

**PROJECT #2:**  
**Algorithms for Optimization of Constrained Engineering  
Systems – 10 Bar Truss Structural Optimization**

by

Joseph P. Kubitschek  
26 October 1998

**ABSTRACT**

The primary purpose of this project was to become familiar with typical algorithms that are used for constrained optimization of engineering systems. For such case, the standard 10-bar truss problem was studied in the context of minimizing the weight of the structure while maintaining structural adequacy with respect to allowable stress and minimum cross-sectional area criteria. The problem was solved using the exterior penalty method with IMSL routine DUMINF; the interior penalty method with DUMINF; and a direct method, DNCONF, available from the IMSL library. Each penalty method algorithm was compared with the direct method (assumed to be the standard here for efficiency, reliability, and accuracy). The results indicate that both the exterior and interior penalty methods are less efficient based on the total number of function evaluations required for convergence to an *a priori*,  $\text{eps}=1.0\text{E-}5$ . Furthermore, although the interior penalty method guarantees a feasible design, it may not always exhibit adequate convergence characteristics and hence may represent less accuracy than the exterior penalty method. Conversely, the exterior penalty method exhibited superior results as compared with the interior penalty method based on reliability and accuracy. However, this was at the expense of convergence to an infeasible design, a difficulty that requires additional considerations in order for the results to be useful.

## INTRODUCTION

The purpose of this project was to develop an improved understanding of various algorithms used for optimization of constrained engineering systems. In this case, the standard 10-bar truss problem was investigated. The problem was solved using the exterior penalty method, the interior penalty method, and a direct method available from the IMSL library. Each penalty method algorithm was compared with the direct method from the standpoints of accuracy, efficiency, and reliability.

### Background

The 10-bar truss configuration, given as figure 6.4.1 (Haftka and Gurdal<sup>1</sup>), represents a problem conducive to basic constrained optimization. Driver code for the exterior and interior penalty methods, using DUMINF, and the direct method, using DNCONF, was developed for this project consistent with that of figure 6.4.1. That is, the same nodal and elemental numbering scheme was used along with all other data specified. However, the code was written in a general form such that any 2-D truss may be analyzed with similar constraints on minimum cross-sectional area and allowable stress. The user simply needs to provide the input file specifying truss geometry, connectivity, loading, minimum cross-sectional area for each member, allowable stress for each member, and the modulus of elasticity for each member. Should additional constraints need to be satisfied, the user must modify the function subroutines (*fcne.f*, *fcni.f*, and *fcnd.f*) to account for those constraints. The analysis code was written by Assistant Professor Ganesh Subbarayan, Department of Mechanical Engineering, University of Colorado at Boulder (*truss2d.f*, *apldbc.f*, *arinlz.f*, *aslstf.f*, *clstrs.f*, *clvol.f*, *facstf.f*, *slvdsp.f*) and the optimization code (DUMINF and DNCONF) was obtained from the IMSL library for unconstrained minimization using finite difference gradient. The final component of the analysis code, *clstif.f*, was developed for this project to compute the stiffness matrix for each element as required for use with *truss2d.f*.

### Solution Techniques

The first task in completing this project was to develop the code for computing the stiffness matrix for each element to be used with the existing analysis code *truss2d.f*. This was achieved by deriving the stiffness matrix for a generalized truss member using the principle of minimum potential energy. The results yield a 4x4 symmetric matrix that was subsequently implemented in the subroutine *clstif.f*. A simple two-bar plane-truss, given as example 5.7.2 (Haftka and Gurdal<sup>1</sup>), was coded and analyzed using *truss2d.f*. This routine, *tbar.f*, is included as appendix A-1. The results verify the correctness of the stiffness matrix code, *clstif.f*. Having successfully coded the stiffness matrix, the next step was to construct the input data file, *project2.dat*. The analysis code was again tested with the complete input data file and demonstrated the correct format. The input data file itself contains all the necessary information for analysis of the 10-bar truss problem and includes information regarding truss geometry (i.e. the number of nodes and elements), spatial location of each node, member connectivity (i.e. how each of the element are connected via the nodes), boundary conditions (i.e. whether nodes are free or fixed), loading (i.e. how and where loads are applied to the truss), modulus of elasticity, and initial cross-sectional areas of each element. Following successful completion of this test, the driver routines for each optimization method were coded with the following considerations:

#### Exterior Penalty Method:

The exterior penalty method was implemented using the available IMSL routine DUMINF. DUMINF has the primary advantage that the finite difference gradient is automatically computed in the absence of an analytic or exact gradient. This reduces, considerably, the work required of the user. However, this routine was developed for unconstrained optimization or minimization problems. Thus, the 10-bar truss problem must be cast as an unconstrained problem (which is in

fact the concept behind penalty methods). The 10-bar truss problem in standard optimization form is

$$\text{Minimize: } w(\mathbf{a}) = \rho V = \rho \mathbf{a}_i l$$

$$\text{Subject to: } \mathbf{g}_i \geq 0.0.$$

Where,

$\mathbf{a}_i$  = matrix containing cross-sectional areas of each element,

$\rho$  = density of material used (0.1 lbs/in<sup>3</sup> in this case),

$\mathbf{g}_i$  = normalized constraints ( $\mathbf{g}_1 = 1.0 - \sigma/\sigma_{all}$  and  $\mathbf{g}_2 = \mathbf{a}_i/\mathbf{a}_{min} - 1.0$  for stress and area constraints in this case),

$l$  = the length of each member ( $l = 360$ in. and is constant in this case).

Thus, the problem becomes one of minimizing the volume (which is simply a linear function of cross-sectional area) and may be written in unconstrained form as

$$\text{Minimize: } \phi(\mathbf{a}, r) = f(\mathbf{a}_i) + r \sum \langle -\mathbf{g}_i \rangle^2.$$

Where,

$\phi(\mathbf{a}, r)$  = unconstrained function of area,

$f(\mathbf{a}_i)$  = scaled volume.

$r$  = penalty parameter.

This approach imposes a penalty each time a constraint is violated and hence asymptotically approaches the minimum value as  $r \rightarrow \infty$ . However, it is important to realize that this method operates in the infeasible region. That is, as  $r \rightarrow \infty$ , the solution is approached from the infeasible region and thus provides infeasible solutions upon convergence to a reasonable  $\epsilon$ . The only means of obtaining an exact solution is to truly reach  $\infty$ . But, for practical purposes, a close approximation is achieved by successive iteration of  $r$  to some large value (e.g.  $1.0 \times 10^5$  in this case). Thus, the driver for DUMINF, *project2\_ef*, and the function subroutine, *fcne.f*, were written to implement this exterior penalty method and are included in appendix A-2.

#### Interior Penalty Method:

Using the same original formulation, the 10-bar truss problem may be written in unconstrained form using the interior penalty method as

$$\text{Minimize: } \phi(\mathbf{a}, r) = f(\mathbf{a}_i) + r \sum (1/\mathbf{g}_i).$$

Where

$\phi(\mathbf{a}, r)$  = unconstrained objective function of area,

$f(\mathbf{a}_i)$  = scaled volume.

$r$  = penalty parameter.

In this case the value of  $r$  is successively decreased such that  $r \rightarrow 0$ . That is the penalty term becomes smaller and the solution is approached from the feasible region. Although this seems a good approach since all solutions short of the exact solution represent feasible designs, this approach is less than desirable for two reasons: 1.) The optimization has the potential of stepping into the infeasible region; 2.) A feasible starting point must be selected to produce quality results. The first problem is easily overcome by imposing a large penalty for stepping into the infeasible

region and was implemented in the function subroutine *fci.f* for this project. However, the second problem is much more difficult to handle and aside from heuristic methods can only be tackled by trial and error, making the efficiency and reliability of this interior penalty method less than desirable in this case. Both the driver routine and the function subroutine are included as appendix A-3 along with the associated results. Finally, the potential for the values of the unconstrained objective to become negative at intermediate steps can be handled by transforming the areas passed to *truss2d.f*. This transformation is required since negative values of area produce a stiffness matrix that is no longer positive definite. One such transformation that guarantees positive values are passed to the analysis code is:

$$\mathbf{x}_i = \mathbf{a}_i^2.$$

#### Direct Method:

The direct method used for this project was the routine DNCONF available from the IMSL library. This routine solves a general non-linear programming problem using successive quadratic programming. Although this method is so called direct it requires the user to develop both the driver routine and the function subroutine that includes the constraints. The problem was formulated in the same manner as for the exterior and interior penalty methods and the analysis code was implemented in the function subroutine *fcnd.f* (as was required for both the exterior and interior penalty method function-subroutines). The function subroutines in each case comprise the interface between the analysis code and the optimization code (i.e. the function subroutines call *truss2d.f*) as well as implementation of the constraints. The results using DNCONF demonstrated it to be superior and hence this algorithm represents the standard for comparison of the exterior and interior penalty methods. Both the driver routine and function subroutine for DNCONF is included as appendix A-4.

## RESULTS AND DISCUSSION

The results of this project provide a means for comparison of each algorithm from the standpoints of efficiency, reliability, and accuracy with the appropriate considerations. Those features are the focus of the following discussion. Tables 1-3 represent the results of each optimization algorithm.

**Table 1.** – Exterior penalty method (w/DUMINF) results.

#### **10-bar Truss Optimization Results (Exterior):**

**Number of function evaluations = 5238**

**Number of iterations = 5**

<b>Volume (in<sup>3</sup>) = 14889.49</b>		
<b>Weight (lbs) = 1488.95</b>		
<b>Node Displacements:</b>		
<b>Node</b>	<b>u(in)</b>	<b>v(in)</b>
1	.60	-2.40
2	-.60	-2.70
3	.30	-.90
4	-.30	-.90
5	.00	.00
6	.00	.00
<b>Areas and Stresses:</b>		
<b>Element</b>	<b>area(in<sup>2</sup>)</b>	<b>stress(psi)</b>
1	7.99	25000.16
2	.01	25000.04

3	8.01	-25000.24
4	3.99	-25000.06
5	.01	.45
6	.01	25000.04
7	5.67	25000.29
8	5.64	-25000.11
9	3.76	37500.33
10	.01	-25000.09

**Table 2.** – Interior penalty method (w/DUMINF) results.

**10-bar Truss Optimization Results (Interior):**

Number of function evaluations = 2273

Number of iterations = 11

<b>Volume (in<sup>3</sup>) = 14717.33</b>		
<b>Weight (lbs) = 1471.73</b>		
<b>Node Displacements:</b>		
<b>Node</b>	<b>u(in)</b>	<b>v(in)</b>
1	.12	-.60
2	-.14	-.63
3	.09	-.26
4	-.09	-.26
5	.00	.00
6	.00	.00
<b>Areas and Stresses:</b>		
<b>element</b>	<b>area(in<sup>2</sup>)</b>	<b>stress(psi)</b>
1	5.19	7098.48
2	1.84	2500.72
3	5.42	-7098.02
4	4.57	-4374.36
5	1.42	.00
6	1.84	2500.72
7	4.65	7089.47
8	4.27	-7089.23
9	4.46	6509.43
10	1.39	-6195.89

**Table 3.** – Direct method (DNCONF) results.

**10-bar Truss Optimization Results (DNCONF):**

Number of function evaluations = 166

<b>Volume (in<sup>3</sup>) = 14976.00</b>		
<b>Weight (lbs) = 1497.60</b>		
<b>Node Displacements:</b>		
<b>node</b>	<b>u(in)</b>	<b>v(in)</b>
1	.60	-2.40
2	-.60	-2.70
3	.30	-.90
4	-.30	-.90
5	.00	.00
6	.00	.00
<b>Areas and Stresses:</b>		
<b>element</b>	<b>area(in<sup>2</sup>)</b>	<b>stress(psi)</b>
1	7.90	25000.00

2	.10	25000.00
3	8.10	-25000.00
4	3.90	-25000.00
5	.10	.00
6	.10	25000.00
7	5.80	25000.00
8	5.52	-25000.00
9	3.68	37500.00
10	.10	-25000.00

### Efficiency

The primary difficulties in efficiently implementing both penalty methods consist of the selection of an adequate starting point and the successive change of the penalty parameter,  $r$ , until convergence is achieved. The first difficulty may be handled either by heuristic methods or simply by trial and error. Trial and error was used in this project. For both methods, the starting point was to set all initial cross-sectional areas at values of  $5.0 \text{ in}^2$ . The second difficulty was handled by choosing a good updated starting value at the beginning of each iteration (i.e. for each successive value of the penalty parameter). This was achieved by incorporating the approach developed by Fiacco and McCormick<sup>6</sup>. Realizing that the optimal solution is approached asymptotically with both methods, the solutions from previous iterations may be used to predict the optimum for the next iteration and hence skip unnecessary function and gradient evaluations. The asymptotic form of the optimum given by Haftka and Gurdal<sup>1</sup> is

$$\mathbf{x}^*(r) = \mathbf{a} + \mathbf{b}/r \text{ as } r \rightarrow 0.$$

Thus, having the optimum for the previous two values of  $r$ , one can estimate  $\mathbf{a}$  and  $\mathbf{b}$  and predict  $\mathbf{x}^*(r)$  for the next iteration as

$$\mathbf{a} = [c\mathbf{x}^*(r^{k-1}) - \mathbf{x}^*(r^k)]/(c - 1.0)$$

$$\mathbf{b} = [\mathbf{x}^*(r^{k-1}) - \mathbf{a}] r^{k-1}$$

$$c = r^{k-1}/r^k.$$

where,  $r$  is the penalty parameter and  $k$  is the iteration designation. A similar form for the interior penalty method is given as

$$\mathbf{a} = [c^{1/2} \mathbf{x}^*(r^{k-1}) - \mathbf{x}^*(r^k)]/(c^{1/2} - 1.0)$$

$$\mathbf{b} = [\mathbf{x}^*(r^{k-1}) - \mathbf{a}] (r^{k-1})^{1/2}$$

$$c = r^k / r^{k-1}.$$

Again, the primary advantage of implementing this method is improved efficiency. Furthermore, this approach eliminates the need to determine a suitable starting value and multiplication factor for the penalty parameter by trial and error (a significant benefit since this requires considerable effort). Both of these approaches were implemented for this project with improved efficiency results. Thus the initial  $r$ -value for the exterior penalty method was set at 1.0 with a multiplication factor of 10.0 following each iteration. Alternatively, the initial  $r$ -value for the interior penalty method was set at 1000 with a multiplication factor of 0.1 following each

iteration. Comparison of the results for each algorithm indicates that the direct method, DNCONF produces the best efficiency followed by the interior penalty method and then the exterior penalty method. The basis for comparison of efficiency is the number of function evaluations required for convergence to a solution. Table 4 provides the efficiency ranking and required number of function evaluations for each algorithm.

**Table 4.** – Efficiency ranking and number of function evaluations.

Algorithm	No. of function evaluations	No. of iterations
DNCONF	166	****
Interior Penalty w/DUMINF	2273	11
Exterior Penalty w/DUMINF	5238	5

It should be noted that the above penalty method results represent markedly improved results over those obtained without implementation of the asymptotic form for prediction of subsequent optimums at each iteration.

#### Reliability

Reliability can be described as the ability of an algorithm to converge regardless of the computational cost. Various convergence criteria exist for implementation of the exterior and interior penalty methods. The first is an extension of the asymptotic approach to prediction of successive starting values at the beginning of each optimization iteration and is given by Haftka and Gurdal<sup>1</sup> as

$$\varepsilon \geq \| \mathbf{x}^* - \mathbf{a} \|.$$

Where,  $\varepsilon$  is the *a priori* specified convergence parameter,  $\mathbf{x}^*$  is the optimum of the present iteration, and  $\mathbf{a}$  is the previously described prediction parameter that makes use of the previous two optimal values at the previous two values of the penalty parameter,  $r$ . A second convergence criterion is also given by Haftka and Gurdal<sup>1</sup> as

$$\varepsilon \geq |(\phi - f)/f|.$$

Where,  $\phi$  is the unconstrained objective function and  $f$  is the constrained objective function. These values are evaluated following each iteration and the relative error is compared with the convergence parameter  $\varepsilon$ . The final convergence criterion is based on the change in the constrained objective function from iteration to iteration. This may be written mathematically as

$$\varepsilon \geq | [f^*(r^k) - f^*(r^{k-1})] / f^*(r^k) |.$$

Any of these criteria may be used. However, the first criterion requires implementation of the asymptotic form for prediction of successive optimal values at successive values of the penalty parameter. The convergence criteria implemented for this project was the change criterion with a value of  $\varepsilon = 0.0001$ .

Although the interior penalty method algorithm resulted in fewer function evaluations than the exterior penalty method algorithm, this result is somewhat misleading since the interior penalty method algorithm did not converge. This being the primary criteria for comparison of reliability since a reliable algorithm should converge regardless of the number of function evaluations. Thus, in general, comparison of efficiency should always include consideration of reliability. In other words, although an algorithm may appear to represent greater efficiency by virtue of the

number of function evaluations, it may be totally useless if it does not converge to a reasonable approximation of the optimum. Therefore, sound judgement in evaluating an algorithm from the standpoint of efficiency is required because of the apparent trade-off between efficiency and reliability in cases such as this.

Additional termination criteria included minimum and maximum values of  $r$  for interior and exterior penalty methods respectively and a maximum number of iterations assuming convergence could not be achieved. Finally, it should be noted that DUMINF has certain internal convergence criteria based on the intrinsically computed finite difference gradient used to determine the optimum values of the objective for each successive iteration.

Accuracy

The final comparison feature of this project was accuracy. Accuracy is a tricky topic in algorithm comparison since it depends strongly in convergence characteristics. However, this characteristic does provide some information regarding algorithm quality in the case of the 10-bar truss problem. DNCONF was used as the standard for comparison and illustrates the exterior penalty method algorithm to have superior accuracy in comparison with the interior penalty method latter algorithm did not converge. Again, sound judgement must be used since the exterior penalty method solution represents an infeasible design. However, it is certainly closer to the true optimum than the results of the interior penalty method and would likely represent a feasible solution with the application of adequate safety factors. Table 5 represents the relative accuracy results for each algorithm as determined from

$$Relative\ Error = (V^*_{DNCONF} - V^*_{penalty\ method})/V^*_{DNCONF}$$

**Table 5.** – Relative error of each penalty method algorithm.

<b>Algorithm</b>	<b>V*</b>	<b>Relative Error</b>
<b>DNCONF</b>	<b>14,976.00</b>	
Exterior Penalty w/DUMINF	14,889.49	-0.0058
Interior Penalty w/DUMINF	14717.33	-0.0173

The relative inaccuracy of the interior penalty method algorithm in this case can easily be seen. Although the exterior penalty method represents an infeasible design it is relatively accurate to well within 1.0% of the optimum obtained using DNCONF. In contrast, the interior penalty method has a relative error of almost 2.0%.

## CONCLUSIONS

- The direct method, DNCONF represents the most efficient routine by virtue of the number of function evaluations required for convergence as demonstrated by the 10-bar truss optimization results.
- In the absence of the user knowing the exact or analytical gradient, the IMSL routine DUMINF is preferred over DUMING because it represents a good optimization routine that requires less effort for implementation since the finite difference gradient is automatically computed by the routine.
- The relative efficiency of each algorithm may be evaluated using the number of function evaluations required to converge to a solution. In this respect, the direct method DNCONF is the most efficient followed by the interior penalty method and finally the exterior penalty method.
- Implementation of the asymptotic prediction approach to selection of successive starting values for successive iterations provided improved efficiency in both cases of the exterior and interior penalty methods and considerably reduced the effort required to determine adequate starting and successive r-values for each method.
- Although the interior penalty method appears to represent a more efficient algorithm and provides feasible designs at all stopping points, convergence may not always be achievable and hence reliability and accuracy are less than desirable in cases such as this. Conversely, the exterior penalty method appears to have superior reliability. However, this is at the expense of having an infeasible solution upon convergence.
- For the purposes of evaluating the quality of optimization algorithms (i.e. which algorithm is best for the problem at hand), the user must use sound engineering judgement in comparison of algorithms from the standpoint of efficiency, reliability, and accuracy. Furthermore, the results, although accurate, may not always produce a feasible design and hence requires additional consideration to obtain useful results. In the case of the exterior penalty method, selection of the appropriate safety factors is a possible means of utilizing the results.

## REFERENCES

1. Haftka, R.T., and Z. Gurdal, Elements of Structural Optimization, 3<sup>rd</sup> Ed., Kluwer Academic Publishers, 1992.
2. Chapra, S.C., and R.P. Canale, Introduction to Computing for Engineers, McGraw-Hill Book Co., 1986.
3. Press, Teukolsky, Vetterling, and Flannery, Numerical Recipes, Cambridge University Press, 1992.
4. Koffman, E.B., and Friedman, F.L., Problem Solving and Structures Programming in FORTRAN 77, 3<sup>rd</sup> edition, Addison-Wesley Publishing Co., Inc., 1987.
5. Loukides, M., UNIX for FORTRAN Programmers, O'Reily and Associates, Inc., 1990.
6. Fiacco, V., and McCormick, G.P., Non-Linear Programming: Sequential Unconstrained Minimization Techniques, John Wiley, 1968.

## APPENDIX A – FORTRAN CODE

- 1.) **2-Bar Truss Test Problem:** Verifies correctness of stiffness matrix routine, *clstif*, and performance of finite element analysis code, *truss2d.f*. Vertical load  $P = -100$  lbs, element areas  $A_{el} = 0.25$  in<sup>2</sup>.

```
program tbar
parameter (maxnnod=200,maxnel=100)
integer nnod,nel,ldstif
integer conn(2,maxnel),bc(2,maxnnod)
real*8 x(2,maxnnod),f(2,maxnnod),u(2,maxnnod)
real*8 e(maxnel),a(maxnel),sig(maxnel),vol
C
data nnod,nel /3,2/
data a(1),a(2),e(1),e(2) /0.25,0.25,30.0e6,30.0e6/
data f(1,1),f(2,1),f(1,2),f(2,2),f(1,3),f(2,3) /0.0,0.0,0.0,
$-100.0,0.0,0.0/
data x(1,1),x(2,1),x(1,2),x(2,2),x(1,3),x(2,3) /0.0,0.0,1.0,
$1.0,2.0,0.0/
data bc(1,1),bc(2,1),bc(1,2),bc(2,2),bc(1,3),bc(2,3) /1,1,0,0,1,
$1/
data conn(1,1),conn(2,1),conn(1,2),conn(2,2) /1,2,2,3/
C
CALL truss2d(nnod,nel,e,a,conn,x,bc,f,u,sig,vol)
C
open(unit=45,file='tbar.out',status='unknown')
write(45,*)'Results of FE code for 2-bar truss:'
write(45,*)'Node displacements:'
write(45,*)'   X       Y'
do 55 i=1,nnod
  write(45,50)u(1,i),u(2,i)
50 format(5x,f10.8,5x,f10.8)
55 continue
write(45,*)'Element areas and stresses:'
write(45,*)'   area   sigma'
do 65 i=1,nel
  write(45,60)a(i),sig(i)
60 format(5x,f10.2,5x,f10.2)
65 continue
write(45,*)'Total volume:'
write(45,70)vol
70 format(5x,f10.2)
C
end

subroutine clstif(nnod,nel,iel,e,a,conn,x,k)
C
real*8 delx,dely,length,e(nel),a(nel),x(2,nnod),stif,k(4,4)
integer n1,n2,nnod,nel,iel,conn(2,nel)
C
n1=conn(1,iel)
n2=conn(2,iel)
C Compute length of element
delx=x(1,n2)-x(1,n1)
dely=x(2,n2)-x(2,n1)
length=sqrt(delx**2+dely**2)
```

```

C Compute stiffness matrix to be returned
stif=a(iel)*e(iel)/length
k(1,1)=stif*(delx/length)**2
k(1,2)=stif*(delx/length)*(dely/length)
k(1,3)=-stif*(delx/length)**2
k(1,4)=-stif*(delx/length)*(dely/length)
k(2,2)=stif*(dely/length)**2
k(2,3)=-stif*(delx/length)*(dely/length)
k(2,4)=-stif*(dely/length)**2
k(3,3)=stif*(delx/length)**2
k(3,4)=stif*(delx/length)*(dely/length)
k(4,4)=stif*(dely/length)**2
k(2,1)=k(1,2)
k(3,1)=k(1,3)
k(3,2)=k(2,3)
k(4,1)=k(1,4)
k(4,2)=k(2,4)
k(4,3)=k(3,4)

```

```

C
end

```

\*\*\*\*\*

**Output:**

Case 1 – Vertical load = -100 lbs, element areas = 0.25 in<sup>2</sup>:

Results of FE code for 2-bar truss:

Node displacements:

X	Y
.00000000	.00000000
.00000000	-.00001886
.00000000	.00000000

Element areas and stresses:

area	sigma
.25	-282.84
.25	-282.84

Total volume:

.71

2.) **10-Bar Truss Problem (EXTERIOR Penalty method):** Optimization of 10-bar truss problem using exterior penalty method and IMSL routine DUMING. The driver code *project2\_e.f* and the subroutines *fcne.f* and *grade.f* were written for this optimization code.

**Input data file:** fn = *project2.dat*

```

6 10
720.0 360.0 0.0 0.0 0 0
720.0 0.0 0.0 -100000.0 0 0
360.0 360.0 0.0 0.0 0 0
360.0 0.0 0.0 -100000.0 0 0
0.0 360.0 0.0 0.0 1 1
0.0 0.0 0.0 0.0 1 1
5.0 30000000.0 5 3 25000.0 0.1
5.0 30000000.0 3 1 25000.0 0.1
5.0 30000000.0 6 4 25000.0 0.1
5.0 30000000.0 4 2 25000.0 0.1
5.0 30000000.0 3 4 25000.0 0.1

```

```
5.0 30000000.0 2 1 25000.0 0.1
5.0 30000000.0 5 4 25000.0 0.1
5.0 30000000.0 6 3 25000.0 0.1
5.0 30000000.0 3 2 75000.0 0.1
5.0 30000000.0 4 1 25000.0 0.1
```

program **project2\_e**

```
C
parameter(maxnnod=20000,maxnel=20000,eps=1.0e-4)
real*8 rho
integer fcount,nnod,nel,conn(2,maxnnod),bc(2,maxnnod)
integer iparam(7),count
real*8 x(2,maxnnod),f(2,maxnnod),u(2,maxnnod)
real*8 xx(maxnel),yy(maxnel)
real*8 e(maxnel),ainit(maxnel),a(maxnel),opta(maxnel),
$amin(maxnel),ascale(maxnel)
real*8 sig(maxnel),sigall(maxnel)
real*8 c,fact,vol,optv,pvol,weight,fscale,rmax,error
real*8 rparam(7)
external fcne,truss2d,apldbc,arinlz,aslstf,clstrs,clstif,clvol,
$facstf,slvdsp,DU4INF,DUMINF
common rho,fcount
C
open(unit=1,file='project2.dat',status='unknown')
C
read(1,*)nnod,nel
do 5 i=1,nnod
  read(1,*)x(1,i),x(2,i),f(1,i),f(2,i),bc(1,i),bc(2,i)
5 continue
C
do 10 i=1,nel
  read(1,*)ainit(i),e(i),conn(1,i),conn(2,i),sigall(i),amin(i)
10 continue
close(1)
C
do 15 i=1,nel
  ascale(i)=5.0
15 continue
C
rmax=1.0e6
error=1.0
count=0
fcount=0
fscale=1.0
rho=1.0
fact=10.0
C
do while((rho.lt.rmax).and.(error.gt.eps).and.(count.lt.100))
  count=count+1
  write(*,*)count
C
CALL DU4INF(iparam,rparam)
iparam(3)=10*iparam(3)
CALL DUMINF(fcne,nel,ainit,ascale,fscale,iparam,rparam,opta,optv)
C
c=1.0/fact
```

```

do 16 i=1,nel
  xx(i)=(c*ainit(i)-opta(i))/(c-1.0)
  yy(i)=(ainit(i)-xx(i))*rho
16 continue
C
do 17 i=1,nel
  opta(i)=opta(i)**2
17 continue
C
pvol=vol
CALL truss2d(nnod,nel,e,opta,conn,x,bc,f,u,sig,vol)
C
error=DABS((vol-pvol)/vol)
rho=rho*fact
do 18 i=1,nel
  ainit(i)=xx(i)+yy(i)/rho
18 continue
C
do 19 i=1,nel
  ainit(i)=opta(i)
19 continue
end do
C
weight=vol*0.1
C
open(unit=45,file='project2_e.out',status='unknown')
write(45,*)'10-bar Truss Optimization Results (Exterior):'
write(45,(' Number of function evaluations = ', i6))fcount
write(45,(' Number of iterations = ', i6))count
write(45,*)'Node Displacements:'
write(45,*)' node      u(in)      v(in)'
do 30 i=1,nnod
  write(45,26)i,u(1,i),u(2,i)
26  format(5x,i3,5x,f10.2,5x,f10.2)
30 continue
write(45,*)'Areas and Stresses:'
write(45,*)'element  area(in^2)      stress(psi)'
do 35 i=1,nel
  write(45,31)i,opta(i),sig(i)
31  format(5x,i3,5x,f10.2,5x,f10.2)
35 continue
write(45,(' Volume(in^3) = ',f10.2))vol
write(45,(' Error = ',f10.9))error
write(45,(' Weight(lbs) = ',f10.2))weight
end

subroutine fcne(nel,anew,func)
C
parameter(maxnnod=20000,maxnel=20000)
real*8 rho
integer fcount,nnod,nel,inel,conn(2,maxnnod),bc(2,maxnnod)
integer iparam(7),count
real*8 x(2,maxnnod),f(2,maxnnod),u(2,maxnnod)
real*8 e(maxnel),ainit(maxnel),a(maxnel),opta(maxnel),
$amin(maxnel),anew(maxnel),ascale(maxnel)

```

```

real*8 sig(maxnel),sigall(maxnel)
real*8 vol,c,ga,gs,func
external truss2d,apldbc,arinlz,aslstf,clstrs,clstif,clvol,
$facstf,slvdsp
common rho,fcount
C
fcount=fcount+1
c=0.0
write(*,*)fcount
write(*,*)rho
C
open(unit=1,file='project2.dat',status='unknown')
read(1,*)nnod,inel
do 5 i=1,nnod
read(1,*)x(1,i),x(2,i),f(1,i),f(2,i),bc(1,i),bc(2,i)
5 continue
C
do 10 i=1,inel
read(1,*)ainit(i),e(i),conn(1,i),conn(2,i),sigall(i),amin(i)
10 continue
close(1)
C
do 15 i=1,nel
a(i)=anew(i)**2
15 continue
C
CALL truss2d(nnod,nel,e,a,conn,x,bc,f,u,sig,vol)
C
do 20 i=1,nel
ga=(anew(i)/amin(i))-1.0
gs=1.0-DABS(sig(i)/sigall(i))
if (ga.gt.0.0) then
ga=0.0
else
ga=rho*(ga)**2
end if
if (gs.gt.0.0) then
gs=0.0
else
gs=rho*(gs)**2
end if
c=c+ga+gs
20 continue
func=c+(vol/18000)
write(*,*)func
return
end

```

**makefile:**

```

OBJ = project2_e.o fcne.o arinlz.o clstrs.o aslstf.o clvol.o truss2d.o apldbc.o
clstif.o facstf.o slvdsp.o $(LINK_FNL_STATIC)

```

```

truss2d: $(OBJ)
f77 -o pr2e $(OBJ)

```

```

clean: $(OBJ)

```

rm -f \$(OBJ)

\*\*\*\*\*

**OUTPUT:** fn = *project2\_e.out*

10-bar Truss Optimization Results (Exterior):

Number of function evaluations = 5238

Number of iterations = 5

Node Displacements:

node	u(in)	v(in)
1	.60	-2.40
2	-.60	-2.70
3	.30	-.90
4	-.30	-.90
5	.00	.00
6	.00	.00

Areas and Stresses:

element	area(in <sup>2</sup> )	stress(psi)
1	7.99	25000.16
2	.01	25000.04
3	8.01	-25000.24
4	3.99	-25000.06
5	.01	.45
6	.01	25000.04
7	5.67	25000.29
8	5.64	-25000.11
9	3.76	37500.33
10	.01	-25000.09

Volume(in<sup>3</sup>) = 14889.49

Weight(lbs) = 1488.95

- 3.) **10-Bar Truss Problem (INTERIOR penalty method):** Optimization of 10-bar truss problem using interior penalty method and IMSL routine DUMING. The driver code *project2\_i.f* and the subroutines *fci.f* and *gradi.f* were written for this optimization code.

**Input data file:** fn = *project2.dat*

```
6 10
720.0 360.0 0.0 0.0 0 0
720.0 0.0 0.0 -100000.0 0 0
360.0 360.0 0.0 0.0 0 0
360.0 0.0 0.0 -100000.0 0 0
0.0 360.0 0.0 0.0 1 1
0.0 0.0 0.0 0.0 1 1
5.0 30000000.0 5 3 25000.0 0.1
5.0 30000000.0 3 1 25000.0 0.1
5.0 30000000.0 6 4 25000.0 0.1
5.0 30000000.0 4 2 25000.0 0.1
5.0 30000000.0 3 4 25000.0 0.1
5.0 30000000.0 2 1 25000.0 0.1
5.0 30000000.0 5 4 25000.0 0.1
5.0 30000000.0 6 3 25000.0 0.1
5.0 30000000.0 4 1 25000.0 0.1
```

```

program project2_i
C
parameter(maxnnod=20000,maxnel=20000,eps=1.0e-5)
real*8 rho
integer fcount,nnod,nel,conn(2,maxnnod),bc(2,maxnnod)
integer iparam(7),count
real*8 x(2,maxnnod),f(2,maxnnod),u(2,maxnnod)
real*8 xx(maxnel),yy(maxnel)
real*8 e(maxnel),ainit(maxnel),a(maxnel),opta(maxnel),
$amin(maxnel),ascale(maxnel)
real*8 sig(maxnel),sigall(maxnel)
real*8 c,fact,pvol,vol,optv,weight,fscale,rmin,norm,error
real*8 rparam(7)
external fcni,truss2d,apldbc,arinlz,aslstf,clstrs,clstif,clvol,
$facstf,slvdsp,DU4INF,DUMINF
common rho,fcount
C
open(unit=1,file='project2.dat',status='unknown')
C
read(1,*)nnod,nel
do 5 i=1,nnod
  read(1,*)x(1,i),x(2,i),f(1,i),f(2,i),bc(1,i),bc(2,i)
5 continue
C
do 10 i=1,nel
  read(1,*)ainit(i),e(i),conn(1,i),conn(2,i),sigall(i),amin(i)
10 continue
close(1)
C
do 15 i=1,nel
  ascale(i)=5.0
15 continue
C
rmin=1.0e-6
error=1.0
norm=0.0
count=0
fcount=0
fscale=1.0
rho=1000
fact=0.1
C
do while((rho.gt.rmin).and.(error.gt.eps).and.(count.lt.100))
  count=count+1
  write(*,*)count
C
CALL DU4INF(iparam,rparam)
iparam(3)=10*iparam(3)
CALL DUMINF(fcni,nel,ainit,ascale,fscale,iparam,rparam,opta,optv)
C
c=DSQRT(fact)
do 16 i=1,nel
  xx(i)=(c*ainit(i)-opta(i))/(c-1.0)
  yy(i)=(ainit(i)-xx(i))/DSQRT(rho)
16 continue

```

```

C   do 17 i=1,nel
      norm=norm+(opta(i)-xx(i))**2
17  continue
C
      do 18 i=1,nel
        opta(i)=opta(i)**2
18  continue
C
      CALL truss2d(nnod,nel,e,opta,conn,x,bc,f,u,sig,vol)
C
      rho=rho*fact
      do 19 i=1,nel
        ainit(i)=xx(i)+yy(i)*DSQRT(rho)
19  continue
      error=DSQRT(norm)
C
      end do
C
      weight=vol*0.1
C
      open(unit=45,file='project2_i.out',status='unknown')
      write(45,*)'10-bar Truss Optimization Results (Exterior):'
      write(45,(' Number of function evaluations = ', i6))fcount
      write(45,(' Number of iterations = ', i6))count
      write(45,*)'Node Displacements:'
      write(45,*)' node          u(in)          v(in)'
      do 30 i=1,nnod
        write(45,26)i,u(1,i),u(2,i)
26  format(5x,i3,5x,f10.2,5x,f10.2)
30  continue
      write(45,*)'Areas and Stresses:'
      write(45,*)'element      area(in^2)      stress(psi)'
      do 35 i=1,nel
        write(45,31)i,opta(i),sig(i)
31  format(5x,i3,5x,f10.2,5x,f10.2)
35  continue
      write(45,(' Volume(in^3) = ',f10.2))vol
      write(45,(' Error = ',f10.9))error
      write(45,(' Weight(lbs) = ',f10.2))weight
      end

      subroutine fcni(nel,anew,func)
C
      parameter(maxnnod=20000,maxnel=20000)
      real*8 rho
      integer fcount,nnod,nel,inel,conn(2,maxnnod),bc(2,maxnnod)
      integer iparam(7),count
      real*8 x(2,maxnnod),f(2,maxnnod),u(2,maxnnod)
      real*8 e(maxnel),ainit(maxnel),a(maxnel),opta(maxnel),
      $amin(maxnel),anew(maxnel),ascale(maxnel)
      real*8 sig(maxnel),sigall(maxnel)
      real*8 vol,c,ga,gs,lpa,lps,func
      external truss2d,apldbc,arinlz,aslstf,clstrs,clstif,clvol,
      $facstf,slvdsp
      common rho,fcount

```

```

C
  fcount=fcount+1
  c=0.0
  write(*,*)fcount
  write(*,*)rho
C
  open(unit=1,file='project2.dat',status='unknown')
  read(1,*)nnod,inel
  do 5 i=1,nnod
    read(1,*)x(1,i),x(2,i),f(1,i),f(2,i),bc(1,i),bc(2,i)
5  continue
C
  do 10 i=1,inel
    read(1,*)ainit(i),e(i),conn(1,i),conn(2,i),sigall(i),amin(i)
10 continue
  close(1)
C
  do 15 i=1,nel
    a(i)=anew(i)**2
15 continue
C
  CALL truss2d(nnod,nel,e,a,conn,x,bc,f,u,sig,vol)
C
  do 20 i=1,nel
    ga=(anew(i)/amin(i))-1.0
    gs=1.0-DABS(sig(i)/sigall(i))
C
    if (anew(i).lt.amin(i)) then
      lpa=1.0e10*(ga)
    end if
C
    if (sig(i).gt.sigall(i)) then
      lps=1.0e10*(gs)
    end if
C
    c=c+(1.0/ga)+(1.0/gs)+lpa+lps
20 continue
C
  func=c+(vol/18000)
  write(*,*)func
  return
  end

```

**makefile:**

```

OBJ = project2_i.o fcni.o arinlz.o clstrs.o aslstf.o clvol.o truss2d.o apldbc.o
clstif.o facstf.o slvdsp.o $(LINK_FNL_STATIC)

```

```

truss2d: $(OBJ)
  f77 -o pr2i $(OBJ)

```

```

clean: $(OBJ)
  rm -f $(OBJ)

```

\*\*\*\*\*

**OUTPUT:** fn = *project2\_i.out*

10-bar Truss Optimization Results (Interior):

Number of function evaluations = 2273

Number of iterations = 11

Node Displacements:

node	u(in)	v(in)
1	.12	-.60
2	-.14	-.63
3	.09	-.26
4	-.09	-.26
5	.00	.00
6	.00	.00

Areas and Stresses:

element	area(in^2)	stress(psi)
1	5.19	7098.48
2	1.84	2500.72
3	5.42	-7098.02
4	4.57	-4374.36
5	1.42	.00
6	1.84	2500.72
7	4.65	7089.47
8	4.27	-7089.23
9	4.46	6509.43
10	1.39	-6195.89

Volume(in^3) = 14717.33

Weight(lbs) = 1471.73

- 4.) **10-Bar Truss Problem (DIRECT method):** Optimization of 10-bar truss problem using available IMSL routine DNCONF. The driver code *project2\_d.f* and the subroutine *fcnd.f* were written for this optimization code.

**Input data file:** *fn = project2.dat*

```
6 10
720.0 360.0 0.0 0.0 0 0
720.0 0.0 0.0 -100000.0 0 0
360.0 360.0 0.0 0.0 0 0
360.0 0.0 0.0 -100000.0 0 0
0.0 360.0 0.0 0.0 1 1
0.0 0.0 0.0 0.0 1 1
5.0 30000000.0 5 3 25000.0 0.1
5.0 30000000.0 3 1 25000.0 0.1
5.0 30000000.0 6 4 25000.0 0.1
5.0 30000000.0 4 2 25000.0 0.1
5.0 30000000.0 3 4 25000.0 0.1
5.0 30000000.0 2 1 25000.0 0.1
5.0 30000000.0 5 4 25000.0 0.1
5.0 30000000.0 6 3 25000.0 0.1
5.0 30000000.0 3 2 75000.0 0.1
5.0 30000000.0 4 1 25000.0 0.1
```

**program project2\_d**

C

```
parameter(maxnnod=2000,maxnel=2000)
integer nfcn,nnod,nel,conn(2,maxnnod),bc(2,maxnnod),ibtype,
$iprint,maxit,me,m
```

```

real*8 x(2,maxnnod),f(2,maxnnod),u(2,maxnnod)
real*8 e(maxnel),a(maxnel),sig(maxnel),sigall(maxnel)
real*8 ainit(maxnel),amax(maxnel),amin(maxnel),ascale(maxnel)
real*8 vol,weight
common nfcn
external fcnd,truss2d,clstif,arinlz,aslstf,apldbc,facstf,slvdsp,
$clstrs,clvol,DNCONF
C
open(unit=35,file='project2.dat',status='old')
C
read(35,*)nnod,nel
do 5 i=1,nnod
  read(35,*)x(1,i),x(2,i),f(1,i),f(2,i),bc(1,i),bc(2,i)
5 continue
C
do 10, i=1,nel
  read(35,*)ainit(i),e(i),conn(1,i),conn(2,i),sigall(i),amin(i)
10 continue
close(35)
C
nfcn=0
m=nel
me=0
ibtype=0
iprint=1
maxit=1000
C
do 15 i=1,nel
  amax(i)=100.0
  ascale(i)=5.0
15 continue
C
CALL DNCONF(fcnd,m,me,nel,ainit,ibtype,amin,amax,ascale,iprint,
$maxit,a,vol)
C
CALL truss2d(nnod,nel,e,a,conn,x,bc,f,u,sig,vol)
C
weight=vol*0.1
C
open(unit=45,file='project2_d.out',status='unknown')
write(45,*)'10-bar Truss Optimization Results (DNCONF):'
write(45,__(' Number of function evaluations = ', i6))nfcn
write(45,*)'Node Displacements:'
write(45,*)' node    u(in)    v(in)'
do 25 i=1,nnod
  write(45,20)i,u(1,i),u(2,i)
20 format(5x,i3,5x,f10.2,5x,f10.2)
25 continue
write(45,*)'Areas and Stresses:'
write(45,*)' element  area(in^2) stress(psi)'
do 35 i=1,nel
  write(45,30)i,a(i),sig(i)
30 format(5x,i3,5x,f10.2,5x,f10.2)
35 continue
write(45,__(' Volume(in^3) = ',f10.2))vol
write(45,__(' Weight(lbs) = ',f10.2))weight

```

end

subroutine **fcnd(m,me,nel,a,active,vol,g)**

```
C
parameter(maxnnod=2000,maxnel=2000)
integer m,me,nnod,nel,inel,conn(2,maxnnod),bc(2,maxnnod)
real*8 x(2,maxnnod),f(2,maxnnod),u(2,maxnnod)
real*8 e(maxnel),ainit(maxnel),amin(maxnel),sig(maxnel),
$sigall(maxnel)
real*8 a(*),g(*),vol
common nfcn
external truss2d,clstif,arinlz,aslstf,apldbc,facstf,slvdsp,
$clstrs,clvol
C
logical active(*)
C
nfcn=nfcn+1
C
open(unit=35,file='project2.dat',status='old')
C
read(35,*)nnod,inel
do 5 i=1,nnod
  read(35,*)x(1,i),x(2,i),f(1,i),f(2,i),bc(1,i),bc(2,i)
5 continue
do 10, i=1,inel
  read(35,*)ainit(i),e(i),conn(1,i),conn(2,i),sigall(i),amin(i)
10 continue
close(35)
C
CALL truss2d(nnod,nel,e,a,conn,x,bc,f,u,sig,vol)
C
do 15 i=1,m
  if (active(i)) then
    g(i)=1.0-DABS(sig(i)/sigall(i))
  end if
15 continue
return
end
```

**makefile:**

```
OBJ = project2_d.o fcnd.o arinlz.o clstrs.o aslstf.o clvol.o truss2d.o apldbc.o
clstif.o facstf.o slvdsp.o $(LINK_FNL_STATIC)
```

```
truss2d: $(OBJ)
  f77 -o pr2d $(OBJ)
```

```
clean: $(OBJ)
  rm -f $(OBJ)
```

\*\*\*\*\*

**OUTPUT:** fn = *project2\_d.out*

10-bar Truss Optimization Results (DNCONF):

Number of function evaluations = 166

Node Displacements:

node	u(in)	v(in)
1	.60	-2.40
2	-.60	-2.70
3	.30	-.90
4	-.30	-.90
5	.00	.00
6	.00	.00

Areas and Stresses:

element	area(in <sup>2</sup> )	stress(psi)
1	7.90	25000.00
2	.10	25000.00
3	8.10	-25000.00
4	3.90	-25000.00
5	.10	.00
6	.10	25000.00
7	5.80	25000.00
8	5.52	-25000.00
9	3.68	37500.00
10	.14	-25000.00

Volume(in<sup>3</sup>) = 14976.00

Weight(lbs) = 1497.60

**APPENDIX B – 2D TRUSS ANALYSIS CODE** (Developed by Asst. Prof. G. Subbarayan,  
Dept. of Mechanical Engineering, University of Colorado at Boulder)

```
subroutine truss2d(nnod,nel,e,a,conn,x,bc,f,u,sig,vol)
implicit real*8(a-h,o-z)

parameter (ldstif=20)

real*8 x(2,nnod),f(2,nnod),u(2,nnod)
real*8 estif(4,4),stif(ldstif,ldstif)
real*8 w(ldstif)
real*8 e(nel),a(nel),sig(nel),vol
integer conn(2,nel),bc(2,nnod)

c ... initialize stiffness and displacement arrays ...
call arinlz(ldstif*ldstif,stif(1,1))
call arinlz(2*nnod,u(1,1))

do 100 iel=1,nel

c ... compute the element stiffness matrix
c ... you need to write this routine

call clstif(nnod,nel,iel,e,a,conn,x,estif)

c ... assemble element stiffness matrix into global matrix

call aslstf(nnod,nel,iel,conn,estif,ldstif,stif)

100 continue

C ..... Apply displacement BCs .....

call apldbc(nnod,bc,ldstif,stif)

C ..... Factorize the stiffness matrix .....

ierr = 0.
call facstf(nnod,ldstif,stif,ierr)

if (ierr.ne.0) then
write(*,*) 'WARNING'
write(*,*) 'Stiffness matrix not positive definite'
write(*,*) 'Terminating analysis'
endif

C ..... Solve for displacements .....

call slvdsp(nnod,ldstif,stif,w,u(1,1),f(1,1))

call clstrs(nnod,nel,e,conn,x,u,sig)

call clvol(nnod,nel,a,conn,x,vol)

return
end
```

subroutine **apldbc(nnod,bc,ldstif,stif)**

implicit real\*8 (a-h,o-z)

real\*8 stif(ldstif,2\*nnod)  
integer bc(2,nnod)  
real\*8 bigfac

C=====

C This routine applies displacement BCs to the nodes.

C Ganesh Subbarayan 10/9/96

C=====

```
bigfac = 1.d20
do 30 inod=1,nnod
  do 20 j=1,2
    if (bc(j,inod).ne.0) then
      k = 2*(inod-1)+j
      stif(k,k) = stif(k,k) + bigfac
    endif
  20 continue
  30 continue
```

RETURN  
END

subroutine **arinlz(isize,array)**

C ..... Passed variables .....

real\*8 array(isize)

C=====

C This routine initializes any array.

C Variables:

C array array to be initialized of size isize.

C Ganesh Subbarayan 01/29/89

C=====

```
do 10 i=1,ysize
  array(i) = 0.D0
  10 continue
```

return  
end

subroutine **aslstf(nnod,nel,iel,conn,estif,ldstif,stif)**

implicit real\*8(a-h,o-z)

```
integer conn(2,nel)
real*8  estif(4,4),stif(ldstif,2*nnod)
```

```
C=====
```

```
C This routine assembles the element stiffness matrix.
```

```
C Variables:
```

```
C  stif Global stiffness matrix
```

```
C Ganesh Subbarayan 10/9/96
```

```
C=====
```

```
inod = conn(1,iel)
jnod = conn(2,iel)
```

```
i = 2*(inod-1)+1
j = 2*(jnod-1)+1
```

```
if (j.gt.i) then
```

```
  k = i
```

```
  l = j
```

```
else
```

```
  k = j
```

```
  l = i
```

```
endif
```

```
  stif(k,k) = stif(k,k) + estif(1,1)
  stif(k+1,k) = stif(k+1,k) + estif(2,1)
  stif(1,k) = stif(1,k) + estif(3,1)
  stif(1+1,k) = stif(1+1,k) + estif(4,1)
  stif(k+1,k+1) = stif(k+1,k+1) + estif(2,2)
  stif(1,k+1) = stif(1,k+1) + estif(3,2)
  stif(1+1,k+1) = stif(1+1,k+1) + estif(4,2)
  stif(1,l) = stif(1,l) + estif(3,3)
  stif(1+1,l) = stif(1+1,l) + estif(4,3)
  stif(1+1,l+1) = stif(1+1,l+1) + estif(4,4)
```

```
return
```

```
end
```

```
subroutine clstrs(nnod,nel,e,conn,x,u,sig)
```

```
  implicit real*8 (a-h,o-z)
```

```
  real*8  e(nel),x(2,nnod),u(2,nnod),sig(nel)
  integer conn(2,nel)
```

```
  real*8  c,s,lenx,leny,len
```

```
  do iel=1,nel
```

```
    inod = conn(1,iel)
```

```
    jnod = conn(2,iel)
```

```

lenx = x(1,jnod)-x(1,inod)
leny = x(2,jnod)-x(2,inod)
len = dsqrt(lenx**2+leny**2)

c = lenx/len
s = leny/len

sig(iel) = e(iel)*(c*(u(1,jnod)-u(1,inod))+
&          s*(u(2,jnod)-u(2,inod)))/len

end do

return
end

```

subroutine **clvol(nnod,nel,a,conn,x,vol)**

```

implicit real*8 (a-h,o-z)

real*8 a(nel),x(2,nnod),vol
integer conn(2,nel)

real*8 lenx,leny,len

vol = 0.d0
do iel=1,nel
  inod = conn(1,iel)
  jnod = conn(2,iel)

  lenx = x(1,jnod)-x(1,inod)
  leny = x(2,jnod)-x(2,inod)
  len = dsqrt(lenx**2+leny**2)

  vol = vol + a(iel)*len

end do

return
end

```

subroutine **facstf(nnod,ldstif,stif,ierr)**

```

implicit real*8 (a-h,o-z)

real*8 stif(ldstif,2*nnod),eps

c =====
c This routine replaces the lower triangular portion
c of the stiffness matrix with its cholesky factor
c
c Algorithm based on Golub and Van Loan 5.2-1
c
c Ganesh Subbarayan 10/9/96

```

c =====

```
    eps = 1.e-10

    do 40 k=1,2*nnod
      do 10 p=1,k-1
        stif(k,k) = stif(k,k) - stif(k,p)**2
10      continue

        if (stif(k,k).lt.eps) then
          ierr = k
          return
        else
          stif(k,k) = dsqrt(stif(k,k))
        endif

        do 30 i=k+1,2*nnod
          do 20 p=1,k-1
            stif(i,k) = stif(i,k) - stif(i,p)*stif(k,p)
20          continue
            stif(i,k) = stif(i,k)/stif(k,k)
30          continue
40          continue

    return
    end
```

subroutine **slvds**(**nnod,ldstif,stif,w,u,f**)

```
    implicit real*8 (a-h,o-z)

    real*8 stif(ldstif,2*nnod),w(2*nnod),u(2*nnod),f(2*nnod)
```

C ..... Forward substitution .....

```
    do 20 i=1,2*nnod
      w(i) = f(i)
      do 10 j=1,i-1
        w(i) = w(i) - stif(i,j)*w(j)
10      continue
      w(i) = w(i)/stif(i,i)
20    continue
```

C ..... Backward substitution .....

```
    do 40 i=2*nnod,1,-1
      u(i) = w(i)
      do 30 j=i+1,2*nnod
        u(i) = u(i) - stif(j,i)*u(j)
30      continue
      u(i) = u(i)/stif(i,i)
40    continue
```

```
    return
    end
```