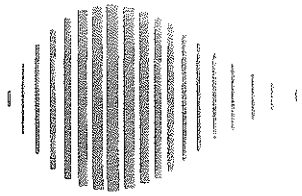


NATIONAL CENTER FOR EARTHQUAKE
ENGINEERING RESEARCH
State University of New York at Buffalo

SYMBOLIC MANIPULATION PROGRAM (SMP) — ALGEBRAIC CODES FOR TWO AND THREE DIMENSIONAL FINITE ELEMENT FORMULATIONS

by

X. Lee and G. Dasgupta
Department of Civil Engineering and
Engineering Mechanics
Columbia University
New York, NY 10027-6699

Technical Report NCEER-87-0006

November 9, 1987

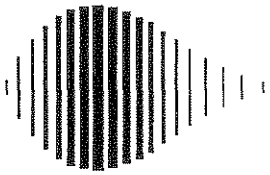
This research was conducted at Columbia University and was partially supported by the National Science Foundation under Grant No. ECE 86-07591.

NOTICE

This report was prepared by Columbia University as a result of research sponsored by the National Center for Earthquake Engineering Research (NCEER) and the National Science Foundation. Neither NCEER, associates of NCEER, its sponsors, Columbia University, nor any person acting on their behalf:

a. makes any warranty, express or implied, with respect to the use of any information, apparatus, method, or process disclosed in this report or that such use may not infringe upon privately owned rights; or

b. assumes any liabilities of whatsoever kind with respect to the use of, or for damages resulting from the use of, any information, apparatus, method or process disclosed in this report.



SYMBOLIC MANIPULATION PROGRAM (SMP) - ALGEBRAIC CODES FOR TWO AND THREE DIMENSIONAL FINITE ELEMENT FORMULATIONS

by

X. Lee[1] and G. Dasgupta[2]

November 9, 1987

Technical Report NCEER-87-0006

NCEER Contract Number 86-4023

NSF Master Contract Number ECE 86-07591

and

NSF Grant Number ECE-85-15249

1 Graduate Research Assistant, Dept. of Civil Engineering and Engineering Mechanics, Columbia University

2 Associate Professor, Dept. of Civil Engineering and Engineering Mechanics, Columbia University

NATIONAL CENTER FOR EARTHQUAKE ENGINEERING RESEARCH
State University of New York at Buffalo
Red Jacket Quadrangle, Buffalo, NY 14261

ABSTRACT

A computer algebra package entitled the Symbolic Manipulation Program (SMP) has been briefly reviewed. Its capabilities of handling the Finite Element Method (FEM) and Matrix Structure Analysis (MSA) are studied. The applications of using SMP in nonlinear FEM and reliability analysis of the nondeterministic systems are demonstrated. The possibilities of translating the output from SMP into higher level languages such as C and FORTRAN are also investigated in order to communicate with the existing computer codes and to obtain the numerical results more efficiently. A number of SMP programs have been developed and are listed in this report. Finally, the potential applications and restrictions of SMP to engineering problems are also discussed herein.

ACKNOWLEDGEMENT

During the preparation of this report, the second author, Dr. Gautam Dasgupta, was partially supported by the Alexander von Humboldt Foundation, Bonn, West Germany.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

SECTION 1
INTRODUCTION

The complexity of constructing an adequate analysis in the computations of engineering mechanics can be typified by the length of algebraic formulations and their tedious manipulations. Hand calculations most often become prohibitively time consuming. On the other hand, statement oriented line by line coding can be efficiently carried out (debugged and verified) by symbolic manipulators, even though most of the problems are solved approximately by using various numerical schemes written in higher level digital computer languages such as FORTRAN and C.

Due to the difficulties mentioned above, the need for computer algebra cannot be overemphasized. The development of computer algebra in recent years has drawn attention in the field of engineering mechanics.

There are a number of currently available computer algebra programs. The most notable ones (besides SMP which is illustrated in this report) are MACSYMA, REDUCE, and SCRATCHPAD. The selection of a particular software depends on the available operating system. (UNIX is gradually becoming the standard operating system, from personal computers to supercomputers, and these programs are available in the UNIX environment.) In this report, SMP was selected over the others to demonstrate the ease of formulation in computational mechanics. The program consists of a series of functions (with definitions consistent with the generalized definition of projections and filters). The computer library is quite rich with mathematical functions. The most remarkable features, which are particularly attractive for computational mechanics problems, are the ease in which mathematical properties can be attributed to variables; simplified features of defining rules of recursive computation; and availability of polynomial manipulations, such as rational approximation and generation of random polynomials, where the variables could be tensors of arbitrary orders.

In this report, applications of a Symbolic Manipulation Program (SMP) are illustrated. Besides saving enormous amounts of manpower, SMP also facilitates conceptual development and theory enhancement.

The examples and SMP programs in this report primarily emphasize the Finite Element Method. This is the basic purpose of this research work. But as can be seen, some of the SMP programs in Appendix B are written in very general format to permit adoption into solving other engineering problems.

The solvability of a given problem using SMP is also investigated in this report, i.e., how to determine if a given problem can be solved, or how required formulas can be obtained in closed forms, or how to obtain the simplest forms of the results. The interactions between SMP and

other higher level numerical evaluation source codes (FORTRAN, C) are also studied and have been used in some of the examples.

Significant applications of computer algebra in engineering mechanics have been reported by Griffiths and Morrison [1], and Rand [2].

SECTION 2
BRIEF REVIEW OF SMP

SMP (Symbolic Manipulation Program) [4,5,6] is a very powerful computer algebra and computer calculus software package. It was developed by Inference Co., California. SMP can be used either interactively or run by a batch job. It has great depth in the most commonly used areas of mathematics and has a library with a variety of mathematical functions. It has been applied to the engineering field in order to minimize the painful hand formulations of complicated algebraic derivations and to reduce the chance of human errors.

Unlike other computer algebra programs, SMP has the most important feature - *control.* The entire operation can be controlled by overriding the defaults, and by defining fundamental mathematical concepts, such as communicative, associative and distributive rules to different functions. The computer operates according to a user defined operational sequence, which remains unchanged until redefined. Therefore, SMP is easily and well controlled by the user and an entire new mathematical function library can be built without limitations.

Basically, SMP contains four parts:

1. *Syntax* includes numbers, symbols, projections, lists, expressions, patterns, and templates. Each of these can have its own properties defined by the user.

2. *Computational Operations* to control the entire manipulation procedure. The proper use of these operations can save considerable amount of computing time and obtain the simplest output forms without running out of computer memories. They contain four major parts:

   a. Fundamental Operations
   b. Relational and Logical Operations
   c. Structural Operations
   d. Mathematical Operations

3. *Non-computational Operations* consist of input/output (I/O) operations. They include I/O, file I/O, graphic, and textual editing operations.

4. *Interface Operations* allow SMP a desirable capability to generate other high level language codes (FORTRAN and C) such that the communications between the computer algebraic output and existing engineering design/analysis programs can be easily established.

In engineering mechanics, computational operations are of great importance. Due to the complexity of engineering problems, limited capacity of computer memories, and unaffordable consumption of CPU time, many engineering problems remain unsolved. For those problems in which solutions exist, the choice of appropriate operational sequence is of utmost importance. As will be demonstrated later in this report, the selected operational sequence dictates the computational efficiency in terms of CPU time and storage.

SECTION 3
SMP IN ENGINEERING MECHANICS

The Finite Element Method (FEM) [7,8] and Matrix Structure Analysis (MSA) [9,10] are the two most commonly used numerical schemes in solving engineering mechanics problems. In general, they both have the common form as:

$$[K]\{u\} = \{F\} \tag{3-1}$$

where:

[K] = Stiffness matrix
{F} = Applied force
{u} = Response (displacement in most cases).

This generic expression holds not only for statics but also for dynamic problems. In dynamic analysis [11], solution to the following problem is sought:

$$[M]\left\{\frac{d^2u}{dt^2}\right\} + [C]\left\{\frac{du}{dt}\right\} + [K]\{u\} = \{F\} \tag{3-2}$$

where:

[K] = Stiffness matrix
[M] = Mass matrix
[C] = Damping matrix

It can be reformed by using different time integration schemes from numerical analysis algorithms as

$$[K^*]\{u\} = \{F^*\} \tag{3-3}$$

where:

$[K^*]$ = Combination of [M], [C], and [K], the effective stiffness matrix
$\{F^*\}$ = Combination of [M], [C], and {F}, the effective forcing function
{u} = Calculated response at certain time t.

For simplicity, this report only concentrates on static problems.

In finite element analysis, stiffness matrices are obtained from shape functions whose formulations depend on the type of elements used as well as the dimensions of the discretized medium. By using SMP computer algebra, the entire procedure is defined exactly as in the theoretical derivations.

The stiffness matrix [K] is given, in general, by

$$[K] = \int_v [B]^T [D] [B] \, dv \tag{3-4}$$

where [D] is the material property matrix. The matrix [B], the strain-displacement relationship, is derived from differentiations of the shape functions. For example, in a 2-D elasticity problem, the [B] matrix is

$$[B] = \begin{pmatrix} \frac{\partial \Phi_i}{\partial x} & 0 & \frac{\partial \Phi_j}{\partial x} & 0 & \frac{\partial \Phi_k}{\partial x} & 0 \\ 0 & \frac{\partial \Phi_i}{\partial y} & 0 & \frac{\partial \Phi_j}{\partial y} & 0 & \frac{\partial \Phi_k}{\partial y} \\ \frac{\partial \Phi_i}{\partial y} & \frac{\partial \Phi_i}{\partial x} & \frac{\partial \Phi_j}{\partial y} & \frac{\partial \Phi_j}{\partial x} & \frac{\partial \Phi_k}{\partial y} & \frac{\partial \Phi_k}{\partial x} \end{pmatrix} \tag{3-5}$$

where Φ_is' are the shape functions (interpolation functions) determined by the element type (kinematically linear/nonlinear). For a simplex triangular element, with vertices (x_α, y_α), $\alpha = i, j, k$:

$$\begin{Bmatrix} \Phi_i \\ \Phi_j \\ \Phi_k \end{Bmatrix} = [A] \begin{Bmatrix} 1 \\ x \\ y \end{Bmatrix} \tag{3-6}$$

where [A] is the 3 x 3 coefficient matrix and can be calculated as

$$[A] = \begin{pmatrix} 1 & 1 & 1 \\ x_i & x_j & x_k \\ y_i & y_j & y_k \end{pmatrix}^{-1} \tag{3-7}$$

The constitutive matrix [D] has the following form

$$[D] = \begin{pmatrix} D_{11} & D_{12} & 0 \\ D_{21} & D_{22} & 0 \\ 0 & 0 & D_{33} \end{pmatrix} \qquad (3\text{-}8)$$

where $D_{ij} = D_{ji}$ is determined for plane strain or plane stress problems.

After the determinations of matrices [B] and [D], the stiffness matrix [K] can be formulated by integrating Eq. (3-4) either numerically or symbolically by using SMP as shown later.

Symbolically, Eq. (3-3) or Eq. (3-1) can be written as:

u : solvei[kstar] (3-9)

where:

solvei = User-defined implicit or explicit scheme
kstar [K,M,C,F] = User-defined "function" to combine K,M,C and F according to the user-defined scheme.

To implement any higher order integration scheme, one simply substitutes an appropriate kstar function.

The above procedure can be formulated easily by computer algebra using SMP. One starts from Eq. (3-8), then goes to Eq. (3-7),..., up to Eq. (3-4) to obtain the stiffness matrix. Example 1 illustrates these calculations (see Appendix A). It can be seen in this example that SMP handles the given 2-D simplex element quite nicely. Now a complex element is examined, the 1-D problems are considered first. The shape functions can be defined as

$$\Phi_i(x) = \sum_{j=0}^{n-1} a_j^i \, X_j \quad i = 1,2,...,n \qquad (3\text{-}10)$$

where:

n = Number of nodes on the elements
X_i = Value of nodal coordinate

In general, the given problem is transformed into a natural (or normal) coordinate system to simplify the formulation of the stiffness matrices. After the coordinate transformation, the shape functions have the form

$$\Phi_i(\xi) = \frac{(\xi - \xi_1)(\xi - \xi_2) \ldots (\xi - \xi_{i-1})(\xi - \xi_{i+1}) \ldots (\xi - \xi_n)}{(\xi_i - \xi_1)(\xi_i - \xi_2) \ldots (\xi_i - \xi_{i-1})(\xi_i - \xi_{i+1}) \ldots (\xi_i - \xi_n)} \tag{3-11}$$

$$x = \sum_{i=1}^{n} X_i \Phi_i(\xi) \quad \text{and} \quad u = \sum_{i=1}^{n} U_i \Phi_i(\xi) \tag{3-12}$$

Then stiffness matrix [K] will be formed as

$$[K] = \int_{-1}^{1} [B(x(\xi))]^T [D] [B(x(\xi))] \left| \det[J(\xi)] \right| d\xi \tag{3-13}$$

where $J(\xi)$ is the Jacobian of the transformation and is given as

$$[J(\xi)] = \frac{dx}{d\xi} = \sum_{i=1}^{n} X_i \frac{d\Phi_i}{d\xi} \tag{3-14}$$

Therefore the term $\frac{d\Phi_i}{dx}$ in matrix [B] will be calculated by using the chain rule as:

$$\frac{d\Phi_i}{d\xi} = \frac{d\Phi_i}{dx} \frac{dx}{d\xi} \tag{3-15}$$

then substituting Eq. (3-14) into Eq. (3-15), the final form is

$$\frac{d\Phi_i}{dx} = [J(\xi)]^{-1} \frac{d\Phi_i}{d\xi} \tag{3-16}$$

and $\Phi_i(\xi)$ is defined by Eq. (3-11). By using Eq. (3-16), the matrix [B] can be evaluated as

$$[B] = [B(\xi)]J[(\xi)]^{-1} \tag{3-17}$$

Then rewrite Eq. (3-13) as follows

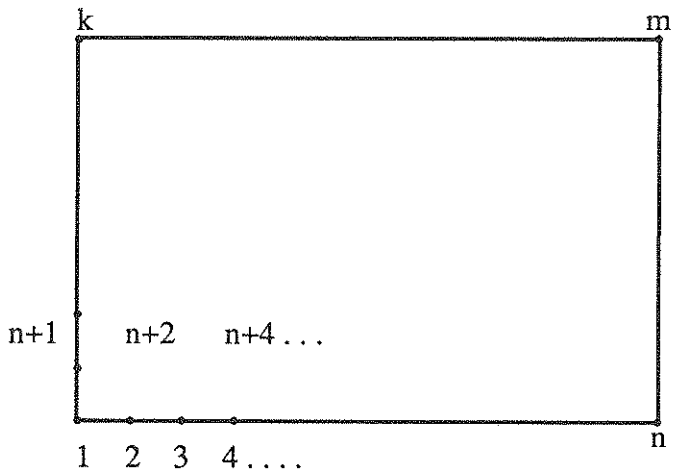
$$[K] = \int_{-1}^{1} [B(\xi)]^T [D] [B(\xi)] \left| \det [J(\xi)] \right| d\xi \tag{3-18}$$

All the aforementioned steps are automated within SMP.

As an example for a 4-Node beam element, SMP is used to obtain the results as shown in Example 2, (see Appendix A).

From Eq. (3-18) above, it is noticed that the variable ξ appears in the denominator in the integrand. This makes the symbolic integration very difficult. In most cases, it is impossible to obtain a closed form solution for the desired stiffness matrix [K]. Therefore, numerical integration algorithms should be used here. This is discussed later in the report.

Now a 2-D rectangular element is considered. Here only Lagrangian elements [7,8] are used in order to utilize the previously discussed shape functions and formulas. The figure below shows a 2-D element with n nodes along x and y coordinates, respectively.



2-D Rectangular Element

Two sets of functions are assumed

$$f_i(\xi) = \sum_{j=0}^{n-1} a_i^j \, \xi^j \tag{3-19}$$

$$g_i(\eta) = \sum_{j=0}^{n-1} b_i^j \, \eta^j \tag{3-20}$$

The shape functions on the given 2-D rectangular element can then be obtained as

$$[\Phi(\xi,\eta)] = \left\{ f_1(\xi), f_2(\xi), \ldots f_n(\xi) \right\}^T \{ g_1(\eta), g_2(\eta), \ldots, g_n(\eta) \} \tag{3-21}$$

or in matrix form as

$$\begin{pmatrix} \Phi_1 & \Phi_{n+1} & \dots \Phi_k \\ \Phi_2 & \Phi_{n+2} & \dots \Phi_{k+1} \\ . & & . \\ . & & . \\ . & & . \\ \Phi_{n-1} & \dots & \dots \Phi_{m-1} \\ \Phi_n & \dots & \dots \Phi_m \end{pmatrix} = \begin{Bmatrix} f_1 \\ f_2 \\ . \\ . \\ . \\ f_{n-1} \\ f_n \end{Bmatrix} \begin{Bmatrix} g_1 \\ g_2 \\ . \\ . \\ . \\ g_{n-1} \\ g_n \end{Bmatrix}^T \tag{3-22}$$

where $k = n(n - 1) + 1$, $m = n^2$, n is the total number of nodes along the edge of the rectangular element in x or y direction. This formulation can easily be accomplished by SMP. Once the shape functions are determined by Eq. (3-22), the Jacobian of the transformation can be evaluated without difficulty by the following procedure:

1. From Eq. (3-12)

$$x = \{X_1, X_2, \dots, X_n\} \begin{Bmatrix} \Phi_1 \\ \Phi_2 \\ . \\ . \\ . \\ \Phi_{n-1} \\ \Phi_n \end{Bmatrix} \tag{3-23}$$

$y(\xi,\eta)$ has the same form as above.

2. Form the partial derivatives

$$\frac{\partial x}{\partial \xi} = \{X_1, X_2, \ldots, X_n\} \begin{Bmatrix} \frac{\partial \Phi_1}{\partial \xi} \\ \frac{\partial \Phi_2}{\partial \xi} \\ . \\ . \\ . \\ \frac{\partial \Phi_{n-1}}{\partial \xi} \\ \frac{\partial \Phi_n}{\partial \xi} \end{Bmatrix} \tag{3-24}$$

similarly, other derivatives can be evaluated.

3. Finally, the Jacobian matrix can be determined as

$$[J] = \frac{\partial x}{\partial \xi}\frac{\partial y}{\partial \eta} - \frac{\partial x}{\partial \eta}\frac{\partial y}{\partial \xi} \tag{3-25}$$

Example 3 (see Appendix A) shows the entire procedure and its calculated results. In those SMP formulations, it can be seen that certain mathematical properties were defined to ease the calculations and simplify the output results.

From SMP Examples 1 through 3, the complexity of the formulations of the stiffness matrix has been demonstrated. A closed form expression of the required stiffness matrix is impossible to obtain due to the presence of the Jacobian expression in the integrand which makes the integration very complicated. Therefore, the stiffness matrix is obtained numerically [12].

Another simple example shown in Example 4 (see Appendix A) demonstrates the power and usefulness of SMP in solving problems in the field of engineering mechanics. Stochastic Finite Element Analysis (SFEA), by means of perturbation method [13] due to system stochasticity, seeks the expansion around the mean X^0 of the stochastic variable x. The variable x can be single or an array of random parameter(s) as shown in the equation below

$$\{\tilde{u}\} = \{u^0\} + \varepsilon\left\{\frac{d\tilde{u}}{dX}\right\} E\{X\} + \frac{\varepsilon^2}{2!}\left\{\frac{d^2\tilde{u}}{dX^2}\right\} E\{X^2\} + \ldots \tag{3-26}$$

where:

$$\{u^0\} = [K^0]^{-1}\{F\} \tag{3-27}$$

$$\left\{\frac{d\tilde{u}}{dX}\right\} = -[K^0]^{-1}\left[\frac{d\tilde{K}}{dx}\right]\{u^0\} \tag{3-28}$$

$$\left\{\frac{d^2\tilde{u}}{dX^2}\right\} = -[K^0]^{-1}\left(\left[\frac{d^2\tilde{K}}{dX^2}\right]\{u^0\} + 2\left[\frac{d\tilde{K}}{dX}\right]\left\{\frac{d\tilde{u}}{dX}\right\}\right) \tag{3-29}$$

In order to obtain terms involving derivatives, $\frac{d^n\tilde{K}}{dx^n}$ must be evaluated. Especially, for the higher order perturbation method, the evaluations of higher order derivatives of [K] are necessary. This can be extremely lengthy and is almost impossible to perform by hand even if the simplex element is used, as demonstrated in the example below.

A simple truss element is chosen as an example in which the value of the nodal coordinate x at node 1 is stochastic as shown in Figure A-4.

The stiffness matrix for u_1 and u_2 degrees of freedom is

$$[K] = \frac{AE}{L}\begin{pmatrix} \cos^2\Theta & \sin\Theta\cos\Theta \\ \sin\Theta\cos\Theta & \sin^2\Theta \end{pmatrix} \tag{3-30}$$

where:

$$L = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{3-31}$$

$$\Theta = \tan^{-1}\frac{y_2 - y_1}{x_2 - x_1} \tag{3-32}$$

A and E represent the area of the element cross section and the material Young's modulus.

From Eq. (3-31) and (3-32), it is clear that [K] is the function of nodal coordinates. As it is assumed, x_1 is the random variable so the derivatives of the truss element stiffness matrix [K] in Eq. (3-30) with respect to x_1 should be evaluated. The SMP evaluation procedure in Example 4 (see Appendix A) calculates the required derivatives.

It can be seen clearly that the derivative calculations of the stiffness matrix of a truss element require repeated simplification by introducing the trigonometrical functions at each calculation step, so that the resulting formula has the simplest form. This is very important. Since the capacity of the computer memory is limited, combined with the fact that the SMP program does not perform any simplifications unless the user specifies it to do so, one can easily run out of

computer memory during the symbolic manipulations if simplifications are not done. On the other hand, the output of the results can be very complicated and unreadable for a very simple problem even though the results can be obtained. This is shown in Example 5 (see Appendix A). Hence, the simplification at each calculation step plays a very important role. But as Example 6 (see Appendix A) demonstrates, the simplification procedure should be kept in a certain sequence to avoid over using the computer memory and obtain the corresponding results. That is, after the previous derivative is obtained and before evaluating the next derivative, the simplification procedure is disabled so that the next calculation will not run into an infinite manipulating loop.

This calculating sequence can be controlled by writing an SMP procedure which contains the proper steps and correct sequence as mentioned before (see SMP Program Listing). The example of the stochastic finite element analysis of a two member truss indicates that the control of the SMP calculation procedure is essential to implement the required solution, and that any improperly defined calculating procedure can make the symbolic manipulation impossible. This proves again that the computer algebra programs are tools for symbolic manipulations and only reduce the unnecessary manpower spent on lengthy algebra. However, it cannot take over the entire calculation without the user control. Users should know how to solve the problems manually before trying to solve them with SMP.

SECTION 4
TRANSLATING SMP OUTPUTS INTO OTHER LANGUAGES

SMP has been shown to have great usage potential and capacity in solving engineering problems. However, it demands more CPU time while performing numerical evaluations. Sometimes one needs to communicate SMP outputs with other existing programs which are written in procedure languages (FORTRAN, C, etc). Therefore, it is necessary to study the capabilities of translating the computer algebraic equations from the SMP output forms into procedure languages in order to execute them numerically. SMP has the capacity to translate its output into FORTRAN or C language upon user specification. Example 7 (see Appendix A) shows how to use the SMP "Prog" function to do this. The only drawback is that the FORTRAN and C output from "Prog" may not be in the format which the user wants it to be in, since it is written in a very lengthy form. Even though SMP can perform these translations, the authors have studied other ways (long hand operations) of translating those formulas into the FORTRAN language. It may not be a simple way, but SMP can certainly do such operations automatically most of the time. The following procedure could be implemented.

1. Use the SMP "Put" function to save the SMP - calculated formulas onto the hard disk in SMP formatted codes under user specified file names.

2. Exit from SMP and return to the normal operating mode.

3. Use the EDIT mode (such as EDIT, EMACS, or vi in the UNIX operating system) to do global substitutions to change those SMP symbols into FORTRAN symbols. For example, x^n in SMP is written as x^n. Then, in the global substitutions, one can use the SUBSTITUTE editing command to change all of the ^ symbols to ** for FORTRAN program usage. Another example is that the brackets, [and], and the braces, { and }, are not permitted in FORTRAN programs; but they can be easily changed into (and) by using the same substituting procedure. Similarly, other non-FORTRAN mathematic symbols can be changed in the same way.

However, there are still some unavoidable disadvantages during the manual operation in EDIT, such as:

1. The first character at each statement may start before the seventh column, which is prohibited in FORTRAN.

2. SMP output does not provide the continuation character on the sixth column for multiple line functions.

3. After the global substitution, there may be more than 72 characters in one line, contrary to what is permitted in FORTRAN.

These problems must be corrected line by line. This can be very inconvenient, especially for multiple line equations. On the other hand, it is almost impossible to check if all of the substitutions have been made properly as specified, even though there are no syntax errors during program compilation. The only way to check them is to evaluate the numerical results by both SMP and its corresponding translated FORTRAN programs and then to compare them to see if they have the same values. This checking method may not even be adequate for all cases.

In the SMP program listing, there is a program called "Datconv." It converts the numerical data from SMP format into a format which can be read directly by FORTRAN or C. Once the numerical data is obtained by SMP, it can be immediately translated into FORTRAN formatted forms for further analysis by using other procedure languages such as FORTRAN/C programs.

SECTION 5
CONCLUSION

As it has been illustrated, computer algebra and computer calculus have the wide capabilities of solving engineering problems and saving man hours on time consuming, lengthy symbolic manipulations. SMP can be practically used in different mathematical evaluations. The algebraic output can be translated into other procedural computer languages and combined to solve more complicated problems. It becomes essential in some theoretical mathematical analysis, especially when closed form calculations are required. The numerical accuracy can be easily controlled by employing SMP or other computer algebra programs since closed form evaluations can yield infinite (arbitrary) precision.

This report is a basic study of the use of SMP in engineering mechanics problems dealing with the finite element formulation.

SECTION 6
REFERENCES

1. Griffiths, S.K. and Morrison, F.A. "Low Peclet Number Heat and Mass Transfer From a Drop in an Electric Field," Journal of Heat Transfer, ASME, Vol. 101, August 1979, pp. 484-488.

2. Rand, H., "Computer Algebra in Applied Mathematics," Pitman, 1984.

3. Panayotidi, T. and DiMaggio, F. "Response of a Pendulum Whose Support is in Steady State Circular Motion," Presented at the ASCE EMD Specialty Conference, SUNY at Buffalo, New York, May, 1987

4. Wolfram, S., "SMP Reference Manual, Version One," Inference Corporation, 1983.

5. Wolfram, S., Greif, J.M., and Kong, M., "SMP Summary, Version One," Inference Corporation, 1983.

6. "SMP Library," Inference Corporation, 1984.

7. Zienkiewicz, O.C., "The Finite Element Method, Third Edition," McGraw-Hill (UK), London, England, 1977.

8. Segerlind, L.J., "Applied Finite Element Analysis," John Wiley & Sons, New York, 1976.

9. McGuire, W. and Gallagher, R.H., "Matrix Structure Analysis," John Wiley & Sons, New York, 1979.

10. Weaver, W., Jr. and Johnston, P.R., "Finite Elements For Structural Analysis," Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1984.

11. Bathe, K.-J., "Finite Element Procedures in Engineering Analysis," Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1982.

12. Reddy, J.N., "An Introduction to The Finite Element Method," McGraw-Hill, New York, 1984.

13. Nakagiri, S. and Hisada, T., "A Note on Stochastic Finite Element Method (Part I, Variation of Stress and Strain Caused by Shape Fluctuation)," SEISAN-SKENKYU, Vol. 32, No. 2, 1980, pp 39.

APPENDIX A
EXAMPLES

Example 1

A given 2-D triangular element is shown below with arbitrary coordinate values on each nodal point, and the material constant matrix [D] is given as

$$[D] = \begin{pmatrix} d_{11} & d_{12} & 0 \\ d_{21} & d_{22} & 0 \\ 0 & 0 & d_{33} \end{pmatrix}$$

where d_{ij} can be determined by the type of problem. For plane stress problem, [D] is calculated as

$$[D] = \frac{E}{1 - \mu^2} \begin{pmatrix} 1 & \mu & 0 \\ \mu & 1 & 0 \\ 0 & 0 & \frac{1 - \mu}{2} \end{pmatrix}$$

or for plane strain problem,

$$[D] = \frac{E(1 - \mu)}{(1 + \mu)(1 - 2\mu)} \begin{pmatrix} 1 & \frac{\mu}{(1 - \mu)} & 0 \\ \frac{\mu}{(1 - \mu)} & 1 & 0 \\ 0 & 0 & \frac{1-2\mu}{2(1 - \mu)} \end{pmatrix}$$

where E is the Young's modulus and μ the Poisson's ratio of the given material.

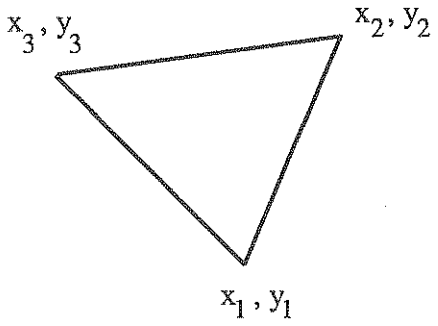


FIGURE A-1 2-D Triangular Element

The SMP output of this example is shown in SMP OUTPUT 1, Appendix B. Notice that the entire output is in the symbolic form and saved in the file named as "TRI K.DAT." It can be reloaded into the SMP program and evaluated at any specified values of (x_i,y_i) and/or d_{ij}. It can also be used for further symbolic manipulations.

Example 2

A 4-Node one dimensional element is shown in Figure A-2.

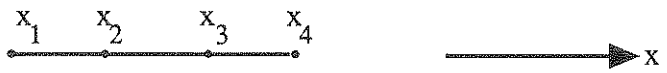


FIGURE A-2 One Dimensional 4-Node Element

From the interpolation functions at each node, calculate the Jacobian of the coordinate transformation and the derivatives of the interpolation functions with respect to x.

The SMP results are represented in SMP OUTPUT 2, Appendix B. In this output, w denotes the variable ξ in the natural coordinate system. From those results, the matrix $[B(\xi)]^T$ can be constructed without difficulties since the derivatives $\frac{d\Phi_i}{d\xi}$ and $J(\xi)^{-1}$ are all evaluated. Eq. (3-15) can be obtained and finally [K] will be determined numerically.

Example 3

Given a 4-node rectangular element as shown in Figure A-3, the interpolation functions at each nodal point are calculated by calling SMP program "ShpFnc."

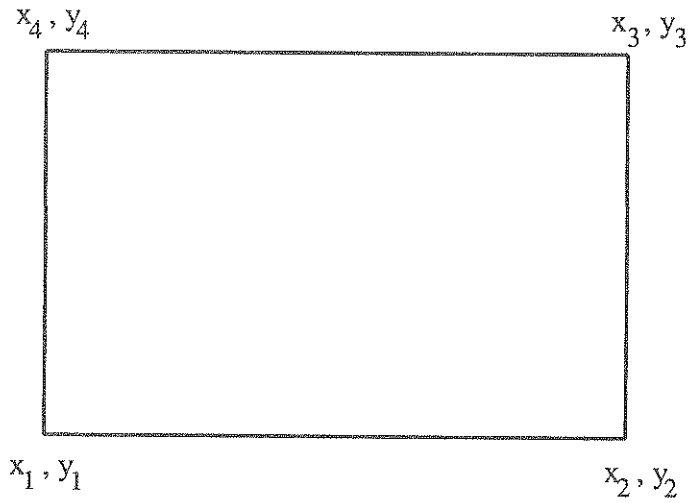


FIGURE A-3 4-Node Rectangular Unit

By following the sequence as described from Eq. (3-19) through Eq. (3-25), the resulting shape functions Φ_i, their derivatives $\frac{d\Phi_i}{d\xi}$ and $\frac{d\Phi_i}{d\eta}$, and the Jacobian $J(\xi,\eta)$ are represented in SMP OUTPUT 3 (see Appendix B).

The stiffness matrix of the rectangular element can be easily obtained by substituting the above results into Eq. (3-18). The final evaluation of the stiffness matrix has to be done numerically, either by SMP or FORTRAN, due to the complexity of the integrand.

Example 4

A 2-D truss element is shown in Figure A-4 with A and E denoting the area of the element cross section and Young's modulus, respectively. A and E are constants through the element.

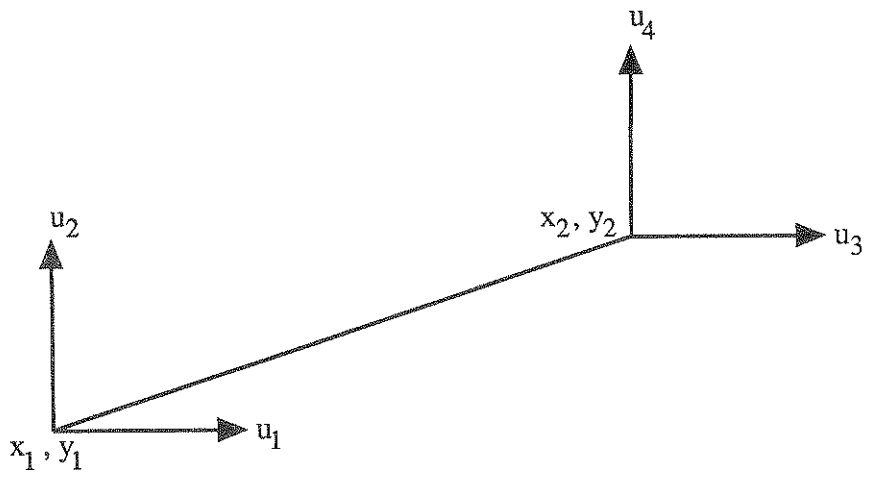


FIGURE A-4 2-D Truss Element

calculate the stiffness matrix and its derivatives up to the 4th order.

The results are shown in SMP OUTPUT 4 (see Appendix A). Notice that the results only show the stiffness matrix about u_1 and u_2 degrees of freedom. The entire stiffness matrix of the element can then be obtained by using these results.

Let $[K_u]$ be the stiffness matrix for u_1 and u_2 degrees of freedom. The stiffness matrix for the entire element $\{u_1,u_2,u_3,u_4\}^T$ is

$$[K] = \begin{pmatrix} [K_u] & -[K_u] \\ -[K_u] & [K_u] \end{pmatrix}$$

In the SMP output, %L and %angle are the length L and the angle Θ of the elements, respectively. Also, it can be seen from this output that the arbitrary order of derivatives for this problem can be easily evaluated by using SMP program "TrusDv." However, this program includes some complicated evaluating procedures to produce the simplest output form. These will be demonstrated by Example 5 and Example 6.

Example 5

In this example, the stiffness matrix and its first and second derivatives are calculated by straight substitutions and evaluations without any simplification. It can be seen from SMP OUTPUT 5

(see Appendix B) that the output is very complicated and lengthy. The second derivative cannot even be recognized. If one continues these calculations, the dynamic memories will run out very soon and the results will not be translated into other higher level languages.

In the next example, the detailed steps to obtain the simplest output possible are demonstrated.

Example 6

This example shows how to simplify an SMP output to obtain the simplest form possible. One should notice that the simplifications occur at every calculation step. Certain manipulation procedures are invoked and reset repeatedly so that the simplifications can be done properly.

From this example, the power of SMP has been illustrated. One can easily control the entire manipulation procedure and let SMP do the calculations the way one has defined. SMP does not perform any "extra" calculations without the user's specifications.

Example 7

In this example, the SMP function "Prog" is used to translate the SMP output from Example 4 into C and FORTRAN languages. The results are shown in SMP OUTPUT 7. SMP performs this translation procedure quite easily. But one should notice that the output form is very lengthy and has too many unnecessary zeros. Despite this "disadvantage," SMP handles the translation very well even for those intrinsic functions, like sin, cos, arctan,..., etc.

APPENDIX B
SMP OUTPUT OF THE EXAMPLES

OUTPUT PAGE

******** S M P O U T P U T 1 ********

```
SMP 1.5.0
18-OCT-1986 14:06:36.05

#I[1]::  <"[-.smp_prog]mkarray";/*SMP program to initialize matrices*/

#I[2]::  <"[-.smp_prog]smplxk";/*SMP program to form stiffness matrix*/

#I[3]::  x:{x1,x2,x3};          /*Nodal coordinates in x-dir.*/

#I[4]::  y:{y1,y2,y3};          /*Nodal coordinates in y-dir.*/

#I[5]::  d:{{d11,d12,0},{d12,d22,0},{0,0,d33}};/* [D] matrix */

#I[6]::  SmplxK[{x,y},d]        /*Form the stiffness matrix of the \
                                  2-D triangular simplex element*/

Dimension of the Elasticity Problem     =       2
Number of Nodes on the Simplex Element  =       3
```

$$\text{Area of the element} = \frac{y1\,(-x2 + x3) - y2\,(-x1 + x3) + y3\,(-x1 + x2)}{2}$$

The shape functions are :

$$\text{Shape Func 1} \qquad \frac{x2\,y3 - x3\,y2 + \%x\,(y2 - y3) + \%y\,(-x2 + x3)}{2\text{xArea}}$$

$$\text{Shape Func 2} \qquad \frac{-x1\,y3 + x3\,y1 - \%x\,(y1 - y3) - \%y\,(-x1 + x3)}{2\text{xArea}}$$

$$\text{Shape Func 3} \qquad \frac{x1\,y2 - x2\,y1 + \%x\,(y1 - y2) + \%y\,(-x1 + x2)}{2\text{xArea}}$$

$$\#O[6]: \quad \left\{\left\{\frac{d11\,(y2 - y3)^2 + d33\,(-x2 + x3)^2}{2(y1\,(-x2 + x3) - y2\,(-x1 + x3) + y3\,(-x1 + x2))},\right.\right.$$

$$\frac{d12\,(-x2 + x3)\,(y2 - y3) + d33\,(-x2 + x3)\,(y2 - y3)}{2(y1\,(-x2 + x3) - y2\,(-x1 + x3) + y3\,(-x1 + x2))},$$

$$\frac{-d11\,(y1 - y3)\,(y2 - y3) - d33\,(-x1 + x3)\,(-x2 + x3)}{2(y1\,(-x2 + x3) - y2\,(-x1 + x3) + y3\,(-x1 + x2))},$$

$$\frac{-d12\,(-x1 + x3)\,(y2 - y3) - d33\,(-x2 + x3)\,(y1 - y3)}{2(y1\,(-x2 + x3) - y2\,(-x1 + x3) + y3\,(-x1 + x2))},$$

$$\frac{d11\,(y1 - y2)\,(y2 - y3) + d33\,(-x1 + x2)\,(-x2 + x3)}{},$$

$$\frac{2(y1\ (-x2 + x3) - y2\ (-x1 + x3) + y3\ (-x1 + x2))}{2(y1\ (-x2 + x3) - y2\ (-x1 + x3) + y3\ (-x1 + x2))},$$

$$\frac{d12\ (-x1 + x2)\ (y2 - y3) + d33\ (-x2 + x3)\ (y1 - y2)}{2(y1\ (-x2 + x3) - y2\ (-x1 + x3) + y3\ (-x1 + x2))}\},$$

$$\{\frac{d12\ (-x2 + x3)\ (y2 - y3) + d33\ (-x2 + x3)\ (y2 - y3)}{2(y1\ (-x2 + x3) - y2\ (-x1 + x3) + y3\ (-x1 + x2))},$$

$$\frac{d22\ (-x2 + x3)^2 + d33\ (y2 - y3)^2}{2(y1\ (-x2 + x3) - y2\ (-x1 + x3) + y3\ (-x1 + x2))},$$

$$\frac{-d12\ (-x2 + x3)\ (y1 - y3) - d33\ (-x1 + x3)\ (y2 - y3)}{2(y1\ (-x2 + x3) - y2\ (-x1 + x3) + y3\ (-x1 + x2))},$$

$$\frac{-d22\ (-x1 + x3)\ (-x2 + x3) - d33\ (y1 - y3)\ (y2 - y3)}{2(y1\ (-x2 + x3) - y2\ (-x1 + x3) + y3\ (-x1 + x2))},$$

$$\frac{d12\ (-x2 + x3)\ (y1 - y2) + d33\ (-x1 + x2)\ (y2 - y3)}{2(y1\ (-x2 + x3) - y2\ (-x1 + x3) + y3\ (-x1 + x2))},$$

$$\frac{d22\ (-x1 + x2)\ (-x2 + x3) + d33\ (y1 - y2)\ (y2 - y3)}{2(y1\ (-x2 + x3) - y2\ (-x1 + x3) + y3\ (-x1 + x2))}\},$$

$$\{\frac{-d11\ (y1 - y3)\ (y2 - y3) - d33\ (-x1 + x3)\ (-x2 + x3)}{2(y1\ (-x2 + x3) - y2\ (-x1 + x3) + y3\ (-x1 + x2))},$$

$$\frac{-d12\ (-x2 + x3)\ (y1 - y3) - d33\ (-x1 + x3)\ (y2 - y3)}{2(y1\ (-x2 + x3) - y2\ (-x1 + x3) + y3\ (-x1 + x2))},$$

$$\frac{d11\ (y1 - y3)^2 + d33\ (-x1 + x3)^2}{2(y1\ (-x2 + x3) - y2\ (-x1 + x3) + y3\ (-x1 + x2))},$$

$$\frac{d12\ (-x1 + x3)\ (y1 - y3) + d33\ (-x1 + x3)\ (y1 - y3)}{2(y1\ (-x2 + x3) - y2\ (-x1 + x3) + y3\ (-x1 + x2))},$$

$$\frac{-d11\ (y1 - y2)\ (y1 - y3) - d33\ (-x1 + x2)\ (-x1 + x3)}{2(y1\ (-x2 + x3) - y2\ (-x1 + x3) + y3\ (-x1 + x2))},$$

$$\frac{-d12\ (-x1 + x2)\ (y1 - y3) - d33\ (-x1 + x3)\ (y1 - y2)}{2(y1\ (-x2 + x3) - y2\ (-x1 + x3) + y3\ (-x1 + x2))}\},$$

$$\{\frac{-d12\ (-x1 + x3)\ (y2 - y3) - d33\ (-x2 + x3)\ (y1 - y3)}{},$$

$$
\begin{Bmatrix}
\dfrac{2(y1\,(-x2+x3)-y2\,(-x1+x3)+y3\,(-x1+x2))}{2(y1\,(-x2+x3)-y2\,(-x1+x3)+y3\,(-x1+x2))},\\[2ex]
\dfrac{-d22\,(-x1+x3)\,(-x2+x3)-d33\,(y1-y3)\,(y2-y3)}{2(y1\,(-x2+x3)-y2\,(-x1+x3)+y3\,(-x1+x2))},\\[2ex]
\dfrac{d12\,(-x1+x3)\,(y1-y3)+d33\,(-x1+x3)\,(y1-y3)}{2(y1\,(-x2+x3)-y2\,(-x1+x3)+y3\,(-x1+x2))},\\[2ex]
\dfrac{d22\,(-x1+x3)^2+d33\,(y1-y3)^2}{2(y1\,(-x2+x3)-y2\,(-x1+x3)+y3\,(-x1+x2))},\\[2ex]
\dfrac{-d12\,(-x1+x3)\,(y1-y2)-d33\,(-x1+x2)\,(y1-y3)}{2(y1\,(-x2+x3)-y2\,(-x1+x3)+y3\,(-x1+x2))},\\[2ex]
\dfrac{-d22\,(-x1+x2)\,(-x1+x3)-d33\,(y1-y2)\,(y1-y3)}{2(y1\,(-x2+x3)-y2\,(-x1+x3)+y3\,(-x1+x2))}\Big\},\\[2ex]
\Big\{\dfrac{d11\,(y1-y2)\,(y2-y3)+d33\,(-x1+x2)\,(-x2+x3)}{2(y1\,(-x2+x3)-y2\,(-x1+x3)+y3\,(-x1+x2))},\\[2ex]
\dfrac{d12\,(-x2+x3)\,(y1-y2)+d33\,(-x1+x2)\,(y2-y3)}{2(y1\,(-x2+x3)-y2\,(-x1+x3)+y3\,(-x1+x2))},\\[2ex]
\dfrac{-d11\,(y1-y2)\,(y1-y3)-d33\,(-x1+x2)\,(-x1+x3)}{2(y1\,(-x2+x3)-y2\,(-x1+x3)+y3\,(-x1+x2))},\\[2ex]
\dfrac{-d12\,(-x1+x3)\,(y1-y2)-d33\,(-x1+x2)\,(y1-y3)}{2(y1\,(-x2+x3)-y2\,(-x1+x3)+y3\,(-x1+x2))},\\[2ex]
\dfrac{d11\,(y1-y2)^2+d33\,(-x1+x2)^2}{2(y1\,(-x2+x3)-y2\,(-x1+x3)+y3\,(-x1+x2))},\\[2ex]
\dfrac{d12\,(-x1+x2)\,(y1-y2)+d33\,(-x1+x2)\,(y1-y2)}{2(y1\,(-x2+x3)-y2\,(-x1+x3)+y3\,(-x1+x2))}\Big\},\\[2ex]
\Big\{\dfrac{d12\,(-x1+x2)\,(y2-y3)+d33\,(-x2+x3)\,(y1-y2)}{2(y1\,(-x2+x3)-y2\,(-x1+x3)+y3\,(-x1+x2))},\\[2ex]
\dfrac{d22\,(-x1+x2)\,(-x2+x3)+d33\,(y1-y2)\,(y2-y3)}{2(y1\,(-x2+x3)-y2\,(-x1+x3)+y3\,(-x1+x2))},\\[2ex]
\dfrac{-d12\,(-x1+x2)\,(y1-y3)-d33\,(-x1+x3)\,(y1-y2)}{}\,,
\end{Bmatrix}
$$

$$\frac{2(y1\ (-x2+x3)-y2\ (-x1+x3)+y3\ (-x1+x2))}{2(y1\ (-x2+x3)-y2\ (-x1+x3)+y3\ (-x1+x2))},$$

$$\frac{-d22\ (-x1+x2)\ (-x1+x3)-d33\ (y1-y2)\ (y1-y3)}{2(y1\ (-x2+x3)-y2\ (-x1+x3)+y3\ (-x1+x2))},$$

$$\frac{d12\ (-x1+x2)\ (y1-y2)+d33\ (-x1+x2)\ (y1-y2)}{2(y1\ (-x2+x3)-y2\ (-x1+x3)+y3\ (-x1+x2))},$$

$$\frac{d22\ (-x1+x2)^2+d33\ (y1-y2)^2}{2(y1\ (-x2+x3)-y2\ (-x1+x3)+y3\ (-x1+x2))}\}\}$$

```
[7]::  Put[%,"tri_k.dat"];  /*Save the calculated results for future uses*/

[8]::  Exit[]
```

```
******** S M P    O U T P U T    2 ********

SMP 1.5.0
18-OCT-1986 19:00:16.05

#I[1]::  <"[-.smp_prog]shpfnc"; /*Interpolation function calculations*/

#I[2]::  x:{x1,x2,x3,x4};/*Define nodal coordinates (4-Node element)*/

#I[3]::  shp:ShpFnc[w,4] /*Shape functions with "w" as variable*/

                 -9(-1 + w) (-1/3 + w) (1/3 + w)   27(-1 + w) (-1/3 + w) (1 + w)
#O[3]:   {---------------------------------------,-----------------------------------,
                              16                                16

                 -27(-1 + w) (1/3 + w) (1 + w)   9(-1/3 + w) (1/3 + w) (1 + w)
                 -----------------------------,-----------------------------}
                              16                                16

#I[4]::  jacob:x.shp;/*Calculate the Jacobian in this 1-D problem*/

#I[5]::  jacob:Ex[jacob];/*Try to simplify the output results*/

#I[6]::  jacob:Cb[jacob,{w,w^2,w^3}];/*Collect terms*/

                 -x1 + 9x2 + 9x3 - x4     w (x1 - 27x2 + 27x3 - x4)
#O[6]:   ---------------------------- + --------------------------
                        16                          16

                   2                                3
                  w  (9x1 - 9x2 - 9x3 + 9x4)       w  (-9x1 + 27x2 - 27x3 + 9x4)
               + --------------------------- + -------------------------------
                              16                             16

#I[7]::  dshp:Ex[D[shp,w]]  /*Derivative of shape functions*/

                                    2                                2                                2
#O[4]:  {1/16 + 9w/8 - 27 w/16,-27/16 - 9w/8 + 81 w/16,27/16 - 9w/8 - 81 w/16,
                                  2
                -1/16 + 9w/8 + 27 w/16}

#I[8]::  invj:1/jacob       /*Inverse of Jacobian*/

                                                                16
#O[8]:  ------------------------------------------------------------------------------
        -x1 + 9x2 + 9x3 - x4 + w (x1 - 27x2 + 27x3 - x4)

                  2                                3
             + w  (9x1 - 9x2 - 9x3 + 9x4) + w  (-9x1 + 27x2 - 27x3 + 9x4

#I[9]::  Exit[]
```

```
                ******** S M P   O U T P U T   3 ********

SMP 1.5.0
19-OCT-1986 10:54:37.05

#I[1]::  <"[-.smp_prog]shpfnc"; /*Shape function calculation program*/

#I[2]::  shpf:ShpFnc[w,2]       /*Shape functions in w-direction*/

#O[2]:   {-(-1 + w)/2, (1 + w)/2}

#I[3]::  shpg:ShpFnc[v,2]       /*Shape functions in v-direction*/

#O[3]:   {-(-1 + v)/2, (1 + v)/2}

#I[4]::  shpm:shpf**shpg;       /*Shape functions as in Eq.(10)*/

#I[5]::  shp:Flat[Trans[shpm]]  /*Shape functions as a vector form*/

#O[5]:   {(-1 + v) (-1 + w)/4, -(-1 + v) (1 + w)/4, -(-1 + w) (1 + v)/4,
           (1 + v) (1 + w)/4}

#I[6]::  x:{x1,x2,x3,x4}.shp   /*x represented by shape functions*/

#O[6]:   x1 (-1 + v) (-1 + w)/4 - x2 (-1 + v) (1 + w)/4 - x3 (-1 + w) (1 + v
           x4 (1 + v) (1 + w)
         + ------------------
                   4

#I[7]::  x:Cb[Ex[x],{v/4,w/4}] /*Reorginazing the output form of x*/

#O[7]:   x1/4 + x2/4 + x3/4 + x4/4

           v (-x1 - x2 + x3 + x4 + w x1 - w x2 - w x3 + w x4)
         + --------------------------------------------------
                                 4

           w (-x1 + x2 - x3 + x4)
         + ----------------------
                   4

#I[8]::  y:{y1,y2,y3,y4}.shp   /*y represedted by shape functions*/

#O[8]:   y1 (-1 + v) (-1 + w)/4 - y2 (-1 + v) (1 + w)/4 - y3 (-1 + w) (1 +
                 y4 (1 + v) (1 + w)
```

```
                    + ------------------
                              4

#I[9]::  xdw:Cb[Ex[D[x,w]],{v/4,w/4}] /*Derivative of x about w*/

                                                 v (x1 - x2 - x3 + x4)
#O[9]:   -x1/4 + x2/4 - x3/4 + x4/4 + ----------------------------
                                                      4

#I[10]::  xdv:Cb[Ex[D[x,v]],{v/4,w/4}] /*Derivative of x about v*/

                                                 w (x1 - x2 - x3 + x4)
#O[10]:  -x1/4 - x2/4 + x3/4 + x4/4 + ----------------------------
                                                      4

#I[11]::  ydw:Cb[Ex[D[y,w]],{v/4,w/4}] /*Derivative of y about w*/

                                                 v (y1 - y2 - y3 + y4)
#O[11]:  -y1/4 + y2/4 - y3/4 + y4/4 + ----------------------------
                                                      4

#I[12]::  ydv:Cb[Ex[D[y,v]],{v/4,w/4}] /*Derivative of y about v*/

                                                 w (y1 - y2 - y3 + y4)
#O[12]:  -y1/4 - y2/4 + y3/4 + y4/4 + ----------------------------
                                                      4

#I[13]::  jacob:Ex[xdw ydv - xdv ydw] /*Evaluate the Jacobian*/

           x1 y2    x1 y3    x2 y1    x2 y4    x3 y1    x3 y4    x4 y2    x4 y3
#O[13]:   -----  -  -----  -  -----  +  -----  +  -----  -  -----  -  -----  +  -----
            8        8        8        8        8        8        8        8

             v x1 y2    v x1 y4    v x2 y1    v x2 y3    v x3 y2    v x3 y4
          -  -------  +  -------  +  -------  -  -------  +  -------  -  -------
               8          8          8          8          8          8

             v x4 y1    v x4 y3    w x1 y3    w x1 y4    w x2 y3    w x2 y4
          -  -------  +  -------  +  -------  -  -------  -  -------  +  -------
               8          8          8          8          8          8

             w x3 y1    w x3 y2    w x4 y1    w x4 y2
          -  -------  +  -------  +  -------  -  -------
               8          8          8          8

#I[14]::  jacob:Cb[jacob,{v/8,w/8}]  /*Represent result in better form*/

                v (-x1 y2 + x1 y4 + x2 y1 - x2 y3 + x3 y2 - x3 y4
                                  - x4 y1 + x4 y3)
#O[14]:   ------------------------------------------------------
                                    8

                  w (x1 y3 - x1 y4 - x2 y3 + x2 y4 - x3 y1 + x3 y2
                                  + x4 y1 - x4 y2)                    x1 y2
               + ------------------------------------------------- + -----
                                    8                                  8

                  x1 y3    x2 y1    x2 y4    x3 y1    x3 y4    x4 y2    x4 y3
```

```
 - ----- - ----- + ----- + ----- - ----- - ----- + -----
     8         8       8       8       8       8       8

#I[15]::  <"[-.smp_prog]jacobin";/*Load Jacobian evaluation program*/

#I[16]::  coord:{{x1,y1},{x2,y2},{x3,y3},{x4,y4}};/*Nodal coordinates*/

#I[17]::  Jacobn[coord,shp,{w,v}] /*Evaluate Jacobian matrix directly*/

            x1 (-1 + v)   x2 (-1 + v)   x3 (1 + v)   x4 (1 + v)
#O[17]:   {{----------- - ----------- - ---------- + ----------,
                 4             4             4            4

                          y1 (-1 + v)   y2 (-1 + v)   y3 (1 + v)
                          ----------- - ----------- - ----------
                               4             4             4

                                    y4 (1 + v)
                                  + ----------},
                                        4

               x1 (-1 + w)   x2 (1 + w)   x3 (-1 + w)   x4 (1 + w)
              {----------- - ---------- - ----------- + ----------,
                    4             4            4             4

                          y1 (-1 + w)   y2 (1 + w)   y3 (-1 + w)
                          ----------- - ---------- - -----------
                               4             4            4

                                    y4 (1 + w)
                                  + ----------}}
                                        4

#I[18]::  jacob1:Cb[Ex[Det[%]],{w/8,v/8}] /*Find Jacobian from above*/

          v (-x1 y2 + x1 y4 + x2 y1 - x2 y3 + x3 y2 - x3 y4

                   - x4 y1 + x4 y3)
#O[18]:   ----------------------------------------------
                          8

            w (x1 y3 - x1 y4 - x2 y3 + x2 y4 - x3 y1 + x3 y2

                   + x4 y1 - x4 y2)                          x1 y2
          + ---------------------------------------------- + -----
                          8                                    8

            x1 y3   x2 y1   x2 y4   x3 y1   x3 y4   x4 y2   x4 y3
          - ----- - ----- + ----- + ----- - ----- - ----- + -----
              8       8       8       8       8       8       8

#I[19]::  jacob-jacob1  /*Compare results of the diff. evaluations*/

#O[19]:   0

#I[20]::  Exit[]        /*Calculations are done. Exit from SMP*/
```

```
                ********  S M P    O U T P U T    4 ********
SMP 1.5.0
19-OCT-1986 23:29:48.05

#I[1]::  <"[-.smp_prog]TrusDv"   /*Load calculation program*/

#I[2]::  coord:{{x1,y1},{x2,y2}};/*Set nodal coordinates*/

#I[3]::  a:TrusDv[coord,x1,4];   /*Evaluate the results*/

 Derivatives with respect to    x1

  Calculating the        1        -th derivative
  Calculating the        2        -th derivative
  Calculating the        3        -th derivative
  Calculating the        4        -th derivative

 TOTAL NUMBER OF DERIVATIVES =  4

#I[4]::  K:a[1]    /*Stiffness matrix*/

#O[4]:   {{1 + Cos[2*angl]   Sin[2*angl]     Sin[2*angl]   1 - Cos[2*angl]
          {{---------------,-----------},{-----------,---------------}}
                  2*L            2*L           2*L            2*L

         /*  DERIVATIVES OF [K] WITH RESPECT TO x1  */

#I[5]::  Kdx1:a[2]  /*First derivative of [K]*/

#O[5]:   {{Cos[*angl]     3Cos[3*angl]   -Sin[*angl]     3Sin[3*angl]
          {{----------  + ------------,-----------  + ------------},
                 2                 2              2              2
            4 *L              4 *L           4 *L           4 *L

             -Sin[*angl]     3Sin[3*angl]   3Cos[*angl]     3Cos[3*angl]
            {-----------  + ------------,-----------  - ------------}}
                   2                 2              2              2
              4 *L              4 *L           4 *L           4 *L

#I[6]::  Kddx1:a[3]  /*Second derivative of [K]*/

#O[6]:   {{1       15Cos[4*angl]   -3Sin[2*angl]     15Sin[4*angl]
          {{-----  + -------------,--------------  + -------------},
                3                3                3              3
           8 *L             8 *L             4 *L           8 *L

            -3Sin[2*angl]     15Sin[4*angl]
           {--------------  + -------------,
                    3                3
               4 *L             8 *L

                                  3       3Cos[2*angl]     15Cos[4*angl]
                                -----  + ------------  - -------------}}
                                   3                3                3
                               8 *L             2 *L             8 *L
```

```
#I[7]::  Kd2x1:a[4]  /*Third derivative of [K]*/

#O[7]:   {{3Cos[%angl]    15Cos[3%angl]    105Cos[5%angl]
          {------------ - -------------- + --------------,
                   4                4                4
              8 %L             16 %L            16 %L

                  -3Sin[%angl]    45Sin[3%angl]    105Sin[5%angl]
                  ------------ - -------------- + --------------},
                             4                4                4
                        8 %L             16 %L            16 %L

           -3Sin[%angl]    45Sin[3%angl]    105Sin[5%angl]
          {------------ - -------------- + --------------,
                   4                4                4
              8 %L             16 %L            16 %L

                   15Cos[%angl]    75Cos[3%angl]    105Cos[5%angl]
                   ------------ + -------------- - --------------}}
                              4                4                4
                         8 %L             16 %L            16 %L

#I[8]::  Kd3x1:a[5]  /*Fourth derivative of [K]*/

#O[8]:   {{9       15Cos[2%angl]    105Cos[4%angl]    945Cos[6%angl]
          {------ + -------------- - -------------- + --------------,
                5                5                5                5
           16 %L            32 %L            16 %L            32 %L

                  -75Sin[2%angl]    105Sin[4%angl]    945Sin[6%angl]
                  -------------- - -------------- + --------------},
                               5                5                5
                          32 %L             8 %L            32 %L

           -75Sin[2%angl]    105Sin[4%angl]    945Sin[6%angl]
          {-------------- - -------------- + --------------,
                       5                5                5
                  32 %L             8 %L            32 %L

                    45      225Cos[2%angl]    315Cos[4%angl]
                  ------ + -------------- + --------------
                       5                5                5
                  16 %L            32 %L            16 %L

                                                    945Cos[6%angl]
                                                  - --------------}}
                                                               5
                                                          32 %L

#I[9]::  Length:a[6]  /*Where %L is the Length of element*/

                      2                2 1/2
#O[9]:   ((-x1 + x2)  + (-y1 + y2) )

#I[10]::  Angl:a[7]   /*and %angl is the Angle of element*/

                 -y1 + y2
#O[10]:   Atan[---------]
                 -x1 + x2
```

```
#I[11]::  Exit[]        /*Entire calculations are finished*/
```

******** S M P   O U T P U T 5 ********

```
SMP 1.5.0
19-OCT-1986 11:43:11.05

#I[1]::  K:AE/L {{Cos[theta]^2,Sin[theta]Cos[theta]},\
                 {Sin[theta]Cos[theta],Sin[theta]^2}}\
                /*Form stiffness matrix of a truss element*/

                          2
           AE Cos[theta]   AE Cos[theta] Sin[theta]
#O[1]:   {{--------------,------------------------},
                L                    L
                                                         2
               AE Cos[theta] Sin[theta] AE Sin[theta]
              {------------------------,--------------}}
                         L                     L

#I[2]::  L:Sqrt[(x2-x1)^2+(y2-y1)^2]   /*Define element length*/

                      2               2 1/2
#O[2]:   ((-x1 + x2)   + (-y1 + y2)  )

#I[3]::  theta:Atan[(y2-y1)/(x2-x1)]    /*Define the angle*/

            -y1 + y2
#O[3]:   Atan[--------]
            -x1 + x2

#I[4]::  K:K        /*Form [K] in long hand form*/

                          -y1 + y2  2
            AE Cos[Atan[--------]]
                          -x1 + x2
#O[4]:   {{------------------------------,
                      2               2 1/2
            ((-x1 + x2)   + (-y1 + y2)  )

                           -y1 + y2              -y1 + y2
              AE Cos[Atan[--------]] Sin[Atan[--------]]
                           -x1 + x2              -x1 + x2
              ------------------------------------------},
                                  2               2 1/2
                      ((-x1 + x2)   + (-y1 + y2)  )

                        -y1 + y2              -y1 + y2
           AE Cos[Atan[--------]] Sin[Atan[--------]]
                        -x1 + x2              -x1 + x2
          {------------------------------------------,
                              2               2 1/2
                  ((-x1 + x2)   + (-y1 + y2)  )

                               -y1 + y2  2
                  AE Sin[Atan[--------]]
                               -x1 + x2
                 ------------------------------}}
```

$$\left((-x1 + x2)^2 + (-y1 + y2)^2\right)^{1/2}$$

#I[5]:: Kdx1:D[K,x1] /*Derivative of [K] with respect to x1*/

$$\frac{-2AE \cos\left[\operatorname{Atan}\left[\frac{-y1 + y2}{-x1 + x2}\right]\right] (-y1 + y2) \left((-x1 + x2)^2 + (-y1 + y2)^2\right)}{\left(1 + \frac{(-y1 + y2)^2}{(-x1 + x2)^2}\right) (-x1 + x2)^2} * \sin\left[\operatorname{Atan}\left[\frac{-y1 + y2}{-x1 + x2}\right]\right]$$

$$- \frac{AE \cos\left[\operatorname{Atan}\left[\frac{-y1 + y2}{-x1 + x2}\right]\right]^2 (2x1 - 2x2)}{2}$$

#O[5]:

$$\left\{\left\{\frac{-2AE \cos\left[\operatorname{Atan}\left[\frac{-y1 + y2}{-x1 + x2}\right]\right] (-y1 + y2) \left((-x1 + x2)^2 + (-y1 + y2)^2\right)}{\left((-x1 + x2)^2 + (-y1 + y2)^2\right)^{3/2}}\right.\right.$$

$$\left(\frac{-AE (-y1 + y2) \sin\left[\operatorname{Atan}\left[\frac{-y1 + y2}{-x1 + x2}\right]\right]^2}{\left(1 + \frac{(-y1 + y2)^2}{(-x1 + x2)^2}\right) (-x1 + x2)^2}\right.$$

$$\left. + \frac{AE \cos\left[\operatorname{Atan}\left[\frac{-y1 + y2}{-x1 + x2}\right]\right]^2 * (-y1 + y2)}{\left(1 + \frac{(-y1 + y2)^2}{(-x1 + x2)^2}\right) (-x1 + x2)^2}\right)$$

$$* \left((-x1 + x2)^2 + (-y1 + y2)^2\right)$$

$$\left. - \frac{AE \cos\left[\operatorname{Atan}\left[\frac{-y1 + y2}{-x1 + x2}\right]\right] (2x1 - 2x2)}{2}\right\},$$

```
                                   -y1 + y2
                        * Sin[Atan[--------]]
                                   -x1 + x2
                   -  -------------------------------------
                                      2
         -----------------------------------------------------},
                             2             2 3/2
                    ((-x1 + x2)  + (-y1 + y2) )

                                 -y1 + y2  2
         -AE (-y1 + y2) Sin[Atan[--------]]
                                 -x1 + x2
       (-------------------------------------
                        2
                 (-y1 + y2)               2
            (1 + ----------) (-x1 + x2)
                         2
                 (-x1 + x2)
                                                      -y1 + y2  2
                                      AE Cos[Atan[--------]]
                                                      -x1 + x2

                                          * (-y1 + y2)
                               + -----------------------------)
                                                    2
                                          (-y1 + y2)              2
                                     (1 + ----------) (-x1 + x2)
                                                    2
                                          (-x1 + x2)

                                    2              2
                        * ((-x1 + x2)  + (-y1 + y2) )

                                    -y1 + y2
                   AE Cos[Atan[--------]] (2x1 - 2x2)
                                    -x1 + x2

                                     -y1 + y2
                     * Sin[Atan[--------]]
                                     -x1 + x2
                 -  ------------------------------------
                                         2
{-------------------------------------------------------------,
                           2              2 3/2
                  ((-x1 + x2)  + (-y1 + y2) )

                          -y1 + y2
        2AE Cos[Atan[--------]] (-y1 + y2)
                          -x1 + x2

                                     2              2
                        * ((-x1 + x2)  + (-y1 + y2) )

                                    -y1 + y2
                        * Sin[Atan[--------]]
                                    -x1 + x2
         -----------------------------------------
                             2
                   (-y1 + y2)             2
```

```
          (1 + ----------) (-x1 + x2)
                        2
               (-x1 + x2)

                                         -y1 + y2  2
                 AE (2x1 - 2x2) Sin[Atan[--------]]
                                         -x1 + x2
              - ------------------------------------
                                 2
   --------------------------------------------------}}
                          2              2 3/2
         ((-x1 + x2)  + (-y1 + y2) )
```

```
#I[6]::  Kddx1:D[Kdx1,x1] /*Second derivative of K with respect to x1*/

#O[6]:   {{(-3/2((2x1 + -2x2)*(-2((AE*Cos[Atan[(-y1 + y2)/(-x1 + x2)]]\
         *(-y1 + y2)*((-x1 + x2)^2 + (-y1 + y2)^2)*Sin[Atan[(-y1 + y2)\
          /(-x1 + x2)]])/((1 + (-y1 + y2)^2/(-x1 + x2)^2)*(-x1 + x2)\
          ^2)) + -1/2(AE*Cos[Atan[(-y1 + y2)/(-x1 + x2)]]^2*(2x1\
           + -2x2)))) + ((-x1 + x2)^2 + (-y1 + y2)^2)*((AE*Cos[Atan[\
          (-y1 + y2)/(-x1 + x2)]]*(2x1 + -2x2)*(-y1 + y2)*Sin[Atan[(\
          -y1 + y2)/(-x1 + x2)]])/((1 + (-y1 + y2)^2/(-x1 + x2)\
          ^2)*(-x1 + x2)^2) + -2(((1 + (-y1 + y2)^2/(-x1 + x2)^2)\
          *(-x1 + x2)^2*((AE*Cos[Atan[(-y1 + y2)/(-x1 + x2)]]^2\
          *(-y1 + y2)^2*((-x1 + x2)^2 + (-y1 + y2)^2))/((1 + (-y1\
           + y2)^2/(-x1 + x2)^2)*(-x1 + x2)^2) + -((AE*(-y1 + y2)\
          ^2*((-x1 + x2)^2 + (-y1 + y2)^2)*Sin[Atan[(-y1 + y2)/(-x1\
           + x2)]]^2)/((1 + (-y1 + y2)^2/(-x1 + x2)^2)*(-x1 + x2)\
          ^2)) + AE*Cos[Atan[(-y1 + y2)/(-x1 + x2)]]*(2x1 + -2x2)\
          *(-y1 + y2)*Sin[Atan[(-y1 + y2)/(-x1 + x2)]]) + -(AE*Cos[\
          Atan[(-y1 + y2)/(-x1 + x2)]]*(-y1 + y2)*(2((-y1 + y2)\
          ^2/(-x1 + x2)) + -2((1 + (-y1 + y2)^2/(-x1 + x2)^2)*(-x1\
           + x2)))*((-x1 + x2)^2 + (-y1 + y2)^2)*Sin[Atan[(-y1 + y2)\
          /(-x1 + x2)]]))/((1 + (-y1 + y2)^2/(-x1 + x2)^2)^2*(-x1\
           + x2)^4)) + -(AE*Cos[Atan[(-y1 + y2)/(-x1 + x2)]]^2)))\
      /((-x1 + x2)^2 + (-y1 + y2)^2)^(5/2),(-3/2((2x1 + -2x2)\
         *((-((AE*(-y1 + y2)*Sin[Atan[(-y1 + y2)/(-x1 + x2)]]^2)\
          /((1 + (-y1 + y2)^2/(-x1 + x2)^2)*(-x1 + x2)^2)) + (AE\
          *Cos[Atan[(-y1 + y2)/(-x1 + x2)]]^2*(-y1 + y2))/((1 + (-y1\
           + y2)^2/(-x1 + x2)^2)*(-x1 + x2)^2))*((-x1 + x2)^2 + (-y1\
           + y2)^2) + -1/2(AE*Cos[Atan[(-y1 + y2)/(-x1 + x2)]]*(2x1\
           + -2x2)*Sin[Atan[(-y1 + y2)/(-x1 + x2)]]))) + ((-x1 + x2)\
          ^2 + (-y1 + y2)^2)*(1/2((AE*(2x1 + -2x2)*(-y1 + y2)*Sin[\
          Atan[(-y1 + y2)/(-x1 + x2)]]^2)/((1 + (-y1 + y2)^2/(-x1\
           + x2)^2)*(-x1 + x2)^2)) + -1/2((AE*Cos[Atan[(-y1 + y2)\
          /(-x1 + x2)]]^2*(2x1 + -2x2)*(-y1 + y2))/((1 + (-y1 + y2)\
          ^2/(-x1 + x2)^2)*(-x1 + x2)^2)) + (2x1 + -2x2)*(-((AE\
          *(-y1 + y2)*Sin[Atan[(-y1 + y2)/(-x1 + x2)]]^2)/((1 + (-y1\
           + y2)^2/(-x1 + x2)^2)*(-x1 + x2)^2)) + (AE*Cos[Atan[(-y1\
           + y2)/(-x1 + x2)]]^2*(-y1 + y2))/((1 + (-y1 + y2)^2/(-x1\
           + x2)^2)*(-x1 + x2)^2)) + ((-2(AE*Cos[Atan[(-y1 + y2)\
          /(-x1 + x2)]]*(-y1 + y2)^2*Sin[Atan[(-y1 + y2)/(-x1 + x2)]\
          ]) + -(AE*Cos[Atan[(-y1 + y2)/(-x1 + x2)]]^2*(-y1 + y2)\
          *(2((-y1 + y2)^2/(-x1 + x2)) + -2((1 + (-y1 + y2)^2/(-x1\
           + x2)^2)*(-x1 + x2)))))/((1 + (-y1 + y2)^2/(-x1 + x2)\
          ^2)^2*(-x1 + x2)^4) + -((2(AE*Cos[Atan[(-y1 + y2)/(-x1\
           + x2)]]*(-y1 + y2)^2*Sin[Atan[(-y1 + y2)/(-x1 + x2)]]))\
           + -(AE*(-y1 + y2)*(2((-y1 + y2)^2/(-x1 + x2)) + -2((1\
           + (-y1 + y2)^2/(-x1 + x2)^2)*(-x1 + x2)))*Sin[Atan[(-y1\
           + y2)/(-x1 + x2)]]^2))/((1 + (-y1 + y2)^2/(-x1 + x2)\
```

```
^2)^2*(-x1 + x2)^4)))*((-x1 + x2)^2 + (-y1 + y2)^2) + -(AE\
*Cos[Atan[(-y1 + y2)/(-x1 + x2)]]*Sin[Atan[(-y1 + y2)\
/(-x1 + x2)]])))/((-x1 + x2)^2 + (-y1 + y2)^2)\
^(5/2)],{(-3/2((2x1 + -2x2)*((-((AE*(-y1 + y2)*Sin[Atan[(-y1\
 + y2)/(-x1 + x2)]]^2)/((1 + (-y1 + y2)^2/(-x1 + x2)^2)\
*(-x1 + x2)^2)) + (AE*Cos[Atan[(-y1 + y2)/(-x1 + x2)]]\
^2*(-y1 + y2))/((1 + (-y1 + y2)^2/(-x1 + x2)^2)*(-x1 + x2)\
^2))*((-x1 + x2)^2 + (-y1 + y2)^2) + -1/2(AE*Cos[Atan[(-y1\
 + y2)/(-x1 + x2)]]*(2x1 + -2x2)*Sin[Atan[(-y1 + y2)/(-x1\
 + x2)]])))) + ((-x1 + x2)^2 + (-y1 + y2)^2)*(1/2((AE*(2x1\
 + -2x2)*(-y1 + y2)*Sin[Atan[(-y1 + y2)/(-x1 + x2)]]^2)\
/((1 + (-y1 + y2)^2/(-x1 + x2)^2)*(-x1 + x2)^2)) + -1/2((\
AE*Cos[Atan[(-y1 + y2)/(-x1 + x2)]]^2*(2x1 + -2x2)*(-y1\
 + y2))/((1 + (-y1 + y2)^2/(-x1 + x2)^2)*(-x1 + x2)^2))\
 + (2x1 + -2x2)*(-((AE*(-y1 + y2)*Sin[Atan[(-y1 + y2)\
/(-x1 + x2)]]^2)/((1 + (-y1 + y2)^2/(-x1 + x2)^2)*(-x1\
 + x2)^2)) + (AE*Cos[Atan[(-y1 + y2)/(-x1 + x2)]]^2*(-y1\
 + y2))/((1 + (-y1 + y2)^2/(-x1 + x2)^2)*(-x1 + x2)^2))\
 + ((-2(AE*Cos[Atan[(-y1 + y2)/(-x1 + x2)]]*(-y1 + y2)\
^2*Sin[Atan[(-y1 + y2)/(-x1 + x2)]]) + -(AE*Cos[Atan[(-y1\
 + y2)/(-x1 + x2)]]^2*(-y1 + y2)*(2((-y1 + y2)^2/(-x1\
 + x2)) + -2((1 + (-y1 + y2)^2/(-x1 + x2)^2)*(-x1 + x2))))\
)/((1 + (-y1 + y2)^2/(-x1 + x2)^2)^2*(-x1 + x2)^4) + -((2(\
AE*Cos[Atan[(-y1 + y2)/(-x1 + x2)]]*(-y1 + y2)^2*Sin[Atan[\
(-y1 + y2)/(-x1 + x2)]]) + -(AE*(-y1 + y2)*(2((-y1 + y2)\
^2/(-x1 + x2)) + -2((1 + (-y1 + y2)^2/(-x1 + x2)^2)*(-x1\
 + x2)))*Sin[Atan[(-y1 + y2)/(-x1 + x2)]]^2))/((1 + (-y1\
 + y2)^2/(-x1 + x2)^2)^2*(-x1 + x2)^4)))*((-x1 + x2)^2\
 + (-y1 + y2)^2) + -(AE*Cos[Atan[(-y1 + y2)/(-x1 + x2)]]\
*Sin[Atan[(-y1 + y2)/(-x1 + x2)]])))/((-x1 + x2)^2\
+ (-y1 + y2)^2)^(5/2),(-3/2((2x1 + -2x2)*(2((AE*Cos[Atan[(-\
y1 + y2)/(-x1 + x2)]]*(-y1 + y2)*((-x1 + x2)^2 + (-y1\
 + y2)^2)*Sin[Atan[(-y1 + y2)/(-x1 + x2)]])/((1 + (-y1\
 + y2)^2/(-x1 + x2)^2)*(-x1 + x2)^2)) + -1/2(AE*(2x1 + -2\
x2)*Sin[Atan[(-y1 + y2)/(-x1 + x2)]]^2))) + ((-x1 + x2)\
^2 + (-y1 + y2)^2)*(-((AE*Cos[Atan[(-y1 + y2)/(-x1 + x2)]]\
*(2x1 + -2x2)*(-y1 + y2)*Sin[Atan[(-y1 + y2)/(-x1 + x2)]])\
/((1 + (-y1 + y2)^2/(-x1 + x2)^2)*(-x1 + x2)^2)) + 2(((1\
 + (-y1 + y2)^2/(-x1 + x2)^2)*(-x1 + x2)^2*((AE*Cos[Atan[(\
-y1 + y2)/(-x1 + x2)]]^2*(-y1 + y2)^2*((-x1 + x2)^2 + (-y1\
 + y2)^2))/((1 + (-y1 + y2)^2/(-x1 + x2)^2)*(-x1 + x2)\
^2) + -((AE*(-y1 + y2)^2*((-x1 + x2)^2 + (-y1 + y2)^2)\
*Sin[Atan[(-y1 + y2)/(-x1 + x2)]]^2)/((1 + (-y1 + y2)\
^2/(-x1 + x2)^2)*(-x1 + x2)^2)) + AE*Cos[Atan[(-y1 + y2)\
/(-x1 + x2)]]*(2x1 + -2x2)*(-y1 + y2)*Sin[Atan[(-y1 + y2)\
/(-x1 + x2)]]) + -(AE*Cos[Atan[(-y1 + y2)/(-x1 + x2)]]\
*(-y1 + y2)*(2((-y1 + y2)^2/(-x1 + x2)) + -2((1 + (-y1\
 + y2)^2/(-x1 + x2)^2)*(-x1 + x2)))*((-x1 + x2)^2 + (-y1\
 + y2)^2)*Sin[Atan[(-y1 + y2)/(-x1 + x2)]]))/((1 + (-y1\
 + y2)^2/(-x1 + x2)^2)^2*(-x1 + x2)^4)) + -(AE*Sin[Atan[(-\
y1 + y2)/(-x1 + x2)]]^2)))/((-x1 + x2)^2 + (-y1 + y2)\
^2)^(5/2)}}

#I[7]::  Exit[]
```

******** S M P   O U T P U T   6 ********

```
SMP 1.5.0
19-OCT-1986 12:30:18.05

#I[1]::  K:{{c^2,s c},{s c,s^2}}/L /*Define stiffness matrix*/

            2            2
           c   c s   c s  s
#O[1]:   {{--,---},{---,--}}
           L    L     L   L

#I[2]::  c:Cos[theta]; s:Sin[theta]; /*Define c = Cos and s = Sin */

#I[3]::  <"[-.smp_prog]trig_1";      /*Load trigonomitric functions I*/

#I[4]::  K:K                         /*Simplifying stiffness matrix K*/

           1 + Cos[2theta] Sin[2theta]   Sin[2theta] 1 - Cos[2theta]
#O[4]:   {{---------------,-----------},{-----------,---------------}}
                 2L             2L            2L            2L

#I[5]::  Put[K,"k.sav"]; /*Save stiffness matrix onto disk*/

#I[6]::  Set[]  /*Clear all of the internal memories*/

#I[7]::  <"k.sav"; /*Reload stiffness matrix back*/

#I[8]::  theta:Atan[(y2-y1)/(x2-x1)]; /*Define the truss element angle*/

#I[9]::  L:Sqrt[(x2-x1)^2+(y2-y1)^2]; /*Define element length*/

#I[10]::  K:K                         /*Obtain K in long hand form*/

                               -y1 + y2
                1 + Cos[2Atan[--------]]
                               -x1 + x2
#O[10]:   {{--------------------------------,
                          2              2 1/2
            2 ((-x1 + x2)  + (-y1 + y2) )

                                      -y1 + y2
                           Sin[2Atan[--------]]
                                      -x1 + x2
                      --------------------------------},
                                     2              2 1/2
                       2 ((-x1 + x2)  + (-y1 + y2) )

                        -y1 + y2
             Sin[2Atan[--------]]
                        -x1 + x2
          {--------------------------------,
                         2              2 1/2
           2 ((-x1 + x2)  + (-y1 + y2) )
```

$$1 - \cos\left[2\operatorname{Atan}\left[\frac{-y1 + y2}{-x1 + x2}\right]\right]$$
$$\left.\left.\frac{}{2\left((-x1 + x2)^2 + (-y1 + y2)^2\right)^{2\,1/2}}\right]\right\}$$

#I[11]:: Kdx1:D[K,x1] /*Derivative of K with respect to x1*/

$$\frac{-2(-y1 + y2)\left((-x1 + x2)^2 + (-y1 + y2)^2\right)\sin\left[2\operatorname{Atan}\left[\frac{-y1 + y2}{-x1 + x2}\right]\right]}{\left(1 + \frac{(-y1 + y2)^2}{(-x1 + x2)^2}\right)(-x1 + x2)^2}$$
$$- \frac{\left(1 + \cos\left[2\operatorname{Atan}\left[\frac{-y1 + y2}{-x1 + x2}\right]\right]\right)(2x1 - 2x2)}{2}$$

#O[11]: $$\left\{\left\{\frac{}{2\left((-x1 + x2)^2 + (-y1 + y2)^2\right)^{2\,3/2}}\right.\right.$$
$$2\cos\left[2\operatorname{Atan}\left[\frac{-y1 + y2}{-x1 + x2}\right]\right](-y1 + y2)$$
$$* \left((-x1 + x2)^2 + (-y1 + y2)^2\right)$$
$$\frac{}{-\left(1 + \frac{(-y1 + y2)^2}{(-x1 + x2)^2}\right)(-x1 + x2)^2}$$
$$(2x1 - 2x2)\sin\left[2\operatorname{Atan}\left[\frac{-y1 + y2}{-x1 + x2}\right]\right]$$
$$- \frac{}{2}$$
$$\left.\left.\frac{}{2\left((-x1 + x2)^2 + (-y1 + y2)^2\right)^{2\,3/2}}\right\}\right.$$

$$\frac{2\cos\left[2\operatorname{Atan}\left[\frac{-y1 + y2}{-x1 + x2}\right]\right](-y1 + y2)\left((-x1 + x2)^2 + (-y1 + y2)^2\right)}{\left(1 + \frac{(-y1 + y2)^2}{(-x1 + x2)^2}\right)(-x1 + x2)^2}$$

$$\frac{(2x1 - 2x2)\,\sin\left[2\,\text{Atan}\left[\frac{-y1 + y2}{-x1 + x2}\right]\right]}{2}$$

$$\left\{\frac{\dots}{2\left((-x1 + x2)^2 + (-y1 + y2)^2\right)^{3/2}},\right.$$

$$\frac{2(-y1 + y2)\left((-x1 + x2)^2 + (-y1 + y2)^2\right)}{}$$

$$* \sin\left[2\,\text{Atan}\left[\frac{-y1 + y2}{-x1 + x2}\right]\right]$$

$$\frac{}{\left(1 + \frac{(-y1 + y2)^2}{(-x1 + x2)^2}\right)(-x1 + x2)^2}$$

$$\frac{(1 - \cos\left[2\,\text{Atan}\left[\frac{-y1 + y2}{-x1 + x2}\right]\right])(2x1 - 2x2)}{2}$$

$$\left.\frac{}{2\left((-x1 + x2)^2 + (-y1 + y2)^2\right)^{3/2}}\right\}\}$$

```
#I[12]::  L: ;theta:;  /*Clear previous definitions of "theta" and "L"*/

#I[13]::  x2-x1:dx;y2-y1:dy;     /*Define dx and dy*/

#I[14]::  dx^2+dy^2:L^2;         /*Define L backward*/

#I[15]::  Atan[dy/dx]:theta;     /*Define theta backward*/

#I[16]::  Kdx1:Kdx1              /*Simplifying Kdx1 ... Stage I*/
```

$$\frac{-2dy\,L^2\,\sin[2\text{theta}]}{2} - \frac{(1 + \cos[2\text{theta}])(2x1 - 2x2)}{2}$$

$$\frac{}{dx^2\left(1 + \left(\frac{dy}{dx}\right)^2\right)}$$

$$\#O[16]: \quad \left\{\left\{\frac{\dots}{2L^3},\right.\right.$$

$$\frac{2dy\,L^2\cos[2\text{theta}]}{2} - \frac{(2x1 - 2x2)\sin[2\text{theta}]}{2}$$

$$\left.\left.\frac{}{dx^2\left(1 + \left(\frac{dy}{dx}\right)^2\right)}\right\}\right.$$

```
                          dx
           --------------------------------------------},
                                             3
                                          2 L

       2
 2dy L   Cos[2theta]     (2x1 - 2x2) Sin[2theta]
 -------------------  -  -----------------------
               2                    2
      2      dy
    dx   (1 + ---)
               2
             dx
{---------------------------------------------,
                                 3
                              2 L

        2
  2dy L   Sin[2theta]     (1 - Cos[2theta]) (2x1 - 2x2)
  -------------------  -  -----------------------------
                2                      2
       2      dy
     dx   (1 + ---)
                2
              dx
  ----------------------------------------------------}}
                                       3
                                    2 L

#I[17]::  2x1-2x2:-2dx; (1+dy^2/dx^2):(dx^2+dy^2)/dx^2;

#I[18]::  Kdx1:Kdx1       /*Simplifying Kdx1 .... Stage II*/

            dx (1 + Cos[2theta]) - 2dy Sin[2theta]
#O[18]:   {{-------------------------------------,
                                 3
                              2 L

                         dx Sin[2theta] + 2dy Cos[2theta]
                         --------------------------------},
                                            3
                                         2 L

           dx Sin[2theta] + 2dy Cos[2theta]
          {--------------------------------,
                              3
                           2 L

                     dx (1 - Cos[2theta]) + 2dy Sin[2theta]
                     --------------------------------------}}
                                            3
                                         2 L

#I[19]::  dx:L Cos[theta]; dy:L Sin[theta]; /*Define dx and dy*/

#I[20]::  Kdx1:Ex[Kdx1]     /*Simplifying Kdx1 .... Stage III*/

           Cos[theta]   Cos[theta] Cos[2theta]   Sin[theta] Sin[2theta
#O[20]:  {{---------- + ---------------------- - ---------------------
                 2                  2                        2
              2 L                2 L                        L
```

```
                                    Cos[theta] Sin[2theta]   Cos[2theta] Sin[theta]
                                    ---------------------- + ---------------------},
                                                 2                        2
                                              2 L                        L

                  Cos[theta] Sin[2theta]   Cos[2theta] Sin[theta]
                 {---------------------- + ----------------------,
                               2                        2
                            2 L                        L

                                    Cos[theta]   Cos[theta] Cos[2theta]
                                    ---------- - ----------------------
                                          2                   2
                                       2 L                 2 L

                                                     Sin[theta] Sin[2theta]
                                                   + ----------------------}}
                                                                  2
                                                                 L

#I[21]::  <"[-.smp_prog]trig_1";     /*Load trigonomitric functions*/

#I[22]::  Kdx1:Kdx1      /*Simplifying Kdx1 .... Stage IV*/

           Cos[theta]   -Cos[-theta] + Cos[3theta]
#O[22]:   {{---------- + -------------------------
                  2                   2
               2 L                 2 L

                                            Cos[-theta] + Cos[3theta]
                                          + -------------------------,
                                                          2
                                                       4 L

               -Sin[-theta] + Sin[3theta]
               --------------------------
                              2
                           4 L

                                           -Sin[theta] + Sin[3theta]
                                         + -------------------------},
                                                         2
                                                      2 L

             -Sin[-theta] + Sin[3theta]   -Sin[theta] + Sin[3theta]
            {-------------------------- + -------------------------,
                           2                            2
                        4 L                          2 L

                             Cos[theta]   -Cos[-theta] + Cos[3theta]
                             ---------- - --------------------------
                                   2                    2
                                2 L                  2 L

                                             Cos[-theta] + Cos[3theta]
                                           - -------------------------}
                                                           2
                                                        4 L

#I[23]::  <"[-.smp_prog]trig_2";     /*trigonomitric functions II*/

#I[24]::  Kdx1:Ex[Kdx1] /*Finial symplification of Kdx1 .... Stage V*
```

```
#O[24]:   {{Cos[theta]/(4 L^2) + 3Cos[3theta]/(4 L^2), -Sin[theta]/(4 L^2) + 3Sin[3theta]/(4 L^2)},

           {-Sin[theta]/(4 L^2) + 3Sin[3theta]/(4 L^2), 3Cos[theta]/(4 L^2) - 3Cos[3theta]/(4 L^2)}}

#I[25]:: Put[Kdx1,"Kdx1.sav"]; /*Save the result onto disk*/

#I[26]:: Set[];    /*Reset all of the internal memories*/

#I[27]:: <"Kdx1.sav";     /*Reload Kdx1 from the disk*/

#I[28]:: theta:Atan[(y2-y1)/(x2-x1)]; L:Sqrt[(x2-x1)^2+(y2-y1)^2];

#I[29]:: Kdx1:Kdx1;       /*Get the long hand form of Kdx1*/

#I[30]:: Kddx1:D[Kdx1,x1]; /*Second derivative of K*/

#I[31]:: theta:; L:;      /*Reset "theta" and "L" values*/

#I[32]:: y2-y1:dy; x2-x1:dx; dx^2+dy^2:L^2; Atan[dy/dx]:theta;

#I[33]:: 1+dy^2/dx^2:(dx^2+dy^2)/dx^2; 2x1-2x2:-2dx;

#I[34]:: Kddx1:Kddx1;     /*Simplification of Kddx1 .... Stage I*/

#I[35]:: dx:L Cos[theta]; dy:L Sin[theta];

#I[36]:: Kddx1:Ex[Kddx1]; /*Simplifications of Kddx1 .... Stage II*/

#I[37]:: <"[-.smp_prog]trig_1"; /*Load trigonomitrical functions*/

#I[38]:: Kddx1:Kddx1      /*Simplifications of Kddx1 .... Stage III*/

#O[38]:   {{-(1 - Cos[2theta])/(8 L^3) + (1 + Cos[2theta])/(4 L^3)

            + 9(-Cos[-2theta] + Cos[4theta])/(8 L^3)

            + 3(Cos[-2theta] + Cos[4theta])/(4 L^3),

           3(-Sin[-2theta] + Sin[4theta])/(4 L^3)

            + 9(-Sin[2theta] + Sin[4theta])/3
```

```
                                                          8 L
                                                                3Sin[2theta]
                                                              - ------------},
                                                                      3
                                                                   8 L

                 3(-Sin[-2theta] + Sin[4theta])
                {------------------------------
                                    3
                                 4 L
                                          9(-Sin[2theta] + Sin[4theta])
                                        + -----------------------------
                                                         3
                                                      8 L
                                                                 3Sin[2theta]
                                                               - ------------,
                                                                       3
                                                                    8 L

                -3(1 - Cos[2theta])   3(1 + Cos[2theta])
                ------------------- + ------------------
                             3                    3
                          8 L                  4 L
                                 9(-Cos[-2theta] + Cos[4theta])
                               - ------------------------------
                                                 3
                                              8 L
                                          3(Cos[-2theta] + Cos[4theta])
                                        - -----------------------------}}
                                                          3
                                                       4 L

#I[39]::  <"[-.smp_prog]trig_2"; /*trigonomitric functions II*/

#I[40]::  Kddx1:Ex[Kddx1]     /*Final simplifications .... Stage IV*/

               1      15Cos[4theta] -3Sin[2theta]    15Sin[4theta]
#O[40]:   {{---- + -------------,------------- + -------------},
                3              3             3               3
             8 L            8 L           4 L             8 L

            -3Sin[2theta]   15Sin[4theta]
          .{------------- + -------------,
                     3               3
                  4 L             8 L

                                   3      3Cos[2theta]   15Cos[4theta]
                                  ---- + ------------ - -------------}}
                                     3            3               3
                                  8 L          2 L             8 L

#[41]::  Exit[]   /*Calculates are finished*/
```

```
                ******** S M P    O U T P U T    7 . ********
SMP 1.5.0
20-OCT-1986 18:57:18.05

#I[1]::  <"smp.dat";  /*Load the output data from Example 4*/

#I[2]::  Prog[k[1,1]] /* Translate SMP output into C language */

      double k() ;
      double t_1 ;

      t_1 = k(1.0000000000000000e+00,1.0000000000000000e+00) ;

#O[2]:   {}

#I[3]::  Prog[k11]    /*The first element of [K]*/

      double angl = 0.0 ;
      double L = 0.0 ;
      double k11 = 0.0 ;
      double cos() ;
      double t_1 ;

      k11 = 5.0000000000000000e-01 * ((1.0000000000000000e+00 +
cos(2.0000000000000000e+00 * angl)) / L) ;
      t_1 = k11 ;

#O[3]:   {}

#I[4]::  Prog[dk4]   /*The fourth derivative of [K]*/

      double L = 0.0 ;
      double angl = 0.0 ;
      double pow() ;
      double cos() ;
      double dk4[3][3] ;
      double sin() ;
      double t_1 ;

      dk4[1][1] = 5.6250000000000000e-01 * (1.0000000000000000e+00
/ pow(L,5.0000000000000000e+00)) + 4.6875000000000000e-01 * (cos(
2.0000000000000000e+00 * angl) / pow(L,5.0000000000000000e+00)) +
-6.5625000000000000e+00 * (cos(4.0000000000000000e+00 * angl) / pow(L,
5.0000000000000000e+00)) + 2.9531250000000000e+01 * (cos(
6.0000000000000010e+00 * angl) / pow(L,5.0000000000000000e+00)) ;
      dk4[1][2] = -2.3437500000000000e+00 *
(sin(2.0000000000000000e+00 * angl) / pow(L,5.0000000000000000e+00))
+ -1.3125000000000000e+01 * (sin(4.0000000000000000e+00 * angl)
/ pow(L,5.0000000000000000e+00)) + 2.9531250000000000e+01 *
(sin(6.0000000000000010e+00 * angl) / pow(L,5.0000000000000000e+00))
      dk4[2][1] = -2.3437500000000000e+00 *
(sin(2.0000000000000000e+00 * angl) / pow(L,5.0000000000000000e+00))
+ -1.3125000000000000e+01 * (sin(4.0000000000000000e+00 * angl) /
pow(L,5.0000000000000000e+00)) + 2.9531250000000000e+01 *
(sin(6.0000000000000010e+00 * angl) / pow(L,5.0000000000000000e+00))
      dk4[2][2] = 2.8125000000000000e+00 * (1.0000000000000000e+00
/ pow(L,5.0000000000000000e+00)) + 7.0312500000000000e+00 * (cos(
2.0000000000000000e+00 * angl) / pow(L,5.0000000000000000e+00)) +
1.9687500000000000e+01 * (cos(4.0000000000000000e+00 * angl) / pow(L,
```

```
5.000000000000000e+00)) + -2.95312500000000000e+01 * (cos(
6.000000000000010e+00 * angl) / pow(L,5.000000000000000e+00)) ;
      t_1 = dk4[2][2] ;

#O[4]:   {}

#I[5]::  Prog[k11,,,2] /* Translate SMP output into FORTRAN language */

      DOUBLE PRECISION ANGL
      DOUBLE PRECISION UL
      DOUBLE PRECISION K11
      DOUBLE PRECISION T1

      K11 = 5.0000000000000000D-01 * ((1.0000000000000000D+00 + DCOS(
     $    2.0000000000000000D+00 * ANGL)) / UL)
      T1 = K11

#O[5]:   {}

#I[6]::  Prog[dk4,,,2]

      DOUBLE PRECISION UL
      DOUBLE PRECISION ANGL
      DIMENSION DK4(2,2)
      DOUBLE PRECISION DK4
      DOUBLE PRECISION T1

      DK4(1,1) = 5.6250000000000000D-01 * (1.0000000000000000D+00 /
     $    UL ** 5.0000000000000000D+00) + 4.6875000000000000D-01 * (
     $    DCOS(2.0000000000000000D+00 * ANGL) / UL **
     $    5.0000000000000000D+00) + -6.5625000000000000D+00 * (DCOS(
     $    4.0000000000000000D+00 * ANGL) / UL **
     $    5.0000000000000000D+00) + 2.9531250000000000D+01 * (DCOS(
     $    6.0000000000000010D+00 * ANGL) / UL **
     $    5.0000000000000000D+00)
      DK4(1,2) = -2.3437500000000000D+00 * (DSIN(
     $    2.0000000000000000D+00 * ANGL) / UL **
     $    5.0000000000000000D+00) + -1.3125000000000000D+01 * (DSIN(
     $    4.0000000000000000D+00 * ANGL) / UL **
     $    5.0000000000000000D+00) + 2.9531250000000000D+01 * (DSIN(
     $    6.0000000000000010D+00 * ANGL) / UL **
     $    5.0000000000000000D+00)
      DK4(2,1) = -2.3437500000000000D+00 * (DSIN(
     $    2.0000000000000000D+00 * ANGL) / UL **
     $    5.0000000000000000D+00) + -1.3125000000000000D+01 * (DSIN(
     $    4.0000000000000000D+00 * ANGL) / UL **
     $    5.0000000000000000D+00) + 2.9531250000000000D+01 * (DSIN(
     $    6.0000000000000010D+00 * ANGL) / UL **
     $    5.0000000000000000D+00)
      DK4(2,2) = 2.8125000000000000D+00 * (1.0000000000000000D+00 /
     $    UL ** 5.0000000000000000D+00) + 7.0312500000000000D+00 *
     $    DCOS(2.0000000000000000D+00 * ANGL) / UL **
     $    5.0000000000000000D+00) + 1.9687500000000000D+01 * (DCOS(
     $    4.0000000000000000D+00 * ANGL) / UL **
     $    5.0000000000000000D+00) + -2.9531250000000000D+01 * (DCOS
     $    6.0000000000000010D+00 * ANGL) / UL **
     $    5.0000000000000000D+00)
      T1 = DK4(2,2)

#O[6]:   {}
```

APPENDIX C
LIST OF SMP PROGRAMS

PROGRAMS PAGE

SMP Programs for Finite Element Analysis

General SMP Programs

```
/**            P R O G R A M :   A S S E M B                    **/
/**                                              June 2, 1985   **/
/**                                                              \
/**  A SMP PROGRAM TO ASSEMBLE THE GLOBAL STIFFNESS              **/
     MATRIX FROM GIVEN ELEMENT MATRICES                         **/
                                                                **/
/**  Assemb[lclk,list,dof]                                      **/
/**        lclk = element matrices {k1,k2,k3,.....}             **/
/**               where k1,k2,... are element matrices          **/
/**               of element 1 and 2, respectively              **/
/**               They are in a full matrix form                **/
/**        list = nodal number listing of each element          **/
/**         dof = degree of freedom at each node                **/
/**                                                             **/
     <FndMax
     Assemb[$lclk,$list,$dof]::Proc[\
           Lcl[%ndof,%ndim,%nods,%glob,%i,%j,%row,%col,\
               %k,%elr,%elc,%rw,%l,%nd,%adj],\
/** Determine Number of Elements and Number of Nodes          **^\
/** on each element                                           **^\
           %ndof:$dof,\
           %ndim:Dim[$lclk],\
           %nelm:%ndim[1],\
           %nods:%ndim[2]/%ndof,\
           Pr["  Assembling The Global Matrix ...."],\
           Pr[" "],\
           Pr["       Total Number of Elements   =",%nelm],\
           Pr["       Number of Nodes on element =",%nods],\
           Pr["       D.O.F. for each node       =",%ndof],\
           Pr[" "],\
/** Assembling the Global Matrix element by element           **^\
           %nd:FndMax[$list],\
           %nd:%ndof %nd,\
           %glob:Ar[{%nd,%nd},0],\
           For[%i:1,%i<=%nelm,Inc[%i],Proc[\
               Pr[" Processing Element # ",%i],\
               For[%j:1,%j<=%nods,Inc[%j],Proc[\
/** Find Row Numbers in local and global systems              **^\
                   %row:(%ndof ($list[%i,%j]-1)),\
                   %elr:%ndof (%j-1),\
                   For[%rw:1,%rw<=%ndof,Inc[%rw],Proc[\
                       %row:%row+1,\
                       %elr:%elr+1,\
/** Find Column Numbers in local and global systems           **^\
                       For[%k:%j,%k<=%nods,Inc[%k],Proc[\
                       %col:(%ndof ($list[%i,%k]-1)),\
                       %elc:%ndof (%k-1),\
                       For[%l:1,%l<=%ndof,Inc[%l],Proc[\
                           %col:%col+1,\
                           %elc:%elc+1,\
                           If[%col>=%row,Proc[\
/** Add element matrix into the global system element         **^\
/** by element                                                **^\
                       %glob[%row,%col]:\
                               %glob[%row,%col]+\
                               $lclk[%i,%elr,%elc],\
                       If[%col~=%row,%glob[%col,%row]:\
                                     %glob[%row,%col]],\
                       ]],\
                   ]]]],\
```

```
        ]]]],\
    ]],\
%glob\
]
```

```
/**          P R O G R A M :   A S S E M B 1                    **/
/**                                                 June 2, 1985 **/
/**  A SMP PROGRAM TO ASSEMBLE THE GLOBAL STIFFNESS
     MATRIX FROM GIVEN ELEMENT MATRICES AND BOUNDARY
     CONDITIONS.                                                **/
/**  Assembl[lclk,list,dof,lstdof]                              **/
/**        lclk = element matrices {k1,k2,k3,.....}             **/
/**               where k1,k2,... are element matrices          **/
/**               of element 1 and 2, respectively              **/
/**               They are in a full matrix form                **/
/**        list = nodal number listing of each element          **/
/**         dof = the maximum number degree of freedom          **/
/**               on the node                                   **/
/**      lstdof = a list of the degree of freedom after         **/
/**               eliminating the restrains and being           **/
/**               generated by DOFlst.                          **/
/**                                                             **/
     <FndMax
     Assembl[$lclk,$list,$dof,$lstdof]::Proc[\
           Lcl[%ndof,%ndim,%nods,%glob,%i,%j,%row,%col,\
               %k,%elr,%elc,%rw,%l,%nd,%adj],\
/** Determine Number of Elements and Number of Nodes           **^\
/** on each element                                            **^\
           %ndof:$dof,\
           %ndim:Dim[$lclk],\
           %nelm:%ndim[1],\
           %nods:%ndim[2]/%ndof,\
           Pr["  Assembling The Global Matrix ...."],\
           Pr[" "],\
           Pr["       Total Number of Elements   =",%nelm],\
           Pr["       Number of Nodes on element =",%nods],\
           Pr["       D.O.F. for each node       =",%ndof],\
           Pr[" "],\
/** Assembling the Global Matrix element by element            **^\
           %nd:FndMax[$list],\
           %nd:%ndof %nd,\
           %glob:Ar[{%nd,%nd},0],\
           For[%i:1,%i<=%nelm,Inc[%i],Proc[\
               Pr[" Processing Element # ",%i],\
               For[%j:1,%j<=%nods,Inc[%j],Proc[\
/** Find Row Numbers in local and global systems               **^\
               %row:(%ndof ($list[%i,%j]-1)),\
               %elr:%ndof (%j-1),\
               For[%rw:1,%rw<=%ndof,Inc[%rw],Proc[\
                   %row:%row+1,\
                   %elr:%elr+1,\
                   %row:$lstdof[%row],\
                   If[%row:0,Jump[%loopr]],\
/** Find Column Numbers in local and global systems            **/\
                   For[%k:%j,%k<=%nods,Inc[%k],Proc[\
                       %col:(%ndof ($list[%i,%k]-1)),\
                       %elc:%ndof (%k-1),\
                   For[%l:1,%l<=%ndof,Inc[%l],Proc[\
                       %col:%col+1,\
                       %elc:%elc+1,\
                       %col:$lstdof[%col],\
                       If[%col=0,Jmp[%loopc]],\
                       If[%col>=%row,Proc[\
```

```
/** Add element matrix into the global system element  **^\
/** by element                                          **^\
                          %glob[%row,%col]:\
                               %glob[%row,%col]+\
                                  $lclk[%i,%elr,%elc],\
                          If[%col~=%row,%glob[%col,%row]:\
                             %glob[%row,%col]],\
                         ]],\
                       Lbl[%loopc],\
                      ]]]],\
                  Lbl[%loopr],\
                 ]]]],\
            ]],\
         %glob\
         ]
```

```
/*                P R O G R A M :    D O F B A R                    */
/*                                                    May 15, 1985  */
/*                                                                  */
/*  Program to generate a list of number of freedoms for a          */
/*  2-D truss problem                                               */
/*                                                                  */
/*  Filters:                                                        */
/*  ---------------------- INPUT DATA ------------------------      */
/*    $nod ----- a Nod matrix which contains a list of              */
/*               end point conditions for both end points,          */
/*               respectively; ie.                                  */
/*             $nod[i] = 0 -----> nodal #i is fixed                 */
/*             $nod[i] = 1 -----> nodal #i has x-freedom            */
/*             $nod[i] = 2 -----> nodal #i has y-freedom            */
/*             $nod[i] = 3 -----> nodal #i has x and y freedoms     */
/*             where Nod is Number of Nodal Points                  */
/*  ---------------------- OUTPUT DATA -----------------------      */
/*    $lst ----- a list which contains the numbered degree of       */
/*               freedom.                                           */
/*------------------------------------------------------------------*/
/*                                                                  */
   DOFbar[$nod,$lst] :: Proc[\
          Lcl[%i,%len,%count,%nl,%add];\
          %add[$i]::Proc[\
                         %count:%count+1;\
                         %nl[$i]:%count;\
                        ];\
          %len:Len[$nod];\
          %count:0;\
          %nl:Ar[2 %len,0];\
          For[%i:1,%i<=%len,Inc[%i],Proc[\
              Lcl[%k1,%k2];\
              %k1:2 (%i-1)+1;\
              %k2:%k1+1;\
              If[$nod[%i]=1,%add[%k1]];\
              If[$nod[%i]=2,%add[%k2]];\
              If[$nod[%i]=3,Proc[\
                 %add[%k1];\
                 %add[%k2];\
                ]];\
             ]];\
           $lst:%nl\
         ]
/*  ---------------------- END OF FILE -----------------------      */
```

```
/*                P R O G R A M :   D O F L S T                 */
/*                                                              */
/*  Program to generate a list of the number of freedoms in     */
/*  any 1-D, 2-D, or 3-D finite element or structure problem    */
/*  for assembling global matrix purpose                        */
/*  The sequence of DOF at the node is (u,v,w,rx,ry,rz)         */
/*                                     or (1,2,3, 4, 5, 6)      */
/*                                                              */
/*  DOFlst[$nod]                                                */
/*  Filters:                                                    */
/*           $nod = list of {{0/1,0/1,0/1,0/1,0/1,0/1},....}    */
/*                  where 0 ---> free in this direction         */
/*                        1 ---> restrained in this direction   */
/*                                                              */
    DOFlst[$nod]::Proc[\
            Lcl[%i,%len,%count,%nl,%add];\
            %add[$kdof]::Proc[\
                              %count:%count+1;\
                              %nl[$kdof]:%count;\
                             ];\
            %nl:Flat[$nod];\
            %len:Len[%nl];\
            Pr[" %len =",%len];\
            %count:0;\
            For[%i:1,%i<=%len,Inc[%i],Proc[\
                If[%nl[%i]=0,%add[%i],%nl[%i]:0];\
               ]];\
            %nl\
          ]
/*  ---------------------- END OF FILE ----------------------   */
```

```
/*                  P R O G R A M :   D O F N O D                  */
/*                                                                 */
/*  Program to generate a list of the number of freedoms in        */
/*  any 1-D, 2-D, or 3-D finite element or structure problem       */
/*  for assembling global matrix purpose                           */
/*  The sequence of DOF at the node is specified by the user       */
/*  as $seqn and the value is set as 0/1 such as 0---> free        */
/*  1 ---> restrained for each DOF                                 */
/*                                                                 */
/*  DOFNod[$nod,$dof,$dim]                                         */
/*       $nod ----- an Array which contains a list of nodal        */
/*                  point conditions as the following :            */
/*            $nod[i] = 0 ---> node is free                        */
/*                    = 1/2 ---> no translation/rotation           */
/*                    = 3/4/5 ---> no x/y/z-displacement           */
/*                    = 7/8/9 ---> no x/y/z-rotation               */
/*                    = 1+7/8/9 ---> no translation+x/y/z rot      */
/*                    = 2+3/4/5 ---> no rotation + x/y/z tran      */
/*             235 ---> node is restrained except y-translat.      */
/*             107 ---> node has y and z-rotations                 */
/*       $dof ----- total degree of freedom on a free node         */
/*       $dim ----- the dimension of the problem                   */
/*                                                                 */
    DOFNod[$nod,$dof,$dim]::Proc[\
          Lcl[%nod,%seq,%inod,%eqone],\
          %eqone[$row,$cstr,$cend]::Proc[Lcl[%i],\
                 For[%i:$cstr,%i<=$cend,Inc[%i],\
                     %seq[$row,%i]:1,\
                    ],\
                ],\
          %nod:Len[$nod],\
          %seq:Ar[{%nod,$dof},0],\
          For[%inod:1,%inod<=$dof,Inc[%inod],Proc[\
              Llc[%dof,%loop,%twod,%threed,
              %dof:$nod[%i],\
              If[%dof=0,Jmp[%loop]],\
              If[%dof=1,%eqone[%i,1,$dim]],\
              If[%dof=2,%eqone[%i,$dim+1,$dof]],\
              If[%dof>=3,Proc[\
                 Lbl[%twod],\
                 Jmp[%loop],\
                 Lbl[%threed],\

                 Jmp[%loop],\
                 ],\
                ],\
            Lbl[%loop],\
              ]],\
           %seq\
          ]
```

```
/*                    P R O G R A M :   D R V T G S                    */
/*                                                        June 10,1985 */
/* A SMP PROGRAM TO EVALUATE THE DERIVATIVES OF THE INTERPOLAT-        */
/* ION FUNCTIONS NUMERICALY AT A GAUSS POINT. THE INTERPOLATION        */
/* FUNCTIONS ARE DEFINED IN THE NATURAL COORDINATE SYSTEM. BY          */
/* USING THE COORDINATE TRANSFORMATION, THE CALCULATED RESULTS         */
/* ARE IN THE GLOBAL COORDINATE SYSTEM. DURING THE PROCESS,            */
/* JACOBIN MATRIX IS USED AND THE RESULTS WILL BE USED TO FORM         */
/* THE B-MATRIX. THEN THE STIFFNESS MATRIX CAN BE OBTAINED             */
/* EASELY BY THE GAUSS-LEGENDRE QUADRATURE INTERGRATION SCHEME         */
/*                                                                     */
/*   DrvtGs[gspt,jacob,dinpl,var]                                      */
/*           gspt = values at a Gaussian point                         */
/*          jacob = the Jacobin matrix                                 */
/*          dinpl = list of the derivatives of the interpolation       */
/*                  function                                           */
/*            var = list of independent variable(s)                    */
/*                                                                     */

     DrvtGs[$gspt,$jacob,$dinpl,$var]::Proc[\
            Lcl[%dim,%var,%bmtrx,%jab,%invj,%dinpl,%vdinp],\
/*  Find the dimension of the problem                                  */^\
            %dim:Len[$var],\
/*  Set up the local variable(s)                                       */^\
            %var:Ar[%dim,Make[%var,$i]],\
            If[%dim=0,Set[%var]],\
            %jab:S[$jacob,$var->%var],\
            %dinpl:S[$dinpl,$var->%var],\
/*  Getting the numerical values of the solution at given gauss
    point                                                              */^\
            %var:$gspt,\
            %invj:Minv[%jab],\
            %vdinp:%dinpl,\
            %bmtrx:%invj.%vdinp,\
            Trans[%bmtrx]\
           ]
```

```
/**          P R O G R A M :   E L M K 2 D 1                    **/
/**                                                             **/
/** THIS IS A PROGRAM TO EVALUATE THE LOCAL STIFFNESS MATRIX \
    OF 2-D SIMPLEX FINITE ELEMENT FOR SECOND-ORDER EQUATIONS \
    INVOLVING A SCALAR-VALUED FUNCTION. THE METHOD USED IN   \
    THIS FORMULATION IS BASED ON THE BOOK - A INTRODUCTION   \
    TO THE FINITE ELEMENT METHOD - BY J.N. REDDY.           **/
/** Ke[xcoord,ycoord,avalue]                                **/
/**   xcoord/ycoord = x/y coordinates of the simplex element **/
/**   avalue = a-values of the second-order equation Len-a=5 **/
/**                                                          **/
    <MPintgl
    Ke[$x,$y,$a] :: Proc[\
                        Lcl[%cdmtx,%cfmtx,%area,%phi,%x,%y,\
                            %dphix,%dphiy,%s11,%s12,%s22,%s00,\
                            %intgr,%eq1,%eq2,%eq3,%dx1,%dx2,\
                            %dx3,%dy1,%dy2,%dy3,%t1,%t2],\
/** Linear equations of the element boundaries              **^\
                        %dx1:$x[2]-$x[1],\
                        %dx2:$x[3]-$x[2],\
                        %dx3:$x[1]-$x[3],\
                        %dy1:$y[2]-$y[1],\
                        %dy2:$y[3]-$y[2],\
                        %dy3:$y[1]-$y[3],\
                        %eq1:%dy1/%dx1 %x+$y[1],\
                        %eq2:%dy2/%dx2 %x+$y[2],\
                        %eq3:%dy3/%dx3 %x+$y[3],\
                        If[%dx1:0,%eq1:$x[1]],\
                        If[%dx2:0,%eq2:$x[2]],\
                        If[%dx3:0,%eq3:$x[3]],\
                 Pr[" Linear Equaitons for the Boundaries"],\
                        Pr[" "],\
                        Pr[" "," equation 1 =",%eq1],\
                        Pr[" "],\
                        Pr[" "," equation 2 =",%eq2],\
                        Pr[" "],\
                        Pr[" "," equation 3 =",%eq3],\
/** Form the coordinate matrix                             **^\
                        %cdmtx:{Ar[3,$i/$i],$x,$y},\
/** Determine the coeficients of the shape functions phi   **^\
                        %cfmtx:Dminv[%cdmtx],\
                        %phi:Ex[%cfmtx[1].{1,%x,%y}],\
                        %phi:Cb[%phi,{%x,%y}],\
                        %cdmtx:%phi/%cfmtx[2],\
                        Pr[" "],\
                 Pr[" The Shape Functions are :"],\
                        Pr[" "],\
                        Pr[" "," shape function 1 =",%cdmtx[1]],\
                        Pr[" "],\
                        Pr[" "," shape function 2 =",%cdmtx[2]],\
                        Pr[" "],\
                        Pr[" "," shape function 3 =",%cdmtx[3]],\
/** Calculate the derivatives of shape functions about x & y **^\
                        %dphix:Ex[D[%phi,%x]],\
                        %dphiy:Ex[D[%phi,%y]],\
                        %dphix:Cb[%dphix,$y],\
                        %dphiy:Cb[%dphiy,$y],\
                        Pr[" "],\
                 Pr[" Derivatives of the shape functions"],\
```

```
                        Pr[" "],\
                        Pr[" with respect to x"],\
                        Pr[" "],\
                        Pr["      Df1/Dx =",%dphix[1]/%cfmtx[2]],\
                        Pr[" "],\
                        Pr["      Df2/Dx =",%dphix[2]/%cfmtx[2]],\
                        Pr[" "],\
                        Pr["      Df3/Dx =",%dphix[3]/%cfmtx[2]],\
                        Pr[" "],\
                        Pr[" with respect to y"],\
                        Pr[" "],\
                        Pr["      Df1/Dy =",%dphiy[1]/%cfmtx[2]],\
                        Pr[" "],\
                        Pr["      Df2/Dy =",%dphiy[2]/%cfmtx[2]],\
                        Pr[" "],\
                        Pr["      Df3/Dy =",%dphiy[3]/%cfmtx[2]],\
/** Evaluate the area of the triangular element                  **/\
                        %area:Ex[%cfmtx[2]]/2,\
                        %area:Cb[%area,$y],\
                        Pr[" "],\
                Pr[" The area of the triangular element is :"],\
                        Pr[" "],\
                        Pr["     Area =",%area],\
/** Obtain s11,s12,s22 and s00 by using Eq. 4.34b in [Reddy] **/\
                        Pr[" "],\
                        Pr[" Calculating S11 matrix ...."],\
                        If[Symbp[$a[1]] | $a[1]~=0,Proc[\
                            %cfmtx:%dphix**%dphix,\
                            %s11:Ex[Num[%cfmtx]]/Den[%cfmtx],\
                            %s11:Cb[%s11,$y],\
                            %s11:%s11/(4 %area),\
                           ]],\
                        Pr[" Calculating S12 matrix ...."],\
                        If[Symbp[$a[2]] | $a[2]~=0|$a[3]~=0,Proc[\
                            %cfmtx:%dphix**%dphiy,\
                            %s12:Ex[Num[%cfmtx]]/Den[%cfmtx],\
                            %s12:Cb[%s12,$y],\
                            %s12:%s12/(4 %area),\
                           ]],\
                        Pr[" Calculating S22 matrix ...."],\
                        If[Symbp[$a[4]] | $a[4]~=0,Proc[\
                            %cfmtx:%dphiy**%dphiy,\
                            %s22:Ex[Num[%cfmtx]]/Den[%cfmtx],\
                            %s22:Cb[%s22,$y],\
                            %s22:%s22/(4 %area),\
                           ]],\
                        Pr[" Calculating S00 matrix ...."],\
                        If[Symbp[$a[5]] | $a[5]~=0,Proc[\
                            Lcl[%lm1,%lm2,%eqs],\
                            %intgr:%phi**%phi,\
                            %eqs:MPintgl[%intgr,{%y,%lm1,%lm2},2,0],\
                            %t1:S[%eqs,{%lm1->%eq3,%lm2->%eq2}],\
                            %t2:S[%eqs,{%lm1->%eq1,%lm2->%eq2}],\
                            %s00:MPintgl[%t1,{%x,$x[3],$x[1]},3,0]+\
                                 MPintgl[%t2,{%x,$x[1],$x[2]},3,0],\
                            %s00:%s00/(2 %area)^2,\
                           ]],\
/** Finaly forming the element stiffness matrix                  **/\
                            %t1:$a[1] %s11,\
                            %t1:%t1+$a[3] Trans[%s12],\
```

```
                    %t2:$a[2] %s12,\
                    %t2:%t2+$a[4] %s22,\
                Pr[" "],\
                Pr[" The Local Stiffness Matrix is :"],\
                Pr[" "],\
                %t1+%t2+$a[5] %s00\
                ]
/** ---------------- END OF THE SMP PROGRAM ----------------**/
```

```
/**          P R O G R A M :   E L M K 3 D 1                    **/
/**                                                             **/
/** THIS IS A PROGRAM TO EVALUATE THE LOCAL STIFFNESS MATRIX \
    OF 3-D SIMPLEX FINITE ELEMENT FOR SECOND-ORDER EQUATIONS \
    INVOLVING A SCALAR-VALUED FUNCTION. THE METHOD USED IN   \
    THIS FORMULATION IS BASED ON THE BOOK - A INTRODUCTION   \
    TO THE FINITE ELEMENT METHOD - BY J.N. REDDY.            **/
/** Ke[xcoord,ycoord,zcoord,avalue]                          **/
/**   xcoord/ycoord/zcoord = x/y/z coordinates of the four   **/
/**                          nodes 3-D simplex element       **/
/**   avalue = a-values of the second-order equation Len-a=7 **/
/**                                                          **/
    <MPintgl
    Ke[$x,$y,$z,$a] :: Proc[\
                       Lcl[%cdmtx,%cfmtx,%area,%phi,%x,%y,%z,\
                           %dphix,%dphiy,%s11,%s12,%s22,%s00,\
                           %intgr,%eq1,%eq2,%eq3,%dx1,%dx2,\
                           %dx3,%dy1,%dy2,%dy3,%t1,%t2],\
/** Linear equations of the element boundaries               **/\
                       %dx1:$x[2]-$x[1],\
                       %dx2:$x[3]-$x[2],\
                       %dx3:$x[1]-$x[3],\
                       %dy1:$y[2]-$y[1],\
                       %dy2:$y[3]-$y[2],\
                       %dy3:$y[1]-$y[3],\
                       %eq1:%dy1/%dx1 %x+$y[1],\
                       %eq2:%dy2/%dx2 %x+$y[2],\
                       %eq3:%dy3/%dx3 %x+$y[3],\
                       If[%dx1:0,%eq1:$x[1]],\
                       If[%dx2:0,%eq2:$x[2]],\
                       If[%dx3:0,%eq3:$x[3]],\
                 Pr[" Linear Equaitons for the Boundaries"],\
                       Pr[" "],\
                       Pr[" "," equation 1 =",%eq1],\
                       Pr[" "],\
                       Pr[" "," equation 2 =",%eq2],\
                       Pr[" "],\
                       Pr[" "," equation 3 =",%eq3],\
/** Form the coordinate matrix                               **/\
                       %cdmtx:{Ar[3,$i,/$i],$x,$y},\
/** Determine the coeficients of the shape functions phi     **/\
                       %cfmtx:Dminv[%cdmtx],\
                       Pr[" "],\
                  Pr[" Coefficients of the shape funcitons "],\
                       Pr[" "],\
                       Pr["       ",%cfmtx[1]],\
                       %phi:Ex[%cfmtx[1].{1,%x,%y}],\
                       Pr[" "],\
                       %phi:Cb[%phi,{%x,%y}],\
                  Pr[" The Shape Functions are :"],\
                       Pr[" "],\
                       Pr[" "," shape function 1 =",%phi[1]],\
                       Pr[" "," shape function 2 =",%phi[2]],\
                       Pr[" "," shape function 3 =",%phi[3]],\
/** Calculate the derivatives of shape functions about x & y **/\
                       %dphix:Ex[D[%phi,%x]],\
                       %dphiy:Ex[D[%phi,%y]],\
                       %dphix:Cb[%dphix,$y],\
                       %dphiy:Cb[%dphiy,$y],\
```

```
                    Pr[" "],\
                Pr[" Derivatives of the shape functions"],\
                    Pr[" "],\
                    Pr[" with respect to x"],\
                    Pr["       Df1/Dx =",%dphix[1]],\
                    Pr["       Df2/Dx =",%dphix[2]],\
                    Pr["       Df3/Dx =",%dphix[3]],\
                    Pr[" "],\
                    Pr[" with respect to y"],\
                    Pr["       Df1/Dy =",%dphiy[1]],\
                    Pr["       Df2/Dy =",%dphiy[2]],\
                    Pr["       Df3/Dy =",%dphiy[3]],\
** Evaluate the area of the triangular element                    **/\
                    %area:Ex[%cfmtx[2]]/2,\
                    %area:Cb[%area,$y],\
                    Pr[" "],\
                Pr[" The area of the triangular element is :"],\
                    Pr[" "],\
                    Pr["     Area =",%area],\
** Obtain s11,s12,s22 and s00 by using Eq. 4.34b in [Reddy] **/\
                    Pr[" "],\
                    Pr[" Sij-Matrices are as the follows :"],\
                    If[Symbp[$a[1]] | $a[1]~=0,Proc[\
                       %cfmtx:%dphix**%dphix,\
                       %s11:Ex[Num[%cfmtx]]/Den[%cfmtx],\
                       %s11:Cb[%s11,$y],\
                       %s11:%area %s11,\
                      ],],\
                    If[Symbp[$a[2]] | $a[2]~=0|$a[3]~=0,Proc[\
                       %cfmtx:%dphix**%dphiy,\
                       %s12:Ex[Num[%cfmtx]]/Den[%cfmtx],\
                       %s12:Cb[%s12,$y],\
                       %s12:%area %s12,\
                      ]],\
                    If[Symbp[$a[4]] | $a[4]~=0,Proc[\
                       %cfmtx:%dphiy**%dphiy,\
                       %s22:Ex[Num[%cfmtx]]/Den[%cfmtx],\
                       %s22:Cb[%s22,$y],\
                       %s22:%area %s22,\
                      ]],\
                    If[Symbp[$a[5]] | $a[5]~=0,Proc[\
                       Lcl[%lm1,%lm2,%eqs],\
                       %intgr:%phi**%phi,\
                       %eqs:MPintgl[%intgr,{%y,%lm1,%lm2},2,0],\
                       %t1:S[%eqs,{%lm1->%eq3,%lm2->%eq2}],\
                       %t2:S[%eqs,{%lm1->%eq1,%lm2->%eq2}],\
                       %s00:MPintgl[%t1,{%x,$x[3],$x[1]},3,0]+\
                            MPintgl[%t2,{%x,$x[1],$x[2]},3,0],\
                      ]],\
/** Finaly forming the element stiffness matrix                    **/\
                       %t1:$a[1] %s11,\
                       %t1:%t1+$a[3] Trans[%s12],\
                       %t2:$a[2] %s12,\
                       %t2:%t2+$a[4] %s22,\
                    %t1+%t2+$a[5] %s00\
                    ]
/** ---------------- END OF THE SMP PROGRAM ----------------**/
```

```
/*                P R O G R A M :   J A C O B I N                    */
/*                                                      June 17, 1985 */
/* A SMP PROGRAM TO OBTAIN THE JACOBIN OF THE COORDINATE TRANS-       */
/* FORMATION IN THE FINITE ELEMENT CALCULATIONS DUE TO NON-LINEAR     */
/* INTERPOLATION FUNCTIONS (NON-LINEAR ELEMENT)                       */
/*                                                                    */
/*  Jacobin[coord,interpl,var]                                        */
/*           coord = list of the nodal coordinates on the element     */
/*         interpl = list of the interpolation functions corres-      */
/*                   ponding to each nodal points.                    */
/*             var = list of the independent variables used for       */
/*                   taking the derivatives of the interpolation      */
/*                   function with respect to.                        */

    Jacobn[$coord,$interp,$var]::Proc[\
           Lcl[%dim,%pts,%drvs,%i,%var,%j,%jac,%coord1],\
/* Find out the dimensions of the given problem                       */^\
           %dim:Len[$var],\
           If[%dim=0,%dim:1],\
/* Find out the total number of nodes on each element                 */^\
           %pts:Len[$interp],\
/* Reordering the coordinate sequence ({p1,p2,..} -> {x(,y,z)})       */^\
           If[%dim=1,%coord1:$coord,%coord1:Trans[$coord]],\
/* Calculate the derivatives of interpolation function                */^\
           For[%i:1,%i<=%dim,Inc[%i],Proc[\
               %var:$var[%i],\
               If[Len[$var]=0,%var:$var],\
/* Doing it node by node in order to prevent memery exhausting        */^\
               For[%j:1,%j<=%pts,Inc[%j],Proc[\
                   %drvs[%i,%j]:D[$interp[%j],%var],\
                  ]],\
              ]],\
/* Form the Jacobin matrix                                            */^\
           For[%i:1,%i<=%dim,Inc[%i],Proc[\
               For[%j:1,%j<=%dim,Inc[%j],Proc[\
                   If[Len[$var]=0,%jac:%coord1.%drvs[1],Proc[\
                      %jac[%i,%j]:%coord1[%j].%drvs[%i],\
                     ]],\
                  ]],\
              ]],\
           %jac\
           ]
```

```
/**              P R O G R A M :   S H P F N C                  **/
/**                                                 June 15,1985 **/
/** SMP PROGRAM TO FORM THE INTERPOLATION FUNCTIONS IN          **/
/** NaTURAL (NORMAL) COORDINATE SYSTEM. THE INTERPOLATION       **/
/** FUNCTIONS ARE EVALUATED IN THE ORDER AS USER                **/
/** SPECIFIED.                                                  **/
/**   ShpFnc[var,order]                                         **/
/**           var   = the given variable symbol use wants       **/
/**                   to be used.                               **/
/**           order = the order of the polyminal function       **/
/**                   which user should specify.                **/
/**                                                             **/
   ShpFnc[$var,$order]::Proc[\
                Lcl[%lp,%i,%j,%dx,%xi,%shp,%den,%cnst],\
/** Find the increment of x                                     **/\
                %lp:$order,\
                %dx:2/(%lp-1),\
/** Find the nodal values in neutural coordinate system         **/\
                %xi[1]:-1,\
                For[%i:2,%i<=%lp,Inc[%i],\
                    %xi[%i]:(%xi[%i-1]+%dx)],\
/** Form the interpolation functions                            **/\
                For[%i:1,%i<=%lp,Inc[%i],Proc[\
                    %shp[%i]:1,\
                    %den:1,\
                    %cnst:%xi[%i],\
                    For[%j:1,%j<=%lp,Inc[%j],Proc[\
                        If[%j~=%i,Proc[\
                           %shp[%i]:%shp[%i] ($var-%xi[%j]),\
                           %den:%den (%cnst-%xi[%j]),\
                         ]],\
                    ]],\
                    %shp[%i]: %shp[%i]/%den,\
                    ]],\
                 %shp\
            ]
```

```
/**              P R O G R A M :   S M P L X K                **/
/**                                                 June 15,1985 **/
/** SMP PROGRAM TO GENERATE A LOCAL ELEMENT STIFFNESS          **/
/** MATRIX OF 1-D, 2-D OR 3-D ELASTICITY PROBLEM               **/
/**   SmplxK[coord,dmtrx]                                      **/
/**          coord = an array containing x/y(/z) nodal         **/
/**                  coordinates in the form {x,y(,z)}         **/
/**                  where x/y(/z) has the length of 3         **/
/**                  for 2-D triangular element or 4 for       **/
/**                  3-D tetrahedron element. Sequence of      **/
/**                  {x1,x2,x3(,x4)},.... should be            **/
/**                  defined as RIGHT-HAND rule.               **/
/**          dmtrx = the material constant matrix which        **/
/**                  will be 3X3 for 2-D , 6X6 for 3-D         **/
/**                  and a constant for 1-D                    **/
/**                                                            **/
      <MkArray
      SmplxK[$coord,$dmtrx]::Proc[\
            Lcl[%dim,%len,%shp,%dshp,%bmtrx,%trnsb,%lav,%dn,\
                %cmtrx,%invc,%nods,%x,%y,%z,%rowb,%colb,%i,\
                %elmk],\
/** Find out the dimension of the given problem, i.e. 2-D **/\
/** or 3-D; and x, y, (and z) coordinates.                **/\
/** %len = 1, 2, or 3 .... representing 1-D, 2-D, or 3-D  **/\
            %dim:Dim[$coord],\
            If[Len[%dim]=1,%len:1,%len:%dim[1]],\
            If[%len>1,%nods:%dim[2],%nods:%dim[1]],\
      Pr["Dimension of the Elasticity Problem     =",%len],\
      Pr["Number of Nodes on the Simplex Element  =",%nods],\
     Pr[" "],\
/** Forming the shape functions                           **/\
               %cmtrx[1]:Ar[%nods,1],\
               If[%len=1,%cmtrx[2]:$coord,\
                  For[%i:1,%i<=%len,Inc[%i],Proc[\
                      %cmtrx[%i+1]:$coord[%i],\
                     ]],\
                 ],\
               %invc:Dminv[%cmtrx],\
               %lav:%invc[2]/(%len!),\
               Pr[" "],\
               If[%len=1,Proc[\
                  %dn:Length,\
                  Pr["    Length of the element =",%lav],\
                 ]],\
               If[%len=2,Proc[\
                  %dn:(%len!) xArea,\.
                  Pr["    Area of the element   =",%lav],\
                 ]],\
               If[%len=3,Proc[\
                  %dn:(%len!) xVolume,\
                  Pr["    Volume of the element =",%lav],\
                 ]],\
               %dshp:%invc[1],\
               If[%len=1,%shp:%dshp.{1,$x}],\
               If[%len=2,%shp:%dshp.{1,%x,%y}],\
               If[%len=3,%shp:%dshp.{1,%x,%y,%z}],\
         Pr[" "],\
         Pr[" The shape functions are :"],\
```

```
            Pr[" "],\
            For[%i:1,%i<=%nods,Inc[%i],Proc[\
                Pr["   Shape Func",%i,%shp[%i]/%dn],\
                Pr[" "],\
               ]],\
/** Initializing B-Matrix .....                                    **/\
                %rowb:Len[$dmtrx],\
                If[%rowb=0,%rowb:1],\
                %colb:%nods (%nods-1),\
                %bmtrx:MkArray[0,{%rowb,%colb}],\
                If[%rowb=1,%bmtrx:Flat[%bmtrx]],\
/** Forming the B-Matrix for 1-D element .....                     **/\
             If[%len=1,Proc[\
                %bmtrx[1]:%dshp[1,2],\
                %bmtrx[2]:%dshp[2,2],\
               ],],\
/** Forming the B-Matrix for 2-D or 3-D element .....              **/\
             If[%len>1,Proc[Lcl[%l,%dof],\
                %dof:%nods-1,\
                For[%i:1,%i<=%nods,Inc[%i],Proc[\
                    %l:%dof (%i-1),\
                    For[%j:1,%j<=%len,Inc[%j],Proc[\
                        %bmtrx[%j,%l+%j]:%dshp[%i,%j+1],\
                       ]],\
                    %bmtrx[%len+1,%l+1]:%dshp[%i,3],\
                    %bmtrx[%len+1,%l+2]:%dshp[%i,2],\
                    If[%len=3,Proc[\
                       %bmtrx[5,%l+1]:%dshp[%i,4],\
                       %bmtrx[5,%l+3]:%dshp[%i,2],\
                       %bmtrx[6,%l+2]:%dshp[%i,4],\
                       %bmtrx[6,%l+3]:%dshp[%i,3],\
                      ]],\
                   ]],\
               ]],\
/** Forming the element stiffness matrix .....                     **/\
          If[%len=1,%elmk:$dmtrx*(%bmtrx**%bmtrx)/%lav,\
             Proc[\
                  %trnsb:Trans[%bmtrx],\
                  %elmk:%trnsb.$dmtrx,\
                  %trnsb:%elmk.%bmtrx,\
                  %trnsb /(%invc[2]^2/%lav)\
                 ]\
            ]\
          ]
```

```
/*              P R O G R A M :    T R U S D V                    */
/*                                                    June 4, 1985 */
/*   Program to calculate the element stiffness matrix of         */
/*   a 2-D truss element. It also calculates the derivatives      */
/*   of the stiffness matrix at 0-Nth order under user's          */
/*   specifications. The output form is a list as                 */
/*   { [1]: K, [2]: dK/dxi, ....., [n]: dnK/dxn,                  */
/*             [n+1]: Length, [n+2]: Angle}                       */
/*   Notice: no area of cross section and Young's modulus         */
/*           are required for input since they are the            */
/*           factor of the entire matrices.                       */
/*           It only calculates the right-upper part [k0] of      */
/*           the stiffness matrix since the final stiffness       */
/*           matrix can be obtained by using [k0] as              */
/*                                                                */
/*                 ⎡  [k0] -[k0] ⎤                                */
/*           [K] = ⎢               ⎥                            */
/*                 ⎣_-[k0]  [k0]_⎦                                */
/*                                                                */
/*  TrusDv[$coord,$var,$order]                                    */
/*        $coord ..... List of nodal coordinates                  */
/*                     $coord : {{x1,y1},{x2,y2}}                 */
/*         $var ..... List of variables which will be             */
/*                     taken for the derivatives                  */
/*                     $var : {var1,var2,....}                    */
/*        $order ..... List of orders of the derivatives          */
/*                     $order : {n1,n2,....}                      */
/*                                                                */
    TrusDv[$coord,$var,$order]::Proc[\
          Lcl[%x1,%x2,%y1,%y2,%L,%res,%vlp,%lp,%lvar,%i,%j,\
              %angl,%nord,%lord,%dfunc,%func,%temp,%end,\
              %sv1,%sv2],\
/*   Form the stiffness matrix                                   */\
          %x1:$coord[1,1],%y1:$coord[1,2],\
          %x2:$coord[2,1],%y2:$coord[2,2],\
          %func:{{Cos[%angl]^2,Sin[%angl]Cos[%angl]},\
                 {Sin[%angl]Cos[%angl],Sin[%angl]^2}}/%L,\
    <TrigFn,\
          %temp:1,%res[%temp]:%func,\
/*   Find out number of variables and order of derivatives       */\
          If[$var=Null,Jump[%end]],\
          %vlp:Len[$var],\
          If[%vlp=0,%lp:1,%lp:%vlp],\
          If[$order=Null,$order:1],\
          If[%vlp=0,%nord:$order,\
             %nord:Sum[$order[%i],{%i,1,%lp}],\
           ],\
/*   Find the derivatives of the stiffness matrix                */\
          For[%i:1,%i<=%lp,Inc[%i],Proc[\
              If[%vlp=0,\
                 Proc[%lvar:$var,%lord:$order],\
                 Proc[%lvar:$var[%i],%lord:$order[%i]],\
               ],\
              Pr[" "],\
              Pr[" Derivatives with respect to ",%lvar],\
              Pr[" "],\
              Lcl[%dx,%dy],\
```

```
            For[%j:1,%j<=%lord,Inc[%j],Proc[\
            Pr["  Calculating the ",%j,"-th derivative"],\
            <SeTrig,\
                %L:,%angl:,%dx:,%dy:,\
                %func:%res[%temp],\
                %L:Sqrt[(%x2-%x1)^2+(%y2-%y1)^2],\
                %angl:Atan[(%y2-%y1)/(%x2-%x1)],\
                %func:%func,\
                %dfunc:D[%func,%lvar],\
                %L:,%angl:,$n %x1-$n %x2::-$n %dx,\
                $n %y1-$n %y2::-$n %dy,\
                %dx^2+%dy^2:%L^2,Atan[%dy/%dx]:%angl,\
                1+%dy^2/%dx^2:(%dx^2+%dy^2)/%dx^2,\
                %dfunc:%dfunc,\
                <TrigFn,\
                <TrigFn,\
                %dx:%L Cos[%angl],%dy:%L Sin[%angl],\
                %temp:%temp+1,%res[%temp]:Ex[%dfunc],\
                Atan:,%dx^2+%dy^2:,1+%dy^2/%dx^2:,\
                $n %x1-$n %x2:,$n %y1-$n %y2:,\
               ]],\
          ]],\
        %sv1:Sqrt[(%x2-%x1)^2+(%y2-%y1)^2],\
        %sv2:Atan[(%y2-%y1)/(%x2-%x1)],\
      Lbl[%end],\
        Pr[" "],\
        Pr[" TOTAL NUMBER OF DERIVATIVES = ",%temp-1],\
        %res[%temp+1]:%sv1,%res[%temp+2]:%sv2,\
        %res\
      ]
/*                                                                  */
/*    ------------------- PROGRAM ENDS --------------------         */
```

```
/*              P R O G R A M :    T R U S S K                    */
/*                                                  June 4, 1985 */
/*   Program to calculate element informations of a 2-d          */
/*   truss member in global coordinate system by using the       */
/*   two ends coordinates and given Yung's modulus and area      */
/*   of cross section for dynamic analysis.                      */
/*                                                               */
/*  TrussK[$a,$e,$rho,$x,$y,$geomt,$stiff,$mass]                 */
/*  Filters:                                                     */
/*   ------------------- INPUT FILTERS -------------------       */
/*      $a ----- area of cross section of given truss member     */
/*      $e ----- Yung's modulus of given truss member            */
/*    $rho ----- density of given truss member                   */
/*   $x/$y ----- x/y coordinates at both ends of truss member    */
/*   ------------------- OUTPUT FILTERS -------------------      */
/*  $geomt ----- geomtry values from internal calculations       */
/*               [1] --- angle theta btw. member and x-coord.     */
/*               [2] --- length of given member                  */
/*               [3] --- total mass of given member              */
/*  $stiff ----- calculated member stiffness matrix (local)      */
/*   $mass ----- calculated member mass matrix (local & diag.)   */
/*---------------------------------------------------------------*/
/*                                                               */
    km[$a,$e,$rho,$x,$y,$geomt,$stiff,$mass]::Proc[\
                  Lcl[%theta,%length,%ctheta,%stheta,%cc,%cs,\
                      %ss,%s,%m,%klcl,%k,%x1,%x2,%y1,%y2];\
/*  set all of output values to be Null first                    */^\
                  $geomt: ;$stiff: ;$mass: ;\
/*  calculate the length of given member                         */^\
                  %length:Sqrt[(%x2-%x1)^2+(%y2-%y1)^2];\
/*  calculate the angle btw. given member and x-coord.           */^\
                  %x1:$x[1];%x2:$x[2],\
                  %y1:$y[1];%y2:$y[2],\
                  If[%x2=%x1,%theta:Pi/2,%theta:\
                                  Atan[(%y2-%y1)/(%x2-%x1)],\
                              %theta:Atan[(%y2-%y1)/(%x2-%x1)]];\
                  %ctheta:Cos[%theta];\
                  %stheta:Sin[%theta];\
                  %s:$a $e/%length;\
                  %cc:%ctheta %ctheta;\
                  %ss:%stheta %stheta;\
                  %cs:%stheta %ctheta;\
/*  calculate total mass of given member                         */^\
                  %m:$rho $a %length/2;\
/*  form geomtry output                                          */^\
                  $geomt:{%theta,%length,%m};\
/*  form local stiffness matrix                                  */^\
                  %klcl:{{%cc,%cs},{%cs,%ss}};\
                  For[%k:1,%k<=2,Inc[%k],Proc[\
                      Lcl[%k1,%k2];\
                      %k1:%k;\
                      %k2:2+%k;\
                      $stiff[%k1]:%s Flat[{%klcl[%k],\
                                          -%klcl[%k]}];\
                      $stiff[%k2]:%s Flat[{-%klcl[%k],\
                                           %klcl[%k]}];\
                     ]];\
```

```
/*  form local (diag.) mass matrix                                    *^\
                       $mass:%m [{%cc,%cs,0,0},{%cs,%ss,0,0},\
                                 {0,0,%cc,%cs},{0,0,%cs,%ss}},\
                      ]

/*    ------------------- PROGRAM ENDS -------------------    */
```

```
/*            P R O G R A M :   C O E F M T X                  */
/*                                                 June 2, 1985 */
/*                                                              */
/*                                                              */
/*  Program to obtain the coefficient matrices of a             */
/*  Matrix Polynominal                                          */
/*  PlyCoef[$poly,$ord]                                         */
/*  Filters:                                                    */
/*   $mtx ---- The given matrix in which each element           */
/*             contains a polynominal with order $ord           */
/*    $n  ---- Row dimension of matrix [$mtx]                   */
/*    $m  ---- Column dimension of matrix [$mtx]                */
/*   $ord ---- The highest order of polynomial                  */
/*-------------------------------------------------------------*/
     coefmtx[$mtx,$n,$m,$ord,%coef]:: Proc[\
              Lcl[%i,%j,%k,%c]\
              For[%i:1,%i<=$n,Inc[%i],Proc[\
                  For[%j:1,%j<=$m,Inc[%j],Proc[\
                     For[%k:0,%k<=$ord,Inc[%k],Proc[\
                         If[$mtx[%i,%j,5,%k]:Null,%c=0,\
                            %c:$mtx[%i,%j,5,%k],\
                            %c:$mtx[%i,%j,5,%k]\
                           ];\
                         %coef[%k,%i,%j]:%c;\
                        ]];\
                     ]];\
                 ]];\
               %coef\
              ]
/*  ------------------- END OF FILE -----------------  */
```

```
/*            P R O G R A M :    D A T C O N V          */
/*                                          June 3, 1986 */
/*  A SMP PROGRAM CONVERTING A DATA FROM SMP OUTPUT     */
/*  TO A FORTRAN STANDARD DATA SO THAT IT CAN BE READ   */
/*  DIRECTLY BY FORTRAN PROGRAM                         */
/*   DatConv[$dat,$desm,$form]                          */
/*         $dat = the given set data                    */
/*        $desm = the number of desimal points          */
/*        $form = desired format (F/E/D by FORATRAN)    */
/*                                                      */
    DatConv[$dat,$desm,$form]::Proc[\
         Lcl[%fform,%dform,%eform,%numb,%i,%dat,%sign],\
         %sign[$z]::If[$z>=0,"+","-"];
         %numb:Len[$dat],\
         If[%numb=0,%numb:1],\
         For[%i:1,%i<=%numb,Inc[%i],Proc[\
         If[$form=Symbp[f]|$form=Symbp[F],Jmp[%fform]],\
         If[$form=Symbp[d]|$form=Symbp[D],Jmp[%eform]],\
         If[$form=Symbp[e]|$form=Symbp[E],Jmp[%eform]],\
       Jmp[%last],\
       Lbl[%fform],
         %dat:N[$dat,$desm]
%i,%data,%last,%vabs,%next,%exp,%aexp,\
              %esign,%t,%zero];\
   Lcl[%frac,%vsign,%digit,%m,%n];\
   For[i:1,i<=3,Inc[i],Proc[\
       %t:$v[i];\
       %vabs:Abs[%t];\
       If[%vabs=0,Jmp[%zero],Jmp[%next]];\
     Lbl[%zero];\
       %g[i]:Fmt[," +.00000000D+00"];\
    Jmp[%last];\
    Lbl[%next];\
       %exp:Floor[N[Log[%vabs,10]]]+1;\
       %aexp:Abs[%exp];\
       %esign:ssign[%exp];\
       %frac:%vabs/(10*^%exp);\
       %vsign:ssign[%t];\
       %digit:Floor[%frac*(10*^8)];\
       %m:Floor[%aexp/10];\
       %n:%aexp-10*%m;\
       %g[i]:Fmt[," ",%vsign,".",N[%digit,8],"D",%esign,%m,%n];\
    Lbl[%last];\
    ]],\
    %g\
    ]
```

```
/**              P R O G R A M :  F D C N S T                    **/
/**                                                             **/
/**                                              May 21,1985    **/
/**   A SMP PROGRAM TO FIND THE CONSTANT TERMS OF GIVEN         **/
/**   n-th ORDER POLYNOMINAL                                    **/
/**       Fdcnst[expr,var,order]                                **/
/**         filters are:                                        **/
/**            expr = expression of the given polynaminal       **/
/**             var = variable, order = order of expr           **/
/**                                                             **/

Fdcnst[$intgr,$var,$ord]::\
          Proc[\
               Lcl[%i,%poly,%resl,%coef,%var],\
               %resl:0,\
               %var:$var,\
               %poly:Ex[$intgr],\
               %poly:Cb[%poly,{%var,%var^$i}],\
               For[%i:1,%i<=$ord,Inc[%i],Proc[\
                        %coef:Coef[%var^%i,%poly],\
                        %resl:%resl+%coef %var^%i\
                  ]],\
               %poly-%resl\
              ]
```

```
/**              P R O G R A M :   F N D M A X                    **/
/**                                                  June 2,1985 **/
/** PROGRAM TO FIND THE MAXIMUM ELEMENT (IN MAGNIGUDE) OF A      **/
/** VECTOR OR AN ARRAY (WHICH HAS ARBITRARY DIMENSIONS)          **/
/**   FndMax[array]                                              **/
/**           array = the given array for which the maximum      **/
/**                   element need to be found                   **/
/**                                                              **/
      FndMax[$array]::Proc[\
                          Lcl[%max,%mag,%vec,%len,%i],\
                          %max:0,\
                          %vec:Flat[$array],\
                          %len:Len[%vec],\
                          If[%len=0,%max:$array,Proc[\
                             For[%i:1,%i<=%len,Inc[%i],Proc[\
                                 %mag:Abs[%vec[%i]],\
                                 %max:Max[%max,%mag],\
                                ]],\
                              ],\
                            ],\
                          %max\
                          ]
```

```
/*                  P R O G R A M :    G S Q D R T                    */
/*                                                      June 16, 1985 */
/* A SMP PROGRAM TO EVALUATION INTEGRATIONS BY USING GAUSSIAN         */
/* QUADRATURE POINTS (A NUMERICAL INTEGRATION METHOD)                 */
/*                                                                    */
/*   GsQdrt[gspt,wts,intgr,var]                                       */
/*          gspt = a list of Gauss points                             */
/*           wts = the weights                                        */
/*         intgr = the integrant to be integrated                     */
/*           var = independent variable in the integrant              */
/*                                                                    */

   GsQdrt[$gspt,$wts,$intgr,$var]::Proc[\
          Lcl[%npt,%gs,%wt,%sum,%i,%intgl],\
          %npt:Len[$gspt],\
          %sum:0,\
          For[%i:1,%i<=%npt,Inc[%i],Proc[\
              %gs:,\
              %intgl:S[$intgr,$var->%gs],\
              %gs:$gspt[%i],\
              %wt:$wts[%i],\
              %sum:%sum+%wt %intgl,\
             ]],\
          %sum\
          ]
```

```
/**                P R O G R A M :  M K A R R A Y                **/
/**                                                             **/
/**                                              May 21, 1986   **/
/**    A SMP PROGRAM TO MAKE A ONE OR TWO DIMENSION ARRAY       **/
/**           MkArray[symb,dim]                                 **/
/**                    symb = symbol to be used presenting      **/
/**                           the array element                 **/
/**                     dim = the array's dimension             **/
/**                                                             **/
/**
     MkArray[$a,$n]::Proc[\
                          Lcl[%l,%arry],\
                          %l:Len[$n],\
                          If[$a=0,%arry:Ar[$n,0]],\
                          If[Symbp[$a] & %l=0,\
                             %arry:Ar[$n,Make[$a,$i]]],\
                          If[Symbp[$a] & %l~=0,\
                             %arry:Ar[$n,Make[(Make[$a,$i]),$j]]],\
                          %arry\
                         ]
```

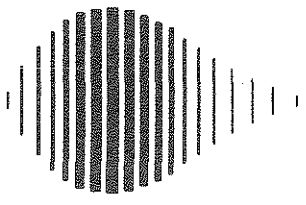
```
/**                    P R O G R A M :  M P I N T G L                  **/
/**                                                                   **/
/**                                                 May 21,1985       **/
/**    A SMP PROGRAM TO CALCULATE THE INTEGRATION OF A                **/
/**    MATRIX IN WHICH EACH ELEMENT IS IN THE n-th ORDER              **/
/**    POLYNOMINAL                                                    **/
/**    MPintgl[expr,{var,lower,upper},order,symp]                     **/
/**    filters are:                                                   **/
/**       expr = expression of the given polynominal                  **/
/**        var = variable, lower/upper = lower/upper limit            **/
/**      order = order of the polynominal                             **/
/**       symp = symmetry indicator (symp=0 -- symmetrical)           **/
/**                                                                   **/
/**
     <Pintgl
     MPintgl[$intgr,$var,$ord,$symp]::\
               Proc[\
                    Lcl[%len1,%len2,%i,%j,%st,%mtrx,%term],\
                    %len1:Len[$intgr],\
                    %len2:Len[$intgr[1]],\
                    For[%i:1,%i<=%len1,Inc[%i],Proc[\
                       If[$symp=0,%st:%i,%st:1],\
                       For[%j:%st,%j<=%len2,Inc[%j],Proc[\
                        %term:$intgr[%i,%j],\
                        %mtrx[%i,%j]:Pintgl[%term,$var,$ord],\
                        If[$symp=0,%mtrx[%j,%i]:%mtrx[%i,%j]],\
                       ]],\
                    ]],\
                    %mtrx\
                    ]
```

```
**                P R O G R A M :  P I N T G L                **/
**                                                            **/
**                                                 May 21,1985 **/
**    A SMP PROGRAM TO CALCULATE THE INTEGRATION OF THE       **/
**    n-th ORDER POLYNOMINAL                                  **/
**    Pintgl[expr,{var,lower,upper},order]                    **/
**    filters are:                                            **/
**       expr = expression of the given polynominal           **/
**        var = variable, lower/upper = lower/upper limit     **/
**      order = order of the polynominal                      **/
**                                                            **/
   <FdCnst
   Pintgl[$intgr,$var,$ord]::\
              Proc[\
                   Lcl[%var,%uplm,%lwlm,%poly,%i,%resl,%tm],\
                   %resl:0,\
                   %var:$var[1],\
                   %lwlm:$var[2],\
                   %uplm:$var[3],\
                   %poly:Ex[$intgr],\
                   %cnst:Fdcnst[%poly,%var,$ord],\
                   %poly:Cb[%poly,{%var,%var^$i}],\
                   For[%i:1,%i<=$ord,Inc[%i],Proc[\
                            Lcl[%coef,%ord],\
                            %coef:Coef[%var^%i,%poly],\
                            %ord:%i+1,\
                            %tm:%uplm^%ord-%lwlm^%ord,\
                            %tm:Ex[%tm]/%ord,\
                            %resl:%resl+ %coef %tm,\
                       ]],\
                    %tm:%uplm-%lwlm,\
                    %cnst:%cnst Ex[%tm],\
                    %cnst+%resl\
                   ]
```

```
/*                P R O G R A M :    T R I G F N                    */
/*                                                    June 4, 1985  */
/*    DEFINE CERTAIN RULES OF THE TRIGONOMITRIC FUNCTIONS.          */
/*                                                                  */

      Cos[$$x]^2::(1+Cos[2 $$x])/2
      Sin[$$x]^2::(1-Cos[2 $$x])/2
      Cos[$$x] Sin[$$x]::Sin[2 $$x]/2
      Sin[$$x1] Sin[$$x2]::-(Cos[$$x1+$$x2]-Cos[$$x1-$$x2])/2
      Cos[$$x1] Cos[$$x2]::(Cos[$$x1+$$x2]+Cos[$$x1-$$x2])/2
      Cos[$$x1] Sin[$$x2]::(Sin[$$x1+$$x2]-Sin[$$x1-$$x2])/2
      Cos[-$$x]::Cos[$$x]
      Sin[-$$x]::-Sin[$$x]
                                                                  */
/*                                                                */
/*        Set Sin[-x] = - Sin[x],  Cos[-x] =   Cos[x],            */
/*            Tan[-x] = - Tan[x], Ctan[-x] = - Ctan[x]            */
/*                                                                */

      Sin[$x_=0 > Nc[$x]] :: -Sin[-$x]
      Sin[$x_= ~ In[$x,Ncfac[$x]]] :: Sin[Ncfac[$x]]
      Cos[$x_=0 > Nc[$x]] :: Cos[-$x]
      Cos[$x_= ~ In[$x,Ncfac[$x]]] :: Cos[Ncfac[$x]]
      Tan[$x_=0 > Nc[$x]] :: -Sin[-$x]
      Tan[$x_= ~ In[$x,Ncfac[$x]]] :: Sin[Ncfac[$x]]
      Ctan[$x_=0 > Nc[$x]] :: -Ctan[-$x]
      Ctan[$x_= ~ In[$x,Ncfac[$x]]] :: Ctan[Ncfac[$x]];
                                                                  */
/*                                                                */
/*    ---------- PROGRAM ENDS ----------                          */
```



National Center for Earthquake Engineering Research
State University of New York at Buffalo