

N89-25170

The Sizing and Optimization Language (SOL)- A Computer Language to Improve the User/Optimizer Interface

**S. H. Lucas
Vigyan Research, Inc.
Hampton, Virginia**

**S. J. Scotti
NASA Langley Research Center
Hampton, Virginia**

PRECEDING PAGE BLANK NOT FILMED

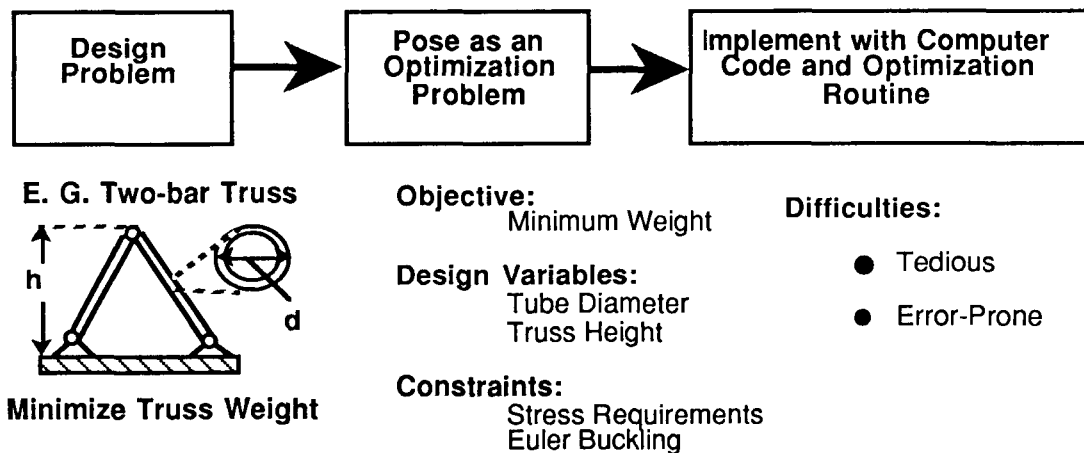
Optimization Techniques Applied to Design Problems

The nonlinear mathematical programming method (formal optimization) has had many applications in engineering design (refs. 1 and 2). This figure illustrates the use of optimization techniques in the design process. The design process begins with the design problem, such as the classic example of the two-bar truss designed for minimum weight as seen in the leftmost part of the figure.

If formal optimization is to be applied, the design problem must be recast in the form of an optimization problem consisting of an objective function, design variables, and constraint function relations. The middle part of the figure shows the two-bar truss design posed as an optimization problem. The total truss weight is the objective function, the tube diameter and truss height are design variables, with stress and Euler buckling considered as constraint function relations.

Lastly, the designer develops or obtains analysis software containing a mathematical model of the object being optimized, and then interfaces the analysis routine with existing optimization software such as CONMIN, ADS, or NPSOL (refs. 3, 4, and 5). This final stage of software development can be both tedious and error-prone.

This paper presents the Sizing and Optimization Language (SOL), a special-purpose computer language whose goal is to make the software implementation phase of optimum design easier and less error-prone.



SOL: A High-Level Computer Language

The use of a high-level computer language, as exemplified by SOL, meets the goals of making the optimum-design process easier and less error-prone, as seen in the figure below.

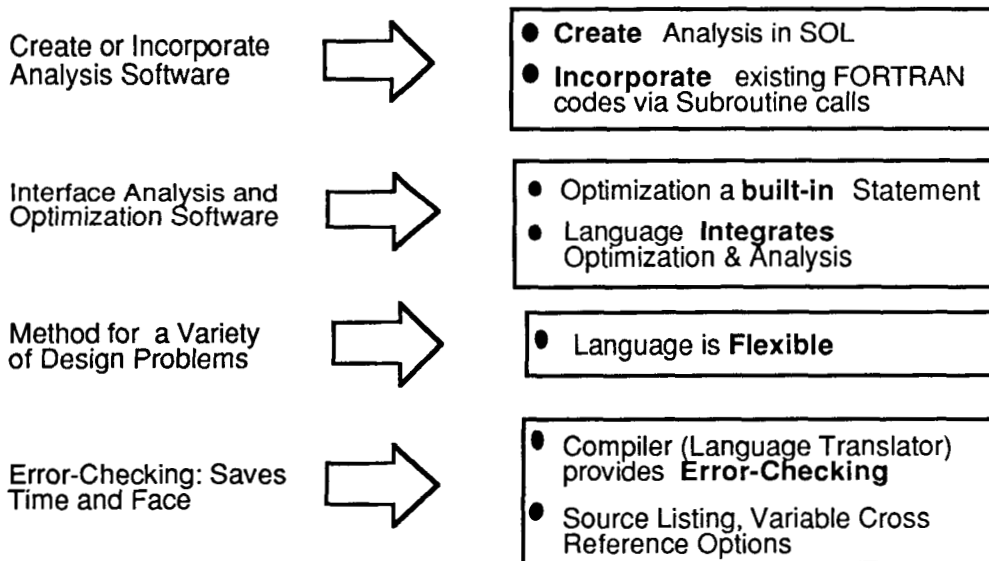
In terms of analysis, SOL provides statements which can either model a design mathematically or can model a design with subroutines and other code. In addition, a FORTRAN block feature permits the user to incorporate existing FORTRAN routines via subroutine calls and parameter-passing.

In terms of optimization, SOL provides an OPTIMIZE statement for describing an optimization problem. The OPTIMIZE description is concise and parallels the mathematical description of an optimization problem. Because the OPTIMIZE statement is a built-in language feature, like a DO or IF/THEN/ELSE statement, the language is the interface between optimization and analysis.

In terms of flexibility, SOL is quite general and can be used to code a variety of design problems.

In terms of error-checking, the SOL compiler provides a vehicle for error-checking specific to optimization problems. As the syntax of SOL statements is checked, semantic checks on the use of the statements can also be performed. Additionally, the compiler offers a listing which includes the SOL program indexed by line number; an optimization summary for each optimization which lists the objective, design variables, and constraints; and a cross-reference giving each variable and the lines on which the variable was used.

GOAL: Make Optimization Use EASIER and LESS ERROR-PRONE



SOL Statements

SOL is a simple but powerful language. A brief overview of the language elements of SOL is offered here, giving a representative list of available SOL statements. Further details can be found in NASA Technical Memorandum 100565 (ref. 6).

SOL offers many traditional language features found in "conventional languages," e.g. FORTRAN or Pascal. SOL provides declaration statements such as variable and subroutine declaration; control statements such as DO loops, IF/THEN/ELSE statements, and subroutine calls; calculation statements (i.e. assignments, math operators and built-in math functions); and output statements such as PRINT.

SOL has unique language features as well, such as an OPTIMIZE statement for describing an optimization problem and an ASSEMBLAGE statement (beyond the scope of this paper) to facilitate the hierarchical modeling of systems. As mentioned earlier, SOL's FORTRAN block allows existing FORTRAN code to appear within a SOL program. To make SOL programs easier to write and more readable, a MACRO feature allows the definition and use of text abbreviations within a SOL program. A single descriptive macro call can replace many lines of SOL code. For example, SOL has a pre-defined ?INCLUDE macro that allows entire text files to be included verbatim as part of a SOL program.

SOL's conventional features are combined with its unique features to solve design problems.

TRADITIONAL LANGUAGE FEATURES:

Declaration	Control	Calculation	Output
Variable Declaration	Conditional DO loops	Assignment	Print
Subroutine Declaration	IF/THEN/ELSE	Math Expressions	
	Iterative DO loops		
	Subroutine Call		

UNIQUE LANGUAGE FEATURES:

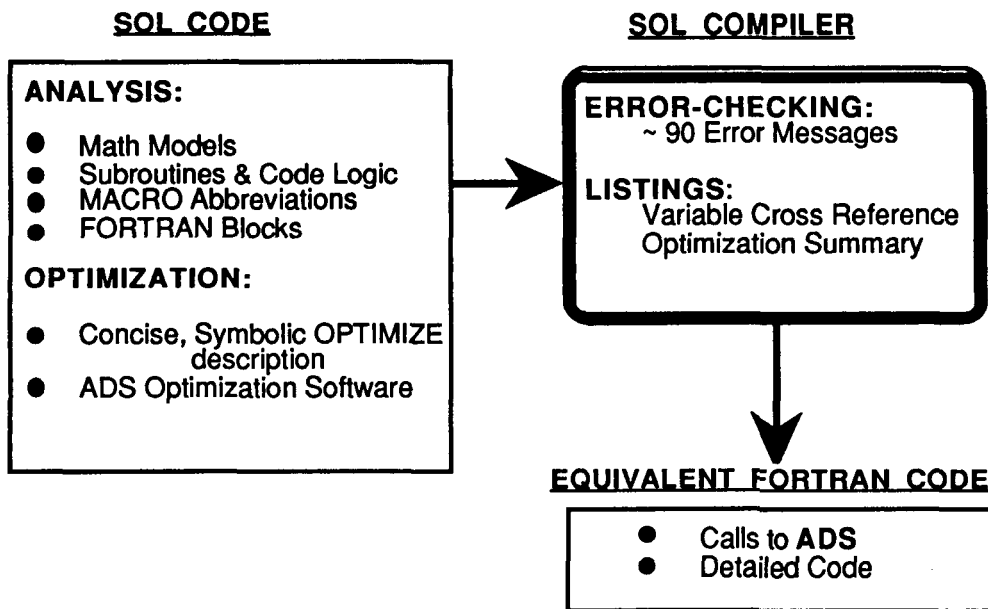
Description	Miscellaneous
OPTIMIZE Statement	FORTRAN block
ASSEMBLAGE Statement	Macro Definition
	Macro Call
	?INCLUDE macro

SOL Capabilities Used to Solve a Design Problem

Using SOL as a tool for engineering design involves writing a sequence of SOL statements that apply numerical optimization methods to a design problem. The process of solving a problem using SOL is shown in the figure below. A program composed of SOL statements is passed as input to the SOL compiler. Within the SOL program, the design can be modeled mathematically or with subroutines and other code. In addition, existing FORTRAN routines can be used via the FORTRAN block feature, and SOL's macro abbreviation feature can be used. SOL's OPTIMIZE statement describes the optimization problem, incorporating the methods of numerical optimization implemented in the ADS optimization routine (ref. 4).

The SOL compiler translates the SOL program into an equivalent FORTRAN program and does error-checking. The compiler offers approximately ninety different error messages, and can produce listings, a variable cross-reference, and an optimization summary which lists the objective, design variables, and constraints. However, SOL does not provide error-checking features for FORTRAN BLOCK code fragments; SOL's error-checking is limited to SOL statements only.

The FORTRAN program produced by the SOL compiler executes to solve the design problem. This resultant FORTRAN program includes subroutine calls to the ADS software and other detailed code.



Math and SOL Description of Two-Bar Truss

SOL's description of an optimization problem parallels the mathematical description of the problem as illustrated in the figure below which shows a minimum-weight, symmetric two-bar truss problem.

The mathematical description appears on the left of the figure. The truss weight is the objective function to be minimized as stated under the heading, "minimize." The design variables and constraint relations appear under the heading, "subject to." The tube diameter (d) and truss height (h) are design variables, with compressive stress and Euler buckling constraints to insure that the truss neither yields nor buckles. A mathematical model of the truss is given under the heading, "where," which includes the additional variables for truss length (L), half-span (B), tube wall thickness (t), load (P), compressive strength of material (σ^{\max}), modulus (E), and material density (ρ). The mathematical model defines the objective and constraint relations as functions of the design variables.

The SOL description on the right parallels the mathematics. The objective function is represented by a single variable (weight). Design variables and constraint function relations appear between the words **USE** and **END USE**. The lower and upper bounds on the design variables appear in brackets following the word, **IN**. In addition, the optimization software requires design variable initial values, which are given with each design variable after the "=" symbol. Compressive stress and Euler buckling constraints follow the design variables.

Finally, equations modeling the truss appear between the words **END USE** and **END OPTIMIZE**. Note that the single SOL variable (buckle), representing the Euler buckling constraint, acts identically to the constraint relation in the mathematical description.

Mathematical Description

Minimize: weight(d, h)

Subject to:

$$1 \leq d \leq 3$$

$$10 \leq h \leq 30$$

$$\sigma^{\text{stress}}(d, h) < \sigma^{\max}$$

$$\sigma^{\text{stress}}(d, h) - \sigma^e(d, h) < 0$$

Where:

$$\text{weight}(d, H) = 2\rho\pi dtL$$

$$\sigma^{\text{stress}}(d, h) = (PL)/(\pi thd)$$

$$\sigma^e(d, h) = \pi^2 E(d^2 + t^2)/(8L^2)$$

SOL Description

OPTIMIZE weight

USE

$$d = 1 \text{ IN } [1, 3]$$

$$h = 15 \text{ IN } [10, 30]$$

$$\text{stress} \text{ .It. MaxStress}$$

$$\text{buckle} \text{ .It. 0}$$

END USE

$$\text{weight} = 2*\rho*\pi*d*t*L$$

$$\text{stress} = (P*L)/(\pi*t*h*d)$$

$$\text{Euler} = (\pi**2*E*(d**2+t**2))/(8*L**2)$$

$$\text{buckle} = \text{stress} - \text{Euler}$$

END OPTIMIZE

SOL Error-Checking Example

A SOL program is passed as input to the SOL compiler which translates the SOL code into an equivalent FORTRAN code, and provides the key feature of error checking for a variety of errors. The figure below illustrates the error-checking capability of the compiler.

An intentionally erroneous SOL program for the two-bar truss problem appears on the left of the figure. The program has been annotated with line numbers to aid the discussion. The inset box on the right lists the actual error messages given by the SOL compiler on receipt of this program. The first error occurs on line 11 where the word IN has been misspelled; the compiler can usually correct the spelling of reserved words when the word is misspelled by a single character. The next error is optimization specific, warning that the constraint variable stated on line 14 has not been assigned a value. The error message leads to the discovery that a typographical error on line 20 is the true culprit. Finally, an error appears for line 17 because the variable for material density, rho, was not initialized.

Either of the last two errors would have caused incorrect optimization results if left undetected. The last two errors are difficult to detect manually; a laborious examination of the optimization results could reveal that the results were incorrect, but would not provide the cause for the poor results.

It is important to note that the compiler is not clairvoyant; it cannot check the correctness of problem formulation nor infer one's intentions. However the example here, although not exhaustive, illustrates the general sorts of errors detected by the compiler.

Erroneous Program	Error Messages
1 : PROGRAM TwoBar	11 : d = 1 INN [1, 3]
2 : t = 0.1	*** ERROR ^ MISPELLED "IN" CORRECTED
3 : P = 3300	
4 : B = 30	14 : buckle .lt. 0
5 : E = 30000000	***ERROR ^OPTIMIZATION VARIABLE HAS NOT BEEN SET
6 : pi = 3.141592554	
7 : MaxStress = 10000	17 : weight = rho*2*pi*d*t*L
8 :	***ERROR ^UNINITIALIZED IDENTIFIER
9 : OPTIMIZE weight	
10 : USE	
11 : d = 1 INN [1, 3]	
12 : h = 15 IN [10, 30]	
13 : stress .lt. MaxStress	
14 : buckle .lt. 0	
15 : END USE	
16 : L = SQRT(B**2 + h**2)	
17 : weight = rho*2*pi*d*t*L	
18 : stress = (P*L)/(pi*t*h*d)	
19 : E_stress = ((pi**2)*E*(d**2+t**2))/(8*L**2)	
20 : buckl = stress - E_stress	
21 : END OPTIMIZE	
22 : END TwoBar	

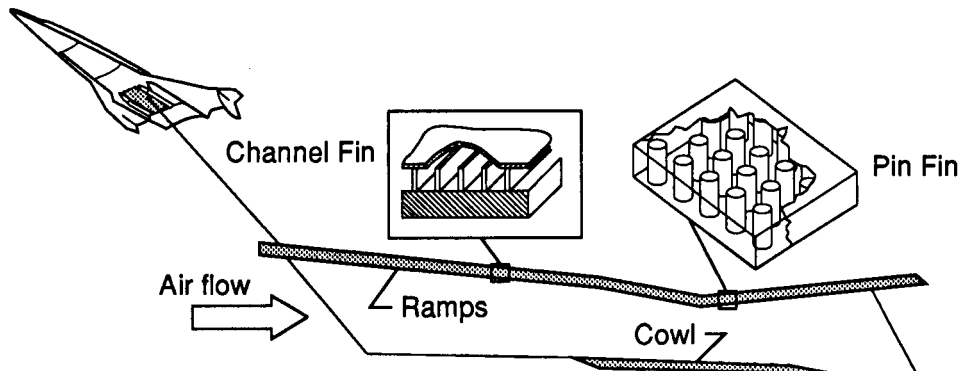
Scramjet Engine Cooling Jacket Application

The design of scramjet engine cooling jackets, in which numerical optimization is used as a design tool, illustrates SOL's use for an engineering application.

A scramjet engine resides on the lower surface of a hypersonic vehicle, as in the schematic below. A conceptual, two dimensional engine cross-section appears in the middle of the figure, showing the ramp and cowl portions of the engine. The heating of the engine surfaces wetted by the airstream is so extreme that an active cooling system is required to maintain a survivable temperature. Only the ramp and cowl portions of the engine are considered here, although other parts of the engine also require active cooling. A promising active cooling system for this application is a system of hydrogen-fuel-cooled, metallic, surface heat exchangers (cooling jackets) attached directly to the engine primary structure.

Both channel-fin and pin-fin cooling jackets were studied, but the example in this paper focuses on a channel-fin design. The design goal is to design cooling jackets which minimize the required coolant flow rate for specified heating rates. The design must also satisfy requirements such as material limits on cooling jacket temperature, fatigue life and stress.

The cooling jacket design problem was recast in the form of an optimization problem and implemented in SOL using SOL's OPTIMIZE statement. Existing FORTRAN routines were incorporated for the analysis of a single cooling jacket panel via SOL's FORTRAN block feature. Other SOL features were used to control the analysis and perform ancillary calculations.



Application Implemented with SOL

- Optimization problem posed in SOL.
- Single panel analysis with existing FORTRAN codes.
- SOL features used to control analysis routines.

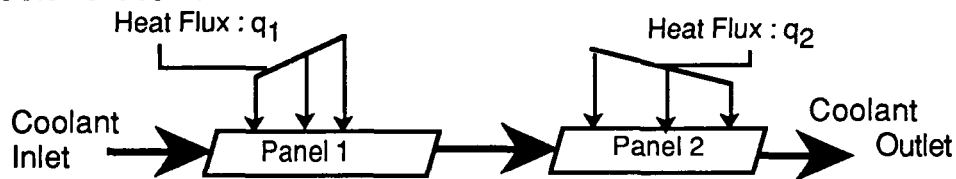
Cooling Jacket Design Problem

The figure below illustrates the scramjet engine cooling jacket design problem in some detail. As seen in the top half of the figure, a coolant flows through cooling jacket panels to remove the incident heat flux (q). Only two panels of equal dimensions are shown in the figure, although many panels of varying sizes can be used. For more details of panel configurations see reference 7.

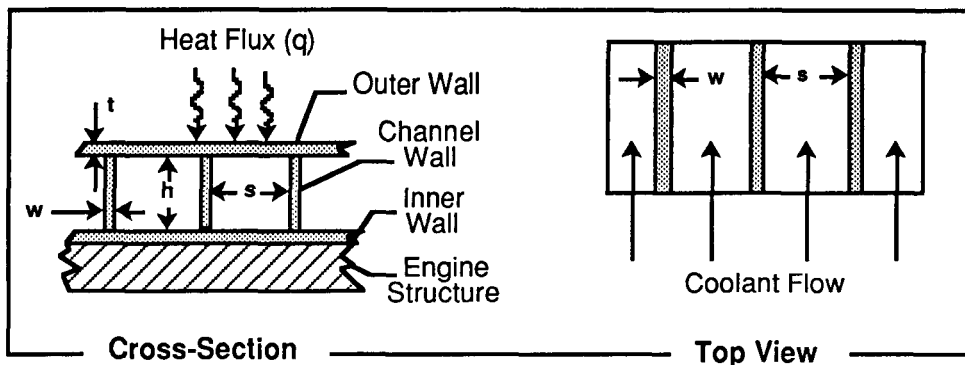
The lower half of the figure illustrates the geometry of a channel-fin cooling jacket. As seen on the left-hand side, a channel-fin geometry can be completely described by the channel width (s), the channel height (h), the channel wall thickness (w), and the outer wall thickness (t). The right-hand side of the figure shows a top view of a channel-fin cooling jacket, illustrating the coolant flow through the jacket channels.

When the design is recast in the form of an optimization problem, several design variables describe the coolant flow conditions and the remainder of the design variables describe the cooling jacket geometry.

Coolant Flow:



Channel Fin Jacket Geometry:



SOL's Use for Cooling Jacket Optimization

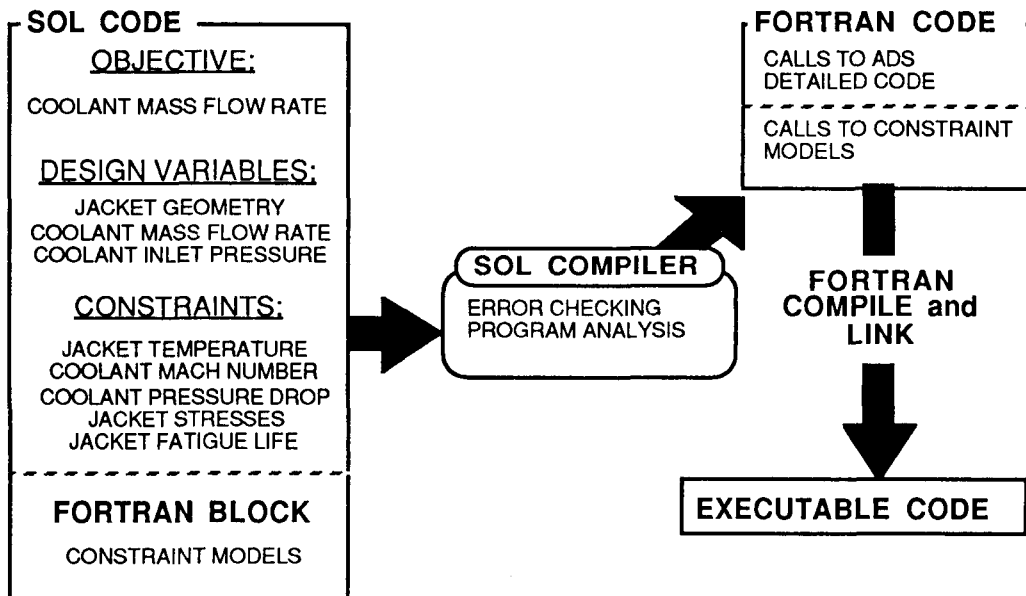
SOL's use in the cooling jacket optimization is shown in the schematic below. SOL code describes the optimization problem as in the left-hand part of the figure. The objective function to be minimized is the coolant mass flow rate. Weight is a more common objective function. But in this application minimizing the coolant mass flow rate can decrease the coolant needed, effectively reducing the total vehicle weight. The design variables consist of variables describing the coolant mass flow rate, the coolant inlet pressure, and several variables to describe cooling jacket geometry.

In addition, constraints on the coolant and jacket conditions are required. As stated earlier, existing FORTRAN routines calculate the constraint function relations. The constraint routines are called from the SOL program by subroutine calls. Design variable values are passed as parameters to the constraint routines, which return constraint function values, also via parameter-passing.

The SOL program for the cooling jacket application is passed as input to the SOL compiler, which produces an equivalent FORTRAN program as output. The compiler also performs program analysis and error-checking on the SOL code.

The output FORTRAN code contains calls to the ADS optimization software which provide's SOL's optimization capability, as well as calls to the constraint modeling routines. The output code also contains detailed code such as variable declarations and so forth.

The FORTRAN code output by the SOL compiler is compiled and linked using the FORTRAN compiler and linker. The resulting executable code is run to perform the cooling jacket optimization.



Cooling Jacket Optimization Description in SOL: An Overview

The figure below gives the SOL program for the cooling jacket optimization problem in outline form with all reserved words shown in boldface type. The program begins with the word **PROGRAM** followed by the name of the program. Before the optimization problem description begins, variables and subroutines are declared, and macro definitions appear. In the figure, the actual code has been replaced with comments, marked by exclamation point symbols, to simplify the discussion. Subsequent figures will discuss each of the comment sections in turn.

The optimization problem description is initiated by the word **OPTIMIZE** and terminated by the words **END OPTIMIZE**. A single variable given after the word **OPTIMIZE** states the objective function. Next, design variable, constraint relation and optimization software option declarations appear between the words **USE** and **END USE**. In the figure below, the three declarations have been replaced with comments. The SOL code for the cooling jacket analysis appears between the words **END USE** and **END OPTIMIZE**, as indicated by a SOL comment in the figure.

The main body of the SOL program terminates with the word, **END**, followed by the name of the program. In SOL, subroutines follow the end of the main program body. In the cooling jacket application, SOL's **?INCLUDE** macro is used to include the contents of the file "cool_jacket.sub," which contains the subroutines for cooling jacket analysis.

```
PROGRAM Cool_Jacket
  ! Variable and Subroutine Declarations
  ! Macro Definitions

OPTIMIZE total_panel_flowrate
USE
  ! Design Variable Declarations
  ! Constraint Function Relation Declarations
  ! Optimization Software Options
END USE
  ! Cooling Jacket Analysis
END OPTIMIZE

END Cool_Jacket

?INCLUDE Cool_Jacket.sub
```

Objective and Design Variable Description in SOL

The figure below details the SOL code for the design variable declaration section of the cooling jacket optimization as outlined previously. The objective function to be minimized, the coolant mass flow rate, follows the word **OPTIMIZE** and is represented by a single variable, "total_panel_flowrate."

Design variable declarations follow the word **USE**. For this application, six design variables are used. Two variables, "panel_flowrate" and "inlet_pressure," describe coolant conditions. Also, four design variables are needed to describe the geometry of each cooling jacket panel. The design variables for a single panel are shown in the figure; each of the panel geometry variables are suffixed with the panel name, "_panel_1." If a second panel were also considered, four additional design variables for the second panel's geometry would be required. Since design variables must have unique names, these variables could have the suffix "_panel_2." This naming convention provides a consistent way to handle multiple-panel optimizations.

The lower and upper bounds on the design variables appear to the right of each design variable enclosed by brackets; the actual numbers are unimportant for this discussion. In addition, initial values for design variables required by the optimization software appear with each design variable following the equals symbol.

```
OPTIMIZE total_panel_flowrate
USE
panel_flowrate      = 3.0      IN [1.000, 4.000]
inlet_pressure      = 1000.0   IN [1000., 1500.]
aspect_ratio_panel_1 = 0.5641 IN [0.400, 0.800]
spacing_panel_1     = 0.02    IN [0.020, 0.025]
outer_wall_panel_1  = 0.016   IN [0.010, 0.018]
channel_wall_panel_1 = 0.09    IN [0.060, 0.120]
! *** Constraint Relation Declarations ***
! *** Optimization Software Options ***
END USE
! *** Cooling Jacket Analysis ***
END OPTIMIZE
```

Constraint Function Relation Descriptions in SOL

The figure below details the SOL code for the constraint function relation declaration section of the cooling jacket optimization as outlined previously.

Six constraints are used for a single cooling jacket panel optimization. A single constraint on coolant pressure drop is represented by the variable, "pressure_drop." In SOL, the relation "less than" is represented by .lt. and the relation "greater than" is represented by .gt. In this case, the "pressure_drop" must be less than 100 psi. Five additional constraints are required for each panel, representing cooling jacket low cycle fatigue life, coolant Mach number at the panel exit, cooling jacket temperature, and cooling jacket stresses. As with the design variables, the cooling jacket panel constraints are suffixed with the panel name. In the figure, the name "_panel_1" is used as a suffix.

```
OPTIMIZE total_panel_flowrate
USE
  ! *** Design Variable Declarations ***
  pressure_drop .lt. 100
  fatigue_life_panel_1 .gt. 600
  gas_mach_out_panel_1 .lt. max_coolant_mach
  outer_temp_panel_1 .lt. 2000
  wall_stress_panel_1 .lt. 1
  web_stress_panel_1 .lt. 1
  ! *** Optimization Software Options ***
END USE
  ! *** Cooling Jacket Analysis ***
END OPTIMIZE
```

Optimizer Option Description in SOL

The figure below details the SOL code for the optimization software option declaration section of the cooling jacket optimization as outlined previously. The ADS optimization software used by SOL offers a variety of optimization algorithms and access to numerous internal parameters such as convergence criteria or maximum number of iterations. The software option declaration section provides access to these parameters from a SOL program.

The software option declaration section appears after the design variable and constraint function relation declarations. The software options section begins with the word **OPTIONS** and ends with the words **END USE**. In the figure below, a sequential quadratic programming strategy is selected along with a golden section method of one-dimensional search. The modified method of feasible directions for constrained minimization is used as the optimizer. SOL automatically supplies default option values for the new user, but the **OPTIONS** section permits a knowledgeable user to take full advantage of the options offered by the ADS software. Also, the **OPTIONS** section separates the description of the optimization problem, the objective; design variables; and constraints, from the details of the particular optimization software used to solve the problem.

The word **normalize** indicates that design variables are to be normalized between the values 0 and 1.0. Scaling variables often make an optimization problem better conditioned and hence easier to solve.

```
OPTIMIZE total_panel_flowrate
USE
! *** Design Variable Declarations ***
! *** Constraint function relation Declarations ***
OPTIONS
  strategy = sequential quadratic
  optimizer = modified feasible directions
  search = golden section
  normalize
END USE
! *** Cooling Jacket Analysis ***
END OPTIMIZE
```

Cooling Jacket Analysis in SOL

The figure below details the SOL code for the cooling jacket analysis, as outlined previously. The analysis computes the values of the objective and constraints as functions of the design variables.

The analysis code appears between the words **END USE** and the words **END OPTIMIZE**. In the figure, the first two assignment statements define variables for the initial coolant pressure and coolant temperature. The next statement gives the location of the first cooling jacket panel.

A SOL macro, **?Channel_Panel**, analyzes a single cooling jacket panel as shown in the figure. The macro abbreviation hides the details of the analysis. The macro itself is shown in boldfaced type, whereas the macro parameters supplied by the user are shown in plain type. The first parameter, "panel_1" is the name of the panel being analyzed. The two parameters that follow "x=" define the length of the panel, and the two parameters for "q=" define the heat flux incident at the start of the panel and at the panel exit. The user simply calls the **?Channel_Panel** macro with the desired parameters to analyze a single cooling jacket panel. The macro defines the necessary variables and calls the external FORTRAN routines to perform the analysis. The user can conduct multiple panel analysis by calling the macro once for each panel analyzed. Although the actual code for the analysis is quite complex, the macro simplifies the complexity into a macro call with five parameters. This discussion focuses on the use of the **?Channel_Panel** macro, some details of how the macro was defined are presented subsequently.

Two assignment statements follow the macro call. The first assigns the objective function a value; the variable "panel_flowrate" is a design variable. The second assignment gives a value to the pressure drop constraint. Other constraint variables are defined by the macro call. The analysis and optimization ends with the words **END OPTIMIZE**.

```
OPTIMIZE total_panel_flowrate
USE
! *** Design Variables, Constraints and Options
END USE
gas_p_in = inlet_pressure ! a design variable
gas_t_in = 1000
Panelstart = 0
?Channel_Panel panel_1 begin x= 0 q= heatrate
                        end x= 75 q= heatrate
total_panel_flowrate = panel_flowrate
pressure_drop = inlet_pressure - gas_p_out
END OPTIMIZE
```

Cooling Jacket Analysis: ?Channel_Panel Macro Use

The figure below shows the creation of the ?Channel_Panel macro used to conduct cooling jacket analysis.

The box in the upper part of the figure shows how the Channel_Panel macro is defined. The definition begins with the word, ?DEF, followed by the name of the macro. The numbered items in plain text (#1, #2, ..., #5) are parameters to the macro. Also, SOL allows text (boldface in the figure) to be used to separate the macro parameters. Often the macro creator will use this delimiting text as a reminder for the parameters' use. For example the text, **begin**, was chosen to indicate that the second and third parameters are the location (x=) and heat flux (q=) at the panel START, whereas "end" is used to indicate that the fourth and fifth parameters are for the panel EXIT.

Macros are text abbreviations; the text that the macro abbreviates appears between the open and close curly braces in the definition. The ?Channel_Panel text initializes all the variables associated with a cooling jacket panel analysis. Only one initialization is shown in the figure with the remainder represented with ellipses. The text also calls an external FORTRAN routine which analyzes a panel, also only shown as a comment in the figure.

When the macro is called, the macro's text is executed with the user supplied parameters inserted. For example, the lower part of the figure shows a call to ?Channel_Panel. The effect is exactly as if the very bottom text box were typed instead of the call; the macro merely abbreviates text. Notice that the parameter, "panel_1," is inserted in the macro text in the place of "#1" when the macro is called.

A macro was used because the variable initializations and call to the external FORTRAN routine, which has 23 parameters, must be repeated for every panel analyzed; tedious typing if multiple panels are analyzed. The macro hides this complexity, replacing the tedious typing with one simple macro call per panel.

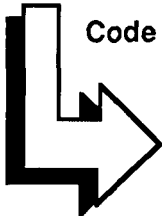
Macro Definition:

```
?DEF ?channel_Panel #1 begin x= #2 q= #3  
                        end x= #4 q= #5  
{  
  webheight = aspect_ratio_#1 * spacing_#1  
  .  
  .  
  .  
  ! Call external subroutine with FORTRAN block  
}
```

Macro Call:

```
?Channel_Panel panel_1 begin x= 0 q= 75 end x= 2000 q= 2500
```

Code SUBSTITUTED for Macro Call:



```
webheight = aspect_ratio_panel_1 * spacing_panel_1  
.  
.  
.  
! Call external subroutine with FORTRAN block
```

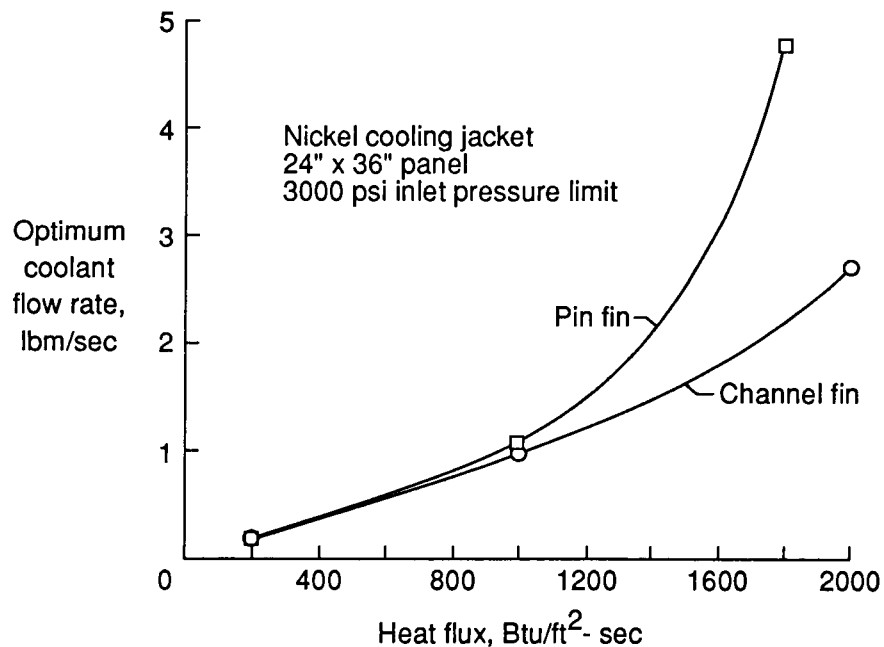

Some Results from Cooling Jacket Study: Channel Fin Versus Pin-fin Comparison

The cooling jacket study produced many results, some of which are illustrated in the figure below. The graph charts optimum coolant flow for channel-fin and pin-fin jackets as a function of heat flux. Results for a Nickel cooling jacket panel, 36 inches wide and 24 inches long with an inlet pressure limit of 3000 psi., are shown in the figure. The graph shows several significant results.

First, a simple energy-balance for determining coolant requirements predicts a linear relationship between coolant flow rate and heat flux. The results are clearly non-linear in the figure.

Second, at the lower heat-flux levels, there is little difference in the value of the optimum coolant flow rate for channel-fins and pin fins. But at high heat fluxes, the channel-fins have lower coolant flow requirements than the pin fins.

Finally, each point on the graph for channel-fins or pin-fins represents an optimum coolant flow rate. In this way, the graph can be interpreted as illustrating the optimum sensitivity of the coolant flow requirements to heat-flux for the given cooling jacket design. Note that smooth curves are faired through the calculated points of this figure, but the actual curves undoubtedly contain slope discontinuities whenever the set of active design constraints changes.



SUMMARY

A special-purpose programming language, SOL, has been developed to expedite implementation of optimization problems and to make the process less error-prone. A more detailed discussion of SOL can be found in reference 6. Currently SOL is only available for DEC VAX/VMS systems.

As a language, SOL provides a high-level interface to the ADS optimization software. SOL integrates optimization and analysis within a single OPTIMIZE description, which parallels the mathematical description of an optimization problem. In terms of analysis, SOL provides language statements which can be used to model a design mathematically, with subroutines and other code, or to model a design with existing FORTRAN routines and parameter-passing. SOL also provides error checking geared to optimization problems to make problem implementation less error-prone. Because optimization is a built-in language statement, the language is the interface.

SOL's use is illustrated in the design of scramjet engine cooling jackets. In this example, the cooling jacket optimization problem was posed in SOL. Existing FORTRAN routines for panel analysis were incorporated into the SOL program using SOL's FORTRAN block feature. Other SOL features were used to control the analysis routines, and provide a simple method of conducting multiple panel analysis. Reference 7 provides details of the scramjet engine cooling jacket application.

SOL, a computer language for optimization, developed.

- NASA TM 100565 details SOL
- Available for DEC VAX/VMS Systems

High-level Interface to Optimizer Software

- Simplifies Optimization Software use
- Reduces Errors with Error-Checking
- Language Integrates Optimization and Analysis

Cooling Jacket application illustrates SOL's use.

- Existing FORTRAN codes used for analysis
- NASA TM 100581 details Cooling Jacket application

References

- ¹Schmit, L.A.: Structural Synthesis -- Its Genesis and Development. *AiAA J.*, vol. 19, no. 10, Oct. 1981, pp. 1249 - 1263.
- ²Ashley, H.: On Making Things Best -- Aeronautical Uses of Optimization. *J. Aircr.*, vol. 19, no. 1, Jan. 1982d, pp. 5 - 28.
- ³Vanderplaats, G.N.: *CONMIN -- A FORTRAN Program for Constrained Function Minimization -- User's Manual*. NASA TM X-62282, 1973.
- ⁴Vanderplaats, G.N.: *ADS -- A FORTRAN Program for Automated Design Synthesis -- Version 1.10*. NASA Contractor Report 177985, Grant NAG1-567, 1985.
- ⁵Gill, P.E.; Murray, W.; Saunders, M.; and Wright, M.: User's Guide for NPSOL (Version 4.0): a FORTRAN Package for Nonlinear Programming. Systems Optimization Laboratory, Stanford University. January 1986. Available from the Stanford Office of Technology Licensing, 350 Cambridge Avenue, Suite 250, Palo Alto, California 94306, USA.
- ⁶Lucas, S.H. and Scotti, S.J.: *The Sizing and Optimization Language, SOL, -- A Computer Language for Design Problems*. NASA TM 100565, 1988.
- ⁷Scotti, S.J.; Martin, C.J.; and Lucas, S.H.: *Active Cooling Design for Scramjet Engines Using Optimization Methods*. NASA TM 100581, 1988.