NASA Technical Paper 3457

1994

CORSS: Cylinder Optimization of Rings, Skin, and Stringers

J. Finckenor, P. Rogers, and N. Otte
George C. Marshall Space Flight Center
Marshall Space Flight Center, Alabama

National Aeronautics and Space Administration
Office of Management
Scientific and Technical Information Program

TABLE OF CONTENTS

	Page
INTRODUCTION	1
OPTIMIZATION TERMS	1
DESIGN OPTIMIZATION	3
Modified Method of Feasible Directions	3
Sequential Linear Programming	4
MMFD Versus SLP	8
PROGRAM DEVELOPMENT AND USE	8
Development	8
Running the Program	9
Program Input	9
Program Output	14
Program Use	15
OVERALL CYLINDER CALCULATIONS	17
STRINGER PROPERTY CALCULATIONS	19
SKIN BUCKLING STRESS CALCULATIONS	21
STRINGER CRIPPLING CALCULATIONS	24
COLUMN BUCKLING CALCULATIONS	26
GENERAL CYLINDER BUCKLING CALCULATIONS	30
VON MISES STRESS CHECK	33
SUMMARY	34
REFERENCES	35
APPENDIX A	37
APPENDIX B – Sample Input File	43
APPENDIX C – Sample Output File	44
APPENDIX D – Program Listing	49

LIST OF ILLUSTRATIONS

Figure	Title	Page
1.	Typical launch vehicle structure	2
2.	Modified method of feasible directions	5
3.	Example of optimization problem	6
4*a*.	Normalized design variable history	7
4*b*.	Design constraint history	7
5.	Sequential linear programming	8
6.	Stringer dimensions	11
7.	Ring parameters	12
8.	Conical analysis	16
9.	Mode shapes for panel and general instability of stiffened cylinders in bending	29

TECHNICAL PAPER

CORSS: CYLINDER OPTIMIZATION OF RINGS, SKIN, AND STRINGERS

INTRODUCTION

Launch vehicle designs typically make extensive use of cylindrical skin stringer construction. Structural analysis methods are well developed for preliminary design of this type of construction. This report describes an automated, iterative method to obtain a minimum weight preliminary design.

Structural optimization has been researched extensively, and various programs have been written for this purpose. Their complexity and ease of use depends on their generality, the failure modes considered, the methodology used, and the rigor of the analysis performed. This computer program employs closed-form solutions from a variety of well-known structural analysis references and joins them with a commercially available numerical optimizer called the "Design Optimization Tool" (DOT).

The program was written to aid in preliminary design work on the National Launch System (NLS) project (fig. 1), but it can be used for any ring and stringer stiffened shell structure of isotropic material that has the same type of loading. Plasticity effects are not included. It performs a more limited analysis than programs such as PANDA, but it provides an easy and useful preliminary design tool for a large class of structures.

This report briefly describes the optimization theory, outlines the development and use of the program, and describes the analysis techniques that are used. Examples of program input and output, as well as the listing of the analysis routines, are included.

OPTIMIZATION TERMS

Design constraints – Equations, usually inequality equations (such as "applied stress must be less than the allowable stress"), that must be satisfied in order for a design to be feasible. DOT needs them in a form so that when the constraint equation value is negative, it is satisfied.

Design variables – The values that define the design, such as the thickness of a plate, that are changed by the computer during optimization.

DOT – A commercially available optimization program. VMA Engineering, Vanderplaats, Miura & Associates, Inc.[1]

Objective function – The function that is to be minimized (or maximized) during the optimization.

Optimization – Numerical method making use of a computer to evaluate many different designs quickly, and to select the best design based on a user-defined objective or cost function.

Note: The variable listed to the right of the equations in this report is that used in the program.

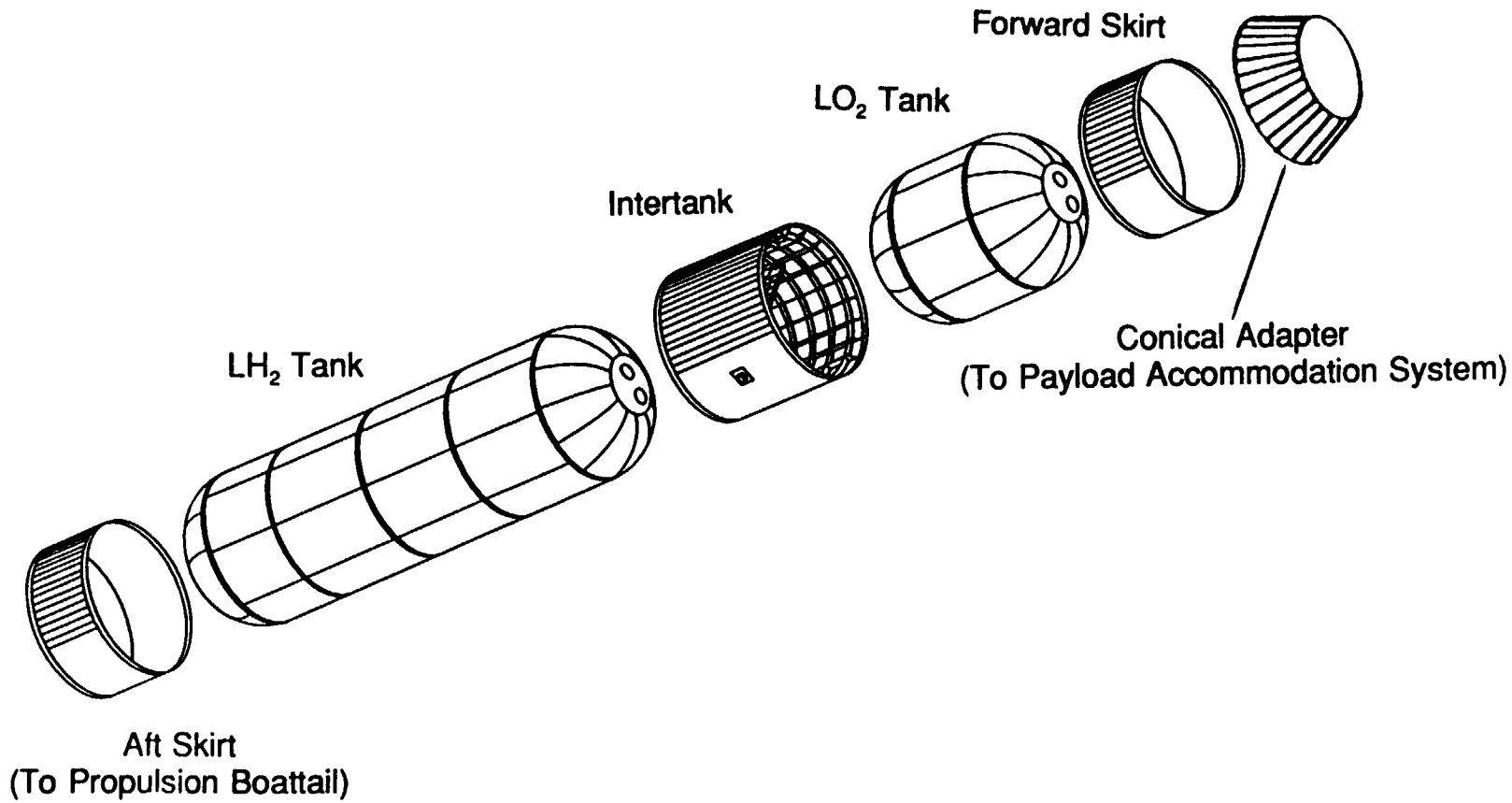


Figure 1. Typical launch vehicle structure.

DESIGN OPTIMIZATION[2 3]

Variables:

F	objective function to be optimized, function of X_i
G_j	the array of limiting constraints, functions of X_i
S	optimization search vector
X_i, XL_i, XU_i	the arrays of the design variables, and the lower and upper design variable limits, respectively
β, Φ, ϕ	optimization parameters.

Modified Method of Feasible Directions

The modified method of feasible directions (MMFD) is an optimization method that follows the driving constraints of a problem to the optimum solution. If the initial values make for a feasible design, the first iteration moves in the direction of steepest descent of the objective function to make it as small as possible until limited by a constraint. Then the constraints are followed to the optimum. It finds optimum solutions quickly and handles equality constraints well.

MMFD minimizes: $F(X_i)$ $\quad i = 1,N$

Subject to: $G_J(X_i) \le 0$ $\quad J = 1,M$

$XL_i \le X_i \le XU_i$

The direction of steepest descent of $\bar{F}$ is found by:

$$\bar{S} = -\nabla \bar{F} \quad (\text{or } \bar{S} = \nabla \bar{F} \text{ depending on the curvature of } \bar{F}) .$$

If there are no active constraints, $G_J < 0$, the intersection of $\bar{S}$ and the limiting constraint are found by any of several root finding methods. This is shown in figure 2 in the move from the initial point to the first iteration.

One or more active constraints serve to push the search direction away from the direction of steepest descent.

If: $G_j \simeq 0$ The set of active constraints, j, is a subset of all constraints, J.

Minimize: $\nabla \bar{F} \cdot \bar{S}$

Subject to: $\nabla G_j \cdot \bar{S} \le 0$

$\bar{S} \cdot \bar{S} \le 1$.

$\bar{S}$ wants to be close to the steepest descent to make $\nabla\bar{F}\cdot\bar{S}$ as negative as possible. $\bar{S}$ cannot get any steeper than the tangent to the constraint curve to keep $\nabla G_j\cdot\bar{S}$ negative. $\bar{S}\cdot\bar{S}$ being less than 1 limits the size of $\bar{S}$, so the subproblem is bounded. This is shown in figure 2 in the move from the first iteration to the second iteration.

If there are one or more violated constraints, the procedure reverts to the method of feasible directions (MFD) to return to the feasible region while increasing $\bar{F}$ as little as possible.

Maximize:	$-\nabla\bar{F}\cdot\bar{S}+\Phi\beta$	Φ = large positive number
Subject to:	$\nabla\bar{G}_j\cdot\bar{S}+\phi_j\beta\le 0$	ϕ = large calculated number for violated constraints
	$\bar{S}\cdot\bar{S}\le 1$.	

Maximizing $-\nabla\bar{F}\cdot\bar{S}+\Phi\beta$ chooses an $\bar{S}$ to minimize the increase in $\bar{F}$ and maximize the value β. β must be small to meet the ∇G_j constraint which is otherwise trying to send $\bar{S}$ directly back to the border of the constraint. A one-dimensional search is also applied to choose a point on the constraint boundary. This is shown in the move from iteration 2 to iteration 3 in figure 2.

Iterations are repeated in the appropriate manner until the optimum is found. The optimization iterations of a beam, as an example of a simple practical problem, are shown in figure 3. The cross-sectional area is the objective function.

A single iteration of DOT using MMFD has several calls through the analysis part of the program to find the gradients of the objective function and the constraints. An iteration is a major change of the design variables after the search direction has been calculated. A typical run of the program may have 10 DOT iterations and 100 analysis runs.

Figure 4 shows a typical optimization history for the forward skirt of the NLS rocket. Figure 4*a* shows the history of the normalized design variables and weight. It can be seen that the optimizer makes small changes to one variable at a time to find the gradients, and then takes large jumps to find the borders of the constraints. Figure 4*b* shows the more critical constraint values of the problem. The relation between specific variables and constraints can be seen in the design variable and the constraint spikes. For example, sharp drops in skin thickness show a sharp rise in the skin buckling constraint.

Sequential Linear Programming

Sequential linear programming (SLP) linearizes both the objective function and the constraints at the current X_i values. It then solves for the X_i that gives the lowest $\bar{F}$ within the now linear constraints. At this new point, the objective function and the constraints are linearized again and the process is repeated (fig. 5). This is a far simpler method to program, and, although it is not as efficient mathematically, it can be very useful in practice. It is not very good for problems which are underconstrained because it can go far from the optimum while still decreasing $\bar{F}$ slightly.

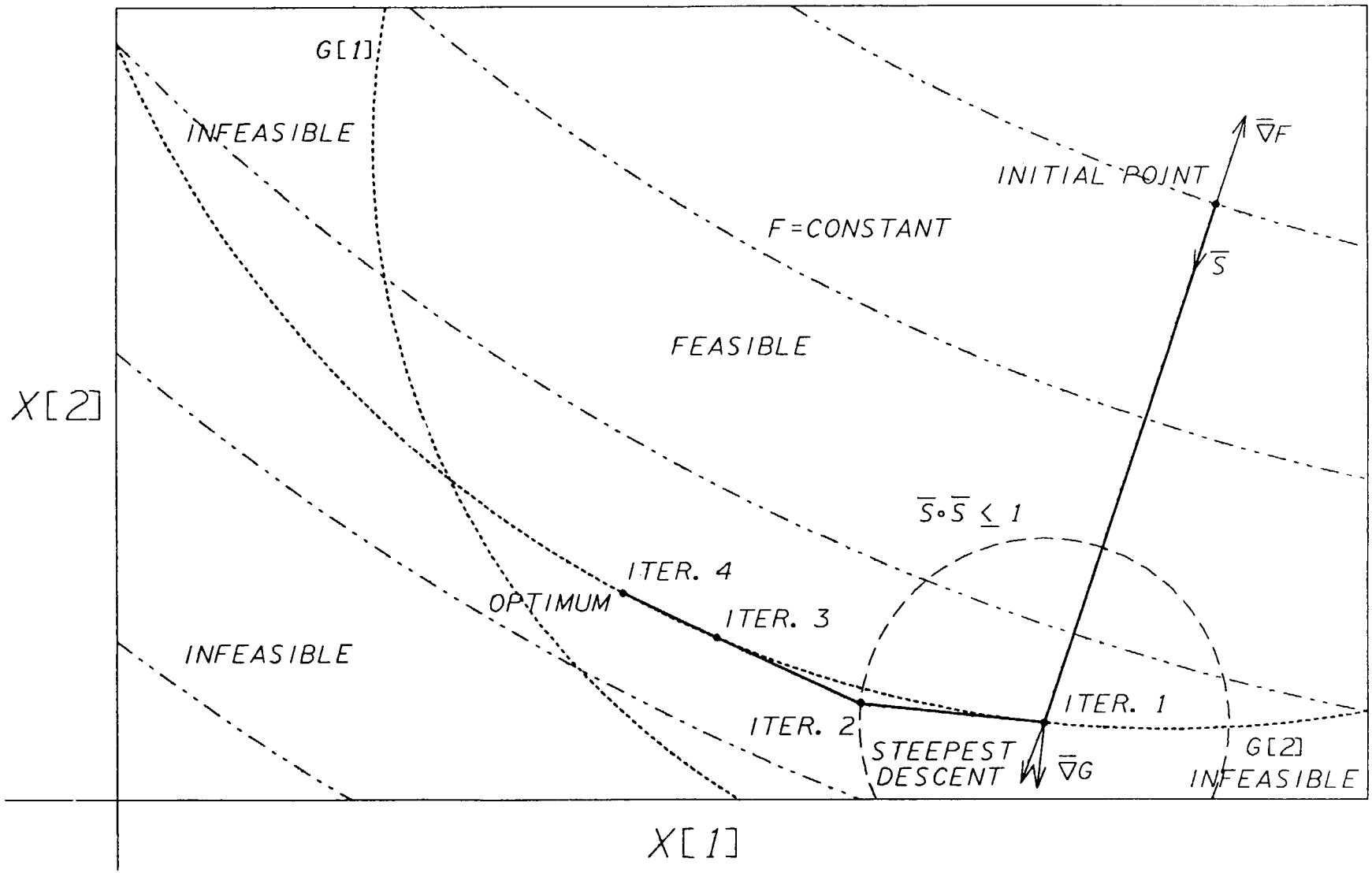
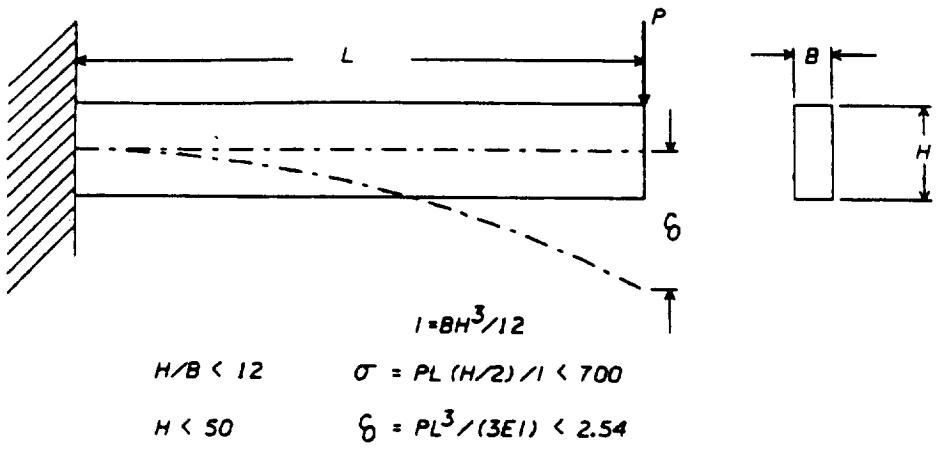


Figure 2. Modified method of feasible directions.



$I = BH^3/12$

$H/B < 12 \qquad \sigma = PL(H/2)/I < 700$

$H < 50 \qquad \delta = PL^3/(3EI) < 2.54$

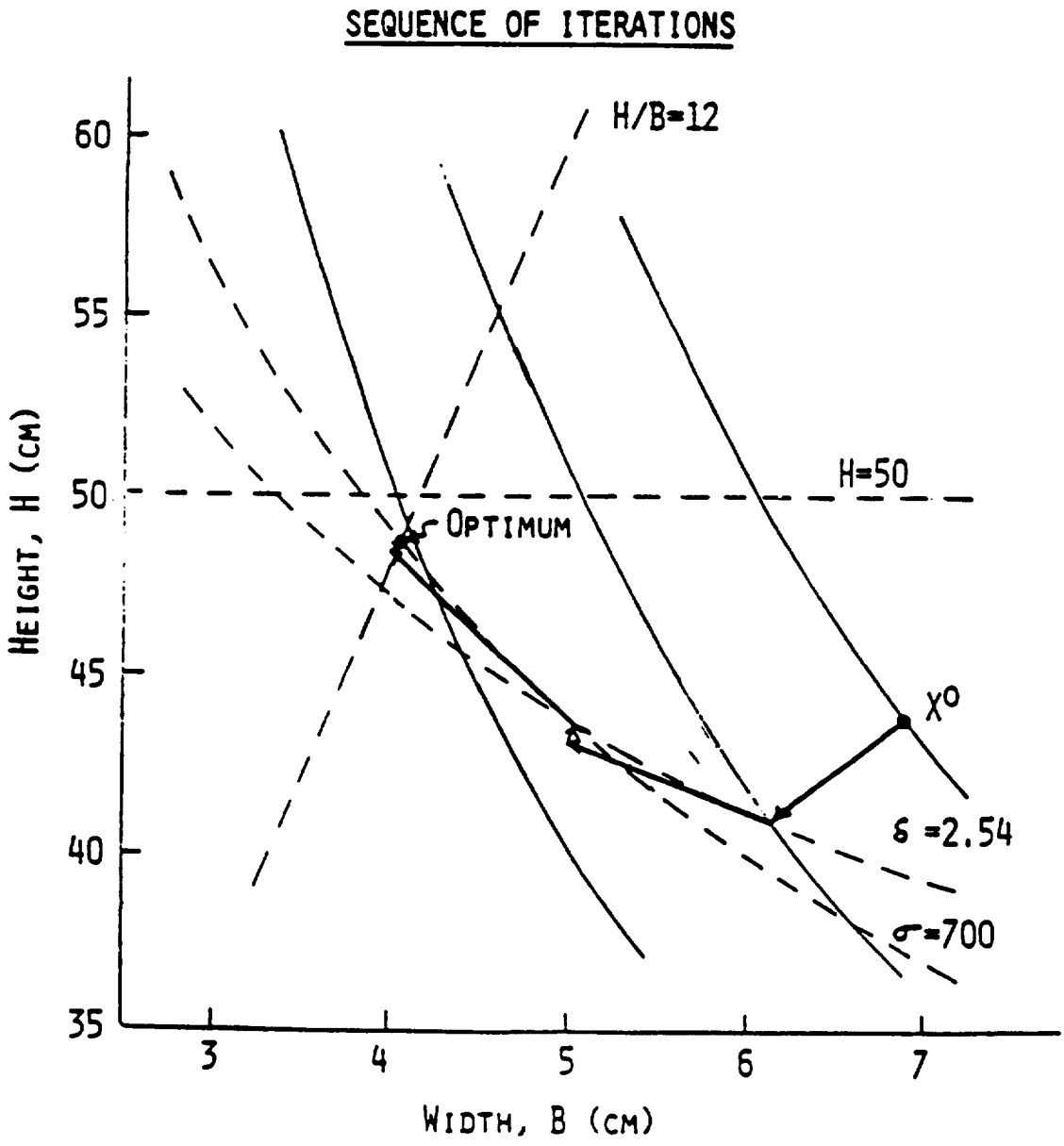


Figure 3. Example of optimization problem.[2]

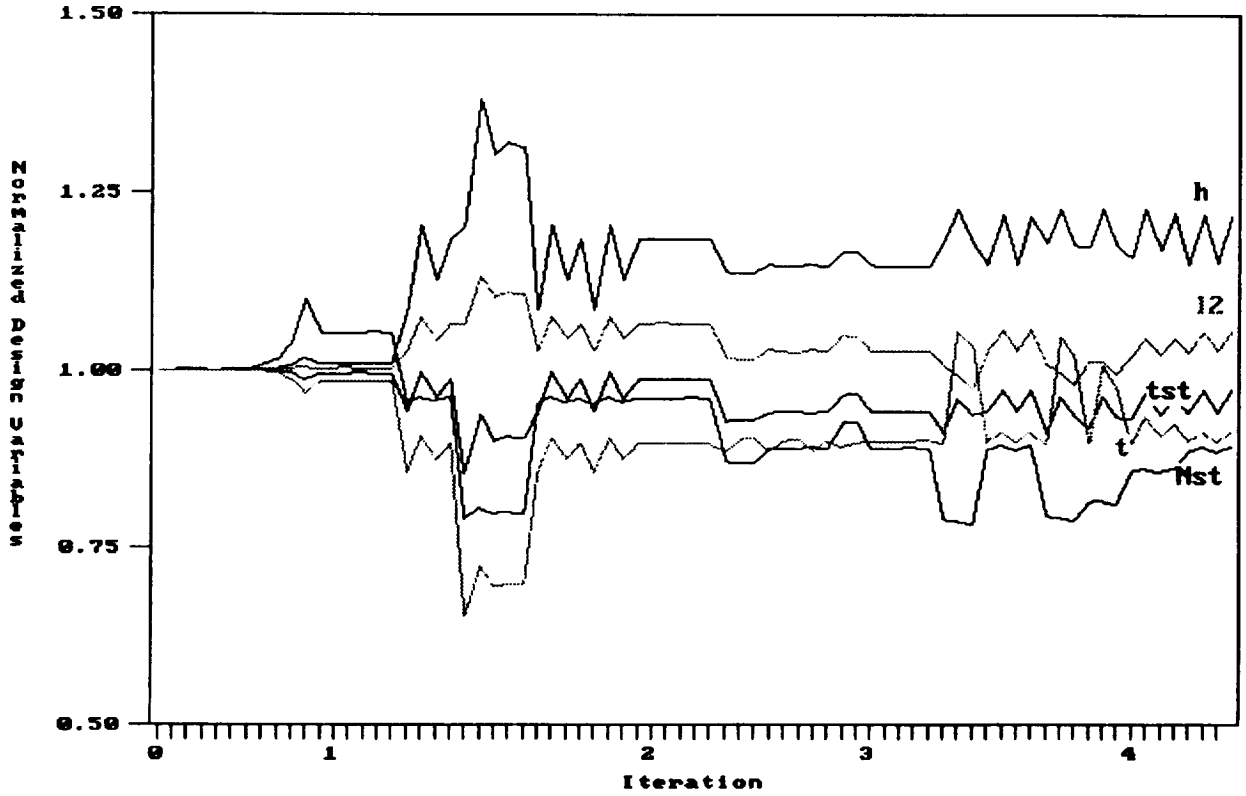


Figure 4*a*. Normalized design variable history.

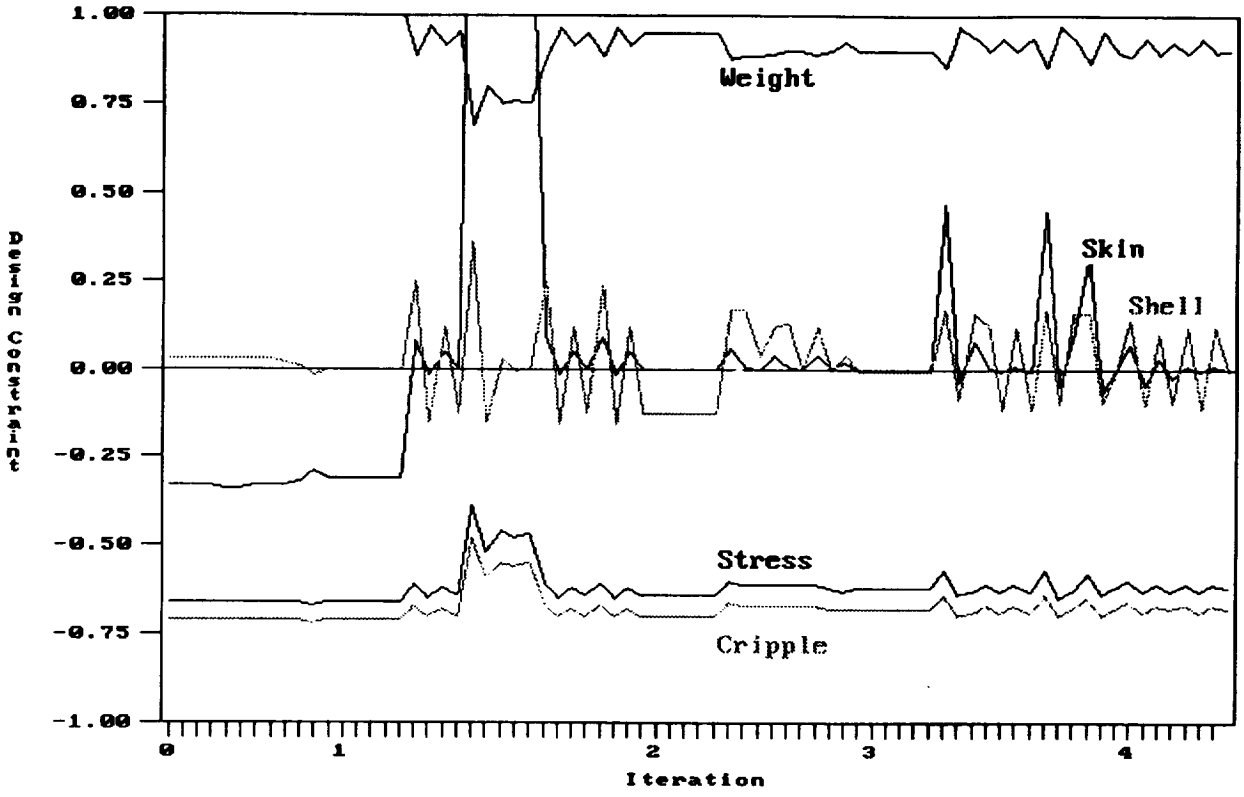


Figure 4*b*. Design constraint history.

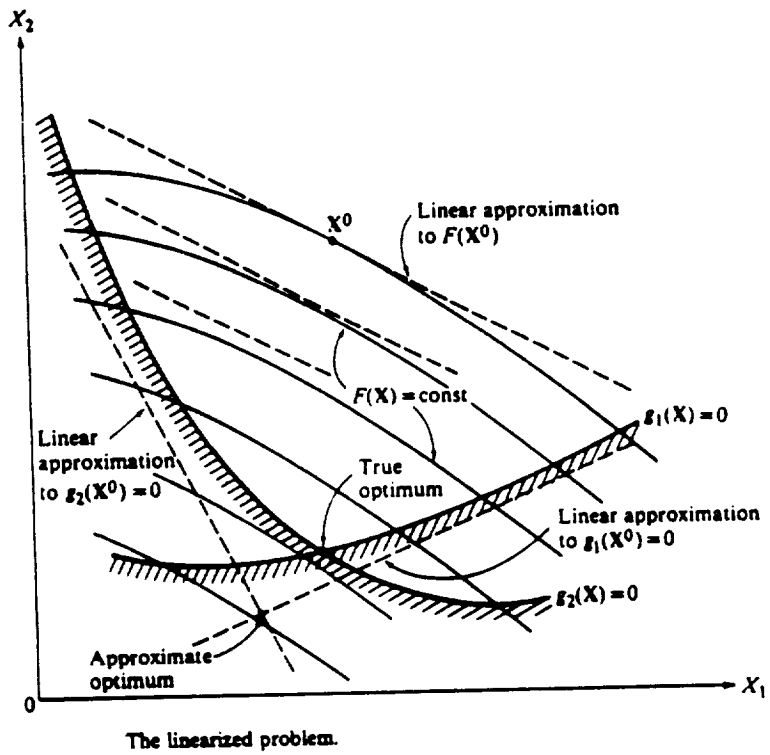


Figure 5. Sequential linear programming.[3]

MMFD Versus SLP

For this problem, the MMFD method (METHOD = 0 or 1 in the input file) usually seems to be the best choice for speed and accuracy of the final solution. In some cases, particularly when most of the constraints are active, DOT may have to make a large number of function calls for a single iteration (more than 15 calls/iteration). In these cases, it may be better to use the SLP (METHOD = 2) method which makes more iterations, but needs only a few function calls for each iteration.

PROGRAM DEVELOPMENT AND USE

Development

CORSS was developed on an 80386 PC, with run times ranging from 6 s to 1 min depending on how close to optimum the initial values are. The DOT portion of the program is written in FORTRAN and was compiled into an optimization library. The header program for DOT, which contains the stress and stability analysis, is written in C.

CORSS was originally written to help design the forward skirt of the NLS, however, it proved useful in many applications on the vehicle. It was used on the payload carrier adapter, the interstage, the oxygen tank, the intertank, the hydrogen tank, and the aft structure.

Running the Program

CORSS uses a text input file and creates a text output file. The DOS command line looks like:

C:> CORSS infile outfile

where *infile* and *outfile* are user defined. A menu-driven, graphical user interface was also written to enhance the use of the program, however it is not required to run the program.

Program Input

The input file (appendix B) includes a title line, DOT flags, an output option flag, material properties, cylinder geometry parameters, stringer parameters, ring parameters, loads parameters, and the design variables with their associated upper and lower bounds. Shown here is the variable name for each value used in the program with a short description (less than one line) that can be kept in the input file. The first line of the input file must be included, and is printed as a title in the output file.

DOT Flags

IPRINT	Screen output by DOT 0=none 7=most
METHOD	method: 0,1 = MMFD, 2=SLP

"IPRINT" and "METHOD" are values used by DOT. "IPRINT" is an output flag. An "IPRINT" of 0 means no output, and an "IPRINT" of 7 is the maximum output by DOT which is sent to the screen and includes design variables, constraint values, gradients, and the objective function value for each iteration. "METHOD" is the numerical optimization method used to find the minimum objective function. If "METHOD" is 0 or 1, the MMFD is used. If "METHOD" is 2, the SLP method is used.

CORSS Output Option Flag

SSP	Output, 0-final, 1-calcs+0, 2-dv/con+0, 3-dv/con+1

"SSP" is the output flag for CORSS. If "SSP" is 0, only the inputs and final values will be printed out. If "SSP" is 1, values used during the calculations, such as effective widths, get printed out in addition to the final values. If "SSP" is 2, the optimization history of the problem and the final values are printed. The optimization history includes the design variables, constraints, weight, and iteration number during each analysis. If "SSP" is 3, the optimization history, calculation values, and final output are all printed.

Material Properties

nu	Poisson's ratio
E	Young's modulus, compressive
SM	Shear modulus
rho	Material density
Stu	Ultimate tensile stress (enter as positive)
Scy	Yield compressive stress (enter as positive)

Cylinder Geometry Parameters

r	Radius of cylinder
l	Length of cylinder
fwt	Extra non-optimized weight

The variables, "r," "*l*," and "fwt" are the overall cylinder geometry parameters. The "fwt" term is only added onto the objective value for completeness of the final weight and is not used in any other calculations. It is intended to be the weight of the end rings (such as the forward skirt to lox tank ring and the forward skirt to interstage ring) plus any other additional weights to be included but not optimized (such as feedthrough reinforcements).

Stringer Parameters

stype	stringer type: "H" for hat-stringers, "I" for I-stringers
LEB	Allow Local Elastic Buckling [Y/N]
mflag	I-str coupled buckling flag, 0=find bucket, #=search m=1 to #
alp	web angle in degrees, 0 is perpendicular to skin
ll	Hat to skin length (2/hat), or height of I bottom flange
MZs	1 for external stringers, –1 for internal stringers

The variables "stype," "LEB," "mflag," "alp," "*ll*," and "MZs" are the stringer parameters (fig. 6). The "stype" term is the stringer type. It must be a capital H for hat-stringers, or a capital I for I-stringers.

"LEB" controls the selection of constriants. For hat-stringers an "N" sets two constraints to be the elastic buckling of the web and the top flange. A "Y" replaces those two constraints with a single constraint for crippling of the entire stringer. "mflag" is for I-stringers only and is not used in the analysis of hat sections. The "alp" term is the angle of the legs on a hat-stringer, with 0 being perpendicular to the skin. The "*ll*" term is the length of one of the two legs on a hat-stringer that connects to the skin. This length is where rivets, welds, etc., would be used to fasten the stringer to the skin.

For I-stringers, an "N" for "LEB" sets a constraint to be elastic buckling of the web, and a "Y" replaces that constraint with crippling of the entire stringer. An I-stringer will always calculate a coupled buckling constraint which prevents the stringer from having too small a top flange to provide a simple support for the web. This buckling constraint is solved iteratively by the computer for a range of buckled waves, m. The "mflag" term controls what waves are searched. With an "mflag" of "0," the program will start with m=1 and increase m as long as the minimum critical stress continues to decrease. This is the suggested method during an optimization to keep run time down. If "mflag" is a number, the minimum buckling stress calculated for the range of m from 1 to the given number will be used for the constraint. This method should be used on an analysis only run to verify that there are no local minimums of the buckling equation. "alp" is the angle of the web of an I-stringer, with 0 being perpendicular to the skin. Applying this angle to the web of an I-stringer approximates a Z-stringer, but no checks are made to see if the Z is physically possible. Also, the coupled buckling term was derived for an 'I' section, and should not be considered valid for a "Z" section. The "*ll*" term is the thickness of the bottom flange of the I. In the NLS hydrogen tank, this was 0.02 in to allow for machining tolerance. This additional thickness is not included in skin buckling calculations.

"MZs" is –1 for internal stringers and 1 for external stringers. It is multiplied by the neutral axis distance of the stringer to place it on the correct side of the skin.

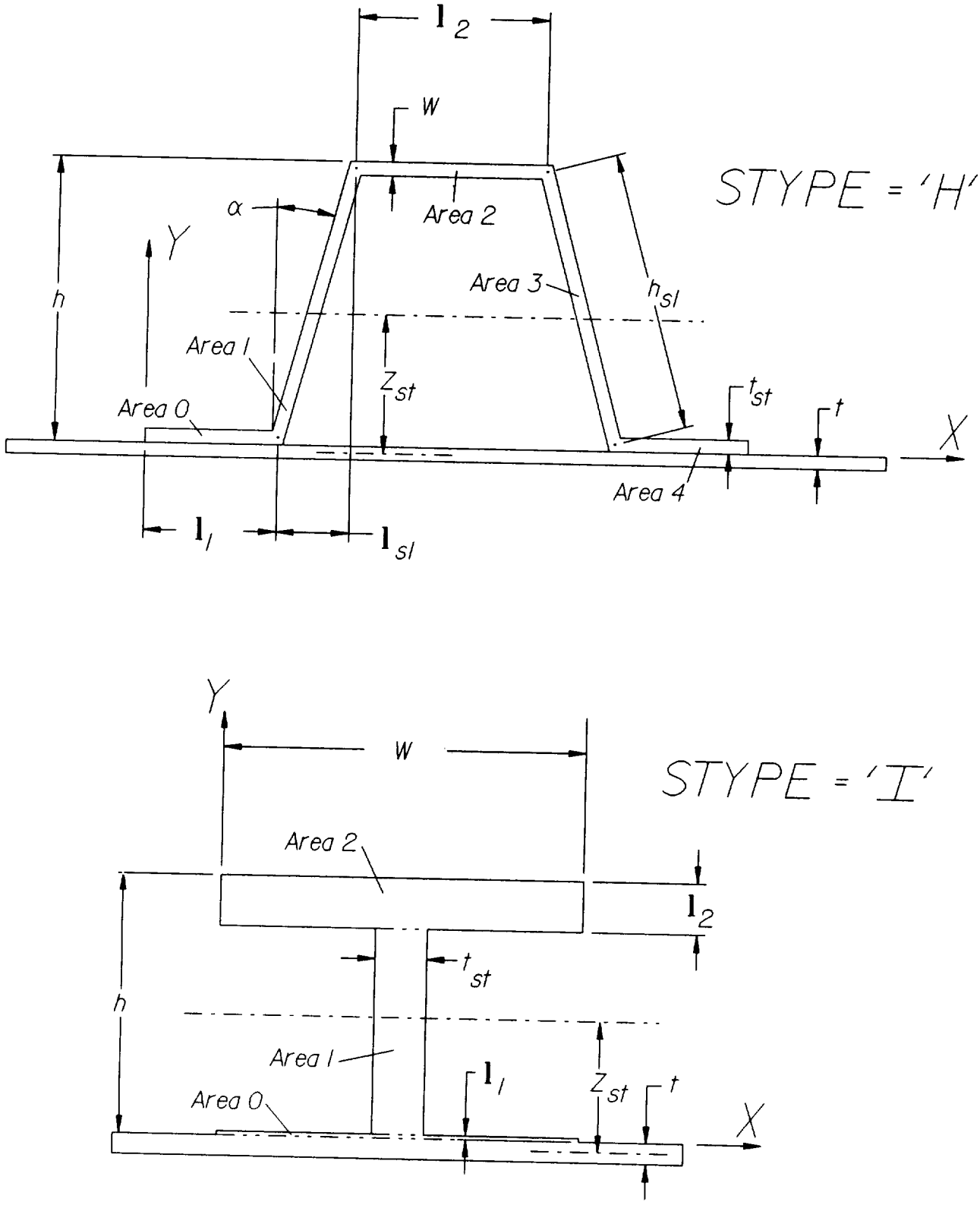


Figure 6. Stringer dimensions.

Ring Parameters

Ar	ring cross-sectional area
Ir	ring Moment of Inertia
Zr	ring centroid (– for internal rings)
Jr	ring torsion constant
Nrng	Number of intermediate rings

The variables "Ar," "Ir," "Zr," "Jr," and "Nrng" are the ring properties (fig. 7). The ring is not designed by the program, therefore the ring properties must be entered. Stringer column buckling calculations use only the ring spacing, assumed even, as the column length. The required stiffness of a ring is printed out. The general cylinder buckling calculations use all of the ring properties. The cross-sectional area of the ring is "Ar," the moment of inertia about the neutral axis parallel to the skin is "Ir." The neutral axis distance from the skin center line, negative for internal rings, is "Zr." The torsion constant of the ring is "Jr." The number of rings, not including end rings, is "Nrng." It is best to model segments between widely spaced ring frames as individual cylinders.

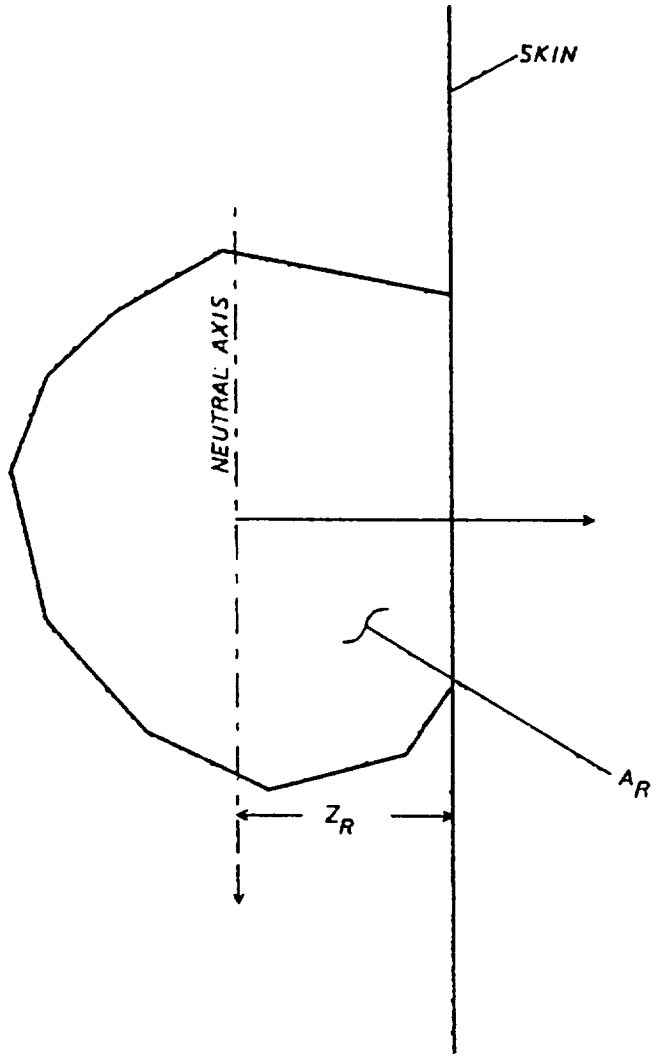


Figure 7. Ring parameters.

Load Parameters

F	Applied axial force, positive is compressive
M	Applied bending moment, enter as positive
V	Applied shear force, enter as positive
Pa	Pressure for axial loads, i.e., tank ullage (+ internal)
Ph	Pressure for hoop loads, i.e., ullage + head (+ internal)
sf	overall safety factor
sfp	skin buckling safety factor

"Pa," the axial pressure load, is separated from "Ph," the hoop pressure load, because of their use in the calculations. A bottom-supported fuel tank, for example, would have a high hoop pressure at the bottom due to both the head pressure and the ullage pressure. However, the head pressure does not provide any tensile load in the axial direction. The "sf" term is the safety factor to be used. This applies to all stress loads and buckling, except skin buckling. The "sfp" term is the skin buckling safety factor and is assumed to be at least 1.0 and less than "sf." For the NLS design, a 1.4 safety factor was applied, with a 1.0 safety factor on skin buckling. This means that at the limit load, no buckling at all can take place, and between 1.0 and 1.4 times the limit load skin buckling is allowed, but all other failure modes must maintain the 1.4 safety factor with the skin buckled.

Design Variables

XL[0]	minimum stringer height	
h	initial stringer height	
XU[0]	maximum stringer height	
XL[1]	minimum, hats: flange length, I's: top flange thickness	
*l*2	initial, hats: flange length, I's: top flange thickness	
XU[1]	maximum, hats: flange length, I's: top flange thickness	
XL[2]	minimum, hats: stringer thickness, I's: web thickness	
tst	initial, hats: stringer thickness, I's: web thickness	
XU[2]	maximum, hats: stringer thickness, I's: web thickness	
XL[3]	minimum, number of stringers	
Nst	initial, number of stringers	
XU[3]	maximum, number of stringers	
XL[4]	minimum, skin thickness	
t	initial, skin thickness	
XU[4]	maximum, skin thickness	
XL[5]	minimum, hats: top flange thickness, I's: width,	XL[5]=W=XU[5]=0
W	initial, hats: top flange thickness, I's: width,	sets W=tst for hats
XU[5]	maximum, hats: top flange thickness, I's: width	

The XL and XU arrays are the upper and lower limits of the design variables. The DOT program requires that the design variables be in the form of an X array, but this is converted in CORSS to make a more readable code. The stringer height, not including skin thickness, is "h." The top flange length of a hat-stringer, or the top flange thickness of an I-stringer, is "*l*2." The typical thickness of a hat, or the web thickness of an I, is "tst." The number of stringers is "Nst," and the skin thickness is "t." If W, XL[5], and XU[5] are all set to 0 for a hat-stringer, a constant thickness of tst is assumed.

Program Output

The first part of the program output (appendix C), which is always printed, is the list of input values. The second part of the output, printed if "SSP" is 2 or 3, is the optimization history with design variables and constraint values from every run through the analysis. The third part of the output, printed when "SSP" is 1 or 3, are values used in the calculations. The final results, which are always printed, include a weight breakdown, a listing of the optimum design variables and the constraining values.

Optimization History

The design variables of each analysis are listed along with the calculated constraint values for that call. Also included are the weight and the DOT iteration number.

Calculations for the Optimum Configuration

The first output from the final optimized analysis is the "stringer property calculations." This gives the individual segment values that are summed to calculate the moments of inertia, neutral axis, and stringer area.

The "overall cylinder calculations" include the moment of inertia of the cylinder and the maximum tension stress, assuming that no skin is buckled. The applied stress without the moment load with the safety factor and the skin buckling safety factor is also output. The ring and stringer spacings are printed.

The "skin buckling and max shear calculations" give the axial stress and the shear flow values for each skin segment through at least 90° of the cylinder and the value of the skin buckling constraint in that segment. The maximum values of shear stress and skin stress are saved, as well as the stress values for the most critical section.

The "stringer crippling calculations" provides information on both stringer crippling and elastic buckling, if required. The stringer crippling stress is always calculated and saved in case it is needed to calculate Johnson-Euler buckling. If local elastic buckling is allowed, the critical buckling stresses are not calculated or printed, and the crippling value is used for a constraint. For hat-stringers, if elastic buckling is not allowed, the buckling values of the top flange and the web are calculated and used for constraints. For I-stringers, the coupled buckling stress is always used for a constraint, and the stress calculated for each number of waves is printed. If local elastic buckling is not allowed, the critical elastic stress of the web is used as a constraint.

"Stringer column buckling calculations" prints the critical buckling load for the combined stringer/skin column, as well as the applied stress, the effective width of skin on the most highly stressed stringer, and the combined moment of inertia and radius of gyration. If the skin buckled, the section starts with the number of iterations that were required to converge on the effective cylinder moment of inertia, and the maximum tension stress calculated from the new section properties. The first table lists the distance from the cylinder center line of each stringer and the skin next to it, the stress for each stringer and skin, and the effective skin width associated with each stringer. Values are only calculated over half the cylinder, and only where the skin is buckled, with the other half being assumed symmetrical. The second table lists the areas and summed properties that are used for the new cylinder moment of inertia calculation. Only the sections that have skin removed are listed, and if hat-stringers are being used, the listing includes whether it is between stringers (skin) or between the legs of a single stringer (stringer).

The "general cylinder buckling calculations" are not performed if the skin has buckled. When the skin does not buckle, the smeared orthotropic cylinder stiffness properties and A-matrices for both axial and pressure buckling are printed. Also shown are the determinants of the two-by-two and the three-by-three A-matrices. If the number of axial half waves equals the number of cylinder segments between rings, the rings are not supporting the cylinder, and a warning that the smeared properties will not be valid is printed. The number of axial half waves and hoop waves is printed out. The axial and bending knockdown factors are printed, as well as the line load as adjusted by the knockdown factors. A message is printed stating that either general cylinder buckling or stringer column buckling provides the greatest support. The more favorable of the two values is used in the shell buckling constraint.

The "stress check" performs a Von Mises stress evaluation at three points, and the most highly stressed point is used in the constraint. The first point is the maximum compression location where the bending moment applies compression at the "top" of the shell. The second point is along the centerline of the moment application where shear is the highest and bending stresses are the lowest. The third point is the maximum tension location where the moment applies tension at the "bottom" of the shell. This point can still be in compression, depending on the axial load applied.

Final Results

The weight of the cylinder is printed along with the optimum design variables. The stringer properties and the required ring stiffness and area are listed. The ring area is a function of the ring depth, Z, which is negative for internal rings. Also listed are the constraint values—which must be negative to be satisfied. The skin buckling ratio and the general cylinder buckling ratio must be less than 1, and the applied loads must be less than the critical loads. A warning is printed for any violated constraints.

Program Use

This design problem is nonlinear, has several discontinuities, and can be susceptible to solving for local minimums. The suggested use of the program is to select a large range between the upper and lower limits of the design variables. Then run the first case with the initial design variables set near the lower bound, and the second with the design variables set near the higher bound. If the results do not agree fairly well, apply engineering judgment to the results from the first two runs to select initial values near what the optimum should be.

It is also important to look at the output to see what constraints are driving the design. There are usually at least two driving constraints. The NLS designs are usually driven by skin buckling and either column buckling or local stringer buckling. If there is a very high value for a constraint, such as a critical skin buckling stress of 400,000 lb/in^2, then that may indicate that a much lower initial value for skin thickness should be tried. If the only driver is shell buckling, that may indicate that additional rings are needed to shorten the column length.

If general cylinder buckling is a driver, it may be important to make sure that the use of fewer rings will not allow buckling. The assumption in general buckling is that the rings are spaced closely enough that their properties can be smeared into an orthotropic shell. For large, widely spaced rings, this is not a valid assumption, and a more conservative case should be checked. It is best to run each segment between widely spaced rings independently.

After running the program and getting results, such as 147.3 stringers and thicknesses of 0.0697 inch, the constraints and initial values can be set to more realistic numbers, such as 147.0 stringers and 0.071 (a standard aluminum gauge thickness). When the initial value and both the upper and lower bounds are set equal, the variable is treated as a constant and is removed from the optimization routines.

This elimination of variables can significantly reduce run time, and allows the program to perform a skin stringer analysis only, with no optimization calls, by setting all the design variables as constants.

By using I-stringers and setting the top flange thickness and width to a very small number, simple blade stringers can be modeled. The width must be the smaller of the two values.

When using I-stringers, the user should be careful that the sum of the bottom flange thickness, *l*1, and the maximum top flange thickness, *l*2, cannot exceed the minimum stringer height, h. If this happens, the program will exit with a warning to prevent a divide by 0 run time error.

Cones

Although CORSS was written for cylinders, cones with a half angle less than 30° can be modeled in a slightly more lengthy process. Assuming that the end rings are sufficient to take out the radial force components, the axial load should be broken down into components along the side of the cone, and the applied axial load and shear adjusted (fig. 8):

$$F_{eq} = F\cos(\theta) \qquad V_{eq} = \frac{V}{\cos(\theta)} \; . \qquad \text{F,V}$$

The first step is to make two runs using a very short length at the top radius and at the bottom radius. The length should be short enough to prevent general cylinder buckling from being a driver. Taking the more conservative run will design the number of stringers, the stringer cross section, and the skin thickness. Then set the skin and stringer values as already designed and check that the general cylinder buckling constraint is met. Use an equivalent cylinder with:[4]

$$r_{eq} = \frac{r_{\text{small end}}}{\cos(\theta)} \qquad l_{eq} = \frac{h_{\text{cone}}}{\cos(\theta)} \; . \qquad \text{r,l}$$

If general cylinder buckling is not satisfied, change the ring locations and start again. It is best to analyze each segment of the cone between rings separately because an optimum cone will have unequal ring spacing to reduce shell buckling.[5]

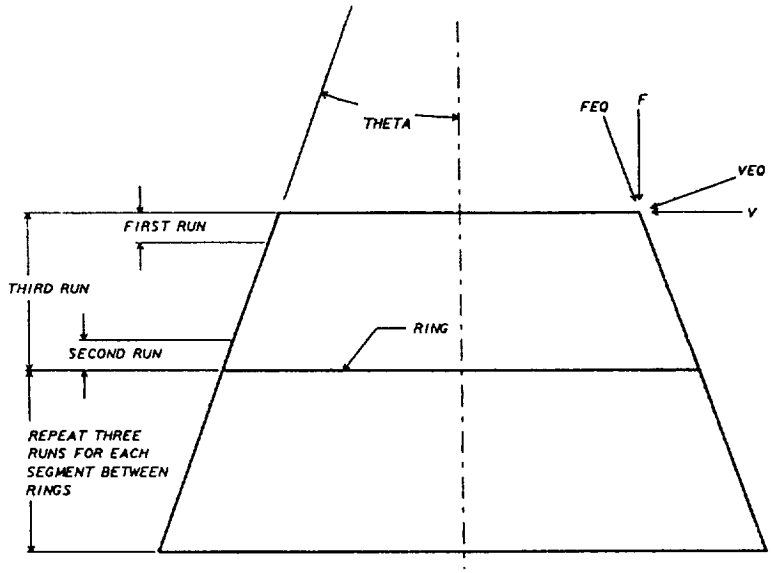


Figure 8. Conical analysis.

OVERALL CYLINDER CALCULATIONS

Variables:

A	cross-sectional area
F	applied axial force
F_{wt}	additional nonoptimized weight
Io	cylinder moment of inertia with no skin buckled
I_{st}	moment of inertia of a single stringer
l	length of cylinder
M	applied bending moment
N_{st}	number of stiffeners
N	Line Load
P	pressure
r	cylinder radius
sf	safety factor
t	skin thickness
Y	distance from center of cylinder
Z	neutral axis distance from the skin
ρ	material density
σ	stress
τ	shear stress

Subscripts:

1,2	principal stress directions
a	axial
cl	center line
cb	Coupled Buckling
cr	critical
$crip$	inelastic crippling
cy	compressive yield
gcb	general cylinder buckling
h	hoop
r	ring
sk	skin
slb	stringer local buckling
st	stringer
t	tensile

The moment of inertia of the entire cylinder is used in many places throughout the program. The initial reference Io is calculated with all of the skin effective and assumes that the stringers are evenly spaced:[6]

$$Io = \pi t r_{cl}^3 + N_{st} I_{st} + \sum A_{st} Y_{st}^2 \ , \qquad \text{Io}$$

$$Y_{st,i} = (r + Z_{st}) \cos\left(\frac{2\pi i}{N_{st}}\right) . \qquad \text{Yst}$$

The maximum applied tension stress is only used to make sure that the maximum stress of the material is not exceeded. This load is only tensile if the moment overcomes the axial compression. Because the input file requires the axial force to be positive for compression, tensile forces are negative. Assuming all the skin is effective:

$$\sigma_t = \frac{-M(sf)r}{Io} + \frac{(F - P_a \pi r^2)(sf)}{2\pi r t + A_{st} N_{st}} \ . \qquad \text{S}$$

The objective function for the optimization is the weight of the cylinder:

$$\text{Weight} = \left[2\pi r_{cl} t l + A_{st} l N_{st} + A_r N_r 2\pi(r + Z_r)\right] \rho + F_{wt} \ . \qquad \text{obj}$$

The limiting constraints of the optimization are:

Skin buckling:

$$\frac{\sigma_{sk}}{\sigma_{cr,sk}} + \left(\frac{\tau_{sk}}{\tau_{cr}}\right)^2 < 1 \ . \qquad \text{G[0]}$$

Shell buckling:

If the skin buckles, the general cylinder buckling equations are invalid, and this failure is controlled by stringer column buckling. If there is no skin buckling, then both column buckling and general cylinder buckling are calculated and the most favorable value is used. This reflects the ability of the stringers to carry load without the skin in some cases, and for the skin to transfer load across stringers in other cases.

Stringer column buckling:

$$\sigma_{st} < \sigma_{cr,st} \ . \qquad \text{G[1]}$$

General cylinder buckling:

$$\frac{N}{N_{gcb}} + \frac{-P_h(sf)}{P_{cr}} < 1 \ . \qquad \text{G[1]}$$

Allowable material stress:

$$\left[(\sigma_1 - \sigma_2)^2 + \sigma_2^2 + \sigma_1^2\right]^{1/2} < 2\sigma_{cy} \ . \qquad \text{G[2]}$$

For hat-stringers:

Inelastic stringer crippling:

$$\sigma_{crip} > \sigma_{st} \ . \qquad \text{G[3]}$$

or local elastic stringer buckling:

$$\sigma_{lb,\,flange} > \sigma_{st} \qquad \text{G[3]}$$

$$\sigma_{lb,\,web} > \sigma_{st} \ . \qquad \text{G[4]}$$

For I-Stringers:

I-coupled buckling:

$$\sigma_{cb} > \sigma_{st} \ , \qquad \text{G[4]}$$

and, inelastic stringer crippling:

$$\sigma_{crip} > \sigma_{st} \ , \qquad \text{G[3]}$$

or local elastic stringer buckling:

$$\sigma_{lb,\,web} > \sigma_{st} \ . \qquad \text{G[3]}$$

STRINGER PROPERTY CALCULATIONS

Variables:

A	area
al	torsional constant component
D	diameter of the maximum inscribed circle at the web/flange junction in an I-stringer
dy	distance of reference axis parallel to the skin
h	stringer height
h_{sl}	slant length of a hat-stringer web
I_{st}	stringer moment of inertia about axis parallel to the skin
Ix	moment of inertia about axis parallel to the skin
J_{st}	torsional constant of the stringer
K_1,K_2	torsional constant components
l_{sl}	projected slant width of a hat-stringer web
l_1	hat: stringer to skin flange length, I: bottom flange thickness
l_2	hat: top flange length, I: top flange thickness
t_{st}	hat: stringer thickness, I: web thickness
W	hat: top flange thickness, I: Top flange width
$\bar{y}$	neutral axis distance from reference line parallel to the skin
Z_{st}	stringer neutral axis distance from the skin center line
α	angle of the web of a hat-stringer

Subscripts:

0,1,2,3,4	individual stringer segments

CORSS computes the area, moment of inertia, torsional constant, and the neutral axis of a stringer. The slant width is also returned for hat-stringers. These returned properties are used throughout the program.

For a hat-stringer[8] (fig. 6):

$$h_{sl} = \frac{h}{\cos(\alpha)} \ , \qquad \text{ha}$$

$$l_{sl} = h_{sl} \sin(\alpha) \ , \qquad \text{la}$$

$$A_0 = l_1 t_{st} \qquad dy_0 = t_{st}/2$$
$$A_1 = h_{sl} t_{st} \qquad dy_1 = h/2$$
$$A_2 = l_2 W \qquad dy_2 = h - W/2$$
$$A_3 = A_1 \qquad dy_3 = dy_1$$
$$A_4 = A_0 \qquad dy_4 = dy_0$$

$$Ix_0 = Ix_4 = \frac{l_1 t_{st}^3}{12} \ , \qquad \text{Ix[0],Ix[4]}$$

$$Ix_1 = Ix_3 = \frac{t_{st} h_{sl}}{12} \left\{ h_{sl}^2 \cos^2(\alpha) + t_{st}^2 \sin^2(\alpha) \right\} \ , \qquad \text{Ix[1],Ix[3]}$$

$$Ix_2 = \frac{l_2 t_{st}^3}{12}\ , \qquad \text{Ix[2]}$$

$$J_{st} = \frac{4[(h+t/2-W/2)(l_2+l_a)]^2}{\left(\frac{l_2+2l_a}{t} + \frac{2h_a}{t_{st}} + \frac{l_2}{W}\right)}\ , \qquad \text{Js}$$

For an I-stringer[9] (fig. 6):

$A_0 = W\,l_1$ $\qquad dy_0 = l_1/2$
$A_1 = t_{st}(h-l_2-l_1)$ $\qquad dy_1 = (h-l_2-l_1)/2+l_1$
$A_2 = W\,l_2$ $\qquad dy_2 = h-l_2/2$

$$Ix_o = \frac{W\,l_1^3}{12}, \qquad \text{Ix[0]}$$

$$Ix_1 = \frac{t_{st}(h-l_2-l_1)^3}{12}\ , \qquad \text{Ix[1]}$$

$$Ix_2 = \frac{W\,l_2^3}{12}, \qquad \text{Ix[2]}$$

$$K1 = W\,l_2^3\left[\frac{1}{3} - .21\ \frac{l_2}{W}\left(1 - \frac{l_2^4}{12\ W^4}\right)\right] \qquad \text{K1}$$

$$K2 = (h-l_2)\ t_{st}^3\left[\frac{1}{3} - .105\frac{t_{st}}{(h-l_2)}\left(1 - \frac{t_{st}^4}{192(h-l_2)^4}\right)\right]\ . \qquad \text{K2}$$

If $l_2 < t_{st}$ $\qquad al = 0.15\,\frac{l_2}{t_{st}}$ else $al = 0.15\,\frac{t_{st}}{l_2}$. $\qquad$ al

If $t_{st} > 2l_2$ $\qquad D = t_{st}$ else $D = l_2 + \frac{t_{st}^2}{4l_2}$. $\qquad$ D

$$J_{st} = K_1 + K_2 + (al)\,D^4 \qquad \text{Js}$$

For both stringer types:

$$\bar{y} = \frac{\sum A\ dy}{\sum A}, \qquad \text{ybar}$$

$$I_x = \sum I_x + \sum (A\ d_y^2) - \bar{y} \sum (A\ d_y), \qquad \text{Ix}$$

$$I_{st} = I_x \qquad Z_{st} = \bar{y} + \frac{t}{2}. \qquad \text{Is, Zs}$$

SKIN BUCKLING STRESS CALCULATIONS

Variables:

A_{st}	stringer area
b_{pl}	distance between simply supported points on the skin
d	ring spacing
E	Young's modulus
F	applied axial force
Io	cylinder moment of inertia with no skin buckling
K, K_1	skin buckling pressure stabilization factors
K_c, K_s	skin buckling factors for compressive and shear loading, respectively
M	applied moment
N_{st}	number of stringers
P_a, P_h	applied axial and hoop pressures, respectively
q	shear flow
r	radius of cylinder
sfp	skin buckling safety factor
t	skin thickness
V	applied shear
Y	distance from neutral axis of cylinder
Z	nondimensional parameter for calculating K_c and K_s
Z_{st}	Stringer neutral axis distance from skin surface
α	skin buckling pressure stabilization factor
σ	stress
τ	shear stress
ν	Poisson's ratio

Subscripts:

i	sequential location index
cr	critical
sk	skin

The skin buckling constraint is a function of axial stress and shear stress:[7]

$$\frac{\sigma_{sk,i}}{\sigma_{cr}} + \left(\frac{\tau_{sk,i}}{\tau_{cr}}\right)^2 < 1 \ . \qquad \text{g0max}$$

The critical skin stress for an axial load is:

$$\sigma_{cr} = \frac{K_c \pi^2 E}{12(1-\nu^2)} \left(\frac{t}{b_{pl}}\right)^2 , \qquad \text{Scrpl}$$

$$Z = \frac{b_{pl}^2}{rt} \sqrt{(1-\nu^2)} \ . \qquad \text{Z}$$

VALUES OF K_c (r/t,Z)

r/t	$Z < 4$	$4 <= Z < 40$ ($y = Z-4$)	$Z >= 40$
100	4	$-0.000179569\, y^3 + 0.0154\, y^2 + 0.0089413\, y + 4$	$0.4\, Z$
300	4	$-0.000159922\, y^3 + 0.0140199\, y^2 + 0.00810034\, y + 4$	$0.375\, Z$
500	4	$-0.000127178\, y^3 + 0.011594\, y^2 + 0.00669873\, y + 4$	$Z/3$
1,000	4	$-0.00100982\, y^3 + 0.00965326\, y^2 + 0.00557741\, y + 4$	$0.3\, Z$
1,500	4	$-6.1688\text{E-}5\, y^3 + 0.00674219\, y^2 + 0.00389547\, y + 4$	$0.25\, Z$

The equations for K_c for Z between 4 and 40 are cubic spline curve fits to plotted curves.[7] For $K_c > 40$ the curves are straight lines on a log-log plot. K_c is linearly interpolated between the appropriate curves with *r/t* values less than 100 being treated as $r/t = 100$, and *r/t* values greater than 1,500 being treated as $r/t = 1{,}500$.

Internal pressure has a stabilizing effect on cylinders and raises the critical skin stress:[8]

$$\alpha = \frac{P_h}{E} \left(\frac{r}{t}\right)^2 , \qquad \text{alpha}$$

$$K = 9\left(\frac{t}{r}\right)^{0.6} \frac{1+0.21\,\alpha\left(\frac{r}{t}\right)^{0.6}}{1+3\alpha}\,, \qquad \text{K}$$

$$K_1 = 0.16\,\frac{r}{t}\left(\frac{t}{d}\right)^{1.3}\,, \qquad \text{K1}$$

$$\sigma_{cr} = \sigma_{cr,P_h=0} + (K+K_1)E\,\frac{t}{r}\,. \qquad \text{Scrpl}$$

The critical shear stress is:[7]

$$\tau_{cr} = \frac{K_s \pi^2 E}{12(1-\nu^2)}\left(\frac{t}{b_{pl}}\right)^2\,, \qquad \text{taucr}$$

$$K_s = \begin{array}{lll} 4.94002871 & + 2.83295655E{-}1\ Z & - 8.48571232E{-}3\ Z^2 \\ + 2.96028833E{-}4\ Z^3 & - 5.59394768E{-}6\ Z^4 & + 5.12790432E{-}8\ Z^5 \\ - 1.79230370E{-}10\ Z^6 & & \end{array}$$

The equation for K_s is for an infinitely long plate and will be conservative.

Axial stress in the skin is:

$$\sigma_{sk,i} = \frac{M(sfp)Y_{sk,i}}{Io} + \frac{(F - P_a\pi r^2)(sfp)}{2\pi rt + A_{st}N_{st}}\,. \qquad \text{Ssk[i]}$$

Internal pressure is not multiplied by the safety factor.

Shear stress in the skin is:

$$q_0 = \frac{-V(sfp)}{I_o}\frac{A_{st}}{2}(r + z_{st})$$

$$q_i = q_{i-1} - \frac{V(sfp)A_{st}Y_{st,i}}{Io}\,, \qquad \text{q[i]}$$

$$\tau_{sk,i} = \frac{q_i}{t}\,. \qquad \text{tau}$$

Each section of skin between stringers is checked, with the highest combination of stress and shear being used in the design.

The buckling load drivers are also retained to trigger "effective skin" calculations later in the program if necessary.

STRINGER CRIPPLING CALCULATIONS

Variables:

A_b	flange area
b	length of web
E	Young's modulus
l	cylinder length
l_2	I-stringer top flange thickness, hat-stringer top flange length
m	number of buckled waves
t_{st}	I-stringer web thickness
W	hat-stringer top flange thickness, I-stringer width
σ	stress
ν	Poisson's ratio

Subscripts:

crip	inelastic crippling
cy	compressive yield
lb	local stringer buckling
st	stringer
x	axial

For hat-stringers, if local elastic buckling is allowed, a single crippling constraint is placed on the stringer. If elastic buckling is not allowed two constraints are applied, buckling of the web and buckling of the top flange.

The equation for crippling of a plate simply supported on both sides is (curve fit from graph of reference 11):

$$\sigma_{crip} = \frac{1.387194}{\left(\sqrt{\frac{\sigma_{cy}}{E}\frac{(l_2 - t_{st})}{W}}\right)^{.807179}}\ \sigma_{cy} \qquad \text{S1}$$

This value is cut off at a maximum value of the material tensile ultimate stress. The crippling of the individual segments is averaged to give the stringer crippling stress:

$$\sigma_{st,crip} = \frac{\Sigma(\sigma_{\text{flange},crip} A_{\text{flange}})}{\Sigma A_{\text{flange}}}\ , \qquad \text{Scrips}$$

This is calculated even if elastic buckling is not allowed in case it is needed for Johnson-Euler buckling of the stringer column.

If local elastic buckling is not allowed, the buckling stress of the flange and web are checked. For a plate simply supported on both sides, the elastic buckling stress for the top flange is:[10]

$$\sigma_{lb} = \frac{4\pi^2 E}{12(1-\nu)}\left(\frac{W}{(l_2 - t_{st})}\right)^2 \qquad \text{S2}$$

I-Stringers have a coupled buckling constraint and either an elastic buckling or a crippling constraint. The crippling stress of half of the flange is that of a plate simply supported on one edge and free on the other (curve fit from graph of reference 11):

$$\sigma_{crip} = \frac{.569311}{\left(\sqrt{\frac{\sigma_{cy}}{E}}\frac{W/2}{l_2}\right)^{.812712}}\ \sigma_{cy}$$

This value is cut off at the material tensile ultimate strength. This is applied to the flange while the web crippling is calculated from the equation of a plate simply supported on both sides. These are averaged into the stringer crippling stress in the same manner as for a hat-stringer.

Elastic buckling is applied only to the cap of the I-stringer The stringer cap is treated as two plates; each half the cap width and simply supported on one side (at the web) and free on the other.

The coupled buckling stress is calculated for checking elastic buckling of the I-stringer web. This failure mode is described in appendix A. The governing equation is:

$$\left(\left[\alpha^2 - \nu\left(\frac{m\pi}{l}\right)^2\right]\sinh(\alpha b)\right)\left(EI\sin(\beta b)\left(\frac{m\pi}{l}\right)^4 + D\left\{\left[\beta^3 + \beta(2-\nu)\left(\frac{m\pi}{l}\right)^2\right]\cos(\beta b)\right\} - A_b\sigma_x \sin(\beta b)\left(\frac{m\pi}{l}\right)^2\right)$$

$$+\left(EI\sinh(\alpha b)\left(\frac{m\pi}{l}\right)^4 - D\left\{\left[\alpha^3 - \alpha(2-\nu)\left(\frac{m\pi}{l}\right)^2\right]\cosh(\alpha b)\right\} - A_b\sigma_x \sin(\alpha b)\left(\frac{m\pi}{l}\right)^2\right)\left(\left[\beta^2 + \nu\left(\frac{m\pi}{l}\right)^2\right]\sin(\beta b)\right) = 0,$$

where

$$\alpha = \sqrt{\left(\frac{m\pi}{l}\right)^2 + \sqrt{\frac{N_x}{D}\left(\frac{m\pi}{l}\right)^2}} \qquad \beta = \sqrt{-\left(\frac{m\pi}{l}\right)^2 + \sqrt{\frac{N_x}{D}\left(\frac{m\pi}{l}\right)^2}} \qquad \text{S2}$$

$$D = \frac{E\,t_{st}^3}{12(1-\nu^2)} \qquad A_b = l_2 w \qquad I = \frac{l_2 w^3}{12} \qquad \sigma_x = \frac{N_x}{t_{st}}\ .$$

This equation is solved iteratively using the secant method. Several trials of the equation are used to insure that the lowest root is bounded by the starting values. The buckling stress is a function of the number of waves in the buckled shape. This is usually 1, but not always. An mflag of 0 will cause the program to start by calculating the stress for 1 wave, and increase the number of waves until the buckling stress increases. An mflag of a number will cause the program to start at 1 wave, and increase to the given value. This could cause problems during an optimization because high numbers will add a significant amount of run-time. Also, if the buckling stresses are too high, overflow errors may occur.

COLUMN BUCKLING CALCULATIONS[6]

Variables:

A	area
b	stringer spacing
b_e	distance between fixed points on the skin
d	ring spacing
E	Young's modulus
F	applied positive compressive axial force
I	moment of inertia (if no subscript, I of entire cylinder with buckled skin)
Io	cylinder moment of inertia with no buckled skin
M	applied moment
P_a, P_h	applied positive outward axial and hoop pressure, respectively
r	cylinder radius
sf	safety factor
t	skin thickness
W	effective skin width
Y	distance from center of cylinder
Z	neutral axis distance
$\bar{y}$	cylinder neutral axis distance from the center
ρ_g	stringer radius of gyration
σ	stress

Subscripts:

cr	critical
$crip$	crippling
cyl	total cylinder
flange	single stringer flange
i	sequential location index
lb	buckled skin
r	ring
sk	skin
st	stringer

The program is written with the ability to apply a different safety factor to skin buckling than to other forms of failure, with the assumption being that some cases will allow the skin to buckle as long as the ultimate load can still be carried. This is the case for the first design cycle of the NLS for which this program was originally written. The skin cannot buckle at a safety factor of 1.0, but can at a safety factor of 1.4, which is the required factor on all other forms of failure.

The amount of buckled skin is dependent on the stress of the stringer, and an iterative process is needed to remove the "ineffective" skin area. The "effective" skin is assumed to be prevented from buckling by the stringers and to carry the fully developed stringer stress. First, the stress is calculated from an assumed cylinder moment of inertia, then the ineffective skin around each stringer is removed, a new cylinder moment of inertia is calculated, and the neutral axis location is moved. The column stress is recalculated with the corrected I and neutral axis. This repeats until a value of I is converged upon, which is usually after only a couple of iterations. The program allows no more than 20 iterations before defaulting to an I of $Io/2$ and printing a warning.

If the skin buckles, it is assumed to carry 90 percent of the critical skin buckling load:

$$\sigma_{lb} = 0.9\ \sigma_{sk,cr} \ . \qquad \text{Slb}$$

The stress is calculated at each stringer, and at the center of each skin panel. For the initial iteration, $\bar{y} = 0$, $I = Io/2$, and $\Sigma A_i = 0$:

$$\sigma_{st,i} = \frac{(F - P_a \pi r^2)(sf)}{A_{cyl} - \Sigma A_i} + \frac{M(sf)(Y_{st,i} + \bar{y})}{I} \ , \qquad \text{Sst[i]}$$

$$\sigma_{sk,i} = \frac{(F - P_a \pi r^2)(sf)}{A_{cyl} - \Sigma A_i} + \frac{M(sf)(Y_{sk,i} + \bar{y})}{I} \ . \qquad \text{Ssk[i]}$$

If P_a is positive, it is assumed to be relieving the stress and is not multiplied by the safety factor.

Effective width for each stringer is calculated from stringer stress:

$$W_i = 0.85\ t \sqrt{\frac{E}{\sigma_{st,i}}} \ . \qquad \text{We[i]}$$

This width extends from a fixed point on the skin, a rivet line for a hat-stringer or the web for an I-stringer, in one direction. Total effective width about a fixed point is 2 W_i.

The effective width is checked against the width between fixed points on the skin. It is assumed that the stress in the skin does not increase after buckling. Since some skin is left in to account for the load that the buckled skin does carry, the ineffective area is:

$$A_i = \frac{t(\sigma_i - \sigma_{lb})}{\sigma_i} \text{[(total width) – (effective width)]} \ . \qquad \text{A}$$

With the ineffective area known, $\bar{y}$ and I can be recalculated:

$$\bar{y} = \frac{\Sigma A_i\, y_i}{(A_{cyl} - \Sigma A_i)} \qquad \text{ybar}$$

$$I = I_o - \Sigma(A_i\, y_i^2) - (A_{cyl} - \Sigma A_i)\bar{y}^2 \ . \qquad \text{I}$$

The new $\bar{y}$ and I are put back into the stress calculations, and the operation is repeated until new values for I and the applied column stress are converged upon.

The total effective skin width of the most highly stressed stringer, b_e, is added to the moment of inertia of the stringer to be used in calculating the buckling load of the total column:

$$I_{sk+st} = \frac{b_e t^3}{12} + I_{st} + A_{st} Z_{st}^2 - \frac{(A_{st} Z_{st})^2}{A_{st} + b_e t} \quad . \qquad \text{Ise}$$

The radius of gyration is:

$$\rho_g = \sqrt{\frac{I_{sk+st}}{A_{st} + 2W_0 t}} \quad . \qquad \text{radg}$$

For long columns, the Euler critical column buckling, simply supported with an effective length of d, is:

$$\sigma_{cr,sk+st} = \left(\frac{\pi \rho_g}{d}\right)^2 E \quad . \qquad \text{Scrstcol}$$

For short columns, the Johnson-Euler buckling equation is used. The weighted average of the inelastic crippling stress is used as the stringer crippling stress:[7]

$$\text{For } \frac{d}{\rho_g} < \pi \sqrt{\frac{2E}{\sigma_{st,crip}}} \; : \; \sigma_{cr} = \sigma_{st,crip} - \frac{\sigma_{st,crip}^2}{4\pi^2 E} \frac{d^2}{\rho_g^2} \quad . \qquad \text{Scrstcol}$$

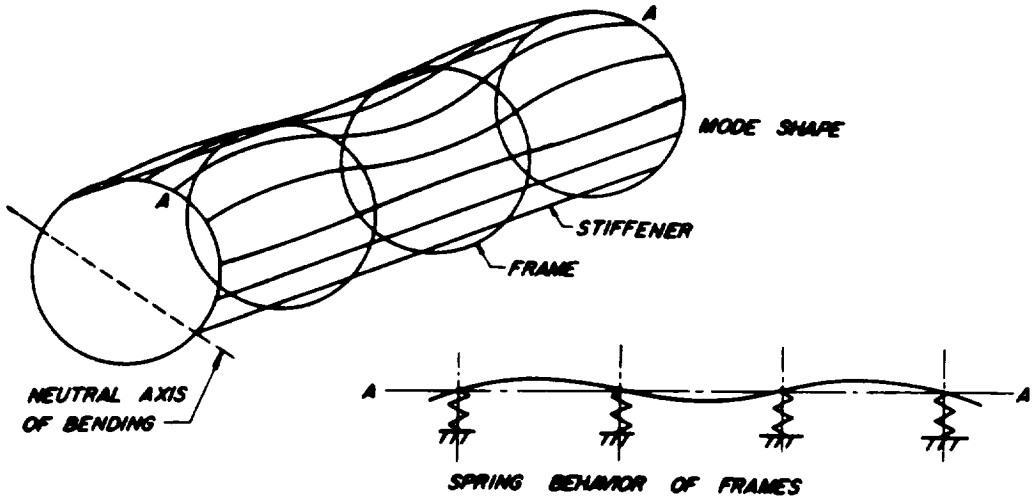
The assumption is that the rings are stiff enough to enforce the panel buckling mode shape (fig. 9a). The required ring stiffness is:[10, 11] (See appendix A, section 2)

$$I_r = \frac{0.172 \, \sigma_{cr,sk+st} (A_{st} + b\, t) r^3}{d\, E} \quad . \qquad \text{EI}$$

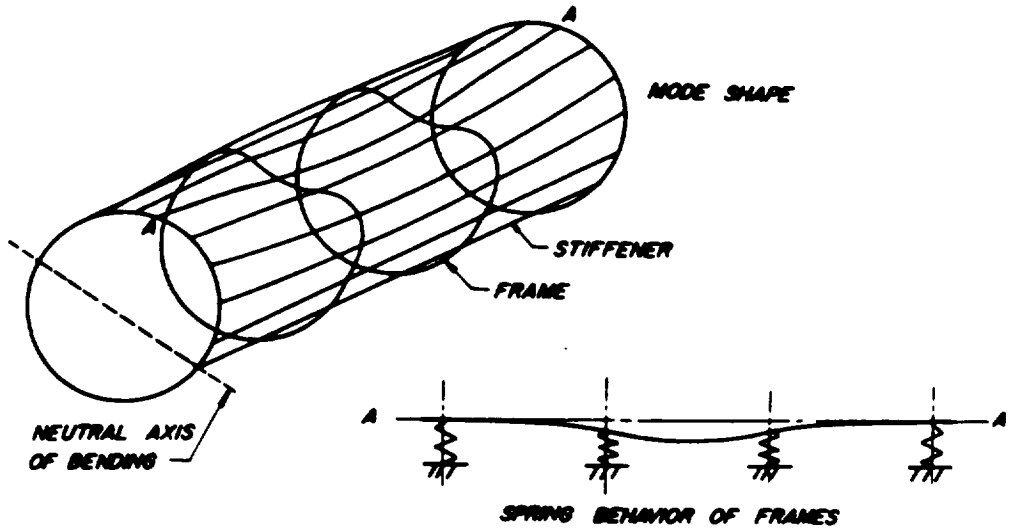
The required ring area is:[10] (See appendix A, section 2)

$$A_r = \frac{4\pi^2 I_{sk+st} r (r - Z_r)}{b d^3} \quad . \qquad \text{Ar}$$

The required stiffness and area are printed, but are not used in any calculations.



(a) Panel instability.



(b) General instability.

Figure 9. Mode shapes for panel and general instability of stiffened cylinders in bending.[7]

GENERAL CYLINDER BUCKLING CALCULATIONS[12]

Variables:

$A_{(1,2,3),(1,2,3)}$	buckling matrix
A_{new}	cylinder area with ineffective skin removed
A_r, A_{st}	ring and stringer areas, respectively
b	stringer spacing
Cx, Cy, Cxy	coupling constants for smeared orthotropic shells
d	ring spacing
Dx, Dy, Dxy	bending stiffnesses for smeared orthotropic shells
E	Young's modulus
Ex, Ey, Exy	extensional stiffnesses for smeared orthotropic shells
F	applied axial force
G	material shear modulus
Gxy	shear stiffness for smeared orthotropic shells
I_r, I_{st}	ring and stringer moments of inertia
J_r, J_{st}	ring and stringer polar moments, respectively
Kxy	coupling constant for smeared orthotropic shells
l	cylinder length
M	applied moment
m	number of axial buckling half waves
n	number of circumferential buckling waves
N, N_F, N_M	total line load, line load from axial forces, line load from bending moment, respectively
N_{cr}	critical line load
P_a, P_h	applied axial and hoop pressure, respectively
P_{cr}	critical hoop pressure
r	cylinder radius
sf	safety factor
t	skin thickness
$\bar{y}$	cylinder neutral axis distance from centroid
Z_r, Z_{st}	ring and stringer neutral axis distance from skin, respectively
ϕ	factor for calculating line load knockdown
γ_F, γ_M	axial and bending knockdowns, respectively
ν	Poisson's ratio

The cylinder buckling limit is a function of axial line load and crushing pressure, and is:

$$\frac{N_M}{N_{cr}\gamma_m} + \frac{N_F}{N_{cr}\gamma_F} + \frac{-P_h(sf)}{P_{cr}} < 1 \ . \qquad \text{G[1]}$$

Internal pressure is considered only to lower the applied compression stress, and no other pressure stabilization is considered.

If the critical line load is at a higher stress than the compressive yield stress of the material, then a warning is printed that if loading increases above limit load, plasticity effects can lower the capability of the shell.

The stringer and ring properties are smeared over the cylinder to provide properties of an equivalent orthotropic cylinder. Since the assumption to make this valid is that the stringers and rings are closely spaced, it is best to model segments between widely spaced rings as individual cylinders.

The smeared, orthotropic properties are:

$$Ex = \frac{Et}{1-\nu^2} + E\frac{A_{st}}{b}\ , \qquad \text{Ex}$$

$$Ey = \frac{Et}{1-\nu^2} + E\frac{A_r}{d}\ , \qquad \text{Ey}$$

$$Exy = \frac{\nu Et}{1-\nu^2}\ , \qquad \text{Exy}$$

$$Gxy = \frac{Et}{2(1-\nu)}\ , \qquad \text{Gxy}$$

$$Dx = \frac{Et^3}{12(1-\nu^2)} + E\frac{I_{st}}{b} + Z_{st}^2 E\frac{A_{st}}{b}\ , \qquad \text{Dx}$$

$$Dy = \frac{Et^3}{12(1-\nu^2)} + E\frac{I_r}{d} + Z_r^2 E\frac{A_r}{d}\ , \qquad \text{Dy}$$

$$Dxy = \frac{Et^3}{6(1+\nu)} + G\frac{J_{st}}{b} + G\frac{J_r}{d}\ , \qquad \text{Dxy}$$

$$Cx = Z_{st} E\frac{A_{st}}{b}\ , \qquad \text{Cx}$$

$$Cy = Z_r E\frac{A_r}{d}\ , \qquad \text{Cy}$$

$$Cxy = Kxy = \frac{t^3}{12(1-\nu^2)}\ . \qquad \text{Cxy,Kxy}$$

These properties are built into a two-by-two and a three-by-three matrix. The values m and n are increased until the lowest critical line load is found. For critical pressure, m is set to 1 and n is increased until the lowest critical pressure is found.

$$A_{1,1} = Ex\left(\frac{m\pi}{l}\right)^2 + Gxy\left(\frac{n}{r}\right)^2 , \qquad \text{A11}$$

$$A_{2,2} = Gxy\left(\frac{m\pi}{l}\right)^2 + Ey\left(\frac{n}{r}\right)^2 , \qquad \text{A22}$$

$$A_{3,3} = Dx\left(\frac{m\pi}{l}\right)^4 + Dxy\left(\frac{m\pi n}{rl}\right)^2 + Dy\frac{n^4}{r^4} + \frac{Ey}{r^2} + 2\frac{Cy}{r}\frac{n^2}{r^2} + 2\frac{Cxy}{r}\left(\frac{m\pi}{l}\right)^2 , \qquad \text{A33}$$

$$A_{1,2} = A_{2,1} = (Exy+Gxy)\frac{m\pi}{l}\frac{n}{r} , \qquad \text{A12,A21}$$

$$A_{2,3} = A_{3,2} = (Cxy+2Kxy)\left(\frac{m\pi}{l}\right)^2\frac{n}{r} + Ey\frac{n}{r^2} + Cy\left(\frac{n}{r}\right)^3 , \qquad \text{A23,A32}$$

$$A_{3,1} = A_{1,3} = \frac{m\pi Exy}{rl} + Cx\left(\frac{m\pi}{l}\right)^3 + (Cxy+2Kxy)\frac{m\pi}{l}\frac{n^2}{r^2} . \qquad \text{A31,A13}$$

The critical line load is calculated by:

$$N_{cr} = \frac{l^2}{(m\pi)^2}\,\frac{\det\begin{pmatrix} A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,1} & A_{3,2} & A_{3,3} \end{pmatrix}}{\det\begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix}} . \qquad \text{Nx}$$

The equations for the knockdown factors are:

$$\gamma_F = 1-0.901(1-e^{-\phi}) \qquad \gamma_M = 1-0.731(1-e^{-\phi}) , \qquad \text{gamF,gamM}$$

$$\phi = \frac{\left(\frac{r}{\sqrt[4]{\frac{D_x}{E_x}\frac{D_y}{E_y}}}\right)^{1/2}}{29.8} . \qquad \text{Phi}$$

The applied line load, N, is calculated from:

$$N_M = \frac{M(sf)(r+\bar{y})}{I}\left(\frac{A_{st}}{b}+t\right) \qquad N_F = \frac{(F-P_a\pi r^2)(sf)}{A_{new}}\left(\frac{A_{st}}{b}+t\right) . \qquad \text{NM, NF}$$

If P_a is positive, it is not multiplied by the safety factor.

The critical crushing pressure is:

$$P_{cr} = 0.75 \frac{r}{n^2} \frac{\det\begin{pmatrix} A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,1} & A_{3,2} & A_{3,3} \end{pmatrix}}{\det\begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix}} . \qquad \text{Sa}$$

VON MISES STRESS CHECK

Anew	cylinder cross sectional area with ineffective skin removed
P	pressure
r	cylinder radius
sf	safety factor
t	skin thickness
σ	stress
τ	shear stress

Subscripts

a	axial
cy	compressive yield
h	hoop
max	maximum
sk	skin
st	stringer
t	tensile
x	axial direction
y	hoop direction
o	at the first (most highly stressed) stringer location
1	first principal stress direction
2	second principal stress direction

The applied Von Mises, or Maximum Distortion Energy, stress in the cylinder is checked against the allowable material stress. Three points are checked: the point of maximum compression, the point of maximum shear, and the point of maximum tension.

The point of maximum compression is where the bending moment and compressive axial force are working together, and the shear stress is a minimum:

$$\sigma_x = \sigma_{st} \qquad \sigma_y = \frac{-P_h r(SF)}{t} \qquad \tau_{xy} = \tau_{sk,o} . \qquad \text{Sx, Sy, tauxy}$$

The point of maximum shear is the point where the moment does not affect the stress:

$$\sigma_x = \frac{(F - P_a \pi r^2)(sf)}{A_{new}} \qquad \sigma_y = \frac{-P_h r\,(SF)}{t} \qquad \tau_{xy} = \tau_{max} \; . \; . \qquad \text{Sx, Sy, tauxy}$$

The point of maximum tension is where the moment creates a tension load:

$$\sigma_x = \sigma_t \qquad \sigma_y = \frac{-P_h r(SF)}{t} \qquad \tau_{xy} = \tau_{sk,o} \; . \qquad \text{Sx, Sy, tauxy}$$

The principal stresses are calculated:

$$\sigma_1, \sigma_2 = \frac{\sigma_x + \sigma_y}{2} \pm \sqrt{\left(\frac{\sigma_x - \sigma_y}{2}\right)^2 + \tau_{xy}^2} \; . \qquad \text{S1, S2}$$

The Von Mises stress is calculated from the principal stresses and compared to the compressive yield stress:

$$\sqrt{\frac{(\sigma_1 - \sigma_2)^2 + \sigma_2^2 + \sigma_1^2}{2}} < \sigma_{cy} \; . \qquad \text{VMS}$$

SUMMARY

CORSS provides a state-of-the-art preliminary design tool, linking the analysis of a skin stringer construction with a numerical optimizer. It allows a large variety of different designs to be evaluated quickly, and easily allows limits, such as from manufacturing facilities, to be included. All of the major failure modes are checked.

REFERENCES

1. Vanderplaats, G., and Hansen, S.: "Design Optimization Tool." VMA Engineering, 1985–1988.

2. Paz, M., and Vanderplaats, G.: "Numerical Optimization Methods and Applications Short Course Notes." University of Alabama, 1990.

3. Vanderplaats, G.: "Numerical Optimization Techniques for Engineering Design: With Applications." McGraw-Hill Book Company, 1984.

4. Baker, E.H., Kovalevsky, L., and Rish, F.L.: "Structural Analysis of Shells." McGraw-Hill Book Company, 1972.

5. Baruch, M., Singer, J., and Harari, O.: "General Instability of Conical Shells With Nonuniformly Spaced Stiffeners Under Hydrostatic Pressure." Technion, Israel Institute of Technology, Department of Aeronautical Engineering, through the European Office of Aerospace Research, United States Air Force AF EOAR 63-58, TAE Report No. 37, 1964.

6. Niles, A.S., and Newell, J.S.: "Airplane Structures." Third Edition, John Wiley and Sons, Inc., 1943.

7. Bruhn, E.F.: "Analysis and Design of Flight Vehicle Structures." Jacobs Publishing, Inc., 1973.

8. Koelle, H.H.: "Handbook of Astronautical Engineering." McGraw-Hill Book Company, 1961.

9. Young, W.C.: "Roark's Formulas for Stress and Strain." Sixth Edition, McGraw-Hill Book Company, 1989.

10. Timoshenko, S.: "Theory of Elastic Stability." Engineering Sciences Monograph, McGraw-Hill Book Company, 1936.

11. "Astronautic Structures Manual," NASA TMX-73305, NASA/MSFC, 1975.

12. "Buckling of Thin-Walled Circular Cylinders," NASA SP-8007, NASA Space Vehicle Design Criteria (Structures), 1968.

13. Bazant, Z.P., and Cedolin, L.: "Stability of Structures," Oxford University Press, Inc., 1991.

APPENDIX A

A1. I-Coupled Buckling Equation Derivation

The method used to solve for the lateral stringer cap buckling load is taken from reference 10, section 9.4, pages 360-370. The stringer is idealized as shown in figure A1 below. The stringer web is modeled as a plate with lateral support at edges $y = 0$ and $y = b$. At edge $y = 0$, the web is rigidly supported by the skin, and at edge $y = b$ the web is elastically supported by the stringer cap. In this analysis, the torsional stiffnesses of the cap and the skin are conservatively neglected, so the edges at $y = 0$ and y = b are both modeled as hinged.

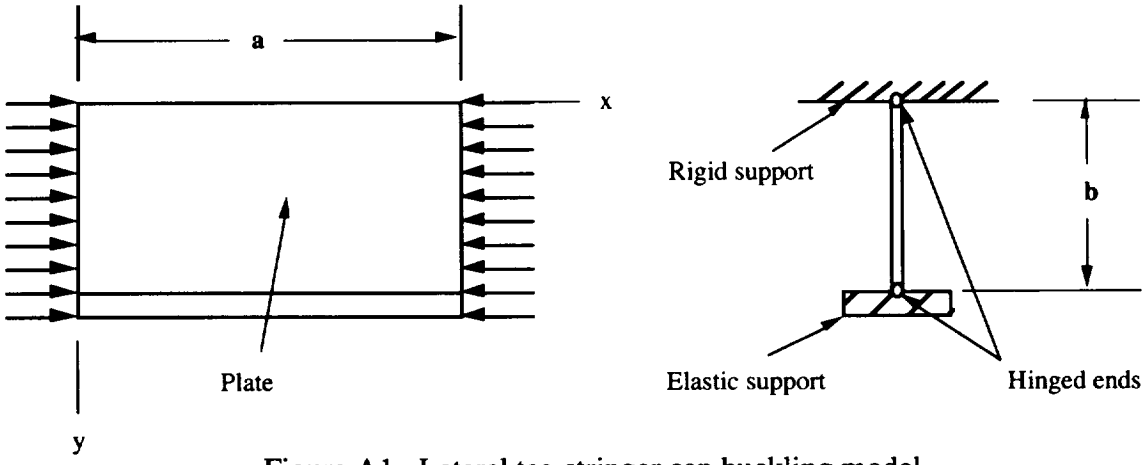


Figure A1. Lateral tee-stringer cap buckling model.

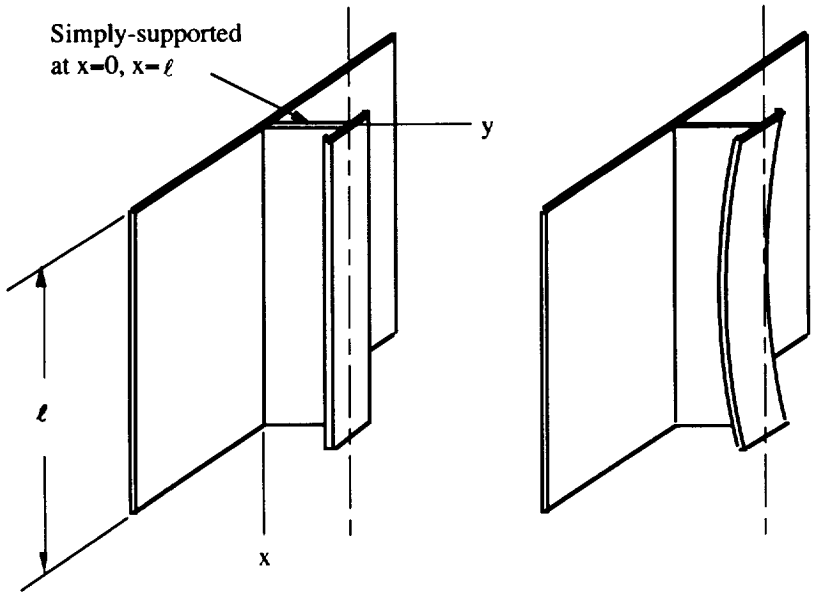


Figure A2. Lateral tee-stringer cap buckling.

The stringer cap is assumed to move laterally as depicted in figure A2. This motion is resisted by the cap in bending. In the limit as the cap becomes infinitely stiff, the result will approach that for a web simply supported along both sides $y = 0$ and $y = b$. The stringer dimensions are shown in figure A3.

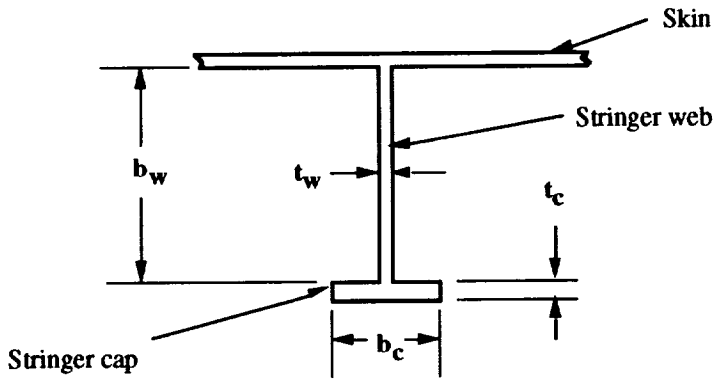


Figure A3. Stringer dimensions.

The governing differential equation for the lateral deflection of the web is

$$\frac{\partial^4 w}{\partial x^4} + 2\frac{\partial^4 w}{\partial x^2 \partial y^2} + \frac{\partial^4 w}{\partial y^4} = -\frac{N_x}{D}\frac{\partial^2 w}{\partial x^2} \quad .$$

The boundary conditions at $x = 0$ and $x = a$ are automatically satisfied by setting

$$w = f(y)\sum_{m=1}^{\infty} q_m \sin\frac{m\pi x}{a} \quad .$$

Substituting into the governing differential equation, we find that for a non-trivial solution $w(x,y)$, the following equation must be satisfied:

$$\frac{d^4 f}{dy^4} - 2\left(\frac{m\pi}{a}\right)^2 \frac{d^2 f}{dy^2} + \left[\left(\frac{m\pi}{a}\right)^4 - \frac{N_x}{D}\left(\frac{m\pi}{a}\right)^2\right] f = 0 \quad .$$

This equation can be easily solved when N_x is a constant (see reference 10, page 361, or reference 13, page 434). Then, assuming

$$\frac{N_x}{D} > \left(\frac{m\pi}{a}\right)^2 \quad ,$$

the general solution is

$$f(y) = A\sinh(\alpha y) + B\sin(\beta y) + C\cosh(\alpha y) + D\cos(\beta y) \quad ,$$

in which

$$\alpha = \sqrt{\left(\frac{m\pi}{a}\right)^2 + \sqrt{\frac{N_x}{D}\left(\frac{m\pi}{a}\right)^2}} \qquad \beta = \sqrt{-\left(\frac{m\pi}{a}\right)^2 + \sqrt{\frac{N_x}{D}\left(\frac{m\pi}{a}\right)^2}} \ .$$

Now, if the side $y = 0$ is simply supported and the side $y = b$ is supported by an elastic beam, we have the following boundary conditions:

$$\text{for y = 0:} \quad w = 0 \qquad \text{(i)}$$

$$\frac{\partial^2 w}{\partial y^2} = 0 \qquad \text{(ii)}$$

$$\text{for y = b:} \quad \frac{\partial^2 w}{\partial y^2} + \nu \frac{\partial^2 w}{\partial x^2} = 0 \quad \text{(iii)}$$

To obtain a second boundary condition for the side $y = b$, bending of the supporting beam must be considered. We make the assumptions that the beam is simply-supported on both ends, that it has the same modulus of elasticity as the plate, and that it is compressed together with the plate so that the compressive force on the beam is equal to $A_b\sigma_x$ where A_b is the cross-sectional area of the beam. Denoting by EI the flexural rigidity of the beam, the differential equation of its deflection curve is

$$EI\frac{\partial^4 w}{\partial x^4} = q - A_b \sigma_x \frac{\partial^2 w}{\partial x^2} \ ,$$

where q is the intensity of the load transmitted from the plate to the beam. From expressions for shearing forces (see Timoshenko and Gere, p. 330), this intensity is:

$$q = D\left[\frac{\partial^3 w}{\partial y^3} + (2-\nu)\frac{\partial^3 w}{\partial x^2 \partial y}\right] .$$

Substituting this value into the deflection curve yields the final boundary condition for $y = b$:

$$EI\frac{\partial^4 w}{\partial x^4} = D\left[\frac{\partial^3 w}{\partial y^3} + (2-\nu)\frac{\partial^3 w}{\partial x^2 \partial y}\right] - A_b \sigma_x \frac{\partial^2 w}{\partial x^2} \qquad \text{(iv)} \ .$$

Conditions (i) and (ii) require that $C = D = 0$. Now, we can write

$$f(y) = A\sinh(\alpha y) + B\sin(\beta y) \ .$$

The remaining boundary conditions can be used to obtain two algebraic equations for A and B.

From (iii):

$$\left(\frac{d^2 f}{dy^2}\sum_{m=1}^{\infty} q_m \sin\left(\frac{m\pi x}{a}\right) - \nu f(y)\left(\frac{m\pi}{a}\right)^2 \sum_{m=1}^{\infty} q_m \sin\left(\frac{m\pi x}{a}\right)\right)\Bigg|_{y=b} = 0 \ .$$

For this series to sum to zero for arbitrary x and y, each term must vanish and we can write

$$\left(\frac{d^2 f}{dy^2} - \nu f(y)\left(\frac{m\pi}{a}\right)^2\right)\Bigg|_{y=b} = 0 \quad .$$

Substituting for $f(y)$ and simplifying yields the first equation for A and B

$$A\left[\alpha^2 - \nu\left(\frac{m\pi}{a}\right)^2\right]\sinh(\alpha b) - B\left[\beta^2 + \nu\left(\frac{m\pi}{a}\right)^2\right]\sin(\beta b) = 0 \quad . \tag{1}$$

Find derivatives for (iv):

$$\frac{\partial^4 w}{\partial x^4} = f(y)\left(\frac{m\pi}{a}\right)^4 \sum_{m=1}^{\infty} q_m \sin\left(\frac{m\pi x}{a}\right)$$

$$\left.\frac{\partial^4 w}{\partial x^4}\right|_{y=b} = (A\ \sinh(\alpha b) + B\ \sin(\beta b))\left(\frac{m\pi}{a}\right)^4 \sum_{m=1}^{\infty} q_m \sin\left(\frac{m\pi x}{a}\right)$$

$$\left.\frac{\partial^3 w}{\partial x^3}\right|_{y=b} = (A\alpha^3 \cosh(\alpha b) + B\beta^3 \cos(\beta b))\sum_{m=1}^{\infty} q_m \sin\left(\frac{m\pi x}{a}\right)$$

$$\frac{\partial^2 w}{\partial x^2} = -f(y)\left(\frac{m\pi}{a}\right)^2 \sum_{m=1}^{\infty} q_m \sin\left(\frac{m\pi x}{a}\right)$$

$$\left.\frac{\partial^2 w}{\partial x^2}\right|_{y=b} = (-A\ \sinh(\alpha b) - B\ \sin(\beta b))\left(\frac{m\pi}{a}\right)^2 \sum_{m=1}^{\infty} q_m \sin\left(\frac{m\pi x}{a}\right)$$

$$\frac{\partial^3 w}{\partial x^2 \partial y} = -\frac{df}{dy}\left(\frac{m\pi}{a}\right)^2 \sum_{m=1}^{\infty} q_m \sin\left(\frac{m\pi x}{a}\right)$$

$$\left.\frac{\partial^3 w}{\partial x^2 \partial y}\right|_{y=b} = (-A\alpha\ \cosh(\alpha b) - B\beta\ \cos(\beta b))\left(\frac{m\pi}{a}\right)^2 \sum_{m=1}^{\infty} q_m \sin\left(\frac{m\pi x}{a}\right)$$

Again, we eliminate the factor $\sum_{m=1}^{\infty} q_m \sin\frac{m\pi x}{a}$ by considering equation (iv) term by term

$$EI(A\ \sinh(\alpha b) + B\ \sin(\beta b))\left(\frac{m\pi}{a}\right)^4$$

$$- D\left[\left\{\left[\alpha^3 - \alpha(2-\nu)\left(\frac{m\pi}{a}\right)^2\right]\cosh(\alpha b)\right\}A - \left\{\left[\beta^3 + \beta(2-\nu)\left(\frac{m\pi}{a}\right)^2\right]\cos(\beta b)\right\}B\right]$$

$$+ A_b\ \sigma_x[-A\ \sinh(\alpha b) - B\ \sin(\beta b)]\left(\frac{m\pi}{a}\right)^2 = 0$$

Collecting terms in A and B, we have the second equation

$$A\left[EI\sinh(\alpha b)\left(\frac{m\pi}{a}\right)^4 - D\left\{\left[\alpha^3 - \alpha(2-\nu)\left(\frac{m\pi}{a}\right)^2\right]\cosh(\alpha b)\right\} - A_b\,\sigma_x\sinh(\alpha b)\left(\frac{m\pi}{a}\right)^2\right]$$

$$+B\left[EI\sin(\beta b)\left(\frac{m\pi}{a}\right)^4 + D\left\{\left[\beta^3 + \beta(2-\nu)\left(\frac{m\pi}{a}\right)^2\right]\cos(\beta b)\right\} - A_b\,\sigma_x\sin(\beta b)\left(\frac{m\pi}{a}\right)^2\right] \quad . \qquad (2)$$

$$= 0$$

For a nontrivial solution, the determinant of the coefficient matrix for equations (1) and (2) must vanish. This will yield a transcendental equation for the critical stress

$$\left(\left[\alpha^2 - \nu\left(\frac{m\pi}{a}\right)^2\right]\sinh(\alpha b)\right)\left(EI\sin(\beta b)\left(\frac{m\pi}{a}\right)^4 + D\left\{\left[\beta^3 + \beta(2-\nu)\left(\frac{m\pi}{a}\right)^2\right]\cos(\beta b)\right\} - A_b\,\sigma_x\sin(\beta b)\left(\frac{m\pi}{a}\right)^2\right) +$$

$$\left(EI\sinh(\alpha b)\left(\frac{m\pi}{a}\right)^4 - D\left\{\left[\alpha^3 - \alpha(2-\nu)\left(\frac{m\pi}{a}\right)^2\right]\cosh(\alpha b)\right\} - A_b\,\sigma_x\sinh(\alpha b)\left(\frac{m\pi}{a}\right)^2\right)\left(\left[\beta^2 + \nu\left(\frac{m\pi}{a}\right)^2\right]\sin(\beta b)\right)$$

$$= 0$$

This equation can be solved by trial and error. The following substitutions will aid in the solution:

$$N_x = \sigma_x t_w \qquad b = b_w \qquad a = l \qquad A_b = b_c\,t_c$$

$$D = \frac{E\,t_w^3}{12(1-\nu^2)} \qquad I = \frac{t_c\,b_c^3}{12}$$

Solve for m = 1, 2, 3, ... to find the lowest critical stress. Check the web and cap as usual to determine which buckling mode, lateral or local, occurs first.

Note that at first glance, $\beta = 0$ is a solution which yields

$$\frac{N_x}{D} = \left(\frac{m\pi}{a}\right)^2 .$$

This solution is not valid, however, because owing to some constraints along $y = 0$ and $y = b$, we always have

$$\frac{N_x}{D} > \left(\frac{m\pi}{a}\right)^2 .$$

A2. Ringframe Sizing Equation Derivation

The method used to solve for the required ringframe bending stiffness is taken from reference 10, section 2.6, pages 70 to76. It was originally suggested by Tom Severs of Martin Marietta Manned Space Systems. In this method, a section of the shell consisting of one stringer and the skin adjacent to it is modeled as a beam on elastic supports. The ringframes are the elastic supports and are assumed to be equally spaced and sized. Their size is calculated so that they will function as if they are infinitely stiff and force buckling nodes in the idealized beam. The required support stiffness is

$$\alpha = \frac{mP}{\gamma l} ,$$

where m is the number of spans, γ is a numerical factor which depends on the number of spans, and $P = m^2 \pi^2 EI / l^2$ is the critical load for one span as for a beam of length l/m with hinged ends. As the number of spans increases, γ decreases, approaching a minimum of 0.25. For conservatism, we choose $\gamma = 0.25$. Then

$$\alpha = \frac{4mP}{l} .$$

Now, to equate this required support stiffness to a required ring stiffness, we use the analysis of rigid rings from the reference 11, in-plane load case 1, B.6.1.1 page 8-9. From this reference, we find for $\phi = 0^\circ$, $K_\Delta \approx 0.043$, and

$$\Delta = K_\Delta \frac{PR^3}{EI} ,$$

from which

$$I_{req} = K_\Delta \frac{PR^3}{E\Delta} .$$

Assuming a unit deflection, $\Delta = 1$, $\frac{P}{\Delta} = \alpha$, and we have

$$I_{req} = K_\Delta \frac{\alpha R^3}{E} = \frac{0.172 mPR^3}{EI} .$$

Converting this equation to be in terms of stress, and using the notation in the main body of the paper

$$I_r = 0.172 \frac{\sigma_{cr,sk+st}(A_{st} + bt) r^3}{dE} .$$

The required ringframe area is calculated using equation 11-21, on page 493 of reference 10

$$A_r = \frac{4\pi^2 I_{sk+st} r(r - Z_r)}{bd^3} .$$

APPENDIX B

Sample Input File

```
FSKTDOC.CIN, NLS MaxQ Loads
0           Screen output by DOT 0=none 7=most,IPRINT
1           METHOD 0,2=Mod.Meth.of Feas.Dirs.,2=Seq.Lin.Prog
3           Output, 0=final,1=final calcs.+0,2=dv/con+0,3=dv/con+1
0.330000    Poisson's Ratio
10800000.00 Young's Modulus, compressive
4000000.000 Shear Modulus
0.111000    density
63000.00000 Ultimate Tensile Strength, enter as positive
53000.00000 Compressive Yield Strength, enter as positive
165.500000  radius of cylinder
48.000000   length of cylinder
0.000000    extra non-optimized weight
H           stringer type, 'H' for hat-stringers, 'I' for I-stringers
Y           Allow Local Elastic Buckling [Y/N]
0           I-str coupled buckling, 0=find bucket, #=search m=1 to #
15.000000   web angle in degrees, 0 is perpendicular to skin
0.920000    hat to skin length (2/hat), or height of I bottom flange
1.000000    1 for external stringers, -1 for internal
0.605000    ring cross sectional area
1.669000    ring Moment Of Inertia
-2.099000   ring centroid location
2.952000    ring Polar Moment Of Inertia
0.000000    number of intermediate rings
410481.90   Applied Axial Force, positive is compressive
91520000.00 Applied Bending Moment, enter as positive
98372.00000 Applied Shear Forvce, enter as positive
0.000000    Axial Pressure (i.e. tank ullage) +internal
-0.087000   Hoop Pressure (i.e. ullage+head) +internal
1.400000    overall safety factor
1.000000    skin buckling safety factor
1.000000    h min, stringer height
1.500000    h init
5.000000    h max
1.000000    12 min, hat: top flange length, I: top flange thickness
1.500000    12 init
5.000000    12 max
0.025000    tst min, hat: stringer thickness, I: web thickness
0.040000    tst init
0.250000    tst max
100.000000  Nst min, number of stringers
180.000000  Nst init
360.000000  Nst max
0.025000    t min, skin thickness
0.070000    t init
0.250000    t max
0.000000    W min, hat: top flange thickness, I: width
0.000000    W init, hat: set min,init and max to 0 for
0.000000    W max,       hats of constant tst
```

APPENDIX C

Sample Output File

```
CORSS - Cylinder Optimization of Rings, Skin and Stringers
     NASA/MSFC/ED52 - Structural Development Branch
     Jeff Finckenor, Sep. 1993, ver. 2.1

FSKTDOC.CIN, NLS MaxQ Loads


********************** INPUT VALUES *****************
FLAGS
     IPRINT = 0      No screen output by DOT
     METHOD = 1      Modified Method of Feasible Directions
     SSP = 3         Optimization history, final calculations and final output

Material Properties
     nu    = 0.330              Poisson's Ratio
     E     = 1.080E+007         Young's Modulus
     SM    = 4.000E+006         Shear Modulus
     rho   = 0.111              Density
     Stu   = 63000.0            Ultimate Tensile Stress
     Scy   = 53000.0            Yield Compressive Stress

Cylinder Geometry
     r    =  165.50             Radius
     l    =  48.00              Length
     fwt =  0.00                Additional Weight

Stringer Parameters
     stype = H                  Hat Stringers
     Local Elastic Buckling IS ALLOWED
     alp    = 15.0              Leg angle
     l1     = 0.920             Stringer/skin length
     MZs    = 1                 Stringers external

Ring Parameters
     No Rings

Loads
     F    = 410481.9            Axial Compression
     M    = 9.152E+007          Bending Moment
     V    = 98372.0             Shear force
     Pa   = 0.000               Axial pressure component
     Ph   = -0.087              Hoop pressure component
     sf   = 1.40                Safety factor
     sfp  = 1.00                Plate buckling safety factor

Design Variables
Var.      Minimum        Initial        Maximum       Name
h          1.0000         1.5000         5.0000       Stringer height
l2         1.0000         1.5000         5.0000       Top flange length
tst        0.0250         0.0400         0.2500       Stringer thickness
Nst         100.0          180.0          360.0       Number of stringers
t          0.0250         0.0700         0.2500       Skin thickness
W          0.0000         0.0000         0.0000       Top Flange Thickness
```

OPTIMIZATION HISTORY

h	l2	tst	Nst	t	W	skin	shell	stres	Crip1	Crip2	Wt	It
1.5000	1.5000	0.0400	180.0	0.0700	0.0400	-0.34	0.19	-0.66	-0.71	-0.72	629	0
1.5000	1.5000	0.0400	180.0	0.0700	0.0400	-0.34	0.19	-0.66	-0.71	-0.72	629	0
1.5015	1.5000	0.0400	180.0	0.0700	0.0400	-0.34	0.19	-0.66	-0.71	-0.72	629	0
1.5000	1.5015	0.0400	180.0	0.0700	0.0400	-0.33	0.19	-0.66	-0.71	-0.72	629	0
1.5000	1.5000	0.0400	180.0	0.0700	0.0400	-0.34	0.19	-0.66	-0.71	-0.72	629	0
1.5000	1.5000	0.0400	180.2	0.0700	0.0400	-0.34	0.19	-0.66	-0.71	-0.72	629	0
1.5000	1.5000	0.0400	180.0	0.0701	0.0400	-0.34	0.19	-0.66	-0.71	-0.72	629	0
1.5096	1.5027	0.0398	185.1	0.0693	0.0398	-0.32	0.16	-0.66	-0.71	-0.72	632	0
1.5251	1.5071	0.0396	193.3	0.0683	0.0396	-0.30	0.12	-0.67	-0.72	-0.72	637	0
1.5656	1.5187	0.0389	214.9	0.0655	0.0389	-0.24	0.03	-0.67	-0.72	-0.73	650	0
1.6718	1.5489	0.0372	271.4	0.0583	0.0372	-0.04	-0.15	-0.68	-0.73	-0.74	683	0
1.5832	1.5237	0.0386	224.2	0.0643	0.0386	-0.22	0.00	-0.68	-0.72	-0.73	656	0
1.5848	1.5237	0.0386	224.2	0.0643	0.0386	-0.22	-0.00	-0.68	-0.72	-0.73	656	1
1.5832	1.5252	0.0386	224.2	0.0643	0.0386	-0.22	-0.00	-0.68	-0.72	-0.73	656	1
.												
.												
.												

(iterations 1 through 6 omitted for brevity)

.												
.												
.												
1.9490	1.4484	0.0313	153.7	0.0649	0.0313	0.01	0.01	-0.60	-0.66	-0.66	544	6
1.9527	1.4479	0.0313	153.6	0.0651	0.0313	-0.00	-0.00	-0.60	-0.66	-0.66	544	6
1.9515	1.4442	0.0312	152.5	0.0648	0.0312	0.05	0.01	-0.60	-0.66	-0.66	541	7
1.9556	1.4394	0.0314	152.3	0.0659	0.0314	0.01	0.00	-0.60	-0.66	-0.67	548	7
1.9556	1.4385	0.0314	152.3	0.0660	0.0314	0.00	0.00	-0.60	-0.66	-0.67	549	7
1.9490	1.4484	0.0313	153.7	0.0649	0.0313	0.01	0.01	-0.60	-0.66	-0.66	544	7
1.9527	1.4479	0.0313	153.6	0.0651	0.0313	-0.00	-0.00	-0.60	-0.66	-0.66	544	7

Number of Iterations to Optimize = 7
FSKTDOC.CIN, NLS MaxQ Loads

STRINGER PROPERTY CALCULATIONS

Area 0	Area 1	Area 2	Area 3	Area 4	
0.028783	0.061082	0.045349	0.061082	0.028783	Area
0.015643	0.974200	1.932756	0.974200	0.015643	d
0.000450	0.059506	0.087648	0.059506	0.000450	A*d
0.000007	0.057971	0.169402	0.057971	0.000007	A*d*d
0.000002	0.018103	0.000004	0.018103	0.000002	I

A = 0.225079, ybar = 0.922166, Ix = 0.130166

OVERALL CYLINDER CALCULATIONS

Io = 1429100 Tension Stress = f(Io) = -9211
Cross sectional area = 102.133537
Faxial = f(F,Pa) = 5626.7 Faxialp = 4019.1
Ring Spacing = 48.0000 Stringer Spacing = 6.7519

SKIN BUCKLING STRESS AND MAX SHEAR CALCULATIONS

Scrpl = 14691.7, taucr = 19833.8
bpl = 3.3801, Kc = 4.0000, Ks = 5.4000, Z = 1.0044

N	q[I]	Ssk[I]	G[0]
0	-1.289	14617.741	-0.005

1	-3.866	14608.922	-0.006
2	-6.437	14582.479	-0.007
3	-8.996	14538.459	-0.010
4	-11.541	14476.931	-0.015
5	-14.066	14397.999	-0.020

.
.
.
(stringers 6 through 37 omitted for brevity)
.
.
.

38	-63.218	4236.436	-0.709
39	-63.166	3804.072	-0.739
40	-63.008	3372.066	-0.768

Max shear stress = 974.4, Max skin stress = 14617.7
Skin Buckling Drivers: Stress = 14617.7, Shear Stress = 19.9

STRINGER CRIPPLING CALCULATIONS

Crippling: Top Flange = 63000.0, Web = 63000.0
Weighted Average Crippling Stress, 63000.00

STRINGER COLUMN BUCKLING CALCULATIONS

Number of iterations to converge on I, 4

Tension load (adjusted for buckled skin) = -9288.7

Stress Carried in Buckled Skin, Slb = 13222.5

I	Y-Stringer	Y-skin	Stress-str.	Stress-skin	Eff. Width
0	166.4546	165.5000	21179.3	21091.7	1.2454
1	166.3161	165.3623	21166.6	21079.0	1.2458
2	165.9008	164.9494	21128.5	21041.1	1.2469
3	165.2095	164.2620	21065.0	20978.0	1.2488
4	164.2431	163.3012	20976.3	20889.9	1.2514
5	163.0035	162.0687	20862.5	20776.7	1.2548

.
.
.
(stringers 6 through 35 omitted for brevity)
.
.
.

36	16.9662	16.8689	7458.7	7449.8	2.0986
37	10.1985	10.1400	6837.5	6832.2	2.1918
38	3.4138	3.3942	6214.8	6213.0	2.2990

I	Area	sum Area	sum Area*y	sum Area*y*y	skin/str.
0	0.043	0.043	7.217	1201.257	stringer
0	0.043	0.086	14.273	2369.121	skin
1	0.043	0.129	21.471	3566.174	stringer
1	0.043	0.172	28.502	4728.941	skin
2	0.043	0.215	35.642	5913.446	stringer
2	0.042	0.257	42.611	7062.970	skin
3	0.043	0.300	49.655	8226.756	stringer
3	0.042	0.342	56.525	9355.079	skin
4	0.042	0.384	63.436	10490.267	stringer

```
  4          0.041          0.425          70.169       11589.726 skin
  5          0.041          0.466          76.912       12688.838 stringer
  5          0.040          0.507          83.473       13752.177 skin
.
.
.
(stringers 6 through 25 omitted for brevity)
.
.
.
 26          0.000          1.420         213.621       32558.371 stringer
 26          0.000          1.420         213.637       32559.602 skin
 27         -0.001          1.420         213.581       32555.406 stringer
 27         -0.001          1.419         213.535       32551.990 skin
 28         -0.001          1.418         213.460       32546.814 stringer
 28         -0.001          1.417         213.418       32543.922 skin
 29         -0.000          1.417         213.391       32542.178 stringer

 Ybar =   2.119,   Cylinder I = 1396105.6,     Anew =     100.7

Euler Column Buckling = 21194.9
Applied Stress =  21089.1
Effective skin width on stringer with max stress = 4.981510
Stringer+Skin I = 0.2512, radius of gyration = 0.6769

GENERAL CYLINDER BUCKLING CALCULATIONS

Ex  =   1146387.8750  Ey  =    786363.3125  Exy =    259499.9063
Dx  =    536564.0000  Dy  =       275.8633  Dxy =    167651.5781
Cx  =    343681.9688  Cy  =     1.08E-034  Cxy =     2.554E-005
Gxy =    263431.7188  Kxy =     2.554E-005

Critical Line Load, m=1, n=13, Ncr=4557.328125

 A = |    6536.1563        2688.4309         198.9806 |
     |    2688.4309        5980.3813         373.2249 |
     |     198.9806         373.2249          42.9972 |

determinant of A(3x3) = 6.21996E+008
determinant of A(2x2) = 3.1861E+007

Critical line load is Ncr = 4557.328125
Knockdown Adjusted Line Load is 4674.877441
at axial waves m = 1, and hoop waves n = 13
gammaF = 0.348735,  gammaM = 0.471615

Critical Pressure

 A = |   56163.5859       15096.5742         198.9806 |
     |   15096.5742      154121.9531        2095.8013 |
     |     198.9806        2095.8013         188.7229 |

determinant of A(3x3) = 1.35038E+012
determinant of A(2x2) = 8.42813E+009
Critical Pressure =   3.73 with 73 circumferential waves

Critical crushing pressure Pcr = 3.731967
Column Buckling provides support above General Cylinder Buckling
```

```
VON MISES STRESS CHECK

Point of Max Compression:
Sx =  21089.1,   Sy =    310.7,   tauxy =    -19.9
S1 =  21089.2,   S2 =    310.7,   Von Mises Stress =  20935.
Point of Max Shear:
Sx =   5705.9,   Sy =    310.7,   tauxy =    974.4
S1 =   5876.4,   S2 =    140.1,   Von Mises Stress =   5807.6
Point of Max Tension:
Sx =  -9288.7,   Sy =    310.7,   tauxy =    -19.9
S1 =    310.7,   S2 =  -9288.8,   Von Mises Stress =   9448.0

-------------------------------------------------------------------------------

Cylinder Weight = 544.2
 (Skin: 359.5, Stringers: 184.7, Rings: 0.0, Flanges: 0.0)

h   = 1.9484,      Stringer Height
l2  = 1.4495,      Hat stringer top flange length
tst = 0.0313,      Hat stringer thickness
Nst =  154.0,      Number of Stringers, b = 6.7519
t   = 0.0649,      Skin Thickness
W   = 0.0313,      Hat stringer top flange thickness

Stringer: I = 0.130166, J = 0.282368, Z = 0.954607, A = 0.225079
End Ring I should be at least 21.14
End Ring Area should be at least 0.00113895*(r+Z)
                                                    G[] value
Skin: (Shear ratio)**2 + Stress ratio           = 0.99497 < 1    -0.00503

Applied Column Stress (SF = 1.40)               =     21089.1
    Critical Stringer Crippling Stress          =  63000.0       -0.66525
    Critical Column Stress                      =  21194.9       -0.00499

Applied Von Mises Stress (SF = 1.40)            =     20935.6
    Yield Compressive Stress                    =  53000.0       -0.60499

General Cylinder Buckling controlled by Critical Column Buckling
Applied Crushing Hoop Pressure (SF = 1.40)      =             0.122
    Critical Buckling Pressure                  =    3.732
    Hoop waves, n = 73
```

APPENDIX D

Program Listing

```
/* C language include files for standard function definitions */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>



/* define pi for use in the program */
#define pi  3.141592654



/* Variable Structures
Many often passed values are grouped into structures of related variables
stiffness - the smeared orthotropic cylinder properties used in General
      Cylinder Buckling Calculations
      Ex, Ey, Exy - smeared extensional stiffnesses
      Gxy - smeared shear stiffness
      Cx, Cy, Cxy, Kxy - smeared coupling constants
      Dx, Dy, Dxy - smeared bending stiffnesses
material - the material properties from the input file
      E - Young's Modulus
      nu - poisson's ratio
      rho - density
      Scy - compressive yield strength
      G - Shear Modulus
      Stu - tensile ultimate strength
load - the loading parameters from the input file
      F - axial force, positive is compressive
      M - bending moment
      Pa - axial pressure (i.e. ullage), internal is positive
      Ph - hoop pressure (i.e. ullage+head), internal is positive
      sf - safety factor to be applied to loads
      sfp - skin buckling safety factor, assumed 1.0<=sfp<=sf
      V - shear force
stringer - the variables that define the stringers
      A - cross sectional area
      I - Moment of Inertia about the neutral axis parallel to the skin
      J - The polar moment of inertia, Ix+Iy
      N - the number of stringers (optimized design variable)
      Z - neutral axis distance from the skin
      alp - web angle
      l1 - hats: stringer to skin length, 2/hat; I: bottom flange thickness
      MZs - Z multiplier, 1 for external stringers, -1 for internal
      h - stringer height (optimized design variable)
      l2 - hat: top flange width, I: top flange thickness (design variable)
      t - skin thickness (optimized design variable)
      W - hat: top flange thickness, I: stringer width (design variable)
      b - stringer spacing
      la - projected length of the hat stringer web on the skin
```

```
      stype - stringer type identifier, H for hats, I for I's
ring - ring parameters from the input file
      A - cross sectional area
      I - moment of inertia about the neutral axis parallel to the skin
      J - Polar Moment of Inertia, Ix+Iy
      N - number of intermediate rings (not end rings)
      Z - neutral axis distance from the skin, + for external
      d - ring spacing
cylinder - cylinder geometry definitions
      fwt - additional weight, not optimized
      l - length of cylinder
      r - radius of cylinder
      t - skin thickness (optimized design variable)
*************************************************************************/
struct stiffness{float Ex, Ey, Exy, Gxy, Dx, Dy, Dxy, Cx, Cy, Cxy, Kxy; };
struct material {float E, nu, rho, Scy, G, Stu; };
struct load     {float F, M, Pa, Ph, sf, sfp, V; };
struct stringer {float A, I, J, N, Z, alp, l1, MZs, h, l2, t, W, b, la;
             char stype;};
struct ring     {float A, I, J, N, Z, d; };
struct cylinder {float fwt, l, r, t; };



/*************************************************************************/
/*************************    file CT.C    *******************************/
/*************************************************************************/

/* MAIN starts the program, initialized the variables and calls the
      optimization routine
called from: DOS command line
calls      : readinput, pinput, initDVs, CALLeval, and corsstol
returns    :
    argc, argv - the number and array of command line parameters,
    G - array of constraint values,
    hap - position of stringer height in X
    i - index,
    in, out - handles for the input and output files,
    IPRINT - DOT output flag,
    l2ap - position of top flange length in X
    LEB - Y/N flag to allow local elastic buckling of the stringer segments
    METHOD - DOT method flag,
    mflag - flag to control search of I-coupled buckling
    NCON - number of constraints,
    NDV - number of design variables,
    Nstap - position of number of stringers in X
    OBJ - weight,
    SSP - CORSS output flag
    tap - position of skin thickness in X
    Title - title of run,
    Tol - array of tolerances associated with X
    tstap - position of stringer thickness in X
    Wap - position of I width: H top flange thickness in X
    X, XL, XU - array of design variables, and lower and upper limits,
    Xinit - array of initial variables for normalizing,
*************************************************************************/
```

```
/* comlineerr is called if there is an error with the command line.  It
      prints the correct usage message and exits the program
called from: MAIN
calls      :
returns    :
*************************************************************************/
void comlineerr();



/* initDVs controls the removal of variables from the design variable arrays
called from: MAIN
calls      : Xadust
returns    : modified X,XL,XU,Xinit arrays, and hap,l2ap,tstap,Nstap,tap,Wap
             indices and the new number of design variables
      i - incrementing index
      NDV - number of design variables
      pos - current X array position
*************************************************************************/
long int initDVs(float XL[], float X[], float XU[], int *hap, int *l2ap,
      int *tstap, int *Nstap, int *tap, int *Wap, float Xinit[],
      long int ndv);



/* Xadjust manipulates the design variable arrays so that only the needed
      design variables are iterated upon
called from: initDVs
calls      :
returns    : next available design variable array position
      k - index
*************************************************************************/
int Xadjust(int newpos, int pos, long int *NDV, float X[], float Xinit[],
          float XL[], float XU[]);



/*************************************************************************/
/*************************    file CTOPT.C    ****************************/
/*************************************************************************/

/* DOT is the numerical optimizer program
called from: ctopt
returns    : new values for X
*************************************************************************/
void fortran DOT(long int *INFO,   long int *METHOD, long int *IPRINT,
             long int *NDV,    long int *NCON,   float    X[],
             float    XL[],    float    XU[],    float    *OBJ,
             long int *MINMAX, float    G[],     float    RPRM[],
             long int IPRM[],  float    WK[],    long int *NRWK,
             long int IWK[],   long int *NRIWK);
```

```
/* CALLeval uses the X array and the 'ap' variable to prepare the variables
      for the analysis of EVAL
calls   : eval
returns :
*************************************************************************/
void CALLeval(struct stringer S, struct ring R, struct material M,
      struct load L, struct cylinder C, float G[], float X[],
      float Xinit[], float *OBJ, FILE *out, int SSP, int hap, int l2ap,
      int Nstap, int tap, int tstap, int Wap, int mflag, char LEB);



/* ctopt controls the optimization calls to eval and DOT
called from: MAIN
calls      : CALLeval and DOT
returns    :
      INFO - DOT completion flag
      IPRM, IWK, RPRM, WK - DOT working arrays
      MINMAX - DOT minimization/maximization flag
      NRIWK, NRWK - IWK and WK sizes
*************************************************************************/
void ctopt(int SSP, FILE *out, float X[], float Xinit[], int hap,
      int l2ap, int tstap, int Nstap, int tap, int Wap, struct stringer S,
      struct ring R, struct material M, struct load L, struct cylinder C,
      float G[], float *OBJ, long int *METHOD, long int *IPRINT,
      long int *NDV, long int *NCON, float XL[], float XU[], int mflag,
      char LEB);


/*************************************************************************/
/*************************    file EVAL.C     ****************************/
/*************************************************************************/

/* EVAL is the function that controls the analysis
called from: CALLeval
calls      : stringer, skinbuck, getScrip, colstress, GCBcalc, stresscheck,
             and finalout
returns    : G array values
      Anew - cylinder area with buckled skin,
      Faxial, Faxialp - axial stress from F and Pa only, w/ sf and sfp
      gamF - general cylinder buckling knockdown factor for axial loads
      gamM - general cylinder buckling knockdown factor for bending
      GCB - flag for identifying the shell buckling failure mode
      i - index
      Io - cylinder total moment of inertia
      mdrv - the critical wavelength value for I-coupled buckling
      mgcb, ngcb - half axial and hoop waves for Nx
      N - knockdown adjusted applied line load
      Nx - critical general cylinder buckling line loads
      ncr - hoop waves for Pcr
      Pcr - critical general cylinder buckling pressure
      rasl, risl - radius average skin line, inner skin line
      St - applied tension stress
      Scol, Scrstcol - applied and critical stringer/column stress
      Scrip1/2, - crippling and local buckling critical stresses
```

```
      ScripS - weighted average of inelastic crippling stress
      Scrpl, Sskbd - skin stresses: crit. skin buckling, skin buck. driver
      Ssk - maximum stress in the skin
      tau, taucr, tauskbd, tau0 - shear stresses: maximum, skin buckling
            critical,skin buckling driver, minimum
      theta, dtheta - angle and angle between stringers
      Yst, Ysk - stringer and skin y distances from cylinder center
*************************************************************************/
void eval(struct stringer S, struct ring R, struct material M, struct load L,
      struct cylinder C, float G[], float *OBJ, FILE *out, int SSP,
      int mflag, char LEB);



/*************************************************************************/
/************************    file STRINGER.C   ***************************/
/*************************************************************************/

/* STRINGER calculates the stringer cross sectional properties
called from: eval
calls      : stringerP
returns    : S.A for stresscheck,
           S.A, S.la, S.I, and S.Z for colstress,
           S.A, S.la for skinbuck
           S.A, S.I, S.Z, S.J for GCBcalc
must be called before stresscheck, colstress, GCBcalc, or skinbuck
      a - array of segment areas
      ad - (segment areas)*(segment distance from skin)
      add - (segment areas)*(segment distance from skin)**2
      alpha - web angle in radians
      d - array of segment distances from the skin
      ha - hat stringer web slant height
      i - index
      I - array of segment moments of inertia
      ix,iy - stringer moments of inertia parallel and perpendicular to
                  the skin, respectively
      na - number of stringer segments
      sumad,sumadd,sumi - summations of the ad, add, and I arrays
      xbar,ybar - stringer neutral axis distances from the skin and the
            side of the stringer, respectively
*************************************************************************/
void stringer(struct stringer *S, struct cylinder C, FILE *out, int SSP);



/* STRINGERP prints section information to the output file
called from: stringer
calls      :
returns    :
*************************************************************************/
void stringerP(int na, float a[], float d[], float ad[], float add[],
            float I[], float A, float bar, float i, char line[],
            FILE *out);
```

```
/*****************************************************************************/
/*************************    file SKINBUCK.C    *************************/
/*****************************************************************************/

/* SKINBUCK calculates the skin buckling constraint
called from: eval
calls      : Ks, Kc
returns    : G[0] for DOT
             Scrpl, Sskbd, tauskbd, taucr for colstress
             Stsk, tau, tau0 for stresscheck
must be called before stresscheck, and colstress
    alpha, K, K1 - pressure stabilization factors
    bpl - distance between fixed points on the skin
    g0, g0max - design ratio, and maximum design ratio
    i - index
    Kcr, Ksh - axial and shear plate buckling constants
    q, qmax - shear flow between stringers and maximum shear flow
    Ssk - stress in the skin
    Z - non dimensional parameter for calculating Kcr and Ksh
*****************************************************************************/
float skinbuck(struct stringer S, float d, struct material M,
      struct load L, struct cylinder C, float Faxialp, float Io, FILE *out,
      float *Scrpl, float *Sskbd, int SSP, float *Stsk, float *tau,
      float *tau0, float *taucr, float *tauskbd, float Ysk[], float Yst[]);



/* Kc interpolates between curves for the skin buckling constants
called from: skinbuck
calls      : Kc## functions
returns    : skin buckling constant
  Kc## functions return the skin buckling ratio for the given r/t ratio
  Ks returns the shear skin buckling constant
*****************************************************************************/
float Kc(float Z, float r, float t);
float Kc100(float Z);
float Kc300(float Z);
float Kc500(float Z);
float Kc1000(float Z);
float Kc1500(float Z);
float Ks(float Z);



/*****************************************************************************/
/*************************    file LEBCRIP.C    *************************/
/*****************************************************************************/

/* GETSCRIP calculates elastic buckling, if needed, and inelastic crippling
called from: eval
calls      : findICBuck
returns    : Scrip for colstress,
             and 1 or 2 critical constraint values for DOT
must be called before colstress
      alpha - web angle in radians
      m - I stringer coupled buckling wave number
```

```
      S1 - crippling stress, or I flange elastic buckling stress, or
           hat flange elastic buckling stress
      S2 - I coupled buckling stress, hat web local buckling (if needed)
      S3 - flange crippling stress
      S4 - web crippling stress
*************************************************************************/
float getScrip(struct stringer S, struct material M, FILE *out,
      float *Scrip1, float *Scrip2, float *ScripS, int SSP, int *mdrv,
      float l, int mflag, char LEB);



/* findICBuck controls the search for the lowest coupled buckling stress
called from: getScrip
calls      : secant
returns    : critical stress with the associated number of waves
      i - index
      m - number of waves
      S - critical stress variable
      Sdrv - lowest critical stress
      Slast - critical stress from the previous iteration
*************************************************************************/
float findICBuc(int mflag, int *mdrv, float l, float tst, float nu, float h,
      float l2, float W, float E, int SSP, FILE *out);



/* secant performs the secant method root solver
called from: findICBuck
calls      : G6eqn
returns    : critical stress from a given number of waves
      count - iteration limiter
      Fj, Fi - function values on different iterations
      hold - used to switch Si and Sj
      i - index
      Smin, Smax - bounds of critical stress range
      Si, Sj - stress values on different iterations
*************************************************************************/
float secant(int m, float l, float tst, float nu, float h, float l2, float W,
      float E);



/* G6eqn is the defining equation for I coupled buckling
called from: secant
calls      :
returns    : value of the function
*************************************************************************/
float G6eqn(float S, float m, float l, float tst, float nu, float h,
      float l2, float W, float E);



/* domainexit exits the program when to prevent a sqrt error which comes
      from having a negative web length
called from: getScrip
```

```
calls      :
returns    : ends the program
*************************************************************************/
void domainexit(float l2, float l1, float h, FILE *out);



/*************************************************************************/
/************************    file COLSTRESS.C    ************************/
/*************************************************************************/

/* COLSTRESS calculates the stress in the stringer and does the stress
 analysis if the skin buckles and redistributes the load
called from: eval
calls      : loopI
returns    : column buckling constraint value, adjusted cylinder area and
      tension stress, stress in the column, and the column buckling stress
must be called after stringer, skinbuck, and getScrip
must be called befor stresscheck
      be - effective skin width
      Ise - skin + stringer moment of inertia
      radg - skin + stringer radius of gyration
      ScrstcolJE - critical Johnson-Euler column stress
      tmp - effective width temporary location
      We0 - minimum effective skin width
*************************************************************************/
float colstress(float *Anew, struct stringer S, float d, float E,
      struct load L, struct cylinder C, float Faxial, float Io, FILE *out,
      float *St, float *Scol, float Scrip, float Scrpl, float *Scrstcol,
      float Sskbd, int SSP, float taucr, float tauskbd, float Ysk[],
      float Yst[]);



/* LOOPI controls the iterations for calculating the cylinder properties
with the skin buckled
called from: colstress
calls      : newI
returns    : stringer column stress, new cross sectional area and tension
             stress, minimum effective skin width
      Acyl - cylinder cross sectional area
      count - iteration limiter
      I - cylinder moment of inertia
      Iold - last iterations moment of inertia
      ybar - distance of cylinder neutral axis from the centerline
*************************************************************************/
float loopI(float *Anew, struct stringer S, float M, float sf, float E,
      struct cylinder C, float Faxial, float Io, FILE *out, float *St,
      float Scrpl, int SSP, float *We0, float Ysk[], float Yst[]);

/* NEWI calculates the new cylinder moment of inertia and neutral axis
called from: loopI
calls      :
returns    : adjusted area, moment of inertia and neutral axis location
             of the cylinder, and the minimum effective skin width
      A - ineffective skin area
```

```
      i - index
      N - 1/2 number of stringers (symmetric)
      Slb - stress carried in buckled skin
      Ssk, Sst - stress in individual skin segments and stringers
      sumA, sumAy, sumAyy - summations of area, area*distance from
            neutral axis, and area*distance**2
      tmp - temporary effective skin width location
      We - effective skin widths
*************************************************************************/
void newI(float Acyl, float *Anew, struct stringer S, float M, float sf,
      float t, float E, float Faxial, float *I, float Io, FILE *out,
      float Scrpl, int SSP, float *We0, float *ybar, float Ysk[],
      float Yst[]);



/*************************************************************************/
/*************************    file STRESS.C       ************************/
/*************************************************************************/

/* STRESSCHECK checks the Von Mises Stress at three locations
called from: eval
calls      : VMStress
returns    : highest Von Mises stress
must be called after stringer, colstress, and skinbuck
    risl - radius at the inner skin line
    VMSA, VMSB, VMSC - Von Mises stress at the three locations
*************************************************************************/
float stresscheck(float Anew, float As, float Ns, float Ph, float r, float t,
      float Faxial, FILE *out, float St, float Scol, int SSP, float tau,
      float tau0, float sf);



/* VMSStress calculates Von Mises Stress given Sx, Sy, and Tauxy
called from: stresscheck
calls      :
returns    : Von Mises stress
      S1, S2 - plane stress principle stresses
      VMS - Von Mises stress
*************************************************************************/
float VMStress(char loc[],  FILE *out, int SSP, float Sx, float Sy,
             float tauxy);


/*************************************************************************/
/*************************    file GENCYL.C       ************************/
/*************************************************************************/

/* GCBcalc controls the general cylinder buckling analysis
called from: eval
calls      : getstiffnesses, gencyl, gammaM, gammaF, and Pcrcalc
returns    : the shell buckling constraint, the number of axial and hoop
             waves in the buckled mode, the critical line load and
             pressure, the adjusted line load, the number of hoop
             waves from pressure buckling, the shell buckling flag,
```

```
            and the knockdown factors
      egdck - structure of smeared orthotropic shell properties
      G1 - temporary constraint value
*************************************************************************/
float GCBcalc(struct ring R, struct stringer S, struct material M,
      struct cylinder C, struct load L, float G[], FILE *out, int SSP,
      int *mgcb, int *ngcb, float *Nx, float *Pcr, float *N,
      float Io, float Faxial, int *ncr, char *GCB, float *gamF,
      float *gamM);



/* GENCYL calculates the general cylinder buckling critical load
called from: GCBcalc
calls      : getA
returns    : critical line load and the hoop and axial waves
      m, n - working values of axial and hoop waves
      mm, nm - tracks minimum numbers of waves
      Nm - tracks minimum line load
      Nmt - temporary critical line load
      Nold - last iterations line load
*************************************************************************/
void gencyl(float l, float Nr, float r, struct stiffness egdck, int *mmin,
      int *nmin, float *Nx, FILE *out, int SSP);



/* Pcrcalc calculates the critical pressure buckling load
called from: GCBcalc
calls      : getA
returns    : critical pressure and number of hoop waves
      n - working number of hoop waves
      P - working pressure
      Pcr - critical pressure
      Pold - last iterations critical pressure
*************************************************************************/
float Pcrcalc(struct stiffness egdck, float r, float l, int *ncr, FILE *out,
      int SSP);


/* GETA calculates det(A[3][3])/det(A[2][2])
called from: gencyl, Pcrcalc
calls      :
returns    : result from division of the determinates
      A11..A33 - A array locations
      detA3x3 - determinate of the full three by three array
      detA2x2 - determinate of [A11 A12 / A21 A22] only
*************************************************************************/
float getA(struct stiffness ES, float r, float l, int m, int n, FILE *out,
        int SSP);



/* GETSTIFFNESSES calculates the smeared orthotropic shell properties
called from: GCBcalc
calls      :
```

```
returns     : the smeared orthotropic cylinder properties
      Er, Es - Young's Modulus of the rings and stringers
      Gr, Gs - Shear Modulus of the rings and stringers
*************************************************************************/
void getstiffnesses(struct ring R, struct stringer S, struct material M,
                struct cylinder C, struct stiffness *egdck, FILE *out,
                int SSP);



/* gammaF and gammaM calculate the axial and bending knockdown factors
called from: GCBcalc
calls       :
returns     : the knockdown factor
*************************************************************************/
float gammaF(struct stiffness ES, float r);
float gammaM(struct stiffness ES, float r);



/*************************************************************************/
/************************    file CIO.C          ************************/
/*************************************************************************/

/* finalout prints the last set of output data
called from: eval
calls       :
returns     :
*************************************************************************/
void finalout(struct ring R, struct stringer S, struct material M,
      struct load L, struct cylinder C, float G[], float gamF, float gamM,
      char GCB, int mgcb, float N, int ncr, int ngcb, float Nx, float obj,
      FILE *out, float Pcrush, float Scol, float Scrip1, float Scrip2,
      float Scrstcol, char LEB, int mdrv, float Io, float Faxial);



/* cinput, finput, iinput read a character, a float and an integer,
      respectively from the input file then skips to the next line
called from: readinput
calls       :
returns     : a character, float, or integer
*************************************************************************/
char  cinput(FILE *infile);
float finput(FILE *infile);
int   iinput(FILE *infile);



/* pinput, repeats the information from the input file so that input and
      output are always kept together
called from: main
calls       :
returns     :
*************************************************************************/
void pinput(struct stringer S, struct ring R, struct material M,
```

```
      struct load L, struct cylinder C, long int IPRINT, long int METHOD,
      FILE *out, int SSP, float X[], float XL[], float XU[], int mflag,
      char LEB);


/* readinput, reads the information in the input file
called from: main
calls      :
returns    : all input information
*************************************************************************/
void readinput(char Title[], FILE *in, long int *IPRINT, long int *METHOD,
      int *SSP, struct material *M, struct cylinder *C, struct stringer *S,
      struct ring *R, struct load *L, float XL[], float X[], float XU[],
      float Tol[], int *mflag, char *LEB);




/*************************************************************************/
/*************************    file CTMAIN.C       ************************/
/*************************************************************************/
#include "ct.h"

void main(argc, argv)
int argc;
char *argv[];
{
  FILE *in,*out;
  long int NDV, NCON, IPRINT, METHOD;
  float X[12],XL[6],XU[6],G[5],OBJ;
  float Xinit[12],Tol[6];
  char Title[80],LEB;
  int i,SSP,hap,12ap,tstap,Nstap,tap,Wap,mflag;
  struct material M;
  struct load L;
  struct stringer S;
  struct ring R;
  struct cylinder C;
  printf("\nmain\n");
  NDV = 6;      NCON = 5;
  if ( (in = fopen(argv[1],"rt")) == NULL)
  {
    printf("\ncan't open input file, First parameter must be input file");
    comlineerr();
  }
  if (strcmp(argv[1],argv[2]) == 0)
  {
    printf("\ninput filename is the same as output filename");
    comlineerr();
  }
  if ( (out = fopen(argv[2],"wt")) == NULL)
  {
    printf("\ncan't open output file, Second param. must be output file");
    comlineerr();
```

```
  }
  fprintf(out,"CORSS - Cylinder Optimization of Rings, Skin and Stringe");
  fprintf(out,"rs\n     NASA/MSFC/ED52 - Structural Development Branch");
  fprintf(out,"\n     Jeff Finckenor, Sep. 1993, ver. 2.1\n");
  printf("CORSS - Cylinder Optimization of Rings, Skin and Stringe");
  printf("rs\n     NASA/MSFC/ED52 - Structural Development Branch");
  printf("\n     Jeff Finckenor, Sep. 1993, ver. 2.1\n\n");
  readinput(Title,in,&IPRINT,&METHOD,&SSP,&M,&C,&S,&R,&L,XL,X,XU,Tol,&mflag,&LEB);
  fclose(in);
  fprintf(out,"\n%s",Title);
  if (0 == R.N) { R.A = .001;  R.I = .001;  R.Z = .001;  R.J = .001; }
  pinput(S,R,M,L,C,IPRINT,METHOD,out,SSP,X,XL,XU,mflag,LEB);
  if ( (S.stype != 'H') && (S.stype != 'I') )
  {
    printf("\n\n%c is not a valid stringer type\a\a\a",S.stype);
    exit(4);
  }
  if ( ('H'==S.stype) && ('N'!=LEB) ) NCON--;
  NDV = initDVs(XL,X,XU,&hap,&l2ap,&tstap,&Nstap,&tap,&Wap,Xinit,NDV);
  if (NDV != 0)
      ctopt(SSP,out,X,Xinit,hap,l2ap,tstap,Nstap,tap,Wap,S,R,M,L,C,G,
            &OBJ,&METHOD,&IPRINT,&NDV,&NCON,XL,XU,mflag,LEB);
  fprintf(out,"\n");
  fputs(Title,out);
  if ( (SSP == 1) || (SSP == 3) ) SSP = 4; else SSP = 5;
  CALLeval(S,R,M,L,C,G,X,Xinit,&OBJ,out,SSP,hap,l2ap,Nstap,tap,
                        tstap,Wap,mflag,LEB);
  for (i=0; i<NCON; i++)
    if (G[i]>0)
    {
      fprintf(out,"\n\n** WARNING G[%d] (=%7.5f>0) IS NOT ",i,G[i]);
      fprintf(out,"SATISFIED, THIS IS NOT A VALID SOLUTION!");
    }
  if ( (Tol[0]!=0) || (Tol[1]!=0) || (Tol[2]!=0) || (Tol[3]!=0) ||
       (Tol[4]!=0) || (Tol[5]!=0) )
  {
    for (i=0; i<NDV; i++)
      {  XL[i] *= Xinit[i];  XU[i] *= Xinit[i];  X[i]  *= Xinit[i];  }
    corsstol(S,R,M,L,C,hap,IPRINT,l2ap,METHOD,NCON,NDV,R.N,Nstap,out,
      SSP,tap,Tol,tstap,Wap,X,XL,XU,mflag,LEB);
      /* corsstol must define IPRM, IWK, MINMAX, NRIWK, NRWK, RPRM, WK */
  }
  fclose(out);
  printf("\a\a\a");
}  /*********************************************************  end main  */



/*************************************************************************/
/*************************    file CT.C    *******************************/
/*************************************************************************/
#include "ct.h"

long int initDVs(float XL[], float X[], float XU[], int *hap, int *l2ap,
      int *tstap, int *Nstap, int *tap, int *Wap, float Xinit[],
```

```
      long int ndv)
{
  int i;
  long int NDV = ndv;
  int pos = 0;
  if ( (XL[pos] == X[pos]) && (X[pos] == XU[pos]) )
       *hap = Xadjust(6,pos,&NDV,X,Xinit,XL,XU);
    else *hap = pos++;
  if ( (XL[pos] == X[pos]) && (X[pos] == XU[pos]) )
       *12ap = Xadjust(7,pos,&NDV,X,Xinit,XL,XU);
    else *12ap = pos++;
  if ( (XL[pos] == X[pos]) && (X[pos] == XU[pos]) )
       *tstap = Xadjust(8,pos,&NDV,X,Xinit,XL,XU);
    else *tstap = pos++;
  if ( (XL[pos]==X[pos]) && (X[pos]==XU[pos]) )
       *Nstap = Xadjust(9,pos,&NDV,X,Xinit,XL,XU);
    else *Nstap = pos++;
  if ( (XL[pos]==X[pos]) && (X[pos]==XU[pos]) )
       *tap = Xadjust(10,pos,&NDV,X,Xinit,XL,XU);
    else *tap = pos++;
  if ( (0==XL[pos]) && (0==X[pos]) && (0==XU[pos]) )
  {  *Wap = *tstap;          NDV--;  }
  else
    if ( (XL[pos]==X[pos]) && (X[pos]==XU[pos]) )
         *Wap = Xadjust(11,pos,&NDV,X,Xinit,XL,XU);
      else *Wap = pos;
  for (i=0; i<NDV; i++)
  {
    Xinit[i] = X[i];
    XL[i] = XL[i]/Xinit[i];
    XU[i] = XU[i]/Xinit[i];
    X[i] = 1.0;
  }
  return (NDV);
}       /************************************ end initDVs **********/



void comlineerr()
{
  printf("\nUsage: C:\>CORSS  file.in  file.out");
  fcloseall();
  exit(1);
}



int Xadjust(int newpos, int pos, long int *NDV, float X[], float Xinit[],
          float XL[], float XU[])
{
  int k;
  X[newpos] = X[pos];
  Xinit[newpos] = 1.;
  for (k=pos; k<(*NDV-1); k++) { XL[k] = XL[k+1]; X[k] = X[k+1]; XU[k]=XU[k+1]; }
  *NDV-=1;
  return(newpos);
```

```
}       /************************************************ end Xadjust */


/*************************************************************************/
/************************    file CTOPT.C    ****************************/
/*************************************************************************/
#include "ct.h"

void ctopt(int SSP, FILE *out, float X[], float Xinit[], int hap,
      int l2ap, int tstap, int Nstap, int tap, int Wap, struct stringer S,
      struct ring R, struct material M, struct load L, struct cylinder C,
      float G[], float *OBJ, long int *METHOD, long int *IPRINT,
      long int *NDV, long int *NCON, float XL[], float XU[], int mflag,
      char LEB)
{
  long int INFO,IPRM[20],IWK[110],j,MINMAX,NRIWK,NRWK;
  float RPRM[20],WK[450];
  int iter, i;
  for (j=0; j<20; j++)
  {
   RPRM[j] = 0.0;
   IPRM[j] = 0;
  }
  NRWK =   450;         /*  make the dimension of IWK and WK the same  */
  NRIWK =  110;
  MINMAX = -1;
  INFO = 0;
  if ( (SSP == 2) || (SSP == 3) )
  {
    fprintf(out,"\n\n OPTIMIZATION HISTORY\n\n");
    fprintf(out,"h      l2     tst    Nst   t      W      skin  shell ");
    fprintf(out,"stres Crip1 Crip2 Wt   It");
  }
  iter = 0;
  do
  {
    if (IPRM[18]!= -1)
    {
      printf("\nIteration #%i",IPRM[18]);
      iter= (int) IPRM[18];
    }
    if ( (SSP ==2) || (SSP ==3) )
    {
      fprintf(out,"\n%5.4f %5.4f %5.4f %5.1f %5.4f %5.4f ",
          X[hap]*Xinit[hap],     X[l2ap]*Xinit[l2ap],
          X[tstap]*Xinit[tstap], X[Nstap]*Xinit[Nstap],
          X[tap]*Xinit[tap],     X[Wap]*Xinit[Wap]);
     fflush(out);
    }
    CALLeval(S,R,M,L,C,G,X,Xinit,OBJ,out,SSP,hap,l2ap,Nstap,tap,tstap,Wap,
            mflag,LEB);
    for (i=0; i<(*NCON); i++) G[i]+=.005;
    DOT(&INFO,METHOD,IPRINT,NDV,NCON,X,XL,XU,OBJ,&MINMAX,G,RPRM,IPRM,
        WK,&NRWK,IWK,&NRIWK);
    printf(" %3.1f %7.4f %7.4f %7.4f %7.4f %7.4f",
```

```
            *OBJ,G[0],G[1],G[2],G[3],G[4]);
    if ( (SSP == 2) || (SSP == 3) )
    {
      fprintf(out,"%5.2f %5.2f %5.2f %5.2f %5.2f",G[0],G[1],G[2],G[3],G[4]);
      fprintf(out," %4.0f %d",*OBJ,iter);
    }
  } while (INFO!=0);  /*** end optimization (INFO=0) loop ***/
  fprintf(out,"\n\nNumber of Iterations to Optimize = %d",iter);
  if (iter > 20) fprintf(out," -- WARNING -- Solution may not have converged.");
}       /*********************************************** end ctopt ****/


void CALLeval(struct stringer S, struct ring R, struct material M,
      struct load L, struct cylinder C, float G[], float X[],
      float Xinit[], float *OBJ, FILE *out, int SSP, int hap, int l2ap,
      int Nstap, int tap, int tstap, int Wap, int mflag, char LEB)
{
  S.h = X[hap]*Xinit[hap];
  S.l2 = X[l2ap]*Xinit[l2ap];
  S.N = X[Nstap]*Xinit[Nstap];
  S.t = X[tstap]*Xinit[tstap];
  C.t = X[tap]*Xinit[tap];
  S.W = X[Wap]*Xinit[Wap];
  eval(S,R,M,L,C,G,OBJ,out,SSP,mflag,LEB);
}       /*************************************** end CALLeval ******/


/*************************************************************************/
/*************************    file EVAL.C     ****************************/
/*************************************************************************/
#include "ct.h"

void eval(struct stringer S, struct ring R, struct material M, struct load L,
      struct cylinder C, float G[], float *OBJ, FILE *out, int SSP,
      int mflag, char LEB)
{
 float Anew, dtheta, Faxial, Faxialp, Io, N, Nx, Pcr, St,
      Scol, Scrip1, Scrip2, ScripS, Scrpl, Scrstcol,
      Ssk, Sskbd, tau, tau0, taucr, tauskbd, theta, Ysk[541], Yst[1081],
      gamF, gamM;
 int   i, mgcb, ncr, ngcb, mdrv;
 char GCB;
 putchar('\n');
 putchar('E');
 if (S.N>1080)
 {
   printf("\n\nNumber of stringers exceeds 1080,  ");
   printf("Number of stringers set to 1080");
   S.N = 1080;
   if (4 == SSP)
   {
     fprintf(out,"\n\nNumber of stringers exceeds 1080,  ");
     fprintf(out,"Number of stringers set to 1080");
   }
```

```
}
if (SSP == 4) fprintf(out,"\n\nSTRINGER PROPERTY CALCULATIONS");
stringer(&S,C,out,SSP);
S.Z *= S.MZs;
R.d = C.l/(R.N+1);
S.b = 2*pi*C.r/S.N;
Anew = 2*pi*C.r*C.t + S.A*S.N;
if (L.Pa<0)
{
  Faxial  = (L.F-L.Pa*pi*C.r*C.r)*L.sf/Anew;
  Faxialp = (L.F-L.Pa*pi*C.r*C.r)*L.sfp/Anew;
}
else
{
  Faxial  = (L.F*L.sf-L.Pa*pi*C.r*C.r)/Anew;
  Faxialp = (L.F*L.sfp-L.Pa*pi*C.r*C.r)/Anew;
}
Io = pi*C.t*C.r*C.r*C.r + S.N*S.I;
theta = 0;              dtheta = 2*pi/S.N;
for (i=0; i<(S.N/2+1); i++)
{
 Yst[i] = (C.r+S.Z)*cos(theta);
 Ysk[i] = C.r*cos(theta);
 Io += 2*S.A*Yst[i]*Yst[i];
 theta+=dtheta;
}
St = -L.M*L.sf*C.r/Io + Faxial;                /* tension stress */
if (SSP == 4)
{
 fprintf(out,"\n\nOVERALL CYLINDER CALCULATIONS");
 fprintf(out,"\n\nIo = %-22.0f  Tension Stress = f(Io) = %2.0f",Io,St);
 fprintf(out,"\nCross sectional area = %f",Anew);
 fprintf(out,"\nFaxial = f(F,Pa) = %8.1f  Faxialp = %8.1f",Faxial,Faxialp);
 fprintf(out,"\nRing Spacing = %-12.4f  Stringer Spacing = %7.4f",R.d,S.b);
 fprintf(out,"\n\nSKIN BUCKLING STRESS AND MAX SHEAR CALCULATIONS");
}
G[0] = skinbuck(S,R.d,M,L,C,Faxialp,Io,out,&Scrpl,&Sskbd,SSP,&Ssk,&tau,&tau0,
           &taucr,&tauskbd,Ysk,Yst);
if (SSP == 4) fprintf(out,"\n\nSTRINGER CRIPPLING CALCULATIONS\n");
getScrip(S,M,out,&Scrip1,&Scrip2,&ScripS,SSP,&mdrv,C.l,mflag,LEB);
if (SSP == 4) fprintf(out,"\n\nSTRINGER COLUMN BUCKLING CALCULATIONS");
G[1] = colstress(&Anew,S,R.d,M.E,L,C,Faxial,Io,out,&St,&Scol,ScripS,Scrpl,
           &Scrstcol,Sskbd,SSP,taucr,tauskbd,Ysk,Yst);
GCB = 'N';
G[3] = Scol/Scrip1 -1.;
G[4] = Scol/Scrip2 -1.;
if (Scol<0)
{
 fprintf(out,"\n\n\nWhole Cylinder is in Tension due to pressure, this");
 fprintf(out," is not a buckling problem.\n\n");
 exit(1);
}
if (SSP == 4) fprintf(out,"\n\nGENERAL CYLINDER BUCKLING CALCULATIONS");
G[1] = GCBcalc(R,S,M,C,L,G,out,SSP,&mgcb,&ngcb,&Nx,&Pcr,&N,Io,Faxial,&ncr,
           &GCB,&gamF,&gamM);
if (4 == SSP) fprintf(out,"\n\nVON MISES STRESS CHECK\n");
```

```
 G[2]=stresscheck(Anew,S.A,S.N,L.Ph,C.r,C.t,Faxial,out,St,Scol,SSP,tau,tau0,
        L.sf)/M.Scy - 1.;
 *OBJ= (pi*2*C.r*C.t*C.l +S.A*C.l*S.N +R.A*R.N*2*pi*(C.r+R.Z))*M.rho +C.fwt;
 if ( (SSP == 4) || (SSP == 5) )
   finalout(R,S,M,L,C,G,gamF,gamM,GCB,mgcb,N,ncr,ngcb,Nx,*OBJ,out,Pcr,Scol,
      Scrip1,Scrip2,Scrstcol,LEB,mdrv,Io,Faxial);
}  /*********************************************************** end eval  */


/*****************************************************************************/
/*************************    file CIO.C          ************************/
/*****************************************************************************/
#include "ct.h"

int iinput(FILE *infile)
{
 char hold[80];                          int i = 0;
 fscanf(infile," %d",&i);  fgets(hold,80,infile);    return(i);
}  /*****************************************************  end iinput  */


float finput(FILE *infile)
{
 char hold[80];                          float f = 0;
 fscanf(infile," %f",&f);    fgets(hold,80,infile);  return(f);
}  /*****************************************************  end finput  */


char cinput(FILE *infile)
{
 char hold[80];                          char c = '0';
 fscanf(infile," %1s",&c);    fgets(hold,80,infile); return(c);
}   /*****************************************************  end cinput  */


void readinput(char Title[], FILE *in, long int *IPRINT, long int *METHOD,
      int *SSP, struct material *M, struct cylinder *C, struct stringer *S,
      struct ring *R, struct load *L, float XL[], float X[], float XU[],
      float Tol[], int *mflag, char *LEB)
{
  fgets(Title,80,in);
  *IPRINT = iinput(in);   *METHOD = iinput(in);   *SSP = iinput(in);
  M->nu = finput(in);     M->E = finput(in);      M->G = finput(in);
  M->rho = finput(in);    M->Stu = finput(in);    M->Scy = finput(in);
  C->r = finput(in);      C->l = finput(in);      C->fwt = finput(in);
  S->stype = cinput(in); *LEB = cinput(in);       *mflag = iinput(in);
  S->alp = finput(in);    S->ll = finput(in);     S->MZs = finput(in);
  R->A = finput(in);      R->I = finput(in);      R->Z = finput(in);
  R->J = finput(in);      R->N = finput(in);      L->F = finput(in);
  L->M = finput(in);      L->V = finput(in);      L->Pa = finput(in);
  L->Ph = finput(in);     L->sf = finput(in);     L->sfp = finput(in);
  XL[0] = finput(in);  X[0] = finput(in);
                  XU[0] = finput(in);  Tol[0] = finput(in);
```

```
  XL[1] = finput(in);  X[1] = finput(in);
                  XU[1] = finput(in);  Tol[1] = finput(in);
  XL[2] = finput(in);  X[2] = finput(in);
                  XU[2] = finput(in);  Tol[2] = finput(in);
  XL[3] = finput(in);  X[3] = finput(in);
                  XU[3] = finput(in);  Tol[3] = finput(in);
  XL[4] = finput(in);  X[4] = finput(in);
                  XU[4] = finput(in);  Tol[4] = finput(in);
  XL[5] = finput(in);  X[5] = finput(in);
                  XU[5] = finput(in);  Tol[5] = finput(in);
}       /****************************** end readinput ***********/


void pinput(struct stringer S, struct ring R, struct material M,
      struct load L, struct cylinder C, long int IPRINT, long int METHOD,
      FILE *out, int SSP, float X[], float XL[], float XU[], int mflag,
      char LEB)
{
 fprintf(out,"\n\n********************** INPUT VALUES *****************");
 fprintf(out,"\nFLAGS");
 fprintf(out,"\n     IPRINT = %d",IPRINT);
 switch ( (int)IPRINT)
 {
   case 0: fprintf(out,"     No screen output by DOT"); break;
   case 1: fprintf(out,"     initial and final DOT output to screen"); break;
   case 2:
       fprintf(out,"     initial/final + OBJ & X[] output to screen");
       break;
   case 3:
       fprintf(out,"     initial/final + OBJ & X[] + G[] output to screen");
       break;
   case 4:
      fprintf(out,"     init./final + OBJ & X[] + G[] + grads. to screen");
      break;
   case 5:
      fprintf(out,"     init./final + OBJ & X[] + G[] + grads. to screen");
      break;
   case 6:
      fprintf(out,"     init./final+OBJ & X[]+G[]+grads.+S+X scales & S info");
      break;
   case 7:
      fprintf(out,"     init./final+OBJ & X[]+G[]+grads.+S+X scales & S info");
      break;
 }
 fprintf(out,"\n     METHOD = %d",METHOD);
 if (2 == METHOD) fprintf(out,"     Sequential Linear Programming");
          else fprintf(out,"     Modified Method of Feasible Directions");
 fprintf(out,"\n     SSP = %d   ",SSP);
 switch (SSP)
 {
  case 0: fprintf(out,"     Final CORSS output only"); break;
  case 1: fprintf(out,"     Final calculations and final output"); break;
  case 2: fprintf(out,"     Optimization history and final output"); break;
  case 3: fprintf(out,"     Optimization history, final calculations and");
        fprintf(out," final output");
```

```
}
fprintf(out,"\n\nMaterial Properties");
fprintf(out,"\n     nu    = %-6.3f            Poisson's Ratio",M.nu);
fprintf(out,"\n     E     = %-11.3E        Young's Modulus",M.E);
fprintf(out,"\n     SM    = %-11.3E        Shear Modulus",M.G);
fprintf(out,"\n     rho   = %-6.3f            Density",M.rho);
fprintf(out,"\n     Stu   = %-8.1f          Ultimate Tensile Stress",M.Stu);
fprintf(out,"\n     Scy   = %-8.1f          Yield Compressive Stress",M.Scy);
fprintf(out,"\n\nCylinder Geometry");
fprintf(out,"\n     r    =  %-7.2f           Radius",C.r);
fprintf(out,"\n     l    =  %-7.2f           Length",C.l);
fprintf(out,"\n     fwt  =  %-7.2f           Additional Weight",C.fwt);
fprintf(out,"\n\nStringer Parameters");
fprintf(out,"\n     stype = %c                ",S.stype);
if (S.stype == 'H')
{
  fprintf(out,"Hat Stringers\n     Local Elastic Buckling");
  if ('N'==LEB) fprintf(out," NOT ALLOWED"); else fprintf(out," IS ALLOWED");
  fprintf(out,"\n     alp    = %-5.1f            Leg angle",S.alp);
  fprintf(out,"\n     l1     = %-6.3f            Stringer/skin length",S.l1);
}
else
{
  fprintf(out,"I Stringers\n     Local Elastic Buckling");
  if ('N'==LEB) fprintf(out," NOT ALLOWED"); else fprintf(out," IS ALLOWED");
  fprintf(out,"\n     increase m from 1");
  if (0==mflag)
    fprintf(out," while coupled buckling stress decreases");
  else
    fprintf(out," to %d",mflag);
  fprintf(out,"\n     alp    = %-5.3f            Web Angle",S.alp);
  fprintf(out,"\n     l1     = %-4.3f            bottom flange thickness",S.l1);
}
fprintf(out,"\n     MZs    = %-2.0f             Stringers ",S.MZs);
if (-1 == S.MZs) fprintf(out,"internal"); else fprintf(out,"external");
fprintf(out,"\n\nRing Parameters");
if (0 == R.N) fprintf(out,"\n     No Rings");
 else
 {
   fprintf(out,"\n     Ar    = %-7.4f          Cross sectional area",R.A);
   fprintf(out,"\n     Ir    = %-8.5f         Moment of inertia",R.I);
   fprintf(out,"\n     Zr    = %-7.4f          Neutral Axis Distance",R.Z);
   fprintf(out,"\n     Nrng  = %-2.0f               Number of Rings",R.N);
 }
fprintf(out,"\n\nLoads");
fprintf(out,"\n     F    = %-9.1f          Axial Compression",L.F);
fprintf(out,"\n     M    = %-11.4G         Bending Moment",L.M);
fprintf(out,"\n     V    = %-8.1f          Shear force",L.V);
fprintf(out,"\n     Pa   = %-6.3f            Axial pressure component",L.Pa);
fprintf(out,"\n     Ph   = %-6.3f            Hoop pressure component",L.Ph);
fprintf(out,"\n     sf   = %-5.2f            Safety factor",L.sf);
fprintf(out,"\n     sfp  = %-5.2f            Plate buckling safety factor",L.sfp);
fprintf(out,"\n\nDesign Variables");
fprintf(out,"\nVar.     Minimum       Initial       Maximum      Name");
fprintf(out,"\nh %14.4f  %12.4f  %12.4f      Stringer height",
                              XL[0],X[0],XU[0]);
```

```
 fprintf(out,"\n12 %13.4f  %12.4f  %12.4f      Top flange ",XL[1],X[1],XU[1]);
 if ('H' == S.stype) fprintf(out,"length"); else fprintf(out,"thickness");
 fprintf(out,"\ntst %12.4f  %12.4f  %12.4f      ",XL[2],X[2],XU[2]);
 if ('H' == S.stype) fprintf(out,"Stringer thickness");
           else fprintf(out,"Web thickness");
 fprintf(out,"\nNst %12.1f  %12.1f  %12.1f      Number of stringers",
                                   XL[3],X[3],XU[3]);
 fprintf(out,"\nt   %12.4f  %12.4f  %12.4f      Skin thickness",
                                 XL[4],X[4],XU[4]);
 fprintf(out,"\nW   %12.4f  %12.4f  %12.4f      ",XL[5],X[5],XU[5]);
 if ('H' == S.stype) fprintf(out,"Top Flange Thickness");
           else fprintf(out,"Stringer Width");
 fprintf(out,"\n\n");
 fflush(out);
}       /******************************** pinput ***************/



void finalout(struct ring R, struct stringer S, struct material M,
      struct load L, struct cylinder C, float G[], float gamF, float gamM,
      char GCB, int mgcb, float N, int ncr, int ngcb, float Nx, float obj,
      FILE *out, float Pcrush, float Scol, float Scrip1, float Scrip2,
      float Scrstcol, char LEB, int mdrv, float Io, float Faxial)
{
 putchar('F');
 fprintf(out,"\n\n------------------------------------------------------------");
 fprintf(out,"----------------------");
 fprintf(out,"\n\nCylinder Weight = %3.1f\n (Skin:",obj);
 fprintf(out," %3.1f, Stringers:
%3.1f",(pi*2*C.r*C.t*C.l*M.rho),(S.A*C.l*S.N*M.rho));
 fprintf(out,", Rings: %3.1f, Flanges: %3.1f)",(R.A*R.N*2*pi*(C.r+R.Z)*M.rho),C.fwt);
 fprintf(out,"\n\nh    = %6.4f,     Stringer Height",S.h);
 fprintf(out,"\n12  = %6.4f,     ",S.l2);
 if (S.stype == 'H') fprintf(out,"Hat stringer top flange length");
 if (S.stype == 'I') fprintf(out,"I top flange thickness");
 fprintf(out,"\ntst = %6.4f,     ",S.t);
 if (S.stype == 'H') fprintf(out,"Hat stringer thickness");
 if (S.stype == 'I') fprintf(out,"I stringer web thickness");
 fprintf(out,"\nNst = %6.1f,     Number of Stringers, b = %6.4f",S.N,S.b);
 fprintf(out,"\nt   = %6.4f,     Skin Thickness",C.t);
 fprintf(out,"\nW   = %6.4f,     ",S.W);
 if ('H' == S.stype) fprintf(out,"Hat stringer top flange thickness");
 if ('I' == S.stype) fprintf(out,"I stringer width");
 fprintf(out,"\n\nStringer: I = %f, J = %f, Z = %f, A = %f",S.I,S.J,S.Z,S.A);
 fprintf(out,"\nEnd Ring I should be at least ");
 fprintf(out,"%g", .172*Scrstcol*(S.A+S.b*C.t)*C.r*C.r*C.r/R.d/M.E );
 fprintf(out,"\nEnd Ring Area should be at least");
 fprintf(out," %g*(r+Z)",4*pi*pi*S.I*C.r/S.b/pow(R.d,3));
 fprintf(out,"\n\t\t\t\t\t\t\t\t  G[] value");
 fprintf(out,"\nSkin: (Shear ratio)**2 + Stress ratio\t\t = %7.5f < 1
%11.5f",G[0]+1,G[0]);
 fprintf(out,"\n\nApplied Column Stress (SF = %4.2f)                = %11.1f",
                                                  L.sf,Scol);
 if ('I'==S.stype)
 {
   fprintf(out,
```

```
    "\n    Critical Coupled Buckling Stress (m = %3d)    = %8.1f",mdrv,Scrip2);
  fprintf(out,"%15.5f",G[4]);
  if ('N'==LEB)
    fprintf(out,
      "\n    Critical Flange Elastic Buckling Stress      = %8.1f",Scrip1);
  else
    fprintf(out,
      "\n    Critical Stringer Crippling Stress           = %8.1f",Scrip1);
  fprintf(out,"%15.5f",G[3]);
}
else
{
  if ('N'==LEB)
  {
    fprintf(out,
      "\n    Critical Local Buckling Stress (flange)      = %8.1f",Scrip1);
    fprintf(out,"%15.5f",G[3]);
    fprintf(out,
      "\n    Critical Local Buckling Stress (web)         = %8.1f",Scrip2);
    fprintf(out,"%15.5f",G[4]);
  }
  else
  {
    fprintf(out,
      "\n    Critical Stringer Crippling Stress           = %8.1f",Scrip1);
    fprintf(out,"%15.5f",G[3]);
  }
}
fprintf(out,"\n    Critical Column Stress ");
if ('N' == GCB)
{
  fprintf(out,"                        = %8.1f",Scrstcol);
  fprintf(out,"%15.5f",G[1]);
}
else fprintf(out,"controlled by General Cylinder Buckling");
fprintf(out,"\n\nApplied Von Mises Stress (SF = %4.2f)              = %11.1f",
                                          L.sf,(G[2]+1)*M.Scy);
fprintf(out,"\n    Yield Compressive Stress                    = %8.1f",
                                              M.Scy);
fprintf(out,"%15.5f",G[2]);
if ('N' == GCB)
{
  fprintf(out,"\n\nGeneral Cylinder Buckling controlled by ");
  fprintf(out,"Critical Column Buckling");
}
else
{
  fprintf(out,"\n\nCylinder: Line Load ratio + Pressure ratio");
  fprintf(out,"       = %7.5f < 1 %11.5f",G[1]+1,G[1]);
  fprintf(out,"\nApplied Cylinder Line Load (SF = %4.2f)           = %11.1f",
      L.sf,( L.M*L.sf*C.r/Io*(S.A/S.b+C.t) + Faxial*(S.A/S.b+C.t) ) );
  fprintf(out,"\nKnockdown Adjusted Line Load                    = %11.1f",
                                          N);
  fprintf(out,"\n    Critical General Cylinder Buckling Line Load = %8.1f",
                                              Nx);
  fprintf(out,"\n    Axial half waves, m = %i, Hoop waves,  n = %i",
```

```
                                        mgcb,ngcb);
   fprintf(out,"\n    Axial   Knockdown Factor, gammaF = %6.4f",gamF);
   fprintf(out,"\n    Bending Knockdown Factor, gammaM = %6.4f",gamM);
   if ( (mgcb != 1) && (mgcb == R.N+1) )
   {
    fprintf(out,"\nAxial half waves = Number of Cylinder segments");
    fprintf(out,"\n    Rings may not support general buckling");
   }
 }
 if (L.Ph<0.0)
 {
   fprintf(out,"\nApplied Crushing Hoop Pressure (SF = %4.2f)       = %16.3f",
                                              L.sf,-L.Ph*L.sf);
   fprintf(out,"\n    Critical Buckling Pressure                   = %8.3f",
                                               Pcrush);
   fprintf(out,"\n    Hoop waves, n = %i",ncr);
 }
 if ((Nx*(S.A/S.b+C.t))>M.Scy)
   {
     fprintf(out,"\n    If load increases over limit load, critical");
     fprintf(out,"\n      buckling will decrease due to plasticity.");
   }
}   /************************************************** end finalout */



/*************************************************************************/
/*************************    file COLSTRESS.C    ************************/
/*************************************************************************/
#include "ct.h"

float colstress(float *Anew, struct stringer S, float d, float E,
      struct load L, struct cylinder C, float Faxial, float Io, FILE *out,
      float *St, float *Scol, float Scrip, float Scrpl, float *Scrstcol,
      float Sskbd, int SSP, float taucr, float tauskbd, float Ysk[],
      float Yst[])
{
  float be,Ise,radg,ScrstcolJE,We0,tmp;
  putchar('C');
  *Scol = L.M*L.sf*C.r/Io + Faxial;
  be = S.b;
  if (L.sfp<L.sf)
  {
    if (L.sf/L.sfp*Sskbd/Scrpl+pow(L.sf/L.sfp*tauskbd/taucr,2) > 1)
    {
      *Scol = loopI(Anew,S,L.M,L.sf,E,C,Faxial,Io,out,St,Scrpl,SSP,
                  &We0,Ysk,Yst);
      if (S.stype == 'H')
      {
      if ( (tmp=S.l1+S.l2+2*S.la) < 2*We0 ) be=tmp;     /* between legs */
        else                                be=2*We0;
      if ( (tmp=S.b-S.l1-2*S.la-S.l2) < 2*We0 ) be+=tmp;
        else                                      be+=2*We0;
      }
      else if ( S.b > 2*We0 ) be=2*We0;
    }
```

```
  }
  Ise = be*C.t*C.t*C.t/12. + S.I + S.A*S.Z*S.Z - S.A*S.A*S.Z*S.Z/(S.A+be*C.t);
  if (Ise<0) Ise = 0.00001;
  radg = sqrt(Ise/(S.A+be*C.t));
  if (d/radg <= pi*sqrt(2*E/Scrip) )
  {
    *Scrstcol = Scrip - Scrip*Scrip/4/pi/pi/E*d*d/radg/radg;
    if (SSP == 4)
      fprintf(out,"\n\nJohnson-Euler Column Buckling (Scrip = %g) = %g",
                  Scrip,*Scrstcol);
  }
  else
  {
    *Scrstcol = pow(pi*radg/d,2)*E;
    if (SSP == 4)
      fprintf(out,"\n\nEuler Column Buckling = %g",*Scrstcol);
  }
  if (SSP == 4)
  {
    fprintf(out,"\nApplied Stress = %8.1f",*Scol);
    fprintf(out,"\nEffective skin width on stringer with max stress = %f",be);
    fprintf(out,"\nStringer+Skin I = %6.4f, radius of gyration = %6.4f",
                                              Ise,radg);
  }
  return( *Scol/(*Scrstcol) -1. );
}  /************************************************  end colstress      */



float loopI(float *Anew, struct stringer S, float M, float sf, float E,
      struct cylinder C, float Faxial, float Io, FILE *out, float *St,
      float Scrpl, int SSP, float *We0, float Ysk[], float Yst[])
{
  float Acyl, I, Iold, ybar;
  int count;
  count = 0;        ybar = 0;          I = Io/2;          Iold = Io;
  Acyl = 2*pi*(C.r-C.t/2)*C.t+S.N*S.A;
  *Anew = Acyl;
  while( fabs( (Iold-I)/Iold ) > .001 )
  {
    Iold = I;                              count++;
    newI(Acyl,Anew,S,M,sf,C.t,E,Faxial,&I,Io,out,Scrpl,SSP,We0,
            &ybar,Ysk,Yst);
    if (count == 20)
    {
      fprintf(out,"\nDid not converge on a Cylinder I in 20 iterations,");
      fprintf(out," I = %2.0f, I set to 1/2 Io =",I);
      I = Io/2;
      fprintf(out," %2.0f, t = %6.4f, Nst = %5.1f",I,C.t,S.N);
      break;
    }
  }
  *St = -M*sf*(C.r-ybar)/I + Faxial*Acyl/(*Anew);
  if (SSP == 4)
  {
    fprintf(out,"\n\nNumber of iterations to converge on I, %d",count);
```

```
    fprintf(out,"\n\nTension load (adjusted for buckled skin) = %8.1f",*St);
    newI(Acyl,Anew,S,M,sf,C.t,E,Faxial,&I,Io,out,Scrpl,6,We0,&ybar,Ysk,Yst);
  }
  return( M*sf*(C.r+ybar)/I + Faxial*Acyl/(*Anew) );
}  /*************************************************      end loopI       */


void newI(float Acyl, float *Anew, struct stringer S, float M, float sf,
      float t, float E, float Faxial, float *I, float Io, FILE *out,
      float Scrpl, int SSP, float *We0, float *ybar, float Ysk[],
      float Yst[])
{
  float A,Slb,Ssk[541+1],Sst[541+1],sumA,sumAy,sumAyy,We[541+1],tmp;
  int i,N;
  N = S.N/2;                                  Slb = .9*Scrpl;
  if (SSP == 6)
  {
    fprintf(out,"\n\nStress Carried in Buckled Skin, Slb = %7.1f",Slb);
    fprintf(out,"\n\n  I    Y-Stringer        Y-skin   Stress-str.");
    fprintf(out,"   Stress-skin    Eff. Width");
  }
  for (i=0; i<N/2+1; i++)
  {
    Sst[i] = Faxial*Acyl/(*Anew) + M*sf/(*I)*(Yst[i]+(*ybar));
    Ssk[i] = Faxial*Acyl/(*Anew) + M*sf/(*I)*(Ysk[i]+(*ybar));
    if (Sst[i] < 0) We[i] = S.b;
      else if ( (We[i] = .85*t*sqrt(E/Sst[i])) > S.b ) We[i] = S.b;
    if (SSP == 6) fprintf(out,"\n%3i %13.4f %13.4f %13.1f %13.1f %13.4f",
                          i,Yst[i],Ysk[i],Sst[i],Ssk[i],We[i]);
  }
  A = 0;       sumA = 0;       sumAy = 0;       sumAyy = 0;      We[N] = We[0];
  if (SSP == 6)
  {
    fprintf(out,"\n\n  I            Area         sum Area      sum Area*y");
    fprintf(out,"    sum Area*y*y skin/str.");
  }
  for (i=0; i<(N/2); i++)
  {
    if ( (S.stype == 'H') && (Sst[i] > 0) )
      if ( (tmp = S.12+2*S.1a+S.11) > (2*We[i]) )
      {
      A = 2*( t*(Sst[i] - Slb)/Sst[i]*(tmp-2*We[i]) );
      sumA += A;       sumAy += A*Yst[i];       sumAyy += A*Yst[i]*Yst[i];
      if (SSP == 6)
        fprintf(out,"\n%3i %15.3f %15.3f %15.3f %15.3f stringer",
                                      i,A,sumA,sumAy,sumAyy);
      }
    if (Ssk[i] > 0)
    {
      if (S.stype == 'H')
      {
      if (  (tmp=S.b-(S.12+2*S.1a+S.11)) > (We[i] + We[i+1]) )
      {
        A = 2*( t*(Ssk[i] - Slb)/Ssk[i]*(tmp-We[i]-We[i+1]) );
        sumA += A;     sumAy += A*Ysk[i];       sumAyy += A*Ysk[i]*Ysk[i];
```

```
        if (SSP == 6) fprintf(out,"\n%3i %15.3f %15.3f %15.3f %15.3f skin",
                                    i,A,sumA,sumAy,sumAyy);
      }
      }
      else if ( S.b > (We[i]+We[i+1]) )
      {
      A = 2*( t*(Ssk[i] - Slb)/Ssk[i]*(S.b-We[i]-We[i+1]) );
      sumA += A;      sumAy += A*Ysk[i];        sumAyy += A*Ysk[i]*Ysk[i];
      if (SSP == 6) fprintf(out,"\n%3i %15.3f %15.3f %15.3f %15.3f skin",
                                        i,A,sumA,sumAy,sumAyy);
      }
    }
  }
  *We0 = We[0];
  *Anew = Acyl-sumA;
  *ybar = sumAy/(*Anew);
  *I = Io - sumAyy - (*Anew)*(*ybar)*(*ybar);
  if (SSP==6) fprintf(out,
            "\n\n Ybar = %7.3f,   Cylinder I = %9.1f,     Anew = %9.1f",
            *ybar,*I,*Anew);
}   /*****************************************************  end newI    */


/*************************************************************************/
/*************************    file STRINGER.C   **************************/
/*************************************************************************/
#include "ct.h"

void stringer(struct stringer *S, struct cylinder C, FILE *out, int SSP)
{
  int i, na;
  float a[5],ad[5],add[5],alpha,d[5],ha,I[5],ix,sumad,sumadd,sumi,ybar,
      k1,k2,al,D;
  /* float iy, xbar; */
  putchar('S');
  alpha = S->alp*pi/180;
  if (S->stype == 'H')
  {
    na = 5;                 ha = (S->h-S->t-S->W)/cos(alpha);
    S->la = ha*sin(alpha);
    a[0] = S->l1*S->t;      a[1] = ha*S->t;             a[2] = S->l2*S->W;
    a[3] = a[1];            a[4] = a[0];
    d[0] = S->t/2;          d[1] = S->h/2;              d[2] = S->h-S->W/2;
    d[3] = d[1];            d[4] = d[0];
    I[0] = S->l1*pow(S->t,3)/12;
    I[1] = S->t*ha/12*
         ( ha*ha*cos(alpha)*cos(alpha) + S->t*S->t*sin(alpha)*sin(alpha) );
    I[2] = S->l2*pow(S->W,3)/12;
    I[3] = I[1];            I[4] = I[0];
    S->J = 4*pow( (S->h+C.t/2-S->W/2)*(S->l2+S->la) , 2.)
        /( (S->l2+2*S->la)/C.t + 2*ha/S->t + S->l2/S->W) ;
  }
  if (S->stype == 'I')
  {
    na = 3;             ha = (S->h-S->l2-S->l1)/cos(alpha);
    a[0] = S->W*S->l1; a[1] = S->t*ha;                  a[2] = S->W*S->l2;
```

```
    d[0] = S->l1/2;     d[1] = (S->h-S->l2-S->l1)/2+S->l1; d[2] = S->h-S->l2/2;
    I[0] = S->W*pow(S->l1,3)/12;
    I[1] = S->t*ha/12
           * (ha*ha*cos(alpha)*cos(alpha)+S->t*S->t*sin(alpha)*sin(alpha));
    I[2] = S->W*pow(S->l2,3)/12;
    k1 = S->W*S->l2*S->l2*S->l2
      *( 1./3. - .21 * S->l2 / S->W * (1. - pow( S->l2 / S->W ,4) / 12 ));
    k2 = (S->h-S->l2)*S->t*S->t*S->t
      *(1./3. - .105*S->t/(S->h-S->l2)*(1. - pow(S->t/(S->h-S->l2),4)/192));
    if (S->l2<S->t) a1 = S->l2/S->t*.15; else a1 = S->t/S->l2*.15;
    if (S->t>(2*S->l2)) D=S->t; else D = S->l2 + S->t*S->t/4./S->l2;
    S->J = k1+k2+a1*D*D*D*D;
  }
  S->A = 0;           sumad = 0;          sumadd = 0;          sumi = 0;
  for (i=0; i<na; i++)
  {
    ad[i] = a[i]*d[i];                          add[i] = ad[i]*d[i];
    S->A += a[i];       sumad += ad[i];         sumadd += add[i];   sumi += I[i];
  }
  ybar = sumad/(S->A);
  ix = sumi + sumadd - ybar*sumad;
  if (SSP == 4) stringerP(na,a,d,ad,add,I,S->A,ybar,ix,
                  "\nA = %f,  ybar = %f,  Ix = %f",out);

/*
  if (S->stype == 'H')
  {
    d[0] = S->l1/2;  d[1] = S->l1+(S->la)/2;
    d[2] = S->l1+(S->la)+S->l2/2;
    d[3] = S->l1+(S->la)+S->l2+(S->la)/2;
    d[4] = S->l1+2*(S->la)+S->l2+S->l1/2;
    I[0] = S->t*S->l1*S->l1*S->l1/12;
    I[1] = S->t*ha/12*( ha*ha*cos(pi/2-alpha)*cos(pi/2-alpha)
                  + S->t*S->t*sin(pi/2-alpha)*sin(pi/2-alpha) );
    I[2] = S->W*S->l2*S->l2*S->l2/12;
    I[3] = I[1];
    I[4] = I[0];
  }
  if (S->stype == 'I')
  {
    d[0] = S->W/2;        d[1] = S->W/2;                d[2] = S->W/2;
    I[0] = S->l1*pow(S->W,3)/12;
    I[1] = S->t*ha/12
           * ( ha*ha*cos(pi/2-alpha)*cos(pi/2-alpha)
           + S->t*S->t*sin(pi/2-alpha)*sin(pi/2-alpha));
    I[2] = S->l2*pow(S->W,3)/12;
  }
  sumad = 0;            sumadd = 0;                 sumi = 0;
  for (i=0; i<na; i++)
  {
    ad[i] = a[i]*d[i];   add[i] = ad[i]*d[i];
    sumad += ad[i];      sumadd += add[i];           sumi += I[i];
  }
  xbar = sumad/(S->A);
  iy = sumi + sumadd - xbar*sumad;
  if (SSP == 4) stringerP(na,a,d,ad,add,I,S->A,xbar,iy,
                  "\nA = %f,  xbar = %f,  Iy = %f",out);
```

```
*/
  S->I = ix;          /*   S->J = ix+iy;    */        S->Z = ybar+C.t/2.;
/***********************
S->J = ix/100;  */
}  /*****************************************************  end stringer  */


void stringerP(int na, float a[], float d[], float ad[], float add[],
            float I[], float A, float bar, float ia, char line[],
            FILE *out)
{
  int i;
  fprintf(out,"\n\n");
  for (i=0;i<na;i++) fprintf(out,"Area %i       ",i);    fprintf(out,"\n");
  for (i=0;i<na;i++) fprintf(out,"%f     ",a[i]);    fprintf(out,"Area\n");
  for (i=0;i<na;i++) fprintf(out,"%f     ",d[i]);       fprintf(out,"d\n");
  for (i=0;i<na;i++) fprintf(out,"%f     ",ad[i]);    fprintf(out,"A*d\n");
  for (i=0;i<na;i++) fprintf(out,"%f     ",add[i]); fprintf(out,"A*d*d\n");
  for (i=0;i<na;i++) fprintf(out,"%f     ",I[i]);         fprintf(out,"I");
  fprintf(out,line,A,bar,ia);
}



/*************************************************************************/
/*************************    file SKINBUCK.C   **************************/
/*************************************************************************/
#include "ct.h"

float skinbuck(struct stringer S, float d, struct material M,
      struct load L, struct cylinder C, float Faxialp, float Io, FILE *out,
      float *Scrpl, float *Sskbd, int SSP, float *Stsk, float *tau,
      float *tau0, float *taucr, float *tauskbd, float Ysk[], float Yst[])
{
  int i;
  float alpha,bpl,g0,g0max,K,K1,Kcr,Ksh,qmax,q[272],Ssk[272],Z;
  if (S.stype == 'I') bpl = S.b;
  if (S.stype == 'H')
    if ( (S.12+2*S.1a+S.11) > (S.b-S.12-2*S.1a-S.11) ) bpl = S.12+2*S.1a+S.11;
                                  else bpl = S.b-S.12-2*S.1a-S.11;
  Z = bpl*bpl/C.r/C.t*sqrt(1-M.nu*M.nu);
  Kcr = Kc(Z,C.r,C.t);
  Ksh = Ks(Z);
  if (Ksh < 5.4) Ksh = 5.4;
  if (Kcr < 4) Kcr = 4.;
  if (4==SSP)
  {
    if (C.r/C.t<100.)
      fprintf(out,"\n\nWARNING: r/t < 100, Kc based on r/t = 100");
    if (C.r/C.t>3000.)
      fprintf(out,"\n\nWARNING: r/t > 3000., Kc based on r/t = 3000");
    if (Z>20000.)
      fprintf(out,"\n\nWARNING: Z > 20000., Kc is extrapolated");
  }
  *Scrpl = Kcr*pi*pi*M.E/12/(1-M.nu*M.nu)*pow((C.t/bpl),2);
```

```
*taucr = Ksh*pi*pi*M.E/12/(1-M.nu*M.nu)*pow((C.t/bpl),2);
if (L.Ph>.001)
{
  alpha = L.Ph*C.r*C.r/M.E/C.t/C.t;
  K = 9*pow(C.t/C.r,.6)*(1+.21*alpha*pow(C.r/C.t,.6))/(1+3*alpha);
  K1 = .16*C.r/C.t*pow(C.t/d,1.3);
  *Scrpl += (K+K1)*M.E*C.t/C.r;
  *taucr += (  0    )   ;
  if (SSP == 4)
    fprintf(out,"\n\nPressure Stabilization: Alpha = %f, K = %f, K1 = %f\n",
                                                alpha,K,K1);
}
if (SSP == 4)
{
  fprintf(out,"\n\nScrpl = %8.1f,     taucr = %8.1f",*Scrpl,*taucr);
  fprintf(out,"\nbpl = %7.4f,     Kc = %8.4f,    Ks = %8.4f",bpl,Kcr,Ksh);
  fprintf(out,",     Z = %7.4f",Z);
}
q[0] = -L.V*L.sfp/Io*S.A/2*Yst[0];
Ssk[0] = L.M*L.sfp*C.r/Io + Faxialp;
g0max = fabs(Ssk[0])/(*Scrpl) + pow( (fabs(q[0])/C.t)/(*taucr) , 2 ) - 1.;
*Sskbd = fabs(Ssk[0]);
*tauskbd = fabs(q[0])/C.t;
if (SSP == 4)
{
  fprintf(out,"\n\n   N                q[I]                Ssk[I]");
  fprintf(out,"                  G[0]");
  fprintf(out,"\n   0 %20.3f %20.3f %20.3f",q[0],Ssk[0],g0max);
}
qmax = fabs(q[0]);
for (i=1; i<(S.N/4+2); i++)
{
  q[i] = q[i-1] - L.V*L.sfp/Io*S.A*Yst[i];
  Ssk[i] = L.M*L.sfp*Ysk[i]/Io + Faxialp;
  g0 = fabs(Ssk[i])/(*Scrpl) + pow( (fabs(q[i])/C.t)/(*taucr) , 2 ) - 1.;
  if (g0>g0max)
  {
    g0max = g0;
    *Sskbd = fabs(Ssk[i]);
    *tauskbd = fabs(q[i])/C.t;
  }
  if (fabs(q[i])>qmax) qmax = fabs(q[i]);
  if (SSP == 4) fprintf(out,"\n%4i %20.3f %20.3f %20.3f",i,q[i],Ssk[i],g0);
}
*tau = qmax/C.t;
*Stsk = Ssk[0];
*tau0 = q[0]/C.t;
if (SSP == 4)
{
  fprintf(out,"\n\nMax shear stress = %8.1f,   Max skin stress = %8.1f",
                                        *tau,*Stsk);
  fprintf(out,
    "\nSkin Buckling Drivers: Stress = %8.1f,     Shear Stress = %8.1f",
                                          *Sskbd,*tauskbd);
}
return(g0max);
```

```
}   /******************************************************** end skinbuck */


float Ks(float Z)
{
 return(    4.94002871              + 2.83295655E-1*Z
        - 8.48571232E-3*Z*Z       + 2.96028833E-4*Z*Z*Z
        - 5.59394768E-6*pow(Z,4) + 5.12790432E-8*pow(Z,5)
        - 1.79230370E-10*pow(Z,6) );
}



float Kc(float Z, float r, float t)
{
  float M,B;
  if (Z<4.) return (4.);
  if (r/t<100)
  {
    M = 0.;
    B = Kc100(Z);
  }
  else if (r/t < 300)
  {
    M = ( Kc300(Z) - Kc100(Z) )/200;
    B = Kc100(Z) - M*100;
  }
  else if (r/t < 500)
  {
    M = ( Kc500(Z) - Kc300(Z) )/200;
    B = Kc300(Z) - M*300;
  }
  else if (r/t < 1000)
  {
    M = ( Kc1000(Z) - Kc500(Z) )/500;
    B = Kc500(Z) - M*500;
  }
  else if (r/t < 1500)
  {
    M = ( Kc1500(Z) - Kc1000(Z) )/500;
    B = Kc1000(Z) - M*1000;
  }
  else
  {
    M = 0.;
    B = Kc1500(Z);
  }
  return( M*(r/t) + B );
}  /************************************************ end Kc *******/



float Kc100(float Z)
{
  float y = Z-4.;
```

```
  if (Z<40) return( -.000179569*y*y*y + .0154754*y*y + .0089413*y + 4.);
       else return(.4*Z);
}



float Kc300(float Z)
{
  float y = Z-4.;
  if (Z<40) return( -.000159922*y*y*y + .0140199*y*y + .00810034*y + 4.);
       else return(.375*Z);
}



float Kc500(float Z)
{
  float y = Z-4.;
  if (Z<40) return( -.000127178*y*y*y + .011594*y*y + .00669873*y + 4.);
       else return(Z/3.);
}



float Kc1000(float Z)
{
  float y = Z-4.;
  if (Z<40) return( -.000100982*y*y*y + .00965326*y*y + .00557741*y + 4.);
       else return(.3*Z);
}



float Kc1500(float Z)
{
  float y = Z-4.;
  if (Z<40) return( -6.1688E-5*y*y*y + .00674219*y*y + .00389547*y + 4.);
       else return(.25*Z);
}


/*************************************************************************/
/*************************    file LEBCRIP.C    **************************/
/*************************************************************************/
#include "ct.h"

float getScrip(struct stringer S, struct material M, FILE *out,
      float *Scrip1, float *Scrip2, float *ScripS, int SSP, int *mdrv,
      float l, int mflag, char LEB)
{
  float S1,S2,S3,S4,alpha;
  int m;
  putchar('C');
  alpha = S.alp*pi/180;
  if (S.stype == 'I')
  {
```

```
    S2 = findICBuc(mflag,mdrv,1,S.t,M.nu,S.h,S.12,S.W,M.E,SSP,out);
    if (4==SSP) fprintf(out,"\nCoupled Buckling Stress, %9.2f",S2);
    S3=.569311 / pow( sqrt(M.Scy/M.E*(S.W/2)/S.12), .812712 ) * M.Scy;
    if ((S.h-S.12-S.11)<=0) domainexit(S.12,S.11,S.h,out);
    S4=1.387194/pow(sqrt(M.Scy/M.E*((S.h-S.12-S.11)/cos(alpha))/S.t),.807179)
          * M.Scy;
    if (S3 > M.Stu) S3 = M.Stu;
    if (S4 > M.Stu) S4 = M.Stu;
    *ScripS = S1 = ( S3*S.12*S.W + S4*S.t*((S.h-S.12-S.11)/cos(alpha)) )
           / ( S.A - S.11*S.W );
    if (4==SSP)
      fprintf(out,"\nCrippling: Top Flange = %8.1f, Web = %8.1f",S3,S4);
    if ('N' == LEB )
    {
      S1=.456*pi*pi/12*M.E/(1-M.nu*M.nu)*pow(S.12/(S.W/2),2);
      if (4==SSP) fprintf(out,"\nFlange Elastic Buckling Stress, %9.2f",S1);
    }
  }
  else
  {
    if (S.12>S.t)
      S3 = 1.387194 / pow( sqrt(M.Scy/M.E*(S.12-S.t)/S.t), .807179 ) * M.Scy;
    else S3 = 0;
    S4 = 1.387194/ pow( sqrt(M.Scy/M.E*((S.h-S.W-S.t)/cos(alpha))/S.t),
.807179)*M.Scy;
    if (S3 > M.Stu) S3 = M.Stu;
    if (S4 > M.Stu) S4 = M.Stu;
    *ScripS = S1 = ( S3*S.W*S.12 + 2*S4*S.t*((S.h-S.W-S.t)/cos(alpha)) ) /
                ( S.A - S.11*S.t*2 );
    if (4==SSP)
      fprintf(out,"\n\nCrippling: Top Flange = %8.1f, Web = %8.1f",S3,S4);
    if ('N'==LEB)
    {
      if (S.12 == 0) S.12 = 1.E-6;
      S1 = 3.29*M.E/(1-M.nu*M.nu)*pow(S.W/(S.12-S.t),2);
      S2 = 3.29*M.E/(1-M.nu*M.nu)*pow(S.t/((S.h-S.W-S.t)/cos(alpha)),2);
      if (4==SSP)
      {
      fprintf(out,"\nCritical Local Elastic Buckling Stress:");
      fprintf(out,"\n     Top Flange = %9.2f, Web = %9.2f",S1,S2);
      }
    }
    else S2 = M.Stu;
  }
  if (4==SSP)
      fprintf(out,"\nWeighted Average Crippling Stress, %9.2f",*ScripS);
  *Scrip1 = S1;
  *Scrip2 = S2;
}  /*****************************************************  end getScrip */



float findICBuc(int mflag, int *mdrv, float l, float tst, float nu, float h,
      float 12, float W, float E, int SSP, FILE *out)
{
  float Sdrv,S,Slast;
```

```
  int i,m;
  m=0;
  S = 1E30;
  if (mflag==0)
  {
    do
    {
      m++;
      Slast = S;
      S = secant(m,1,tst,nu,h,12,W,E);
      if (4==SSP) fprintf(out,"\nm = %d, Critical Stress = %f",m,S);
    } while ((S-Slast)<0);
    *mdrv = m-1;
    Sdrv = Slast;
  }
  else
  {
    Sdrv = S;
    for (i=1; i<=mflag; i++)
    {
      S = secant(i,1,tst,nu,h,12,W,E);
      if (S<Sdrv) { Sdrv=S; *mdrv=i; }
      if (4==SSP) fprintf(out,"\nm = %d, Critical Stress = %f",i,S);
    }
  }
  return (Sdrv);
}       /********************************** findICBuc ***********/


float secant(int m, float l, float tst, float nu, float h, float l2, float W,
      float E)
{
  float Fj,hold,Fi,Smin,Smax,Si,Sj;
  int i,count;
  count=0;
  Smin = Sj = m*m*pi*pi*E*tst*tst/l/l/12/(1-nu*nu)+1.;
  Fj = G6eqn(Sj,(float)m,l,tst,nu,h,l2,W,E);
  Si = 2*Sj;
  Fi = G6eqn(Si,(float)m,l,tst,nu,h,l2,W,E);
  while ( ((Fi<0)&&(Fj<0)) || ((Fi>0)&&(Fj>0)) )
  {
    Si *= 2;
    Fi = G6eqn(Si,(float)m,l,tst,nu,h,l2,W,E);
  }
  Sj = Si/2;
  Smax = Si;
  Fj = G6eqn(Sj,(float)m,l,tst,nu,h,l2,W,E);
  do
  {
    if ( Si<Smin ) Si=Smax;
    hold = Si;
    Fi = G6eqn(Si,(float)m,l,tst,nu,h,l2,W,E);
    if ( (Si==Sj) || (Fi==Fj) ) break; else Si = Sj - Fj/( (Fi-Fj)/(Si-Sj) );
    Sj = hold;
    Fj = Fi;
```

```
    if (count++>20) break;
  } while ( fabs(Fi) > .001 );
  return (Si);
}       /************************************ secant **********/


float G6eqn(float S, float m, float l, float tst, float nu, float h,
      float l2, float W, float E)
{
  float M,Nx,D,k1,alpha,beta,b,I,Ab,k2,k3;
  M = m*m*pi*pi/l/l;
  Nx = S*tst;
  D = E*tst*tst*tst/12/(1-nu*nu);
  k1 = sqrt(Nx*M/D);
  alpha = sqrt(M+k1);
  beta = sqrt(-M+k1);
  b = h-l2;
  I = l2*W*W*W/12;
  Ab = W*l2;
  k2 = E*I*sin(beta*b)*M*M;
  k2 += D*(( beta*beta*beta+beta*(2-nu)*M)*cos(beta*b));
  k2 -= Ab*S*sin(beta*b)*M;
  k2 *= (alpha*alpha-nu*M)*sinh(alpha*b);
  k3 = E*I*sinh(alpha*b)*M*M;
  k3 -= D*((alpha*alpha*alpha-alpha*(2-nu)*M)*cosh(alpha*b));
  k3 -= Ab*S*sinh(alpha*b)*M;
  k3 *= (beta*beta+nu*M)*sin(beta*b);
  return(k2+k3);
}


void domainexit(float l2, float l1, float h, FILE *out)
{
  printf("\a\a\n\nTop flange thickness + bottom flange thickness");
  printf(" > stringer height");
  printf("\n%20f + %23f > %15f",l2,l1,h);
  printf("\nThis will cause a SQRT domain error, please make sure");
  printf("\nthe smallest Stringer Height is greater then the");
  printf("\nBottom Flange Thickness plus the largest Top Flange");
  printf(" Thickness");
  fprintf(out,"\n\nTop flange thickness + bottom flange thickness");
  fprintf(out," > stringer height");
  fprintf(out,"\n%20f + %23f > %15f",l2,l1,h);
  fprintf(out,"\nThis will cause a SQRT domain error, please make sure");
  fprintf(out,"\nthe smallest Stringer Height is greater then the");
  fprintf(out,"\nBottom Flange Thickness plus the largest Top Flange");
  fprintf(out," Thickness");
  fcloseall();
  exit(2);
}


/*****************************************************************************/
```

```
/*************************    file GENCYL.C       ************************/
/*************************************************************************/
#include "ct.h"

float GCBcalc(struct ring R, struct stringer S, struct material M,
      struct cylinder C, struct load L, float G[], FILE *out, int SSP,
      int *mgcb, int *ngcb, float *Nx, float *Pcr, float *N,
      float Io, float Faxial, int *ncr, char *GCB, float *gamF,
      float *gamM)
{
  struct stiffness egdck;
  float G1 = 1000;
  putchar('G');
  getstiffnesses(R,S,M,C,&egdck,out,SSP);
  if ((G[0]<0) || (L.sf<=L.sfp))      /* if skin doesn't buckle */
  {
    gencyl(C.l,R.N,C.r,egdck,mgcb,ngcb,Nx,out,SSP);
    *gamM = gammaM(egdck,C.r);
    *gamF = gammaF(egdck,C.r);
    *N  =   (L.M*L.sf*C.r/Io*(S.A/S.b+C.t))/(*gamM)
        + Faxial*(S.A/S.b+C.t)/(*gamF);
    if (4==SSP)
    {
      fprintf(out,"\n\nCritical line load is Ncr = %f",*Nx);
      fprintf(out,"\nKnockdown Adjusted Line Load is %f",*N);
      fprintf(out,"\nat axial waves m = %d, and hoop waves n = %d",*mgcb,*ngcb);
      fprintf(out,"\ngammaF = %f,  gammaM = %f",*gamF,*gamM);
    }
    G1 = (*N)/(*Nx) - 1;
    if (L.Ph<0.0)
    {
      *Pcr = Pcrcalc(egdck,C.r,C.l,ncr,out,SSP);
      G1 += -L.Ph*L.sf/(*Pcr);
      if (4==SSP) fprintf(out,"\n\nCritical crushing pressure Pcr = %f",*Pcr);
    }
    if (G1<G[1])
    {
      G[1] = G1;
      *GCB = 'Y';
      if (4 == SSP)
      fprintf(out,
      "\nGeneral Cylinder Buckling provides support above Column Buckling");
    }
    else
    if (4==SSP)
      fprintf(out,
      "\nColumn Buckling provides support above General Cylinder Buckling");
    return ( G[1]);
  }
  if (4==SSP)
  {
    fprintf(out,"\nGeneral Cylinder Buckling not checked because of");
    fprintf(out," buckled skin");
  }
  return (G[1]);
}       /************************************ end GCBcalc ********/
```

```
void gencyl(float l, float Nr, float r, struct stiffness egdck, int *mmin,
      int *nmin, float *Nx, FILE *out, int SSP)
{
  float Nm,Nold,Nmt;
  int m,n,mm,nm;
  m = 1;  n = 1;
  mm = 1; nm = 1;
  *mmin = 1; *nmin = 1;
  Nm = l*l/m/m/pi/pi*getA(egdck,r,l,m,n,out,SSP);
  Nold = Nm*2;
  *Nx = Nm;
  while ((*Nx)<Nold)
  {
    Nold = (*Nx);
    n=4;
    Nm = l*l/m/m/pi/pi*getA(egdck,r,l,m,n,out,SSP);
    do
    {
      if ( (Nmt = l*l/m/m/pi/pi*getA(egdck,r,l,m,n,out,SSP)) < Nm )
      {  Nm = Nmt;      mm = m;      nm = n;      }
      n++;
    } while (Nm == Nmt);
    if (Nm < (*Nx) )  {  *Nx = Nm;      *mmin = mm;      *nmin = nm;      }
    m++;
  }
  if (SSP == 4)
  {
    fprintf(out,"\n\nCritical Line Load, m=%d, n=%d, Ncr=%f",*mmin,*nmin,*Nx);
    Nmt = getA(egdck,r,l,*mmin,*nmin,out,6);
  }
  if (  ( (SSP==4) || (SSP==5) ) && ( (*mmin!=1) && (m==Nr+1) )  )
  {
    fprintf(out,"\n\nWARNING! for the critical load case, there is one");
    fprintf(out,"axial half wave\nper section between rings.  The rings");
    fprintf(out,"may not help\nsupport general cylinder buckling.");
  }
}  /*************************************************  end gencyl      */


float Pcrcalc(struct stiffness egdck, float r, float l, int *ncr, FILE *out,
      int SSP)
{
  float P,Pcr,Pold;
  int n;
  Pcr = .75*r/1/1*getA(egdck,r,l,1,1,out,SSP);
  for (n=2; n<150; n++)
  {
    if ( (P = .75*r/n/n*getA(egdck,r,l,1,n,out,SSP)) < Pcr )
    {
      Pcr = P;   *ncr = n;
    }
  }
```

```
  if (SSP == 4)
  {
    fprintf(out,"\n\nCritical Pressure");
    P = getA(egdck,r,l,1,*ncr,out,6);
    fprintf(out,
      "\nCritical Pressure = %6.2f with %i circumferential waves",Pcr,*ncr);
    if ( *ncr>=149 )
    {
      fprintf(out,"\nWARNING: Program may not have calculated lowest ");
      fprintf(out,"critical pressure.  ncr may be greater than %d",*ncr);
    }
  }
  return(Pcr);
}  /************************************************  end Pcrcalc      */



float getA(struct stiffness ES, float r, float l, int m, int n, FILE *out,
        int SSP)
{
  float A11,A22,A33,A12,A21,A23,A32,A31,A13,detA3x3,detA2x2;
  A11 = ES.Ex*m*m*pi*pi/l/l + ES.Gxy*n*n/r/r;
  A22 = ES.Ey*n*n/r/r + ES.Gxy*m*m*pi*pi/l/l;
  A33 = ES.Dx*pow( m*pi/l , 4 ) + ES.Dxy*pow( m*pi*n/l/r , 2 ) +
      ES.Dy*pow( n/r , 4 ) + ES.Ey/r/r + 2*ES.Cy/r*n*n/r/r +
      2*ES.Cxy/r*m*m*pi*pi/l/l;
  A12 = (ES.Exy+ES.Gxy)*m*pi/l*n/r;
  A21 = A12;
  A23 = (ES.Cxy + 2*ES.Kxy)*m*m*pi*pi/l/l*n/r + ES.Ey*n/r/r +
      ES.Cy*pow( n/r , 3 );
  A32 = A23;
  A31 = ES.Exy/r*m*pi/l + ES.Cx*pow(m*pi/l,3) +
      (ES.Cxy+2*ES.Kxy)*m*pi/l*n*n/r/r;
  A13 = A31;
  detA3x3 = A11*A22*A33 + A12*A23*A31 + A13*A21*A32
        - A13*A22*A31 - A11*A23*A32 - A12*A21*A33;
  detA2x2 = A11*A22 - A12*A21;
  if (SSP == 6)
  {
    fprintf(out,"\n\n A = | %12.4f     %12.4f     %12.4f |",A11,A12,A13);
    fprintf(out,"\n     | %12.4f     %12.4f     %12.4f |",A21,A22,A23);
    fprintf(out,"\n     | %12.4f     %12.4f     %12.4f |",A31,A32,A33);
    fprintf(out,"\n\ndeterminant of A(3x3) = %G",detA3x3);
    fprintf(out,"\ndeterminant of A(2x2) = %G",detA2x2);
  }
  return( detA3x3/detA2x2 );
}   /*********************************************************  end getA  */



void getstiffnesses(struct ring R, struct stringer S, struct material M,
                struct cylinder C, struct stiffness *egdck, FILE *out,
                int SSP)
{
  float Er, Es, Gr, Gs;
  Es = M.E;        Er = M.E;        Gs = M.G;        Gr = M.G;
```

```
  if (0 == R.N) R.d = 1.0E35;
  egdck->Ex = M.E*C.t/(1-M.nu*M.nu) + Es*S.A/S.b;
  egdck->Ey = M.E*C.t/(1-M.nu*M.nu) + Er*R.A/R.d;
  egdck->Exy = M.nu*M.E*C.t/(1-M.nu*M.nu);
  egdck->Gxy = M.E*C.t/2/(1+M.nu);
  egdck->Dx = M.E*C.t*C.t*C.t/12/(1-M.nu*M.nu) + Es*S.I/S.b + S.Z*S.Z*Es*S.A/S.b;
  egdck->Dy = M.E*C.t*C.t*C.t/12/(1-M.nu*M.nu) + Er*R.I/R.d + R.Z*R.Z*Er*R.A/R.d;
  egdck->Dxy = M.E*C.t*C.t*C.t/6/(1+M.nu) + Gs*S.J/S.b + Gr*R.J/R.d;
  egdck->Cx = S.Z*Es*S.A/S.b;
  egdck->Cy = R.Z*Er*R.A/R.d;
  egdck->Cxy = C.t*C.t*C.t/12/(1-M.nu*M.nu);
  egdck->Kxy = egdck->Cxy;
  if (SSP == 4)
  {
    fprintf(out,"\n\nEx  = %14.4f  Ey  = %14.4f  Exy = %14.4f",
            egdck->Ex,egdck->Ey,egdck->Exy);
    fprintf(out,"\nDx  = %14.4f  Dy  = %14.4f  Dxy = %14.4f",
            egdck->Dx,egdck->Dy,egdck->Dxy);
    fprintf(out,"\nCx  = %14.4f  Cy  = %14.4G  Cxy = %14.4G",
            egdck->Cx,egdck->Cy,egdck->Cxy);
    fprintf(out,"\nGxy = %14.4f  Kxy = %14.4G",egdck->Gxy,egdck->Kxy);
  }
}   /*********************************************  end getstiffnesses  */


float gammaF(struct stiffness ES, float r)
{
  return ( 1-.901*(1-exp(-sqrt(r/pow(ES.Dx*ES.Dy/ES.Ex/ES.Ey,.25))/29.8)) );
}


float gammaM(struct stiffness ES, float r)
{
  return ( 1-.731*(1-exp(-sqrt(r/pow(ES.Dx*ES.Dy/ES.Ex/ES.Ey,.25))/29.8)) );
}


/*************************************************************************/
/*************************    file STRESS.C      ************************/
/*************************************************************************/
#include "ct.h"

float stresscheck(float Anew, float As, float Ns, float Ph, float r, float t,
      float Faxial, FILE *out, float St, float Scol, int SSP, float tau,
      float tau0, float sf)
{
  float VMSA,VMSB,VMSC;
  putchar('S');
  VMSA = VMStress("Max Compression",out,SSP,Scol,-Ph*r/t*sf,tau0);
  VMSB = VMStress("Max Shear",out,SSP,Faxial*(2*pi*(r-t/2)*t+As*Ns)/Anew,
           -Ph*r/t*sf,tau);
  VMSC = VMStress("Max Tension",out,SSP,St,-Ph*r/t*sf,tau0);
  if (VMSB>VMSA) VMSA = VMSB;
```

```
  if (VMSC>VMSA) VMSA = VMSC;
  return(VMSA);
}       /************************************************** stresscheck */


float VMStress(char loc[],  FILE *out, int SSP, float Sx, float Sy,
             float tauxy)
{
  float S1,S2,VMS;
  S1 = (Sx+Sy)/2 + sqrt( (Sx-Sy)*(Sx-Sy)/4 + tauxy*tauxy );
  S2 = (Sx+Sy)/2 - sqrt( (Sx-Sy)*(Sx-Sy)/4 + tauxy*tauxy );
  VMS = sqrt( ( (S1-S2)*(S1-S2) + S2*S2 + S1*S1 )/2 );
  if (4 == SSP)
  {
    fprintf(out,"\nPoint of %s:",loc);
    fprintf(out,"\nSx = %8.1f,  Sy = %8.1f,  tauxy = %8.1f",Sx,Sy,tauxy);
    fprintf(out,"\nS1 = %8.1f,  S2 = %8.1f,  Von Mises Stress = %8.1f",
                                              S1,S2,VMS);
  }
  return(VMS);
}       /************************************************** VMStress */
```

☆ U.S. GOVERNEMENT PRINTING OFFICE 1994—533-108/00017

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	January 1994	Technical Paper

4. TITLE AND SUBTITLE

CORSS: Cylinder Optimization of Rings, Skin, and Stringers

5. FUNDING NUMBERS

6. AUTHOR(S)

J. Finckenor, P. Rogers, and N. Otte

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

George C. Marshall Space Flight Center
Marshall Space Flight Center, Alabama 35812

8. PERFORMING ORGANIZATION REPORT NUMBER

M-740

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

National Aeronautics and Space Administration
Washington, DC 20546

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

NASA TP-3457

11. SUPPLEMENTARY NOTES

Prepared by Structures and Dynamics Laboratory, Science and Engineering Directorate.

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Unclassified — Unlimited
Subject category: 64

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Launch vehicle designs typically make extensive use of cylindrical skin stringer construction. Structural analysis methods are well developed for preliminary design of this type of construction. This report describes an automated, iterative method to obtain a minimum weight preliminary design.

Structural optimization has been researched extensively, and various programs have been written for this purpose. Their complexity and ease of use depends on their generality, the failure modes considered, the methodology used, and the rigor of the analysis performed. This computer program employs closed-form solutions from a variety of well-known structural analysis references and joins them with a commercially available numerical optimizer called the "Design Optimization Tool" (DOT).

Any ring and stringer stiffened shell structure of isotropic materials that has beam type loading can be analyzed. Plasticity effects are not included. It performs a more limited analysis than programs such as PANDA, but it provides an easy and useful preliminary design tool for a large class of structures.

This report briefly describes the optimization theory, outlines the development and use of the program, and describes the analysis techniques that are used. Examples of program input and output, as well as the listing of the analysis routines, are included.

14. SUBJECT TERMS	15. NUMBER OF PAGES
optimization, stiffened shell, skin-stringer	90
	16. PRICE CODE: A05

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	Unlimited