0.0e+00 -3.20e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00
1    2  7.96818e-01  1.26984e+00  3.17460e+00  3.17460e+00 2.90e+04 1.20e+04 0.0 0.0 90.0
2    3  5.31212e-01  5.64371e-01  1.41093e+00  1.41093e+00 2.90e+04 1.20e+04 0.0 0.0 90.0
3    4  5.55228e-01  6.16556e-01  1.54139e+00  1.54139e+00 2.90e+04 1.20e+04 0.0 0.0 90.0
4    5  6.02914e-01  7.27009e-01  1.81753e+00  1.81753e+00 2.90e+04 1.20e+04 0.0 0.0 90.0
6    7  6.43887e-01  8.29181e-01  2.07296e+00  2.07296e+00 2.90e+04 1.20e+04 0.0 0.0 90.0
7    8  3.11395e-01  1.93933e-01  4.84832e-01  4.84832e-01 2.90e+04 1.20e+04 0.0 0.0 90.0
8    9  2.88560e-01  1.66532e-01  4.16331e-01  4.16331e-01 2.90e+04 1.20e+04 0.0 0.0 90.0
9   10  5.18087e-01  5.36828e-01  1.34207e+00  1.34207e+00 2.90e+04 1.20e+04 0.0 0.0 90.0
11   12  2.66983e+00  1.42561e+01  3.56402e+01  3.56402e+01 2.90e+04 1.20e+04 0.0 0.0 90.0
12   13  2.94643e+00  1.73630e+01  4.34073e+01  4.34073e+01 2.90e+04 1.20e+04 0.0 0.0 90.0
13   14  3.23476e+00  2.09274e+01  5.23187e+01  5.23187e+01 2.90e+04 1.20e+04 0.0 0.0 90.0
14   15  2.72914e+00  1.48964e+01  3.72409e+01  3.72409e+01 2.90e+04 1.20e+04 0.0 0.0 90.0
16   17  6.51944e-01  8.50059e-01  2.12515e+00  2.12515e+00 2.90e+04 1.20e+04 0.0 0.0 90.0
17   18  4.34629e-01  3.77804e-01  9.44508e-01  9.44508e-01 2.90e+04 1.20e+04 0.0 0.0 90.0
18   19  5.29026e-01  5.59738e-01  1.39934e+00  1.39934e+00 2.90e+04 1.20e+04 0.0 0.0 90.0
19   20  7.93541e-01  1.25941e+00  3.14854e+00  3.14854e+00 2.90e+04 1.20e+04 0.0 0.0 90.0
21   22  4.34628e-01  3.77804e-01  9.44508e-01  9.44508e-01 2.90e+04 1.20e+04 0.0 0.0 90.0
22   23  4.32840e-01  3.74699e-01  9.36748e-01  9.36748e-01 2.90e+04 1.20e+04 0.0 0.0 90.0
23   24  4.32839e-01  3.74699e-01  9.36747e-01  9.36747e-01 2.90e+04 1.20e+04 0.0 0.0 90.0
24   25  4.92000e-01  4.84126e-01  1.21031e+00  1.21031e+00 2.90e+04 1.20e+04 0.0 0.0 90.0
1    6  1.93168e-01  7.46274e-02  1.86568e-01  1.86568e-01 2.90e+04 1.20e+04 0.0 0.0 0.0
2    7  6.51944e-01  8.50059e-01  2.12515e+00  2.12515e+00 2.90e+04 1.20e+04 0.0 0.0 0.0
3    8  2.99568e-01  1.79483e-01  4.48706e-01  4.48706e-01 2.90e+04 1.20e+04 0.0 0.0 0.0
4    9  7.96820e-01  1.26984e+00  3.17462e+00  3.17462e+00 2.90e+04 1.20e+04 0.0 0.0 0.0
5   10  2.89752e-01  1.67912e-01  4.19780e-01  4.19780e-01 2.90e+04 1.20e+04 0.0 0.0 0.0
6   11  2.36093e-01  1.11480e-01  2.78700e-01  2.78700e-01 2.90e+04 1.20e+04 0.0 0.0 0.0
7   12  5.31212e-01  5.64374e-01  1.41094e+00  1.41094e+00 2.90e+04 1.20e+04 0.0 0.0 0.0
8   13  3.54141e-01  2.50832e-01  6.27082e-01  6.27082e-01 2.90e+04 1.20e+04 0.0 0.0 0.0
9   14  5.31212e-01  5.64374e-01  1.41094e+00  1.41094e+00 2.90e+04 1.20e+04 0.0 0.0 0.0
10   15  2.36093e-01  1.11481e-01  2.78701e-01  2.78701e-01 2.90e+04 1.20e+04 0.0 0.0 0.0
11   16  1.28778e-01  3.31674e-02  8.29186e-02  8.29186e-02 2.90e+04 1.20e+04 0.0 0.0 0.0
12   17  4.01930e-01  3.23096e-01  8.07739e-01  8.07739e-01 2.90e+04 1.20e+04 0.0 0.0 0.0
13   18  2.89752e-01  1.67912e-01  4.19782e-01  4.19782e-01 2.90e+04 1.20e+04 0.0 0.0 0.0
14   19  3.54141e-01  2.50832e-01  6.27082e-01  6.27082e-01 2.90e+04 1.20e+04 0.0 0.0 0.0
15   20  1.57395e-01  4.95464e-02  1.23866e-01  1.23866e-01 2.90e+04 1.20e+04 0.0 0.0 0.0
16   21  4.09021e-01  3.34597e-01  8.36488e-01  8.36488e-01 2.90e+04 1.20e+04 0.0 0.0 0.0
17   22  5.38840e-01  5.80699e-01  1.45175e+00  1.45175e+00 2.90e+04 1.20e+04 0.0 0.0 0.0
```

```
18    23  4.49692e-01  4.04446e-01  1.01112e+00  1.01112e+00 2.90e+04 1.20e+04 0.0 0.0 0.0
19    24  5.74285e-01  6.59608e-01  1.64902e+00  1.64902e+00 2.90e+04 1.20e+04 0.0 0.0 0.0
20    25  3.54140e-01  2.50832e-01  6.27079e-01  6.27079e-01 2.90e+04 1.20e+04 0.0 0.0 0.0
26     1  1.93167e-01  7.46274e-02  1.86568e-01  1.86568e-01 2.90e+04 1.20e+04 0.0 0.0 0.0
27     2  5.31212e-01  5.64373e-01  1.41094e+00  1.41094e+00 2.90e+04 1.20e+04 0.0 0.0 0.0
28     3  1.93167e-01  7.46274e-02  1.86568e-01  1.86568e-01 2.90e+04 1.20e+04 0.0 0.0 0.0
29     4  5.31212e-01  5.64374e-01  1.41093e+00  1.41093e+00 2.90e+04 1.20e+04 0.0 0.0 0.0
30     5  2.89751e-01  1.67912e-01  4.19780e-01  4.19780e-01 2.90e+04 1.20e+04 0.0 0.0 0.0
21    31  4.34628e-01  3.77804e-01  9.44508e-01  9.44508e-01 2.90e+04 1.20e+04 0.0 0.0 0.0
22    32  3.84222e-01  2.95253e-01  7.38132e-01  7.38132e-01 2.90e+04 1.20e+04 0.0 0.0 0.0
23    33  3.54141e-01  2.50832e-01  6.27079e-01  6.27079e-01 2.90e+04 1.20e+04 0.0 0.0 0.0
24    34  5.31212e-01  5.64373e-01  1.41093e+00  1.41093e+00 2.90e+04 1.20e+04 0.0 0.0 0.0
25    35  4.32838e-01  3.74699e-01  9.36748e-01  9.36748e-01 2.90e+04 1.20e+04 0.0 0.0 0.0
36     1  6.49261e-01  8.43077e-01  2.10769e+00  2.10769e+00 2.90e+04 1.20e+04 0.0 0.0 90.0
37     6  5.31213e-01  5.64374e-01  1.41093e+00  1.41093e+00 2.90e+04 1.20e+04 0.0 0.0 90.0
38    11  1.41113e+00  3.98259e+00  9.95646e+00  9.95646e+00 2.90e+04 1.20e+04 0.0 0.0 90.0
39    16  6.51944e-01  8.50059e-01  2.12515e+00  2.12515e+00 2.90e+04 1.20e+04 0.0 0.0 90.0
40    21  4.34628e-01  3.77804e-01  9.44508e-01  9.44508e-01 2.90e+04 1.20e+04 0.0 0.0 90.0
5    41  5.31212e-01  5.64373e-01  1.41093e+00  1.41093e+00 2.90e+04 1.20e+04 0.0 0.0 90.0
10    42  4.39568e-01  3.86441e-01  9.66102e-01  9.66102e-01 2.90e+04 1.20e+04 0.0 0.0 90.0
15    43  1.41762e+00  4.01926e+00  1.00481e+01  1.00481e+01 2.90e+04 1.20e+04 0.0 0.0 90.0
20    44  7.53777e-01  1.13636e+00  2.84089e+00  2.84089e+00 2.90e+04 1.20e+04 0.0 0.0 90.0
25    45  5.25111e-01  5.51486e-01  1.37871e+00  1.37871e+00 2.90e+04 1.20e+04 0.0 0.0 90.0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
1 1 1 1 0 1
1 1 1 1 0 1
1 1 1 1 0 1
1 1 1 1 0 1
1 1 1 1 0 1
1 1 1 1 0 1
1 1 1 1 0 1
1 1 1 1 0 1
1 1 1 1 0 1
1 1 1 1 0 1
1 0 1 0 1 1
1 0 1 0 1 1
1 0 1 0 1 1
1 0 1 0 1 1
1 0 1 0 1 1
1 0 1 0 1 1
1 0 1 0 1 1
1 0 1 0 1 1
1 0 1 0 1 1
1 0 1 0 1 1
```

REFERENCES

Adeli, Hojjat, and Kumar, Sanjay "Concurrent Structural Optimization on Massively Parallel Supercomputer," Journal of Structural Engineering, Vol. 121, No. 11, pp. 1588–1597, November 1995

Arbel, Ami, Exploring Interior–Point Linear Programming, The MIT Press, Cambridge, Massachussetts, 1993

Arora, J. S., Introduction to Optimum Design, McGraw Hill, New York, 1989

Arora, J. S. and Thanedar, P. B., "Computational Methods for Optimum Design of Large Complex Systems," Computational Mechanics, Vol. 1, pp 221–242, 1986

Arora, J. S. and Belegundu, A. D., "Structural Optimization by Mathematical Programming Methods," AIAA Journal, Vol. 22, No. 6, pp 854–856, 1984

Arora, J. S. and Haug, E. J., "Methods of Design Sensitivity Analysis in Structural Optimization," AIAA Journal Vol. 17, No. 9, pp 970–974, 1979

Belegundu, A. D. and Arora, J. S., "A Study of Mathematical Programming Methods for Structural Optimization," International Journal for Numerical Methods in Engineering, Vol. 21, 1583–1599, 1985

Bixby, Robert E., Gregory, John W., Lustig, Irvin J., Marsten, Roy E., and Shano, David F., "Very Large–Scale Linear Programming: A Case Study in Combining Interior Point and Simplex Methods," Rutcor Research Report #34–91, June, 1991

Boggs, P.T., Kearsley, A.T., Tolle, J.W., "A Practical Algorithm for General Large Scale Nonlinear Optimization Problems," SIAM Journal on Optimization, Vol. 9, No. 3, pp 755–778, 1999

Boggs, Paul T., Domich, Paul D., and Rogers, Janet E., "An Interior Point Method for General Large Scale Quadratic Programming Problems," Annals of Operations Research, Vol. 62, pp 419–437, 1996

Byrd, Richard H., Hribar, Mary E., and Nocedal, Jorge, "An Interior Point Algorithm for Large–Scale Nonlinear Programming," SIAM Journal on Optimization, Vol. 9, No. 3, pp 877–900, 1999

Camp, C., Pezeshk, S., and Cao, G., "Optimized Design of Two Dimensional Structures Using a Genetic Algorithm," Journal of Structural Engineering, pp 551–559, May 1998

Cassis, Juan H., and Schmit, Lucien A., "Optimum Structural Design with Dynamic Constraints," Journal of the Structural Division, ASCE, ST10, pp 2053–2071, October 1976

Chen, C. Y. and Schmit, L. A., "Minimum Weight Design of Elastic Redundant Trusses Under Multiple Loading Conditions," AIAA Journal, Vol. 10, No. 2, pp 155–162, 1972

Haug and Arora, Applied Optimal Design, John Wiley & Sons, New York, 1979

Hornlein, H,, Schittkowski, K., Software Systems for Structural Optimization, International Series of Numerical Mathematics, Vol. 110, Birkhauser, 1993

Jarre, F, Kocvara, M., and Zowe, J., "Optimal Truss Design by Interior–Point Methods," SIAM Journal on Optimization, Vol. 8, No. 4, pp 1084–1107, 1998

Karmarkar, N., "A New Polynomial Time Algorithm for Linear Programming," Combinatorica, Vol. 4, pp 373–395, 1984

Kelly, C. T., Iterative Methods for Optimization, Frontiers in Applied Mathematics, Society for Industrial and Applied Mathematics, Philadelphia, 1999

Kirsch, Uri, Optimum Structural Design, McGraw–Hill Book Company, New York, 1981

Lin, Chih–Jen and More, Jorge J. "Newton's Method for Large Bound–Constrained Optimization Problems," SIAM Journal on Optimization, Vol. 9, No. 4, pp 1100–1127, 1999

More, J. J. and Wright, S. J., Optimization Software Guide, Frontiers in Applied Mathematics, Vol. 14, Society for Industrial and Applied Mathematics, Philadelphia, 1993

Pedersen, P., "On the Optimal Layout of Multi–Purpose Trusses," Computers and Structures, Vol. 2, pp 695–712, 1972

Pedersen, P., "Optimal Joint Positions for Space Trusses," Journal of the Structural Division, ASCE, Vol. 99 ST10, pp 2459–2477, December 1973

Prasad, B. and Haftka, R. "Optimal Structural Design with Plate Finite Elements," Journal of the Structural Division, Vol. 105, No. 11, pp 2367–2382, November 1979

Reinschmidt, K. F. and Russel, A. D., "Applications of Linear Programming in Structural Layout and Optimization," Computers and Structures, Vol. 4, pp 855–869, 1974

Rozvany, G. I. N. and Zhou, M., "Optimality Criteria Methods for Large Structural Systems," , A Research Project of the Deutsche Forschungsgemeinschaft, March 1993

Schittkowski, K., "On Convergence of a Sequential Quadratic Programming Method with an Augmented Lagrangian Line Search Function," Math. Operations Forsch. U. Statist., Vol. 14 No. 2, pp 197–216

Schittkowski, K., Zillober, C., and Zotemantel, R., "Numerical Comparison of Nonlinear Programming Algorithms for Structural Optimization," Structural Optimization, Vol. 7, 1994

Sheu, C. Y. and Schmit, L. A., "Minimum Weight of Elastic Redundant Trusses Under Multiple Static Loading Conditions," AIAA Journal, Vol, 10, No. 2, February 1972

Spillers, W. R., "Optimal Design for Plate Buckling," Journal of Structural Engineering, Vol. 116, No. 3, pp 850–858, March 1990

Spillers, W. R., Iterative Structural Design, North Holland Publishing, 1975

Svanberg, K., New Directions in Optimum Structural Design, edited by Artek, Gallagher, Ragsdell and Zienkiewicz, John Wiley & Sons, ISBN 0–471–90291–8 pp 327–341, 1984

Sved, G., and Ginos, Z., "Structural Optimization Under Multiple Loading," International Journal of Mechanical Sciences, Vol. 10, pp 803–805, 1968

Taha, H. A., Operations Research An Introduction, Fifth Edition, MacMillan Publishing Company, New York, 1992

Thanedar, P. B., Arora, J. S., Tseng, O. K., and Park, G. J., "Performance of Some SQP Algorithms on Structural Design Problems," International Journal for Numerical Methods in Engineering, Vol. 23, pp 2187–2203, 1986

Thanedar, P. B., Arora, J. S., Li, G. Y., and Lin, T. C., "Robustness, Generality and Efficiency of Optimization Algorithms for Practical Applications," Structural Optimization, Vol. 2, pp 203–212, 1990

Timoshenko, S., Theory of Plates and Shells, McGraw Hill Book Company, New York, 1940

Vanderbei, R. J., "Affine – Scaling for Linear Programs with Free Variables," Mathematical Programming 43, pp 31–44, 1989

Vanderplatts, G. N. and Moses, F., "Automated Design of Trusses for Optimum Geometry," Journal of the Structural Division, ASCE, Vol. 98 ST3, pp 671–690, March 1972

Vanderplaats, G. N., "Structural Optimization – Past, Present and Future," AIAA Journal, Vol. 19, pp 992–1000, 1982

Wang, C., and Salmon, C., Introductory Structural Analysis, Prentice Hall, New Jersey, 1984

Wright, Stephen J., Primal–Dual Interior–Point Methods, Society for Industrial and Applied Mathematics, 1997

Xie, Y. M., and Steven, G. P,. "A Simple Approach to Structural Frequency Optimization," Computers and Structures, Vol. 53, No. 6, pp 1487–1491, 1994

Xu, Lei, Sherbourne, Archie N., and Grierson, Donald E., "Optimal Cost Design of Semi–Rigid, Low–Rise Industrial Frames," Engineering Journal, Third Quarter, pp 87–97, 1995