

OptdesX™
A SOFTWARE SYSTEM FOR OPTIMAL ENGINEERING DESIGN
USERS MANUAL
Release 1.0

Design Synthesis, Inc.
3883 N. 100 E.
Provo, UT 84604
(801) 378-6544
FAX: (801) 378-5037

Alan Parkinson
Richard Balling
Joseph Free
Jeff Talbert
Daren Davidson
Greg Gritton
Liana Borup
Brandon Busaker

OptdesX is a proprietary program with restricted rights granted to licensees. Copyright © 1992 Design Synthesis Inc. Permission to copy this manual is granted to holders of a valid software license.

1.	Installing OptdesX on Your System	1-1
1.1	DEC Ultrix, IBM AIX, HP UX, Silicon Graphics Irix Operating Sys.....	1-1
1.2	DEC VMS Operating System	1-4
1.3	Sun Operating System.....	1-6
1.4	Linking Your Own Model to OptdesX	1-8
1.5	OptdesX Colors, Fonts.....	1-8
2.	Overview.....	2-1
2.1	For People Who Hate to Read Manuals.....	2-1
2.2	An Overview of OptdesX	2-1
2.3	OptdesX Terminology.....	2-1
2.4	Scaling of Design Variables and Functions	2-4
2.5	How OptdesX Communicates with Analysis Software	2-5
2.6	Philosophy of the OptdesX Windows Interface.....	2-6
2.7	Types of Widgets	2-6
2.8	Window Features	2-8
2.9	Dismiss and Help Buttons.....	2-9
2.7	Files and File Selection	2-10
3.	OptdesX Tutorials	3-1
3.1	Introduction.....	3-1
3.2	Linking an Analysis Model to OptdesX.....	3-1
3.3	Tutorial 1: Optimizing with Continuous Variables	3-3
3.4	Tutorial 2: Optimizing with Discrete Variables.....	3-53
3.5	Tutorial 3: Gradients	3-80
4.	Linking Analysis Models to OptdesX.....	4-1
4.1	Introduction.....	4-1
4.2	The Conventional Interface in Fortran	4-2
	4.2.1 UNIX Systems	4-2
	4.2.2 DEC VMS Systems.....	4-3
	4.2.3 Example Fortran ANASUB routines	4-3
4.3	The Conventional Interface in C	4-5
	4.3.1 UNIX Systems	4-5
	4.3.2 DEC VMS Systems.....	4-7
	4.3.3 Example C ANASUB routines	4-7
4.4	The Stand-Alone Interface in Fortran	4-9
	4.4.1 Introduction.....	4-9
	4.4.2 A Simple Example	4-9
	4.4.3 A More Complicated Example	4-11
	4.4.4 The System Command for VAX/VMS	4-16
4.5	The Stand-Alone Interface in C	4-16

5.	Variables Window Reference.....	5-1
5.1	Overview.....	5-1
5.2	Continuous Variables.....	5-2
5.2.1	Reference Diagram.....	5-2
5.2.2	Unmapped Analysis Variables.....	5-3
5.2.3	Design Variables.....	5-3
	Design Variable Names and Values.....	5-3
	At Bounds Icon.....	5-3
	Continuous-Discrete Button.....	5-3
	# Map Value Field.....	5-4
	Minimum and Maximum Value Fields.....	5-5
5.3	Discrete Variables.....	5-6
5.3.1	Discrete Variable Reference Diagram.....	5-6
5.3.2	Discrete Window Operation.....	5-6
	Filename.....	5-6
	Row # Value Field.....	5-6
	Related Discrete Variables.....	5-7
5.3.3	Discrete File Format.....	5-8
5.4	Compute Functions, Post Process, Hardcopy.....	5-8
6.	Functions Window Reference.....	6-1
6.1	Overview.....	6-1
6.2	Single Objective Problems.....	6-2
6.2.1	Reference Diagram.....	6-2
6.2.2	Unmapped Analysis Functions.....	6-3
6.2.3	Design Functions.....	6-3
	Design Function Names and Values.....	6-3
	Objective and Constraint Buttons.....	6-3
	# Map Value Field.....	6-3
	Allowable and Indifference Value Fields.....	6-5
6.3	Multiple Objective Problems.....	6-6
6.3.1	Multiple Objective Reference Diagram.....	6-6
6.3.2	Multiple Objective Operation.....	6-6
6.3.3	Multiple Objective Solution Method.....	6-7
7.	Gradients Window Reference.....	7-1
7.1	Overview.....	7-1
7.2	Gradients of the Analysis Model.....	7-2
7.2.1	Reference Diagram.....	7-2
7.2.2	The Gradients Box.....	7-3
7.2.3	Display Options.....	7-4
	Value-Color Radio Box.....	7-4
	Unscaled-Scaled Radio Box.....	7-5
7.2.4	Numerical Options.....	7-6
7.3	Derivatives as a Check of Problem Scaling.....	7-6

7.4	Determine Best Method Feature	7-9
7.5	Gradients of an Optimum.....	7-12
8.	Optimize Window Reference.....	8-1
8.1	Reference Diagram	8-1
8.2	Operation.....	8-2
8.3	OptdesX Algorithms--General Information	8-2
8.3.1	Generalized Reduced Gradient.....	8-2
8.3.2	Sequential Quadratic Programming	8-2
8.3.3	Branch and Bound	8-3
8.3.2	Simulated Annealing	8-3
8.3.2	Exhaustive Search	8-3
8.4	Essential Algorithm Information	8-3
8.4.1	Running Several Iterations at a Time	8-3
8.4.2	Effects of Scaling	8-3
8.4.3	Crashes	8-4
8.4.4	Failure to Make Progress.....	8-4
8.4.5	Global vs. Local Optima.....	8-4
8.4.6	Optimization of Noisy Functions	8-4
8.4.7	Stopping Messages, General	8-5
8.5	GRG Algorithm	8-5
8.5.1	Description and Performance	8-5
8.5.2	Storage Requirements.....	8-6
8.5.3	Parameters	8-6
8.5.4	Stopping Messages	8-8
8.5.5	Optimization Summary	8-8
8.5.6	History Plot.....	8-9
8.6	SQP Algorithm.....	8-5
8.6.1	Description and Performance	8-10
8.6.2	Storage Requirements.....	8-11
8.6.3	Parameters	8-11
8.6.4	Stopping Messages	8-11
8.6.5	Optimization Summary	8-12
8.6.6	History Plot.....	8-12
8.7	Branch and Bound Algorithm	8-12
8.7.1	Description and Performance	8-12
8.7.2	Storage Requirements.....	8-14
8.7.3	Parameters	8-14
8.7.4	Stopping Messages	8-15
8.7.5	Optimization Summary	8-15
8.7.6	History Plot.....	8-16
8.8	Simulated Annealing Algorithm	8-18
8.8.1	Description and Performance	8-18
8.8.2	Storage Requirements.....	8-19
8.8.3	Parameters	8-19

8.8.4	Stopping Messages	8-20
8.8.5	Optimization Summary	8-20
8.8.6	History Plot.....	8-21
8.9	Exhaustive Search Algorithm	8-22
8.9.1	Description and Performance	8-22
8.9.2	Storage Requirements.....	8-23
8.9.3	Parameters	8-23
8.9.4	Stopping Messages	8-24
8.9.5	Optimization Summary	8-24
8.9.6	History Plot.....	8-25
8.10	Optimization Options Window	8-26
8.10.1	During Optimization.....	8-26
8.10.2	Files	8-27
8.10.3	Summary Window Options	8-27
8.11	Optimization Summary Window	8-28
8.10.1	Operation	8-28
8.10.2	Buttons.....	8-28
9.	Explore Window Reference.....	9-1
9.1	Overview.....	9-1
9.2	Explore Analysis.....	9-2
9.2.1	1-d Explore	9-2
9.2.1	2-d Explore.....	9-3
9.3	Explore Optimum	9-3
9.4	Explore Files	9-7
10.	Graph Window Reference	10-1
10.1.	Overview.....	10-1
10.2.	Sensitivity and History Graphs	10-2
10.2.1	Reference Diagram.....	10-2
10.2.2	Operation	10-3
	General.....	10-3
	Graphing Data from an Explore Optimum	10-3
10.3	Contour Plots	10-5
10.3.1	Reference Diagram.....	10-5
10.3.2	Operation	10-5
	General.....	10-5
	Equal Value Contours	10-6
	Zooming In.....	10-7
	Appropriate Mesh Density	10-8
	Boundary Values.....	10-10
	Graphing Data from an Explore Optimum	10-11
10.4	Hardcopy.....	10-12
11.	File Window Reference	11-1

11.1	Overview.....	11-1
11.2	Opening Files.....	11-2
11.3	Saving Files.....	11-2
11.4	New, Remove, Path.....	11-3
11.4.1	The New Button.....	11-3
11.4.2	The Remove Button.....	11-5
11.4.3	The Path Button.....	11-6
11.5	Example Analysis and Setup Files.....	11-6
12.	Error Reference.....	12-1

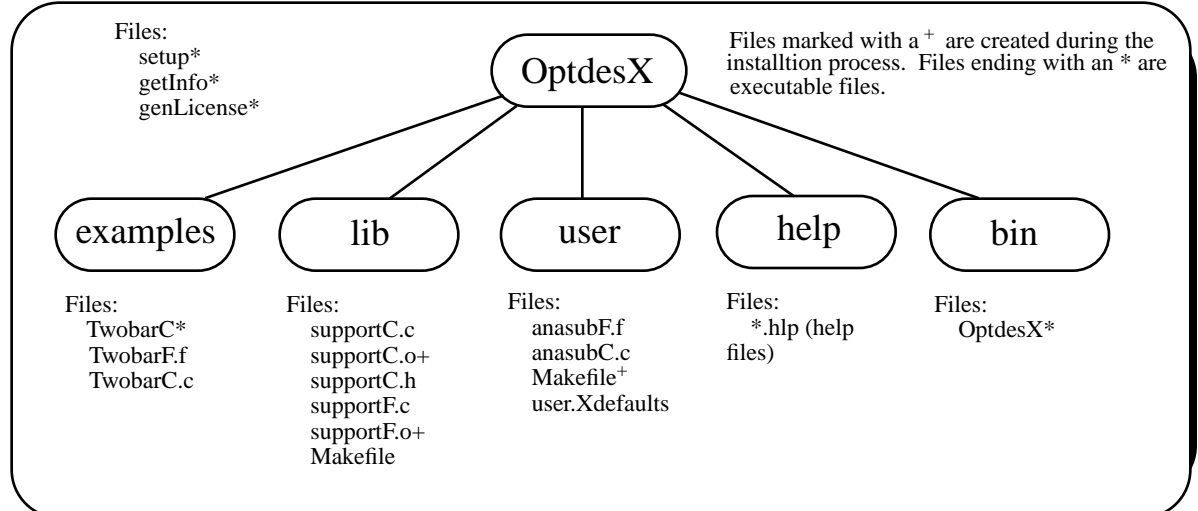
1 Installing OptdesX on Your Computer

1.1 Items Shipped with OptdesX

The following items are shipped on the OptdesX tape:

1. The OptdesX executable program (`OptdesX`)
2. The Inter-Process Control (IPC) support source code modules (`supportC.c`, `supportC.h`, `supportF.c`)
3. Template routines to help you link your model to OptdesX (`anasubC.c`, `anasubF.f`)
4. Source code and an executable module which demonstrates how to link your model to OptdesX (`TwobarC.c`, `TwobarF.f`, and `Twobar`)
5. The setup program to install OptdesX on your system after the directory tree has been downloaded from the tape (`setup`)
6. A program which generates a software license agreement which must be signed and returned within 30 days in order to continue using OptdesX (`genLicense`)
7. A program (`getinfo`) which is used to obtain the CPU ID number of the computer OptdesX is installed on. The `genLicense` program calls `getinfo` to put the CPU ID number into the license agreement.
8. The help files used by OptdesX (`*.hlp`)
9. An X-Windows default resources file, which can be used to customize colors and fonts (`user.Xdefaults`)

The diagram below shows the directory structure of these files on the tape. After installation, the same directory structure should exist on your system.



Directory tree

1.2 Requirements

OptdesX requires the user to write and compile a short program which contains (or calls) the user's engineering model. This program may be written in either C or Fortran. The templates for these programs can be found in `anasubC.c` and `anasubF.f`, respectively. Depending on which programming language is preferred, either a C or Fortran compiler is required to use OptdesX.

1.3 Installation Procedure

Note: You must be logged in as “root” when the following steps are performed so that the system “owns” the software and the appropriate X-Windows defaults files can be created.

1. Make your current directory the one where you will install OptdesX. For example:

```
cd /usr/local
```

2. Load all the files from the tape into that directory. The tape was created with the standard Unix tar command. If the default device for tar is the tape drive then type the following to load the tape into your directory:

```
tar -x
```

If the default device for tar is something other than the tape drive, specify your tape drive using a command such as:

```
tar -xf /dev/<tape>
```

where the keyword <tape> is replaced by the name of your drive.

3. Change directories to the OptdesX directory.

```
cd OptdesX
```

4. Run the setup program, which will prompt you for the directory path in which the software was installed, the name of your company, your license number, and your validation key. Your company's individual license number and validation key should have been printed on a card and sent to you or placed in the inside front cover of the user's manual. If you have lost this card, call Customer Support at Design Synthesis. The setup program is run simply by typing its name, as shown below:

```
setup
```

When you are finished with this step, replace your license key card in the inside front cover of the user's manual, or store it in some other safe place. You will need to refer to your license number when calling customer support.

5. Generate the OptdesX software license agreement by running the genLicense program. To invoke the program, simply type the following:

```
genLicense
```

The genLicense program will prompt you for the type of software license you are buying, such as a Stand Alone, or Network-3 license. It will also prompt you for the name of your company, the address of your company, and whether or not you are buying the Robust Design Toolpack. The genLicense program will then look in the current directory for an info file corresponding to each machine on which OptdesX will be run. For example, suppose you are installing a Stand Alone Workstation license on a machine called gandalf. The genLicense program will look in the current directory for a file called gandalf.info. If the file is not found, genLicense will invoke another program, called getinfo, to query the system and create the gandalf.info file. The info file contains such information as the computer make and model, operating system version, node name, and the CPU ID number. This information is

made a part of the license agreement between your company and Design Synthesis, and is stored in a text file called `LicenseAgreement` in the current directory. *This licenseAgreement file should be printed out, signed and returned to Design Synthesis as soon as possible.* The installation instructions described above are sufficient to install OptdesX on a single machine. If OptdesX is to be installed on multiple machines, the process is still quite simple, but depends heavily on the configuration of your network. You should call Design Synthesis for specific instructions.

6. As a security device, the license validation key delivered to you with the software is only good for 45 days, after which OptdesX will cease to operate unless you receive and install a new validation key from Design Synthesis. A new validation key, generated specifically for your CPU, will be sent to you immediately upon receipt of a signed license agreement and full payment for the software. This new validation key, good for the remainder of your license period, should be installed on your system using the `setup` program described in step 4 above. The `setup` program is even easier to run the second time, however, since all the information entered the first time is saved and presented as default values. Only the validation key needs to be re-typed.

7. The program is now installed, and root privileges are no longer needed. However, each OptdesX user should add the path to the OptdesX executable (eg `/usr/local/OptdesX/bin`) to the `PATH` variable in their C shell initialization (`.cshrc`) file. This is so that when the user issues the OptdesX command, Unix will know where to find the OptdesX executable. For example, suppose a user's path is currently set as shown below:

```
PATH = /bin
```

The `PATH` variable should be changed to be something like what is shown below:

```
PATH = /bin /usr/local/OptdesX/bin
```

8. To insure that the software is working properly, the user should run through the tutorial found in Section 3.3 of this manual. To execute the tutorial, copy the `Twobar` executable from `OptdesX/examples` to a directory in your own area and type:

```
OptdesX Twobar
```

1.4 Linking Your Own Model to OptdesX

Detailed instructions for linking your model to OptdesX are given in Sections 4.2 and 4.3 of the manual.

1.5 OptdesX Colors, Fonts

1.5.1 Colors

The OptdesX colors may be altered to suit your preferences. Copy the `user.xdefaults` file located in the `OptdesX/user` directory to your local directory. After changing the colors in this file, merge it with the `.xdefaults` file in your home directory so that your preferences will take effect automatically when you log in.

For example, if you would like to have a grey background with white text and a red help menu bar, you would change lines in the `.xdefaults` file to look like the following:

```
OptdesX*foreground:"white"  
OptdesX*background:"grey50"  
OptdesX*menu*background:"red"
```

You can also specify colors as hexadecimal numbers, which we have done for the default colors--which will give you a slate blue background with white text.

For IBM, Sun and Silicon Graphics systems: to merge this file with your current `.Xdefaults` file, enter the following command:

```
cat user.Xdefaults >> .Xdefaults
```

(Make sure you type `>>` and not `>` or you will overwrite your current `.Xdefaults` file.)

To make these changes take effect you must type the following command:

```
xrdb -merge .Xdefaults
```

For DEC Ultrix systems: to merge this file with your current `.Xdefaults` file, enter the following command:

```
cat user.Xdefaults >> .Xdefaults
```

(Make sure you type `>>` and not `>` or you will overwrite your current `.Xdefaults` file.)

To make these changes take effect, log out and log in again.

For HP systems running HP VUE: Assuming you are on a computer named "computer1", to merge this file with your current `vue.resources` file, enter the following command:

```
cat user.Xdefaults >> ~/.vue/sessions/current/vue.resources
```

To make these changes take effect, type the following command:

```
xrdb -merge ~/.vue/sessions/current/vue.resources
```

Your user environment should now start up with the colors you specified.

1.5.2 Fonts

OptdesX was designed to use a particular, standard set of X fonts. Do not change these fonts. If you find that fonts do not come up properly, there are a couple of things you should check.

1. Check to make sure the fonts are listed in one of the `fonts.dir` or `fonts.alias` files in the `/usr/lib/X11/fonts/*` subdirectories. You should `grep` for the following strings:

```
grep "fixed-medium-r-semicondensed--13-120-75-75-c-60"  
fonts.dir fonts.alias
```

and

```
grep "helvetica-bold-r-normal--12-120-75-75-p-70" fonts.dir  
fonts.alias
```

If these strings are found, check to see if the file associated with these fonts has accidentally been deleted. If these strings are not found, look for a font that has similar specifications and substitute that string for the font string found in the `sys.Xdefaults` resource file.

2. If the fonts have been installed in a directory other than the `/usr/lib/X11/fonts` directory, try use the following command to set the proper font path:

```
xset +fp /usr/local/yourfontpath/
```

3. If neither of these suggestions resolves your problem, contact Design Synthesis for support.

2 Overview

2.1 For People Who Hate to Read Manuals

Most people do not like to read manuals--they want to get something accomplished, and they want to read the minimum in the manual to learn to do what they want to do. With this in mind, we would like to indicate what that minimum is. You should first read this section, which is short, and which gives a broad overview of the software. Next, you should execute the tutorials in Section 3. These tutorials will introduce you to all of the features of the software. Sections 4.2 and 4.3 explain how to link your own model to OptdesX. Before proceeding to apply OptdesX to solve your own problem, however, it is very important that you read Section 8.4, "Essential Algorithm Information." This section covers the most common errors in using OptdesX and discusses how to avoid them.

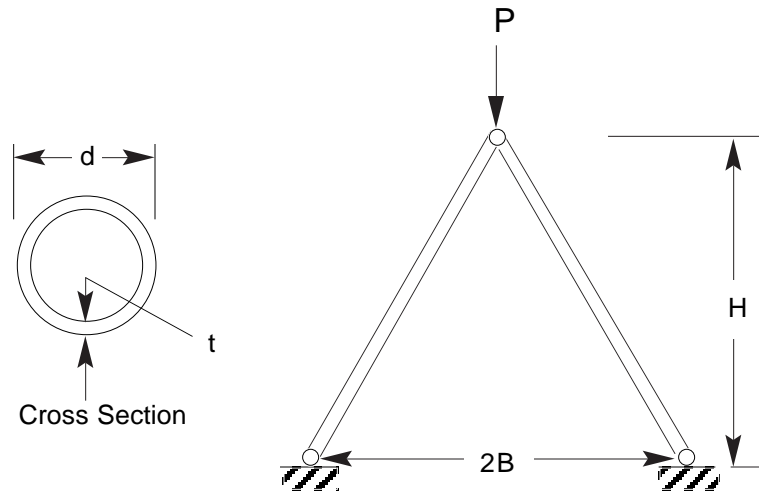
2.2 An Overview of OptdesX

OptdesX is an interactive computer program for computer-aided optimization and design. It has been developed to address the needs of an engineer in a design environment. OptdesX allows a designer to:

- adjust design variables to find a feasible design.
- adjust design variables to find an optimal design.
- optimize with multiple objectives and examine trade-offs among competing objectives
- optimize with both continuous and discrete variables, including components from vendor catalogs.
- define design problems that include sophisticated mappings from analysis to design space.
- redefine a design problem very quickly using point and click operations
- examine derivatives of the analysis model.
- determine the optimal derivative perturbation and method to minimize the effects of noise in the analysis model.
- examine the sensitivity of a particular solution to changes in the design variables.
- obtain graphical representations of the design space.
- plot the history of the design evolution.
- periodically save or recall a promising design for further reference or optimization.

2.3 OptdesX Terminology

To effectively use OPTDES it is important to understand some optimization terminology. Consider the design of a simple tubular two-bar truss shown below (problem originally from Fox, *Optimization Methods for Engineering Design*).



The Twobar Truss

The truss must be designed to withstand the load P without failing by yielding or by buckling. The designer may also be interested in the deflection at the point where the load is applied. A *design* of the truss is specified by a unique set of values for the *analysis variables*: height (H), diameter (d), thickness (t), separation distance (B), modulus of elasticity (E), and material density (ρ). A design is *analyzed* by evaluating a mathematical model of the truss, which in this case means calculating the following four analysis functions (These equations are referred to as analysis functions because they are functions of the above-named analysis variables.)

$$\text{Weight} = 2\pi d t \rho \sqrt{(B/2)^2 + H^2}$$

$$\text{Stress} = \frac{P \sqrt{(B/2)^2 + H^2}}{2\pi d H}$$

$$\text{Buckling Stress} = \frac{\pi^2 E (d^2 + t^2)}{8 [(B/2)^2 + H^2]}$$

$$\text{Deflection} = \frac{P [(B/2)^2 + H^2]^{3/2}}{2\pi d H^2 E}$$

Based on the results of the analysis, the designer makes a judgement as to the acceptability of the design.

The distinction between design and analysis is important. In simplified terms, design refers to

the process of deciding upon the values of the analysis variables. Analysis refers to the process of calculating analysis function values given the variable values. Most engineering software is analysis software. It enables the designer to predict the performance of a proposed design, but it does not tell how to change the variables to improve the design. Determining how to change variables to improve the design is the purpose of optimization software.

Normally not all of the analysis variables can be selected to be optimization variables. Some analysis variables are not free to be adjusted by the designer. In the case of the twobar truss, for example, the analysis variables density, load, modulus of elasticity and separation width are fixed. The analysis variables that are selected to be adjusted by the optimization algorithms are called the *design variables*. The design variables are a subset of the analysis variables. For the beginning tutorial in Section 3.2, only height and diameter are chosen to be design variables.

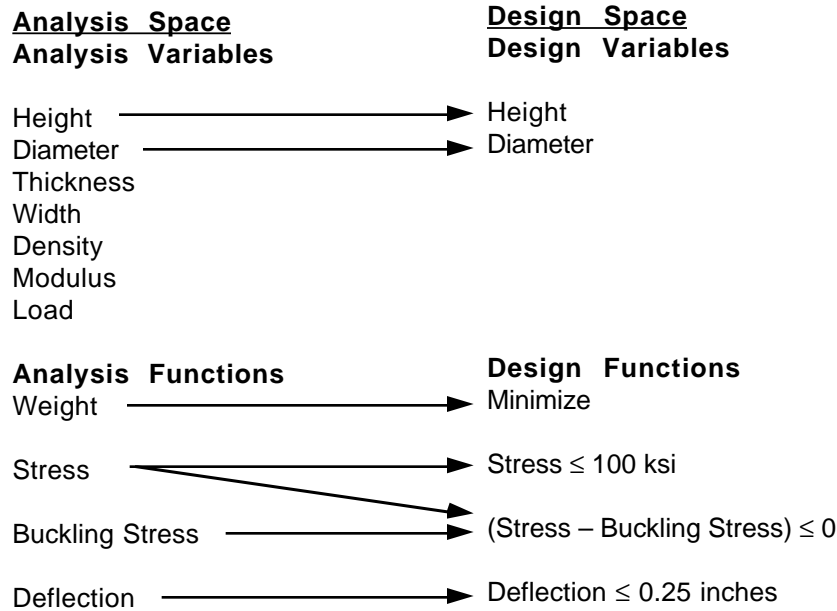
Once the variables are selected, the designer chooses the *design functions*, i.e., the objective functions and constraints. As with design variables, the design functions are a subset of the analysis functions. For one of the examples in the tutorial, weight is the objective function, and yielding stress, the difference between the yielding and buckling stress (it is this difference that determines whether the truss will buckle), and deflection are the constraints. Constraints may either be inequality or equality constraints. In this case all the constraints are inequalities. That is, yielding stress must be less than or equal to the yield strength of the material, S_y , buckling (the difference in yielding and buckling stress) must be less than or equal to zero, and deflection must be less than or equal to some specified value, D_u . We can state the optimization problem as:

Find Height and Diameter to:
Minimize Weight

Subject to Constraints:
Stress $\leq S_y$
(Stress-Buckling Stress) ≤ 0
Deflection $\leq D_u$

Note that this is just one of several optimization problems that might be posed for the twobar truss. Another problem would be to vary diameter, thickness and height such that stress is minimized and subject to constraints on weight and buckling (ignoring deflection). The specifying of an optimization problem, i.e. the selection of design variables and functions from the set of analysis variables and functions, is referred to as the *mapping between analysis space and design space*. The above mapping for the twobar truss, used often in this manual, is given below.

Mapping Analysis Space to Design Space The Twobar Truss



2.4 Scaling of Design Variables and Functions

Variables and functions in OptdesX are scaled internally using information provided during the optimization setup. Scaling can dramatically improve the performance of the optimization algorithms. The scaling formulae used by OptdesX are presented here so you can be aware of what is going on inside the software and enter data that will result in good scaling.

Each design variable has a minimum and maximum. These limits are used to scale design variables to be between -1 and 1, according to the expressions:

$$dv_i = \frac{\overline{dv}_i - C_{1i}}{C_{2i}} \quad \text{where} \quad C_{1i} = \frac{\text{Max}_i + \text{Min}_i}{2}$$

$$C_{2i} = \frac{\text{Max}_i - \text{Min}_i}{2}$$

where \overline{dv}_i is the unscaled value of the design variable i , dv_i is the scaled value, and Max_i and Min_i are the upper and lower limits, respectively.

Each design function has an allowable and indifference value. An allowable value is the limiting value for a constraint or the worst value you will accept. An indifference value is the value

below which you are indifferent to further improvement; it is the goal value for the function. When a design function is made a constraint, the allowable value is considered a hard limit which should not be violated at the optimum. When a design function is made an objective, the indifference value is considered a goal for the function; if the value reaches this goal, the objective essentially drops out of the problem while other objectives are improved. (If it is the only objective, however, it will not drop out but will be driven below its indifference value if possible.) The allowable and indifference values are used for scaling, according to the formula,

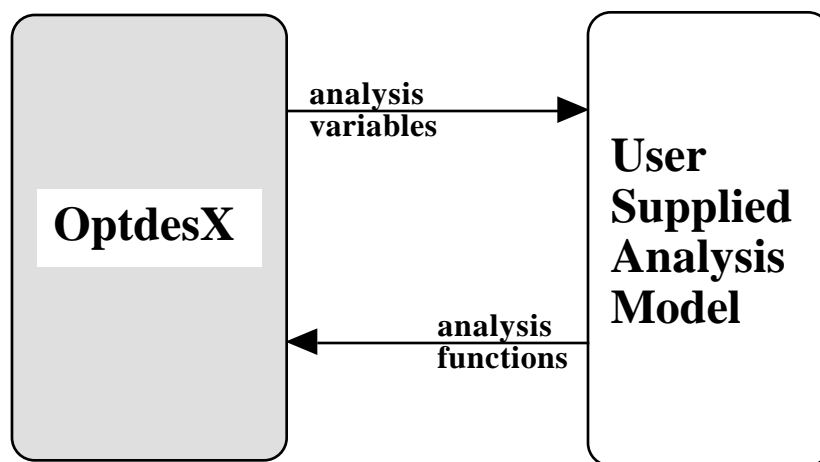
$$df_i = \frac{\bar{df}_i - C_i}{\text{Allowable}_i - \text{Indifference}_i} \quad \text{where } C_i = \begin{array}{l} \text{Allowable}_i \text{ for constraints} \\ \text{Indifference}_i \text{ for objectives} \end{array}$$

where \bar{df}_i is the unscaled value and df_i is the scaled value. Even if a function is not an objective, you should pick an appropriate indifference value (as if it were an objective) so that it can be scaled properly. If you are uncertain what appropriate indifference or allowable values should be, another rule to use is this: the difference of the allowable and indifference values, when divided into the function, should give a value between 1 and 10.

For “minimize” objectives or less than (\leq) constraints, the allowable value must be greater than the indifference value. The reverse is true for “maximize” objectives and greater than (\geq) constraints.

2.5 How OptdesX Communicates with Analysis Software

In order to apply OptdesX, you must supply analysis software for the problem of interest. During optimization, OptdesX will change the values of the analysis variables. It will then call the analysis model, passing in new variable values; the analysis model will calculate the corresponding analysis functions and pass them back. This operation is depicted in the figure below.



Operation of OptdesX and User-Supplied Analysis Model

The analysis model is a separate program written in C or Fortran, which communicates with OptdesX using Inter-Process Communication (IPC). Skeleton analysis programs, which contain the necessary IPC support routines, are provided for you in both languages. Section 4 gives detailed information on how to link your problem to OptdesX.

2.6 Philosophy of the OptdesX Windows Interface

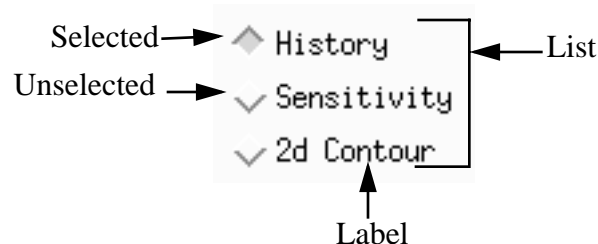
OptdesX is *window-driven* software, as opposed to menu-driven software. With the exception of a single menu in the Help facility, all actions in the software are initiated by opening and making selections in any of seven different windows. The look and feel of the interface is consistent across these windows. If, for example, you know how to select and open a file in one window, the procedure is the same in another.

Windows are manipulated through *widgets*. Widgets are specific interface components, such as buttons, lists, scroll bars, text fields, etc. OptdesX uses standard widgets from the Motif widget set. Operation of these widgets, as well as other general OptdesX procedures, are explained in the following section. Please read this section carefully, as many parts of the manual use terminology explained here.

2.7 Types of Widgets

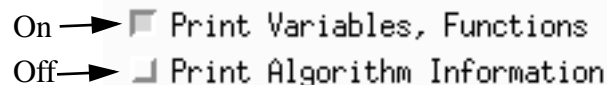
1. Radio Box

A Radio box is used to make one selection from several choices. As shown below, it consists of a list of diamond-shaped buttons with labels to the right of the buttons. Any new selection will deactivate the previous selection, just as with the buttons on a car radio. An item in a radio box is selected by clicking the mouse on the diamond box or the label.



2. Toggle Button

The toggle button is used to toggle between two states, usually on and off. A toggle button is selected or turned on by clicking on the toggle button or its label.



3. Pushbutton

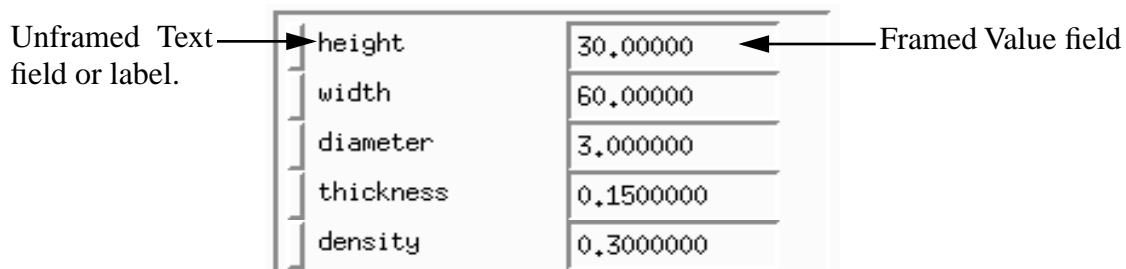
A pushbutton is used to start or cancel an operation as well as to open windows. A pushbutton has three states: unselected, selected, or dimmed. Two of these states are shown in the figure. A dimmed pushbutton indicates that the pushbutton cannot be selected. A pushbutton is

selected or pushed by clicking the mouse on the button.



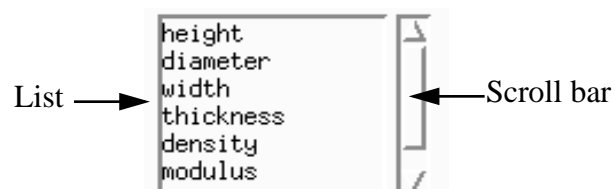
4. Text Field or Value Field

A text field holds a character string. A value field holds a numerical value. These fields can be framed or unframed as shown in the example. A framed text or value field indicates that that the text or value can be edited. An unframed field cannot be edited. A framed field can be edited character by character by clicking at the desired location and typing the new characters/ numbers, or the entire field can be replaced by double clicking (the entire field will then be highlighted) and typing the new value. If you have Motif 1.1 or higher, you can also tab through the editable fields in a window. The tabbing automatically highlights the value which can then be replaced by typing the new value. After entering a new value, you should register the value by entering a carriage return.



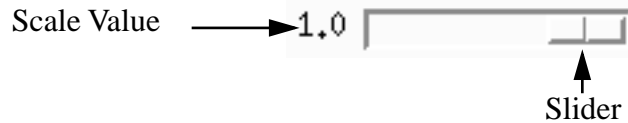
5. Lists

A List is vertical list of items represented by labels. There are three types of lists--single-select, extended-select, and multiple-select. A single-select list allows you to select only one item in the list by clicking on the label. Extended-select and multiple-select lists allow more than one item to be selected. In an extended-select list, items are selected by clicking and dragging. Items can be de-selected by pressing the control key while clicking on the label. Items are selected in multiple-select lists by clicking on each label. This acts as a toggle, so items can be de-selected by clicking again on the highlighted label. A scrollable list allows you to scroll through a list that is too long to display.



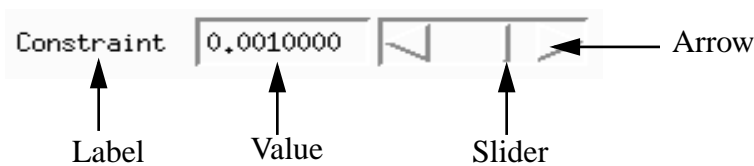
6. Scale slider bar

A scale slider bar is used to indicate a value from within a range of values. It is a sliding mechanism that is moved, by dragging with the mouse, until the proper setting is achieved.



7. Combination Scale slider/scroll bar

OptdesX uses a combination slider/scroll bar. This consists of a label, an editable value field, a slider bar and arrow heads. The value can be changed three ways: by editing the value field, by clicking on the arrows, or by dragging the slider



2.8 Window Features

2.8.1 Activating Windows

At any one time, only one window can *be active (or activated)*. The active window is the only window that can receive input from the keyboard or mouse. Typically a window is made active by either placing the cursor inside it or by clicking on it. When a window is made active, it is brought to the top, or brought forward, of all the windows on the screen.

Sometimes windows cannot be brought to the top using the mouse because they are completely covered up. As will be shown in the tutorial, a window can always be brought to the top in OptdesX by clicking its pushbutton in the Main Selection box of the OptdesX interface.

2.8.2 Resizing and Panning

A window can be moved on the screen by dragging its title bar to the new location. Some windows can be re-sized. Resizing is done by dragging the resize handles (which pop up when the cursor is placed on the border) to the desired size. Resize handles to enlarge a window proportionately are located in any corner. Except for graphics windows, all OptdesX windows that can be resized can only be resized in the Y direction.

The relative sizes of some windows inside main windows can be changed by *paning*. Windows that can be panned are separated by a thin horizontal line that ends with a thicker “handle” on the right, as shown below. Panning is done by placing the cursor on the handle (the cursor then changes to a “+”) and dragging.

BEFORE: part of the upper portion of the window is not shown as indicated by the scroll bar. The window will be paned to display all information

Function	Value	Type	#Map	Allowable	Indifference
weight	35.98735	↓	1	40.00000	10.00000
stress	33.01160	↕	1	100.00000	50.00000
stress-buckling	-152.5061	↕	2	0.000000	-50.00000
Sum	stress			33.01160	
-	buckling			185.5177	

Paning Handle

Dismiss Help

AFTER: The window has been paned. Note the change in relative sizes of the upper and lower boxes. The scroll bar is not displayed indicating that all the information is shown.

Function	Value	Type	#Map	Allowable	Indifference
weight	35.98735	↓	1	40.00000	10.00000
stress	33.01160	↕	1	100.00000	50.00000
stress-buckling	-152.5061	↕	2	0.000000	-50.00000
Sum	stress			33.01160	
-	buckling			185.5177	
deflection	0.06602320	↕	1	0.2500000	0.05000000

Dismiss Help

This window can also be made longer in the y direction by placing the cursor on the border and dragging. This is useful if you have a lot of functions and you do not want to scroll through them.

2.8.3 Tabbing Through Value Fields

When an optimization problem is first defined, many values often need to be changed. A convenient way to access value fields is by tabbing through them. In OptdesX, the tab key will move the cursor across a row in the Variable or Function windows. When tabbed, the field will be selected so you can immediately type the new value. Tabbing to a new field will automatically register the current value, i.e. you do not need to type a carriage return. The up arrow key will move the cursor up one row; the down arrow key will move the cursor down one row. Tabbing is further explained in the window below.

Pressing the tab key from this point will cause the next value field in this row (in this case, the #Map field) to be selected.

Pressing the down arrow key will cause the cursor to move to the row below it (in this case the row with diameter).

Variable	Value	Type	#Map	Minimum	Maximum
height	14.21299	C	1	10.00000	30.00000
diameter	1.690593	C	1	1.000000	3.000000
thickness	60.00000				
density	0.1500000				
modulus	0.3000000				
load	30000.00				
	66.00000				

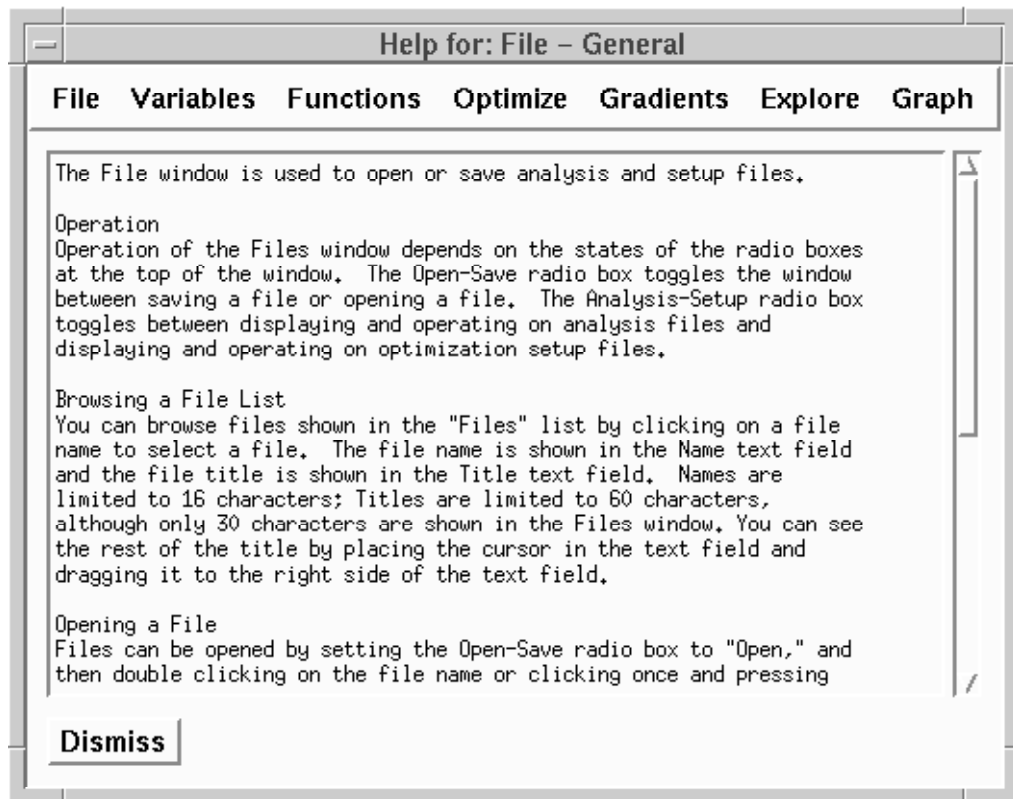
Dismiss Compute Functions Post Process Hardcopy Help

2.9 Dismiss and Help Pushbuttons

All windows in OptdesX contain a Dismiss pushbutton. In many cases this duplicates functionality included with the window manager; however, OptdesX was developed to be independent of a particular window manager. When closing a window, you should use this button rather than use a dismiss function provided by the window manager.

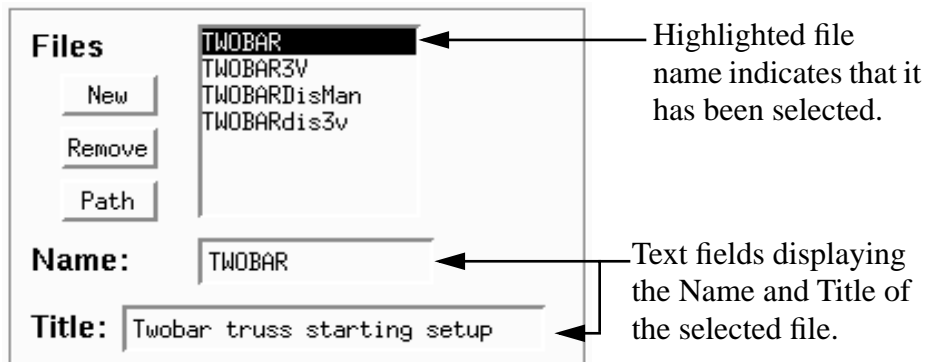
A main window that has been dismissed can be re-opened on the screen by pressing its button in the Main Window Selection box.

The Help pushbutton in a window provides reference information for that window. An example of a Help window is shown below. Any Help file can be accessed by selecting the Help file name from the menu at the top of any Help window.



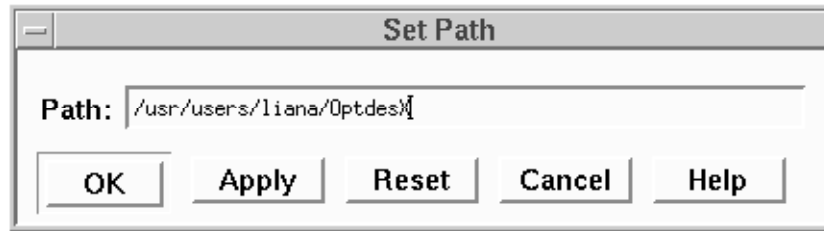
2.10 Files and File Selection

OptdesX creates and uses many different kinds of files. Each type of file is given a special extension when it is created. Windows which use files display them in a scrollable list, illustrated below. These windows screen the files according to their extensions and only display those that are appropriate for the window and options selected. File operations in all windows in OptdesX follow the same protocol: one click on the file name selects it. Its title and name are then shown in text fields. The file can then be opened by clicking the Open Pushbutton, or deleted using the Remove button. Alternatively, two clicks will select and open a file.



The path pushbutton is an option available in all windows where files are created or opened.

The path option is used to change the current or active directory. An example of the Set Path window is shown below. The current path is changed by editing the text field to indicate the new path and selecting the “OK” or “Apply” pushbutton.



3 OptdesX Tutorials

3.1 Introduction

In this section we will demonstrate the use of OptdesX by applying it to the twobar truss problem, which was introduced in Section 2. Features of the software will be explained in three separate tutorials. The first tutorial is the most extensive, covering the operation of all windows. The second tutorial focuses on optimization with discrete variables. The third tutorial focuses on gradients and scaling. The second and third tutorials build on the first, and you should execute all three if you wish to be proficient in using the software.

3.2 Linking an Analysis Model to OptdesX

The link between OptdesX and an analysis model is made in a subroutine called “ANAFUN.” This subroutine can be written in C or Fortran. OptdesX calls the ANAFUN routine and passes in the analysis variable values; the ANAFUN routine calculates function values and passes them back. If your analysis model is a stand-alone program, you use ANAFUN to write an input file for your model, execute the model as a stand-alone program, and read an output file from your model. More information on this is given in Section 4, “Linking Analysis Models to OptdesX.” If you have source code for your analysis model or are writing it from scratch, then you can calculate the functions directly in ANAFUN or call other routines from ANAFUN that make the calculations. The important thing to remember is that when execution returns from ANAFUN, OptdesX expects to have new function values. How the functions are computed is completely up to you.

Besides ANAFUN, there are two other routines you may use if you wish to (if you don’t use them, however, you must supply dummies). The ANAPRE routine is available for pre-processing; it is called once when OptdesX is first started. A typical use of ANAPRE is to initialize your model, perhaps by reading a data file. As will be shown, you also specify a model name in ANAPRE which OptdesX uses. The ANAPOS routine can be used for post-processing. ANAPOS can be called automatically during an optimization or directly from the windows interface. A typical use of ANAPOS is to display a picture of the design. You determine when ANAPOS is called. Collectively the ANAPRE, ANAFUN and ANAPOS routines are called the “ANASUB” routines. Data can be shared between the ANASUB routines using common blocks in Fortran or global variables in C.

The ANASUB routines for the twobar truss are given below. ANAPRE is used only to give the model the name, “Twobar Truss.” ANAPOS is a dummy. ANAFUN contains the analysis model--it calculates weight, stress, buckling stress and deflection. A C version of the ANASUB routines is given in Section 4.3.3. Files containing the ANASUB routines for the truss are included with the software in the `/OptdesX/examples/` directory.

Linking an Analysis Model to OptdesX 3-2

```
C=====
c...subroutine ANAPRE
c    pre-processing routine
C-----
    subroutine anapre(modelN)
    character*17 modelN

    modelN = 'Twobar Truss'

    return
    end
C-----
C=====
c...subroutine ANAFUN
c    Analysis routine for Twobar Truss
C-----
    subroutine anafun
    double precision hght,wdth,diam,thik,dens,modu,load,leng,
    &pi,area,iovera,wght, strs,buck,defl

c...input scalar AV values
    call avdsca(hght,'height')
    call avdsca(wdth,'width')
    call avdsca(diam,'diameter')
    call avdsca(thik,'thickness')
    call avdsca(dens,'density')
    call avdsca(modu,'modulus')
    call avdsca(load,'load')

c...intermediate constants
    leng = sqrt((wdth/2.d0)**2+hght**2)
    pi = 3.1415927d0
    area = pi*diam*thik
    iovera = (diam**2+thik**2)/8.d0

C...compute functions
    wght = 2.d0*dens*area*leng
    strs = load*leng/2.d0/area/hght
    buck = (pi**2*modu*iovera/leng**2)
    defl = load*leng**3/2.d0/modu/area/hght**2

c...output scalar AF values
    call afdzca(wght,'weight')
    call afdzca(strs,'stress')
    call afdzca(buck,'buckling')
    call afdzca(defl,'deflection')

    return
    end
```

← Anapre is used here only to give the model a name. The model name is limited to 16 characters.

← The analysis variables are passed in from OptdesX through these subroutine calls.

← The analysis functions are passed back through these subroutine calls

```

c=====
c...subroutine ANAPOS
c      post processing routine
c-----
c      subroutine anapos
c
c      return
c      end
c-----

```

At the beginning of ANAFUN we have a number of calls to a routine “avdsca,” such as:

```
call avdsca(hght, 'height')
```

This call has two arguments: the analysis variable value, which is passed in from OptdesX, and the analysis variable name. Analysis variable and function names are limited to 16 characters (however, because OptdesX must sometimes append a character to a name, the final character may be truncated, essentially limiting names to 15 characters).

The acronym “avdsca” stands for “analysis variable double scalar.” In the current release of OptdesX, all variables must be of this data type. In the future we anticipate supporting other types such as integers, vectors, and matrices.

The function values are passed back in calls such as,

```
call afdsca(wght, 'weight')
```

This routine passes both the function value and name to OptdesX.

3.3 Tutorial 1: Optimizing with Continuous Variables

In the tutorial, actions to be taken by you are numbered and are in bold type. After completing the tutorial you should be able to:

- Start OptdesX
- Create an Analysis file
- Perform design trial and error
- Save/Restore an analysis or optimization problem setup
- Set up an optimization problem
- Determine the best gradient method
- Optimize with continuous variables
- Create and graph History files
- Modify an optimization problem
- Explore and graph design space

Every time a new window is introduced, a reference diagram is shown that explains each feature of the window. You may read as much or as little of these reference diagrams as you wish.

Starting OptdesX

For this tutorial, the OptdesX routines are linked with the Twobar example analysis program provided with the software. The tutorial program should have already been created by the installer of OptdesX; it is usually placed in a directory such as `/usr/local/OptdesX/tutorial/` with the default name `OptdesX`. Instructions for the creation of the tutorial program are given in Section 1.

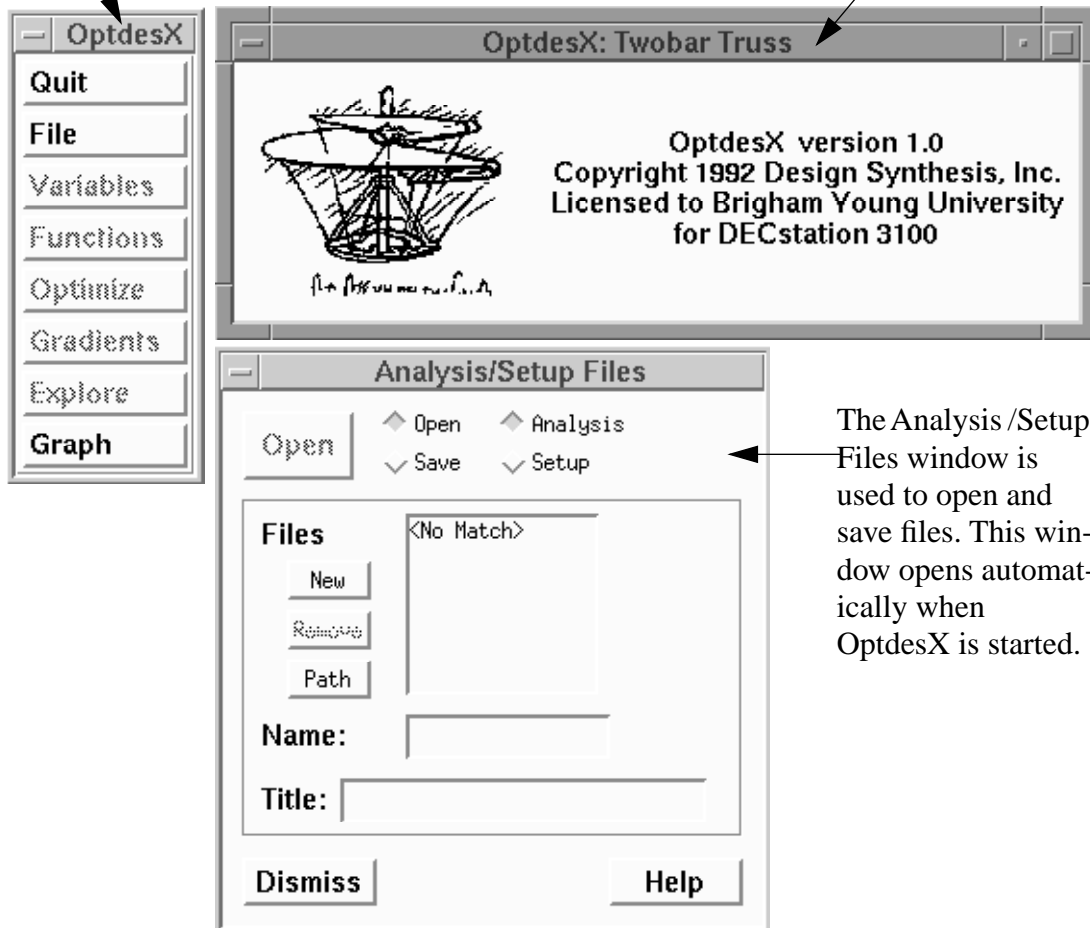
The tutorial displays presented in this manual were developed on a Unix-based DECstation 3100 computer. If you are executing OptdesX on another type of computer, there may be minor differences between the numerical values in this manual and the values on your screen.

1 Begin execution of the OptdesX software.

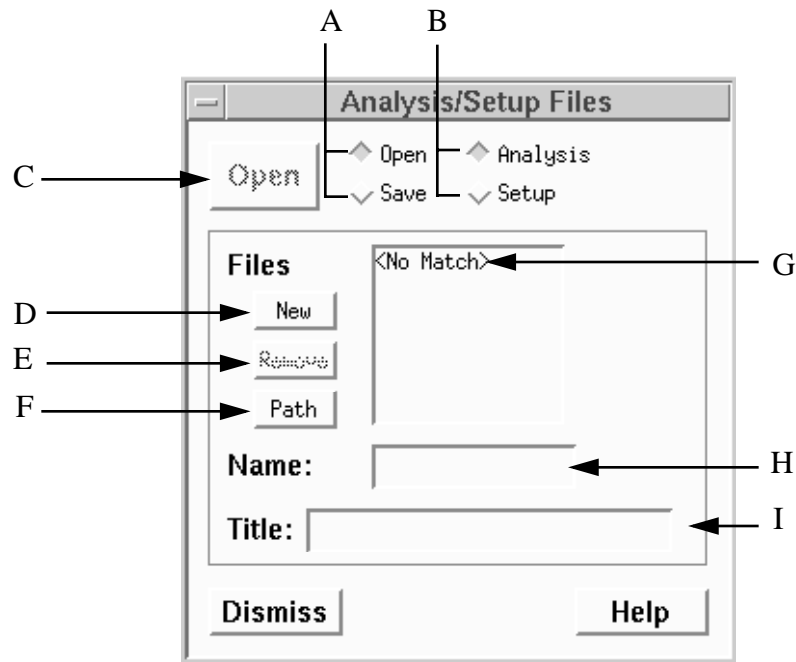
On a UNIX machine, assuming the tutorial is placed in the directory given above, you would type `/usr/local/OptdesX/tutorial/OptdesX`. The following three windows open:

This is the Main Window Selection box. All other windows are opened from here.

This is the "parent" window of all OptdesX windows. The title includes the model name. This window can be used to iconify the whole application.



Reference information for the File window is given on the next page.



A - Open-Save Radio box.

B - Analysis-Setup Radio box.

C - File pushbutton used to open or save an Analysis or Setup file.

D - New pushbutton used to create a new Analysis file.

E - Remove pushbutton used to remove or delete the selected file. This button is dimmed in the figure because a file has not been selected.

F - Path pushbutton used to change the current directory by opening the Path window.

G - Scrollable list of Analysis or Setup files (depending on the state of the Analysis-Setup radio box) in the current directory. The “No Match” message indicates there are no Analysis files in the current directory.

H - Name text field used for the Analysis or Setup file name. File names are limited to 16 characters.

I - Title text field used for the Analysis or Setup file title. Titles are limited to 60 characters. The special characters <Date> cause the current date and time to be inserted into the title.

File operations in all windows in OptdesX follow the same protocol: one click on the file name selects it. Its name and title are then shown in the Name and Title text fields. After selecting a file, it can be opened by clicking the Open pushbutton, or deleted using the Remove button. Alternatively, two clicks will select and open a file.

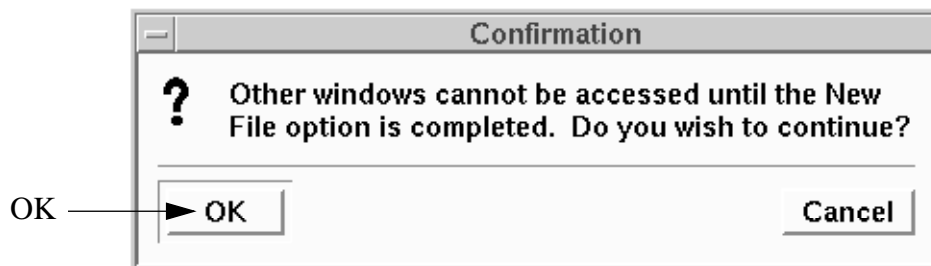
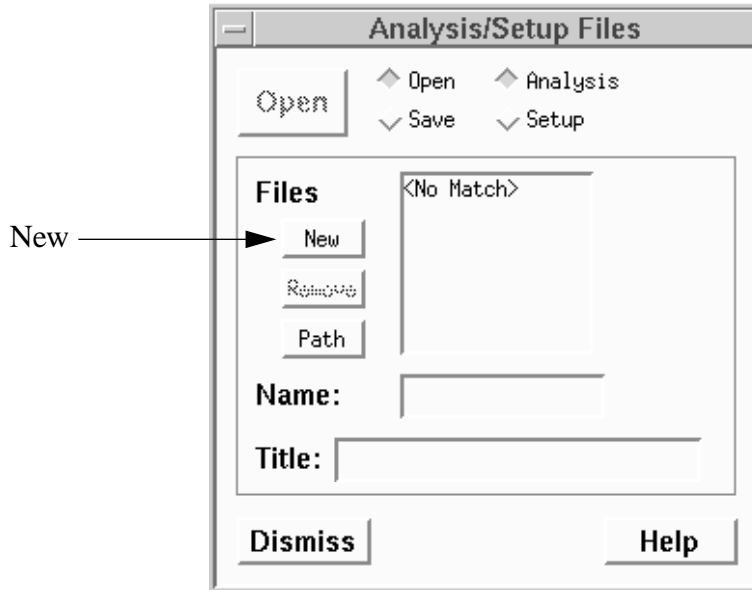
Creating an Analysis File

At the start of a new problem, an Analysis file must be created that contains initial values for variables and functions. This Analysis file can be created in two ways: by typing the file manually using your system editor or by using the OptdesX “New File” facility. In this section you will use the New File facility to create an Analysis file.

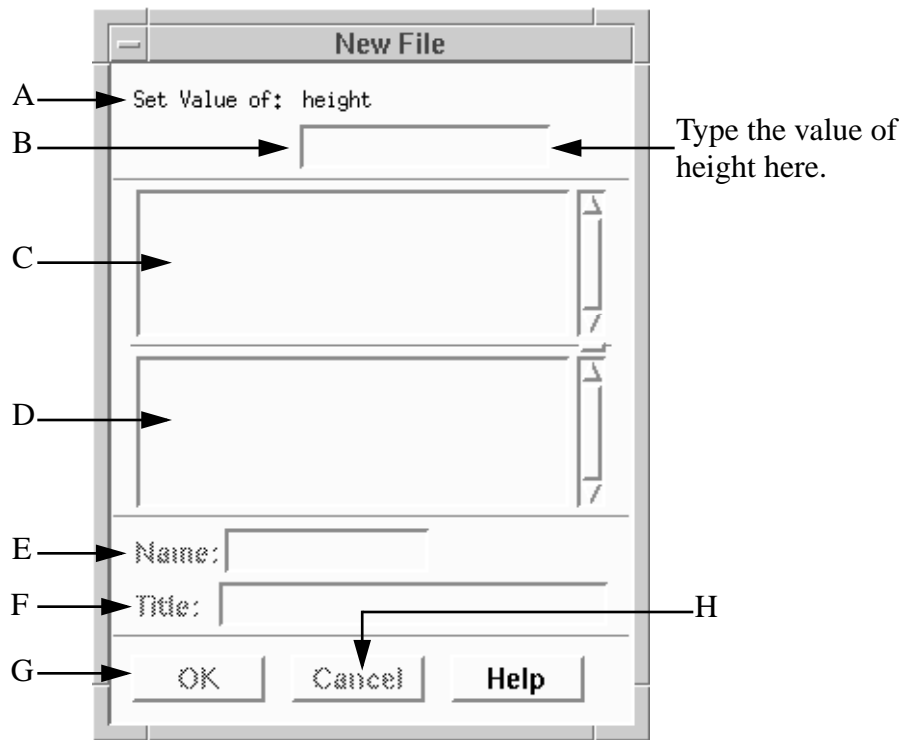
The New File facility is usually only used the first time OptdesX is executed with a particular

analysis model. Afterwards Analysis files are more easily created by changing the design in OptdesX and pushing the “Save” button.

2. Select the New pushbutton in the Analysis/Setup Files window. Click OK in the Confirmation box.



The New File window is modal, meaning that once this option is selected it must be completed before anything else can be done. The Confirmation box allows you to confirm that this is what you wish to do. The New File window will open on your screen.



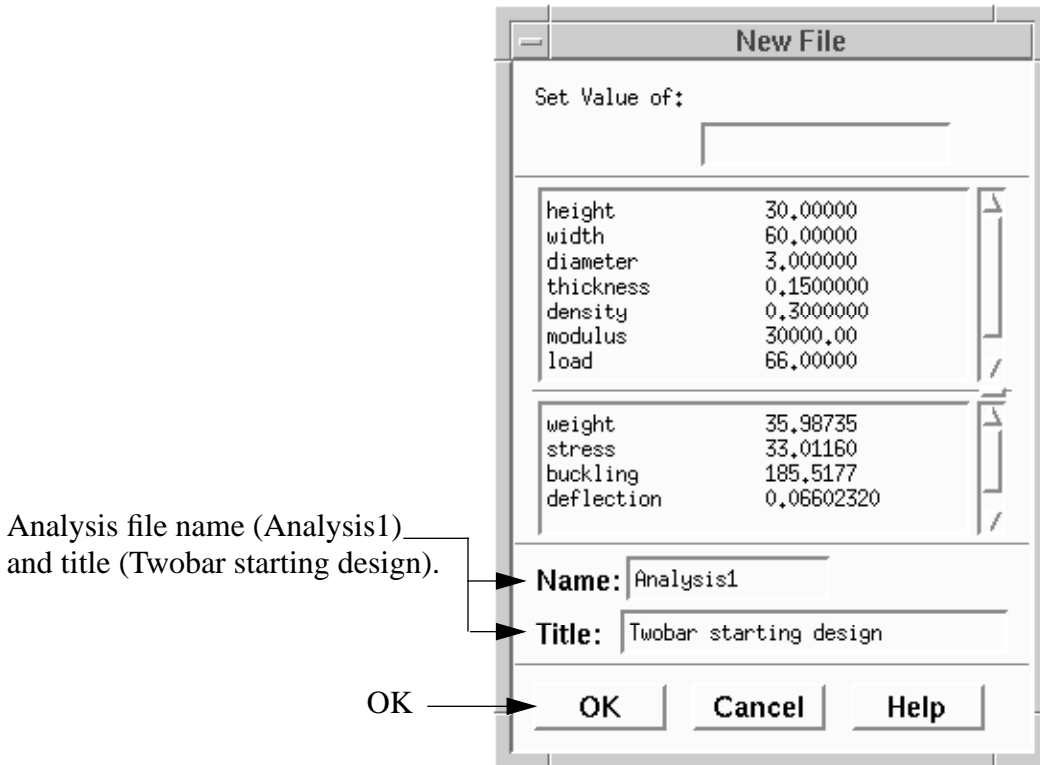
- A - Name text field. This field acts as a prompt--you are to type the value for the variable in the value field below it.
- B - Analysis variables value field.
- C - Scrolled window of analysis variables already entered.
- D - Scrolled window of analysis function values. These are computed after all variables are entered.
- E - Analysis file name. The file name is limited to 16 characters.
- F - Title text field. The title is limited to 60 characters (only 30 are displayed--to see the rest of a long title, place the cursor at the right boundary and drag).
- G - OK pushbutton used to save Analysis file after all values have been entered.
- H - Cancel pushbutton used to close the New File window without saving the analysis file. This option cannot be selected until all the variable values have been entered.

3. Insert the following values into the analysis variable value fields: height=30.0, width=60.0, diameter=3.0, thickness=0.15, density=0.30, modulus=30000.0, and load=66.0.

The New Window opens with only the value field for the first analysis variable displayed. Insert the value in the field by typing. Register the value by entering a carriage return. The variable will be displayed in the scrolled window below it, and you will be prompted for a new variable value. If you make a mistake but do not notice it until after the carriage return is entered, you cannot correct the error. You must enter values for the rest of the variables and then cancel and start over again.

Once all the variable values have been entered, the functions are calculated and displayed in the middle box of the New File window as shown below.

4. We will use the default name (“Analysis1”). Enter the title, “Twobar starting design.”



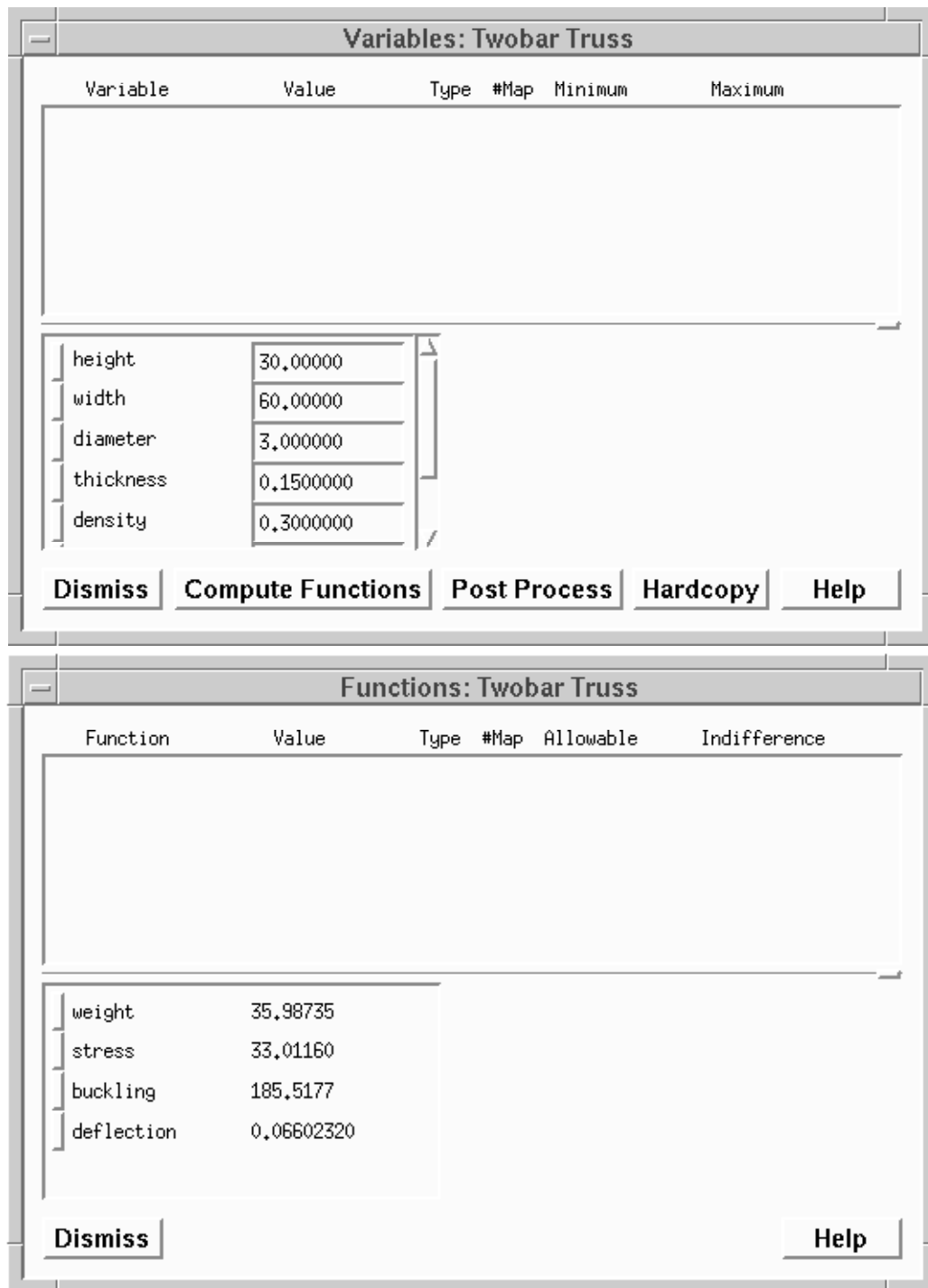
5. Select the OK pushbutton to save the new Analysis file.

The file “Analysis1.ana” has now been created. The “ana” extension is added by OptdesX and indicates that this is an analysis file. Note that Analysis1 has now been added to the Files List in the Analysis/Setup Files window (the .ana extension is not shown). The Analysis1.ana file could also have been created manually by typing the file using your system editor. A listing of the file is as follows:

```
Twobar starting design
Twobar Truss
d 0 0 height
30.0
d 0 0 width
60.0
d 0 0 diameter
1.00
d 0 0 thickness
0.150
d 0 0 density
0.30
d 0 0 modulus
30000.0
d 0 0 load
66.0
```

This file looks straightforward--it contains names and values--except for the “d 0 0” that precedes every name. This notation indicates to OptdesX that the name which follows is for a double precision scalar. All variables must be preceded by these characters. As mentioned, you could have skipped the “New File” option and just written this file manually using your system editor.

The Variables and Functions Windows are now opened. Note that the variables appear in a scrolled list.

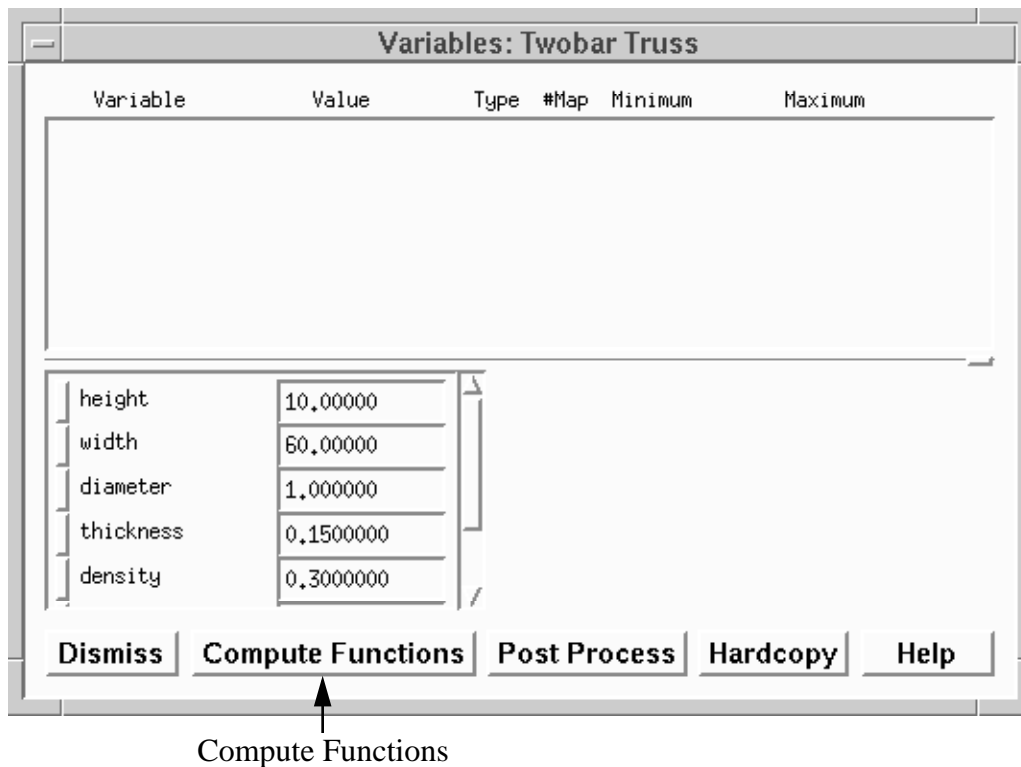


Design Trial-and-Error

Design trial-and-error is the process of selecting new analysis variables and computing the corresponding analysis functions. This is easily done in the Variables and Functions windows. You may wish to compute functions for several different designs.

6. Change height and diameter to 10.0 and 1.0 respectively.

Analysis variable values can be changed by placing the cursor in the value field and deleting/ changing the value character by character, or by double clicking to select the entire field and replacing it with a new one. As with all numerical values entered in OptdesX, register each value by typing a carriage return after it is entered. Alternatively, tabbing to a new field will register the number for the current field.



7. Press the Compute Functions pushbutton.

Compute Functions calls the ANAFUN routine to calculate the analysis functions. The analysis function values change to reflect the new variable values. Try changing the analysis variables values and computing functions on your own.

Restoring an Analysis File

At some point in the optimization process you may wish to use analysis variable values set previously and stored in an analysis file. We will restore the initial design we created when we first started the software.

8. Open the Analysis1 file by double clicking the file name, or clicking once and pressing the Open button.



After selecting the file, the File Name and Title are displayed. When the file is opened, a “Restoring Analysis” message is printed, and the variable and function values restored from the file are displayed in the variable and function windows.

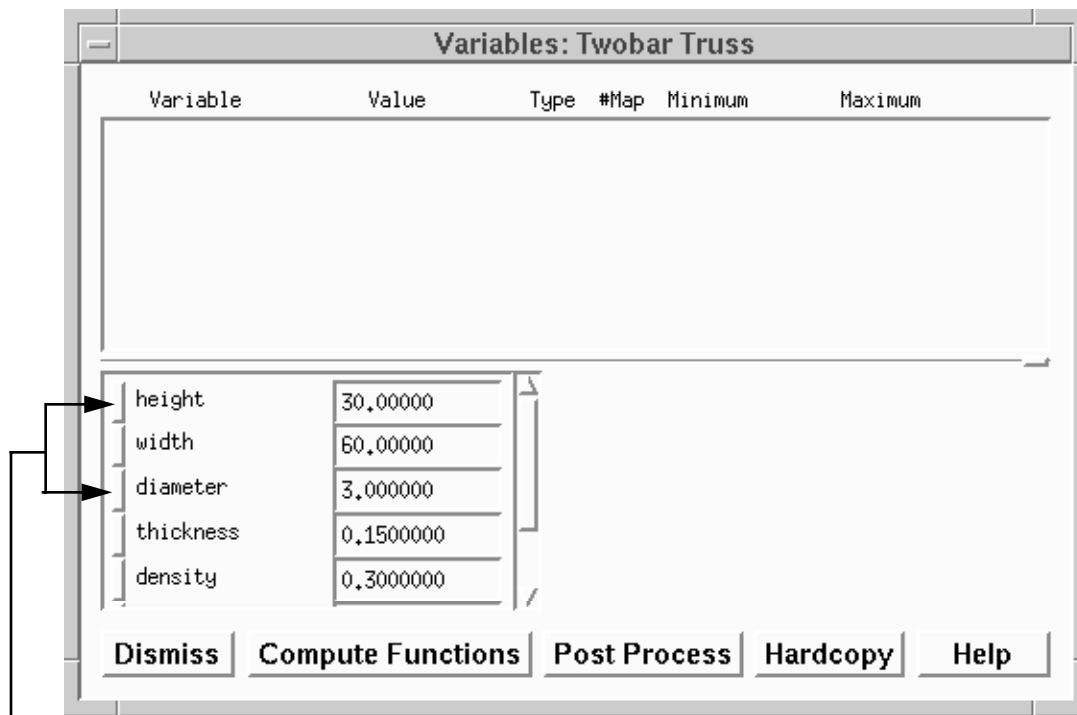
Optimization Problem Setup

An optimization problem is defined by specifying design variables, objective(s) and constraints. For this problem we will initially set diameter and height to be design variables; we will make weight the objective function, and we will select three constraints:

- Stress must be less than or equal to 100 ksi
- Stress minus buckling stress must be less or equal to than zero
- Deflection must be less than or equal to 0.25 inches.

Variable Setup

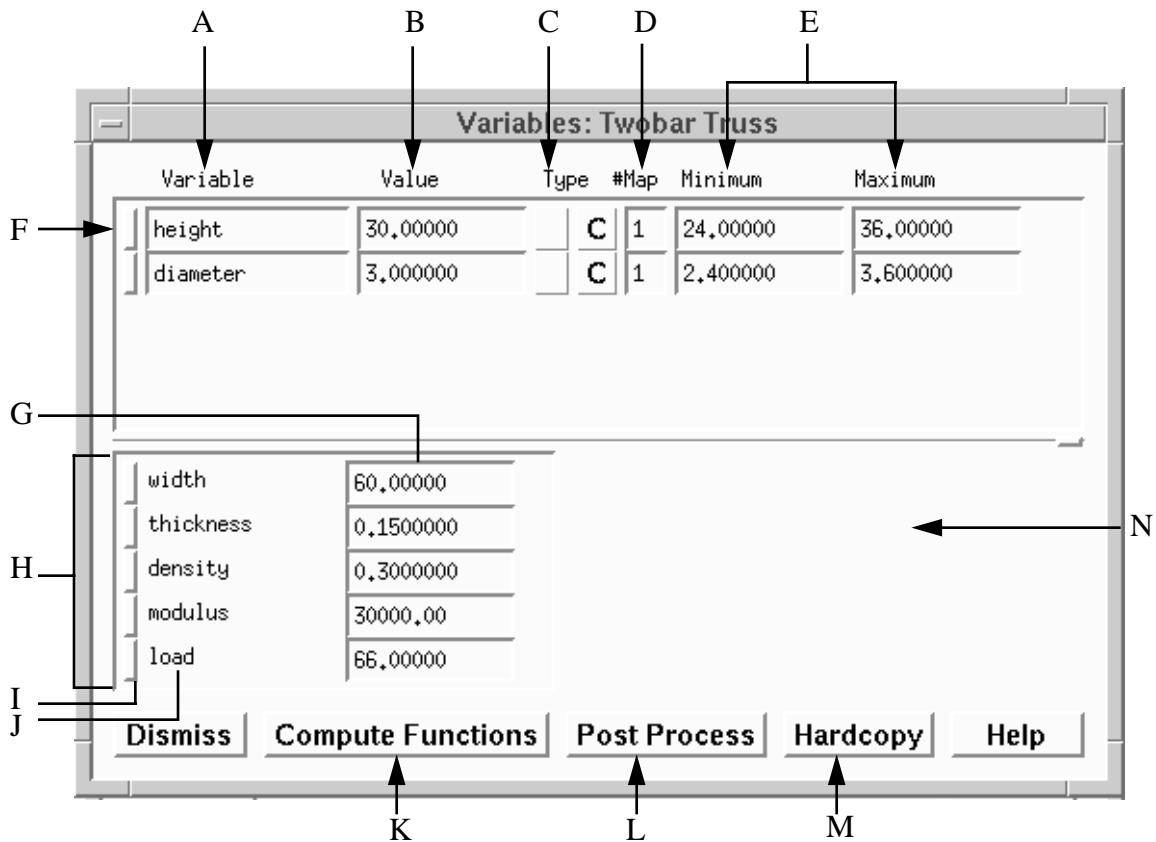
9. Select height and diameter to be design variables by pressing their Mapping buttons.



Mapping buttons.

When mapping buttons are pressed, the variables are deleted from the list of unmapped analysis variables (the list shown) and appear in the upper half of the window as design variables.

A reference diagram for the Variables window follows.



A - Design Variable Name text field. This can be edited as indicated by the frame around it. Names are normally changed only when the “#Map” field is made greater than one.

B - Design Variable value field. This field can be edited.

C - Type pushbuttons which indicate that a variable is at its upper or lower bound (left side) or that a variable is continuous or discrete (right side). Bound icons are shown below. A “C” indicates a continuous variable and a “D” indicates a discrete variable.



Bound icon indicating the variable is at its upper bound.



Bound icon indicating the variable is at its lower bound

D - # Map value field which displays the number of analysis variables mapped to one design variable.

E - Minimum and Maximum value fields. Default values are +/- 20% of the current value.

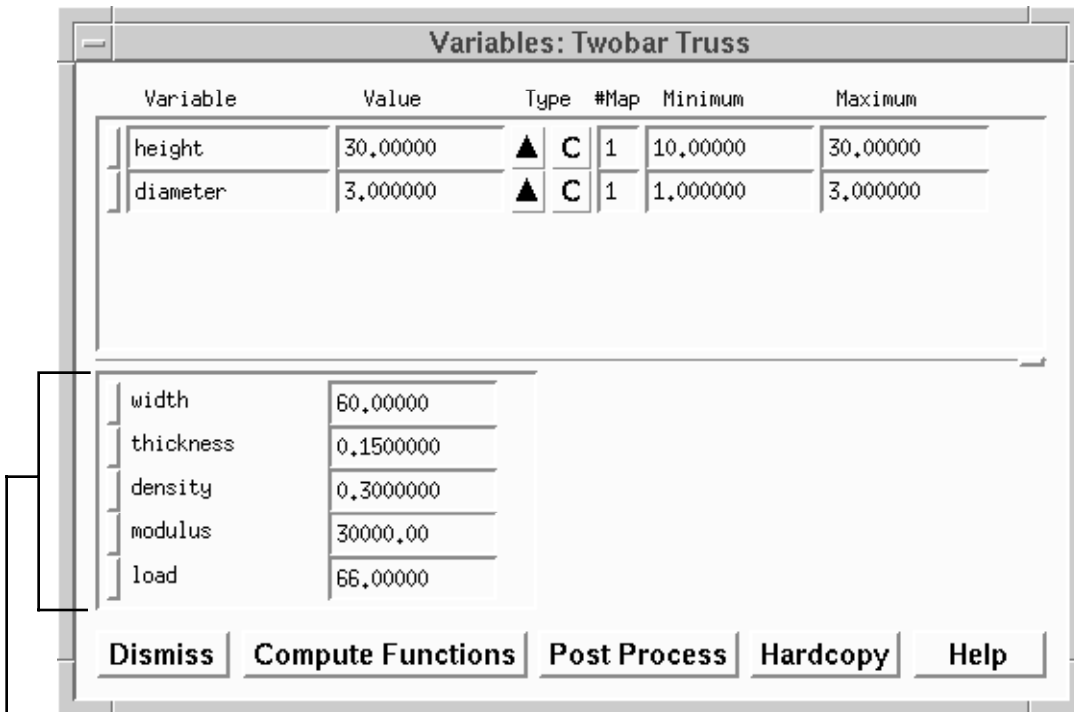
F - Unmapping pushbutton. Pushing this button deletes the design variable; it disappears from the list in the upper half of the window and reappears in the list for unmapped analysis variables.

G - Analysis Variable value field. This value remains constant during optimization. The value can be edited as indicated by its frame.

H - List of Unmapped Analysis variables.

- I - Mapping button used to map an analysis variable to a design variable.
- J - Analysis Variable Name text field. This text field can not be edited.
- K - Compute Functions pushbutton used to call ANAFUN to calculate the analysis functions.
- L - Post Processing pushbutton used to call ANAPOS.
- M -Hardcopy pushbutton used to obtain hardcopy of the setup information.
- N- Discrete variable information space. Discrete variables are discussed in the next tutorial.

10. Set minimum and maximum values to those shown in the window below.



Unmapped analysis variables

The triangle icon, called the “Bounds icon,” that appears under “Type” indicates that the variable is at its upper bound. A triangle icon pointing down would indicate that the variable is at its lower bound. Once the analysis variables are mapped to the design variable list both the name and value may be changed--although names would normally be changed only when the “# Map” field is made greater than one. The unmapped analysis variables remain constant during optimization; however, the constant value can be changed.

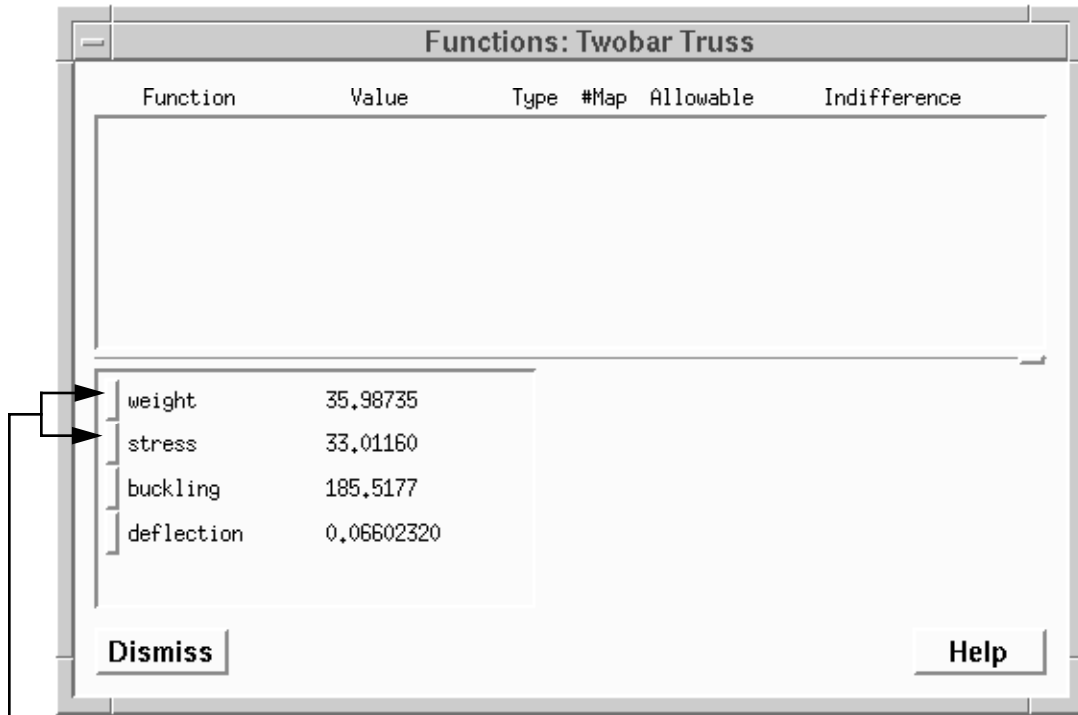
Function Setup

11. Map weight to be a design function by pressing the weight Mapping button.

12. Map and copy stress into the design function list by holding down the shift key while pressing the stress Mapping button.

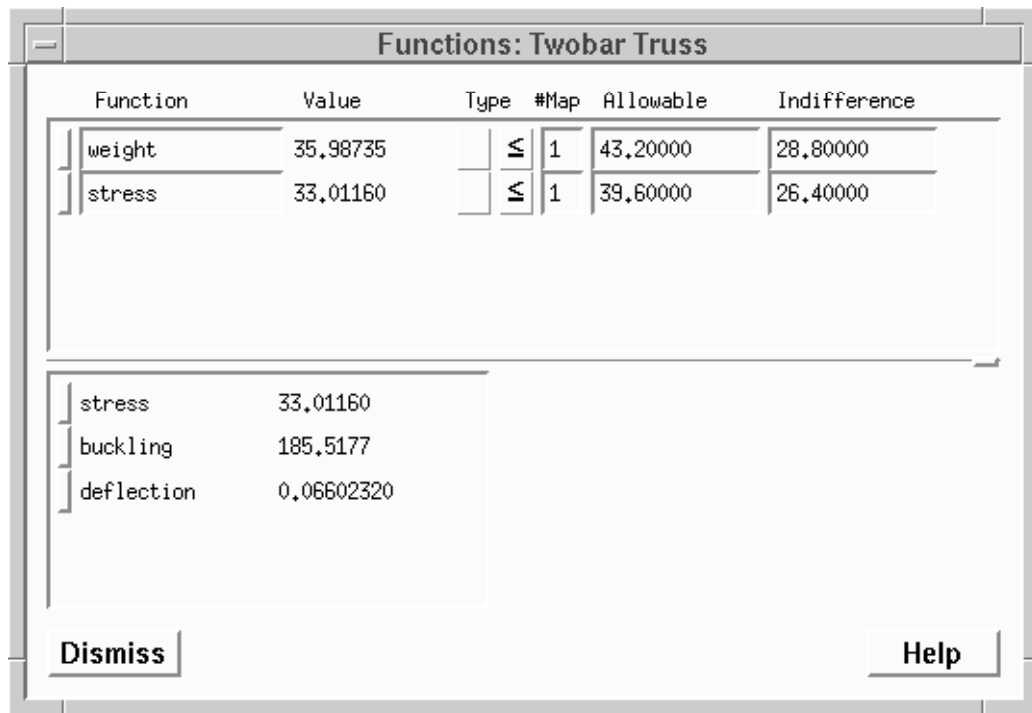
The stress function will be used in two constraints. It is therefore necessary to copy stress into

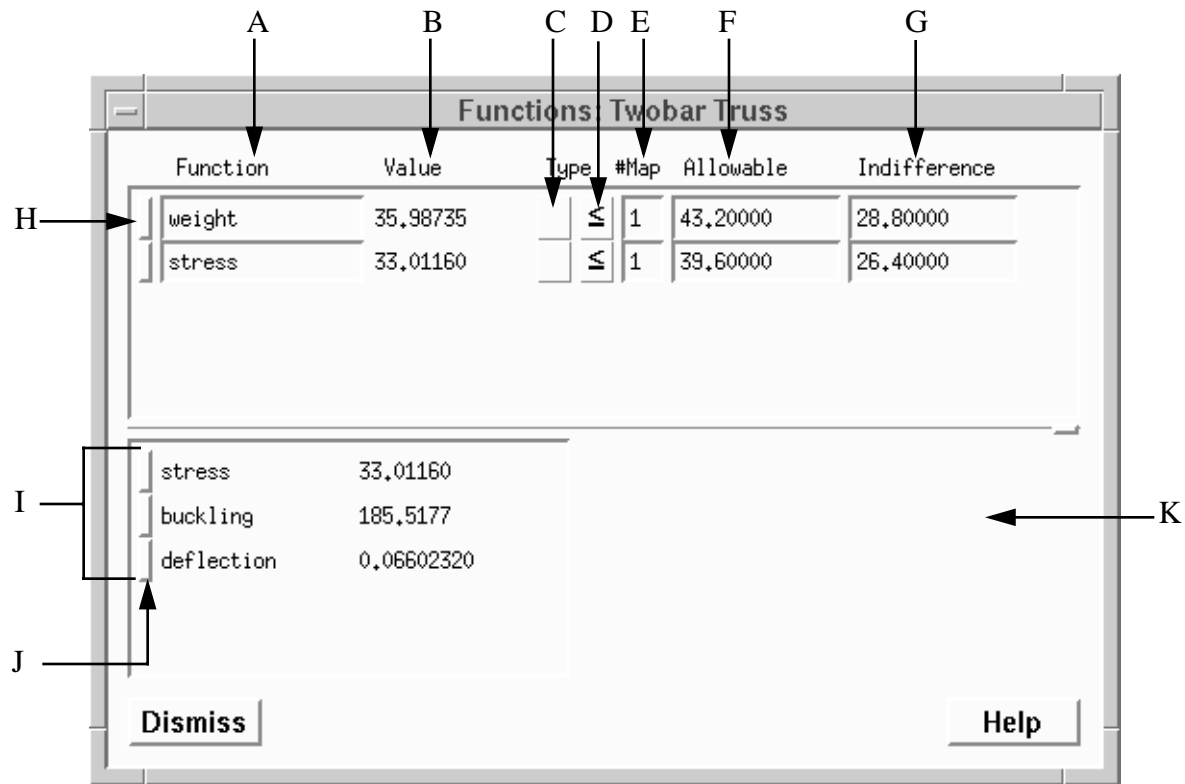
the design function list. A function is copied into the design function list by pressing the mapping button while holding down the shift key.



Weight and stress mapping buttons.

Your window should now look like:





- A - Design Function Name Text field. The name can be edited as indicated by its frame, although usually it is not edited unless #Map is greater than one.
- B - Design Function value fields. These values are calculated in ANAFUN.
- C - Objective button. The objective button is pressed until the appropriate icon is selected: downward arrow to minimize, upward arrow to maximize, and blank if not an objective. If an objective is at its indifference value its icon is boxed; if it is better than its indifference value it is highlighted.



Objective at Indifference Value



Objective better than Indifference Value

- D - Constraint button. Constraints can be blank (not a constraint), “≤,” “≥,” or “=.” The constraint button is pressed until the appropriate icon is selected. If a constraint is binding the constraint icon is boxed; if a constraint is violated the constraint is highlighted.



Binding Constraint



Violated Constraint

- E - #Map value field used to map two or more analysis functions to one design function.
- F - Allowable value field used as a limiting value for a constraint, or the “worst” value you will accept.
- G- Indifference Value field used as a goal value for an objective and beyond which

you are “indifferent” to further improvement, preferring instead that other objectives were improved.

H- Unmapping button used to unmap a design function.

I- List of Unmapped Design Functions.

J- Mapping button used to map analysis functions to design functions.

K- Multiple Objective Space--This space is reserved for objective weighting coefficient scales used with multiple objectives, as shown later in the tutorial.

13. Make weight an objective to be minimized by pressing the Objective button (left button under Type) until it is a downward arrow.

Functions are made objectives using the Objective button. Press the Objective button until the downward arrow is displayed. When you click on the Objective button, you will see blank (not an objective) or downward arrow (minimize), but not upward arrow (maximize). This is because the allowable value is greater than the indifference value. If you wish to maximize, you must first make the indifference value greater than the allowable value.

14. Remove weight as a constraint by pressing the Constraint button (right button under Type) to show “blank.”

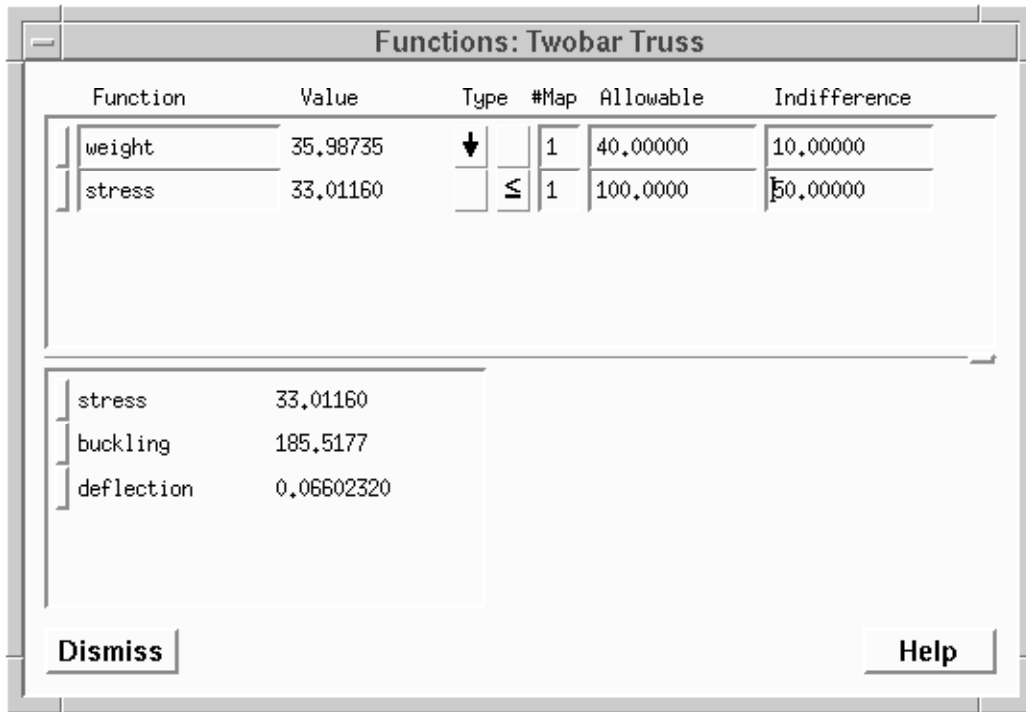
Functions are made (or removed as) constraints using the Constraint button. Press the Constraint button until “blank” is displayed. When you press the Constraint button, you will see blank (not a constraint), “ \leq ” or “ $=$ ”, but you will not see “ \geq ”. This is because the allowable value is greater than the indifference value. If you wish to make a function a “ \geq ” constraint, you must first make the indifference value greater than the allowable value.

15. Change allowable and indifference values to those shown in the window below.

The allowable value is the limiting value for a constraint, or the worst value you will accept. The indifference value is the value below which you are “indifferent” to further improvement, and would prefer instead that other objectives were improved. It is the goal value you wish to achieve.

All functions must have allowable and indifference values. OptdesX uses the difference of these values to define a range for scaling. Even, if, for example, a function is not an objective, you should pick an appropriate indifference value (as if it were an objective) so that OptdesX can scale the function. Scaling is important! If you are uncertain what appropriate indifference or allowable values should be, another rule to use is this: the absolute value of the difference in the allowable and indifference values, when divided into the function value, should give a number between 1 and 10.

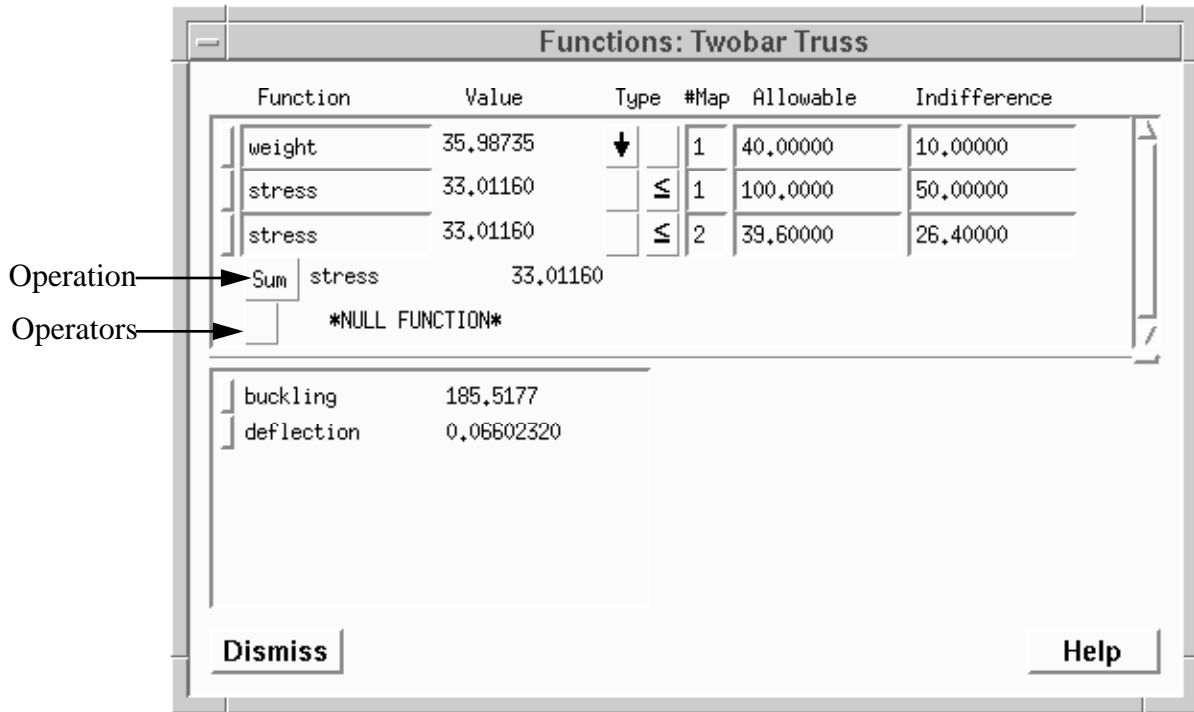
As noted, for “minimize” objectives or less than (\leq) constraints, the allowable value must be greater than the indifference value. The reverse is true for “maximize” objectives or greater than (\geq) constraints. Hopefully the reasoning behind this is apparent: for minimize objectives or less than constraints the goal value (indifference value) is less than the worst acceptable value (allowable value) and vice versa for maximize objectives or greater than constraints.



16. Make a second constraint using stress by pressing the Mapping button for stress.

17. Change #Map to “2” for stress in the design function list. Register this value with a carriage return.

This operation will map two analysis functions to one design function. This is necessary because the buckling constraint is the difference of two analysis functions (stress minus buckling stress).



When two or more analysis functions are mapped to one design function, the design function value is calculated as the sum, product, min or max of these functions, depending on the state of the Operation pushbutton. “Sum” specifies that the functions which follow will be added or subtracted; the operators are “-” and “+.” “Prod” specifies that the functions which follow will be multiplied or divided; the operators are “*” and “/.” “Max” or “min” return the maximum or minimum values of the functions respectively.

When #Map is changed to 2 or more, NULL FUNCTION labels are displayed for each function to be mapped. The next function(s) selected replace NULL FUNCTION.

18. Change the Null Function to buckling by clicking on the buckling Mapping button.

19. Change the sum operator to “~”.

20. Change the name of the design function to “stress-buckling” by double clicking in the name field and typing this new name.

There is a general rule being applied here. All of the design and analysis functions must have unique names. The design function defined as the difference of stress and buckling stress cannot be called “stress” because “stress” is already being used.

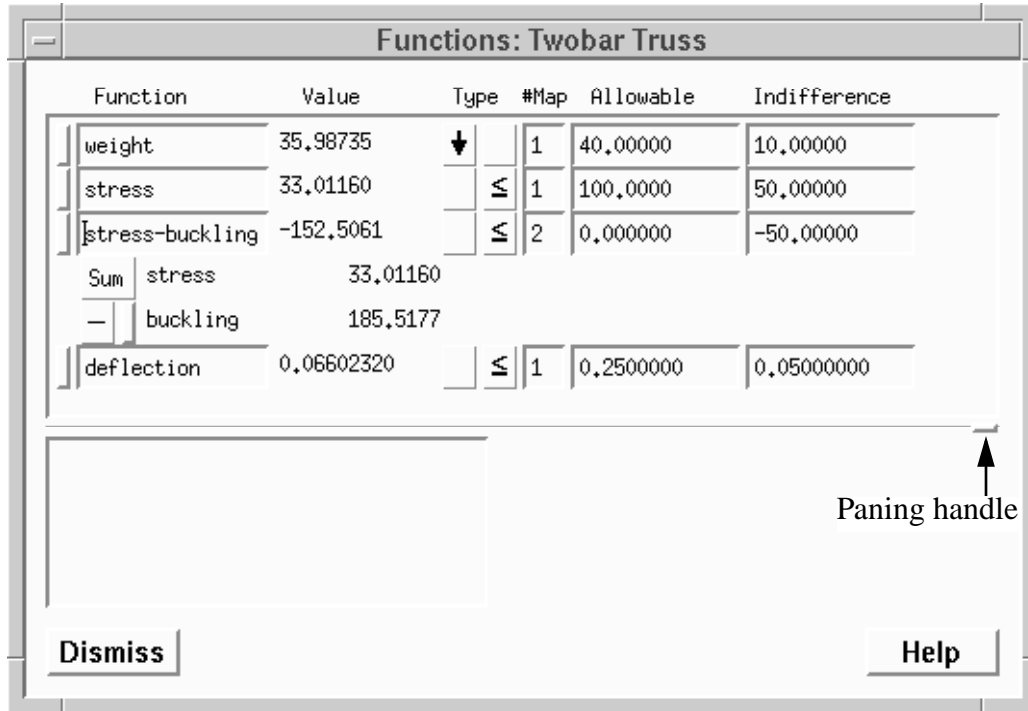
21. Set the allowable and indifference values to those shown in the window below.

22. Enlarge the design function box with the panning handle.

The Variable and Functions windows are separated by a fine horizontal line that ends on the right side in a “handle.” When the cursor is placed on this handle and dragged (the cursor

changes to a “+”) the relative sizes of the windows are changed.

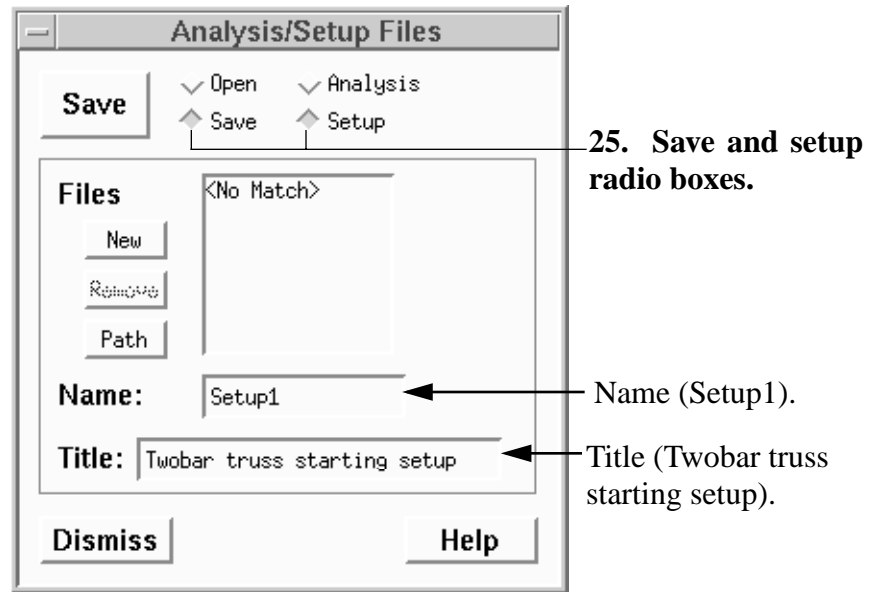
23. Make deflection a constraint by pressing the Mapping button for deflection. Set the allowable and indifference values to those shown in the window below.



Save Setup

A Setup file is saved to store the definition of the optimization problem. Once saved, a Setup file can be restored again at any time. It is important to understand the difference between a Setup file and an Analysis file. The Setup file contains the definition of the optimization problem; the Analysis file contains variable names and values. For a given analysis model, any Analysis file can be restored with any Setup file and vice versa. If a Setup file is opened prior to opening an Analysis file, the variable and functions values from the most current Analysis file are restored.

24. Set the Analysis/Setup Files window to “Save, Setup.”



26. We will use the default name (“Setup1”). Type the title, “Twobar truss starting setup.”

27. Press the Save pushbutton.

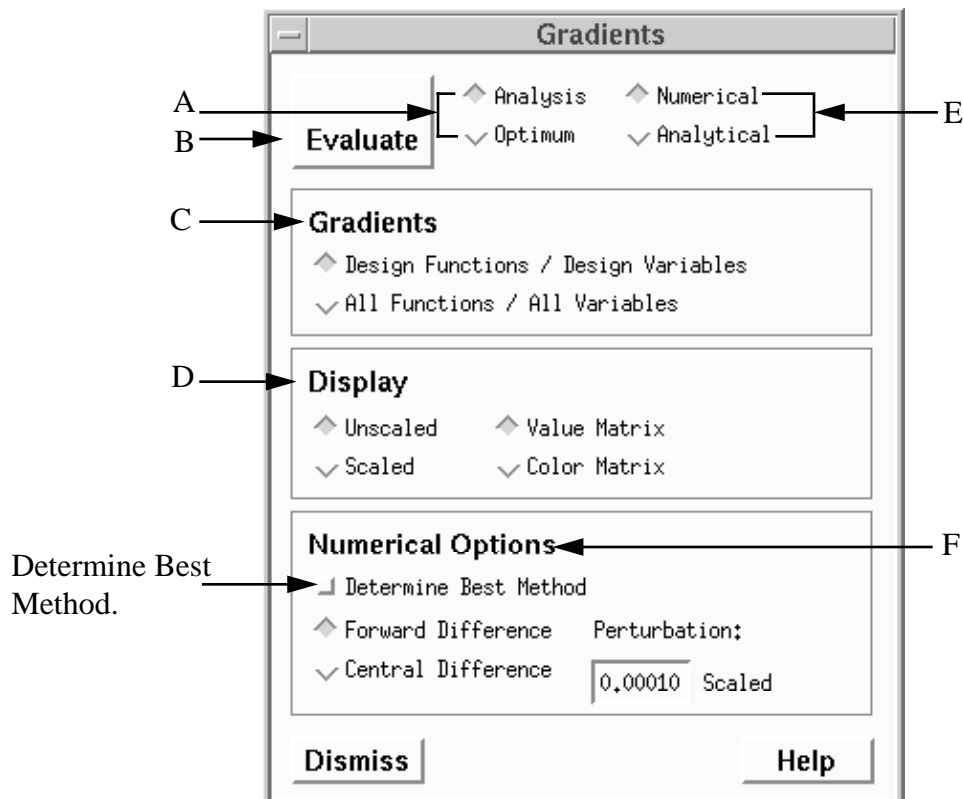
Setup1.opt has now been created and added to the Files list. When the Analysis-Setup radio box is set to “Setup,” only the Setup files are shown. In like manner, when the radio box is set to “Analysis,” only Analysis files are shown.

Optimize

After the problem is defined, the next step is optimization. OptdesX has five optimization algorithms--two for problems with continuous variables, and three for problems with discrete variables. In this tutorial we consider only continuous variables (in the next tutorial, we make the twobar truss discrete). The continuous algorithms in OptdesX require derivatives, or more precisely, gradients, which are vectors of derivatives. Gradients are calculated numerically by the software. In this section you will learn how to determine the best method for calculating gradients, and then you will optimize the truss using the generalized reduced gradient and sequential quadratic programming methods.

28. Select Gradients from the OptdesX main menu.

A reference diagram for the Gradients Window follows.

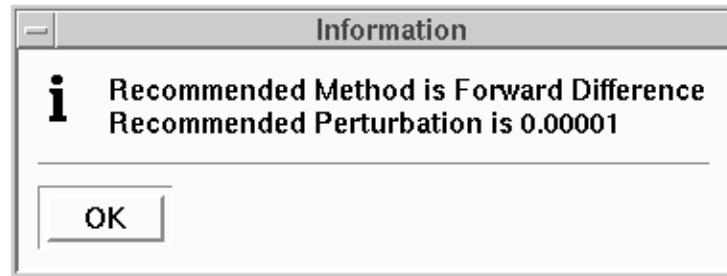


- A - Analysis-Optimum radio box used to select gradients of the analysis model or gradients of the optimal objective. To evaluate gradients of the optimal objective an optimization must first be performed.
- B - Evaluate pushbutton that causes the gradients to be evaluated. If the gradients are current the pushbutton changes to “Display.” Once gradients are taken, they remain current as long as the design point, the numerical method and perturbation are not changed.
- C - Gradients box used to select the gradients to be evaluated. Choices in this box are dependent on the state of the Analysis-Optimum radio box. When set to “Analysis” (the usual case) choices are “Gradients of design functions with respect to design variables” (the gradients the algorithms use) or “Gradients of all functions with respect to all variables.”
- D - Display box used to select the way gradients are displayed--either in scaled or unscaled form, and either as a matrix of values or a matrix of colors.
- E - Numerical-Analytical radio box used to select analytical or numerical gradients. The analytical gradients capability is currently not functional.
- F - Numerical Options box used to select the type of numerical method and perturbation value. The Determine Best Method toggle button is used to estimate the best numerical method and perturbation for a particular model.

29. Set the Determine Best Method toggle button in the Gradients window.

30. Press the Evaluate pushbutton.

An information box shown below will open with the recommended method and perturbation.



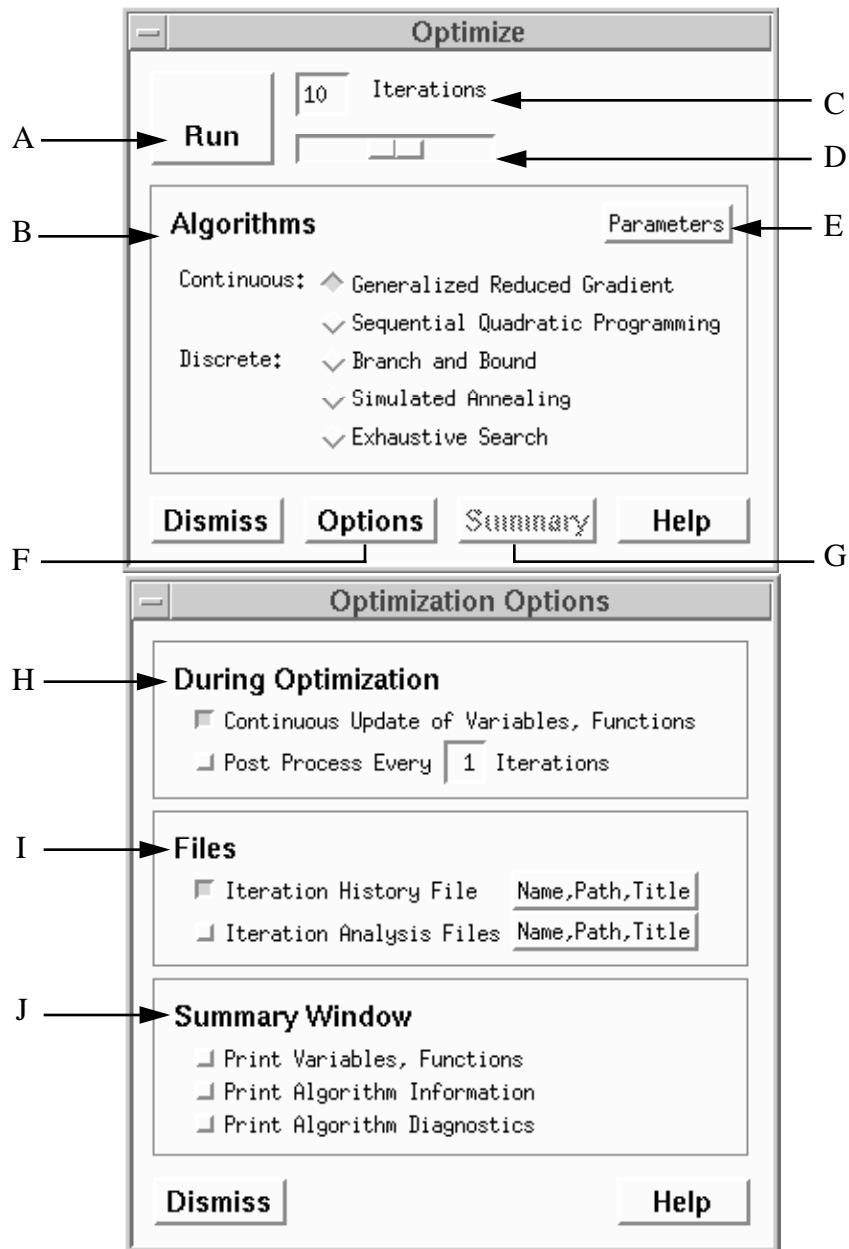
The Determine Best Method feature helps you to determine the best numerical method (central or forward difference) and the best perturbation for your particular analysis model. Analysis models are often “noisy,” meaning the function values lack significant figures because of approximation, solution method error (such as would result from the finite element method or numerical integration), or round-off error. When noise is present, it is often amplified in the derivatives, causing premature termination or failure of the optimization algorithms.

Using the design variable at the top of the design variable list, OptdesX will test your model for noise and then report what it believes the best method is. If you suspect noise is present, you should determine the best method for several variables (by modifying the design variable list so different variables are at the top). The worst case should then be chosen.

In this case, we have smooth functions, so a forward difference derivative with a small perturbation is recommended. The perturbation and method are automatically set to these values.

31. Press the OK pushbutton to dismiss the information window.**32. Dismiss the Gradients window by pressing the Dismiss button.****33. Select Optimize from the OptdesX main menu.****34. Select the Options pushbutton in the Optimize window.**

Reference diagrams for these two windows are given on the next page.

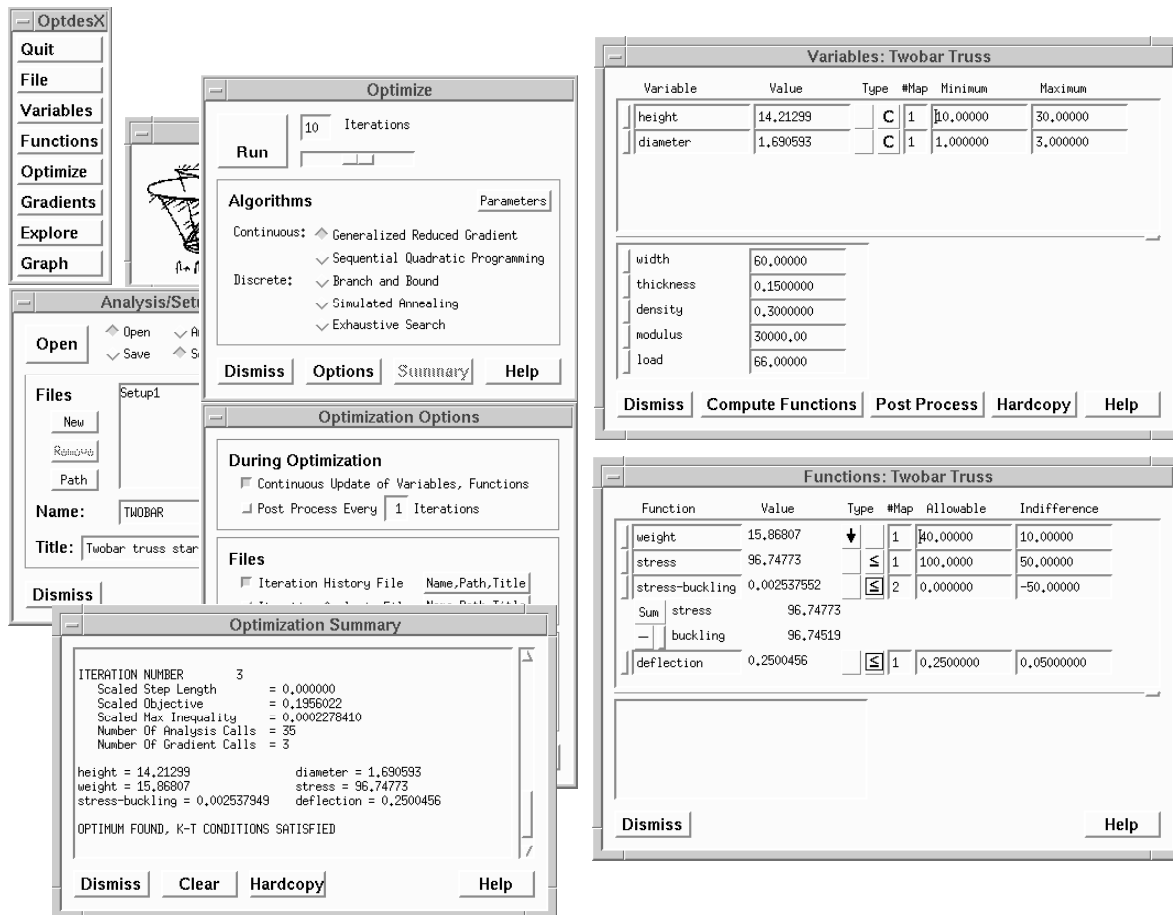


- A - Run pushbutton used to begin execution of an optimization algorithm. After execution is started, the button label changes to “Stop,” and pushing it will stop execution. In general, execution will continue until an optimum is reached or until the specified number of iterations/combinations/cycles has been completed.
- B - Optimization Algorithm radio box.
- C - Iteration value field used to set the number of optimization iterations. This number can be changed with the cursor or the scale in D.
- D - Iteration scale.
- E - Parameters pushbutton used to display and set algorithm parameters. Each algorithm has associated with it tolerance and/or convergence parameters. In this tutorial we will use the default parameters.

- F - Options pushbutton used to display Optimization options.
- G - Summary pushbutton which opens (or if open, brings forward) the Summary window, which shows optimization information. The button is dimmed because we have not optimized yet.
- H - During Optimization box used to specify whether variables and functions will be continuously updated and/or whether ANAPOS should be called during the optimization.
- I - Files box used to specify what files will be created during an optimization. The default is to generate a History file (with the default name “History”), which can then be plotted. Intermediate designs created during the optimization can also be stored.
- J - Summary Window box. When an optimization is started, a summary window is opened to inform you of the progress made. Additional information beyond the default can be printed to the summary window by selecting the appropriate toggle buttons.

35. Select the Print Variables, Functions toggle button in the Optimization Options--Summary Window box.

36. Select the Run pushbutton in the Optimize window.



Your screen should look similar to the screen above. The Optimization Summary window

prints scaled step length, scaled objective, scaled max inequality, and the number of analysis and gradient calls for each iteration of the optimization. Variable and function values are also printed because that option was selected in the Options window. The stopping message, “Optimum Found, K-T Conditions Satisfied,” indicates that the mathematical conditions for a local optimum have been satisfied. It is the strongest stopping message OptdesX will report. As the optimization proceeds, the values in the Variables and Functions windows are changed--the final values are the optimal values. The boxed constraint icon indicates that the stress-buckling and deflection constraints are binding. We can see, however, that these two constraints are slightly violated. This is because there is a constraint tolerance that is used to determine if a constraint is binding--as long as the function value is within plus or minus this tolerance of the allowable value, the constraint is considered binding and satisfied. In GRG the default constraint tolerance is 0.001 in scaled space, or about one part in one thousand. Weight has decreased from approximately 36 pounds to 15.8 pounds. Height and diameter decreased from 30 and 3 inches to 14.2 and 1.7 inches respectively. These changes can be plotted using the History file that was written during the optimization.

Saving the Optimum Design

Once an optimum has been found, you may wish to save the optimum design for future reference.

37. Set the Analysis/Setup Files window to “Save, Analysis.”

38. Use the default File name (“Analysis2”). Type the title, “Twobar truss GRG optimum.”

39. Press the Save pushbutton.

Optimizing with Sequential Quadratic Programming

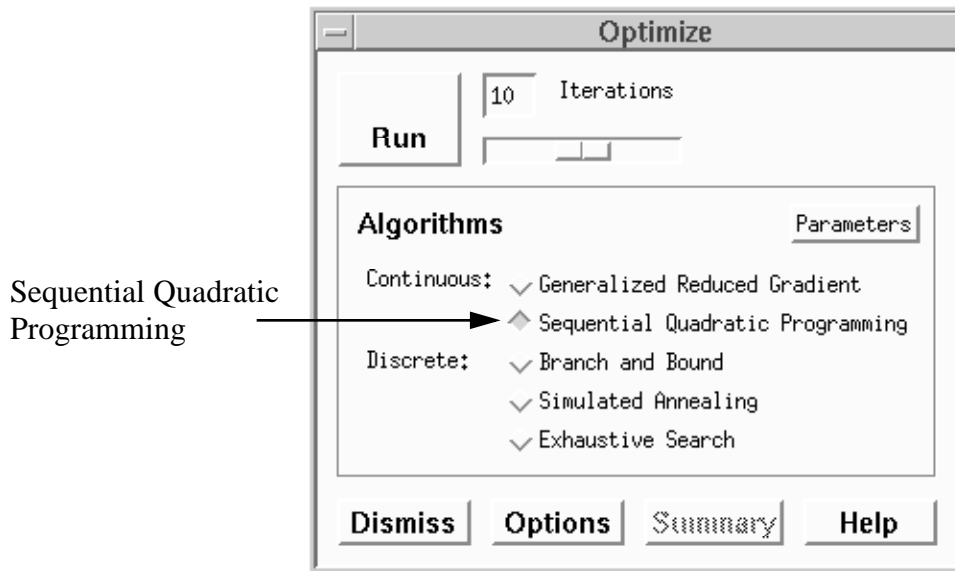
We will now try a different optimization algorithm, Sequential Quadratic Programming. In order to start this algorithm from the same starting point as GRG (for comparison), we will restore the Analysis1 file with the initial analysis variable values.

40. Set the Analysis/Setup Files window to “Open, Analysis.”

41. Open the Analysis1 file by clicking on it twice or clicking once and pressing the Open button.

The initial analysis variable and function values are displayed in the variable and function windows.

42. Select Sequential Quadratic Programming in the Optimize window.



43. Press the Run pushbutton in the Optimize window.
 The variable and function values at the end of the optimization are shown.

Variable	Value	Type	#Map	Minimum	Maximum
height	14,21345	C	1	10,00000	30,00000
diameter	1,690558	C	1	1,000000	3,000000

width	60,00000
thickness	0,1500000
density	0,3000000
modulus	30000,00
load	66,00000

Function	Value	Type	#Map	Allowable	Indifference
weight	15.86783	↓	1	40.00000	10.00000
stress	96.74722	≤	1	100.00000	50.00000
stress-buckling	0.007143672	≤	2	0.000000	-50.00000
Sum stress	96.74722				
- buckling	96.74007				
deflection	0.2500392	≤	1	0.2500000	0.05000000

Buttons: Dismiss, Help

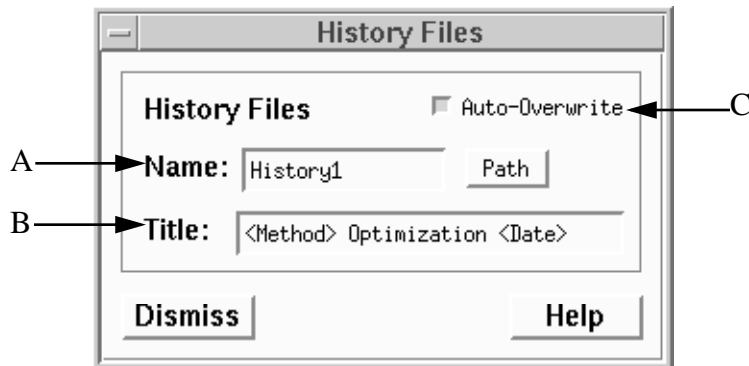
The optimum is the same as with the GRG algorithm.

History Files and Plots

A History file is a file used to store data generated during optimization. This data can be plotted to show how things changed during the optimization. In this section you will learn how History files are generated and how to graphically display history information.

44. Select the History File Name, Path, Title pushbutton in the Optimization Options window. (The Optimization Options window is opened by pressing the Options pushbutton in the Optimization window.)

This will show you the options which can be set for History files.



A - Name text field. If Auto-Overwrite is on (the default), a file named “History1” is created during an optimization and overwrites any previous History files of that

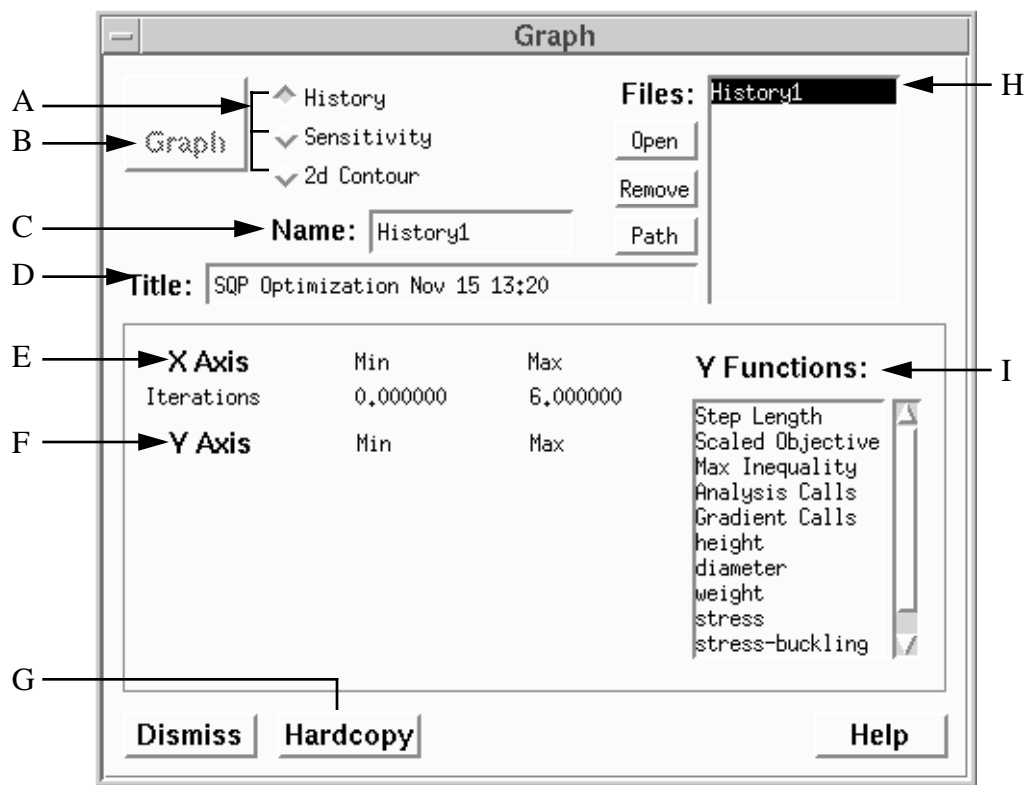
name. If Auto-Overwrite is off, a history file is created and saved for each optimization. The file names are numbered sequentially, i.e. History1, History2, etc. The History file name is limited to 16 characters.

- B - Title text field. The default title includes the optimization method and the date and time. The title is limited to 60 characters, although only 30 are displayed here.
C -Auto-Overwrite toggle button.

45. Select the Dismiss pushbutton in the History Files window.

46. Select Graph from the OptdesX main menu.

An Analysis or Setup file does not have to be opened to access the Graph window. The Graph window is used to plot previously generated files such as History files.



A - File type radio box. Choices are: History (one variable changing), Sensitivity (one variable changing), and 2d Contour (two variables changing). Sensitivity and 2d Contour files are created using the Explore window.

B - Graph pushbutton used to open a graphics window and plot the data from the selected file.

C - Name text field indicating the name of the file selected in the Files list.

D - Title text field displaying the title of the file selected and opened in the Files list.

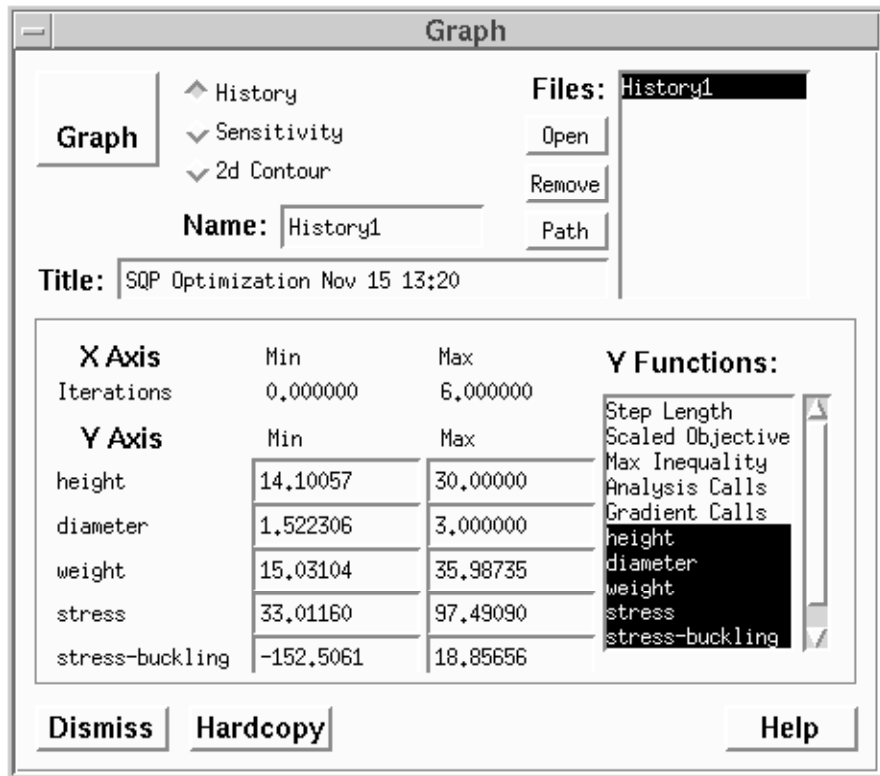
E - X Axis information. This is read from the file.

F - Y Axis information. For a sensitivity or history plot the Y axes are determined by the functions selected in the Y Functions list. Up to five functions can be plotted.

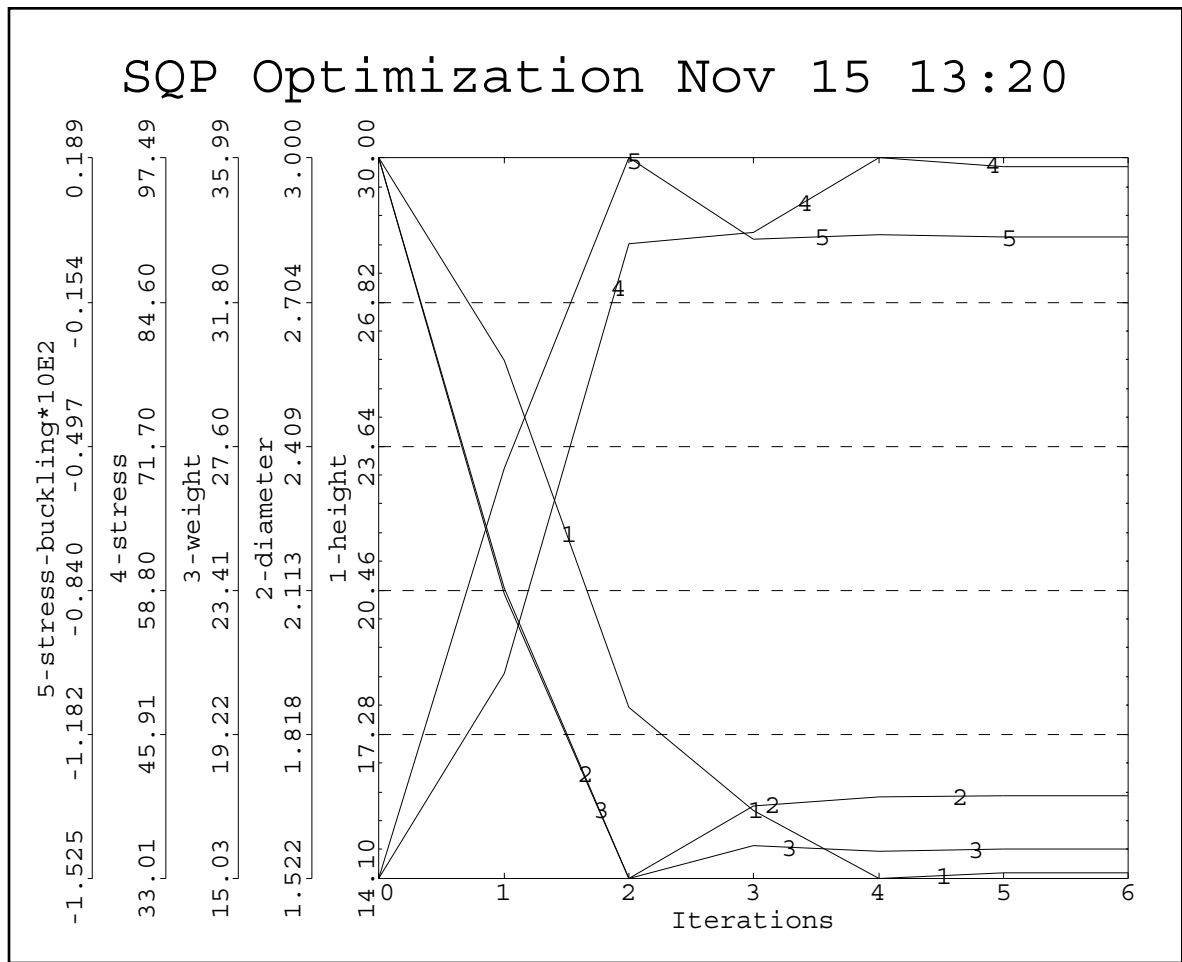
G - Hardcopy pushbutton used to open the Hardcopy window and obtain a hardcopy of the plot.

- H - Files scrollable list displaying the appropriate data files in the current directory for the option selected (History, Sensitivity or 2d Contour).
- I - Y Functions scrollable list showing functions which can be plotted along the Y Axis.

47. Set the File type radio box to “History.”
48. Select and open the History file created during the optimization process.
49. Select height, diameter, weight, stress, and stress-buckling from the Y Functions list. You can select up to five functions to be plotted at a time.



50. Press the Graph pushbutton in the Graph Window.
51. Enlarge the graph by placing the cursor in any corner and dragging.



This plot shows how the selected variables and functions changed during the optimization. Both height and diameter started at their upper bounds and decreased. Weight and stress are conflicting functions: as one decreased the other increased.

52. Press the Dismiss pushbutton in the Optdes Plot and Graph windows.

Modifying an Optimization Problem

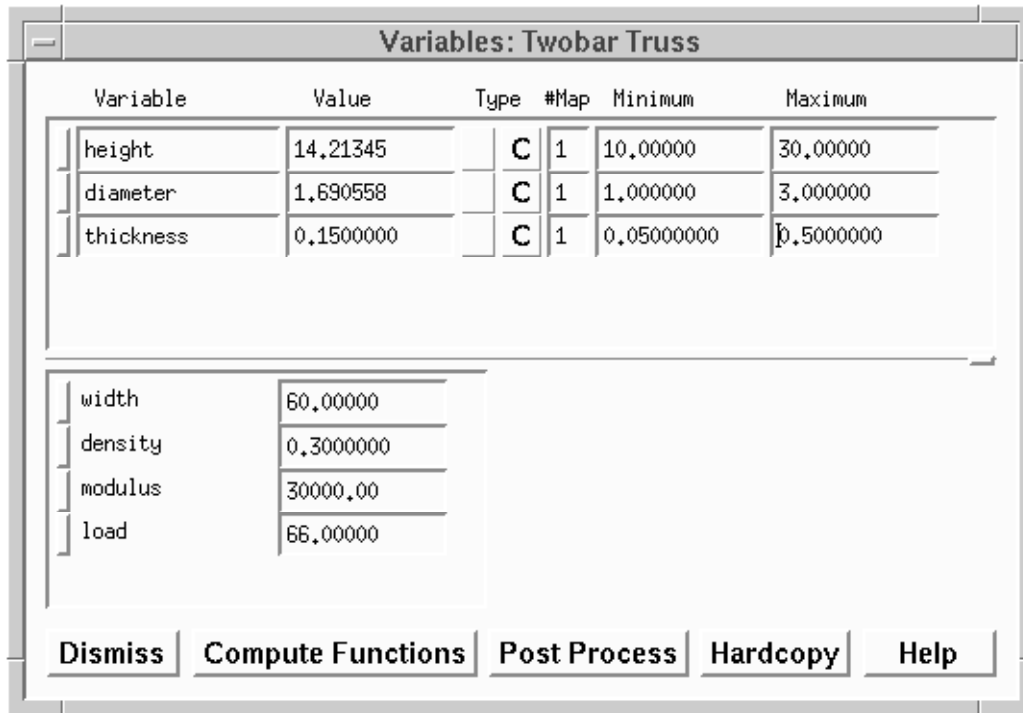
At this point we will illustrate how you can easily change the optimization problem definition. In this section you will modify the two-bar truss problem setup by:

- Adding a design variable
- Using multiple objectives
- Changing objective weighting coefficients.

Adding a Variable

The optimization problem definition will be modified by adding thickness as a design variable; up to this point thickness has been held constant at 0.15 inches.

53. Add thickness to the design variable list by pressing its mapping button.



54. Set minimum and maximum values to those shown in the window above.

55. Select the Generalized Reduced Gradient algorithm in the Optimize window.

56. Press the Run pushbutton in the Optimize window.

Values at the optimum are shown in the following windows.

Variables: Twobar Truss

Variable	Value	Type	#Map	Minimum	Maximum
height	30.00000	▲ C	1	10.00000	30.00000
diameter	2.204429	C	1	1.000000	3.000000
thickness	0.06740422	C	1	0.05000000	0.5000000

width	60.00000
density	0.3000000
modulus	30000.00
load	66.00000

Functions: Twobar Truss

Function	Value	Type	#Map	Allowable	Indifference
weight	11.88285	▼	1	40.00000	10.00000
stress	99.97601	⏏	1	100.0000	50.00000
stress-buckling	-0.03705780	⏏	2	0.000000	-50.00000
Sum stress	99.97601				
- buckling	100.0131				
deflection	0.1999520	⏏	1	0.2500000	0.05000000

As expected, adding thickness as a design variable improved the optimum. When thickness was held constant at 0.15 inches, the optimum weight was 15.9 lbs; now it is 11.9 lbs. Stress and stress-buckling are binding constraints; previously stress-buckling and deflection were binding.

Adding Another Objective

The optimization problem will now be modified by adding stress as an objective. Stress has been used as a constraint thus far in the tutorial. Weight and stress are competing objectives: in order for one to decrease, the other must increase and vice versa. OptdesX makes it easy to explore trade-offs such as this using point and click operations.

57. Add Stress as an objective by pressing its Objective button.

Stress is now an objective and a constraint. We will expect stress and weight to trade-off against each other.

Function	Value	Type	#Map	Allowable	Indifference
weight	11.88285	↓	1	40.00000	10.00000
stress	99.97601	↓	1	100.0000	50.00000
stress-buckling	-0.03705780	↖	2	0.000000	-50.00000
Sum stress	99.97601				
buckling	100.0131				
deflection	0.1999520	↖	1	0.2500000	0.05000000

weight	1.0	<input type="text"/>
stress	1.0	<input type="text"/>

Dismiss Help

Slider bars used to change the weighting coefficients for multiple objectives.

Slider bars are displayed for each objective in a multiple objective problem. The slider bar is used to set the objective weighting coefficient and thereby weight or rank the importance of the objectives. The default values of 1.0 mean that each objective is equally weighted or ranked.

58. Press the Run pushbutton in the Optimize window.

Values in the windows will change to those shown below.

Variables: Twobar Truss						
Variable	Value	Type	#Map	Minimum	Maximum	
height	30.00000	▲ C	1	10.00000	30.00000	
diameter	2.218758	C	1	1.000000	3.000000	
thickness	0.1043102	C	1	0.05000000	0.5000000	
width	60.00000					
density	0.3000000					
modulus	30000.00					
load	66.00000					

Dismiss Compute Functions Post Process Hardcopy Help

Functions: Twobar Truss						
Function	Value	Type	#Map	Allowable	Indifference	
weight	18.50863	↓	1	40.00000	10.00000	
stress	64.18629	↓ W	1	100.0000	50.00000	
stress-buckling	-37.26021	W	2	0.000000	-50.00000	
Sum	stress					64.18629
-	buckling					101.4465
deflection	0.1283726	W	1	0.2500000	0.05000000	
				weight	1.0	<input type="text"/>
				stress	1.0	<input type="text"/>

Dismiss Help

Note the compromise made between stress and weight. Weight increased from 11.9 to 18.5 lbs; stress started at 99.98 ksi and decreased to 64.18 ksi. At this new optimum there are no binding constraints, although height is at its upper bound.

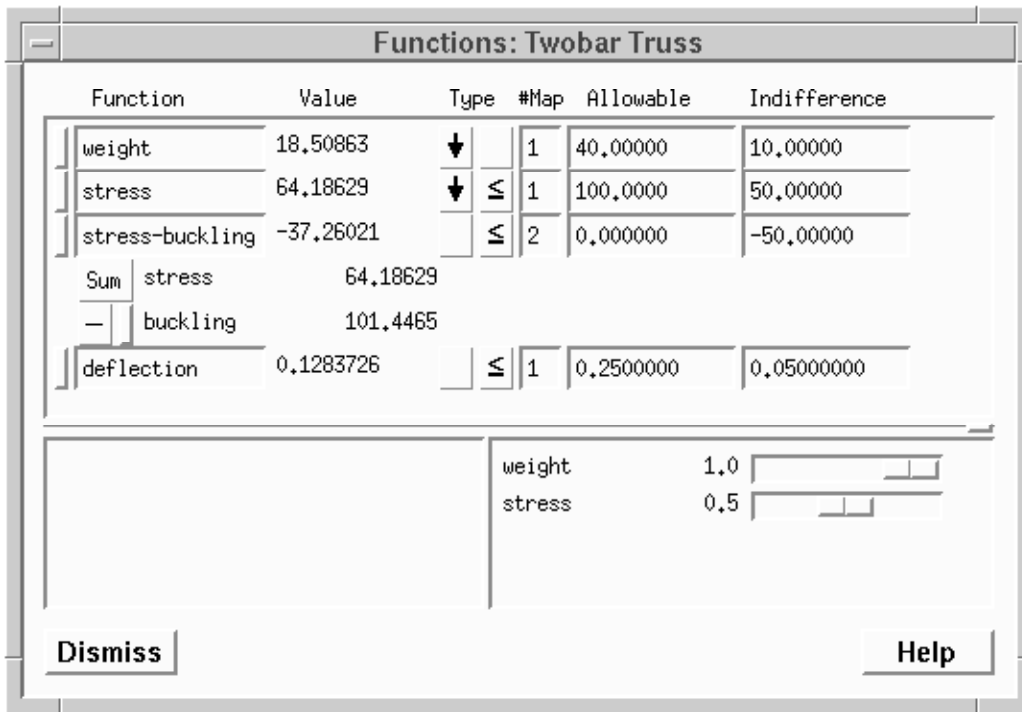
Changing an Objective Weighting Coefficient

In the previous optimization problem weight and stress were equally weighted or ranked. We

will further experiment with the trade-off of these functions by reducing the weighting coefficient of stress using the slider bar. By decreasing the importance of stress we expect that at the new optimum stress will increase and weight will decrease.

59. Change the weighting coefficient of stress to 0.5.

With a weighting coefficient of 1.0 for weight and 0.5 for stress, the function weight is given twice as much emphasis as stress during the optimization.



Sometimes a slight change in the weighting coefficients is not enough to cause the algorithms to move when they are at an optimum. If you find this to be the case, start the problem from a point some distance from the optimum.

60. Select the Run pushbutton in the Optimize window.

The optimal values are shown.

Variables: Twobar Truss						
Variable	Value	Type	#Map	Minimum	Maximum	
height	29.96350	C	1	10.00000	30.00000	
diameter	2.208888	C	1	1.000000	3.000000	
thickness	0.09365964	C	1	0.05000000	0.5000000	
width	60.00000					
density	0.3000000					
modulus	30000.00					
load	66.00000					

Dismiss Compute Functions Post Process Hardcopy Help

Functions: Twobar Truss						
Function	Value	Type	#Map	Allowable	Indifference	
weight	16.53481	↓	1	40.00000	10.00000	
stress	71.84847	↓	1	100.0000	50.00000	
stress-buckling	-28.77847	↔	2	0.000000	-50.00000	
Sum stress	71.84847					
- buckling	100.6269					
deflection	0.1436970	↔	1	0.2500000	0.05000000	
		weight	1.0	[Progress Bar]		
		stress	0.5	[Progress Bar]		

Dismiss Help

Weight has decreased to 16.53 lbs. from the previous optimum of 18.50 lbs. Also, as was expected, stress increased from 64.18 to 71.84 ksi. Again no constraints are binding.

Restoring a Setup File

At some point in the optimization process you may wish to use a previous optimization setup.

61. Set the Analysis/Setup File window to “Open, Setup.”

62. Open the Setup1 file by double clicking the file name, or clicking once and pressing the Open button.

When the file is opened, a “Restoring Setup” message is printed, and the optimization setup is restored from the file and displayed in the variable and function windows. We will now open an Analysis file previously saved. Remember that any Analysis file can be restored with any Setup file.

63. Set the Analysis/Setup File window to “Open, Analysis.”

64. Open the Analysis1 file by double clicking the file name, or clicking once and pressing the Open button.

Note that the initial analysis and function values are now displayed in the Variable and Function windows.

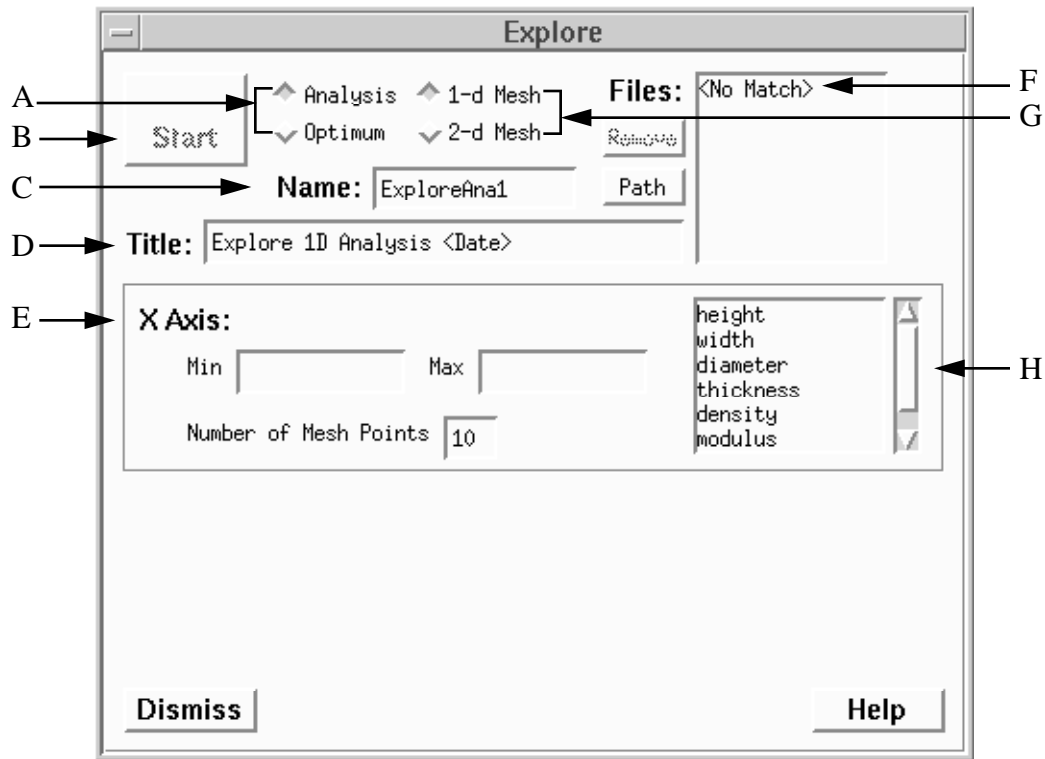
Exploring Design Space

Design space can be graphically explored by creating sensitivity plots (plots showing how functions(s) change with respect to one variable) or contour plots (plots showing how function(s) change with respect to two variables). The first step is to create an Explore file by overlaying design space with a grid and evaluating the analysis model at each grid point. This is done in the Explore window. Once an Explore file is created, it can be opened in the Graph Window and plotted. In this section of the tutorial you will learn how to generate:

- One Dimensional Explore Analysis files and Sensitivity Plots
- Two Dimensional Explore Analysis files and Contour Plots
- One Dimensional Explore Optimum files and Sensitivity Plots
- Two Dimensional Explore Optimum files and Contour Plots.

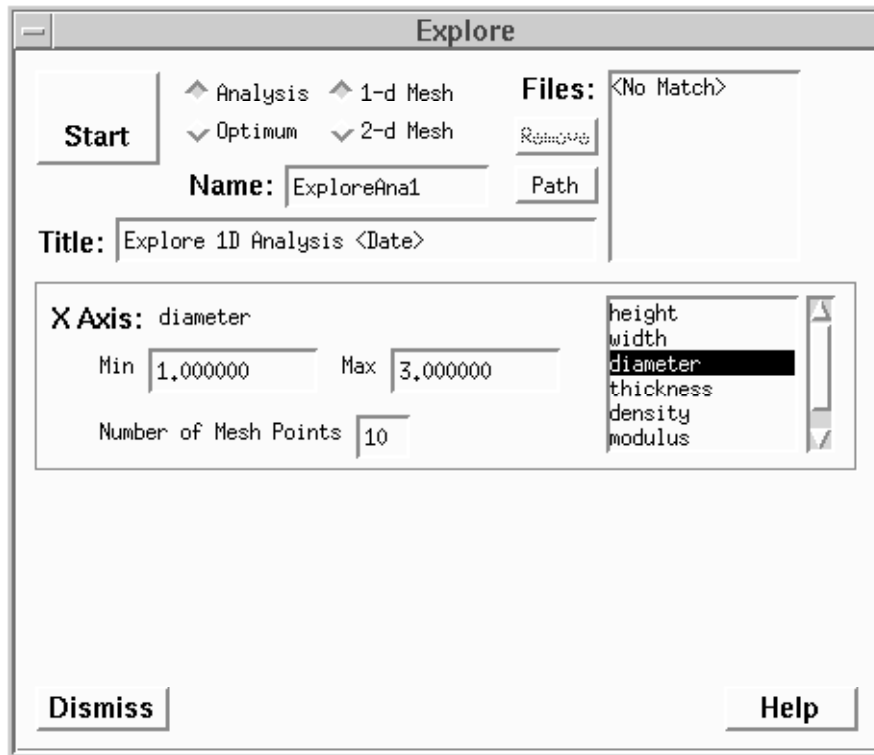
One Dimensional Explore Analysis files and Sensitivity Plots

65. Select Explore from the OptdesX main menu.



- A - Analysis-Optimum radio box used to specify whether the analysis model is to be evaluated or an optimization is to be performed at each grid point.
- B - Pushbutton that causes the design space to be meshed, performing an optimization or analysis at each grid point, according to the option selected.
- C - Explore File Name text field. The default name is “ExploreAna1.” Each successive default file name is incremented, i.e. “ExploreAna2,” “ExploreAna3,” etc. The file name is limited to 16 characters. An extension “.ex1” or “.ex2” is appended to the file name depending of the state of the 1-d, 2-d Mesh radio box.
- D - Explore File Title text field. The title is limited to 60 characters--40 are shown.
- E - X Axis Box used to specify the meshed variable for the X Axis. Default Min and Max values are taken from the design variable list, if the variable is a design variable, or they are set to be a percentage of the current value. These, of course, are editable. You should collapse the Min and Max values around a particular point if you wish to “zoom in” on the region surrounding the point. The Number of Mesh points determines the number of analyses (or optimizations) done. Mesh points are spread evenly between the Min and Max values.
- F - File scrollable list used to see the other Explore files already in the directory. The message shown here indicates that there are no other Explore files in the directory.
- G - 1-d, 2-d Mesh radio box. For a 1-d mesh, only one variable is changed; for a 2-d mesh, two variables are changed. When a 2-d mesh is selected, a Y Axis Box appears in the lower part of the window. The total number of grid points that will be evaluated is the product of the mesh points for the X and Y axes.
- H - Mesh Variable scrollable list showing variables that can be selected as mesh variables.

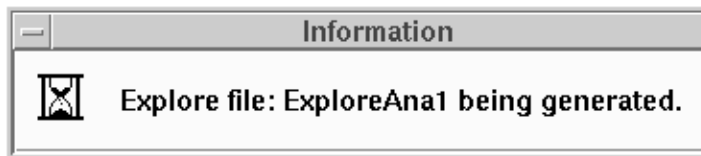
66. Select diameter in the Mesh Variable scrollable list in the X Axis box.



The Minimum and Maximum values in the Variables window are used as the default minimum and maximum values in Explore. You can change these values. We will use the default name (“ExploreAna1”) and title (“Explore 1D Analysis <Date>”). The keyword <Date> will be replaced by the date and time the Explore file is made.

67. Press the Start pushbutton in the Explore window.

An information box will be displayed letting you know that the Explore file is being generated.



68. Select Graph from the OptdesX main menu.

Refer to the section on History Files and Plots for a detailed description of this window.

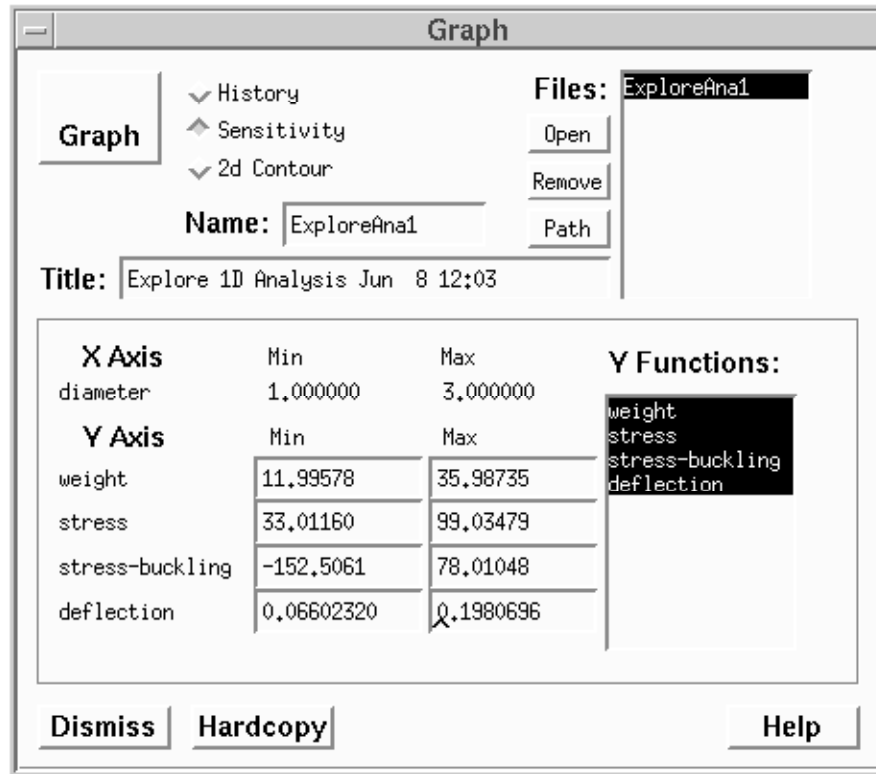
69. Set the File type radio box in the Graph window to “Sensitivity.”

70. Select the 1-d file just generated (ExploreAna1) in the Files list and open it.

Files can be opened by pressing the open button or by double clicking the file name.

71. Select the following functions to graph from the Y Functions scrollable list: weight, stress, stress-buckling, and deflection.

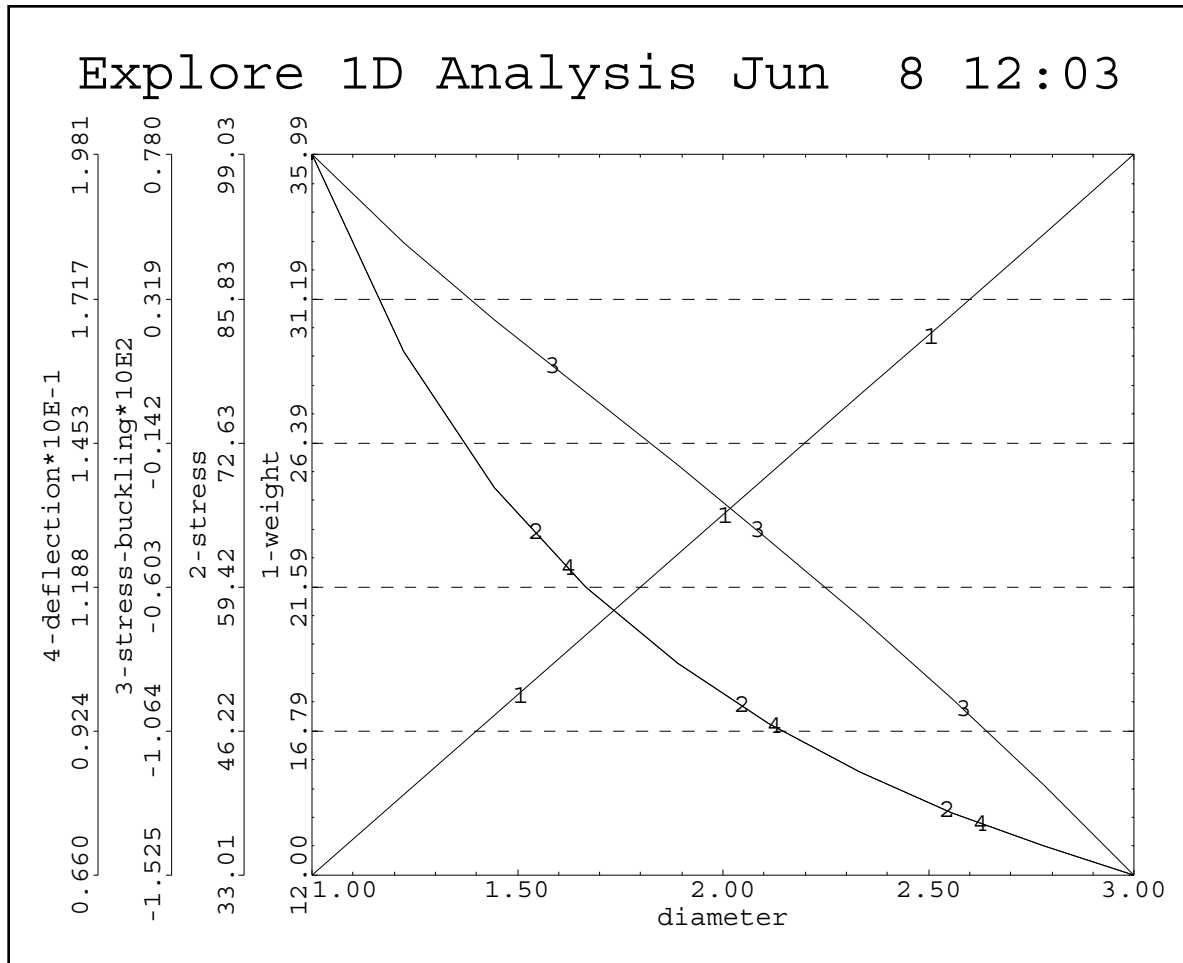
For a sensitivity plot up to five functions can be plotted at a time.



These functions will be plotted on the Y axis. Each function will be plotted separately to show how it changes with respect to the X axis variable (diameter). The minimum and maximum values of Y axes can be edited.

72. Press the Graph pushbutton in the Graph window

Enlarge the graph by placing the cursor in any corner, clicking and dragging.



This plot shows how each of the functions varies with diameter. The functions along the Y axis are numbered and correspond to the curves in the plot. As diameter increases, weight increases and stress, stress-buckling, and deflection decrease.

73. Press the Dismiss button for the graph.

Two Dimensional Explore Analysis file and Contour Plot

A sensitivity plot shows how functions change with respect to one variable. A contour plot shows how functions change with respect to two variables. We will now generate the data for and graph a contour plot.

74. Activate the Explore window.

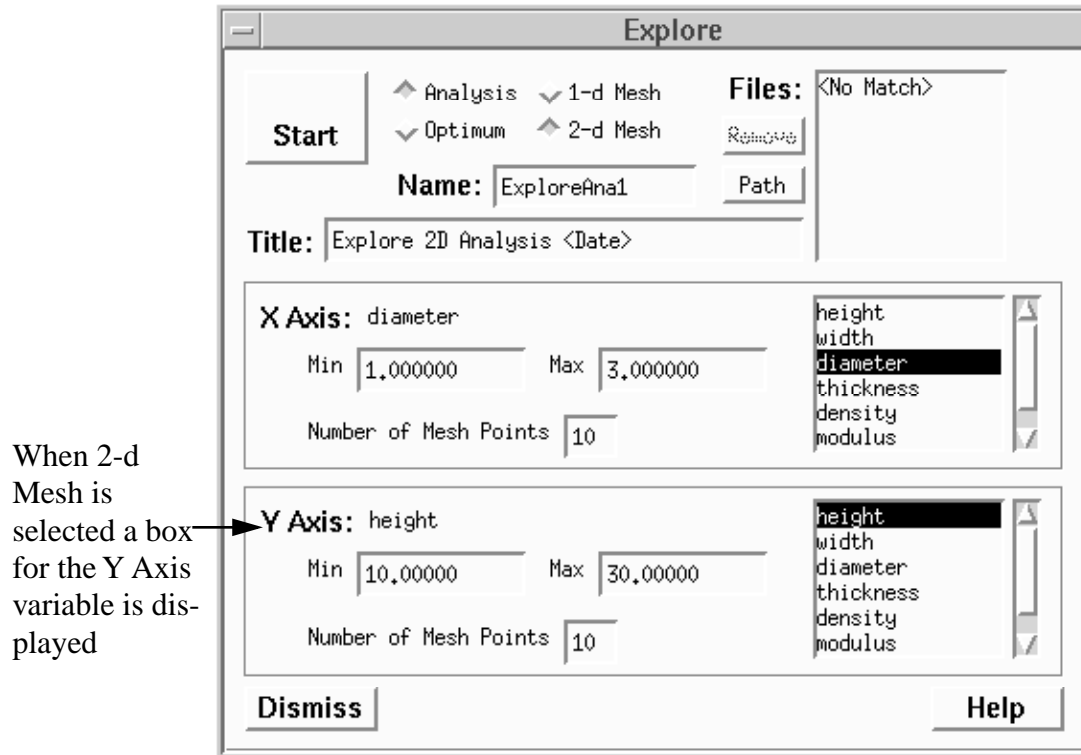
Windows can be activated by clicking anywhere inside the window or selecting the window (Explore in this case) from the Main Window Selection box.

75. Select “2-d Mesh” in the 1-d, 2-d Mesh radio box in the Explore window.

A contour plot is interpolated from a two dimensional mesh in design space. The total number

of analyses done is equal to the product of the number of mesh points for the X and Y axes.

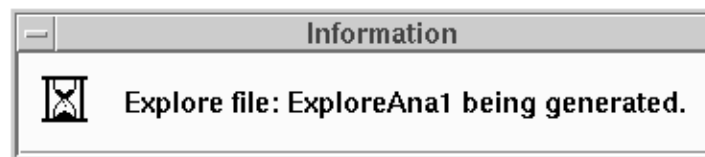
76. Select diameter from the scrollable list in the X Axis box and height from the scrollable list in the Y Axis box.



We will use the default name (“ExploreAna1”) and title (“Explore 2D Analysis <Date>”). Since this is a 2-d Explore file, it will not overwrite the 1-d Explore we previously created with the same name.

77. Select the Start pushbutton from the Explore window.

An information box will be displayed letting you know that the Explore file is being generated.

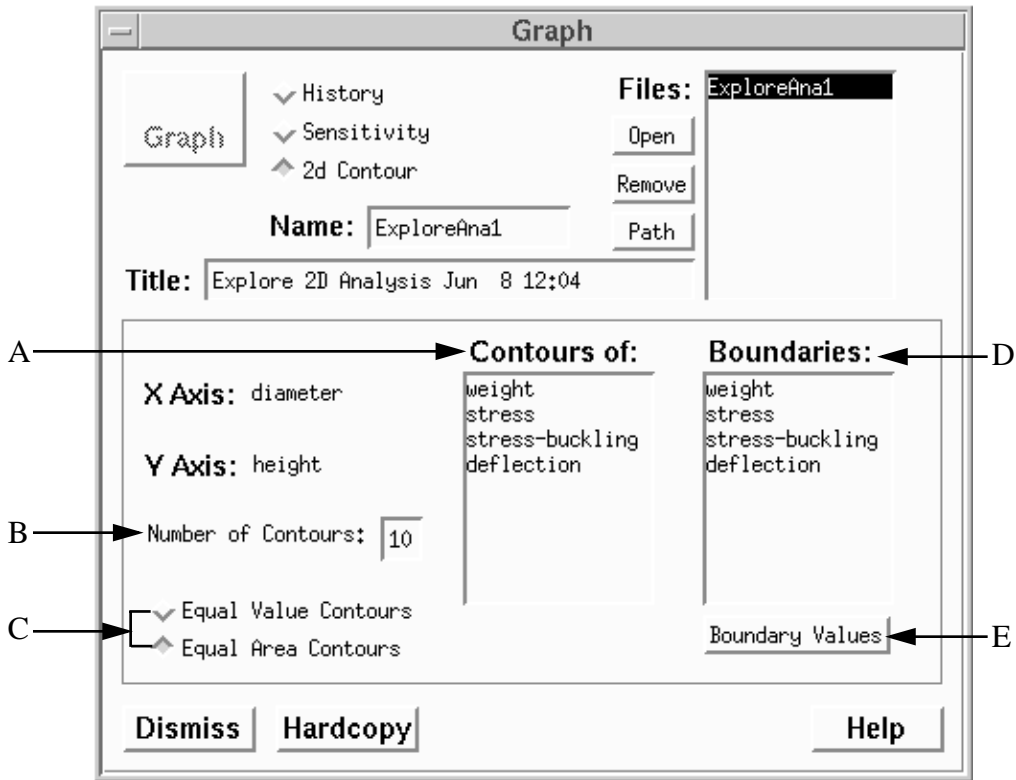


78. Activate the Graph window.

79. Select “2-d Contour” in the radio box in the Graph window.

80. Select the 2-d Explore file just generated (ExploreAna1) and open it.

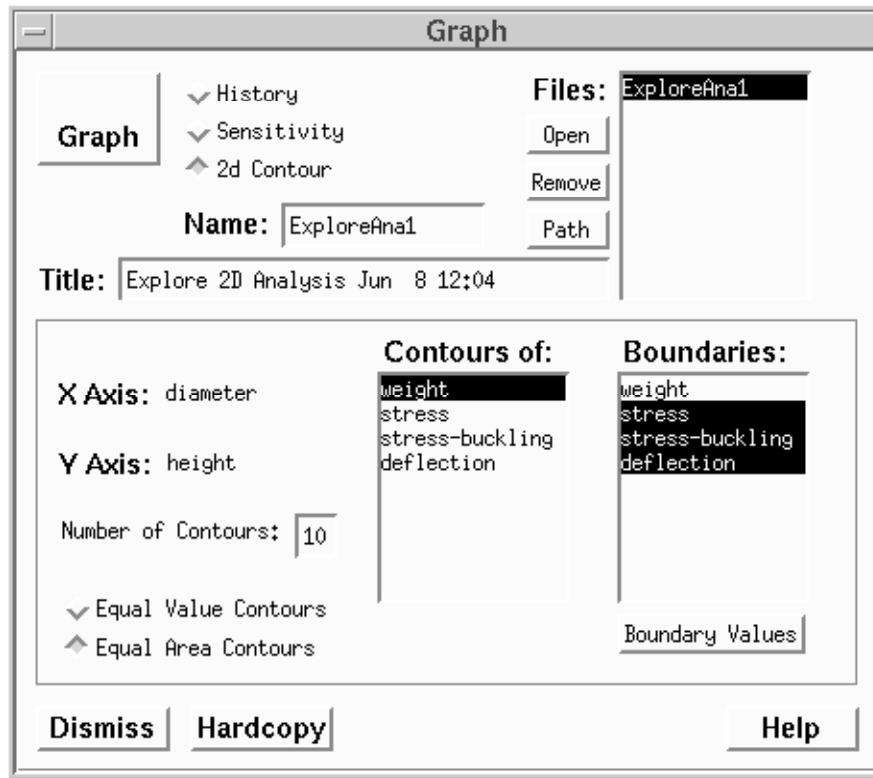
A reference diagram for Graph when plotting 2d Contour files follows.



- A - Contours scrollable list of design functions. One of these functions can be selected to have its contours drawn on the plot.
- B - Number of Contours value field.
- C - Contour Type radio box used to select the spacing of the contours. Equal Value Contours sets the spacing to be numerically equal intervals of the function being plotted. Equal Area Contours sets the physical spacing of the contours to be approximately equal across the graph.
- D - Boundaries scrollable list. As many functions as desired can be selected. Typically the constraints in the problem are selected.
- E - Boundary Value pushbutton used to open the Boundary Values window. This allows you to change the boundary values for each function. Typically the allowable values for the constraints are made the boundary values.

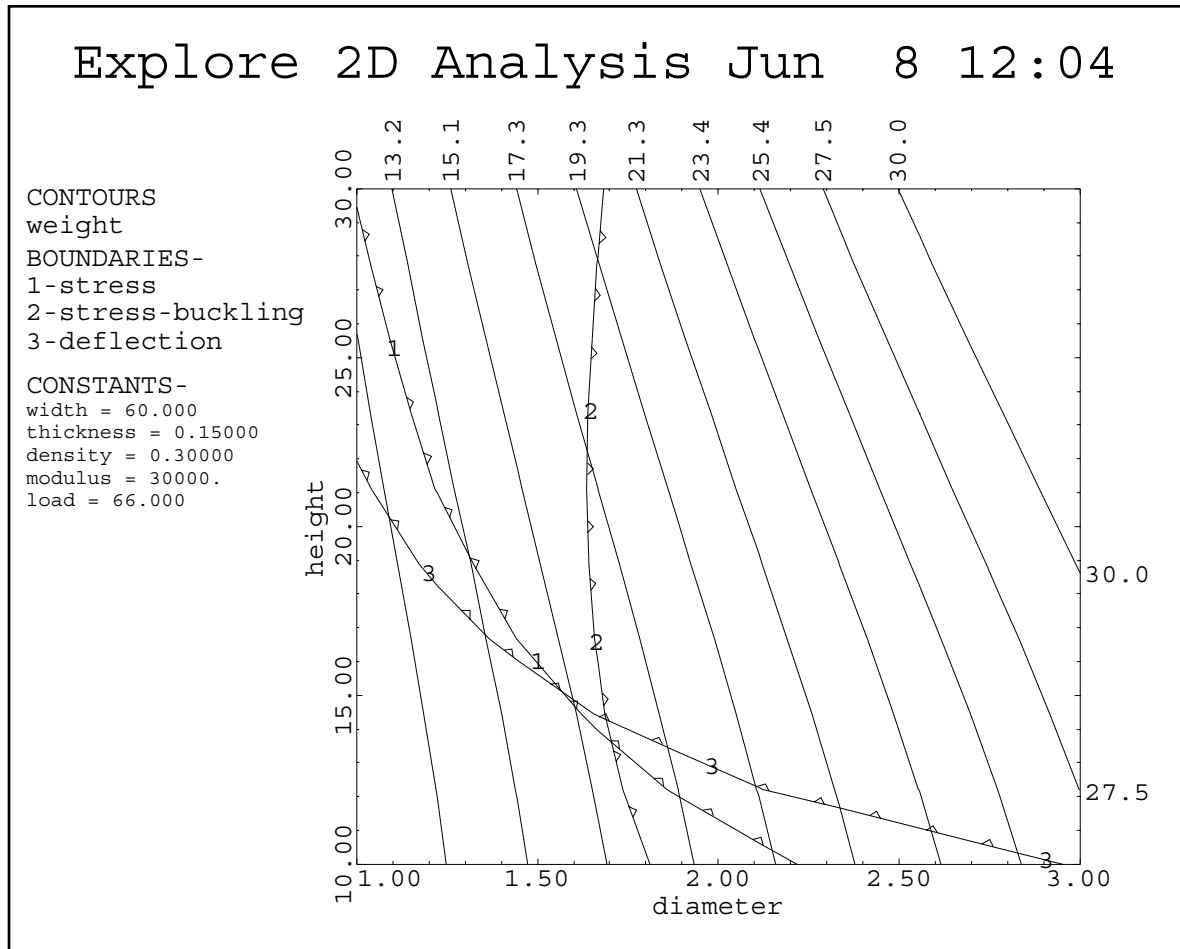
81. Select weight from the Contours scrollable list and the remaining functions from the Boundaries scrollable list.

Usually you will want to graph contours of the objective function and see boundaries of the constraint functions. On any particular plot only one function can be contoured, but as many boundaries as are listed can be selected. Thus the Contours list requires a single selection, but the Boundaries list allows multiple selections.



82. Press the Graph pushbutton from the Graph window.

You should enlarge this graph on your screen by placing the cursor in any corner and dragging.



The solid lines show contours of weight. The lines with arrows are boundaries of constraints. The arrows point to the feasible side of design space. In this case the optimum weight occurs at the intersection of constraint 2 and constraint 3, where weight has a value of 15.8 pounds.

83. Press the Dismiss button for the graph.

One Dimensional Explore Optimum and Sensitivity Plot

The previous two explore plots showed how the analysis functions change as we change one or two variables. At each grid point of the explore mesh, we called the analysis routine.

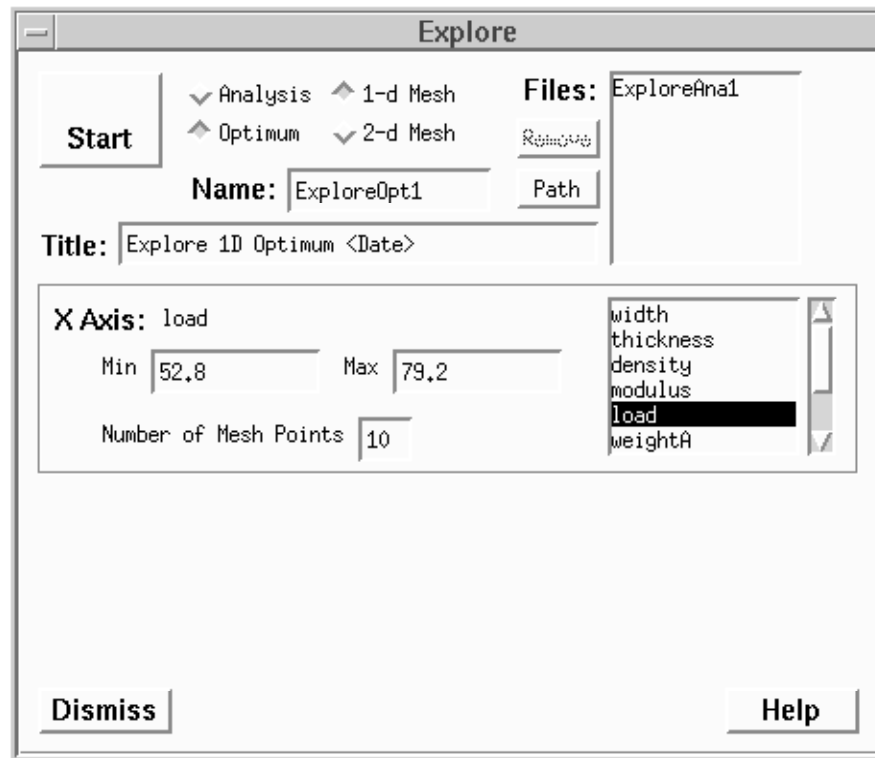
Now we will go one step further--we will examine how the *optimal weight* changes as we change values that are held constant during an optimization, such as unmapped analysis variables and constraint allowable values. At each grid point of the explore mesh, we will perform an optimization.

84. Activate the Explore window.

85. Set the radio boxes in the Explore window to "1-d, Optimum."

86. Select load from the scrollable list in the X Axis box as the variable to be explored.

We will examine how changing load changes the optimal value of weight.



We will use the default file name (“ExploreOpt1”) and title (“Explore 1d Optimum <Date>”).

87. In the Optimize window, set the number of iterations to be 100.

We need to make sure the algorithm does not terminate because it hits the iteration limit. A rule of thumb is that the number of iterations should be at least 25 times the number of design variables.

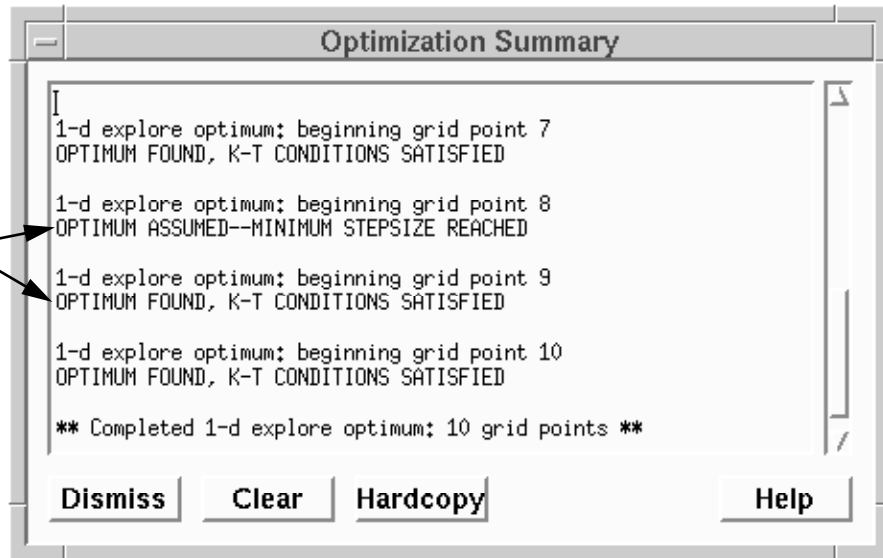
88. In the Optimization Options window, During Optimization box, turn off the continuous update of variables and functions. (The Optimization Options window is opened by pressing the Options pushbutton in the Optimization window.)

This will make the Explore Optimum run significantly faster.

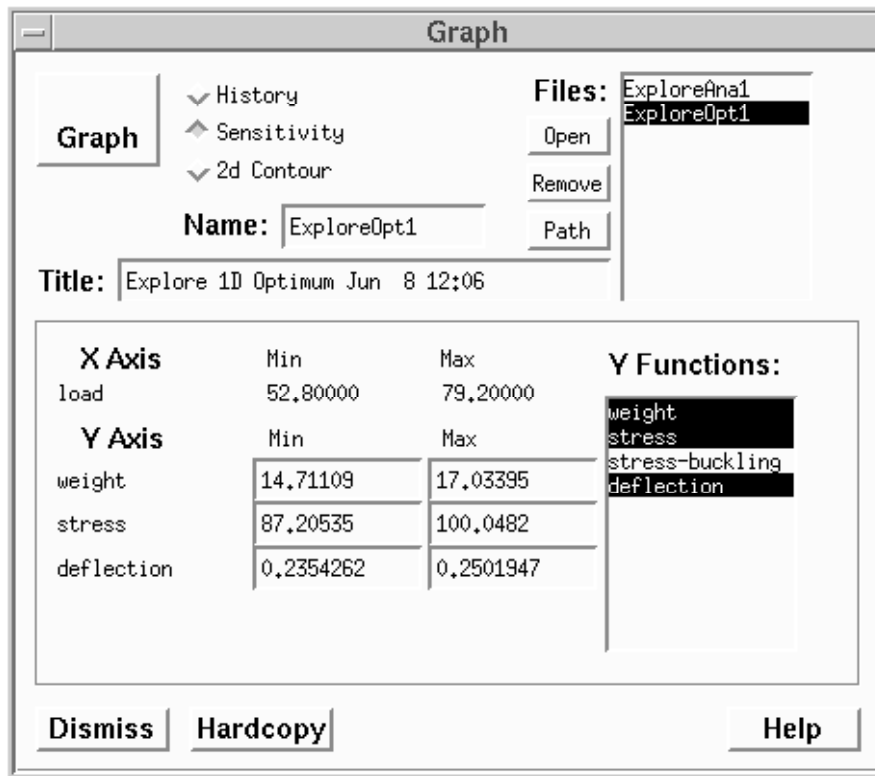
89. Press the Start pushbutton from the Explore window.

An optimization will be done at each of the ten mesh points. The Optimization Summary window will open and display brief information concerning each optimization. Doing an Explore Optimum has its risks! First of all, it obviously will take much more computer time because an optimization is done at each mesh point. Secondly, the software does not discriminate between optimizations that end successfully and those that do not. For example, if a feasible point cannot be found, the design point at the end of the optimization (which is infeasible) is entered in the Explore file. You should select ranges carefully, making sure they are not too extreme, before doing an Explore Optimum. Also, check the Summary window carefully to make sure each mesh point is valid.

After an Explore Optimum you should check to make sure each optimization terminated successfully



90. Activate the Graph Window.



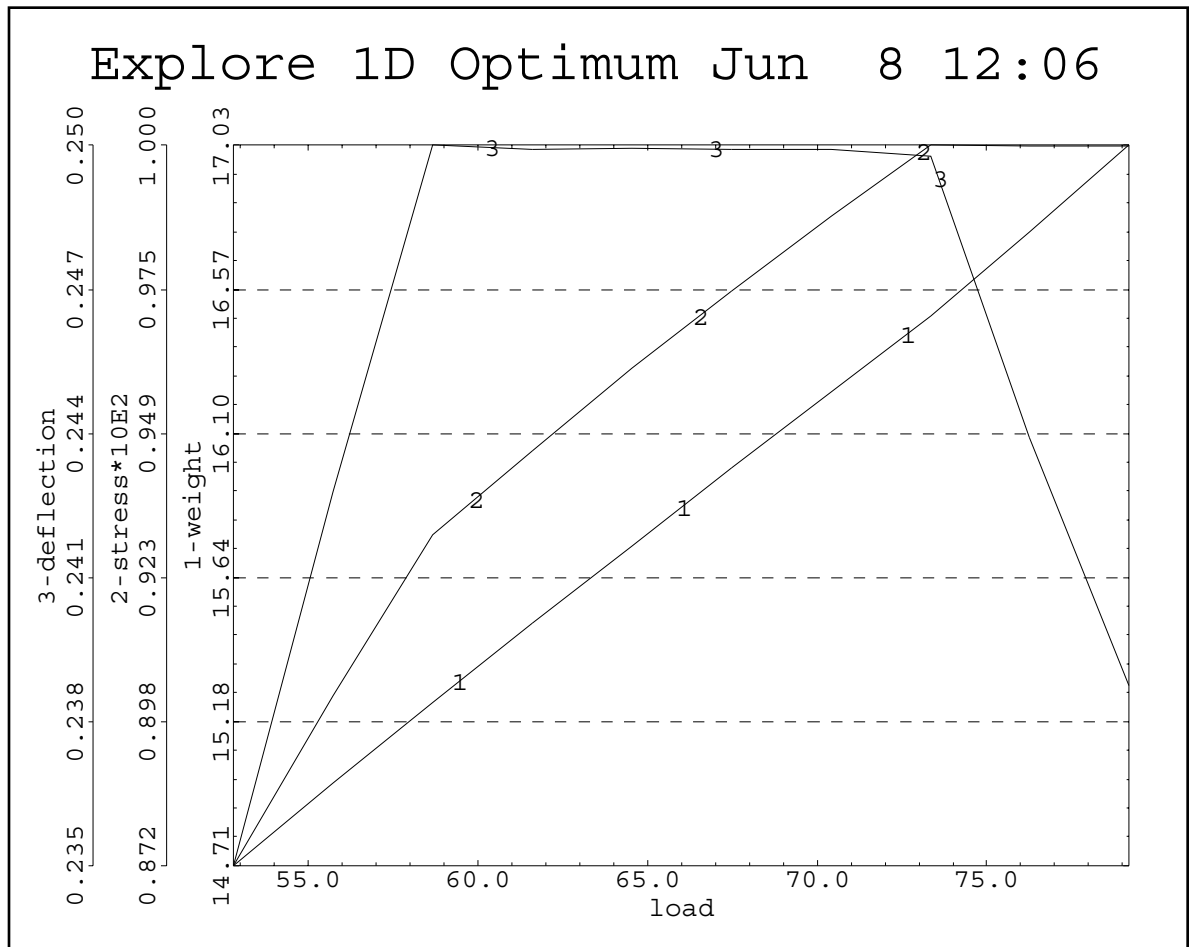
91. Select “Sensitivity” in the File type radio box in the Graph window.

92. Select the 1-d Explore Optimum file just generated (ExploreOpt1) and open it.

93. Select weight, stress, and deflection from the scrollable list of Y functions.

Usually it does not make sense to plot constraints that are always binding in a 1-d explore optimum plot. This is because they only range within plus or minus the constraint tolerance of the allowable. For this case, stress-buckling was binding at each optimum, so we won't plot it here. (How did we know stress-buckling was always binding? You can ascertain this by examining the min and max values of the function when it is selected. If the difference is less than one part in one thousand, then the change is smaller than the constraint tolerance. After observing this, you can de-select the function by clicking on it in the functions list. After obtaining the graph below, you may want to graph stress-buckling as an experiment.)

94. Select the Graph pushbutton from the Graph window.



This plot shows that as we increase load, the optimal weight increases nearly linearly; stress increases until it is binding at a load of approximately 73 ksi, and deflection is binding for loads of about 58 to 74 ksi.

95. Dismiss the graph.

Two Dimensional Explore Optimum and Contour Plot

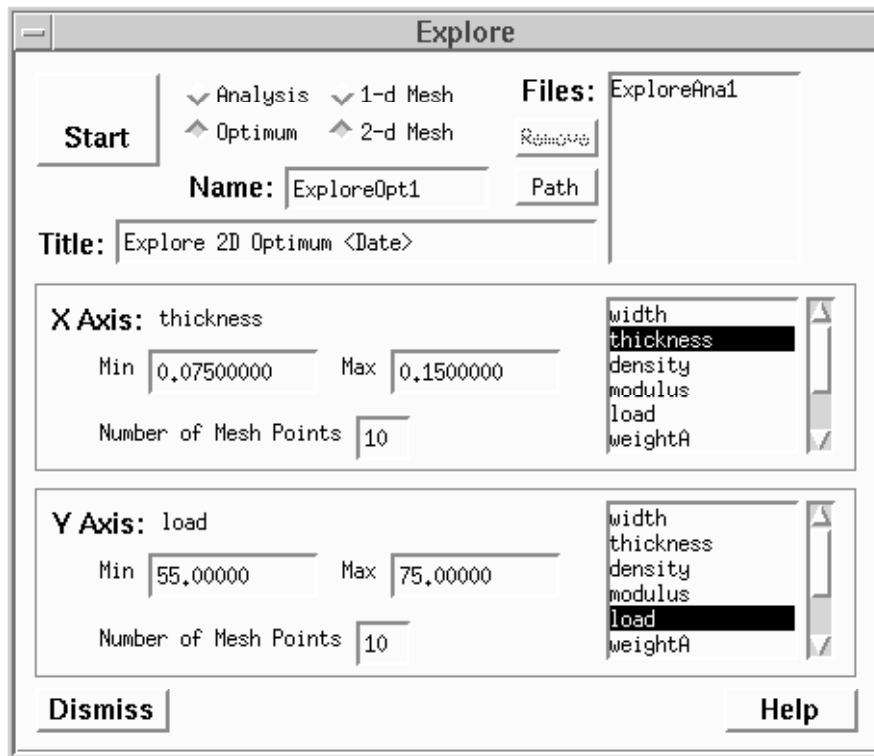
Now we will see how the optimum weight changes as we change two variables--load and thickness.

96. Activate the Explore window.

97. Select “2-d” in the 1-d, 2-d radio box in the Explore window.

98. Select thickness as the X-Axis variable and load as the Y-Axis variable to be explored. Set the ranges of thickness and load to be those shown.

We will use the default file name (“ExploreOpt1”) and title (“Explore 2D Optimum <Date>”).



99. Press the Start pushbutton in the Explore window.

An optimization will now be done for each of the mesh points in two dimensional design space. Since we have selected 10 mesh points for each variable, 100 optimizations will be done. The Optimization Summary window will give brief information about each optimization.

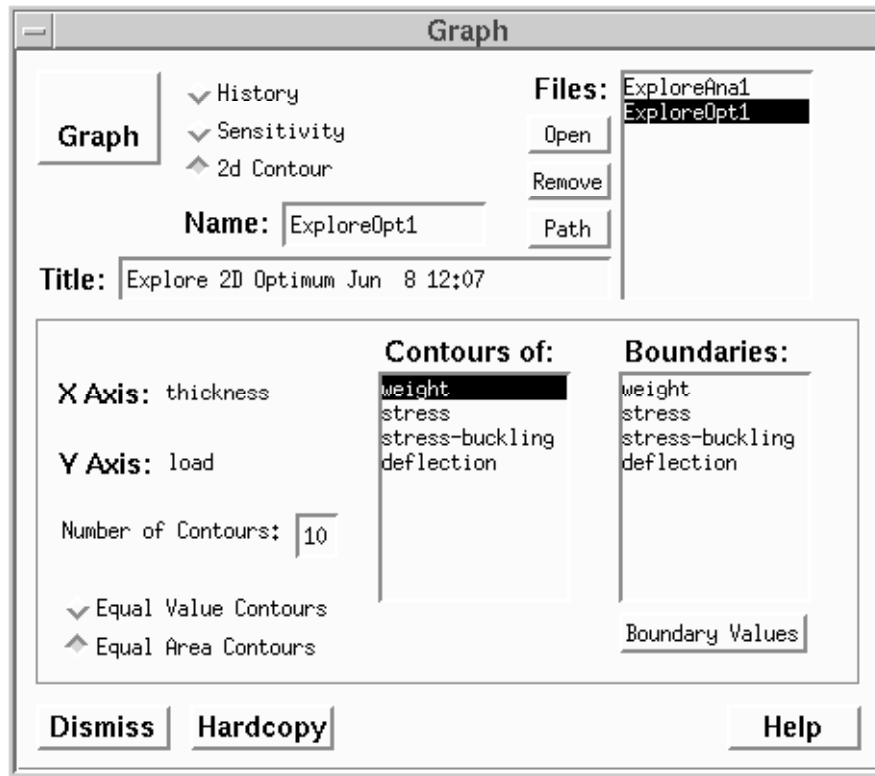
100. When the explore is finished, activate the Graph Window.

101. Select “2d Contour” in the File type radio box in the Graph window.

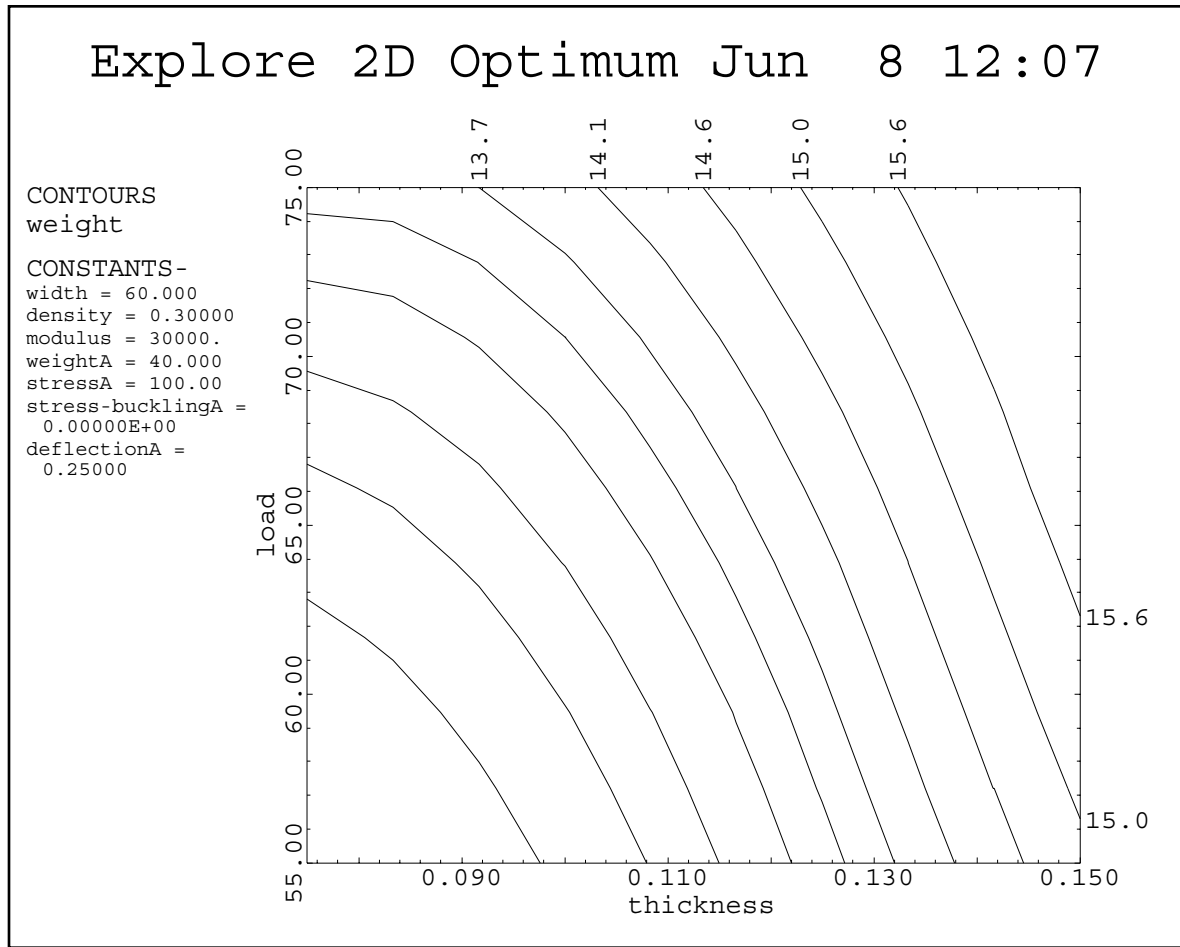
102. Select and open the file just created (ExploreOpt1).

103. Select weight as the function to be contoured.

Usually for a 2-d Explore Optimum it only makes sense to plot the objective. By definition each of the mesh points is feasible (at least they better be, if we want a useful plot!), so we can't show boundaries that demarcate feasible and infeasible space.



104. Press the Graph pushbutton



The solid lines show contours of the optimal weight. The graph shows how the optimal value of weight changes as we change load and/or thickness.

105. Press Quit in the main selection box to exit OptdesX.

Congratulations! You made it through the first OptdesX tutorial.

3.4 Tutorial 2: Optimizing with Discrete Variables

In this section we will define discrete variables for the twobar truss. After completing this tutorial you should be able to:

- Set up a discrete optimization problem
- Optimize using Branch and Bound, Simulated Annealing and Exhaustive Search algorithms

Discrete Optimization Problem Setup

Before beginning the tutorial, you must create two data files called “pipes” and “widths.” The contents of these two files are given below.

Contents of the file “pipes:”

```
19 2
3.0 .30
3.0 .28
3.0 .26
2.5 .28
2.5 .26
2.5 .24
2.5 .22
2.0 .24
2.0 .22
2.0 .20
2.0 .18
1.5 .20
1.5 .18
1.5 .16
1.5 .14
1.0 .16
1.0 .14
1.0 .12
1.0 .10
```

The file “pipes” contains the available values of diameter and thickness for pipes in the twobar truss. On the first row, “19” indicates that there are 19 different pipes, and thus 19 lines in the file after the first line. The “2” on the first row indicates that each pipe contains data for two variables, diameter and thickness. Values for diameter and thickness for each pipe appear on each of the lines following the first line. Data in these files must be separated by white space, i.e., blanks or tabs, but not commas.

Contents of the file “widths:”

```
11 1
70.
68.
66.
64.
62.
60.
58.
56.
54.
52.
50.
```

The file “widths” contains the 11 discrete values for width for the truss.

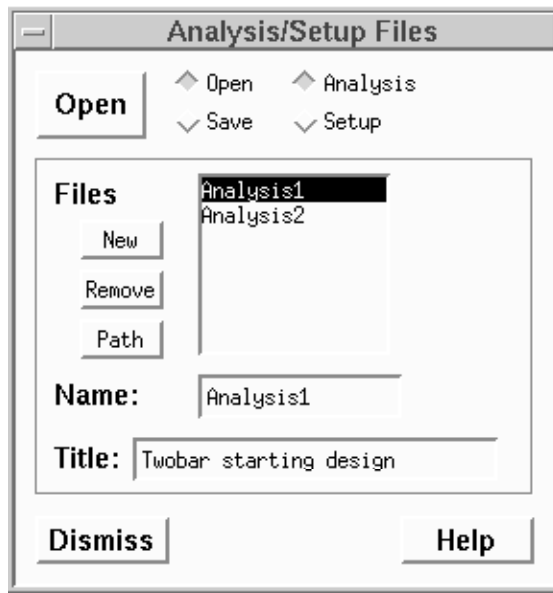
The discrete tutorial is started by restoring Analysis and Setup files saved in the first tutorial. If for some reason these files are not available, manually recreate the Setup given in the Variables and Functions windows on the next page.

1. Start the Twobar tutorial.

The tutorial is usually located in a directory such as /usr/local/OptdesX/tutorial/, and the default name is “OptdesX.”

2. Set the Analysis/Setup Files window to “Open, Analysis.”

3. Open the Analysis1 file by double clicking on the file name, or clicking once and pressing the Open button.



4. Set the Analysis/Setup Files window to “Open, Setup.” Open the Setup1 file.

The initial variable and function values and setup are displayed in the Variables and Functions windows shown below. The Functions window has been panned to show all the functions.

Variables: Twobar Truss

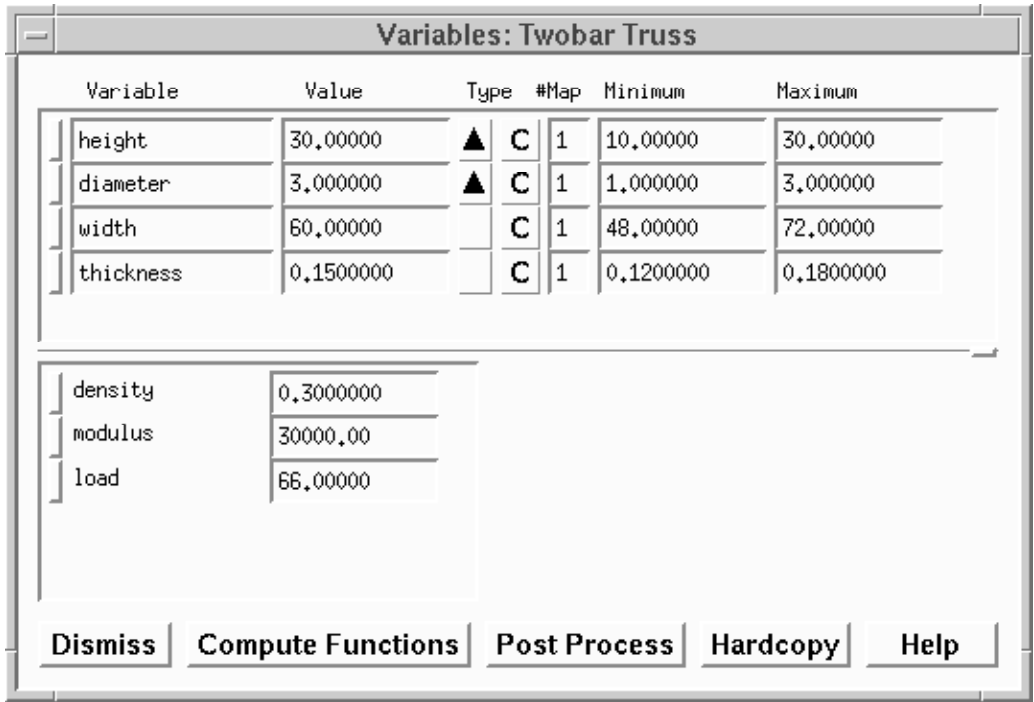
Variable	Value	Type	#Map	Minimum	Maximum
height	30,00000	▲ C	1	10,00000	30,00000
diameter	3,000000	▲ C	1	1,000000	3,000000

width	60,00000
thickness	0,1500000
density	0,3000000
modulus	30000,00
load	66,00000

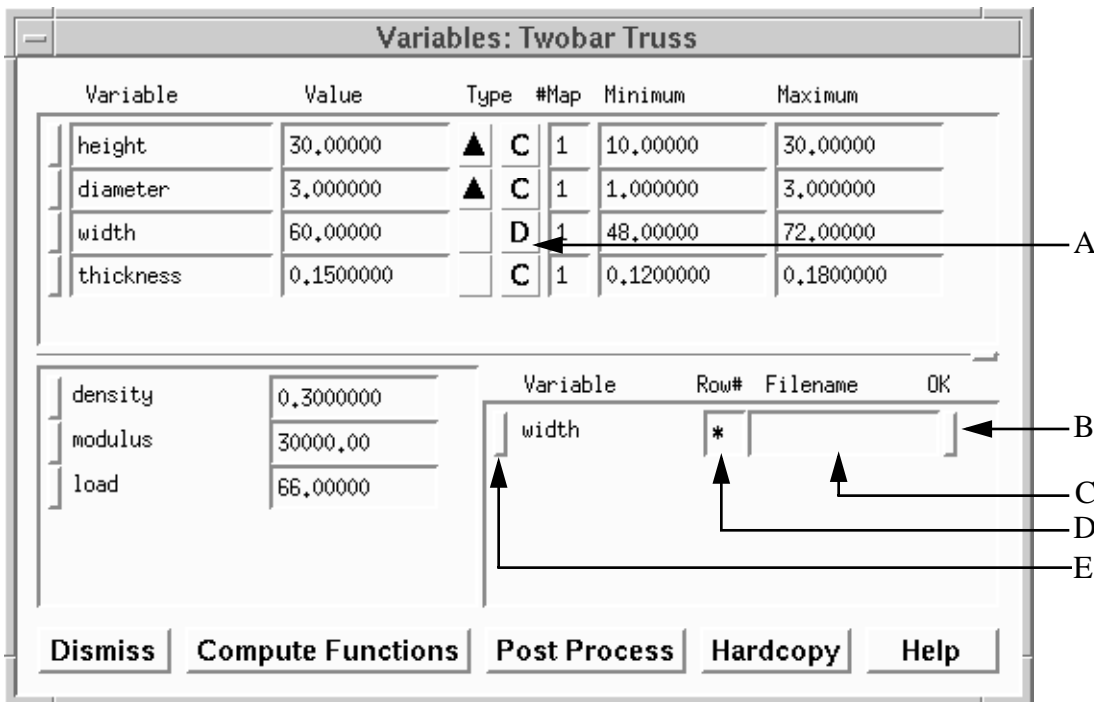
Functions: Twobar Truss

Function	Value	Type	#Map	Allowable	Indifference
weight	35,98735	▼	1	40,00000	10,00000
stress	33,01160	↖	1	100,0000	50,00000
stress-buckling	-152,5061	↖	2	0,000000	-50,00000
Sum stress	33,01160				
- buckling	185,5177				
deflection	0,06602320	↖	1	0,2500000	0,05000000

5. Make width and thickness design variables by pressing their Mapping buttons.



6. Make width a discrete variable by pushing the “C” Type pushbutton.



A - Type pushbutton used to specify that a variable is continuous or discrete. The “D” indicates a discrete variable.

B - OK pushbutton used to read in the data file. If the button is highlighted this indicates the file has successfully been read.

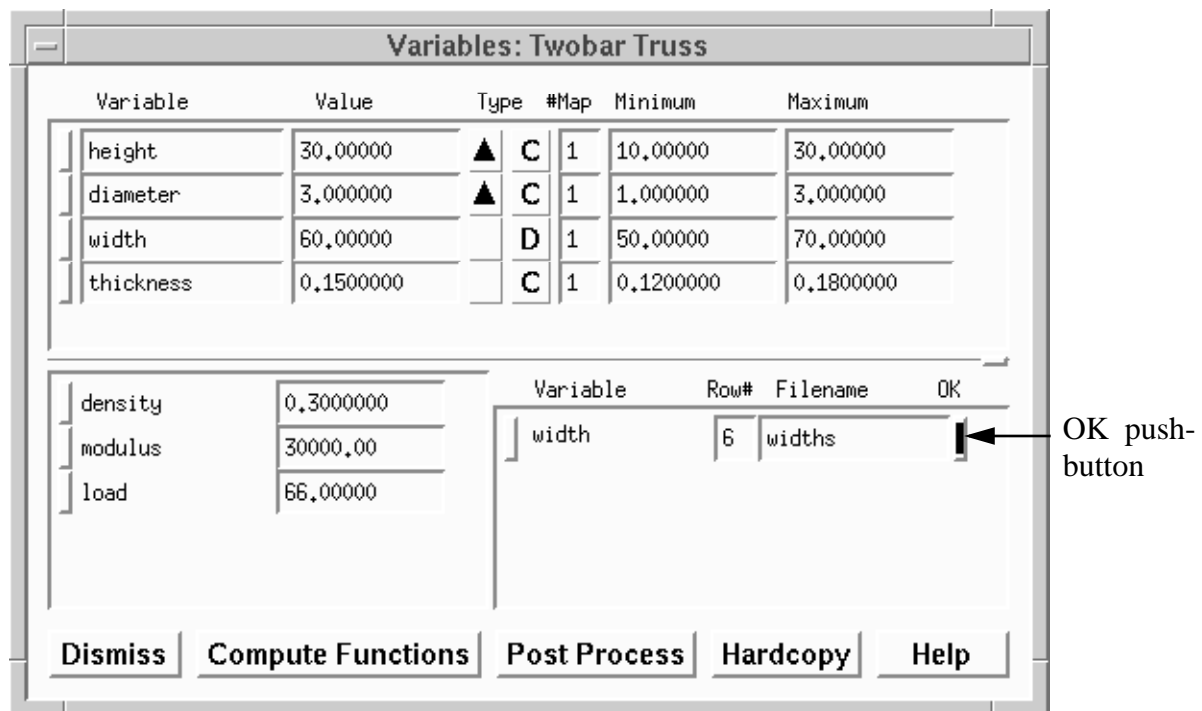
- C - Discrete data file name. This name is limited to 15 characters. If the name is registered with a carriage return, the file will be read and the OK button will highlight.
- D - Row# text field. This gives the row number in the file that corresponds to the current variable value. An asterisk means the current value does not correspond to any discrete values in the file. This field is editable. Typing the row number of a value in the file will set it to the value.
- E - Unmapping button that unmaps the discrete variable, making it continuous.

7. Type in the discrete file name for width, “widths,” in the Filename text field.

8. Push the OK pushbutton to read the file. (Alternatively, if you register the file name with a carriage return when typing it in, this will cause the file to be read.)

The OK pushbutton will highlight indicating that the discrete data file has been read. The Minimum and Maximum for width will be reset to the minimum and maximum values in the discrete file. The value 6 will appear in the Row# value field. This indicates that width is currently equal to the value of the 6th row in the data file.

If the file cannot be read, the button will not highlight and an error message will be printed. This indicates something is wrong with the file format. Check the file and make sure it is the same as shown in the manual. Note that commas are not allowed as delimiters in the file.



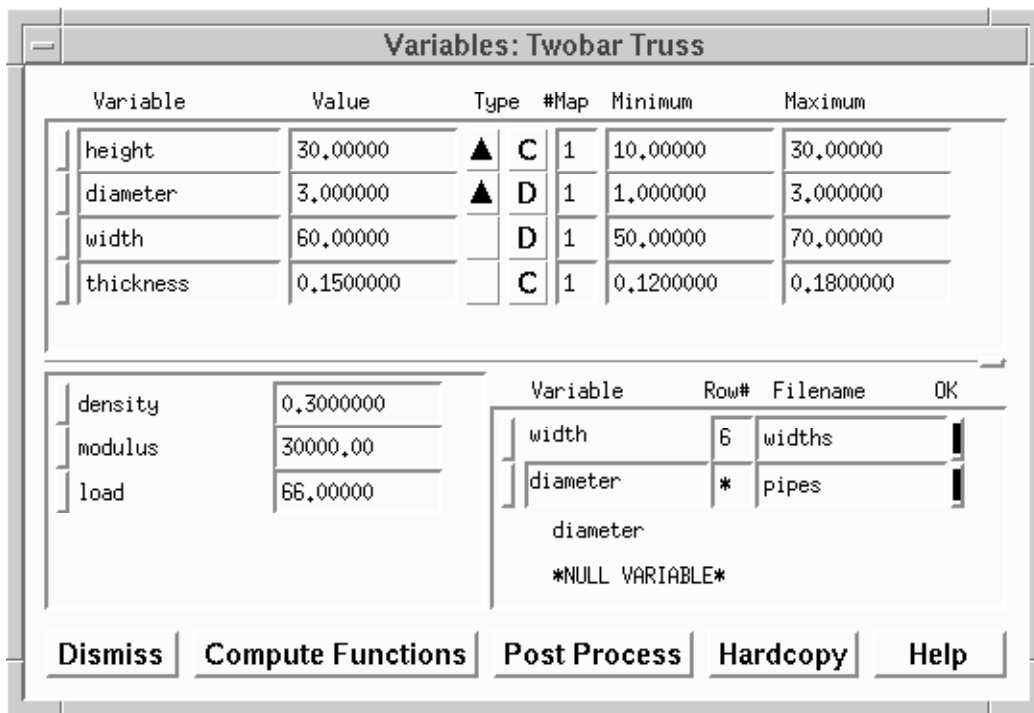
9. Change the diameter Type to discrete by pushing the “C” Type pushbutton.

10. Type in the discrete file name for diameter, “pipes,” in the Filename text field.

11. Push the OK pushbutton to read the file.

We will now map diameter and thickness to be a *related discrete variable*. Two or more variables can be made a related discrete variable when their discrete values are related to each other. For example, for the truss, we wish to select the bars from standard size pipes. This implies that thickness and diameter are discrete, since they must be selected from standard sizes. It also implies that thickness and diameter are related: as diameter increases, the available thicknesses increase also. OptdesX handles related discrete variables as a case of mapping two or more design variables to be one discrete variable.

There are no #Map fields for the discrete variable space. How does OptdesX know that we wish to make a related discrete variable? If the discrete file has more than one column of data, OptdesX assumes you wish to have a related discrete variable. Additional rows open up underneath the current row, with the first row given the current row name and additional rows named "NULL VARIABLE." The next design variable(s) made discrete will fill these slots.

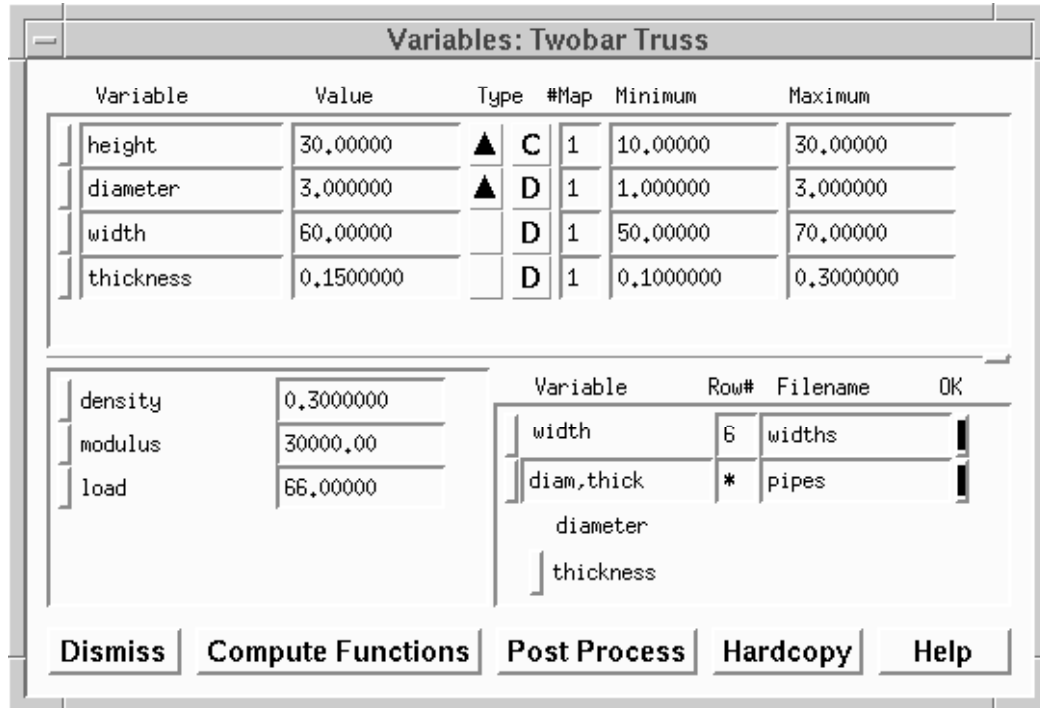


12. Change the Null Variable to thickness by making thickness discrete.

The asterisk in the Row# field indicates that the current values of diameter and thickness do not match any of the discrete values.

13. Change the label of the related discrete variable to "diam,thick."

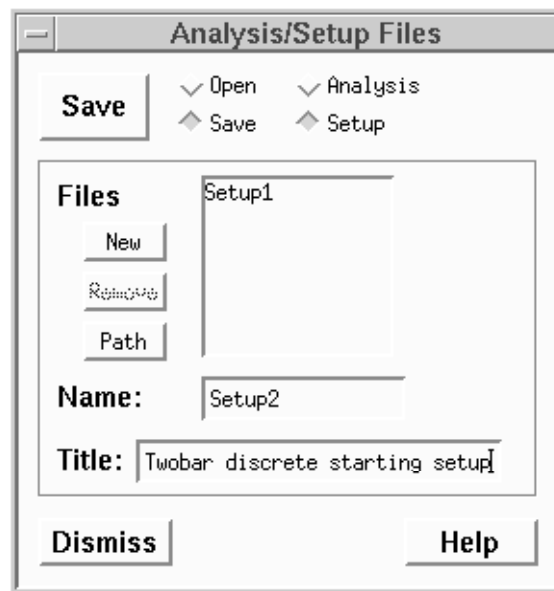
Related discrete variables must be given a unique name.



14. Set the Analysis/Setup Files window to “Save, Setup.”

We will save our discrete setup so we can restore it at any time.

15. Use the default name (“Setup2”) and enter the title, “Twobar discrete starting setup.”



16. Push the Save pushbutton.

Continuous Optimization

Usually the first step in a discrete optimization problem is to find the continuous optimum. We can do this with the GRG or SQP algorithms. If a continuous algorithm is executed when discrete variables are mapped, OptdesX treats the problem strictly as a continuous one, ignoring the discrete nature of the discrete variables and treating them as continuous variables. If the discrete variables were not differentiable, you would need to first unmap them.

17. Run the GRG algorithm from the Optimize Window.

The continuous optimum is shown below.

Variables: Twobar Truss

Variable	Value	Type	#Map	Minimum	Maximum
height	20,24372	C	1	10,00000	30,00000
diameter	1,669169	D	1	1,000000	3,000000
width	50,00000	D	1	50,00000	70,00000
thickness	0,1000000	D	1	0,1000000	0,3000000

density	0,3000000
modulus	30000,00
load	66,00000

Variable	Row#	Filename	OK
width	11	widths	<input type="checkbox"/>
diam,thick	*	pipes	<input type="checkbox"/>
diameter			
thickness			

Dismiss Compute Functions Post Process Hardcopy Help

Functions: Twobar Truss

Function	Value	Type	#Map	Allowable	Indifference
weight	10,12118	↓	1	40,00000	10,00000
stress	100,0008	↕	1	100,0000	50,00000
stress-buckling	-0,005635835	↕	2	0,000000	-50,00000
Sum stress	100,0008				
- buckling	100,0064				
deflection	0,1703930	↕	1	0,2500000	0,05000000

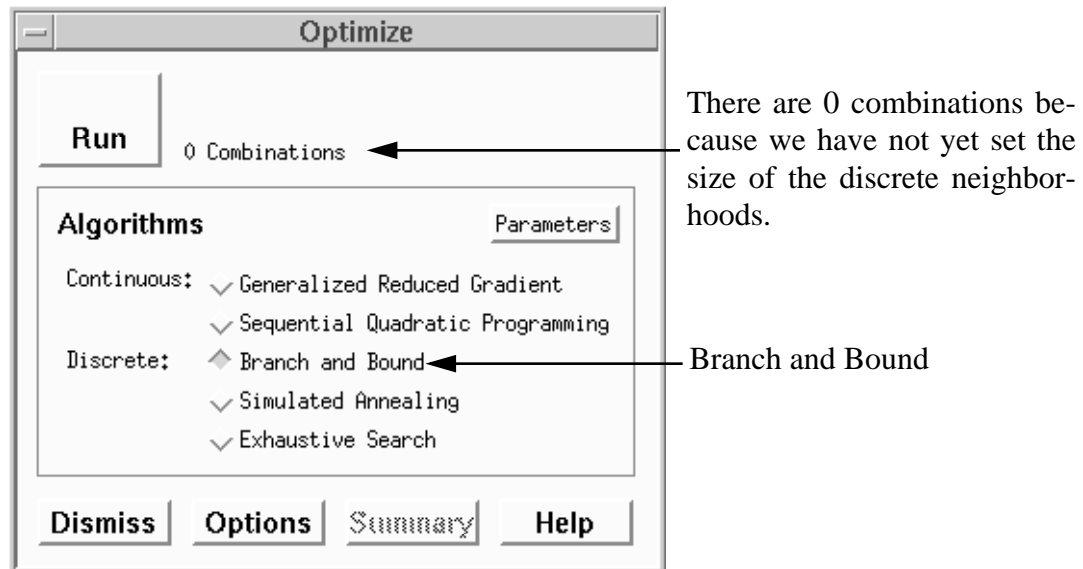
Dismiss Help

18. Select “Save, Analysis” in the Analysis/Setup Files window.

Each discrete algorithm will be started from the continuous optimum so we will save this design to recall it later.

19. Enter the name, “Analysis3,” and the title, “Continuous optimum of first discrete problem.”**20. Push Save in the Analysis/Setup Files window.****Discrete Optimization with Branch and Bound**

The Branch and Bound (BNB) algorithm works by developing a tree structure. At each node (or leaf) in the tree, an optimization is done in order to obtain an estimate of the objective for the nodes below it. If this estimate is worse than a known solution, the branch is “pruned,” i.e., it is not examined further. Because a BNB tree can become prohibitively large, computation and combinations can be reduced by specifying neighborhoods for each discrete variable and by making a linear approximation to the optimization problem at intermediate leaves on the tree. We will explore these options on this problem. A complete description of the BNB algorithm is given in Section 8.7.

21. Select Branch and Bound in the Optimize window.

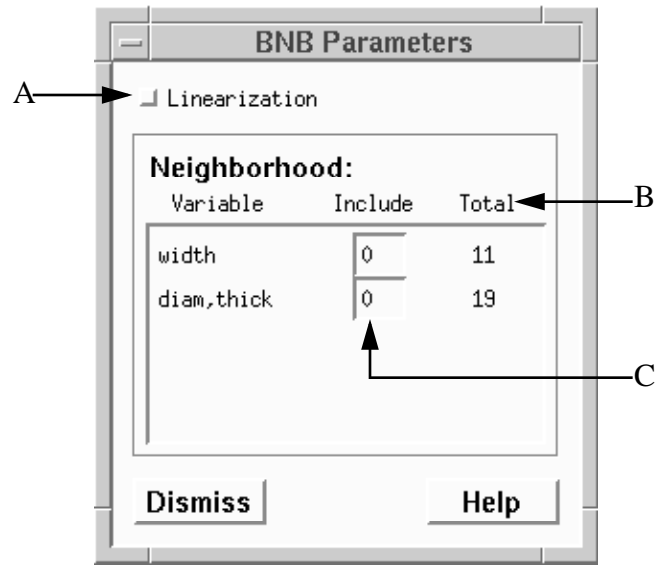
There are 0 combinations because we have not yet set the size of the discrete neighborhoods.

Branch and Bound

The scale and value field for iteration are replaced by the number of combinations, for the neighborhoods, in the discrete problem.

22. Push Parameters in the Optimize window.

We will reduce the size of the discrete optimization problem by limiting the discrete search to a neighborhood around the continuous optimum. Neighborhoods are set in the Parameters box for the algorithm.



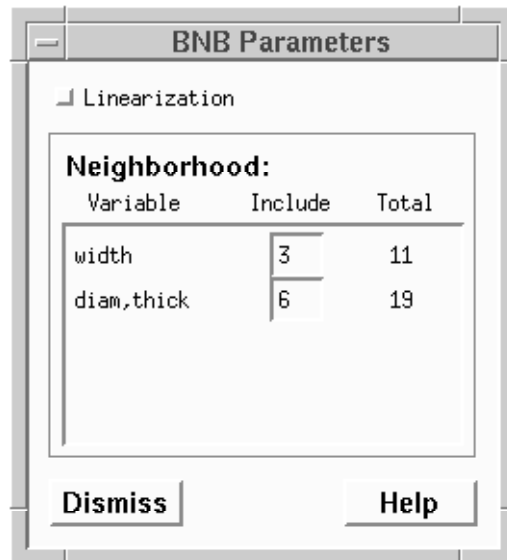
A - Linearization Toggle button.

B - Total value fields giving the total number of discrete values for each variable.

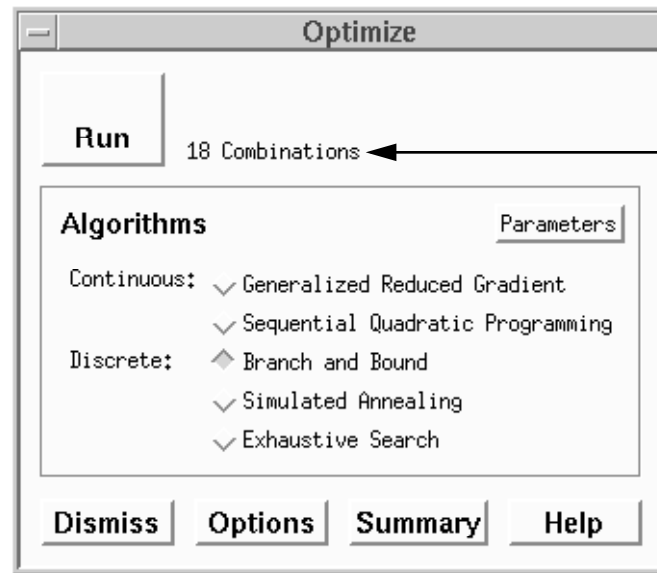
C - Include value fields showing the number of discrete values to be included in the search neighborhood.

23. Change the number of discrete values included for each discrete variable to those shown below.

Register the values by typing a carriage return after each one is entered.



24. Select the Dismiss button in the BNB Parameters window.



Note the change in the number of combinations

25. Push Run in the Optimize window.

You will see the discrete value fields change in the Variables window as the optimization proceeds. Also, data about the progress of the algorithm is displayed in the Optimization Summary window.

Some explanation is required to interpret this data. The summary includes the node number of the tree, values of the discrete variables at the node, and the results of a nonlinear or linear optimization at the node. Nodes are numbered consecutively in order of evaluation. The values of the discrete variables are printed using a format such as,

1 = 6, 2 = 0

which means discrete variable number 1 (which would be the first variable in the discrete variable list in the Variables window) is set to the value given by the 6th row in the corresponding file; the zero for discrete variable 2 means it was continuous at this node (recall that the BNB algorithm starts with all discrete variables continuous, and gradually makes them all discrete).

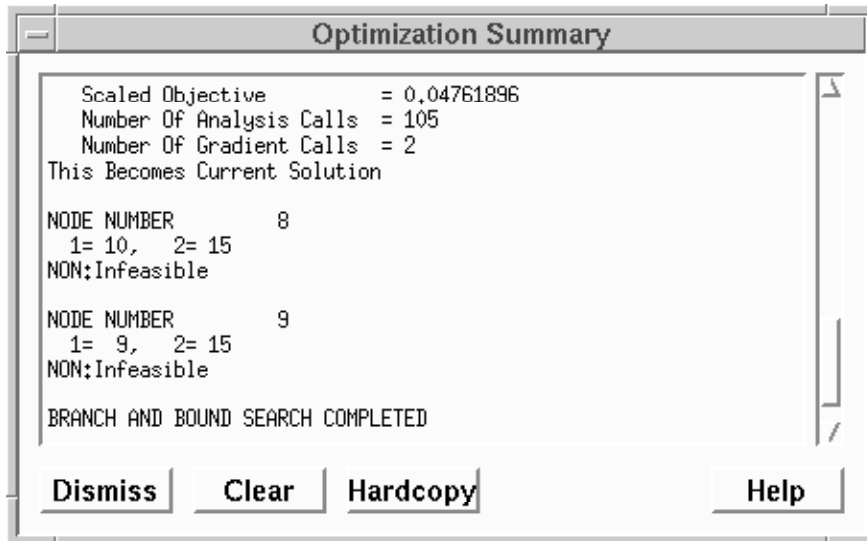
The evaluation of the node is given in the next line of information, such as,

NON: Feasible --Scaled Obj = 1.1427

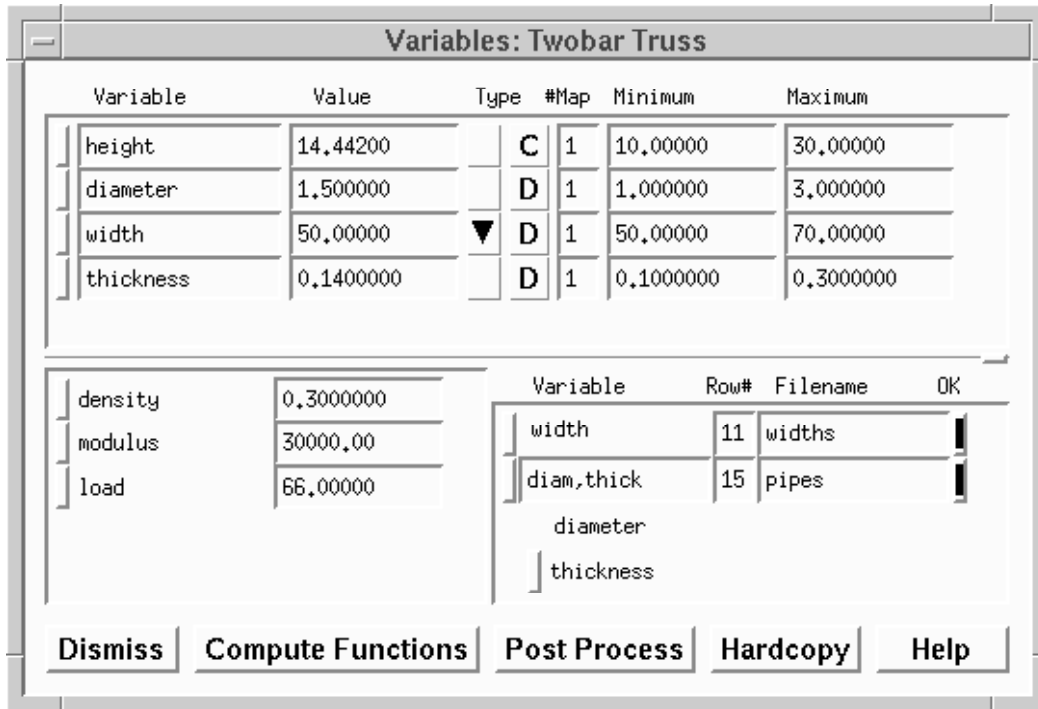
This indicates a nonlinear optimization (“NON”) was done at the node, and the result was a feasible design with the scaled objective given. If this objective is less than the best objective so far, this becomes the next node that is expanded to make a new level in the tree. If this objective is the best and all discrete variables are at discrete values, the message is printed, “This Becomes Current Solution.”

For example, for Node Number 9, shown in the Optimization Summary window on the next page, the first discrete variable, width, is set to row 9 in file “widths;” diam,thick is is set to

row 15 in “pipes.” The nonlinear optimization could not find a feasible design for the continuous variables, so this combination is discarded.



The Branch and Bound optimum is shown below.



Function	Value	Type	#Map	Allowable	Indifference
weight	11.42857	↓	1	40,00000	10,00000
stress	99,99741	↕	1	100,0000	50,00000
stress-buckling	-0,7740430	↕	2	0,000000	-50,00000
Sum stress	99,99741				
buckling	100,7715				
deflection	0,1923902	↕	1	0,2500000	0,05000000

Row #15 from “pipes” and #11 from “widths” give the discrete optimum design.

Optimizing with Linear Branch and Bound

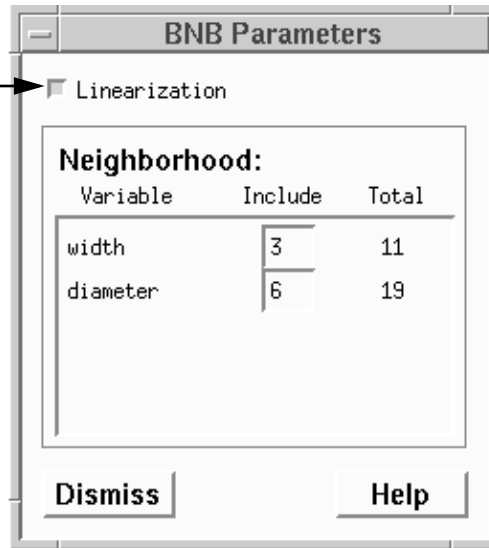
In the previous optimization we did a nonlinear optimization at each node of the BNB tree. Now we will try a strategy that can greatly reduce computation during BNB: we will approximate all functions with a linear Taylor’s series (constructed about the starting point, which for this example is the continuous optimum) and solve the optimization with the simplex method. These optimizations do not require any calls to the analysis model and so can be done very rapidly. If a candidate solution is found, a nonlinear optimization is done to verify the design. As with any approximation, there is some error involved which is dependent on the analysis model and the size of the discrete neighborhoods.

26. Restore the GRG optimum file by selecting “Open, Analysis” in the Analysis/Setup Files window.

27. Open the Branch and Bound parameters window by pressing the Parameters push-button in the Optimize window.

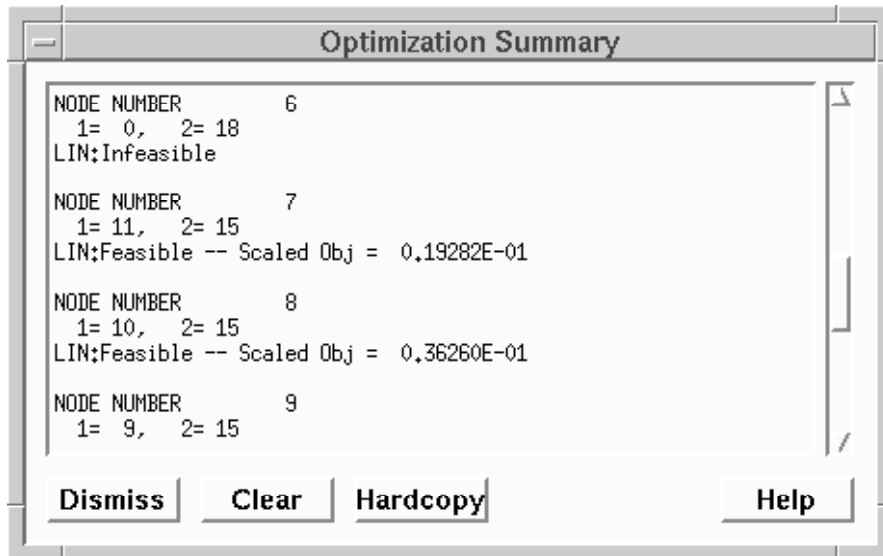
28. Turn on Linearization.

Linearization toggle button on.



29. Dismiss the BNB Parameters window.

30. Select Run in the Optimize window.



With the linearization option turned on, a message such as,

```
LIN: Feasible -- Scaled Obj = 2.143
```

is printed, which gives the result of the linear optimization. Sometimes two messages are printed:

```
LIN: Feasible -- Scaled Obj = 2.143
NON: Feasible -- Scaled Obj = 2.055
```

These messages indicate that the result of the linear optimization was good enough to be con-

sidered a candidate solution, so a nonlinear optimization was done to confirm the linear optimum.

The linear branch and bound optimum is shown in the Variables and Functions windows: Inspection with the previous optimum shows it is the same.

Variables: Twobar Truss

Variable	Value	Type	#Map	Minimum	Maximum
height	14.44200	C	1	10.00000	30.00000
diameter	1.500000	D	1	1.000000	3.000000
width	50.00000	D	1	50.00000	70.00000
thickness	0.1400000	D	1	0.1000000	0.3000000

<table border="1" style="width: 100%; border-collapse: collapse;"> <tr><td>density</td><td>0.3000000</td></tr> <tr><td>modulus</td><td>30000.00</td></tr> <tr><td>load</td><td>66.00000</td></tr> </table>	density	0.3000000	modulus	30000.00	load	66.00000	<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th>Variable</th> <th>Row#</th> <th>Filename</th> <th>OK</th> </tr> </thead> <tbody> <tr> <td>width</td> <td>11</td> <td>widths</td> <td><input type="checkbox"/></td> </tr> <tr> <td>diam,thick</td> <td>15</td> <td>pipes</td> <td><input type="checkbox"/></td> </tr> <tr> <td>diameter</td> <td></td> <td></td> <td></td> </tr> <tr> <td>thickness</td> <td></td> <td></td> <td></td> </tr> </tbody> </table>	Variable	Row#	Filename	OK	width	11	widths	<input type="checkbox"/>	diam,thick	15	pipes	<input type="checkbox"/>	diameter				thickness			
density	0.3000000																										
modulus	30000.00																										
load	66.00000																										
Variable	Row#	Filename	OK																								
width	11	widths	<input type="checkbox"/>																								
diam,thick	15	pipes	<input type="checkbox"/>																								
diameter																											
thickness																											

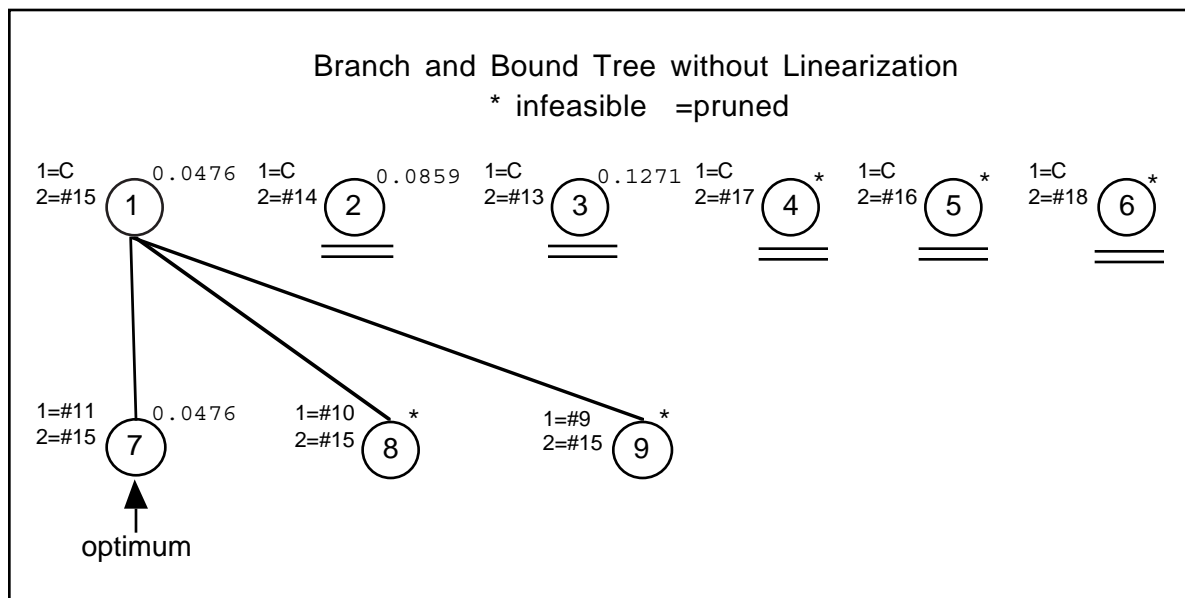
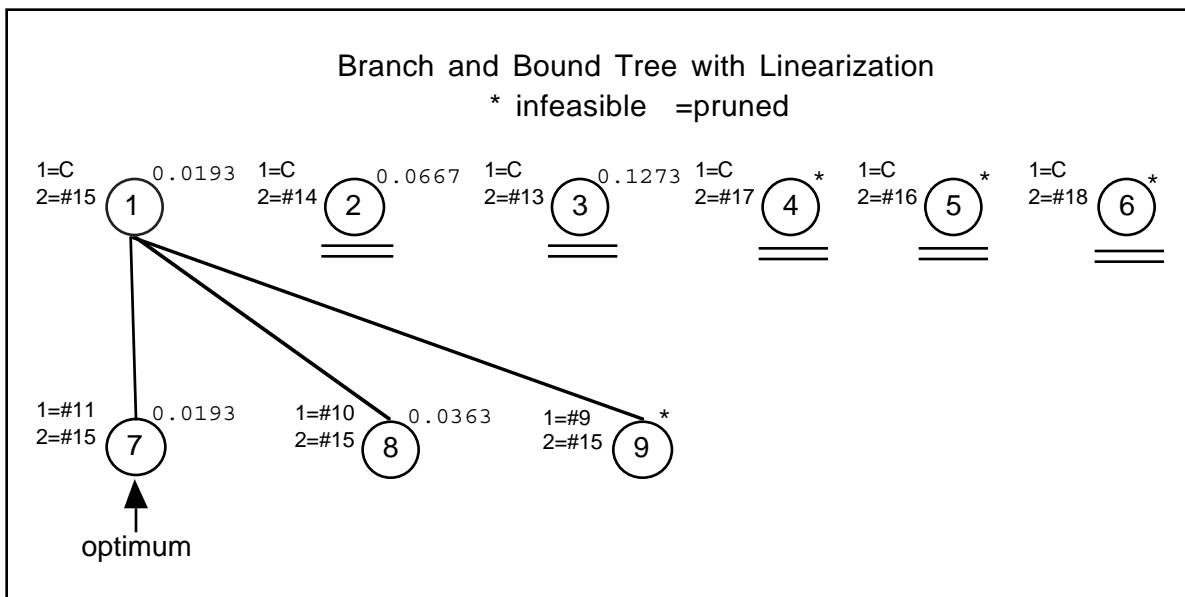
Functions: Twobar Truss

Function	Value	Type	#Map	Allowable	Indifference
weight	11.42857	↓	1	40.00000	10.00000
stress	99.99741	≤	1	100.0000	50.00000
stress-buckling	-0.7740430	≤	2	0.000000	-50.00000
Sum stress	99.99741				
- buckling	100.7715				
deflection	0.1923902	≤	1	0.2500000	0.05000000

Discussion of Branch and Bound Trees Developed in Example

The BNB trees evaluated by OptdesX are shown below. The first tree is the one developed when linearization is selected. Recall that linearization means that at each node the optimization problem is carried out on a Taylor's approximation of the functions. The BNB tree without linearization is given in the second figure. The two trees look the same but close inspection reveals some differences. The BNB tree without linearization shows node 8 to be infeasible, whereas for the linearized BNB a feasible solution was found *to the approximate, linear problem* for this node. This node was pruned anyway, so it made little difference

Also, the objective values for the two trees are different. The scaled objective values for the tree with linearization are from the solution to the approximate problem. The error in the objective is on the order of 100%; however, the trends in the objective are the same, so we found the same solution.



Although the structure of the two trees is the same in this case, this is not true in general. For highly nonlinear problems, and for problems where very large discrete neighborhoods are selected, the two trees can be quite different. The decision to select linearization is best made after a little experimentation on the problem. Of course if computation times are not important, then nonlinear branch and bound should be chosen, since the motivation behind linearization is to reduce the number of calls to the ANAFUN routines.

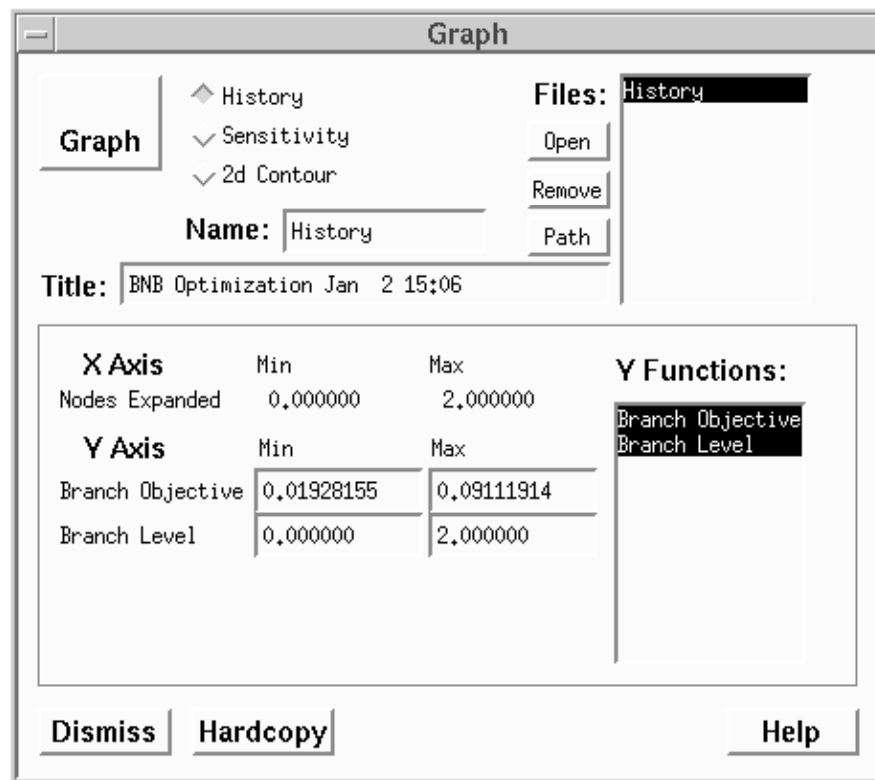
Branch and Bound History Files and Graphs

During the BNB optimization, OptdesX created a History file containing data of the optimization. We can graph this data in the Graph Window.

31. Select Graph from the OptdesX main menu.

32. Select History in the File Type radio box in the Graph window.

33. Open the History file by double clicking on the file name or clicking once on the file name and pressing the Open button.

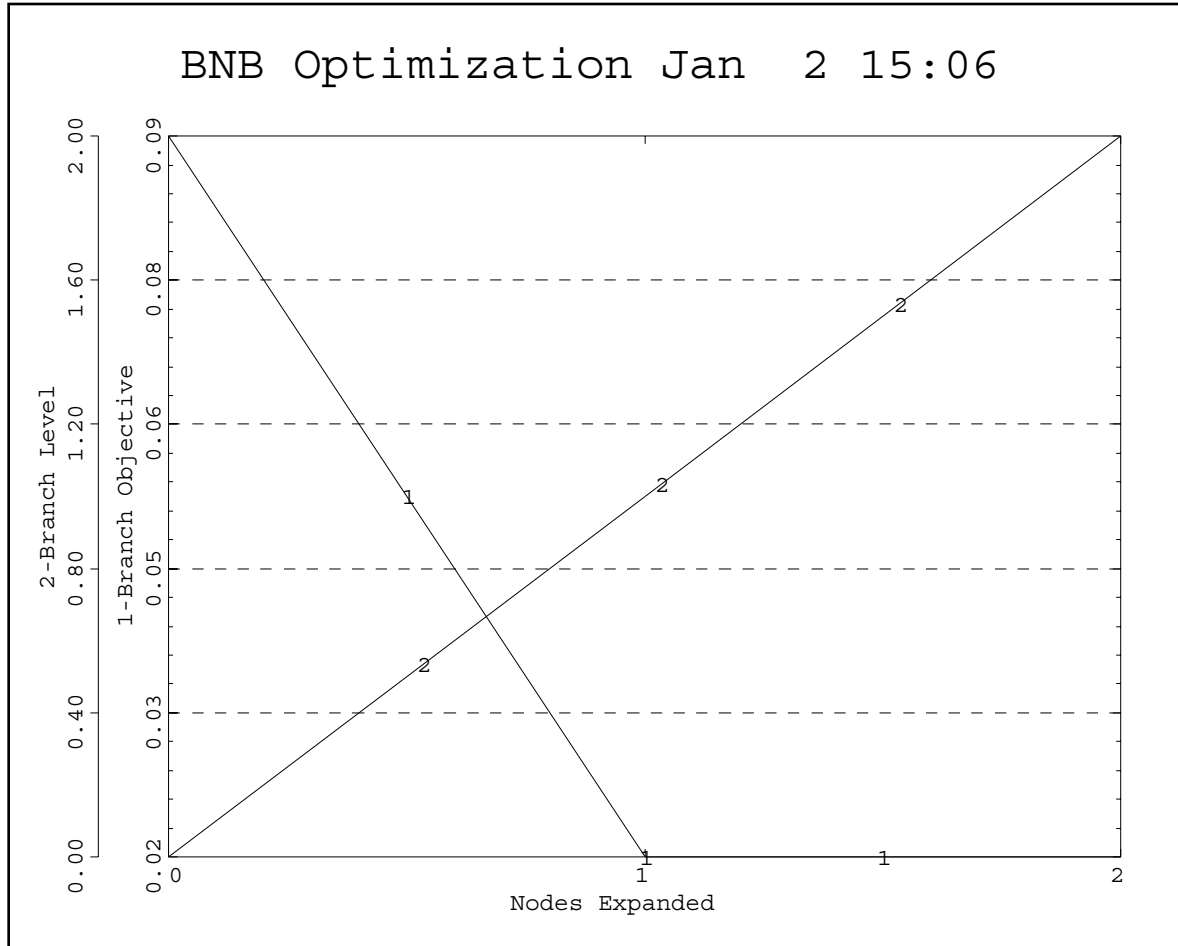


34. Select Branch Objective and Branch Level from the Y Functions list.

35. Push Graph

This History plot is not very interesting because the BNB tree for this problem is so small. The graph plots the Objective value and Level in the tree of the nodes that were expanded and nodes

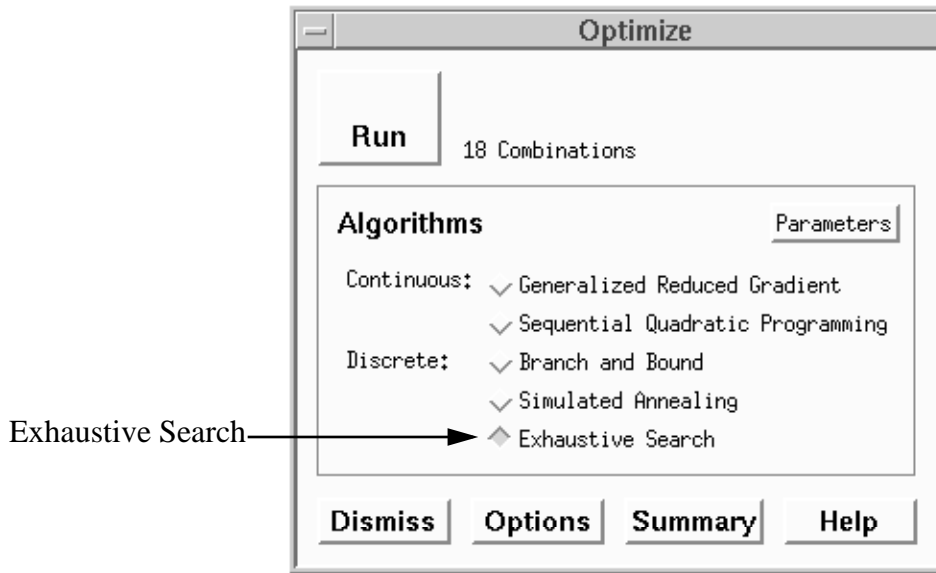
that were chosen to be candidate designs. In this case, there were only two--node 1 was chosen to be expanded at level 1 of the tree (it is graphed as “Nodes Expanded 1”) and node 7 (graphed as “Nodes Expanded 2”) was selected as a candidate and the final optimal solution. A more complex tree is explained in Section 8.7.6.



Optimizing with Linear Exhaustive Search

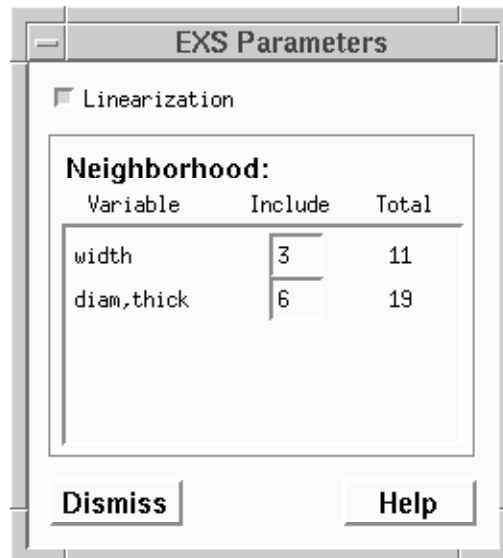
The linear exhaustive search strategy is selected next. Exhaustive search will explore every possible discrete design in the neighborhood. The linearized strategy approximates all functions with a first order Taylor’s series (the point of expansion being the continuous optimum) at each node except when a discrete solution is identified that is a candidate optimal solution. Then a nonlinear optimization is done.

36. Set the Analysis/Setup Files window to “Open, Analysis.”
37. Restore the GRG optimum by opening the Analysis3 file.
38. Select Exhaustive Search in the Optimize window.



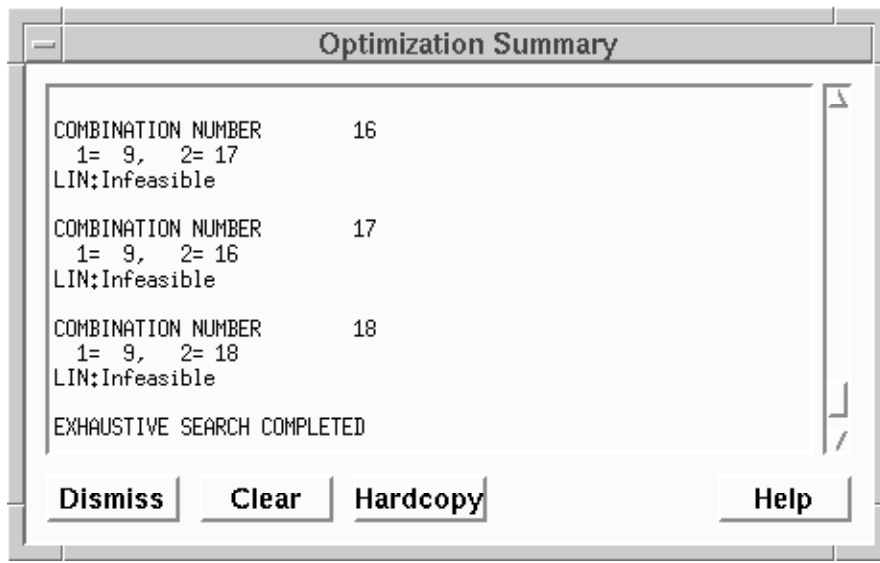
39. Press the Parameters pushbutton in the Optimize window.

The same parameters can be set for EXS as for BNB. We will keep the same neighborhoods and use the linearization strategy. For each combination OptdesX will do an optimization for the continuous variables. This optimization will be done with the simplex method on a linear approximation to the nonlinear problem.



40. Push Run in the Optimize window.

The information in the Optimization Summary window shows the combination being evaluated, whether it is feasible or infeasible, and, if feasible, what the Scaled Objective value is. When a feasible discrete solution is found with a better objective value than the current solution, it is made the current solution.



The linear EXS optimum is the same one obtained with the BNB algorithm (Refer back several pages to the Variables and Functions windows which show the optimum for BNB).

How does EXS compare to BNB for solving this problem? EXS evaluated 18 combinations. At each combination a continuous optimization in one variable was executed. BNB evaluated 10 nodes. At each of these nodes a continuous optimization was executed; for 7 nodes the optimization problem had two variables; for 3 nodes the problem had 1 variable. Overall comparison of computation: about the same for the two approaches. This is true partly because this problem was so small. For larger problems BNB is usually much more efficient, sometimes evaluating less than a fraction of 1% of the possible combinations.

Of course, we could also optimize with the linearization strategy turned off. You may wish to do this to verify that you obtain the same optimum.

Optimizing with Four Non-differentiable Discrete Variables

So far in the tutorial we have executed the BNB and EXS algorithms on a problem with two discrete variables, one of which, “diam,thick,” related two design variables together. The BNB algorithm requires the discrete variables to be differentiable, since it models discrete variables as continuous while it builds the BNB tree. The EXS algorithm does not have this requirement; however, unless the problem is small, EXS is impractical. One way to make the problem small is to restrict the search to discrete neighborhoods close to the continuous optimum--the catch is that getting the continuous optimum requires the discrete variables to be differentiable.

What can we do if our discrete variables are not differentiable, i.e. cannot be modeled as continuous variables? The best algorithm for this situation is Simulated Annealing (SAN). In this section we will optimize a problem with four discrete variables and we will not assume they are differentiable.

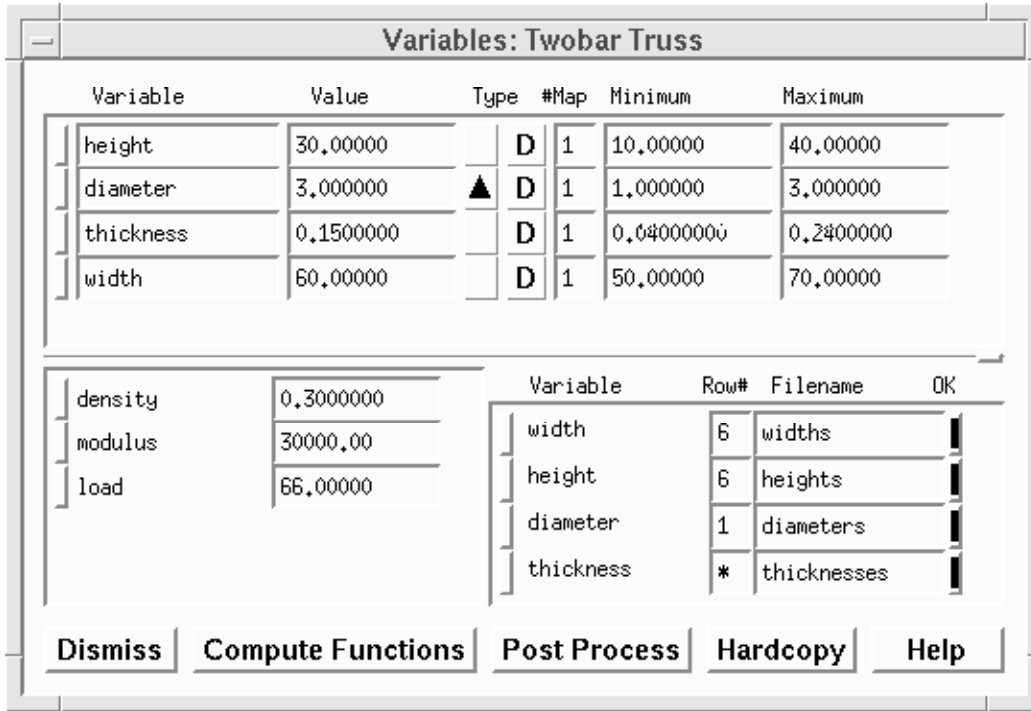
As a side note we should state the SAN is a good algorithm as well when the variables are differentiable. An option which can be set in the parameters of SAN uses derivatives to filter out many analysis calls, making the algorithm even more efficient. SAN in OptdesX does have the limitation that it cannot handle mixed problems. It can be applied to pure discrete problems only.

41. Using your system editor, create the discrete files, “heights,” “diameters,” and “thicknesses,” with the data shown below.

We will also use the file “widths,” which has already been created. The contents of these files are:

heights:	diameters:	thicknesses:
16 1	11 1	11 1
40.	3.	.24
38.	2.8	.22
36.	2.6	.20
34.	2.4	.18
32.	2.2	.16
30.	2.0	.14
28.	1.8	.12
26.	1.6	.10
24.	1.4	.08
22.	1.2	.06
20.	1.0	.04
18.		
16.		
14.		
12.		
10.		

42. Map all four design variables to be discrete, and read in the discrete file for each. Set the variables to the values shown below:

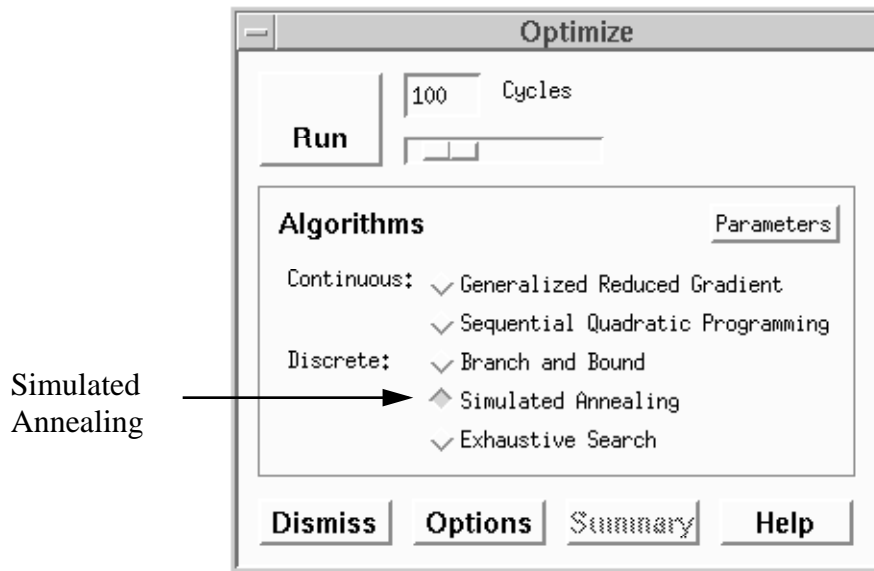


When the discrete files are read, the Mins and Maxes are reset to correspond to the values in the files.

43. Select the Simulated Annealing Algorithm.

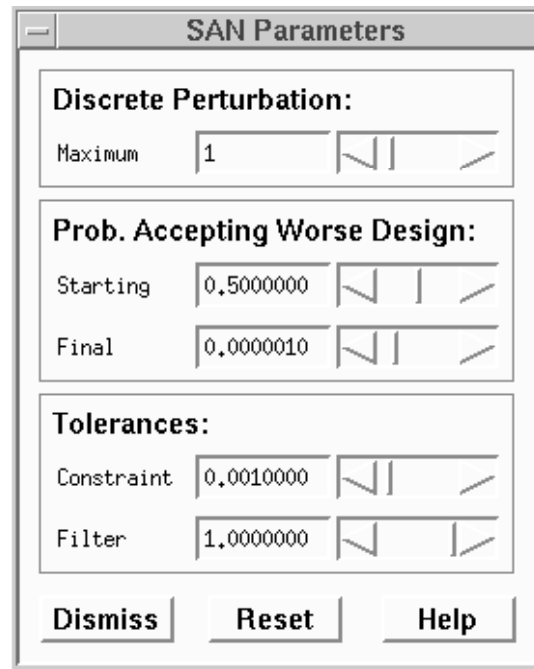
The Simulated Annealing (SAN) algorithm is used to optimize problems with discrete variables only. The algorithm mimics the process of annealing of solids in nature, perturbing the design randomly until a lower “energy” (i.e. objective) is reached. The algorithm has the capability to escape local optima by occasionally accepting designs, according to a probability function, with a worse objective value. Initially the algorithm is started at a high “temperature,” meaning there is a relatively high probability a perturbed design with a worse objective will be accepted as the current design. As the algorithm progresses, the design “freezes,” and the probability that a poorer design is accepted becomes small. More detailed information about SAN is given in Section 8.8.

Because SAN only optimizes discrete variables, any design variables that are not discrete remain constant during a SAN optimization, just as if they were analysis variables. SAN can be started from a feasible or infeasible design. If started infeasible, it temporarily ignores the objective while it minimizes the maximum constraint violation. As soon as the design is feasible, it restarts itself and begins optimizing the objective.



44. Open the Parameters window for SAN.

Even though we will use the default parameters for SAN, it is instructive to examine the parameters for the algorithm:



Maximum Discrete Perturbation

At each “temperature” or cycle of the SAN algorithm, each discrete variable of the design is perturbed randomly in turn. This parameter sets the maximum perturbation. The default of 1 means that each variable will be perturbed from its current value to the next lowest or highest discrete value. A value of 2 would mean each variable will be perturbed up or down 1 or 2 closest discrete values from the current value.

Probability of Accepting a Worse Design

Starting--This is the starting probability that a perturbed design with a worse objective value will be accepted and become the current design.

Final--As SAN proceeds and the design “freezes,” the probability of replacing the current design with a worse design gradually approaches the final probability until at termination it is equal to it. Typically this is a small value.

Constraint Tolerance

A constraint is considered feasible in scaled space if its value is less than or equal to the allowable plus this tolerance.

Filter Tolerance

If gradient information is available, it can be used to make SAN even more efficient. The filter tolerance allows OptdesX to filter out many poor designs based on approximations of the functions made with gradients. Since we are assuming the variables are non-differentiable, we will leave this value at one, which turns the filter off.

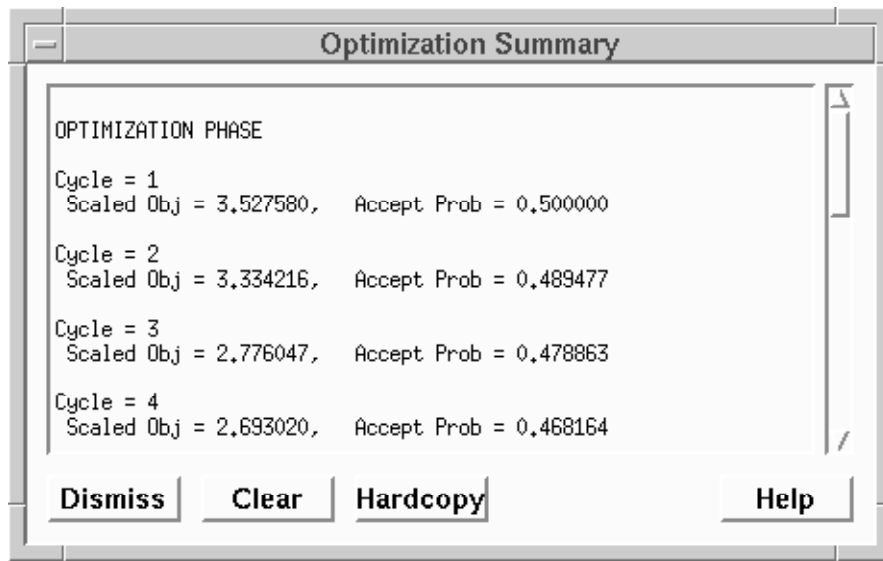
45. Close the Parameters window.

46. Press the Optimize button to begin optimization.

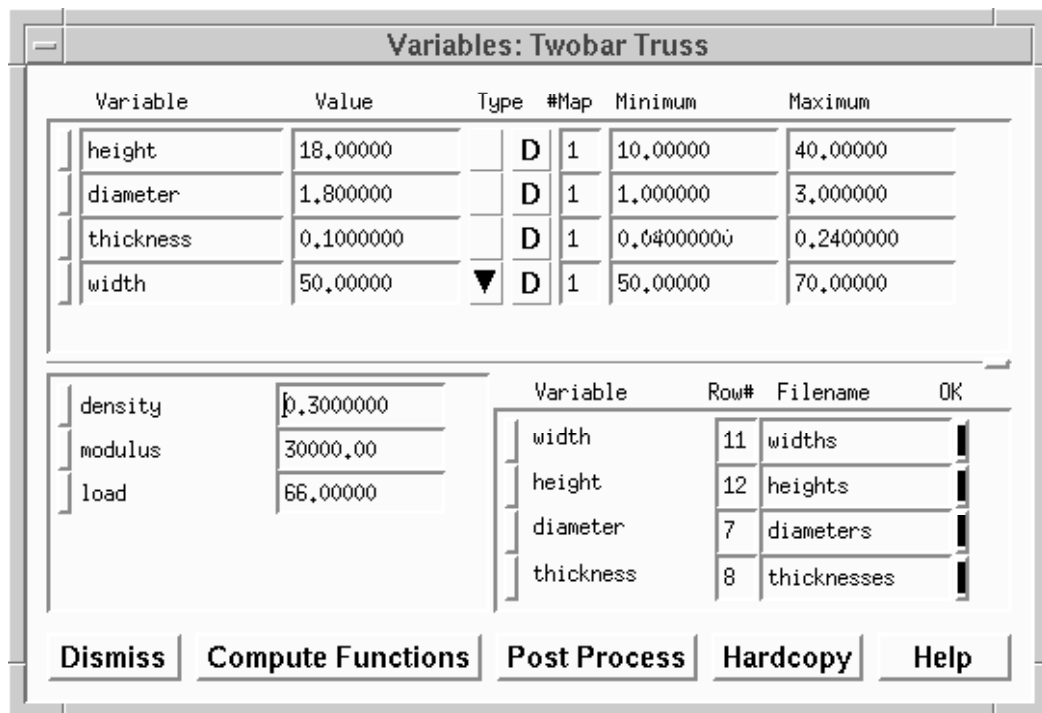
We will run 100 cycles. For each cycle, each of the discrete variables is perturbed in turn. If the new design is better, it is accepted as the current design. If it is worse, it may still be accepted according to the Boltzmann probability function (See Section 8.8).

We are starting the design quite a distance from the optimum. We do not set neighborhoods for SAN like we do for BNB and EXS. This is OK--SAN can move through discrete space without suffering from the combinatorial explosion that BNB and EXS are subject to. SAN is a heuristic algorithm--it does not guarantee that it will find the global optimum, but it often finds very good solutions with only a fraction of the computation of other algorithms. In fact, due to the random nature of the design perturbations, SAN will often return different solutions when run multiple times from the same starting point.

The Optimization Summary window provides some information about the algorithm. Each cycle corresponds to a “temperature;” each temperature sets the probability of accepting a worse design as the current design. The scaled objective value and acceptance probability are printed.



Your optimum should match these Variables and Functions:



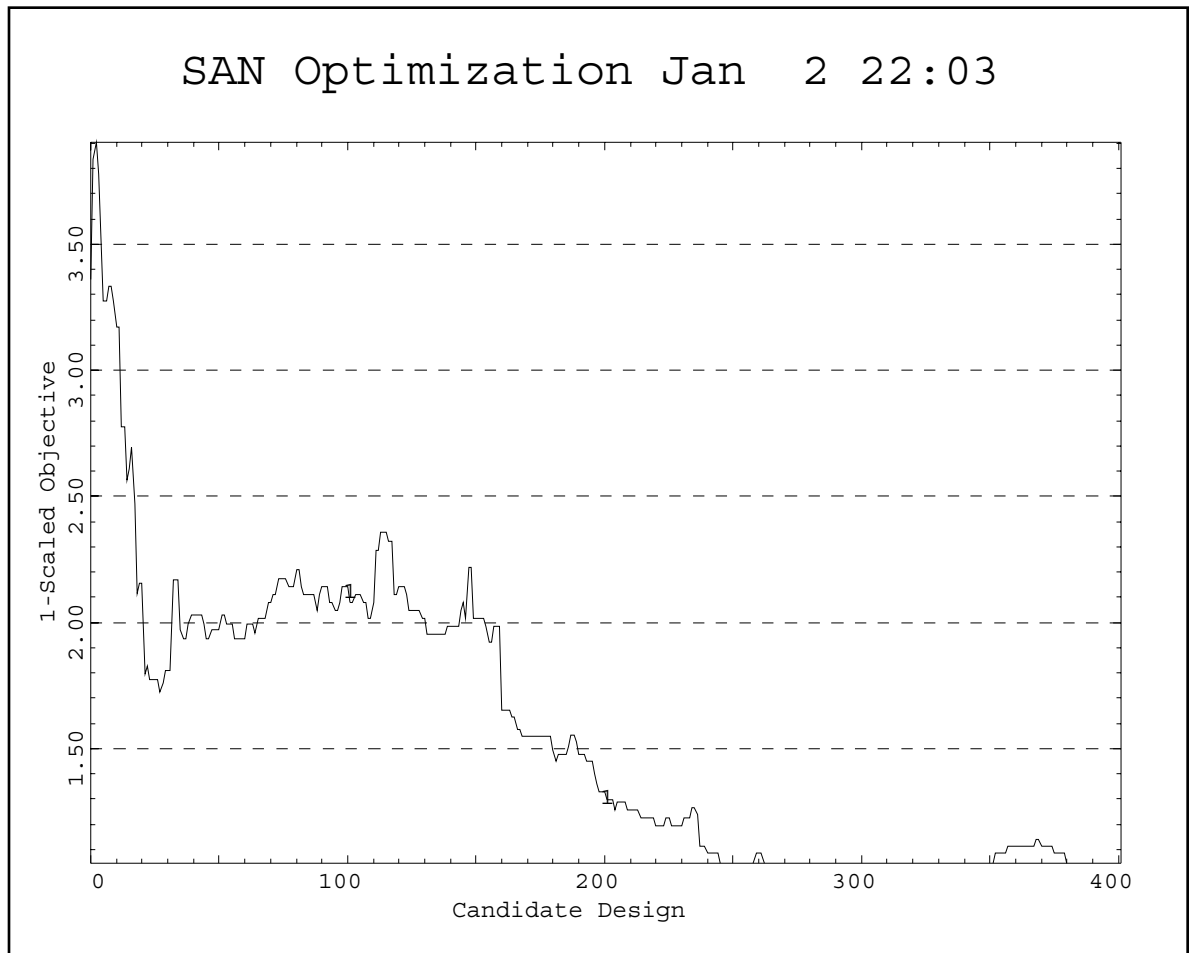
Function	Value	Type	#Map	Allowable	Indifference
weight	10.45218	↓	1	10.00000	0.000000
stress	99.87393	↖	1	100.0000	50.00000
stress-buckling	-26.87613	↖	2	0.000000	-50.00000
Sum stress	99.87393				
- buckling	126.7501				
deflection	0.1755192	↖	1	0.2500000	0.000000

Dismiss Help

47. Open the Graph window. Select the History file just generated by SAN.
 There are two functions which can be plotted: Scaled Violation and Scaled Objective. Scaled Violation is useful only if the beginning design is infeasible, which was not the case here.

48. Select “Scaled Objective” as the function to be plotted.

<input checked="" type="checkbox"/> History <input type="checkbox"/> Sensitivity <input type="checkbox"/> 2d Contour		Files: History Open Remove Path
Name: History		
Title: SAN Optimization Dec 27 17:21		
X Axis Candidate Design Min: 0.000000 Max: 401.0000		Y Functions: Scaled Violation Scaled Objective
Y Axis Scaled Objective Min: 1.045218 Max: 3.903153		
Dismiss		Hardcopy
		Help

49. Press the Graph pushbutton.

This graph shows how the objective changed as the optimization proceeded. Increases in the objective represent perturbed designs that were worse than the current design but which were accepted as the current design according to the Boltzmann probability function. As the optimization progresses, this happens less and less frequently, until, at the end, the design is frozen. Note, however, that we did get a little “burp” in the graph near the end of the optimization, where worse designs were accepted.

The abscissa is given as “Candidate Designs.” A candidate design is a design that is accepted as the current solution. Since there are ndv ($ndv = \text{no. of design variables}$) designs evaluated for each cycle, dividing Candidate Designs by ndv gives the number of cycles executed.

50. Quit OptdesX. You are finished with the Discrete tutorial.

3.5 Tutorial 3: Gradients

In this tutorial we will learn how to use the Gradients window to help insure the success of the optimization algorithms and to interpret information about the sensitivity of functions. The gradient vector, denoted as ∇f , is defined to be the vector of partial derivatives of a function (which we consider to be a column vector, although to save space below we write the transpose of the gradient, which is a row vector):

$$\nabla f^T = \left[\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial x_3}, \dots, \frac{\partial f}{\partial x_n} \right]$$

The gradient vectors of several functions form a matrix, where each row is composed of the derivatives of all the functions with respect to a single variable, and each column is a gradient vector, i.e. the derivatives of one function with respect to all the variables.

Gradients are important for a number of reasons: gradients indicate the sensitivity of functions to variables; the magnitudes of the gradients provide a check on the scaling of the problem, and accurate gradients are essential for the successful operation of the GRG, SQP and BNB optimization algorithms in OptdesX.

We will execute OptdesX again on the Twobar Truss problem to illustrate the use of this window. You may wish to refer to Section 7, the Gradients Window Reference, to obtain additional information.

Starting OptdesX

We will begin by restoring files for the Twobar tutorial. If for some reason these files are not available, you should just manually reproduce the Setup given in the Variables and Functions windows on the next page.

1. Start the Twobar tutorial.

The tutorial is usually located in a directory such as `/usr/local/OptdesX/tutorial/`, and the default name is “OptdesX.”

2. Set the Analysis/Setup Files window to “Open, Analysis.”

3. Open the Analysis1 file by double clicking the file name, or clicking once and pressing the Open button.

4. Set the Analysis/Setup Files window to “Open, Setup.”

5. Open the Setup1 file by double clicking the file name, or clicking once and pressing the Open button.

6. Make width and thickness design variables by pressing their Mapping buttons.

7. Set minimum and maximum values to those shown in the window below.

Variable	Value	Type	#Map	Minimum	Maximum
height	30,00000	▲ C	1	10,00000	30,00000
diameter	3,000000	▲ C	1	1,000000	3,000000
width	60,00000	C	1	40,00000	80,00000
thickness	0,1500000	C	1	0,05000000	0,5000000
density	0,3000000				
modulus	30000,00				
load	66,00000				

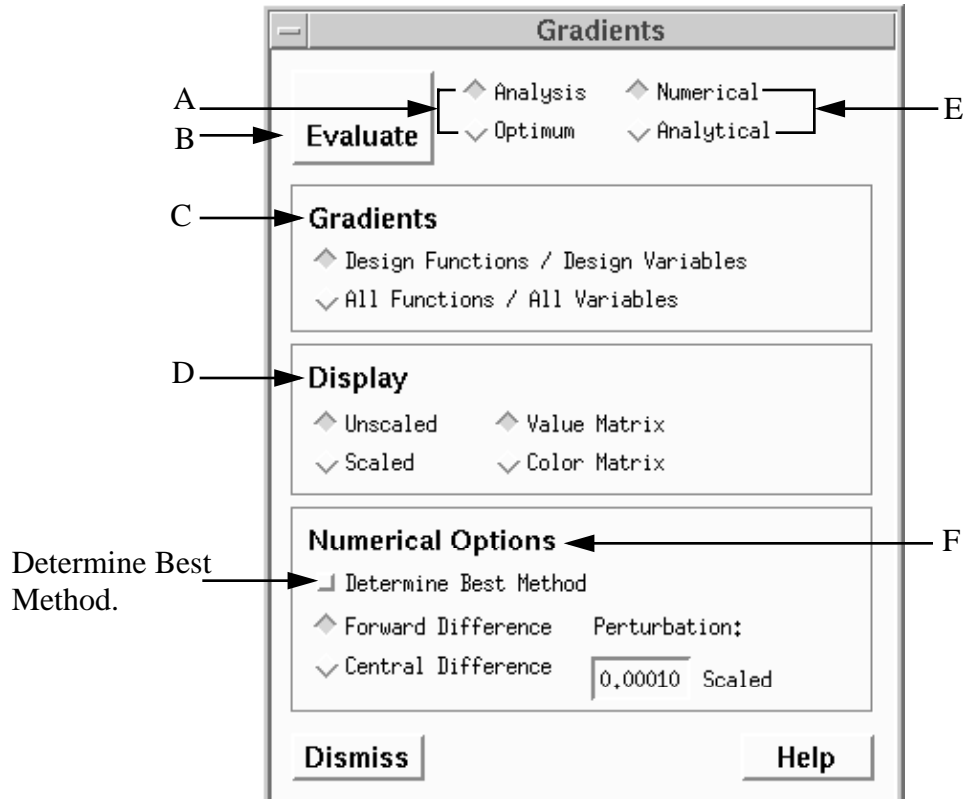
Dismiss Compute Functions Post Process Hardcopy Help

Function	Value	Type	#Map	Allowable	Indifference
weight	35,98735	▼	1	40,00000	10,00000
stress	33,01160	≤	1	100,0000	50,00000
stress-buckling	-152,5061	≤	2	0,000000	-50,00000
Sum stress	33,01160				
- buckling	185,5177				
deflection	0,06602320	≤	1	0,2500000	0,05000000

Dismiss Help

Analysis Gradient Calculation and Display

8. Open the Gradients window by pressing the Gradients pushbutton in the OptdesX main menu.



- A - Analysis-Optimum radio box used to select gradients of the analysis model or gradients of the optimal objective.
- B - Evaluate pushbutton that causes the gradients to be evaluated. If the gradients are current the pushbutton changes to “Display.” Once gradients are taken, they remain current as long as the design point, the numerical method and perturbation are not changed.
- C - Gradients box used to select the gradients to be evaluated. Choices are gradients of design functions with respect to design variables (the gradients the algorithms use) or gradients of all functions with respect to all variables.
- D - Display box used to select the way gradients are displayed--either in scaled or unscaled form, and either as a matrix of values or a matrix of colors.
- E - Numerical-Analytical radio box. This box is not functional--all gradients are currently evaluated numerically in OptdesX.
- F - Numerical Options box used to select the type of numerical method and perturbation value. The Determine Best Method toggle button is used to estimate the best numerical method and perturbation for a particular model.

9. Press Evaluate in the Gradients window.

An information box will be displayed letting you know that the gradients are being computed. A value matrix window will then open displaying unscaled gradients of the design functions. The Gradients box is set by default to “Design Functions / Design Variables.” Thus gradients with respect to density, modulus, and load are not be taken.



10. Enlarge the window by grabbing the right border and dragging to the right.

	weight	stress	stress-buckling	deflection
height	0.5997942	-0.5501704	5.633651	-0.001100304
diameter	11.99578	-11.00350	-134.3756	-0.02200700
width	0.2998971	0.2750989	3.367010	0.001650621
thickness	239.9157	-220.0443	-226.2133	-0.4400886

$$\frac{\partial \text{weight}}{\partial \text{thickness}}$$

The elements of this matrix are the derivatives of the function given at the top of the column with respect to the variable at the left of the row. The derivatives are evaluated numerically by OptdesX using the method (forward or central difference) and the perturbation shown in the Numerical Options box. In this case we used the default method, which is forward difference. This required four calls to the ANAFUN subroutine, one call for each design variable.

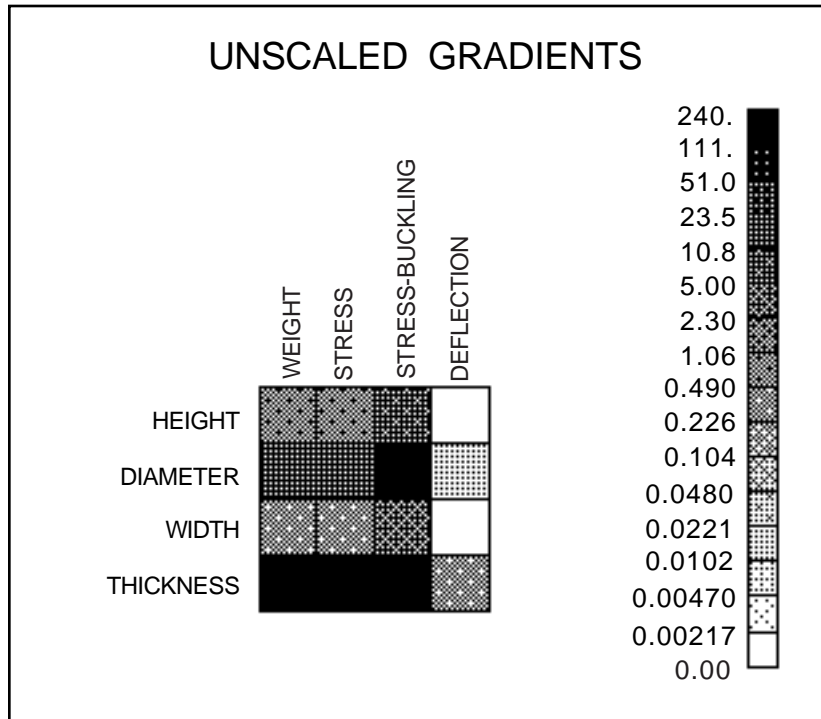
11. Set the Display Value-Color radio box to “Color.”

We can also display gradients as a matrix of colors. A color matrix gives you a sense of the relative magnitudes of the gradient matrix as a whole.

12. Press the Display pushbutton in the Gradients window.

A Color matrix representation of the gradients will be displayed.

Dark blue cells indicate zero (or very small derivatives); while dark red cells indicate large derivatives. A row or column of dark blue cells usually means there is an error in your model--a row of zero derivatives means the associated variable does not affect any of the functions, while a column of zero derivatives means the associated function is not affected by any of the variables. Both cases usually do not make sense in terms of an optimization problem.



13. Set the Unscaled-Scaled radio box to “Scaled.” Set the Value-Color radio box to “Value.”

Once evaluated, gradients are stored and can be displayed a number of ways as long as they remain current. Gradients remain current if the design point, the method and the perturbation do not change. The label on the Evaluate pushbutton changes to “Display” to indicate that the gradients are current.

We will display the scaled values of the gradients. These are the values the algorithms use.

14. Push Display in the Gradients window.

The screenshot shows a window titled "Scaled Gradients Design Functions/Variables". It contains a table with the following data:

	weight	stress	stress-buckling	deflection
height	0,1999314	-0,1100341	1,126730	-0,05501520
diameter	0,3998595	-0,2200700	-2,687512	-0,1100350
width	0,1999314	0,1100396	1,346804	0,1650621
thickness	1,799368	-0,9901994	-1,017960	-0,4950997

A "Dismiss" button is visible at the bottom left of the window.

The OptdesX algorithms work in scaled space. Variables are scaled to between -1 and +1 using minimum and maximum values; functions are scaled to be on the order of 1 using allow-

able and indifference values. Scaling formulae are given in Section 2.4. Since a derivative involves both functions and variables, it is affected by the scaling of both, according to the relationship,

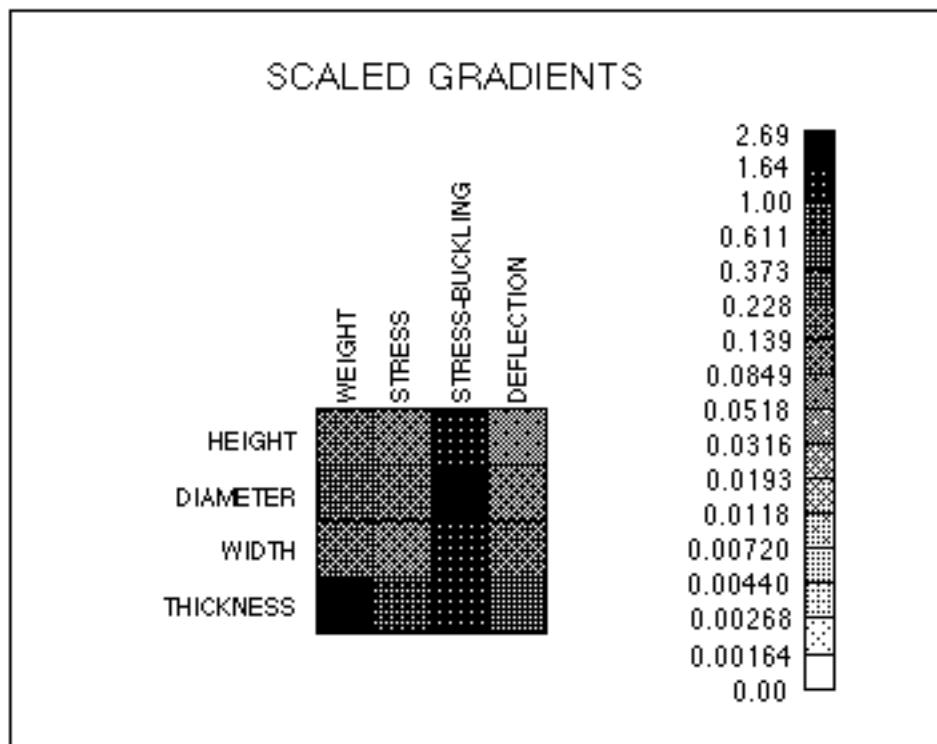
$$\left(\frac{\partial f_i}{\partial x_j}\right)_{\text{scaled}} \approx \left(\frac{\partial f_i}{\partial x_j}\right)_{\text{unscaled}} \left\{ \frac{(\text{Max}_j - \text{Min}_j)}{2(\text{Allowable}_i - \text{Indifference}_i)} \right\}$$

Scaling is important! A poorly scaled problem can cause premature termination of the algorithms. One way to check the scaling of a problem is to check the gradients: when the problem is properly scaled, the gradients should all be roughly the same order of magnitude.

Comparison of the unscaled and scaled gradient matrices shows that scaling has had a “leveling” effect: the stress-buckling gradient has been reduced from magnitudes on the order of 100 to magnitudes on the order of 1, while the deflection gradient has been increased from magnitudes on the order of 0.001 to 0.1.

15. Dismiss the Scaled and Unscaled Value Matrices.

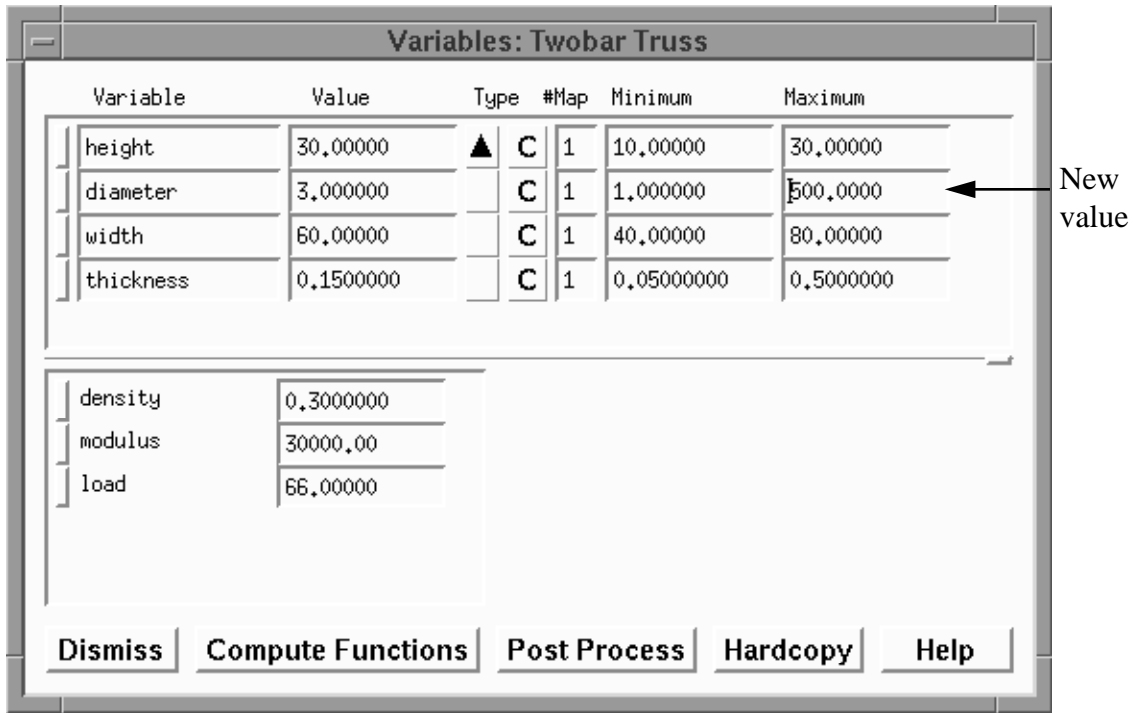
16. Set the Value-Color radio box to “Color.” Push Display in the Gradients window.



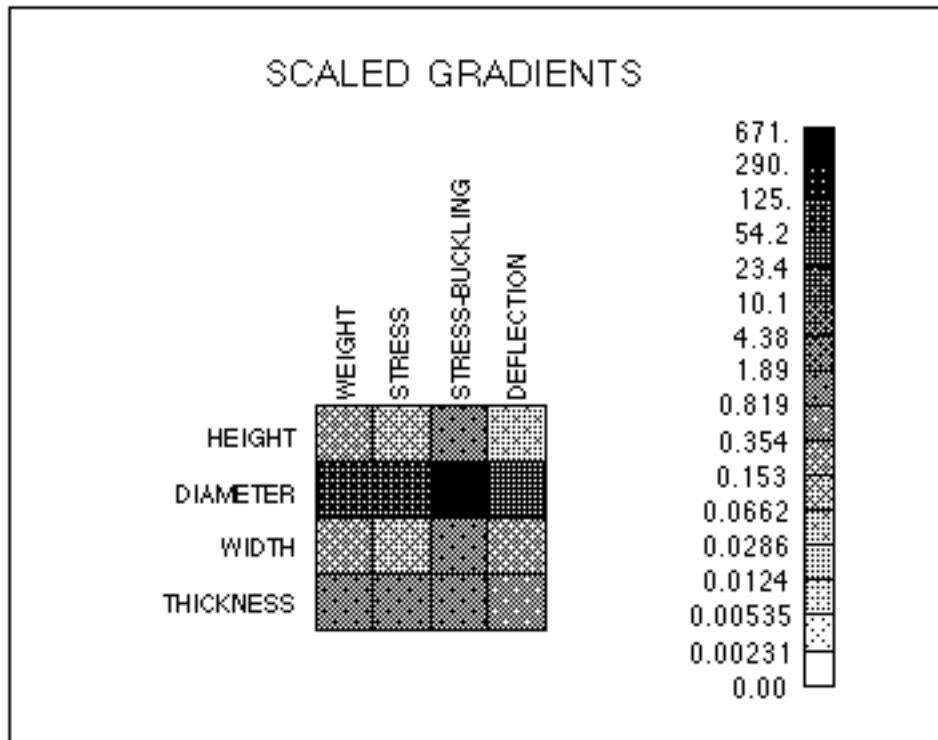
We can see that scaling has not only reduced the range of the derivatives, but the colors are also more uniform across the matrix.

17. Change the scaling of diameter by changing the maximum to 500.0 as shown in the window below.

We will change the scaling to see how it affects the values of the gradients.



18. Press Display in the Gradients window.



The row associated with diameter is now dark red, while the rest of the matrix is light in color. Before the change in scaling, the range of the color matrix was 0-2.69, while now its range is

0-671. The derivatives with respect to diameter are not well scaled. We can also see this in a Value matrix.

19. Set the Value-Color radio box to “Value;” press Display.

	weight	stress	stress-buckling	deflection
height	0.1999314	-0.1100341	1.126730	-0.05501520
diameter	99.76494	-54.90746	-670.5343	-27.45373
width	0.1999314	0.1100396	1.346804	0.1650621
thickness	1.799368	-0.9901994	-1.017960	-0.4950997

The derivatives in this row have been affected by the change in scaling

20. Set the maximum of diameter back to 3.00.

21. Change the allowable of the objective, weight, to 1000.0 as shown below.

Function	Value	Type	#Map	Allowable	Indifference
weight	35.98735	↓	1	1000.000	10.00000
stress	33.01160	↕	1	100.00000	50.00000
stress-buckling	-152.5061	↕	2	0.000000	-50.00000
Sum stress	33.01160				
- buckling	185.5177				
deflection	0.06602320	↕	1	0.2500000	0.05000000

22. Push Display in the Gradients Window.

Increasing the difference of the allowable and indifference values has reduced the scaled de-

derivatives of weight. The derivatives are not well scaled.

	weight	stress	stress-buckling
height	0,006058527	-0,1100341	1,126730
diameter	0,01211695	-0,2200700	-2,687512
width	0,006058527	0,1100396	1,346804
thickness	0,05452629	-0,9901994	-1,017960

This column has been affected by the change in scaling.

23. Dismiss all Value and Color Matrices.

24. Reset the allowable for weight back to 40.

What is the “moral to this story?” It is that you can check the scaling of your problem by checking the magnitudes of the gradients. When well scaled they will all be within one or two orders of magnitude. If a row of derivative values is not in line with the others, you need to adjust the min or max for the associated variable; if a column is out of line, you need to adjust the allowable or indifference for the associated function.

25. Select All Functions/All Variables and Value Matrix in the Gradients and Display boxes.

Gradients

Evaluate

- Analysis
- Numerical
- Optimum
- Analytical

Gradients

- Design Functions / Design Variables
- All Functions / All Variables

Display

- Unscaled
- Value Matrix
- Scaled
- Color Matrix

26. Press Evaluate in the Gradients window.

Previously we evaluated derivatives for the design functions with respect to the design vari-

ables. Since density, modulus and thickness were not design variables, no derivatives with respect to them were taken. Now we display all the derivatives.

	weight	stress	stress-buckling	deflection
height	0,5997942	-0,5501704	5,633651	-0,001100304
diameter	11,99578	-11,00350	-134,3756	-0,02200700
width	0,2998971	0,2750989	3,367010	0,001650621
thickness	239,9157	-220,0443	-226,2133	-0,4400886
density	119,9578	0,000000	0,000000	0,000000
modulus	0,000000	0,000000	-0,006183924	-2,200773e-06
load	0,000000	0,5001757	0,5001757	0,001000351

Dismiss Hardcopy

27. Dismiss the Value matrix.

Derivatives of the Optimum

To display derivatives of the optimum, we must be at an optimum, and we must have just run the SQP or GRG algorithms.

28. Select the Optimize pushbutton from the OptdesX main menu.

29. Push Run in the Optimize window.

Your optimum should be the same as the one shown below.

Variable	Value	Type	#Map	Minimum	Maximum
height	20,09930	C	1	10,00000	30,00000
diameter	2,674187	C	1	1,000000	3,000000
width	40,00000	▼	1	40,00000	80,00000
thickness	0,05540890	C	1	0,05000000	0,5000000

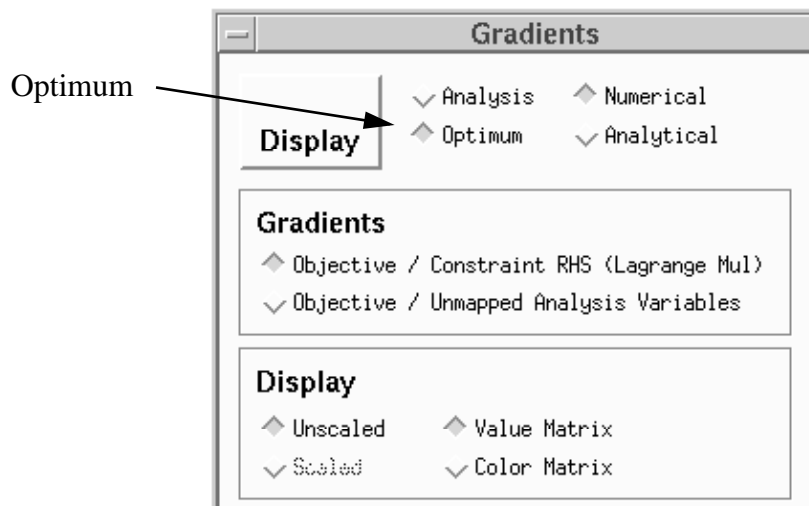
density	0,3000000
modulus	30000,00
load	66,00000

Dismiss Compute Functions Post Process Hardcopy Help

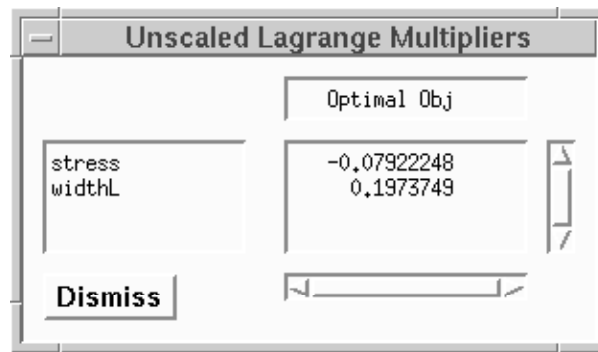
Function	Value	Type	#Map	Allowable	Indifference
weight	7,919460		1	40,00000	10,00000
stress	100,0080		1	100,0000	50,00000
stress-buckling	-229,3397		2	0,000000	-50,00000
Sum stress	100,0080				
- buckling	329,3477				
deflection	0,1333457		1	0,2500000	0,05000000

Dismiss Help

30. Set the Analysis-Optimum radio box in the Gradients window to “Optimum.”
 The contents of the Gradients box changes to “Objective / Constraint RHS (Lagrange Mul)” and “Objective / Unmapped Analysis Variables.”



31. Press Display in the Gradients window.



Lagrange multipliers are derivatives of the *optimal* objective with respect to changes in the allowable values of constraints. These derivatives are zero except for the binding constraints. In the previous optimization, width was at its lower bound (bounds can be viewed as simple constraints) and stress was at its allowable value. The Lagrange multipliers for these two constraints are given above. (The multiplier for the lower bound for width is denoted by the “L” following the variable name; a multiplier for the upper bound would have a “U” appended to the variable name.)

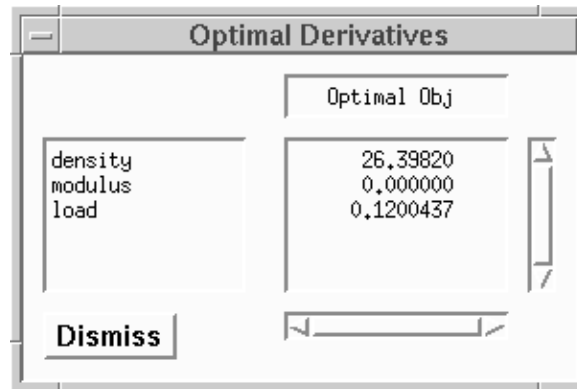
These values can be interpreted in the following way: if we were to increase the right hand side of the stress constraint from 100 to 101, the optimal objective would decrease by approximately -0.0792 . Likewise, if the lower bound for width were to increase by 1, the optimal objective would increase by approximately 0.197 pounds.

Since the Lagrange multipliers are derivatives, they are valid only for infinitesimal changes of the constraint right hand sides. For finite changes, the Lagrange multipliers will only approximately predict the change in the optimal objective. Also, it is assumed that the perturbation is small enough that the same set of binding constraints applies at the new optimum. If the set of binding constraints changes as a result of a perturbation, then predictions based on the current Lagrange multipliers are invalid.

32. Dismiss the Value Matrix window.

33. Select “Objective/Unmapped Analysis Variables” in the Gradients box.

34. Press Evaluate in Gradients window.

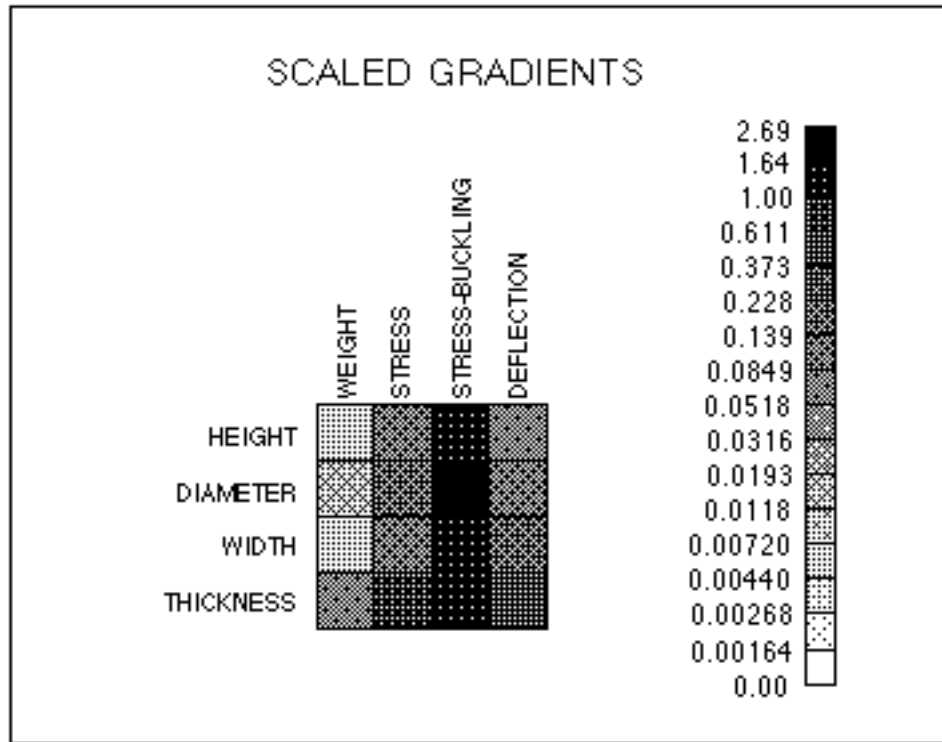


These are derivatives of the *optimal* objective with respect to the unmapped analysis variables. Thus these derivatives show you how the optimum would change if a constant to the optimization were changed. The interpretation of these derivatives is similar to Lagrange multipliers: if load, for example, were increased by 1, the optimal value of the objective would increase by approximately 0.120 pounds.

You can verify optimal derivatives by making a small perturbation to the original problem, re-optimizing, and comparing the change in the objective function with the predicted change given by the derivative.

Because of the way multiple objective problems are handled (Section 6.3.3), optimal derivatives are not available for multiple objective problems.

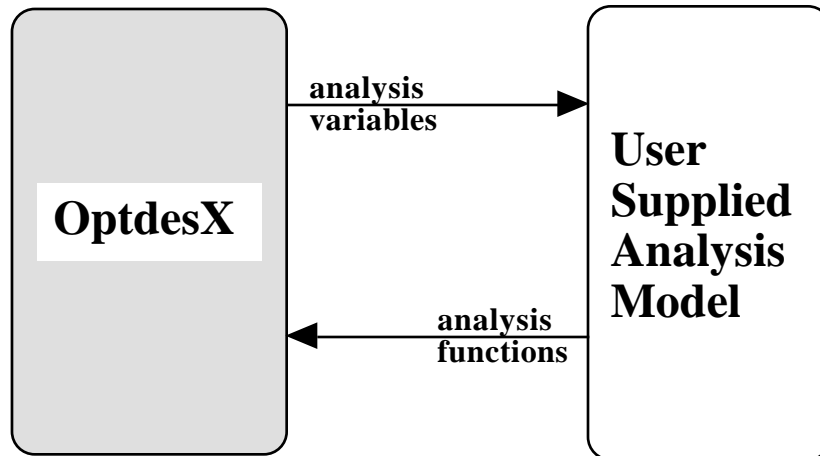
More information about optimal derivatives is given in Section 7.5 of the manual.



4 Linking Analysis Models to OptdesX

4.1 Introduction

In order to apply OptdesX, you must supply analysis software for the problem of interest. During optimization, OptdesX will change the values of the analysis variables. It will then call the analysis model, passing in new variable values; the analysis model will calculate the corresponding analysis functions and pass them back. This operation is depicted in the figure below.



Operation of OptdesX and User-Supplied Analysis Model

The link between OptdesX and your model is made in a subroutine called “ANAFUN” in Fortran, or “anafunC” in C. If you have source code for your model or are writing it from scratch, then you calculate the functions directly in ANAFUN or call other routines from ANAFUN that make the calculations. This is what we call the “conventional” interface. If you do not have access to source code because, for example, the model is developed using commercial software, then you can still link the model to OptdesX by having ANAFUN write an input file for your model, execute the model, and read an output file generated by the model. This type of link is referred to as the “stand-alone” interface. In the following sections we will illustrate both kinds of interfaces in Fortran and C.

How does OptdesX know whether your model is in C or Fortran? Actually it doesn’t. OptdesX calls both ANAFUN and anafunC every time it needs new function values. One of these should be a dummy.

Besides ANAFUN, there are two other routines you may use if you wish to (if you don’t use them, however, you must supply dummies). The ANAPRE routine is available for pre-processing; it is called once when OptdesX is first started. A typical use of ANAPRE is to initialize your model, perhaps by reading a data file. You also specify a model name in ANAPRE. The ANAPOS routine can be used for post-processing. ANAPOS can be called automatically during an optimization or directly from the windows interface (by pressing the Post Process pushbutton in the Variables window). A typical use of ANAPOS is to display a picture of the

design. You determine when ANAPOS is called. Collectively the ANAPRE, ANAFUN and ANAPOS routines are called the “ANASUB” routines. Data can be shared between the ANASUB routines using common blocks in Fortran or global variables in C.

4.2 The Conventional Interface in Fortran

For the conventional interface, OptdesX calls the analysis model directly through the ANAFUN subroutine. The model is either written as part of the ANAFUN routine or it is called from the ANAFUN routine. The steps to link to the ANASUB routines are as follows.

4.2.1 UNIX Systems

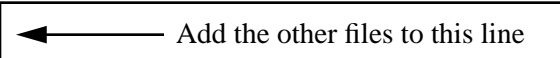
1. Create a separate subdirectory for the model. We recommend that all OptdesX files associated with a particular model be in a separate directory.
2. Copy the files `anasubF.f` and `Makefile`, usually found in the `OptdesX/user` directory, into your local directory.
3. Using `anasubF.f` as a template, write your computer model. (Section 4.2.3 shows an example Fortran analysis model.) Do not change the name of the `anasubF.f` file.
4. Type `make`. This will cause `Makefile` to be executed. The `Makefile` will compile your model and, assuming the compilation is successful, link it with the OptdesX routines, creating an executable called “`OptdesX`” in your area. The `Makefile` infers that your model is Fortran-based from the existence of the `anasubF.f` (or `anasubF.o`) files--do not rename these files. It will also link in dummy C routines in the file `anadumC.o`.
5. OptdesX is ready to execute. Type `OptdesX`.

If you need to link in other files with your model, you will need to edit the `Makefile` to include these files. Suppose for example you need to link in two other files called `otherfile1.f` and `otherfile2.f`. The line of the `Makefile` which would be edited is shown below.

```

#                               OptdesX Makefile
#
# The output from this makefile (assuming all goes well) is a program
# named 'OptdesX'
#
OTHER_OBJS = otherfile1.o otherfile2.o
#
#####
#
# Don't mess with anything in the following section!
#
# Any dependency lists for OTHER_OBJS should be added at the
# BOTTOM of this file (where indicated).
#
#####
#

```



When you execute the Makefile (by typing `make`), the additional files will automatically be compiled if necessary and then linked in with OptdesX.

You can also run OptdesX with a stand-alone model that is not called as a subroutine from OptdesX. Section 4.4 describes how to do this.

4.2.2 Fortran Analysis Models--DEC VMS Systems

1. Create a separate subdirectory for the model. We recommend that all OptdesX files associated with a particular model be in a separate directory.
2. Copy the files `ANASUBF.FOR` and `OPTBUILD.COM`, usually found in the `OPTDESX.USER` directory, into your local directory.
3. Using `ANASUBF.FOR` as a template, write your computer model. (Section 4.2.3 shows an example Fortran analysis model.) Do not change the name of the `ANASUBF.FOR` file.
4. Compile your model using your Fortran compiler.
5. Type `@OPTBUILD`. This will cause `OPTBUILD` to be executed. It will link your model with the OptdesX routines, creating an executable `OPTDESX.EXE` in your area. The command file infers that your model is Fortran-based from the existence of the `ANASUBF.OBJ` file--do not rename this file. It will also link in dummy C routines in the file `ANADUMC.OBJ`, since OptdesX is set up to call routines in either C or Fortran.
6. OptdesX is ready to execute. Type `RUN OPTDESX`.

If you need to link in other files with your model, you will need to edit `OPTBUILD.COM` to include these files. Suppose for example you need to link in two other files called `OTHERFILE1.OBJ` and `OTHERFILE2.OBJ`. The line of `OPTBUILD.COM` which would be edited is shown below.

```
$ !
$ ! Build OptdesX for VAX/VMS
$ !
$ !   Optdesx will be linked to your model and additional libraries as
$ !   specified in the variables OTHER_OBJS below:
$ !
$ OTHER_OBJS = ",OTHERFILE1.OBJ,OTHERFILE2.OBJ"
$ !
$ ! NOTE:  Before beginning, the system administrator should define the
$ !         system logicals OPTDESLIB and OPTDESHELP.
$ !         Otherwise OPTBUILD will not work and OptdesX will not find
$ !         necessary help files.
$ !
```

← Add other files here. Note that they are in quotes and start with a comma

4.2.3 Example Fortran ANASUB routines for the Twobar Truss

In the tutorial in Section 3 we illustrated the use of the conventional interface in Fortran for the twobar truss; for completeness we will present it again here.

```

=====
c...subroutine ANAPRE
c    pre-processing routine
c-----
    subroutine ANAPRE(modelN)
    character*17 modelN

    modelN = 'Twobar Truss'

    return
    end

=====
c...subroutine ANAFUN
c    Analysis routine for Twobar Truss
c-----
    subroutine ANAFUN
    double precision hght,wdth,diam,thik,dens,modu,load,leng,
    &pi,area,iovera,wght, strs,buck,defl

c...input scalar AV values
    call avdsca(hght,'height')
    call avdsca(wdth,'width')
    call avdsca(diam,'diameter')
    call avdsca(thik,'thickness')
    call avdsca(dens,'density')
    call avdsca(modu,'modulus')
    call avdsca(load,'load')

c...intermediate constants
    leng = sqrt((wdth/2.d0)**2+hght**2)
    pi = 3.1415927d0
    area = pi*diam*thik
    iovera = (diam**2+thik**2)/8.d0

C...compute functions
    wght = 2.d0*dens*area*leng
    strs = load*leng/2.d0/area/hght
    buck = (pi**2*modu*iovera/leng**2)
    defl = load*leng**3/2.d0/modu/area/hght**2

c...output scalar AF values
    call afdsca(wght,'weight')
    call afdsca(strs,'stress')
    call afdsca(buck,'buckling')
    call afdsca(defl,'deflection')

    return
    end

=====
c...subroutine ANAPOS
c    post processing routine
c-----

```

← Anapre is used here only to give the model a name. The model name is limited to 16 characters.

← The analysis variables are passed in from OptdesX through these subroutine calls. A variable name is limited to 16 characters.

← The analysis functions are passed back to OptdesX through these subroutine calls

```
subroutine ANAPOS
```

```
  return
```

```
end
```

c-----

At the beginning of ANAFUN we have a number of calls to a routine “avdsca,” such as:

```
call avdsca(hght, 'height')
```

This call has two arguments: the analysis variable value, which is passed in from OptdesX, and the analysis variable name. The name is limited to 16 characters for analysis variables (15 characters for design variables). Do not use leading or trailing blanks as part of the name. You can however, have blanks inside the name as long as the name begins and ends with a non-blank character. For example “Inside Diameter” is a valid name.

The acronym “avdsca” stands for “analysis variable double scalar.” In the current release of OptdesX, all variables and functions must be of this type. In the future we anticipate supporting other data types such as integers, vectors and matrices.

The function values are passed back in calls such as,

```
call afdsc(wght, 'weight')
```

The same restrictions that apply to variable names apply to function names: they should not contain leading or trailing blanks and they are limited to 16 characters for analysis functions (15 characters for design functions).

4.3 The Conventional Interface in C

The analysis model is coded in C in a function called “anafunC”. As with Fortran, there are also pre and post-processing routines, “anapreC” and “anaposC,” which collectively are referred to as the “anasubC” routines. The steps to link a C model to OptdesX are as follows.

4.3.1 C Analysis Models--UNIX Systems


1. Create a separate subdirectory for the model. We recommend that all OptdesX files associated with a particular model be in a separate directory.
2. Copy the files `anasubC.c` and `Makefile`, usually found in the `OptdesX/user` directory, into your local directory.
3. Using `anasubC.c` as a template, write your computer model. Do not change the name of the `anasubC.c` file.
4. Type `make`. This will cause `Makefile` to be executed. The `Makefile` will compile your model, and, assuming the compilation is successful, link it with the OptdesX routines, creating an executable called “OptdesX” in your area. The `Makefile` infers that your model is C-based from the existence of the `anasubC.c` (or `anasubC.o`) files--do not rename these files. It will also link in a dummy Fortran routine called `anadumF.o`.

5. OptdesX is ready to execute. Type OptdesX.

If you need to link in other files with your model, you will need to edit the Makefile to include these routines. Suppose for example you need to link in two other files called otherfile1.c and otherfile2.c. The lines of the Makefile which would be edited are shown below.



```

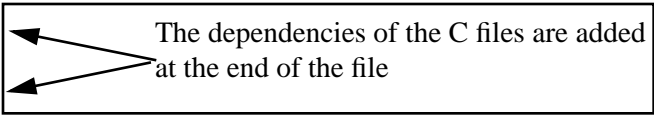
#                               OptdesX Makefile
#
# The output from this makefile (assuming all goes well) is a program
# named 'OptdesX'
#

OTHER_OBJS = otherfile1.o otherfile2.o  Add the other files to this line

#####
#
# Don't mess with anything in the following section!
#
# Any dependency lists for OTHER_OBJS should be added at the
# BOTTOM of this file (where indicated).
#
#####
#
#
etc.
etc.
etc.
# Dependency and command list for anafunF
#
anafunF.o:
    $(F77) -c $(F77FLAGS) anafunF.f

#####
#
# Add dependency lists for the OTHER_OBJS files below this header.
#
#####

otherfile1.o:    otherfile1.h 
otherfile2.o:    otherfile2.h 



```

When you execute the Makefile (by typing make), the additional files will automatically be compiled if necessary and then linked in with OptdesX.

You can also run OptdesX with a stand-alone model that is not called as a subroutine from OptdesX. Section 4.4 describes how to do this.

4.3.2 C Analysis Models--DEC VMS Systems

1. Create a separate subdirectory for the model. We recommend that all OptdesX files associated with a particular model be in a separate directory.
 2. Copy the files ANASUBC.C and OPTBUILD.COM, usually found in the OPTDESX.USER directory, into your local directory.
 3. Using ANASUBC.C as a template, write your computer model. (Section 4.3.3 shows an example C analysis model.) Do not change the name of the ANASUBC.C file.
 4. Compile your model with the C compiler. Note that the template file, ANASUBC.C, has the path to the ANASUBC.H include file "built in" through the use of a system logical.
 5. Type @OPTBUILD. This will cause OPTBUILD to be executed. It will link your model and the OptdesX routines, creating an executable OPTDESX.EXE in your area. The command file infers that your model is C-based from the existence of the ANAFUNC.OBJ file--do not rename this file. It will also link in a dummy Fortran routine called ANADUMF.OBJ.
5. OptdesX is ready to execute. Type RUN OPTDESX.

If you need to link in other files with your model, you will need to edit OPTBUILD.COM to include these files. Suppose for example you need to link in two other files called OTHERFILE1.OBJ and OTHERFILE2.OBJ. The line of OPTBUILD.COM which would be edited is shown below.

```
$ !
$ ! Build OptdesX for VAX/VMS
$ !
$ !   Optdesx will be linked to your model and additional libraries as
$ !   specified in the variables OTHER_OBJS below:
$ !
$ OTHER_OBJS = ",OTHERFILE1.OBJ,OTHERFILE2.OBJ"
$ !
$ ! NOTE:  Before beginning, the system administrator should define the
$ !         system logicals OPTDESLIB and OPTDESHELP.
$ !         Otherwise OPTBUILD will not work and OptdesX will not find
$ !         necessary help files.
$ !
```

← Add other files here. Note that they are in quotes and start with a comma

4.3.3 Example C ANASUB routines for the Twobar Truss.

A C "template" routine (anasubC.c) is provided for your convenience in making a C OptdesX model. As an illustrative example, the C version of the ANASUB routines for twobar truss follows:

```
/*=====*/
/* include files */
/*-----*/
#include <math.h>
#include "anasubC.h"
```


Linking Analysis Models to OptdesX 4-8

```
#define PI 3.1415927

/*=====*/
/*...function anapreC*/
/*      pre-processing function */
/*-----*/
void anapreC(char *modelName)
{
    strcpy(modelName,"Twobar Truss");
}

/*=====*/
/*...function anafunC*/
/*      Analysis Function for Twobar Truss */
/*-----*/
void anafunC(void)
{
    double hght, wdth, diam, thik, dens, modu, load, leng, area, iOverA,
           wght, strs, buck, defl;

    /* input scalar AV values */
    avdscaC(&hght,"height");
    avdscaC(&wdth,"width");
    avdscaC(&diam,"diameter");
    avdscaC(&thik,"thickness");
    avdscaC(&dens,"density");
    avdscaC(&modu,"modulus");
    avdscaC(&load,"load");

    /* intermediate constants */
    leng = sqrt(wdth * wdth / 4. + hght * hght);
    area = PI * diam * thik;
    iOverA = (diam * diam + thik * thik) / 8.;

    /* compute functions */
    wght = 2. * dens * area * leng;
    strs = load * leng / 2. / area / hght;
    buck = (PI * PI * modu * iOverA ) / (leng * leng);
    defl = load * pow(leng,3.) / modu / area / (hght * hght);

    /* output scalar AF values */
    afdscaC(wght,"weight");
    afdscaC(strs,"stress");
    afdscaC(buck,"buckling");
    afdscaC(defl,"deflection");
}

/*=====*/
/*...function anaposC*/
/*      post processing function */
/*-----*/
void anaposC(void)
{
}
```

Anapre is used here only to give the model a name. The model name is limited to 16 characters.

The analysis variables are passed in through these function calls. Note that you must pass the address of the variable.

The analysis functions are passed back through these subroutine calls

Please note the following restrictions on variable and function names: they must be limited to 16 characters (15 characters for design variables and functions) and they should not contain leading or trailing blanks.

4.4 The Stand-Alone Interface in Fortran

4.4.1 Introduction

Sometimes it is either undesirable or impossible to code the analysis model as a subroutine callable by OptdesX. For example, the model may be written using commercial software for which source code is unavailable. In these instances can OptdesX still be used? The answer is an unqualified yes! To link OptdesX to a stand-alone program, you use ANAFUN as a “driver” for the stand-alone analysis model. When used as a driver, ANAFUN performs several tasks: 1) it extracts the analysis variables in the usual way, using calls to “avdsca,” 2) it writes the analysis variables into an input file that can be read by the stand-alone program, 3) it executes the stand-alone program using a “system command” (available in UNIX or DEC VMS) 4) it reads a data file written by the stand-alone program, and 5) it passes analysis function values back to OptdesX.

The key to the process is the system command. A system command temporarily suspends the execution of OptdesX and returns control to the operating system. At this point any command can be executed at the operating system level. Of course, the command we wish to execute is to run the stand-alone program. When this is finished, control is returned to OptdesX at the point where the system command was executed.

Note: The steps for linking OptdesX and creating an executable program are exactly the same as given in Section 4.2. These steps will not be repeated here. Only the contents of the ANAFUN routine are different, as illustrated here by example.

4.4.2 A Simple Example

We will illustrate this process for the twobar truss. ANAFUN will be used as a driver to run a stand-alone program called “Twobar.” A listing of the ANASUB routines follows.

```

C=====
c...subroutine ANAPRE
c      preprocessing routine
c-----
      subroutine ANAPRE(modelN)
      character*17 modelN

      modelN = 'Twobar Truss'
      return
      end
C=====
c...subroutine ANAFUN
c      Analysis routine for Twobar Truss
c-----
      subroutine ANAFUN
      double precision hght,wdth,diam,thik,dens,modu,load,leng,

```

```
&pi, area, iovera, wght, strs, buck, defl
```

```
c...input scalar AV values
  call avdsca(hght, 'height')
  call avdsca(wdth, 'width')
  call avdsca(diam, 'diameter')
  call avdsca(thik, 'thickness')
  call avdsca(dens, 'density')
  call avdsca(modu, 'modulus')
  call avdsca(load, 'load')
```



← The analysis variables are extracted from OptdesX

```
c...write AV values to a file
  open(unit=1, file='var.d', status='unknown')
  write(1, *) hght
  write(1, *) wdth
  write(1, *) diam
  write(1, *) thik
  write(1, *) dens
  write(1, *) modu
  write(1, *) load
  close(1)
```



← The analysis variables are written to a data file

```
c...call executable to compute scalar AF values
```

```
  call system('twoobar')
```

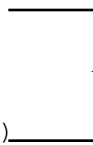
← The stand-alone program is executed. (This example is for UNIX.)

```
c...read AF values from a file
  open(unit=1, file='func.d', status='unknown')
  read(1, *) wght
  read(1, *) strs
  read(1, *) buck
  read(1, *) defl
  close(1)
```



← The analysis functions, which have been written to a data file by the stand-alone program, are read from the data file.

```
c...output scalar AF values
  call afdzca(wght, 'weight')
  call afdzca(strs, 'stress',)
  call afdzca(buck, 'buckling')
  call afdzca(defl, 'deflection')
```



← The analysis functions are sent back to OptdesX

```
  return
end
```

```
=====
c...subroutine ANAPOS
c  postprocessing routine
c-----
  subroutine ANAPOS
  return
  end
```

A listing of the stand-alone program follows:

```
=====
```

```

c...program TWOBAR
c      stand-alone program for the twobar truss
c-----
      program TWOBAR
      double precision hght,wdth,diam,thik,dens,modu,load,leng,
      &pi,area,iovera,wght, strs,buck,defl
c...read AV values from a file
      open(unit=1,file='var.d',status='unknown')
      read(1,*)hght
      read(1,*)wdth
      read(1,*)diam
      read(1,*)thik
      read(1,*)dens
      read(1,*)modu
      read(1,*)load
      close(1)

c...compute scalar AF values

c...intermediate constants
      leng = sqrt((wdth/2.d0)**2+hght**2)
      pi = 3.1415927d0
      area = pi*diam*thik
      iovera = (diam**2+thik**2)/8.d0

c...compute functions
      wght = 2.d0*dens*area*leng
      strs = load*leng/2.d0/area/hght
      buck = (pi**2*modu*iovera/leng**2)
      defl = load*leng**3/2.d0/modu/area/hght**2

c...write AF values to a file
      open(unit=1,file='func.d',status='unknown')
      write(1,*)wght
      write(1,*)strs
      write(1,*)buck
      write(1,*)defl
      close(1)

      stop
      end

```

The file written by ANASUB is read.

The analysis functions are written to an output file.

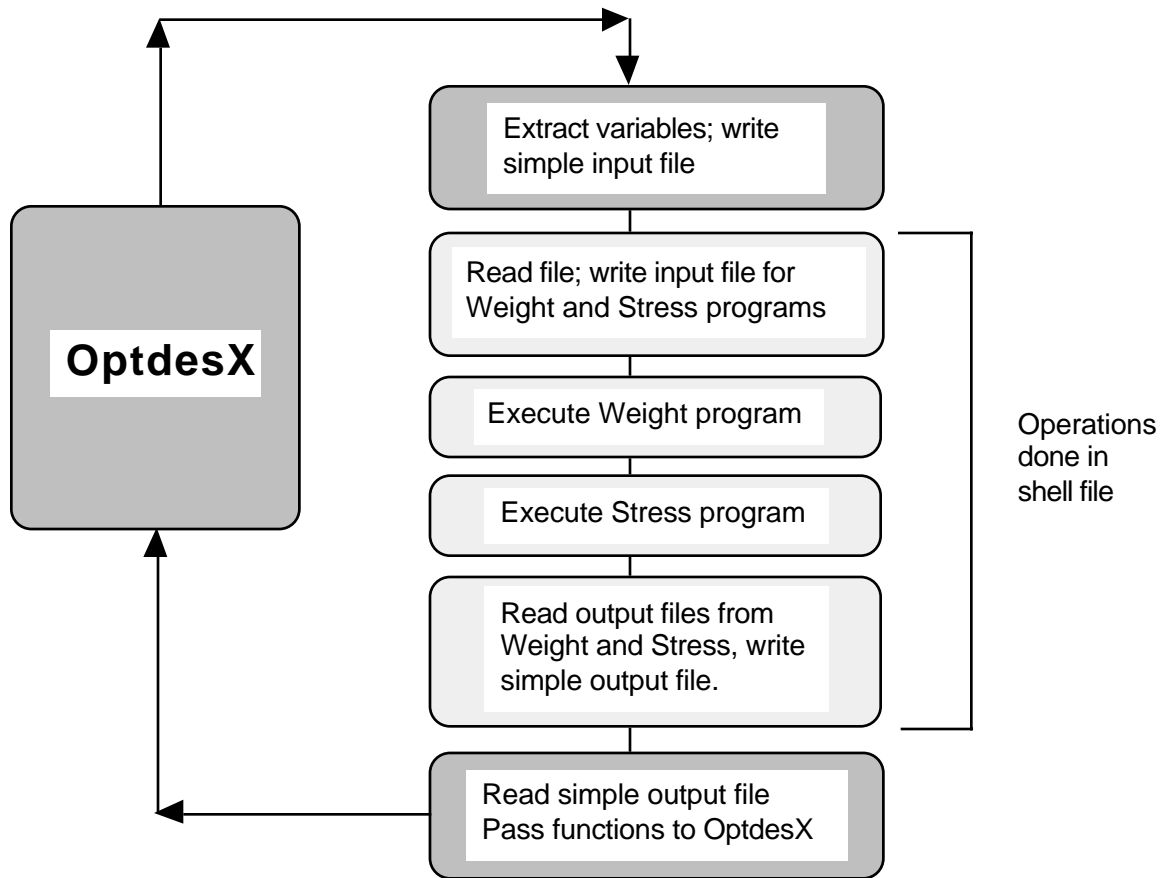
What will happen when an algorithm is executed? Every time the analysis model is called, an input data file will be written, the model will be executed by the operating system, the model will read the input file and calculate the analysis functions, the model will write an output file, and this file will be read by ANAFUN. There is significant overhead associated with all of this I/O, and the result is that the algorithm will run more slowly.

4.4.3 A More Complicated Example

In the previous example we had a very simple case: the stand-alone program reads the variables in a free-format, and it writes the functions in free format. What if the stand-alone program has a very lengthy or complicated input or output file? This is not unusual--commercial

software, which is written to be general, often requires extensive input and writes extensive output. Or, as a related situation, what if we have a whole chain of analysis programs we need to execute?

The key to handling this type of situation is to develop a shell file (in UNIX) or a command file (in VMS) that contains all of the necessary commands to execute the entire analysis chain. We will show how this can be done. For this next example, we will write a simple input file from inside ANAFUN; we will then execute a pre-processor that will read this file and write two input files: one for an analysis program that only computes the weight of the truss and one for a program that computes the stress, buckling stress and deflection; we then compute the weight; we compute the stress, buckling stress and deflection; we execute a post-processor that reads files from the weight and stress programs and extracts the function values, writing a simple file that can be read by ANAFUN; finally control is returned to ANAFUN where the function values are read and passed back to OptdesX. This process is illustrated in the diagram.



Listings of the programs needed to execute this chain are given on the following pages.

```

-----
c...subroutine ANAFUN
c      Stand-Alone interface for chain of analysis programs
-----
      subroutine anafun
  
```

```
double precision hght,wdth,diam,thik,dens,modu,load,
&wght, strs,buck,defl
```

c...input scalar AV values from OptdesX

```
call avdsca(hght,'height')
call avdsca(wdth,'width')
call avdsca(diam,'diameter')
call avdsca(thik,'thickness')
call avdsca(dens,'density')
call avdsca(modu,'modulus')
call avdsca(load,'load')
```

The analysis variables are extracted from OptdesX

c...write values to a file

```
open(file='Optin',unit=25,status='unknown')
write(25,100) hght
write(25,100) wdth
write(25,100) diam
write(25,100) thik
write(25,100) dens
write(25,100) modu
write(25,100) load
close(25)
```

The analysis variables are written to a data file

A shell file is executed. The shell file contains the commands that execute the chain of analysis programs. The contents of this file are given on the next page. (This example is for Unix.)

c...execute chain of analysis programs

```
call system('Chain.sh')
```

c...read AF values from file

```
open(file='Optout',unit=25,status='old')
read(25,*) wght
read(25,*) strs
read(25,*) buck
read(25,*) defl
close(25)
```

The analysis functions, which have been written to a file by a post processor, are read from the data file.

c...send scalar AF values to OptdesX

```
call afdzca(wght,'weight')
call afdzca(strs,'stress')
call afdzca(buck,'buckling')
call afdzca(defl,'deflection')
```

The analysis functions are sent back to OptdesX

```
return
```

```
100 format(1x,g14.7)
end
```

```
=====
c...subroutine ANAPOS
c    postprocessing routine
c-----
```

```
subroutine anapos()
return
end
```

The contents of the file Chain.sh consists of the four commands which execute the chain of analysis programs:

```
prepro
weight
stress
postpro
```

Chain.sh must have its protection set such that it is executable.

Listings of the pre-processor, the program Weight, the program Stress, and the post-processor follow.

```

=====
c...program preprocess
c      writes input files for weight and stress
c-----
      program weight
      double precision hght,wdth,diam,thik,dens,modu,load

c...read in file from OptdesX
      open(file='Optin',unit=20,status='old')
      read(20,*) hght,wdth,diam,thik,dens,modu,load

c...write out values for weight program
      open(file='Win',unit=22,status='unknown')
      write(22,100) hght,wdth,diam,thik,dens

c...write out values for stress program
      open(file='Sin',unit=25,status='unknown')
      write(25,100) hght,wdth,diam,thik,modu,load

      close(20)
      close(22)
      close(25)

100  format(5(1x,g14.7))
      stop
      end

=====
c...program weight
c      calculates weight for twobar truss
c-----
      program weight
      double precision hght,wdth,diam,thik,dens,pi

c...read in input values
      open(file='Win',unit=25,status='old')
      read(25,*) hght,wdth,diam,thik,dens
      close(25)

c...write message

```

```

        write(*,*) 'Executing Weight'

c...intermediate constants
    leng = sqrt((width/2.d0)**2+hght**2)
    pi = 3.1415927d0
    area = pi*diam*thik

C...compute weight, write to file
    wght = 2.d0*dens*area*leng
    open(file='Wout',unit=25,status='unknown')
    write(25,100) wght
    close(25)

100  format(g14.7)
      stop
      end

C=====
c...program stress
c      calculates stress for twobar truss
c-----
      program stress
      double precision hght,width,diam,thik,modu,load,leng,
&pi,area,iovera, strs,buck,defl

c...read in data
    open(file='Sin',unit=25,status='old')
    read(25,*) hght,width,diam,thik,modu,load
    close(25)

c...write message
    write(*,*) 'Executing Stress'

c...intermediate constants
    leng = sqrt((width/2.d0)**2+hght**2)
    pi = 3.1415927d0
    area = pi*diam*thik
    iovera = (diam**2+thik**2)/8.d0

c...compute functions
    strs = load*leng/2.d0/area/hght
    buck = (pi**2*modu*iovera/leng**2)
    defl = (load*leng**3)/(2.d0*modu*area*hght**2)

    open(file='Sout',unit=25,status='unknown')
    write(25,100) strs,buck,defl
    close(25)

100  format(3(1x,g14.7))
      stop
      end

C=====
c...program postprocess

```



```

c      reads output files of weight, stress; writes file for OptdesX
c-----
      program postpr
      double precision wght, strs, buck, defl

c...read in file from Weight
      open(file='Wout',unit=20,status='old')
      read(20,*) wght

c...read in file from Stress
      open(file='Sout',unit=22,status='old')
      read(22,*) strs,buck,defl

c...write file for OptdesX to read
      open(file='Optout',unit=25,status='unknown')
      write(25,100) wght
      write(25,100) strs
      write(25,100) buck
      write(25,100) defl

      close(20)
      close(22)
      close(25)

100  format(5(1x,g14.7))
      stop
      end

```

4.4.4 The System Command for DEC VMS

The examples in the previous sections were for UNIX. We list here the system command for DEC VMS for reference. To execute a program called “two-bar” from inside a FORTRAN program, the appropriate statement would be,

```
call lib$spawn('run two-bar')
```

4.5 The Stand-Alone Interface in C

In this Section we present a C version of the Stand-Alone interface. As with the Fortran version, we will use anafunC to extract the analysis variables from OptdesX, write the variables into a file, execute the truss program as a stand-alone program, read a data file written by the truss program, and then pass the function values back to OptdesX. Please refer to the introduction to Section 4.4 for additional information.

Note: The steps for linking OptdesX and creating an executable program are exactly the same as given in Section 4.3. These steps will not be repeated here. Only the contents of the anafunC routine are different, as illustrated here by example.

```

/*-----*/
/* include files */
/*-----*/
#include <stdio.h>
#include <math.h>

```

```

#include "anasubC.h"

#define PI 3.1415927

/*=====*/
/*...function anapreC*/
/*      pre-processing function */
/*-----*/
void anapreC(char *modelName)
{
    strcpy(modelName, "Twobar Truss");
}

/*=====*/
/*...function anafunC*/
/*      Analysis function */
/*-----*/
void anafunC(void)
{
    double hght, wdth, diam, thik, dens, modu, load, leng, area, iOverA,
           wght, strs, buck, defl;
    FILE *in, *out;

    /* input scalar AV values */
    avdsca (&hght,"height");
    avdsca (&wdth,"width");
    avdsca (&diam,"diameter");
    avdsca (&thik,"thickness");
    avdsca (&dens,"density");
    avdsca (&modu,"modulus");
    avdsca (&load,"load");

    /*write AV values to a file*/
    out=fopen("var.d","w");
    fprintf(out,"%#22.13lg\n",hght);
    fprintf(out,"%#22.13lg\n",wdth);
    fprintf(out,"%#22.13lg\n",diam);
    fprintf(out,"%#22.13lg\n",thik);
    fprintf(out,"%#22.13lg\n",dens);
    fprintf(out,"%#22.13lg\n",modu);
    fprintf(out,"%#22.13lg\n",load);
    fclose(out);

    /*call executable to comput scale AF values*/
    system("twobar");

    /*read AF values from a file*/
    in=fopen("func.d","r");
    fscanf(in,"%lf",&wght);
    fscanf(in,"%lf",&strs);
    fscanf(in,"%lf",&buck);
    fscanf(in,"%lf",&defl);

```

The analysis variables are extracted from OptdesX

The analysis variables are written to a data file

The stand-alone program is executed. (This example is for UNIX.)

The analysis functions, which have been written to a data file by the stand-alone program, are read from the data file.

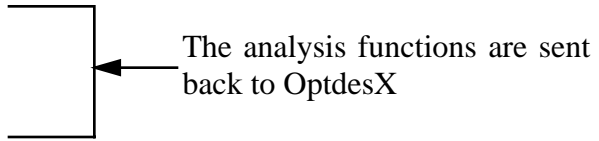
```

    fclose(in);

    /* output scalar AF values */
    afdzca (wght,"weight");
    afdzca (strs,"stress");
    afdzca (buck,"buckling");
    afdzca (defl,"deflection");
}

/*-----*/
/*...function anaposC*/
/*      post-processing function */
/*-----*/
void anaposC(void)
{
}

```



A listing of the stand-alone truss analysis program follows.

```

/*-----*/
/* include files for stand-alone program */
/*-----*/
#include <stdio.h>
#include <math.h>

#define PI 3.1415927

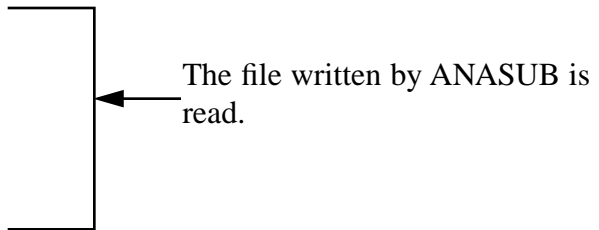
/*-----*/
/* Stand-Alone Program for Twobar Truss */
/*-----*/
void main()
{
    double hght, wght, diam, thik, dens, modu, load, leng, area, iOverA,
           wght, strs, buck, defl;
    FILE *in, *out;

    /* read AV values */
    in=fopen("var.d","r");
    fscanf(in,"%lf",&hght);
    fscanf(in,"%lf",&wght);
    fscanf(in,"%lf",&diam);
    fscanf(in,"%lf",&thik);
    fscanf(in,"%lf",&dens);
    fscanf(in,"%lf",&modu);
    fscanf(in,"%lf",&load);
    fclose(in);
    /* compute scalar AF values */

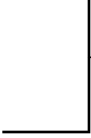
    /* intermediate constants */
    leng = sqrt(wght * wght / 4. + hght * hght);
    area = PI * diam * thik;
    iOverA = (diam * diam + thik * thik) / 8.;

    /* compute functions */
    wght = 2. * dens * area * leng;
}

```



```
strs = load * leng / 2. / area / hght;  
buck = (PI * PI * modu * iOverA ) / (leng * leng);  
defl = load * pow(leng,3.) / modu / area / (hght * hght);  
  
/* write AF values to a file */  
out=fopen("func.d","w");  
fprintf(out,"##22.13lg\n",wght);  
fprintf(out,"##22.13lg\n",strs);  
fprintf(out,"##22.13lg\n",buck);  
fprintf(out,"##22.13lg\n",defl);  
fclose(out);  
}
```



The analysis functions are written to an output file.

(This page intentionally left blank.)

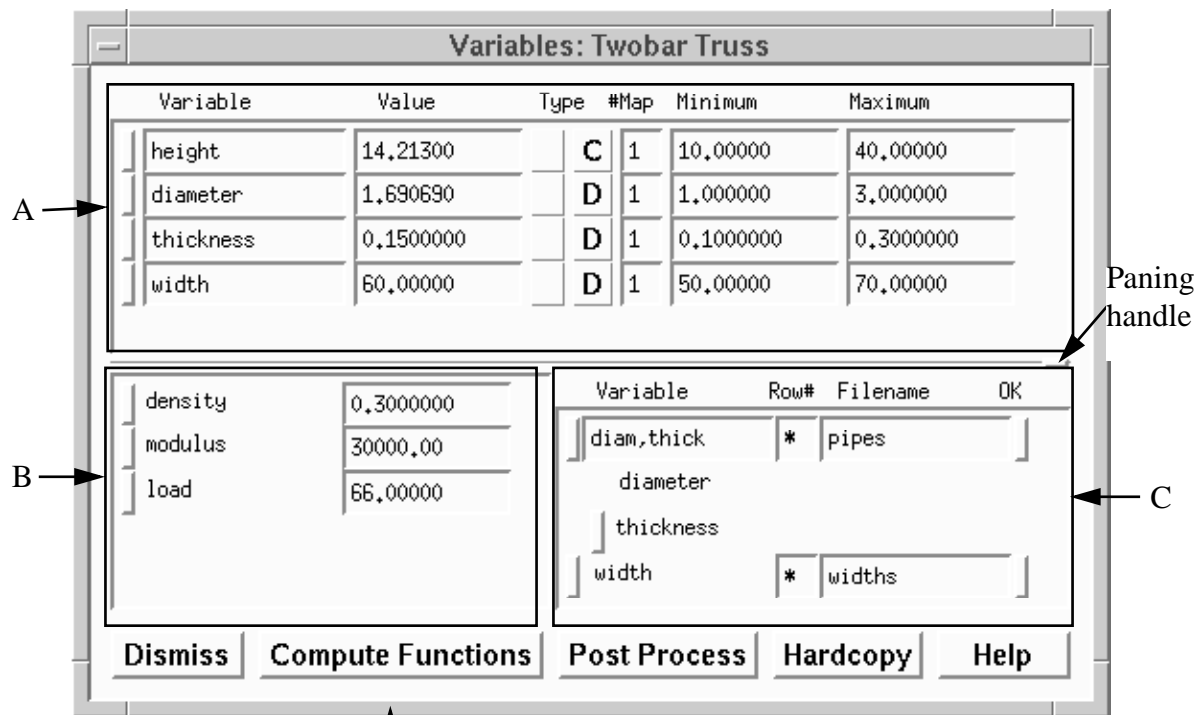
5 Variables Window Reference

5.1 Overview

The Variables window is used to select optimization variables, and display and change variable values.

The Variables window is divided into three parts: the lower left section for displaying unmapped analysis variables (box B in the diagram below), the upper section for displaying design variables (box A), and the lower right section for displaying discrete variables (box C).

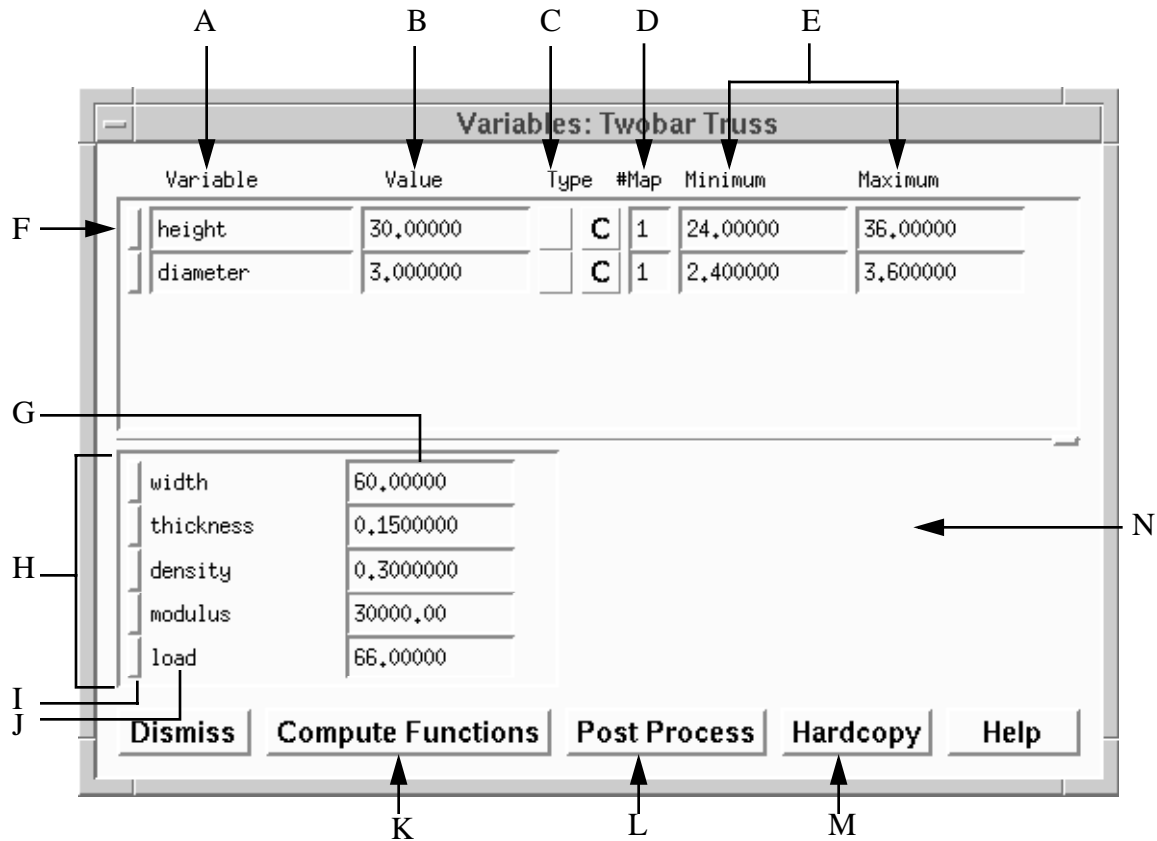
The window is divided in half by a thin dividing line that ends at the right side in a thicker “handle.” This handle panes the window: when the cursor is placed on it and dragged, the cursor form changes to a “+,” and the relative sizes of the upper and lower windows are changed. To show more variables than panning allows, the window can also be made longer by placing the cursor on the top or bottom border and dragging.



Placing the cursor on the top or bottom border and dragging will make the window longer, allowing you to see more variables.

5.2 Continuous Variables

5.2.1 Reference Diagram



A - Design Variable Name text field. This can be edited as indicated by the frame. Names are normally changed only when the “#Map” field is made greater than one.

B - Design Variable value field. This field can be edited.

C - Type pushbuttons which indicate that a variable is at its upper or lower bound (left side) or that a variable is continuous or discrete (right side). Bound icons are shown below. A “C” indicates a continuous variable and a “D” indicates a discrete variable.



Bound icon indicating the variable is at its upper bound.



Bound icon indicating the variable is at its lower bound

D - # Map value field which displays the number of analysis variables mapped to one design variable.

E - Minimum and Maximum value fields. Default values are +/- 20% of the current value.

F - Unmapping pushbutton. Pushing this button deletes the design variable; it disappears from the list in the upper half of the window and reappears in the list for unmapped analysis variables.

- G - Analysis Variable value field. This value remains constant during optimization. The value can be edited as indicated by its frame.
- H - List of Unmapped Analysis variables.
- I - Mapping button used to map an analysis variable to a design variable.
- J - Analysis Variable Name text field. This text field can not be edited.
- K - Compute Functions pushbutton used to call ANAFUN to calculate the analysis functions.
- L - Post Processing pushbutton used to call ANAPOS.
- M -Hardcopy pushbutton used to obtain hardcopy of the setup information.
- N- Discrete variable information space. Discrete variables are discussed further in Section 5.3.

5.2.2 Unmapped Analysis Variables

The lower left window displays the unmapped analysis variables. These are inputs to the analysis model that are not currently mapped to be optimization variables. The button to the left of the name (I in the reference diagram) is the Mapping button. When pressed, the variable is made a design variable; it will appear in the upper part of the window and be deleted from the unmapped analysis variable list. Values of the unmapped analysis variables can be changed by editing the value inside the frame; the corresponding function values are computed by pressing the “Compute Functions” button.

5.2.3 Design Variables

Design Variable Names and Values

The upper half of the window displays the design variables. Design variables are adjusted by the algorithms during optimization. The button to the left of the name (F in the reference diagram) is the Unmapping button. When pressed, the variable is deleted from the list of design variables and reappears in the unmapped variable list. Both the names and the values of the design variables may be edited by placing the cursor in the text field and changing the variable name/value character by character or by double clicking to select and replace the entire field. Usually a name is edited only when several analysis variables are mapped to one design variable--in this case a new unique name must be chosen.

At Bounds Icons

To the right of the value field are two buttons that are underneath the label “Type.” The button on the left (C in the reference diagram) displays the At Bounds Icon, which is a triangle pointing up for a variable at its upper bound or a triangle pointing down for a variable at its lower bound.

Continuous-Discrete button

The button to the right under type is the Continuous-Discrete button, marked with a “C” for the default state of continuous. Pressing this button makes a design variable discrete: the icon changes to a “D,” and the variable appears in a list in the lower right of the window where additional information is specified. This is a one-way button: variables are made discrete by pushing it, but they cannot be “unmade” by pushing it again--to “unmake” a discrete variable (i.e. make it continuous) you press the Umapping button in the discrete section (see below).

Map Value Field

Under the column “# Map” is a frame specifying the number of variables mapped to the design variable. When the default is changed from 1, additional fields open up below the design variable; in the first row the variable name is the same as the design variable, while the other rows are marked “NULL VARIABLE.” The next unmapped analysis variables that are made design variables will fill the Null variable slots.

When several analysis variables are mapped to one design variable, these analysis variables are all assigned the same value; during an optimization, these variables all change together. This is useful when there are variables in a design problem which, for reasons of symmetry, aesthetics or economics, should have the same value. Mapping several analysis variables to one design variable also reduces the size of the optimization problem, usually with a corresponding decrease in the computation required to reach a solution.

An example of a many-to-one mapping is shown in the figure below, where channel widths 1,2,3 have been mapped to one design variable--thus they will all have the same value. The same applies to fin widths 1,2,3.

This name must be changed so it doesn't conflict with this one.

Variable	Value	Type	#Map	Minimum	Maximum
channel width 1	0,5000000	C	3	0,4000000	0,6000000
channel width 1	0,5000000				
channel width 2	0,5000000				
channel width 3	0,5000000				
fin width 1	0,7500000	C	3	0,6000000	0,9000000
fin width 1	0,7500000				
fin width 2	0,7500000				
fin width 3	0,7500000				
channel width 4	0,5000000				
channel width 5	0,5000000				

Dismiss Compute Functions Post Process Hardcopy Help

All variable names must be unique, so when several analysis variables are mapped to a design variable, its name must be made unique to all the others.

To delete an analysis variable which is one of several mapped to a design variable, you press the unmapping button to the left of the name, as shown for Channel widths 2 and 3. The name “NULL VARIABLE” will then appear in these slots, to be filled by the next analysis variables made design variables. Note that the original analysis variable from which the design variable was created cannot be deleted, e.g. channel width 1. To delete this variable, you would have to delete the design variable and then begin the mapping process over again.

Minimum and Maximum value fields

In these fields you specify the minimum and maximum bounds for each design variable. The default values are +/- 20% of the current value. These bounds are considered hard limits by the algorithms, and variables will not be set outside these limits. If you accidentally try to set a design variable outside these limits, an error message is printed and the variable is reset to be at a limit.

Minimum and Maximum values are used by OptdesX for scaling of the design variables according to the formula:

$$dv_i = \frac{\overline{dv}_i - C_{1i}}{C_{2i}} \quad \text{where} \quad C_{1i} = \frac{\text{Max}_i + \text{Min}_i}{2}$$

$$C_{2i} = \frac{\text{Max}_i - \text{Min}_i}{2}$$

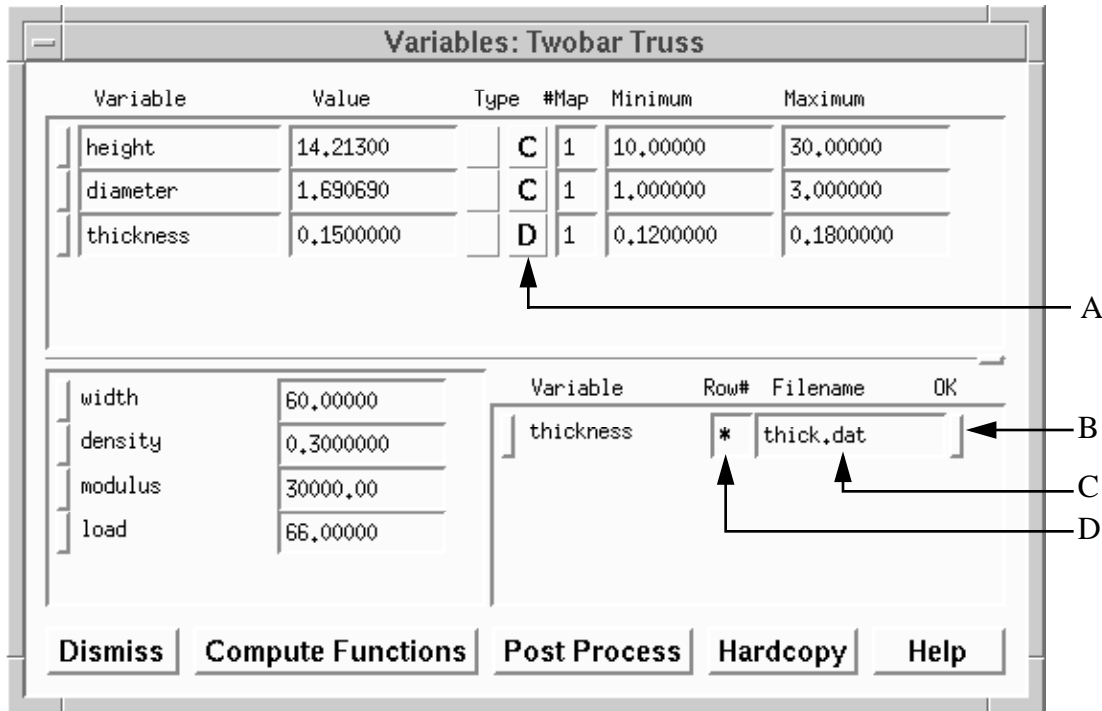
where dv_i is the unscaled value of design variable i , \overline{dv}_i is the scaled value, and Min_i and Max_i are the Minimum and Maximum values respectively.

Bounds should be as tight as possible without overly restricting the design problem. If you have variables that do not have upper and/or lower bounds, bounds should not be set to extremely large or extremely small values, since this will adversely affect scaling. If for example, you have a variable that does not have an upper bound, and a reasonable value for this variable is 10, then an upper bound of 50 would be appropriate, not 1.e10. If this variable happens to hit the upper bound, the bound can always be relaxed.

5.3 Discrete Variables

5.3.1 Discrete Reference Diagram

When variables are made discrete by pushing the Continuous-Discrete button, they appear in the lower right corner of the window. Fields are shown for Variable name, Row# and File Name. Additionally there is an Unmapping button to the left of the name field. These fields are shown in the reference diagram below.



- A - Continuous-Discrete pushbutton. The “D” indicates a discrete variable.
- B - OK pushbutton used to read in a data file containing the discrete values. The button highlights when the values are successfully read.
- C - Discrete File Name text field. The name is limited to 15 characters.
- D - Row # value field giving the row number of a discrete value in the file. An asterisk indicates the current value does not correspond to a discrete value in the file.

5.3.2 Discrete Window Operation

Filename

In the field under Filename, you specify the file which holds discrete values for that variable. When the file name is registered, either by typing a carriage return in the name field or pressing the “OK” button, OptdesX attempts to read the file. If successful, the OK button highlights.

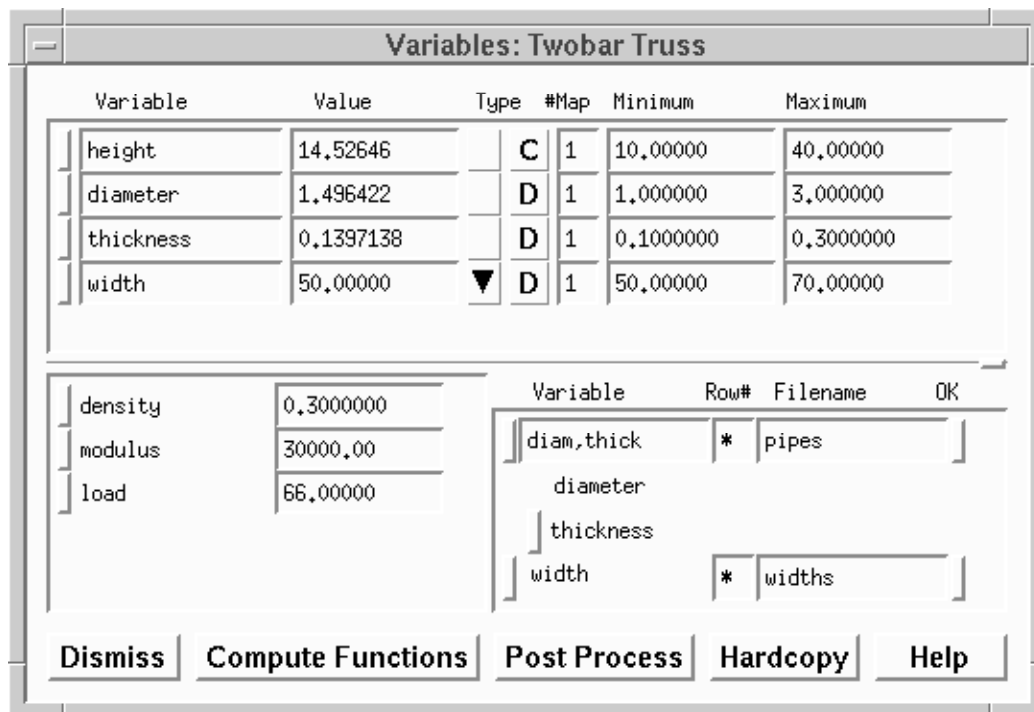
Row #Value Field

This represents the row number of a discrete value in a file. You may set a row number and the discrete variable will be set to the corresponding discrete value, which is shown in the design variable value field. Or, you may set the discrete variable value directly in the upper portion of the Variables window and, if this matches a discrete value in the file, the Row #

will change accordingly. An asterisk (*) indicates that the current variable value does not match any discrete values.

Related Discrete Variables

Related discrete variables often occur when you wish to include data from catalogs for standard size parts. For example, pipes come in standard sizes which make pipe diameter and pipe thickness discrete. Furthermore, these discrete variables are related: only certain thicknesses are available with certain diameters and vice-versa. OptdesX handles related discrete variables as a case of mapping several design variables to one discrete variable, where the discrete variable now represents a discrete combination of the variables mapped to it. If a discrete file has more than one column of data, OptdesX assumes that you wish to have related discrete variables. Additional rows open up underneath the current row, with the first row given the current row name and additional rows named "NULL VARIABLE." The next design variables that are made discrete fill these slots. Related discrete variables must be given a unique name. An example of related discrete variable is given below.



In the example, diameter and thickness have been mapped to one discrete variable, "diam,thick." The name of the discrete variable was edited to make it unique (its default name was diameter, which conflicts with the first design variable mapped to it)--thus it has a frame around it while the other discrete variable, width, does not. OptdesX knew that there were to be two variables mapped to "diam,thick" because the file "pipes" has two columns of data. The first column in "pipes" contains discrete values for diameter, and the second column contains discrete values for thickness. One row of data from this file sets values for both diameter and thickness.

Discrete variables may be deleted by pressing the Unmapping button to the left of the variable

name. As with other cases of many-to-one mappings, the original design variable from which the many-to-one discrete variable was created--diameter--cannot be deleted without deleting the discrete variable itself.

5.3.3 Discrete File Format

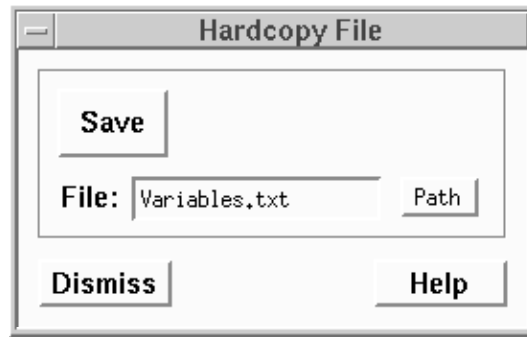
A listing of a discrete file is given for the file “pipes” used in the preceding example. The first line in the file gives the number of rows and columns; this is followed by the discrete data. The entries in the file need to be separated by “white space,” i.e., blanks or tabs but not commas. In this case we have 19 rows, where each row represents a discrete combination, and 2 columns, where the first column is the data for diameter and the second column is the data for thickness. A file that did not relate two variables together, as this one does, would have only one column of data.

```
19 2
3.0 .30
3.0 .28
3.0 .26
2.5 .28
2.5 .26
2.5 .24
2.5 .22
2.0 .24
2.0 .22
2.0 .20
2.0 .18
1.5 .20
1.5 .18
1.5 .16
1.5 .14
1.0 .16
1.0 .14
1.0 .12
1.0 .10
```

5.4 Compute Functions, Post Process, Hardcopy

The “Compute Functions” button calls the ANAFUN routines, calculating the analysis functions and displaying them in the Functions window. The “Post Process” button calls the ANAPOS routines; these are optional routines that you can use for whatever purpose you wish. A typical use would be to compute some additional information about the design that you would like to see periodically. Note that if you print information from ANAPOS it will not be displayed in an OptdesX window but in a terminal window.

The “Hardcopy” button opens a window to create a file that contains the optimization setup and variables and function values. This file can then be printed to obtain a hardcopy of this information. The hardcopy window is given below, and this is followed by an example listing of a Variables/Functions hardcopy file.



The file name in this window is limited to 20 characters (16 characters plus four characters for an extension).

An example listing of a hardcopy file for the Twobar truss follows:

===== Variables =====

Name	Value	T	Map	Min	Max
height	30.00000	C	1	10.00000	30.00000
diameter	3.000000	C	1	1.000000	3.000000

Name	Value	Name	Row	File
width	60.00000			
thickness	0.1500000			
density	0.3000000			
modulus	30000.00			
load	66.00000			

===== Functions =====

Name	Value	O	T	Map	Allowable	Indifference
weight	35.98735	v		1	40.00000	10.00000
stress	33.01160	<		1	100.0000	50.00000
stress-buckling	-152.5061	<		2	0.000000	-50.00000
Sum stress	33.01160					
- buckling	185.5177					
deflection	0.06602320	<		1	0.2500000	0.05000000

Name	Value	Name	Weight

(This page intentionally left blank.)

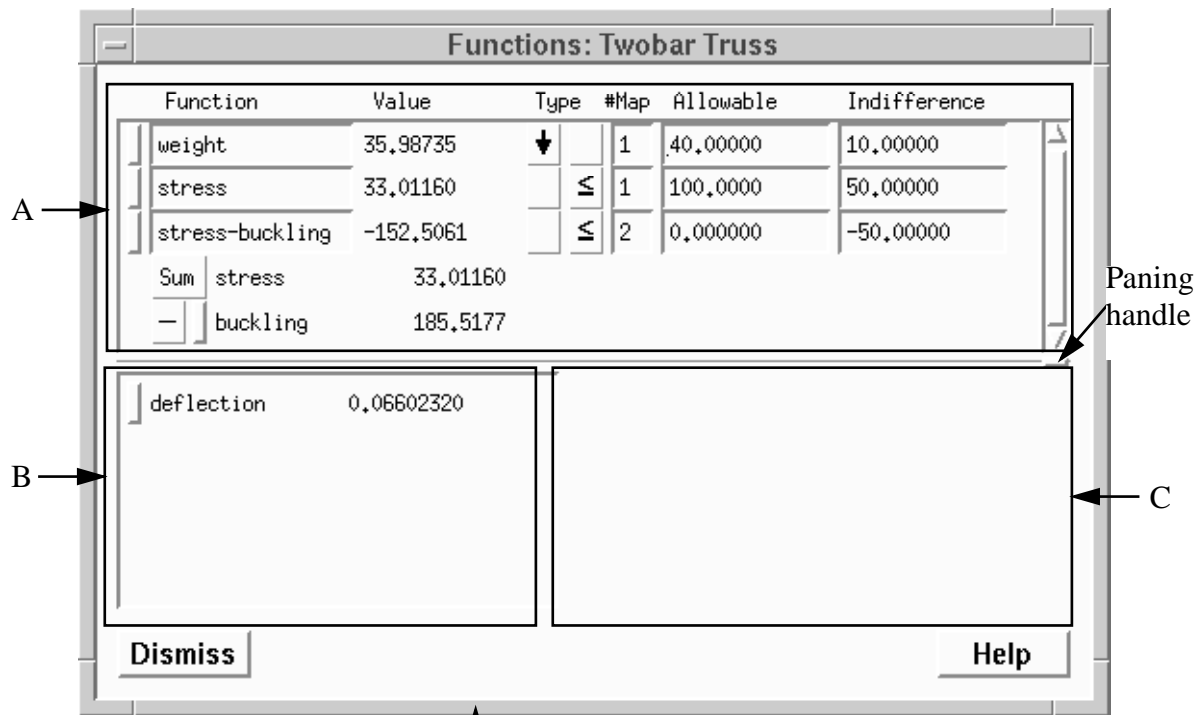
6 Functions Window Reference

6.1 Overview

The Functions window is used to select optimization functions and display function values.

The Functions window is divided into three parts: the lower left section for displaying unmapped analysis functions (box B in the diagram), the upper section for displaying design functions (box A), and the lower right section for displaying objective weighting coefficients for multiple objective problems.

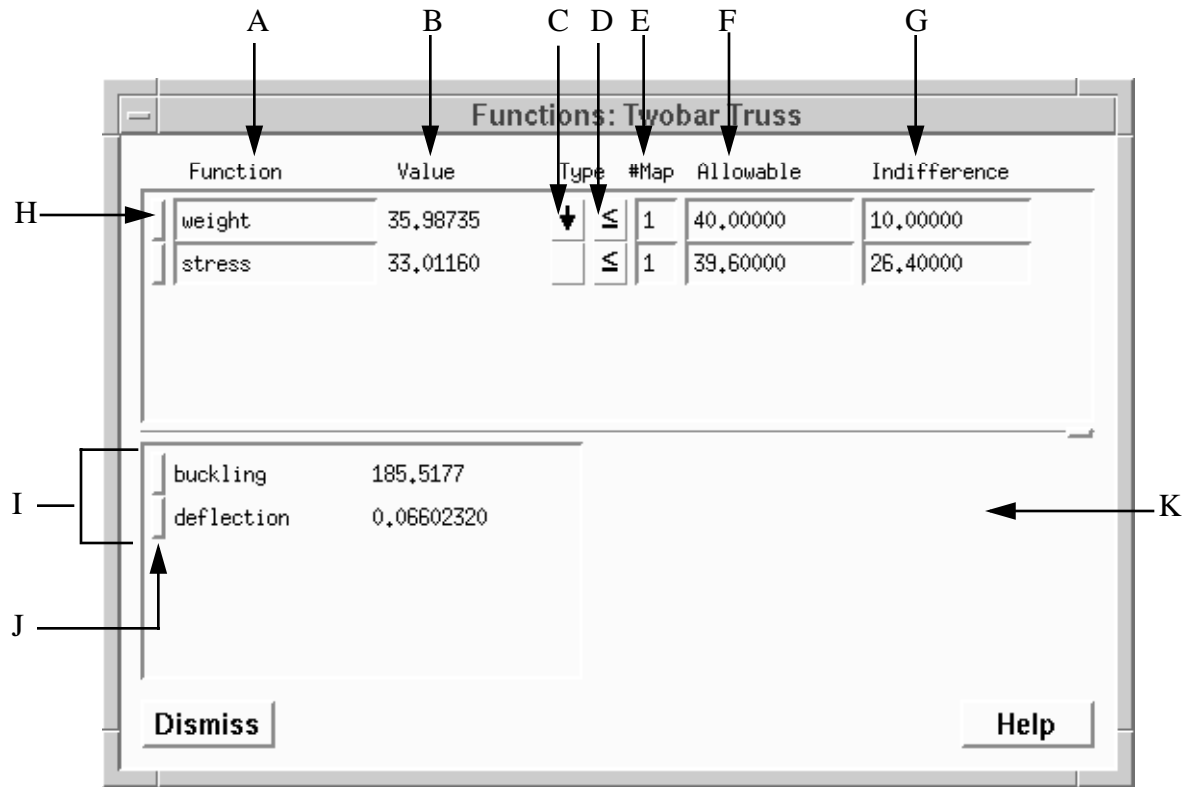
The window is divided in half by a thin dividing line that ends at the right side in a thicker “handle.” This handle panes the window: when the cursor is placed on it and dragged, the cursor form changes to a “+,” and the relative sizes of the upper and lower windows are changed. The window can also be made longer, to show more functions than panning allows, by placing the cursor on the top or bottom border and dragging.



Placing the cursor on the top or bottom border and dragging will make the window longer, allowing you to see more functions.

6.2 Single Objective Problems

6.2.1 Reference Diagram



- A - Design Function Name Text field. The name can be edited as indicated by its frame, although usually it is not edited unless #Map is greater than one.
- B - Design Function value fields. These values are calculated in ANAFUN.
- C - Objective button. The objective button is pressed until the appropriate icon is selected: downward arrow to minimize, upward arrow to maximize, and blank if not an objective. If an objective is at its indifference value its icon is boxed; if it is better than its indifference value it is highlighted.



Objective at Indifference Value



Objective better than Indifference Value

- D - Constraint button. Constraints can be blank (not a constraint), “≤,” “≥,” or “=.” The constraint button is pressed until the appropriate icon is selected. If a constraint is binding the constraint icon is boxed; if a constraint is violated the constraint is highlighted.



Binding Constraint



Violated Constraint

- E - #Map value field used to map two or more analysis functions to one design function.

- F - Allowable value field used as a limiting value for a constraint, or the worst value you will accept.
- G- Indifference Value field used as a goal value for an objective and beyond which you are “indifferent” to further improvement, preferring instead that other objectives were improved.
- H- Unmapping button used to unmap a design function.
- I- List of Unmapped Design Functions.
- J - Mapping button used to map analysis functions to design functions.
- K- Multiple Objective Space--This space is reserved for objective weighting coefficient scales used with multiple objectives, as shown in Section 6.3.

6.2.2 Unmapped Analysis Functions

The lower left of the window displays the unmapped analysis functions. These are outputs of the analysis model that are not currently mapped to be design functions. The button to the left of the name is the Mapping button. When pressed, the function is made a design function; it appears in the upper part of the window and is deleted from the list of unmapped analysis functions. Sometimes one analysis function is mapped to two or more design functions, as, for example, when an analysis function is to be both a “less than” and “greater than” constraint. In such a case, the analysis function can be copied into the design function list by holding down the shift key while pressing the mapping button.

6.2.3 Design Functions

Design Function Names and Values

The upper half of the window displays the design functions. The button to the left of the name is the Unmapping button. When pressed, the function is deleted from the list of design functions and reappears in the unmapped function list. The names of the design functions may be edited, although this is usually only done when several analysis functions are mapped to one design function--in this case a new unique name must be chosen.

Values for design functions are not editable, since they are computed from the analysis model.

Objective and Constraint buttons

To the right of the value field are two buttons (C and D in the diagram) that are underneath the label “Type.” You press the first button, the Objective button, to make a design function an objective. As this button is pressed, icons will show blank (not an objective), upward arrow (maximize) or downward arrow (minimize). The second button is the Constraint button, which you press to make a function a constraint. Choices for constraints are blank (not a constraint), less than (\leq), greater than (\geq) and equality ($=$). Any function can be an objective, a constraint, or both.

Choices for objectives and constraints are tied to allowable and indifference values. When the allowable is greater than the indifference value, the Objective button will not show upward arrow (maximize); the constraint button will not show greater than (\geq). To make these choices available, the allowable must be set to be less than the indifference value.

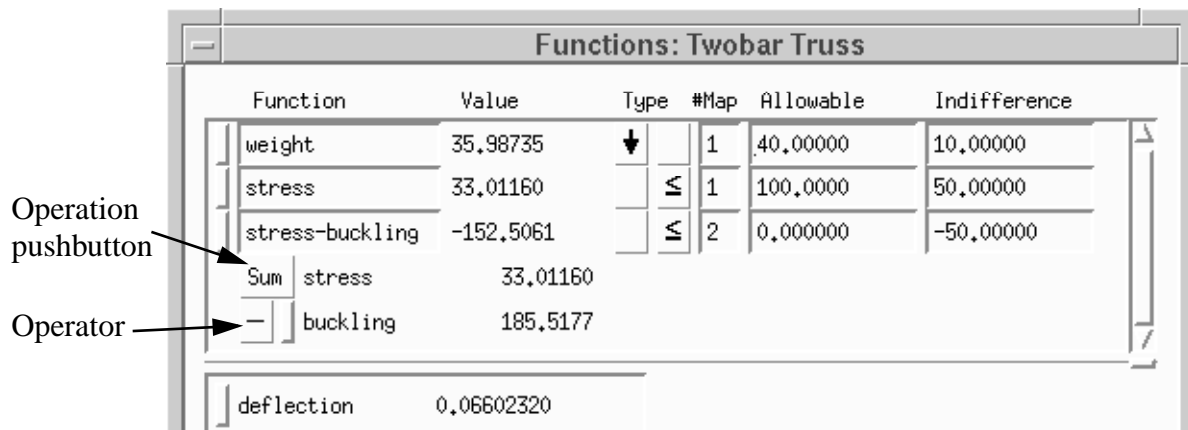
Map Value Field

Under the column “# Map” is a frame specifying the number of functions mapped to the

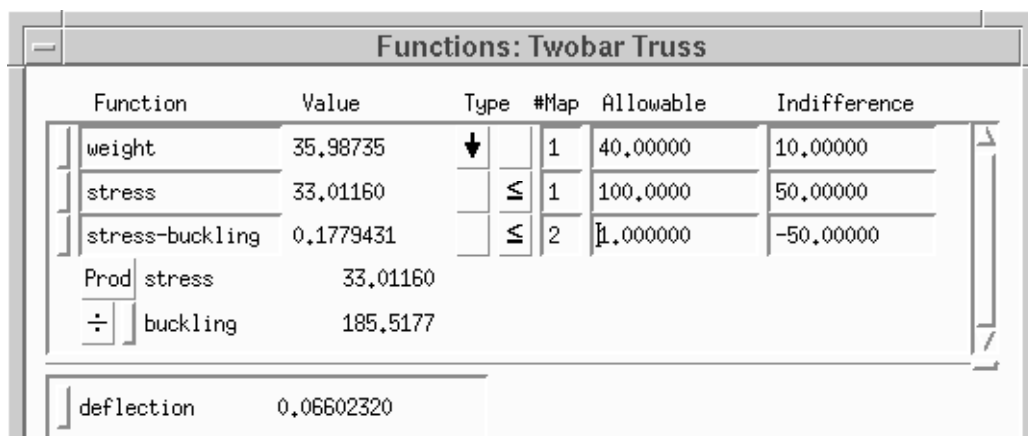
design function. When the default is changed from 1, additional fields open up below the design function; in the first row the function name is the same as the design function, while the other rows are marked “NULL FUNCTION.” The next unmapped analysis functions selected to be mapped will fill the Null Function slots.

When several analysis functions are mapped to one design function, the design function value is calculated as the sum, product, min or max of these functions, depending on the state of the Operation pushbutton, as shown below. “Sum” specifies that the functions which follow will be added or subtracted; the operators are “-” and “+”. “Prod” specifies that the functions which follow will be multiplied or divided; the operators are “*” and “/”. “Max” and “Min” return the maximum or minimum values of the functions respectively. Operators are changed by pressing the operator button.

The figure below shows an example of a many-to-one mapping used in the tutorial. In the example, a constraint, stress minus buckling ≤ 0 , has been created. Since stress is also a constraint by itself it was copied up to the design function list by holding down the shift key when the mapping button was pushed.



Another way to model the buckling constraint is, stress/buckling stress ≤ 1 . This is easily accomplished by changing the Operation button from “Sum” to “Prod,” as shown below.



The Max function is illustrated in the next example. The maximum of stress1, stress2, and stress3 will be taken as the function value for the constraint. Min and Max functions should be used with caution, since these functions are non-differentiable at breakpoints where the Min or Max changes from one of the set to another.

Function	Value	Type	#Map	Allowable	Indifference
weight	35.98735	↓	1	43.20000	28.80000
Max stress	185.5177	↕	3	39.60000	26.40000
Max stress1	33.01160				
stress2	185.5177				
stress3	0.06602320				

To delete an analysis function which is one of several mapped to a design function, you press the unmapping button to the left of the name. The name “NULL FUNCTION” will then appear in the slot, to be filled by the next analysis function that is made a design function. Note that the original analysis function from which the design function was created cannot be deleted (It does not have an Unmapping button; see “stress1” above). To delete this analysis function, you would have to delete the design function and then begin the mapping process over again.

All function names must be unique, so when several analysis functions are mapped to a design function, its name must be made unique to all other names.

Allowable and Indifference Value Fields

The Allowable Value is the limiting value for a constraint, or the worst value you will accept. The Indifference Value is the value below which you are indifferent to further improvement; it is the goal value for the function. When a design function is made a constraint, the allowable value is considered a hard limit which should not be violated at the optimum. When a design function is made an objective, the indifference value is considered a goal for the function; if the value reaches this goal, the objective essentially “drops out” of the problem while other objectives are improved, except for the case of a single objective, when it will not drop out.

All functions must have allowable and indifference values. Besides their use as limits for objectives and constraints, the difference of these values is used for scaling, according to the formula:

$$df_i = \frac{\overline{df}_i - C_i}{\text{Allowable}_i - \text{Indifference}_i} \quad \text{where } C_i = \begin{matrix} \text{Allowable}_i \text{ for regular constraints} \\ \text{Indifference}_i \text{ for objective constraints} \end{matrix}$$

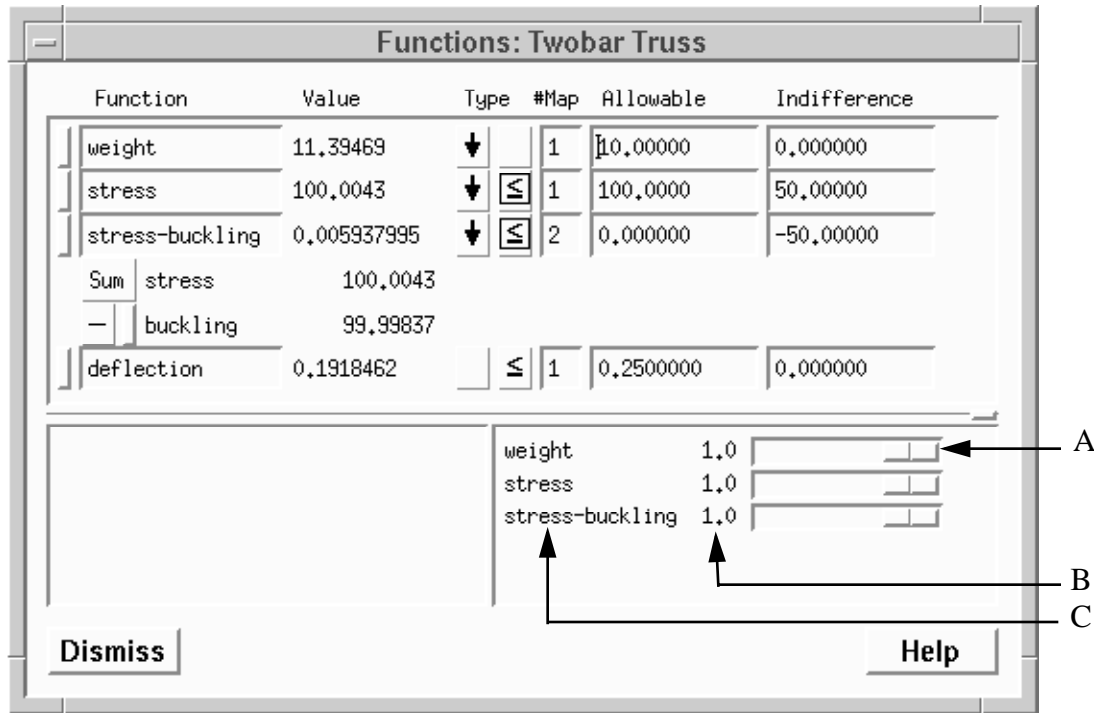
where df_i is the scaled value and \overline{df}_i is the unscaled value. Even if a function is not an objective, for example, you should pick an appropriate indifference value (as if it were an objective)

so that it can be scaled properly. If you are uncertain what appropriate indifference or allowable values should be, another rule to use is this: the difference of the allowable and indifference values, when divided into the function, should give a value between 1 and 10.

For minimize objectives or less than (\leq) constraints, the allowable value must be greater than the indifference value. The reverse is true for maximize objectives and greater than (\geq) constraints.

6.3 Multiple Objective Problems

6.3.1 Multiple Objective Reference Diagram



- A- Objective weighting coefficient slider bar.
- B- Objective weighting coefficient value.
- C- Objective name.

6.3.2 Multiple Objective Operation

When more than one objective is chosen, slider bars and weighting coefficients appear in the lower right hand corner of the window. The relative magnitudes of these coefficients reflect the emphasis the objectives will be given during the optimization. If for example, all weights are equal, all objectives are emphasized equally. If one weight is twice another, then that objective receives twice as much emphasis as the other.

6.3.3 Multiple Objective Solution Method

As with most approaches to multiple objective optimization, OptdesX uses a transformation that converts the multiple objective problem to a single objective problem. After studying several different approaches on more than 300 problems, we selected the “Min-Max” method because of its superior ability to compromise among objectives. The Min-Max method uses a dummy variable, which we call γ , as its objective. A statement of the multiple objective problem follows:

$$\text{Min } \gamma \quad 0 \leq \gamma \quad (1)$$

$$w_i \left\{ \frac{f_i - f_i^*}{\hat{f}_i - f_i^*} \right\} - \gamma \leq 0 \quad \text{for all } i \text{ objectives} \quad (2)$$

$$\left\{ \frac{f_i - \hat{f}_i}{\hat{f}_i - f_i^*} \right\} \leq 0 \quad \text{for all } i \text{ objectives that are also constraints} \quad (3)$$

$$g_j \leq 0 \quad \text{for all } j \text{ regular inequality constraints} \quad (4)$$

$$h_k = 0 \quad \text{for all } k \text{ equality constraints} \quad (5)$$

where f^* is the indifference value of function f ,
 \hat{f} is the allowable value, and
 w_i is the weighting coefficient

How does this method work? Assume, for the sake of discussion, that all objectives are to be minimized and all weighting coefficients have a value of one. The objectives are scaled in (2) such that they are equal to one at their allowable values and equal to zero at their indifference values. As γ is decreased from some positive starting number, the largest scaled objective will become binding in the “objective constraint” represented by (2). For γ to be reduced further, this objective must be reduced. This will be done until the next largest objective becomes binding, at which point both objectives must be reduced for γ to be minimized. Thus this method “pushes” all objectives away from their allowable values towards their indifference values. At an optimum all binding objectives will be pushed an equal amount towards their indifference values.

The effect of the weighting coefficients becomes apparent from examination of (2). When equal to one, a coefficient has no effect on the problem; when equal to zero, the associated objective disappears. When equal to a value of, for example, 0.5, the scaled value of the objective is reduced by half. Thus this objective will not become binding--and thereby be reduced--until its scaled value is twice that of other binding objectives. Within the range defined by its indifference and allowable values, this objective will be twice as far away from its indifference as the others.

Typically objectives in an engineering problem are conflicting--reducing one increases another, and the optimum occurs at a compromise between the two. Thus you should not specify too many objectives in a problem or the algorithm will not be able to move. Limit multiple objectives to two or three functions.

(This page intentionally left blank.)

7 Gradients Window Reference

7.1 Overview

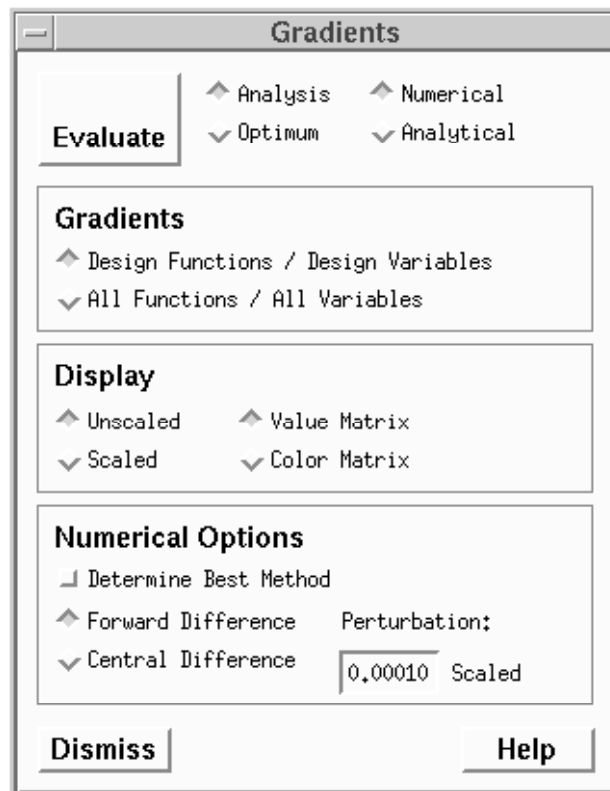
The Gradients window is used to evaluate and display gradients (derivatives) of the analysis and/or design functions. The gradient vector, denoted as ∇f , is defined to be the vector of partial derivatives of a function (which we consider to be a column vector, although to save space below we write the transpose of the gradient, which is a row vector):

$$\nabla f^T = \left[\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial x_3}, \dots, \frac{\partial f}{\partial x_n} \right]$$

You may wish to evaluate gradients for several reasons: gradients have inherent value as indicators of the sensitivity of functions to variables; the magnitudes of the gradients provide a check on the scaling of the problem, and accurate gradients are essential for the successful operation of the GRG, SQP and BNB optimization algorithms in OptdesX.

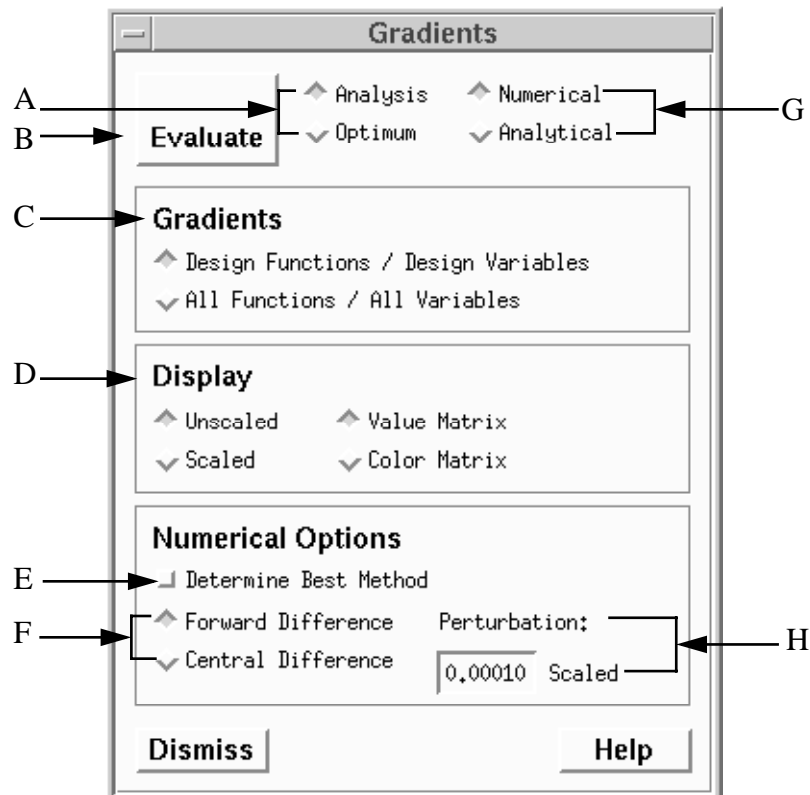
Pressing the Evaluate pushbutton causes gradients to be evaluated. Gradients are stored and may be displayed at any time as long as they remain current. Gradients remain current if the point of evaluation, numerical method and perturbation are not changed.

The Gradients box is divided into several sections, as shown below. The Analysis-Optimum radio box toggles between gradients of the analysis model (Section 7.2) and gradients of the optimal objective (Section 7.5). The Gradients box sets the scope of the evaluation (Sections 7.2.2, 7.5); the Display box determines the way gradients are displayed (Section 7.2.3); the Numerical Options box sets the method and perturbation (Section 7.2.4).



7.2 Gradients of the Analysis Model

7.2.1 Reference Diagram



- A - Analysis-Optimum radio box used to select gradients of the analysis model or gradients of the optimal objective.
- B - Evaluate pushbutton that causes gradients to be evaluated. If the gradients are current the pushbutton changes to "Display." Once gradients are taken, they remain current as long as the design point, the numerical method and perturbation are not changed.
- C - Gradients box used to select the gradients to be evaluated. Choices are gradients of design functions with respect to design variables (the gradients the algorithms use) or gradients of all functions with respect to all variables.
- D - Display box used to select the way gradients are displayed--either in scaled or unscaled form, and either as a matrix of values or a matrix of colors.
- E - Determine Best Method toggle button used to estimate the best numerical method and perturbation for a particular model.
- F - Numerical Derivative Type: Central difference or Forward Difference
- G - Numerical-Analytical radio box. This box is not currently functional--all gradients are evaluated numerically in OptdesX.
- H - Numerical Perturbation (scaled space).

7.2.2 The Gradients Box

When the Analysis-Optimum radio box is set to “Analysis,” you can select the scope of the gradients evaluation to be the design functions with respect to the design variables, or to be all functions with respect to all variables.

You select gradients of design functions with respect to design variables when you wish to examine the gradients the optimization algorithms will use. The algorithms only need these particular derivatives because the unmapped analysis functions and variables are not part of the optimization problem. Design function gradients can be viewed in either scaled or unscaled form.

For example, suppose we have an optimization problem defined as shown in the Variables and Functions windows given below:

Variable	Value	Type	#Map	Minimum	Maximum
height	30,00000	▲ C	1	10,00000	30,00000
diameter	3,000000	▲ C	1	1,000000	3,000000
width	60,00000	C	1	40,00000	80,00000
thickness	0,1500000	C	1	0,05000000	0,5000000

density	0,3000000
modulus	30000,00
load	66,00000

Dismiss Compute Functions Post Process Hardcopy Help

Function	Value	Type	#Map	Allowable	Indifference
weight	35,98735	▼	1	40,00000	10,00000
stress	33,01160	≤	1	100,0000	50,00000
stress-buckling	-152,5061	≤	2	0,000000	-50,00000
Sum stress	33,01160				
Sum buckling	185,5177				
deflection	0,06602320	≤	1	0,2500000	0,05000000

Dismiss Help

We have four design functions and four design variables. Thus the gradients matrix for “Design functions / Design variables” would be a four by four matrix, where each column is a gradient vector. Derivatives with respect to density, modulus and load would not be evaluated or shown, since they are not design variables. When “Display” options are set to “Unscaled” and “Value Matrix” the following window is opened (this window has been enlarged horizon-

tally by dragging the right border.):

	weight	stress	stress-buckling	deflection
height	0,5997942	-0,5501704	5,633651	-0,001100304
diameter	11,99578	-11,00350	-134,3756	-0,02200700
width	0,2998971	0,2750989	3,367010	0,001650621
thickness	239,9157	-220,0443	-226,2133	-0,4400886

$$\frac{\partial \text{weight}}{\partial \text{thickness}}$$

Now suppose we select derivatives of all functions with respect to all variables. We should have a seven by four matrix, since derivatives with respect to density, modulus and load are evaluated and shown. The matrix display is:

	weight	stress	stress-buckling	deflection
height	0,5997942	-0,5501704	5,633651	-0,001100304
diameter	11,99578	-11,00350	-134,3756	-0,02200700
width	0,2998971	0,2750989	3,367010	0,001650621
thickness	239,9157	-220,0443	-226,2133	-0,4400886
density	119,9578	0,000000	0,000000	0,000000
modulus	0,000000	0,000000	-0,006183924	-2,200773e-06
load	0,000000	0,5001757	0,5001757	0,001000351

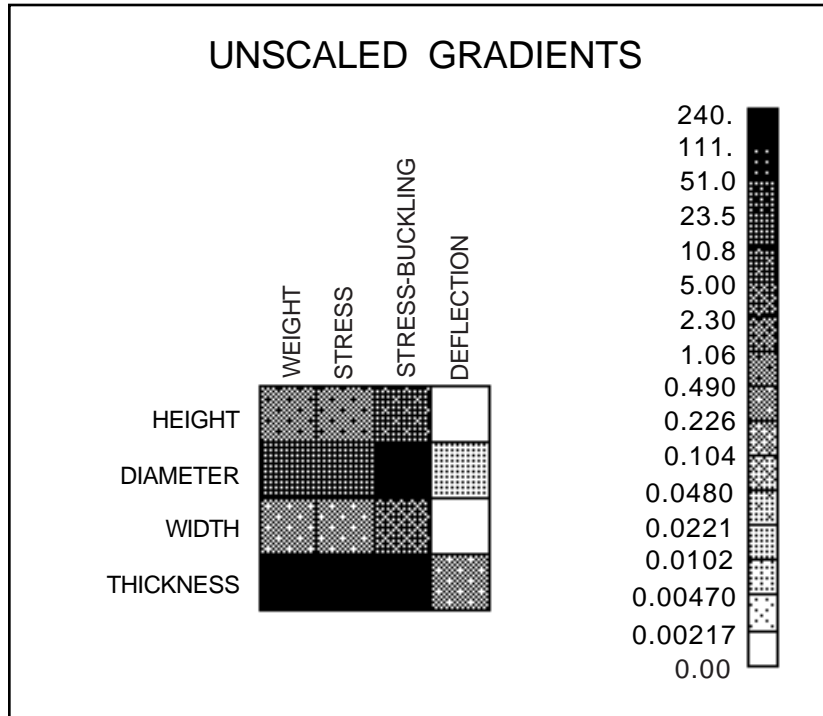
7.2.3 Display Options

Once gradients have been evaluated, there are several ways they can be displayed without having to re-evaluate them again, providing the gradients remain current. Gradients are current if the point of evaluation, perturbation and numerical method remain the same. To indicate that gradients are current the Evaluate button changes to “Display.”

Value-Color Radio Box

The Value-Color radio box lets you set either an “analog” (i.e. color) or “digital” (i.e. value) display of gradients. We presented examples of gradient value matrices in the previous section. The value matrix window can be enlarged in the horizontal or vertical directions to show more of the matrix, or the matrix can be scrolled.

You can also display gradients as a color matrix. A color matrix gives you a sense of the relative magnitudes of the gradient matrix as a whole. A color matrix representation of the first gradient value matrix given above (where colors have been replaced by patterns for the manual), is:



This graph shows that the gradients range in absolute value from approximately 0. to 240. The three dark cells along the row for thickness indicate that these derivatives are relatively large, while the light cells for the derivatives of deflection with respect to height and width indicate zero or small derivatives. Note that the color scale is a log scale.

A row of zero derivatives, which would show up as a row of dark blue cells, indicates that the variable represented by the row does not affect any functions. A column of blue cells indicates that the function represented by the column is not a function of any of the variables. Both of these states usually mean there is an error in the analysis model. Obviously, for example, it would make no sense to have an optimization variable that didn't affect any of the functions--changing the variable would have no effect on the problem. Similarly, a function that was not affected by any variables could not be optimized. A row or column of red cells, for a scaled matrix, may mean that the scaling should be adjusted. You can change scaling for a row by changing the mins and maxes for the associated variable or for a column by changing allowable and indifference values for the corresponding function. See Section 7.3 for more information on this subject.

Unscaled-Scaled Radio Box

The Unscaled-Scaled Radio Box lets you display gradients in either scaled or unscaled form. The scaled option can be selected only when gradients are set for "Design functions / Design variables." Scaled gradients cannot be displayed for "All functions / All Variables" since infor-

mation needed for scaling is not available for unmapped variables and functions.

The OptdesX algorithms work in scaled space. One way to check the appropriateness of problem scaling is to examine scaled gradients. This is discussed in detail in Section 7.3.

7.2.4 Numerical Options

The OptdesX algorithms require gradients. OptdesX evaluates gradients numerically, using either a central or forward difference method, depending on what you select in this box. A central difference derivative is given by:

$$\frac{\partial f_i}{\partial x_j} \approx \frac{f_i(x_1, x_2, \dots, x_j + \delta x, \dots, x_n) - f_i(x_1, x_2, \dots, x_j - \delta x, \dots, x_n)}{2\delta x}$$

A forward difference derivative is given by:

$$\frac{\partial f_i}{\partial x_j} \approx \frac{f_i(x_1, x_2, \dots, x_j + \delta x, \dots, x_n) - f_i(x_1, x_2, \dots, x_j, \dots, x_n)}{\delta x}$$

where δx is the derivative perturbation, which can also be set in this box. When the scope is “Design Functions / Design Variables,” or when the algorithms are run, the perturbation set here is a scaled number, and is multiplied by $(\text{Max}_i - \text{Min}_i)/2$ to obtain the unscaled perturbation for each variable. When the scope is “All Functions / All Variables,” the perturbation is an unscaled number.

Choosing an appropriate method and perturbation can be critical to the success of an optimization. When using a numerical derivative, there is a trade-off between two errors: truncation error and round-off error. If you have smooth analytical functions, round-off error is usually not important, and you can use a forward difference method with a small perturbation to minimize truncation error. If, however, you have noisy functions, then round-off error can become significant. Round-off error is decreased by using a larger perturbation, but this increases truncation error. To reduce truncation error with a large perturbation, you can use a central difference method, which has much smaller truncation error than the forward difference method (on the order of δx^2 instead of δx), but which requires $2n$ calls to the analysis model (n = no. of design variables) to obtain gradients, instead of n calls.

“The Determine Best Method” feature can assist you in determining the right method and perturbation. See Section 7.4 for a detailed discussion of this feature.

7.3 Derivatives as a Check of Problem Scaling.

The OptdesX algorithms work in scaled space. Variables are scaled to between -1 and +1 using minimum and maximum values; functions are scaled to be on the order of 1 using allowable and indifference values. Scaling formulae are given in Section 2.4. Since a derivative involves both functions and variables, it is affected by the scaling of both, according to the

relationship,

$$\left(\frac{\partial f_i}{\partial x_j}\right)_{\text{scaled}} \approx \left(\frac{\partial f_i}{\partial x_j}\right)_{\text{unscaled}} \left\{ \frac{(\text{Max}_j - \text{Min}_j)}{2(\text{Allowable}_i - \text{Indifference}_i)} \right\}$$

Scaling is important! A poorly scaled problem can cause premature termination of the algorithms. One way to check the scaling of a problem is to check the gradients: when the problem is properly scaled, the gradients should all be the same order of magnitude. Consider the example given in Section 7.2.2. The unscaled gradients were given there as,

Unscaled Gradients Design Functions/Variables				
	weight	stress	stress-buckling	deflection
height	0,5997942	-0,5501704	5,633651	-0,001100304
diameter	11,99578	-11,00350	-134,3756	-0,02200700
width	0,2998971	0,2750989	3,367010	0,001650621
thickness	239,9157	-220,0443	-226,2133	-0,4400886

Dismiss Hardcopy

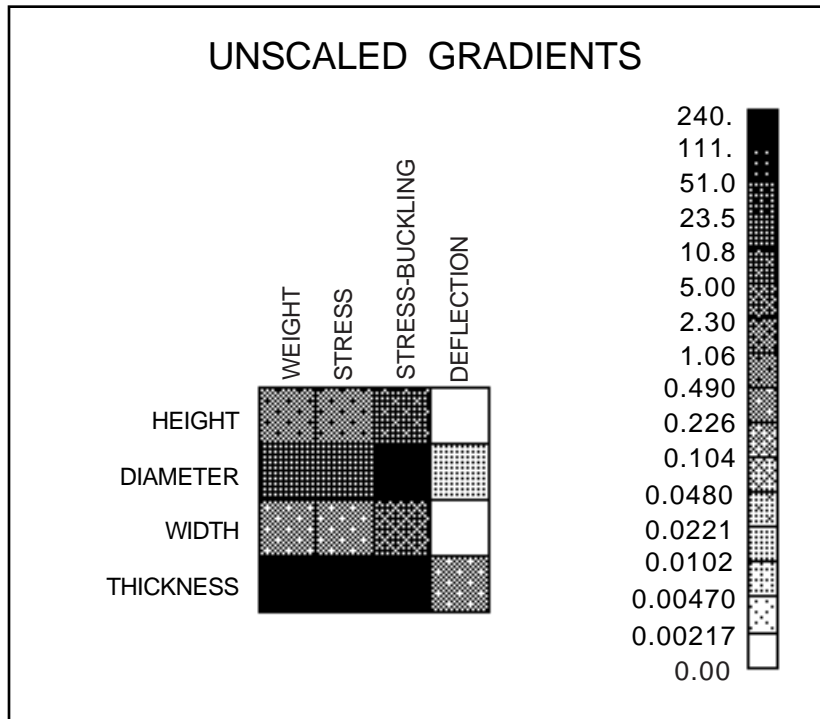
The scaled gradients are,

Scaled Gradients Design Functions/Variables				
	weight	stress	stress-buckling	deflection
height	0,1999314	-0,1100341	1,126730	-0,05501520
diameter	0,3998595	-0,2200700	-2,687512	-0,1100350
width	0,1999314	0,1100396	1,346804	0,1650621
thickness	1,799368	-0,9901994	-1,017960	-0,4950997

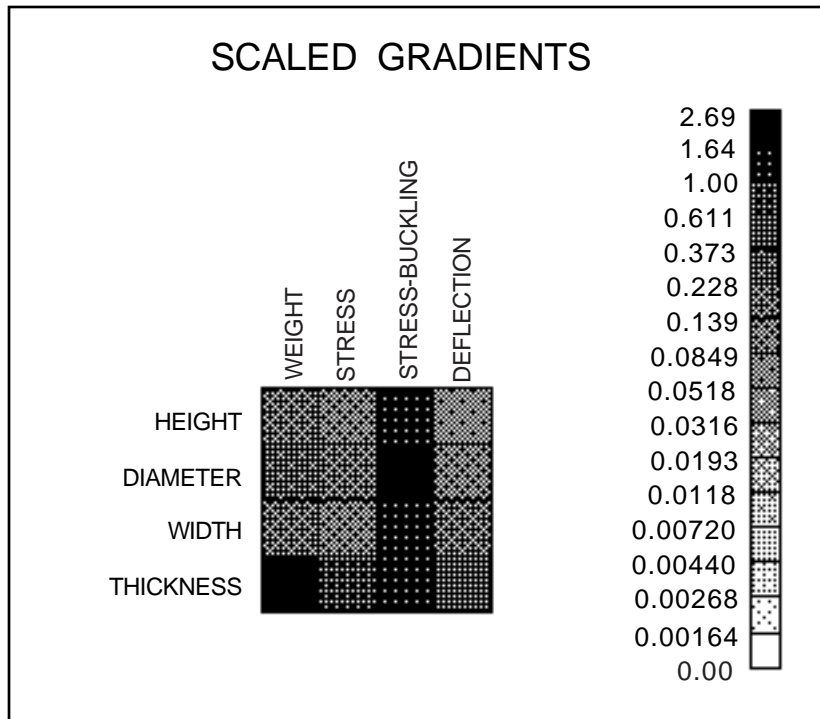
Dismiss Hardcopy

Comparison of these value matrices shows that scaling has had a “leveling” effect: the stress-buckling gradient has been reduced from magnitudes on the order of 100 to magnitudes on the order of 1, while the deflection gradient has been increased from magnitudes on the order of 0.001 to 0.1.

It is also instructive to compare color matrices. The unscaled color matrix is,

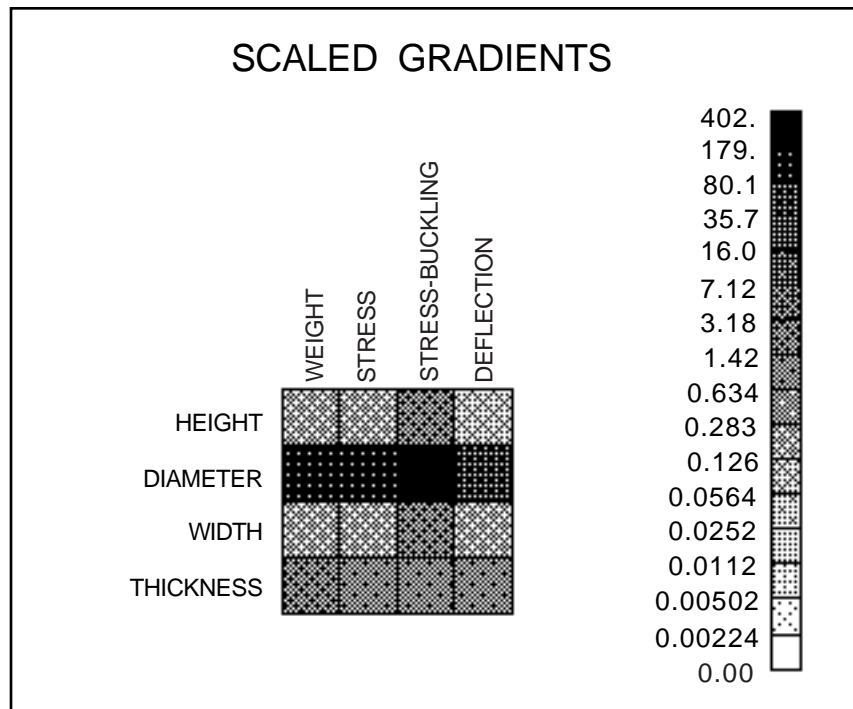


whereas the scaled color matrix is,



The scaled matrix has a narrower range of colors, but more importantly, its range is 0-2.69, while the range of the unscaled matrix is 0-240.

The preceding example showed a case of good scaling. How might we recognize a case of bad scaling? Suppose one of the variables, such as diameter, was scaled poorly--because we chose a maximum of 300 instead of 3. The color matrix becomes.



All of the derivatives associated with diameter are now much larger than the others, as indicated by the warmer colors and the scale bar that goes from 0 to 402. Similarly, problems with scaling for a function would show up as a column of extreme colors along with an inappropriate range. Scaled derivatives are made smaller in value by collapsing Mins and Maxes for the variables and increasing the difference of the allowable and indifference values for functions.

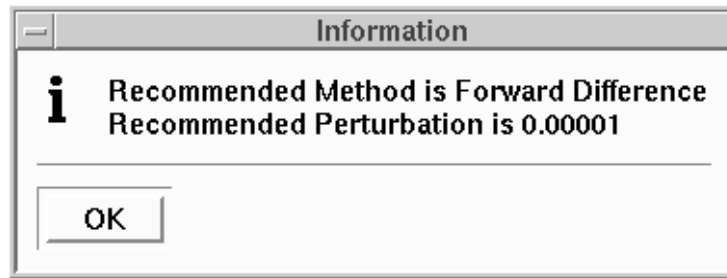
7.4 Determine Best Method Feature

The Determine Best Method feature helps you to determine the best numerical method (central or forward difference) and the best perturbation for your particular analysis model. Analysis models are often “noisy,” meaning the function values lack significant figures because of approximation, solution method error (such as would result from the finite element method or numerical integration), or round-off error. Noise is amplified in numerical derivatives and can cause premature termination or failure of the optimization algorithms. It is critical that the numerical method and perturbation minimize the effects of noise as much as possible.

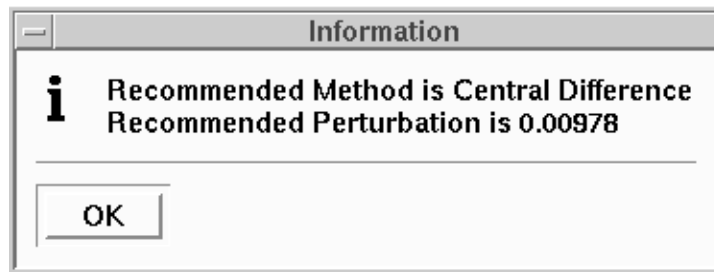
The Determine Best Method feature is executed by toggling the feature on and then pressing the Evaluate button. Using the design variable at the top of the design variable list, OptdesX will test your model for noise and then report what it believes the best numerical parameters are. The perturbation and method will automatically be set to the recommended values, and the toggle will be turned off. The procedure requires five to seven analysis calls.

If you suspect noise is present, you should determine the best method for several variables (by modifying the design variable list so different variables are at the top). The worst case should then be chosen.

For the Twobar Truss example, we have smooth functions, so after executing this feature Opt-desX reports,



Suppose, however, we introduce noise into the model such that each function only has three significant figures (a random number having been added to the function values at the fourth significant figure). When we do this, the Determine Best Method feature reports,



The method and perturbation are automatically set to these values. When we evaluate the gradients with these parameter values we get,

	weight	stress	stress-buckling	deflection
height	0.5727282	-0.5626663	5.648463	-0.001141948
diameter	11.92448	-11.05745	-134.5457	-0.02178800
width	0.2957615	0.2547978	3.326828	0.001647946
thickness	238.3126	-219.9102	-224.9108	-0.4397791

Dismiss Hardcopy

We can compare these to the true derivatives:

	weight	stress	stress-buckling	deflection
height	0.5997942	-0.5501704	5.633651	-0.001100304
diameter	11.99578	-11.00350	-134.3756	-0.02200700
width	0.2998971	0.2750989	3.367010	0.001650621
thickness	239.9157	-220.0443	-226.2133	-0.4400886

Dismiss Hardcopy

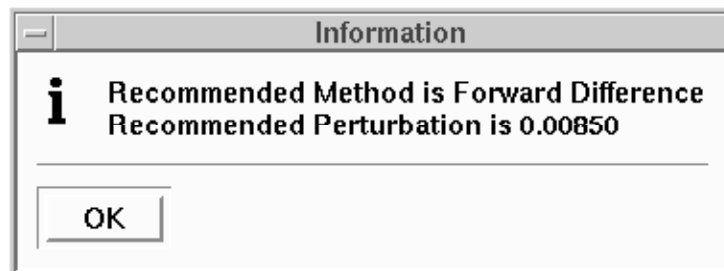
What would the gradients have been if we had evaluated them with the default settings of the forward difference method with a perturbation of 0.0001? The results are given below:

	weight	stress	stress-buckling	deflection
height	1,617549	-4,641735	-1,291064	-0,006897617
diameter	51,72636	15,44060	-115,6681	-0,03725717
width	2,378913	-1,214029	0,8611921	-0,001724811
thickness	498,6496	-155,5962	-25,67773	-0,4042404

Dismiss Hardcopy

Some of the derivatives are not even of the right sign! As you can imagine, such derivatives would result in very poor optimization results.

For the example just cited, the optimal derivative method and perturbation were estimated using the first design variable in the list, which was height. Somewhat different results would be obtained if a different variable were at the top of the dv list. For example, for the same problem, but with diameter used to estimate the best numerical parameters instead of height, OptdesX recommends,



The gradients at these parameters are,

	weight	stress	stress-buckling	deflection
height	0,5052349	-0,6072870	5,612368	-0,001157268
diameter	11,76654	-10,93999	-134,7272	-0,02246217
width	0,2638490	0,2984332	3,390833	0,001650637
thickness	237,4006	-219,2087	-227,1783	-0,4353114

Dismiss Hardcopy

which are still very acceptable. Actually, for the example chosen, we are right at the threshold between changing from a forward difference to a central difference method. Different vari-

ables will give different results.

When you have a noisy model, we recommend that you evaluate the best method and perturbation for several variables, and then take the most conservative values, which would be the central difference method and the largest perturbation recommended for the central difference method.

7.5 Gradients of an Optimum

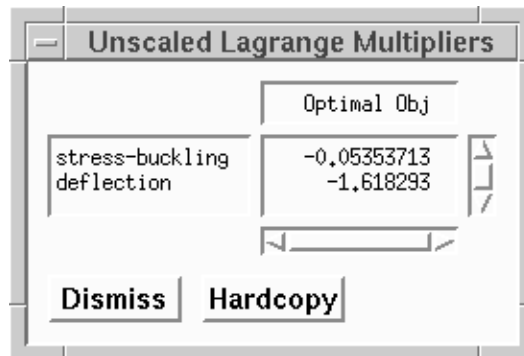
The Gradients window allows you to evaluate gradients at two levels: gradients of “regular” analysis functions, and gradients of *optimal* functions; specifically, gradients of the objective function at the optimum. Such gradients indicate the sensitivity of the optimal objective value to small changes in the optimization problem. To predict this sensitivity, the effects of the perturbation upon both the objective function and the binding constraints must be considered.

To obtain gradients of the optimum, you must be at an optimum, and you must have just completed running an algorithm. It is your responsibility to insure this (the software has limited capabilities to detect this, and it can easily be fooled). Gradients of the optimum are selected by setting the Analysis-Optimum radio box to “Optimum.” This causes the choices in the Gradients box to change to: “Objective / Constraint RHS (Lagrange Multipliers)” or “Objective / Unmapped Analysis Variables.”

The first choice, “Objective / Constraint RHS (Lagrange Multipliers),” is the vector of Lagrange Multipliers, which are automatically calculated during the course of an optimization with the GRG and SQP algorithms (thus no new calls to the ANAFUN routine are needed). The Lagrange multipliers, denoted by λ , are derivatives of the optimal objective value with respect to the right hand side of a constraint, or mathematically,

$$\lambda_j = \frac{\partial f^*}{\partial b_j}$$

where the asterisk denotes the optimal value of the objective, and b_j is the right hand side of constraint j . Thus this derivative tells us how the optimal value of the objective would change for a small positive change in the right hand side of a binding constraint. For example, in the Twobar truss, at the optimum, the stress-buckling and deflection constraints are binding. The vector of Lagrange multipliers is given as,



These values can be interpreted in the following way: if we were to increase the right hand side of the stress-buckling constraint from zero to 1.0, the optimal objective would decrease by approximately -0.0535 . Likewise, for the deflection constraint, if the right hand side were to increase by 1, the optimal objective would decrease by approximately -1.62 (increasing deflection by this amount however, is unrealistic, since this is not a small perturbation for deflection). Lagrange multipliers for constraints that are not binding have a value of zero and so are not shown.

For problems with multiple objectives, “objective constraints” are added to the original problem (see Section 6.3.3 for a description of these constraints). A Lagrange multiplier for a binding objective constraint is given the name of the objective with an “O” appended to it.

Since variable bounds can also be viewed as simple constraints, variables at bounds will also have Lagrange multiplier values. When displayed, the names of these values will be the name of the variable with a “U” appended for the upper bound or an “L” appended for the lower bound. The interpretation of the multiplier is the same: it gives the change in the objective with respect to a small positive change in the bound value.

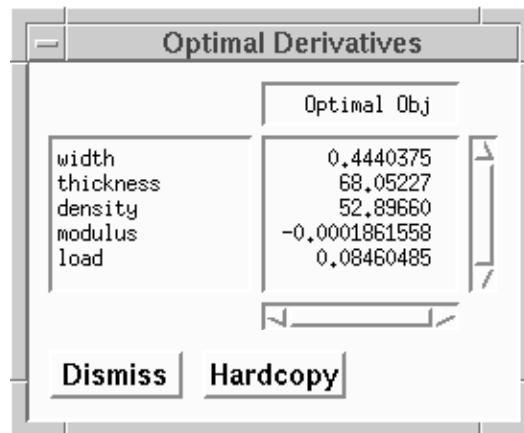
Since the Lagrange multipliers are derivatives, they are valid only for infinitesimal changes of the constraint right hand sides. For finite changes, the Lagrange multipliers will only approximately predict the change in the optimal objective. Also, it is assumed that the perturbation is small enough that the same set of binding constraints applies at the new optimum. If the set of binding constraints changes as a result of a perturbation, then predictions based on the current Lagrange multipliers are invalid.

The second choice for optimal gradients is “Objective / Unmapped Analysis Variables,” or the derivatives of the optimal objective with respect to the unmapped analysis variables. These derivatives can help you determine the sensitivity of an optimum to off-nominal conditions, since often unmapped analysis variables represent quantities which are assumed constant but which are actually somewhat uncertain. Derivatives of the optimal objective with respect to unmapped analysis variables are defined mathematically by the relationship,

$$\frac{\partial f^*}{\partial v_i} = \frac{\partial f}{\partial v_i} - \sum_{j=1}^{nb} \lambda_j \frac{\partial g_j}{\partial v_i}$$

where v_i is unmapped analysis variable i , f is the objective, g_j is binding constraint j , λ_j is the associated Lagrange multiplier, and the summation limit nb is the number of binding constraints. Computing these optimal derivatives requires Lagrange multipliers as well as regular derivatives. Thus OptdesX will have to evaluate regular derivatives with respect to the unmapped analysis variables to compute these optimal derivatives. Continuing the Twobar

truss example,



	Optimal Obj
width	0.4440375
thickness	68.05227
density	52.89660
modulus	-0.0001861558
load	0.08460485

Dismiss Hardcopy

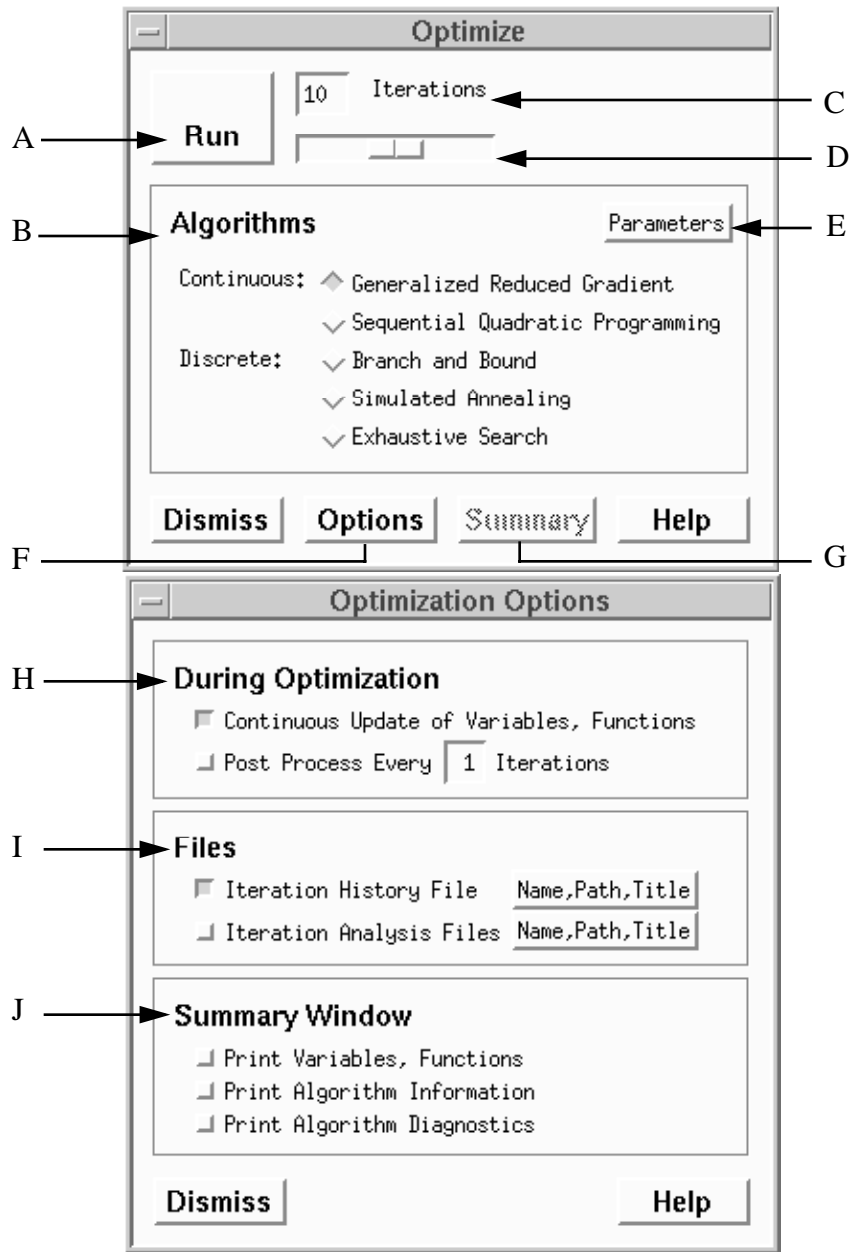
The interpretation of these values is similar to Lagrange multipliers: if width, for example, were increased by 1, the objective function would increase by approximately 0.444.

You can verify optimal derivatives by making a small perturbation to the original problem, reoptimizing, and comparing the change in the objective function with the predicted change given by the derivative.

Because of the way multiple objective problems are handled (Section 6.3.3), optimal derivatives with respect to unmapped analysis variables are not available for multiple objective problems.

8 Optimize Window Reference

8.1 Reference Diagram



- A - Run pushbutton used to begin execution of an optimization algorithm. After execution is started, the button label changes to “Stop”, and pushing it will stop execution. In general, execution will continue until an optimum is reached or until the specified number of iterations/combinations/cycles has been completed.
- B - Algorithms box used to select an optimization algorithm. GRG is the default optimization algorithm.
- C - Iteration value field used to set the number of iterations for continuous optimization. This number can be changed with the value field in C or the scale in D.

- D - Iteration scale.
- E - Parameters pushbutton used to display and set algorithm parameters. Each algorithm has associated with it tolerance and/or convergence parameters.
- F - Options pushbutton used to display Optimization options.
- G - Summary pushbutton which opens (or if open, brings forward) the Optimization Summary window, which shows optimization information. The button is dimmed until an optimization has been done.
- H- During Optimization box used to specify whether variables and functions will be continuously updated and/or whether ANAPOS should be called during the optimization.
- I - Files box used to specify the files that will be created during an optimization. The default is to generate a History file (with the default name “History”), which can then be plotted. Intermediate designs created during the optimization can also be stored.
- J - Summary window options box used to select additional information beyond the default to be displayed in the Optimization Summary window.

8.2 Operation

The Start button begins execution of the optimization algorithm selected. After execution is started, the button label changes to “Stop,” and pushing it will halt execution. In general, execution will continue until an optimum is reached or until the specified number of iterations/combinations/cycles has been completed.

When execution is begun an Optimization Summary window is opened, giving information about the optimization. Additional information beyond the default summary can be specified in the Options Window, which is accessed by pressing the Options button. Hardcopy of the information in the Optimization Summary window can be obtained by pressing the Hardcopy pushbutton.

8.3 OptdesX Algorithms--General Information

8.3.1 Generalized Reduced Gradient

The Generalized Reduced Gradient (GRG) is used to solve problems with continuous variables. The GRG algorithm is actually a GRG-SQP hybrid developed at Brigham Young University. It is more robust than the SQP algorithm, i.e., it has been able to solve a wider variety of problems. Although it may go slightly infeasible during an iteration, GRG will return a feasible design at the end of an iteration, thereby guaranteeing that a better solution will be available if the optimization is terminated early. This is the algorithm you should try first.

8.3.2 Sequential Quadratic Programming

The Sequential Quadratic Programming (SQP) algorithm is used to solve problems with continuous variables. It is potentially the fastest and most efficient algorithm. However, it sometimes goes quite infeasible at intermediate iterations, which can cause trouble for some analysis software. If GRG seems to be making poor progress, you should try the SQP algorithm. SQP is often the best algorithm for problems containing several equality constraints.

8.3.3 Branch and Bound

The Branch and Bound (BNB) algorithm is used to solve mixed problems (those with both continuous and discrete variables) or pure discrete problems, using the classical branch and bound method. A restriction of this method is that the discrete variables must be able to be modeled as continuous variables while the BNB tree is evaluated. At each node in the tree an optimization is performed in order to obtain an estimate of the objective for the nodes below it. If this estimate is worse than a known solution, the branch is “pruned,” i.e., it is not examined further. Because a branch and bound tree can become prohibitively large, computation and combinations can be reduced by specifying neighborhoods for each discrete variable and by making a linear approximation to the optimization problem at intermediate leaves on the tree. These options are selected by pressing the Parameters button.

8.3.4 Simulated Annealing

The Simulated Annealing (SAN) algorithm is used to optimize problems with discrete variables only. The algorithm mimics the process of annealing of solids in nature, perturbing the design randomly until a lower “energy” (i.e. objective) is reached. The algorithm has the capability to escape local optima by occasionally accepting designs, according to a probability function, with a worse energy value. For discrete problems that are differentiable, a “filter” can be set such that perturbations that are made in unpromising directions are filtered out, thereby reducing the number of calls to the analysis model. On large problems SAN often finds a good solution with fewer function calls than BNB.

8.3.5 Exhaustive Search

The Exhaustive Search (EXS) algorithm can be used to solve mixed problems or pure discrete problems. As its name implies, EXS methodically examines every discrete combination. If there are continuous variables in the problem, an optimization is performed for each combination. Computation and combinations can be reduced by specifying neighborhoods for each discrete variable and by making a linear approximation to the optimization problem for continuous variables. These options are selected by pressing the Parameters button.

8.4 Essential Algorithm Information

8.4.1 Running Several Iterations at a Time

The GRG and SQP algorithms are much more effective when they are run several iterations at a time. This is because the algorithms can use information from previous iterations to correct the search direction for the current iteration. When the algorithms are run one iteration at a time, information about the previous directions is lost.

Sometimes it is necessary to obtain information about the design after each optimization iteration. You can turn on additional information beyond the default by selecting the toggle buttons in the Optimization Options window. If you require your own special information, you can set a toggle button in the During Optimization box of the Optimization Options window to call ANAPOS every iteration or every several iterations.

8.4.2 Effects of Scaling

The algorithms in OptdesX work in scaled space. Minimum and Maximum values are used to scale all variables to be between -1 and +1. Allowable and Indifference values are used to

scale functions to be on the order of 1. Poor scaling can adversely affect algorithm performance. One way to check scaling is to examine the gradients matrix; see Section 7.3. You should take care to select minimum and maximum bounds that are as “tight” as possible. Recall that allowable values are the worst values you would be willing to accept; indifference values are your goal values for the functions. If you are uncertain what appropriate indifference or allowable values should be, another rule to use is this: the difference of the allowable and indifference values, when divided into the function, should give a value between 1 and 10.

8.4.3 Crashes--Undefined Functions

The vast majority of program crashes occur inside the ANAFUN subroutine. This happens when either the analysis software has not been fully debugged, or OptdesX drives to some design that the analysis routine cannot handle--a design that divides by zero, takes the log of a negative number, etc. Often these problems can be solved by appropriately restricting the variables using tighter minimum and maximum bounds. Also, on some systems, a divide by zero does not crash the software; rather character strings such as “NaN” or “Infinity” are passed back as values. **The operation of OptdesX is undefined when it receives such values.** The screen may lock up; the algorithms may not terminate execution, etc. **You must fix this condition.** You might put some “write” statements in the analysis software to monitor the designs that are being passed by OptdesX to determine exactly why ANAFUN is crashing. If a trace-back of the crash shows that the crash is not in the analysis routine, then please keep a record of the problem that would be sufficient to duplicate it and notify Design Synthesis.

8.4.4 Failure to make progress

There can be several reasons why the algorithms can fail to make progress. By far the most common is that the analysis software is not working properly and is computing nonsensical function values. You should take the necessary time to verify that the analysis software is working properly. After linking to OptdesX, do some trial-and-error design by setting the variable values and pushing the Compute Functions button to calculate the corresponding functions. You should examine a spectrum of designs. You may need to code some debug printouts in the analysis software so that software execution can be closely observed as it is called by OptdesX. If proper execution of the analysis software has been verified and an algorithm fails, you should try another algorithm. If all fail, it is still likely that an error exists in the analysis software somewhere.

8.4.5 Global vs. Local Optima

In general, for a nonlinear non-quadratic problem, it is impossible to know if an optimum is a global optimum or not. However, if the problem has been started from several different spots in design space and the algorithms always drive to the same point, then it is likely that a global optimum has been found.

8.4.6 Optimization of Noisy Functions

Numerical noise is common in engineering problems. Noise is defined as a lack of significant figures in the function values caused by approximation, solution technique, or round-off error. Design problems that involve finite element or finite difference representations, or that involve numerical integration, typically have noisy function values. Noise is amplified in the gradients and can cause premature termination or failure of the optimization algorithms. It is critical that

the effects of noise be minimized as much as possible.

The first thing you should do is examine your model and make calculations as noise-free as possible. This may involve tightening convergence tolerances on loops, etc., or rewriting sections of the model to avoid the noisy calculations altogether.

OptdesX calculates derivatives numerically using either a finite forward or central difference (Section 7.2.4). The Determine Best Method feature in Gradients window can help you determine the numerical parameters for gradients that minimize noise. You should execute this feature before you optimize a model for the first time.

8.4.7 Stopping Messages, General

The algorithms will terminate when either 1) the number of iterations/combinations/cycles specified has been completed, 2) an optimum is achieved and can be mathematically verified, or 3) the algorithm is not making progress so an optimum is assumed (the most common reason for an algorithm not to make progress is because an optimum has been reached, but due to tolerance bands on constraints, the mathematical conditions for an optimum are not quite satisfied).

If you get an “OPTIMUM REACHED” message, then the algorithms have identified a local optimum. An “OPTIMUM ASSUMED” message usually means a local optimum has been found, but you may wish to check this by running a different algorithm to see if further progress can be made.

8.5 Generalized Reduced Gradient Algorithm

8.5.1 Description and Performance

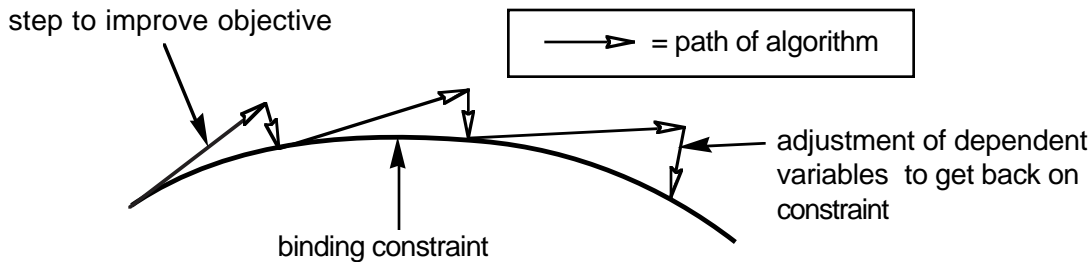
The Generalized Reduced Gradient algorithm is actually a GRG-SQP hybrid developed at Brigham Young University. The algorithm is robust and efficient. It has the desirable property of returning a feasible design at the end of each iteration. A complete description of the algorithm may be found in Parkinson A. and M. Wilson, “Development of a Hybrid GRG-SQP Algorithm for Constrained Nonlinear Programming,” *J. of Mech. Trans. and Automation in Design*, Trans. ASME, vol 110, Sept. 1988, p. 308.

An iteration of the algorithm is comprised of two parts: 1) determining a search direction, and 2) stepping in the search direction (performing a line search) until either the objective becomes worse, or new constraints become binding. Gradients are evaluated once each iteration. The analysis model may be evaluated many times during the course of the line search.

The search direction is found by solving a quadratic programming problem (QP problem) for the direction that best reduces the objective while staying inside of or tangent to the linearized constraint surface. The hessian matrix in the QP problem is developed using a BFGS (Broyden, Fletcher, Goldfarb, Shanno) update.

Once the direction is found, the line search is executed. The design variables are partitioned into two sets: independent variables and dependent variables. The number of dependent vari-

ables is equal to the number of binding constraints. The independent and dependent variables are adjusted according to the search direction for some small step. If the function is improved and the design is feasible, the adjustment of the variables continues for a larger step. If, however, the step has made the design infeasible, the dependent variables only are adjusted, using a Newton-Raphson technique, to make the design feasible again. Thus most steps have two parts: an adjustment of all the variables to improve the objective (according to a linear approximation of the constraints), and an adjustment of the dependent variables to restore feasibility. The path of the GRG algorithm has sometimes been called “hemstitching,” since it looks like the figure below:



The line search continues until the objective begins to get worse (we have gone too far), a new constraint becomes binding, requiring that we add to the set of dependent variables, or the Newton-Raphson technique for maintaining feasibility fails. We halt the iteration and obtain a new search direction when any of these conditions occurs.

Most engineering problems are constrained at the optimum. The GRG strategy of closely following binding constraints works well, it appears, because this rapidly takes the algorithm to the location of the constrained optimum.

If GRG is started from an infeasible point, it will temporarily ignore the objective while it reduces the most violated constraint. Once this constraint is satisfied, it is added to the constraint set and the next most violated constraint is reduced; this process is continued until a feasible point is found. When feasible, GRG begins to reduce the objective.

8.5.2 Storage Requirements

The GRG algorithm requires approximately the following amount of double precision storage:

$$\text{storage} = 3n_v^2 + 5n_v + 6n_f + n_v * n_f$$

where n_v is the number of design variables and n_f is the number of design functions. If you exceed allocated storage, an error message is printed (the GRG algorithm is written in Fortran, so storage cannot be allocated dynamically). The default size for double precision storage is 40000. Contact Design Synthesis if you need a version of the software with more storage.

8.5.3 Parameters

The default parameter values of GRG are set to provide a good compromise between solution accuracy and computational expense. Do not change these values arbitrarily. The GRG parameters are accessed by pressing the Parameters pushbutton when the GRG algorithm is selected. The following window is opened:

Tolerance Values:	
Constraint	0.0010000
Objective	0.0010000
Active Set	1.0000000
Step Length Values:	
Initial	0.0200000
Minimum	0.0001000

Dismiss Reset Help

You can edit values in this window in three ways: by changing the value directly in the value field, by dragging the slider bar, or by clicking on the arrow buttons.

Constraint Tolerance

When the scaled value of a constraint is zero plus or minus this tolerance, it is considered binding. Note that this definition allows the constraint to be slightly violated within the bounds of this tolerance. Reducing the constraint tolerance will force the binding constraints to be satisfied more tightly but will also require additional function calls.

Objective Tolerance

If the change in the objective function on three successive iterations is less than this tolerance, the algorithm will terminate with the message “Optimum Assumed, Objective Function not Changing.” Reducing this tolerance may allow GRG to squeeze a little more improvement out of the objective before stopping, at the expense of more function calls.

Active Set Tolerance

This parameter is not currently functional in the software and should always be set to “1.0”. It will be used in later releases to modify the GRG algorithm to be an active constraint algorithm for large problems.

Initial Steplength

This is the value of the steplength, in scaled space, at the beginning of an iteration. If the objective function is worse, the step is cut back. If the objective is still worse, a quadratic function is fit to the starting point and two worse steps and the minimum of the function predicted from the quadratic. If the objective improves after the initial step, the step is doubled until it reaches a maximum value; it is then increased by a constant amount each step.

Reducing this value will allow GRG to more accurately determine an optimal solution, particularly for an unconstrained problem where in the final iterations GRG may take very small

steps. This accuracy will come at the expense of additional function calls.

Minimum Steplength

This value is used by GRG to determine if the optimization should be terminated from lack of progress. If the total steplength achieved during an iteration is less than this value, the algorithm terminates with the message, “Optimum Assumed, Minimum Stepsize Reached.”

Reducing this value can force GRG to continue to search. However, sometimes reducing this value will have no effect, since sometimes the steplength can be set to zero (because, for example, the optimal steplength predicted from a quadratic function is negative), causing termination no matter how small the minimum steplength is.

8.5.4 Stopping Messages

When a mathematical optimum has been achieved, GRG will report,

“OPTIMUM ACHIEVED -- KUHN TUCKER CONDITIONS SATISFIED.”

If GRG fails to make progress because the objective is not changing or a minimum stepsize is reached, either of the following messages will be reported,

“OPTIMUM ASSUMED -- OBJECTIVE FUNCTION NOT CHANGING”

“OPTIMUM ASSUMED -- MINIMUM STEPSIZE REACHED.”

If the starting design is infeasible and a feasible point cannot be found, a message will be printed beginning as,

“FEASIBLE POINT NOT FOUND...”

8.5.5 Optimization Summary

Example text from the Optimization Summary window for a GRG run is given below.

```

ITERATION NUMBER          0
  Scaled Step Length      = 0.000000
  Scaled Objective        = 0.7993840
  Scaled Max Inequality   = -0.8397035
  Number Of Analysis Calls = 0
  Number Of Gradient Calls = 0

ITERATION NUMBER          1
  Scaled Step Length      = 1.065373
  Scaled Objective        = 0.3115309
  Scaled Max Inequality   = -1.393977e-05
  Number Of Analysis Calls = 11
  Number Of Gradient Calls = 1
    
```

An explanation of these parameters is as follows. The step length, which is the vector length of the change in the design variables, gives some idea of the distance the algorithm moved during an iteration. Recall that variables are scaled to be between -1 and +1, so step lengths will typically be less than 1.0. A step length greater than 0.75 is considered large; steps greater than about 0.2 represent substantial moves in design space. As an algorithm approaches an opti-

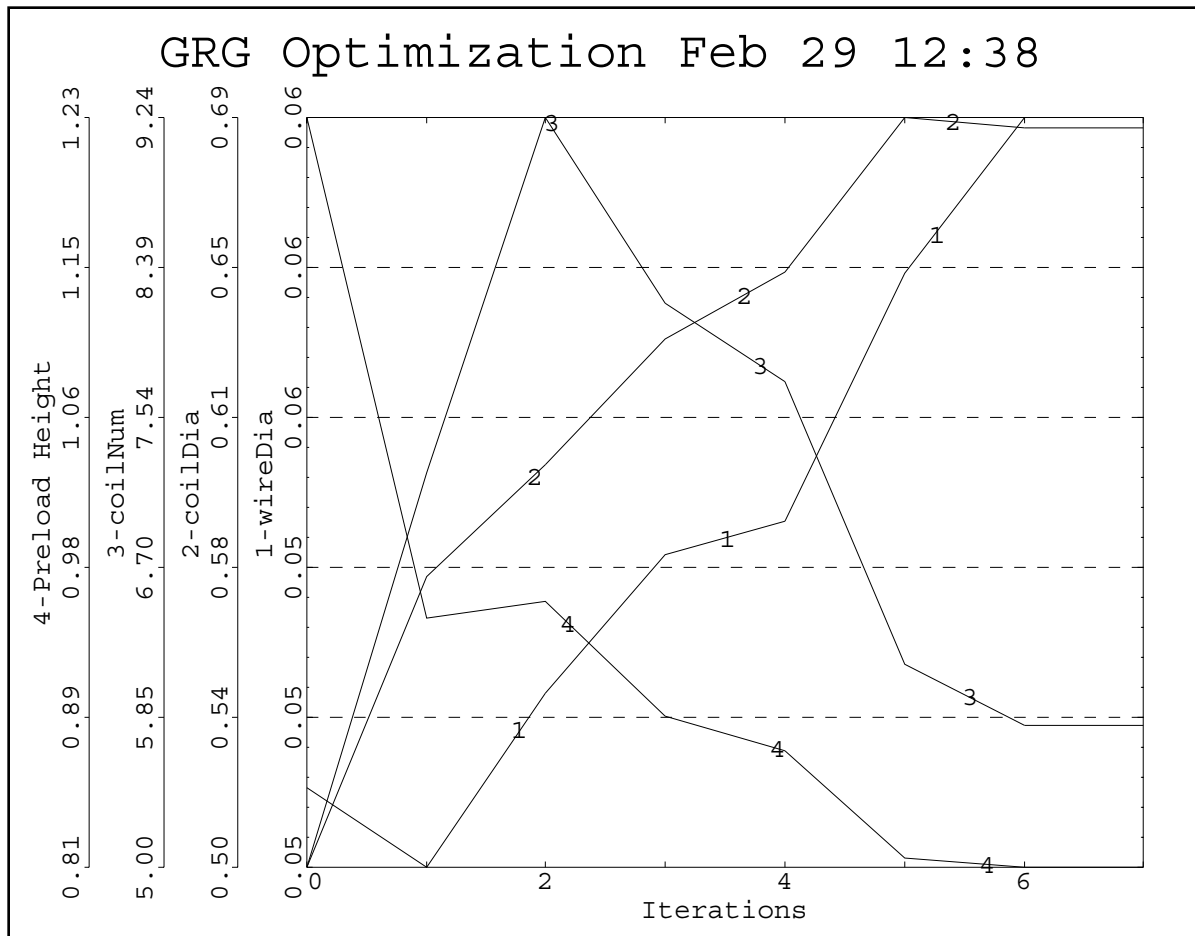
imum, the steplength will usually become small, until at the final step it is close to zero.

The maximum inequality and/or maximum equality are measures of the feasibility of the design; in scaled space constraints are feasible if less than 0.0. Thus the maximum value gives the maximum violation of the constraints in scaled space. For the example summary text no constraints were binding at Iteration 0 (the starting point) as indicated by the Scaled Max Inequality value of -0.8397 . At the end of Iteration 1, at least one constraint was binding since the Scaled Max Inequality is now $-1.394e-5$, which is zero within the limits of the constraint tolerance.

The number of analysis calls is the total number of times the functions were evaluated (including calls for gradients) and is a measure of the computational expense of the optimization. The number of gradient calls is a similar measure; GRG and SQP usually require one gradient call per iteration. Each gradient call requires n analysis calls (n =no. of design variables) for a forward difference derivative and $2n$ analysis calls for a central difference derivative.

8.5.6 History Plot

During optimization, GRG will write a "History" file that contains a record of the optimization. The parameters stored in a history file include the parameters written in the Summary window, explained above, and all optimization variables and functions. The History file can be plotted in the Graph window.



This particular history file shows how four design variables changed during an optimization.

8.6 Sequential Quadratic Programming Algorithm

8.6.1 Description and Performance

The Sequential Quadratic Programming Algorithm is used to solve continuous problems. The SQP algorithm in OptdesX is patterned after Powell's description (see Powell, M.J.D. "Algorithms for Nonlinear Functions that Use Lagrangian Functions," *Math. Programming*, vol. 14, 1978, pp. 224-228).

One view of the SQP algorithm is that it applies the Newton-Raphson method of solving nonlinear equations to solve for the solution point of the mathematical conditions that hold at a constrained optimum. These conditions are called the "Kuhn-Tucker" conditions. If we have the optimization problem,

$$\begin{array}{ll} \text{Min} & f(\mathbf{x}) \\ \text{s.t.} & g_i(\mathbf{x}) - b_i \geq 0 \quad i = 1, \dots, k \\ & g_i(\mathbf{x}) - b_i = 0 \quad i = k+1, \dots, m \end{array}$$

where \mathbf{x} is the vector of design variables, then at an optimum, \mathbf{x}^* , the following conditions will be satisfied,

all constraints will be feasible

$$\begin{array}{ll} \nabla f(\mathbf{x}^*) - \sum_{i=1}^m \lambda_i^* \nabla g_i(\mathbf{x}^*) = 0 & \\ \lambda_i^* [g_i(\mathbf{x}^*) - b_i] = 0 & i = 1, \dots, k \\ \lambda_i^* \geq 0 & i = 1, \dots, k \\ \lambda_i^* \text{ unrestricted} & i = k+1, \dots, m \end{array}$$

where λ_i is the Lagrange multiplier for constraint i . It turns out that the Newton-Raphson iteration to solve this set of equations is the same as the solution to a particular QP problem. Thus at every iteration an internal QP solver is called to solve the QP problem.

The solution to the QP problem, which is in terms of a change in the design variables, can be viewed as a search direction. A step with a steplength of 1.0 is made in this direction; if a penalty function composed of the objective and violated constraints is reduced at this new point the iteration is finished. Otherwise the step is cut back until the penalty function is reduced.

The K-T conditions given above require the constraints to be satisfied and the objective to be at an optimum. As the algorithm converges, constraint violations and the objective are reduced simultaneously. Thus feasibility of the constraints is only guaranteed at the optimum. However, because SQP does not require the constraints to be satisfied at intermediate iterations, it is also very efficient in terms of the number of analysis calls required to reach an optimum.

8.6.2 Storage Requirements

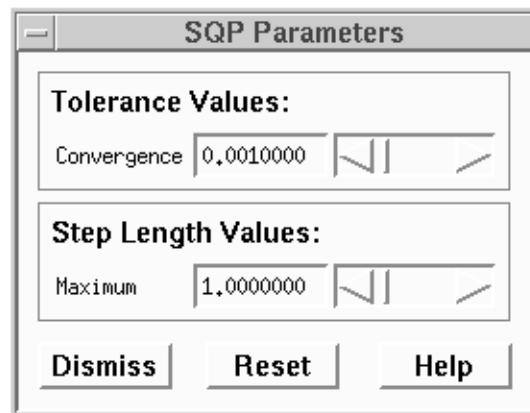
The SQP algorithm requires approximately the following amount of double precision storage:

$$\text{storage} = 3n_v^2 + 19n_v + 4n_f + n_v * n_f$$

where n_v is the number of design variables and n_f is the number of design functions. If you exceed allocated storage, an error message is printed (the SQP algorithm is written in Fortran, so storage cannot be allocated dynamically). The default size for double precision storage is 40000. Contact Design Synthesis if you need a version of the software with more storage.

8.6.3 Parameters

The default parameter values of SQP are set to provide a good compromise between solution accuracy and computational expense. Do not change these values arbitrarily. The SQP parameters are accessed by pressing the Parameters pushbutton when the SQP algorithm is selected. The following window is opened:



Convergence Tolerance

This is the tolerance, in scaled space, that SQP must satisfy before it terminates. The tolerance includes both a measure of the violation of any constraints and a prediction of the expected improvement in the objective function. Usually when this tolerance is satisfied, the Kuhn-Tucker conditions are also satisfied. Reducing the tolerance will force tighter constraint satisfaction and greater objective function accuracy, at the expense of more function calls.

Maximum Steplength

Sometimes the steplength of SQP is quite large, particularly at the beginning of an optimization where it is also less accurate. This can cause the design to become highly infeasible. This parameter limits the maximum steplength of SQP to be this value. This is a scaled value--since the variables range from -1 to +1 in scaled space, this represents a relatively large step.

8.6.4 Stopping Messages

SQP only has one stopping message:

“OPTIMUM REACHED -- CONVERGENCE TOLERANCE SATISFIED.”

When this message is printed the algorithm is usually at a local optimum.

8.6.5 Optimization Summary

The optimization summary for SQP is the same as for GRG. Please refer Section 8.5.5 for a description of the summary information.

8.6.6 History Plot

The SQP history file is the same as the GRG history file. Please refer to Section 8.5.6 for a complete description of this file and graph.

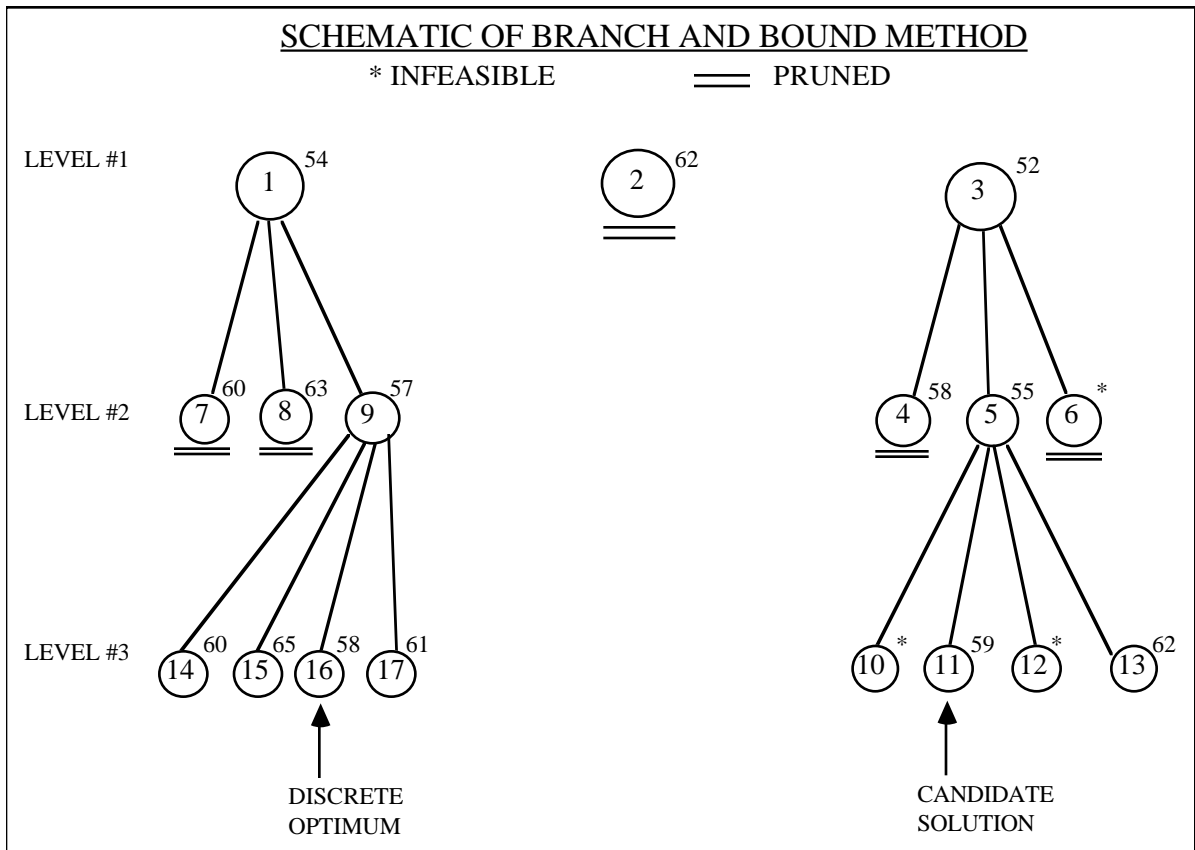
8.7 Branch and Bound Algorithm

8.7.1 Description and Performance

The **B**ranch **a**nd **B**ound strategy works by developing a tree structure. Initially, at the root of the tree, only one discrete variable is allowed to take on discrete values: other discrete variables are modeled as continuous. At each level in the tree one more discrete variable is made discrete. The various combinations of values for discrete variables constitute nodes in the tree. We start by progressing down the tree according to the discrete variable combinations that appear to be the best. At each node, an optimization is performed for the continuous variables and those discrete variables modeled as continuous at that node. Assuming we are minimizing, the objective value of this optimization problem then becomes a lower bound for any branches below that node, i.e. the objective value will underestimate (or, at best, be equal to) the true value of the objective since, until we reach the bottom of the tree, some of the discrete variables are allowed to be continuous for the optimization. Once a solution has been found for which all discrete variables have discrete values (we reach the bottom of the tree), then any node which has an objective function higher than the solution in hand can be “pruned,” which means that these nodes are not considered further.

As an example, suppose we have 3 discrete variables: variables 1 and 2 have 3 possible discrete values and variable 3 has 4 possible discrete values. A branch and bound tree is illustrated on the next page.

In this tree, “Level #1” represents the nodes where variable 1 is allowed to be discrete and variables 2 and 3 are continuous. For Level #2, variables 1 and 2 are discrete; only variable 3 is continuous. In Level #3, all variables are discrete. The numbers in the circles, where a circle represents a node, show the order in which the nodes were evaluated. The number shown at the upper right of each circle is the optimum objective value for the optimization problem performed at the node. An asterisk means no feasible solution could be found to the optimization problem



At the first level, three optimizations are performed with variable 1 at its 1st, 2nd and 3rd discrete values (variables 2 and 3 continuous). The best objective value was obtained at node 3. This node is expanded further. Nodes 4, 5, 6 correspond to variable 1 at its 3rd discrete value and variable 2 at its 1st, 2nd and 3rd discrete values respectively, with variable 3 continuous. The best objective value for nodes 1, 2, 4, 5, and 6 is at node 1, so it is expanded into nodes 7, 8, and 9. Now the best objective value among unexpanded nodes is at node 5, so it is expanded. Nodes 10, 11, 12, 13 correspond to variable 1 at its 3rd discrete value, variable 2 at its 2nd discrete value, and variable 3 at its 1st, 2nd, 3rd, and 4th values. The best objective value, 59, is obtained at node 11. This becomes the temporary optimal solution. Any nodes with objectives higher than 59 can automatically be pruned since the objective only increases as we go down the tree from a node and make more and more variables discrete. Thus nodes 2, 7, 8, and 13 are pruned. Node 9, however, looks promising, so we expand this node. As is shown, we eventually find at node 16, with variable 1 at its 1st discrete value, variable 2 at its 3rd discrete value, and variable 3 at its 3rd discrete value, a better optimum than we had before. At this point all further nodes are pruned and we can stop.

The number of discrete combinations may be large enough to prohibit a nonlinear optimization at each node. Therefore, you can linearize the objective and constraints about the continuous optimum. Then at each node a linear optimization is done--this doesn't require any additional calls to the analysis subroutine. A nonlinear optimization is performed for the current best solution to verify actual feasibility. Linearized strategies are thus faster but involve

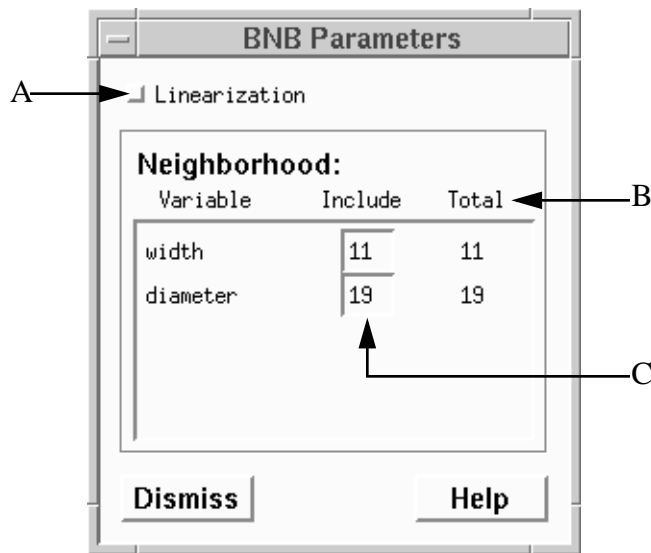
approximations which may lead to solutions which are not the true discrete optimum.

8.7.2 Storage Requirements

Storage requirements for the BNB algorithm are problem dependent. The algorithm stores the BNB tree; how big this tree gets depends on how many nodes are pruned, which cannot be predicted beforehand. If you exceed allocated storage, an error message is printed. The default size for double precision storage is 40000. This is enough storage for a tree with 2000-8000 nodes, depending on how much storage is needed for the optimization done at each node. Contact Design Synthesis if you need a version of the software with more storage.

8.7.3 Parameters

The BNB parameters are accessed by pressing the Parameters button when the BNB algorithm is selected. The following window is opened:



A - Linearization Toggle button.

B - Total value fields giving the total number of discrete values for each variable.

C - Include value fields giving the number of discrete values to be included in the search neighborhood.

Linearization Toggle Button

When set, this option causes BNB to use a linear approximation to the design functions rather than call the analysis model. This can greatly reduce computation. The linear Taylor's approximation is based on function values and gradients evaluated at the BNB starting point. All optimizations in the BNB tree are then done with linear approximations and solved with the simplex method, rather than GRG. If, based on the approximations, a candidate solution is found, a nonlinear optimization is done to verify the feasibility of the design. As with any approximation, there is some error involved which is dependent on both the analysis model and the size of the discrete neighborhoods chosen. The error in the approximation may lead BNB to evaluate a somewhat different tree than the true one, giving a different discrete optimum. Typically in large combinatorial problems, however, there are many discrete solutions that are close in value, so this is usually not serious.

Neighborhoods

After discrete files are read in, the number of discrete values for each discrete variable is shown in this table. You can reduce the size of the BNB tree by restricting the search to a “neighborhood” around the current point (which is presumably the continuous optimum or some other reasonable starting design). The number of combinations to be examined can thus be drastically reduced, with the trade-off that potentially promising combinations outside the neighborhoods are eliminated from consideration.

8.7.4 Stopping Messages

The BNB algorithm only has two stopping messages,
 “BRANCH AND BOUND SEARCH COMPLETED”

or “Error: No Solution to Branch and Bound”

The first message is printed when the entire tree has been evaluated. The second is printed if the tree has been evaluated and no solution exists.

8.7.5 Optimization Summary

Example text from an Optimization Summary window for a BNB run follows.

```

NODE NUMBER          4
  1= 10,   2=  0
NON:Feasible -- Scaled Obj =   2.3143

NODE NUMBER          5
  1=  9,   2=  0
NON:Feasible -- Scaled Obj =   2.5442

NODE NUMBER          6
  1= 11,   2=  0
NON:Feasible -- Scaled Obj =   2.0982

NODE NUMBER          7
  1= 12,   2=  6
NON:Feasible -- Scaled Obj =   1.9440
  Scaled Step Length      = 0.2218415
  Scaled Objective        = 1.943974
  Number Of Analysis Calls = 174
  Number Of Gradient Calls = 4
This Becomes Current Solution

NODE NUMBER          8
  1= 12,   2=  7
NON:Feasible -- Scaled Obj =   1.8096

```

Some explanation is required to interpret this data. The algorithm prints the node number of the tree, values of the discrete variables at the node, and the results of a nonlinear or linear optimization at the node. Nodes are numbered consecutively in order of evaluation. The values of the

discrete variables are printed using a format such as,

```
1 = 6, 2 = 0
```

which means discrete variable number 1 (which would be the first variable in the discrete variable list in the Variables window) is set to the value given by the 6th row in the corresponding file; the zero for discrete variable 2 means it was continuous at this node.

The evaluation of the node is given in the next line of information, such as,

```
NON:Feasible -- Scaled Obj = 1.1427
```

This indicates a nonlinear optimization (“NON”) was done at the node, and the result was a feasible design with the scaled objective given. For BNB, if this objective is less than the best objective so far, this becomes the next node that is expanded to make a new level in the tree. If this objective is the best and all discrete variables are at discrete values, the message is printed, “This Becomes Current Solution.”

If the linearization option is turned on, a message such as,

```
LIN:Feasible --Scaled Obj = 2.143
```

is printed, which gives the result of the linear optimization. Sometimes both are printed:

```
LIN:Feasible -- Scaled Obj = 2.143
NON:Feasible -- Scaled Obj = 2.055
```

This indicates that the result of the linear optimization was good enough to be considered a candidate solution, so a nonlinear optimization was done to confirm the linear optimum.

8.7.6 History Plot

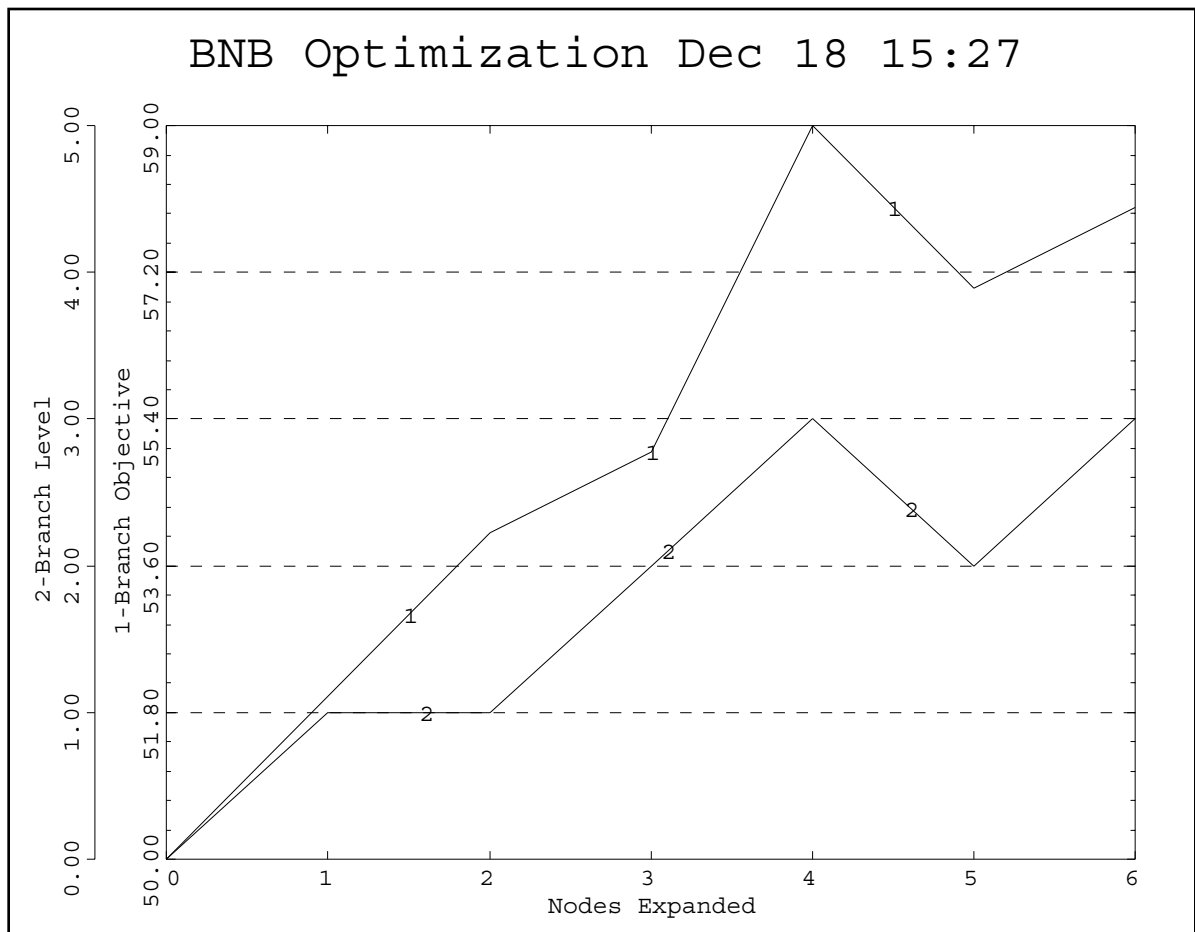
An example history plot is shown below for the BNB tree given in Section 8.7.1. We will refer back to this tree to explain this history plot. The beginning data point, Nodes Expanded 0, with an objective of 50, and a branch level of 0, is the objective value when branch and bound was started. It is given for reference.

To understand the second data point, we need to define the label for the abscissa, “Nodes Expanded.” A node is expanded by developing nodes below it. A node is selected to be expanded because it has the lowest objective of all the nodes developed to that point. In terms of the example, the first node to be expanded is node 3, with a cost of 52. The node is expanded by evaluating nodes 4, 5, 6. Thus on the history plot Nodes Expanded 1 refers to node 3. Its objective, 52, and branch level, 1, are plotted. The second node to be expanded is node 1, with an objective of 54. It is expanded by evaluating nodes 7, 8, 9. This node is also on the first level of the branch and bound tree. This data is plotted at Nodes Expanded 2. Now we jump back to node 5, which is the third node expanded. Its objective of 55 and branch level of 2 are plotted.

When node 5 is expanded, we evaluate nodes 10, 11, 12, 13. This gives us four discrete solutions. Of these, node 11 is the best, so it is selected as a candidate discrete solution. It is plotted as Nodes Expanded 4. We can infer from the plot that this data point represents a discrete solution, since it occurs at branch level 3, and we only have 3 discrete variables. Now that we have a discrete solution in hand, we can prune the tree of any nodes with higher objectives. From this point on, the only nodes that are expanded are those with lower objectives than Node Expanded 4--as indicated by the dip the plot takes after this point. The final node, Node Expanded 6, is the discrete solution.

Thus the BNB history plot is a very abbreviated summary of the progress of the algorithm. It shows the objective and level of each node that is expanded. The objective value will gradually increase until a candidate solution is found, at which point all nodes with worse objectives are pruned. Any nodes expanded after a discrete solution is found will have an objective value less than or equal to this solution.

The branch level gives you some idea of where the node is in the tree. If the branch level stays the same or decreases from one point to another (as it does in the example from Nodes Expanded 1 to 2), it means BNB jumped to a different branch on the tree. When the branch level equals the number of discrete variables, the point plotted is a candidate solution.



8.8 Simulated Annealing Algorithm

8.8.1 Description and Performance

Simulated ANnealing is used to solve pure discrete problems. Design variables that are not made discrete variables remain constant during the execution of this algorithm.

SAN models a phenomenon in nature--the annealing of solids--to optimize a complex system. Annealing refers to heating a solid to the liquid state and then cooling it slowly so that thermal equilibrium is maintained. Atoms then assume a globally minimum energy state. In 1953 Metropolis created an algorithm to simulate the annealing process. The algorithm makes small random displacements of atoms, causing a change in energy. If the change is negative, the energy state of the new configuration is lower and the configuration is accepted. If the change is positive, the new configuration has a higher energy state; however, it may still be accepted according to the Boltzmann probability factor:

$$P = \exp\left(\frac{-\Delta E}{k_b T}\right)$$

where ΔE is the energy change, k_b is the Boltzmann constant and T is the temperature. The Boltzmann probability is compared to a random number between 0 and 1 drawn from a uniform distribution; if it is higher, the configuration is accepted.

When applied to engineering design, an analogy is made between energy and the objective function. The design is started at a high “temperature,” where it has a high objective value. Random perturbations are then made to the design. If the objective is lower, the new design is made the current design; if it is higher, it may still be accepted according to the probability given by the Boltzmann factor. This allows the algorithm to escape local minima. As the temperature is gradually lowered, the probability that a worse design is accepted becomes smaller. Typically at high temperatures the gross features of the design emerge which are then refined at lower temperatures.

Although it has been applied to continuous problems, Simulated Annealing is especially effective when used for combinatorial or discrete optimization. The algorithm is not guaranteed to find the best optimum; however, it will often find near optimum designs with many fewer design evaluations than other algorithms.

In the OptdesX implementation of SAN, you do not specify a starting “temperature” for the design. Rather, you specify the probability that a worse design will be accepted at the beginning and end of the annealing process. These values then set what the appropriate temperatures should be. Typically the probability of accepting a worse design at the beginning of the optimization is on the order of 0.5; the probability of accepting a worse design at the end should be small--on the order of 1.e-6.

You also must specify the number of cycles the algorithm should run. The number of cycles corresponds to the number of temperatures and defines how quickly or slowly the design is cooled. The selection of the appropriate number of cycles is sometimes a matter of trial-and-error. A high number of cycles better simulates the annealing process but can require excessive

computation. For each cycle, each variable is perturbed in turn randomly; the design is evaluated for acceptance or rejection after every perturbation. Thus each cycle is composed of ndv design evaluations, where ndv is the number of design variables.

For design problems that have differentiable models, you can turn on a filter parameter that filters out a percentage of perturbations that are predicted by a Taylor's approximation to be worse than the current design. This can greatly reduce the number of analysis calls. This feature is explained fully in the next section.

If SAN is started at an infeasible point, the maximum constraint violation, instead of the objective function, is reduced until a feasible design is found or the specified number of cycles is reached. As soon as a feasible design is identified, the algorithm starts over again and begins working on the objective.

Further information on SAN can be found in, Kirkpatrick, S., C.D. Gelatt, and M. P. Vecchi, "Optimization by Simulated Annealing," *Science*, vol. 220, no. 4598, May 83, p. 671. Bohachevsky, I., M. Johnson, and M. Stein, "Generalized Simulated Annealing for Function Optimization," *Technometrics*, vol. 28, no. 3, August 86, p.209; Aarts E. and J. Korst, *Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing*, Wiley, 1989.

8.8.2 Storage Requirements

The SAN algorithm is written in C; storage is allocated dynamically. The algorithm requires very little storage, so you should never have a problem running out.

8.8.3 Parameters

The SAN parameters are accessed by pressing the Parameters pushbutton when the SAN algorithm is selected. The following window opens:

The image shows a dialog box titled "SAN Parameters" with the following fields and values:

Section	Parameter	Value
Discrete Perturbation:	Maximum	1
Prob. Accepting Worse Design:	Starting	0.5000000
	Final	0.0000010
Tolerances:	Constraint	0.0010000
	Filter	1.0000000

Buttons at the bottom: Dismiss, Reset, Help

Maximum Discrete Perturbation

At each “temperature” or cycle of the SAN algorithm, each discrete variable of the design is perturbed randomly in turn. After each perturbation, the new candidate design is accepted or rejected. This parameter sets the maximum perturbation. The default of 1 means that each variable will be perturbed from its current value to the next lowest or highest discrete value. A value of 2 means each variable will be perturbed up or down 1 or 2 closest discrete values from the current value.

Probability of Accepting a Worse Design

Starting--This is the starting probability that a perturbed design with a worse objective value will be accepted and become the current design.

Final--As SAN proceeds and the design “freezes,” the probability of replacing the current design with a worse design gradually approaches the final probability until at termination it is equal to it. Typically this is a small value.

Constraint Tolerance

In scaled space a constraint is considered satisfied if its scaled value is less than zero (a scaled value of zero means the constraint is at its allowable value) plus this tolerance. Note that this allows a constraint to be slightly violated within the bounds of this tolerance.

Filter Tolerance

The filter tolerance is associated with a modification to SAN that allows it to use gradient information to filter out many poor candidate designs. A value of one gives regular SAN. When less than one, gradients are evaluated at the start of SAN and combined with the current design to construct an approximation to the objective function and constraints. The approximations are then evaluated for a proposed random perturbation. If the approximation predicts the perturbation to result in a feasible, better design, the design is automatically passed through the filter. If the approximation predicts the perturbation to be worse, either with respect to feasibility or objective value, the perturbed design is rejected if a randomly generated uniform deviate is greater than the filter value. Thus with a filter value of 0.1, only 10% of the predicted worse designs will be passed through the filter.

The rest of SAN is unchanged: worse designs still must pass the Boltzmann probability check.

8.8.4 Stopping Messages

SAN runs until the specified number of cycles are completed. The design it has at completion is by default the optimum. When finished SAN merely reports,

“SIMULATED ANNEALING COMPLETED”

8.8.5 Optimization Summary

Example text from the Optimization Summary window for a SAN run looks like,

OPTIMIZATION PHASE

Cycle = 1

Scaled Obj = 1.308038, Accept Prob = 0.500000

```
Cycle = 2
  Scaled Obj = 1.308038,   Accept Prob = 0.489477

Cycle = 3
  Scaled Obj = 1.144533,   Accept Prob = 0.478863

Cycle = 4
  Scaled Obj = 1.347344,   Accept Prob = 0.468164

Cycle = 5
```

The summary includes a statement of the phase of the algorithm, the cycle number, the scaled objective value and the acceptance probability.

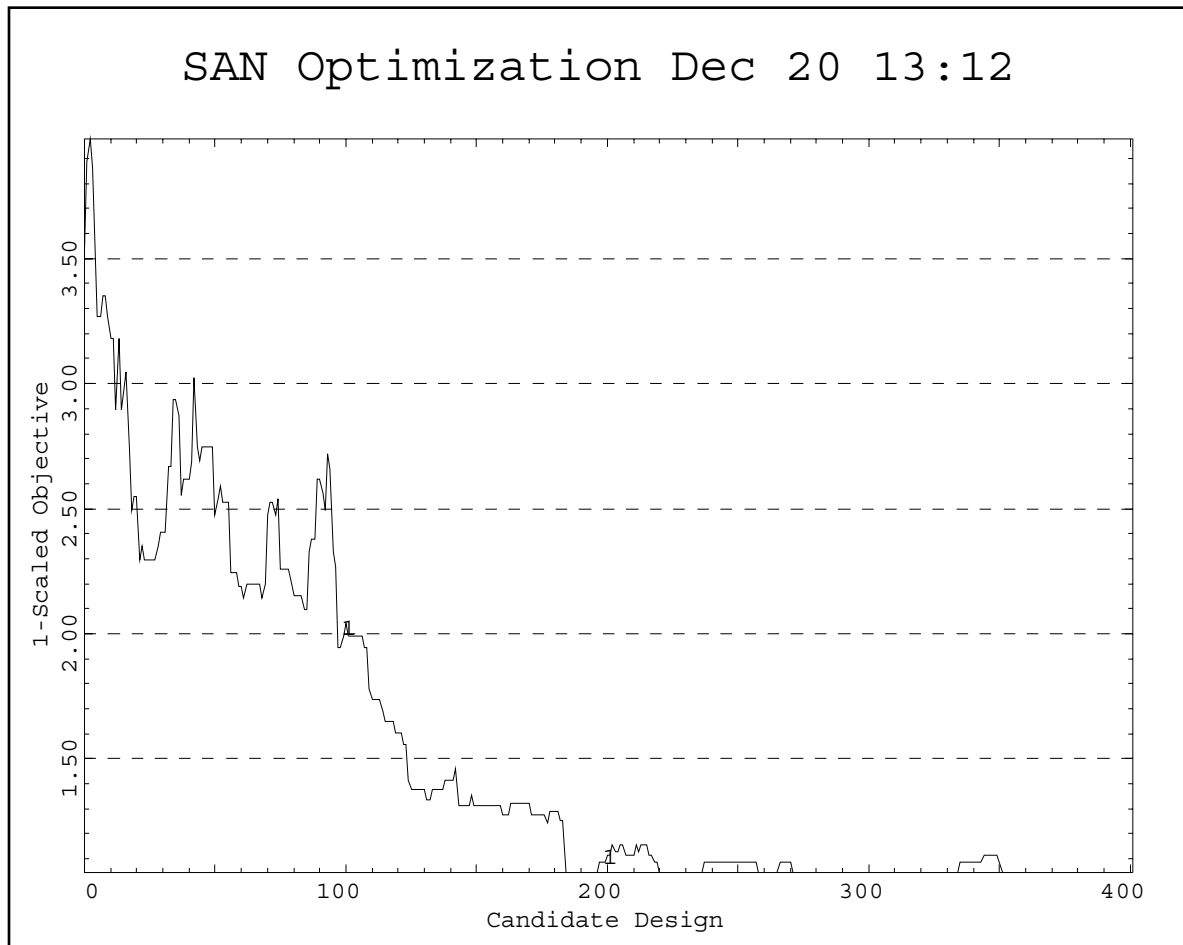
The SAN algorithm has two phases: Feasibility phase, where a feasible design is sought by minimizing the maximum constraint violation at each cycle, and Optimization phase, where the objective is reduced. If the starting design is feasible, there is no Feasibility phase.

Each cycle represents a temperature. At each temperature, each design variable is perturbed. For each perturbation the design is evaluated as a candidate to replace the current solution. The objective shown is the scaled objective value at the end of all perturbations. The acceptance probability is the probability that a perturbed design that is worse than the current design will be accepted and become the new current solution. This probability decreases as the optimization proceeds.

8.8.6 History Plot

An example history plot is given below. Two parameters can be plotted: the objective value and the maximum violation of the constraints. The maximum violation is interesting only if the starting design is infeasible.

The abscissa is given as “Candidate Designs.” A candidate design is a design that is accepted as the current solution. In general as a design freezes worse designs will be accepted less often than in earlier cycles. Since there are ndv (ndv = no. of design variables) designs evaluated for each cycle, dividing Candidate Designs by ndv gives the number of cycles executed.



8.9 Exhaustive Search Algorithm

8.9.1 Description and Performance

The EXhaustive Search strategy examines all possible combinations of discrete variables to locate the discrete optimum. EXS can be used to solve pure discrete or mixed problems. For a mixed problem, an optimization in the continuous variables is done for every combination.

In terms of a branch and bound (BNB) tree (see Section 8.7.1), EXS examines those nodes found at the bottom of an unpruned branch and bound tree structure. If the BNB strategy executes without much pruning, more nodes could be evaluated for BNB than for EXS. In the example given for BNB, 17 nodes were evaluated; EXS would have evaluated $3 \times 3 \times 4 = 36$ combinations. The efficiency of BNB relative to EXS therefore depends on how much pruning is done.

The number of variable combinations may be so large as to prohibit a nonlinear optimization for each combination. To decrease computation, you can linearize the objective and constraints about the starting point (which is often the continuous optimum). Then for each combination a LP optimization is done--this doesn't require any additional calls to the analysis subroutine. A nonlinear optimization is performed for the current best solution to verify actual feasibility.

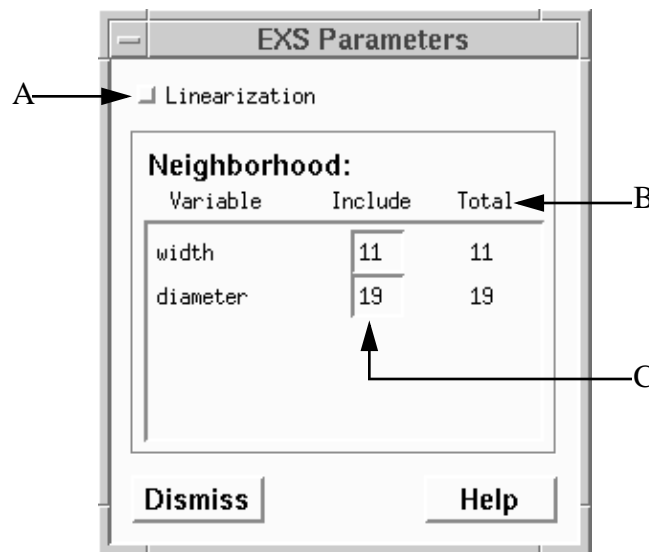
Linearized strategies are thus faster but involve approximations which may lead to solutions which are not the true discrete optimum.

8.9.2 Storage Requirements

The EXS algorithm requires very little storage. Only enough storage is needed to perform an optimization for any continuous variables in the problem. If you exceed allocated storage, an error message is printed (the EXS algorithm is written in Fortran, so storage cannot be allocated dynamically). The default size for double precision storage is 40000. Contact Design Synthesis if you need a version of the software with more storage.

8.9.3 Parameters

The parameters for EXS are set by pressing the Parameters pushbutton in the algorithms box when EXS is selected. The following window is opened:



A - Linearization Toggle button.

B - Total value fields giving the total number of discrete values for each variable.

C - Include value fields giving the number of discrete values to be included in the search neighborhood.

Linearization Toggle Button

When set, this option causes EXS to use a linear approximation to the design functions rather than call the analysis model. This can greatly reduce computation. The linear Taylor's approximation is based on function values and gradients evaluated at the EXS starting point. All optimizations are then done with linear approximations and solved with the simplex method, rather than GRG. If, based on the approximations, a candidate solution is found, a nonlinear optimization is done to verify the feasibility of the design. As with any approximation, there is some error involved which is dependent on both the analysis model and the size of the discrete neighborhoods chosen (see below). The error in the approximation may cause the algorithm to report a discrete optimum that is not the true one. Typically in large combinatorial problems, however, there are many discrete solutions that are close in value, so this is usually not serious.

Neighborhoods

After discrete files are read in, the number of discrete values for each discrete variable is shown in this table. You can reduce the combinations by restricting the search to a “neighborhood” around the current point (which is presumably the continuous optimum or some other reasonable starting design). The trade-off is that potentially promising combinations outside the neighborhoods are eliminated from consideration.

8.9.4 Stopping Messages

The EXS algorithm has two stopping messages,
 “EXHAUSTIVE SEARCH COMPLETED”

or “Error: No Solution to Exhaustive Search”

The first message is printed when all combinations have been evaluated. The second is printed when no solution exists for any of the discrete combinations.

8.9.5 Optimization Summary

Example text from an Optimization Summary Window for an EXS run is given below.

```

    Scaled Step Length      = 0.000000
    Scaled Objective        = 1.144533
    Number Of Analysis Calls = 0
    Number Of Gradient Calls = 0

COMBINATION NUMBER      1
  1= 15,   2= 11
NON:Feasible -- Scaled Obj =  1.1427
    Scaled Step Length      = 0.006297650
    Scaled Objective        = 1.142659
    Number Of Analysis Calls = 8
    Number Of Gradient Calls = 2
This Becomes Current Solution

COMBINATION NUMBER      2
  1= 15,   2= 10
NON:Infeasible

COMBINATION NUMBER      3
  1= 15,   2=  9
NON:Infeasible

COMBINATION NUMBER      4
  1= 14,   2= 11
NON:Feasible -- Scaled Obj =  1.2579

COMBINATION NUMBER      5
  1= 14,   2= 10
NON:Feasible -- Scaled Obj =  1.3082

```

Some explanation is required to interpret this data. The algorithm prints the combination number, values of the discrete variables, and the results of a nonlinear or linear optimization for the combination. Combinations are numbered consecutively in order of evaluation. The values of the discrete variables are printed using a format such as,

$$1 = 6, 2 = 3$$

which means discrete variable number 1 (which would be the first variable in the discrete variable list in the Variables window) is set to the value given by the 6th row in the corresponding file; discrete variable 2 is set to the values given by the 3rd row in the discrete file.

The evaluation of the combination is given in the next line of information, such as,

```
NON:Feasible -- Scaled Obj = 1.1427
```

This indicates a nonlinear optimization (“NON”) was done for the combination, the result was a feasible design with the scaled objective given. If this objective is less than the best objective so far, the message is printed, “This Becomes Current Solution.”

If the linearization option is turned on, a message such as,

```
LIN:Feasible -- Scaled Obj = 2.143
```

is printed, which gives the result of the linear optimization. Sometimes both are printed:

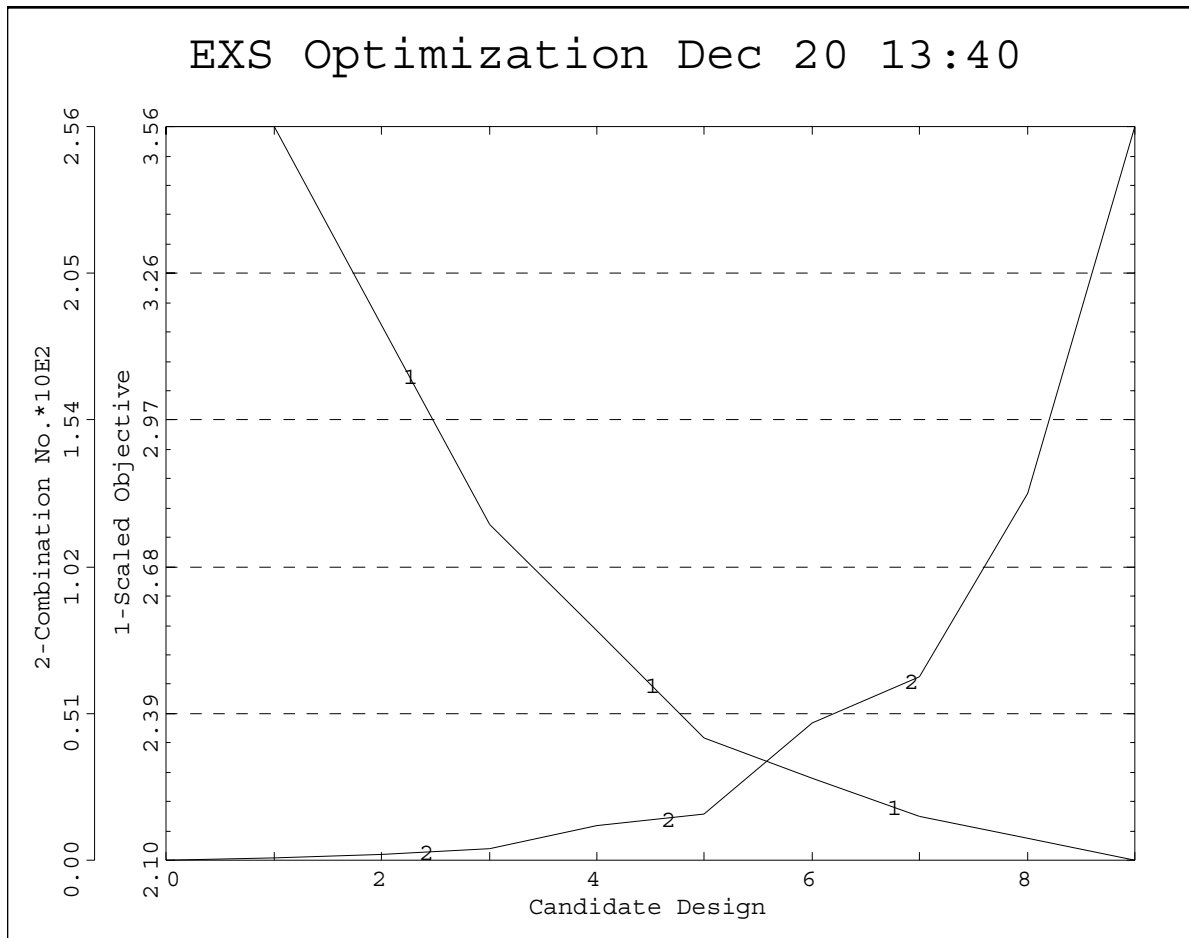
```
LIN:Feasible -- Scaled Obj = 2.143
NON:Feasible -- Scaled Obj = 2.055
```

This indicates that the result of the linear optimization was good enough to be considered a candidate solution, so a nonlinear optimization was done to confirm the linear optimum.

8.9.6 History Plot

An example EXS history plot is given on the next page. The plot shows two measures of progress for the exhaustive search--the objective value for the current solution, and the node number of the current solution.

If EXS evaluates a discrete combination that is feasible and better than any that precede it, it designates this solution as the current solution, or candidate design. This plot shows the objective values for the sequence of current solutions, as well as the combination at which they occurred.



8.10 Optimization Options Window

The Optimization Options window allows you to control the information displayed and the files written during optimization.

8.10.1 During Optimization

Continuous Update

The continuous update of variables and functions causes the windows to be updated as the optimization proceeds. This can, on some systems, entail a significant computational overhead. When turned off, variables and functions are only updated periodically, including the optimum or final design. The default is to continuously display variables and functions.

Post Process

The Post Process toggle button allows you to call ANAPOS during the optimization at iteration intervals set by you. This makes it possible, for example, to display a picture of a design or print intermediate results during optimization without stopping the algorithms. (The GRG and SQP algorithms are much more effective if run several iterations at time.)

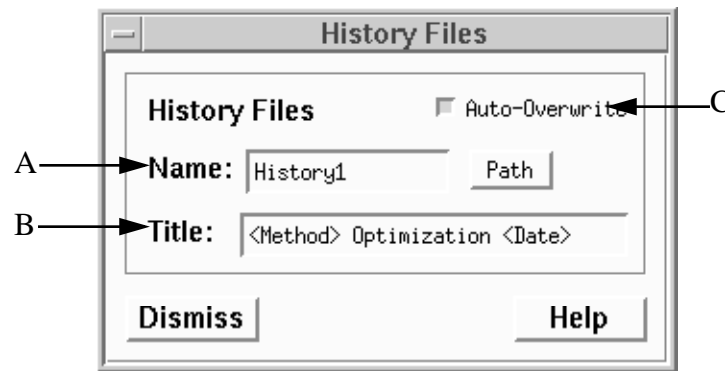
For GRG or SQP, an iteration is defined to be one cycle of finding a search direction and step-

ping in that direction. For EXS and BNB, one iteration is finding a candidate solution. Usually for BNB this occurs after much of the BNB tree has already been evaluated. For SAN, an iteration is defined to be one cycle (or one temperature) of the algorithm.

8.10.2 Files

History Files

A History file records the history of the optimization in terms of parameters such as steplength, objective value, etc. at each iteration. This file can then be graphed in Graph. The default is to write a history file for every optimization, but unless changed in "Name, Path, Title," the file is named "History" and overwrites any previous files. The History file window is shown below.



A - Name text field. If Auto-Overwrite is on (the default), a file named "History" is created during an optimization and overwrites any previous History files of that name. If Auto-Overwrite is off, a history file is created and saved for each optimization. The file names are numbered sequentially, i.e. History1, History2, etc. The History file name is limited to 16 characters.

B - Title text field. The default title includes the optimization method and the date and time. The title is limited to 60 characters, although only 30 are displayed.

C -Auto-Overwrite toggle button.

Analysis Files

The Iteration Analysis Files toggle button, when set, writes out an analysis file (containing variable and function values) at each iteration. The designs in these files can then be restored as desired. The default is not to write these files. The Iteration Analysis files window, opened by pressing the "Name, Path, Title" button, is the same as the History File window shown above.

8.10.3 Summary Window Options

During optimization a summary window opens and displays information about the optimization. The Summary toggle buttons, when set, display additional information beyond the default in the window. "Print Algorithm Information" and "Print Algorithm Diagnostics" are only available for GRG and SQP algorithms.

8.11 Optimization Summary Window

The Summary window displays results from an optimization. The Summary window can be brought to the top of all windows, if covered up, by pressing the Summary button in the Optimize window.

8.11.1 Operation

The Summary window automatically opens when an optimization is performed. This window is scrollable, so that all information can be viewed, up to a maximum limit of approximately 500,000 characters, at which point previous information is overwritten.

The information shown in the window depends on which algorithm was chosen for optimization and on the options selected in the Options window.

8.11.2 Buttons

The "Clear" button clears all information from the window. The "Hardcopy" button opens a window that lets you dump the data in the window to a file, to be printed.

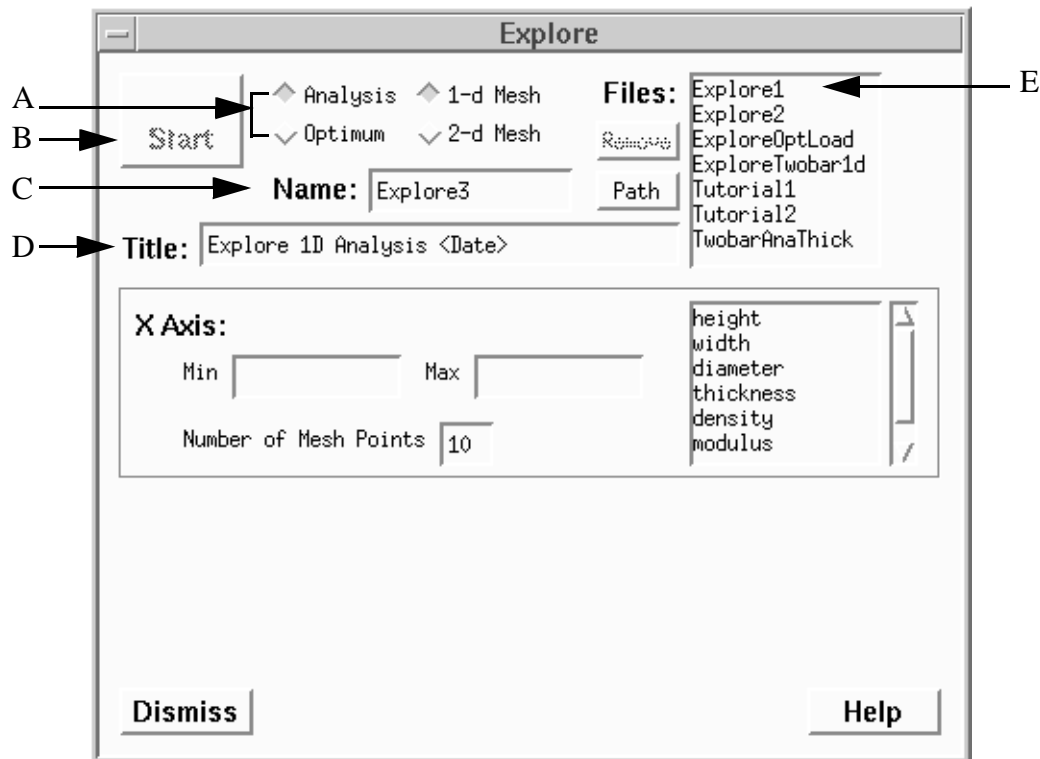
9 Explore Window Reference

9.1 Overview

The Explore window is used to mesh design space to obtain data needed to construct sensitivity and contour plots.

The “Start” pushbutton, label B in the diagram below, causes the design space to be meshed according to options selected. At each mesh point the analysis model is evaluated or an optimization is performed, depending on the state of the Analysis-Optimum Radio Box, indicated by label A. The resulting data is stored in a file which can be opened in the Graph Window and plotted. After an explore is finished, the variables and functions are set back to the design point that was current when the explore was started.

The file name and title are written into text fields pointed to by label C and D. The name is limited to 16 characters; the title is limited to 60 characters, only 40 of which can be viewed at any one time. The text field can be scrolled by placing the cursor at the side and dragging. Existing Explore files are shown in the Files list pointed to by F. After an Explore file has been written, the default name in the Name text field is updated by appending a number to it--this is done to help prevent the accidental overwriting of an existing file.



A - Analysis-Optimum radio box.

B - Start pushbutton that causes the design space to be meshed.

C - Explore File Name text field. The default name is “ExploreAna1”. Each successive default file name is incremented, i.e. “ExploreAna2”, “ExploreAna3”, etc. The file

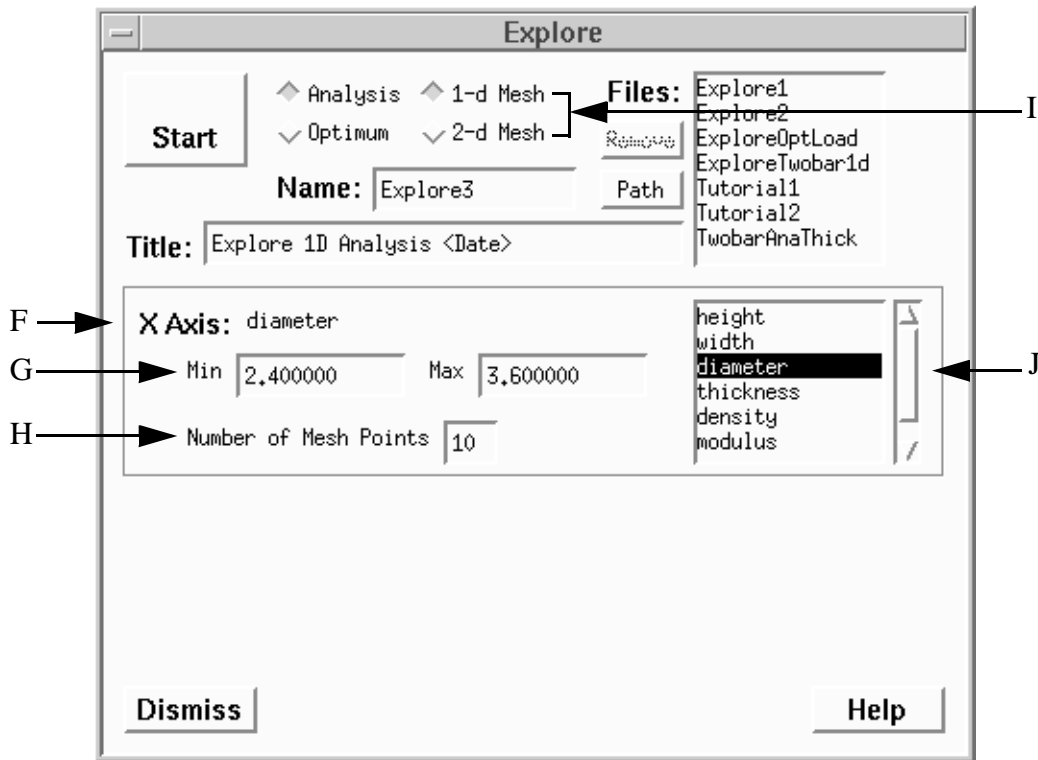
name is limited to 16 characters. An extension “.ex1” or “.ex2” is appended to the file name (but not shown) depending of the state of the 1-d, 2-d Mesh radio box.

D - Explore File Title text field. The title is limited to 60 characters. The keyword <Date> will insert the current date into the title.

E - File scrollable list used to see the Explore files already in the directory.

9.2 Explore Analysis

9.2.1 1-d Explore



F - X Axis Box used to specify the meshed variable for the X Axis.

G - Minimum and Maximum values of the range for the mesh. Defaults are taken from the Variables window, or are set to be percentage of the current value. These are editable.

H - Number of Mesh Points value field.

I - 1-d, 2-d Mesh radio box used to set the number of variables that will be meshed.

J - Mesh Variable scrollable list showing variables that can be selected as mesh variables.

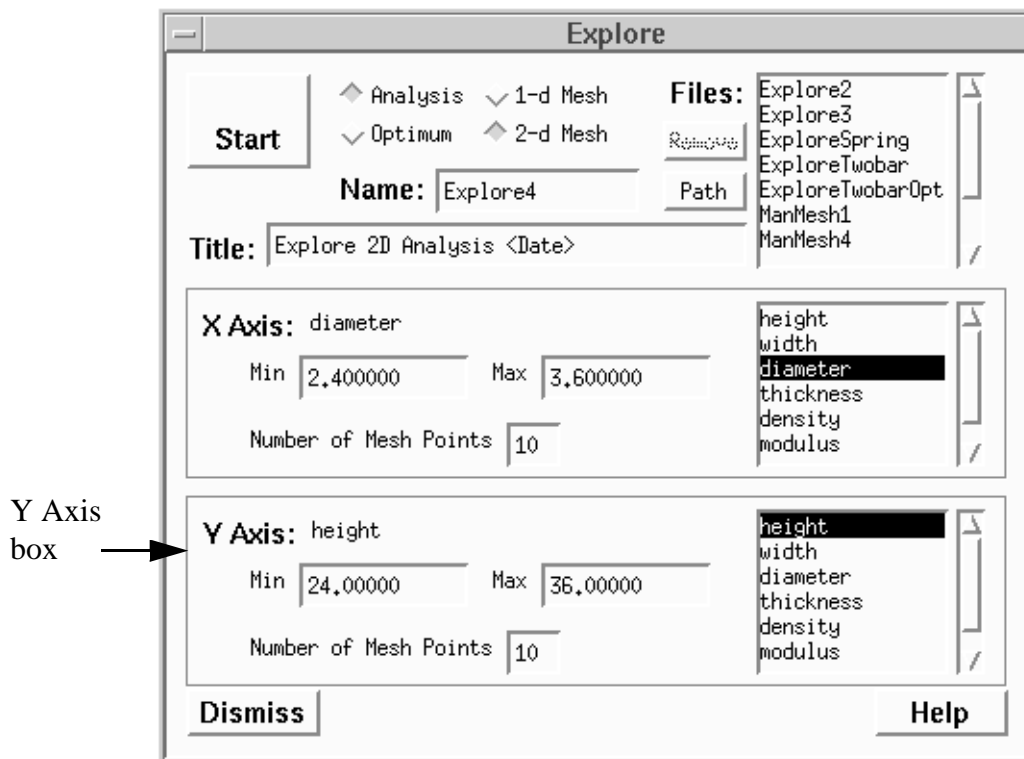
A 1-d explore analysis causes the selected variable to be meshed at even intervals between the minimum and maximum values. With all other variables held constant, the ANAFUN routine is called at each mesh point. The functions values are stored in the Explore file. This file can then be graphed in the Graph window to create a sensitivity plot.

Any design or analysis variable can be the meshed variable. The number of mesh points determines the number of analyses done--more mesh points results in a higher resolution graph.

The Min and Max values should be chosen carefully. By collapsing the Min and Max around a point, you can create a graph that “zooms in” on the point. Also, depending on the analysis model, it is sometimes possible to specify a range for which the analysis model is not entirely defined because of a divide by zero, etc. On some computer systems this situation will cause an immediate floating overflow, halting execution. On other systems, a floating overflow causes a character string such as “Infinity” or “NaN” to be written into the Explore file. The problem may not be detected until you attempt to graph the Explore file in the Graph window, at which point OptdesX will print an error message. The only recourse when this situation occurs is to regenerate the Explore file with a better range, or to manually edit the Explore file, which is not particularly easy (See Section 9.4 for a description of this file).

9.2.2 2-d Explore

A 2-d explore is used to create a 2-d contour plot showing how functions change with respect to two variables. When the 1-d, 2-d radio box is toggled to 2-d, a Y axis box opens up that is similar to the X Axis box:



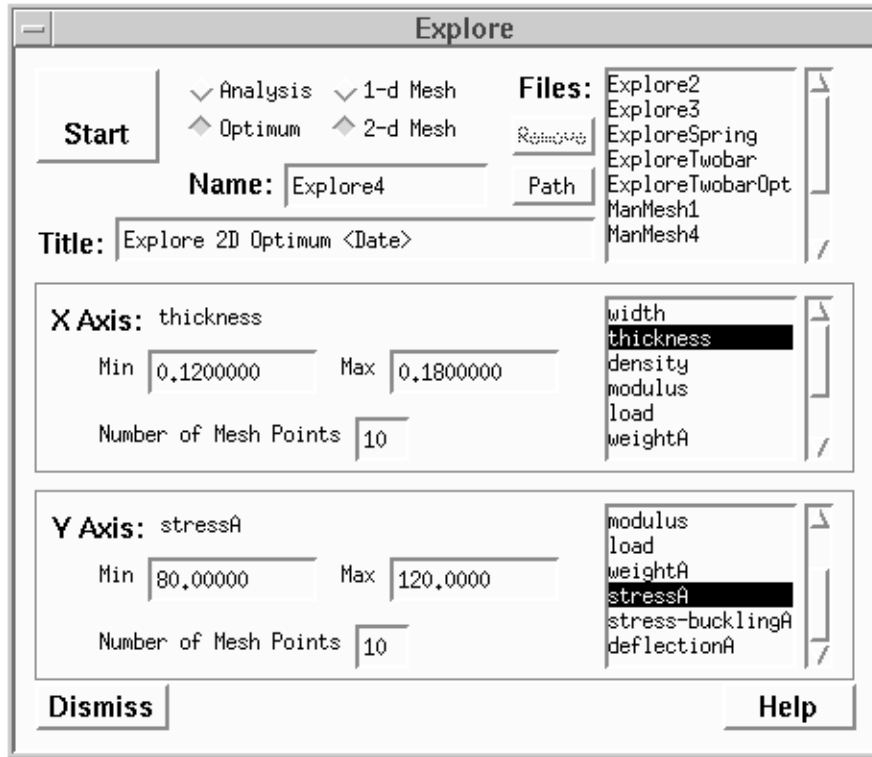
Everything in this box is the exact counterpart to the X Axis box. The same list of variables appears in the Y Axis list; you can choose any variable, except the one chosen for the X Axis. The number of analyses for a two-dimensional explore is equal to the product of the number of mesh points for the X and Y axes.

9.3 Explore Optimum

When the Analysis-Optimum radio box is set to “Optimum,” an optimization is performed at each mesh point. Thus an optimum explore shows how the *optimal* design changes as a con-

stant to the optimization problem is varied. Just as with explore analysis, you can do a 1-d or 2-d explore optimum.

The main difference in the window is in the variables that show up in the X and Y axis lists:



The mesh variables for an explore optimum must be constants with respect to the optimization problem. Obviously a design variable cannot be a mesh variable, since the design variables will be adjusted independently by the optimization algorithm to find the optimum at every mesh point.

There are two sets of mesh variables for an explore optimum: unmapped analysis variables and allowable values for constraints. Allowable values for constraints are given names comprised of the constraint function name with an “A” appended to it. In the example above, “stressA,” is the allowable value for the stress constraint.

If the definition of the optimization problem is changed in the Variables or Functions window, the lists in the Explore window are automatically updated when Explore is activated.

There is an important interaction between the Explore and Optimize windows when an explore optimum is performed:

- 1) the algorithm used for the explore optimum is the algorithm chosen in the Optimize window. Any algorithm appropriate for the problem can be used.
- 2) the number of iterations/combinations/cycles run is taken from the Optimize win-

dow. If the selected algorithm is GRG or SQP, you should select enough iterations to make sure the algorithm does not terminate early, i.e., terminate because it has reached the number of iterations limit. A rule of thumb is to set the number of iterations to 25 times the number of design variables. This is usually enough to make sure the iteration limit is not reached.

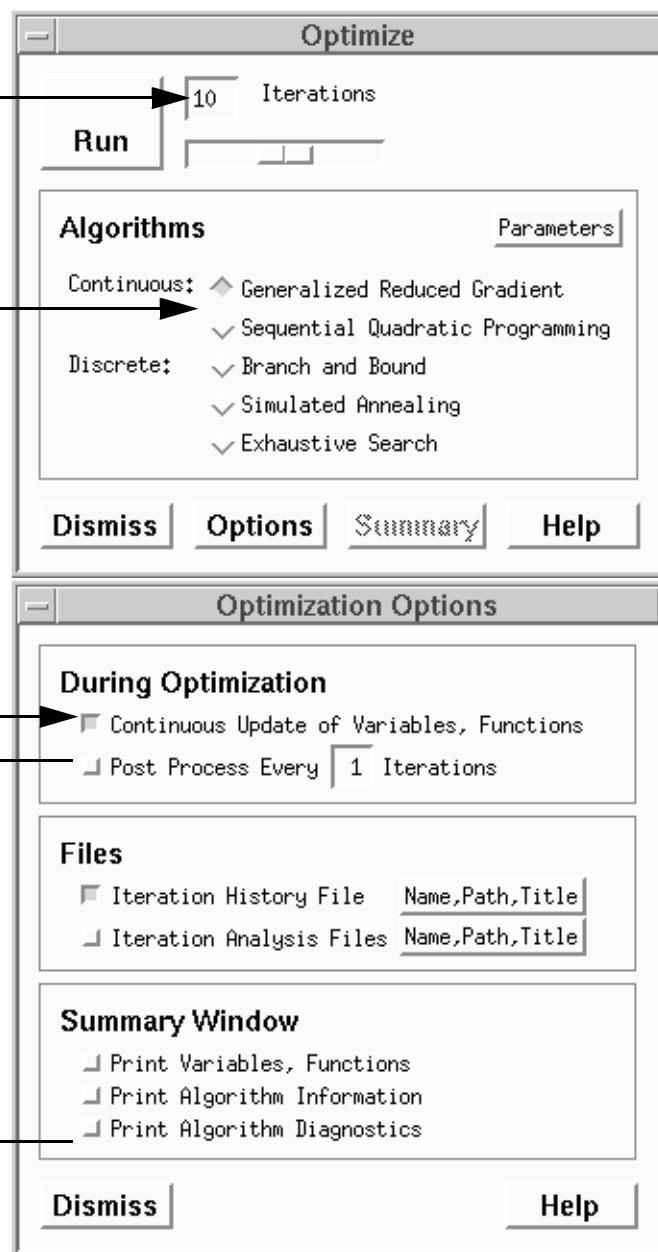
- 3) the variable and function values are not displayed if the Continuous update toggle is turned off in the Optimization Options window. Usually the Continuous update should be turned off as this makes an explore optimum run up to 100 times faster.
- 4) the other toggles in the Optimization Options window are not operative--History and intermediate Analysis files cannot be selected, and the Summary Window information does not apply.

The number of iterations should be set to a large enough number that it will not be reached.

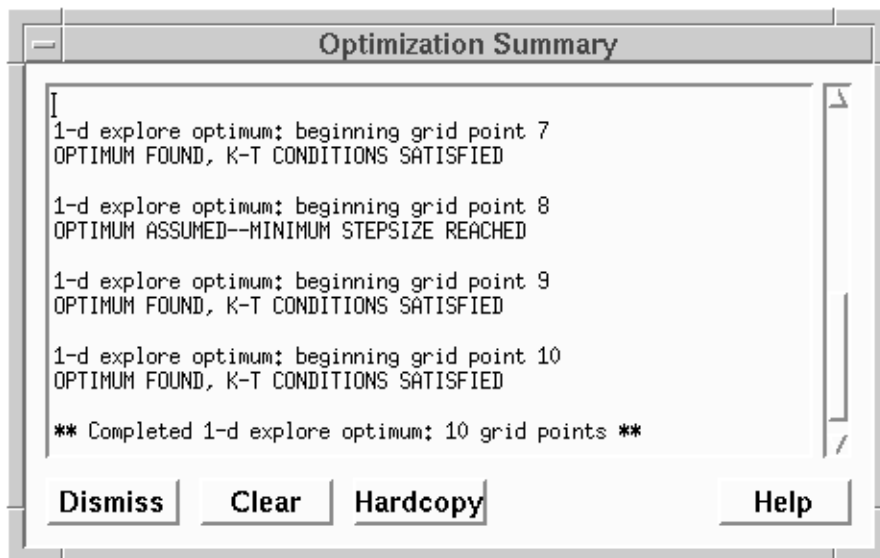
The optimization algorithm chosen here is the algorithm that is used for an explore optimum

The continuous update of variables should normally be turned off--the explore optimum will run much faster.

These options are not available during an explore optimum



During an explore optimum, an Optimization Summary window will open and display brief information about each optimization:



An explore optimum has its risks! First of all, it obviously will take much more computer time because an optimization is done at each mesh point. Secondly, the software does not discriminate between optimizations that end successfully and those that do not. For example, if a feasible point cannot be found, the design point at the end of the optimization (which is infeasible) is entered in the Explore file. You should select ranges carefully, making sure they are not too extreme. After an explore optimum you should check to make sure each optimization terminated successfully.

When data from a 1-d explore optimum is graphed, it does not usually make sense to plot a constraint that is always binding. Such a constraint only varies by plus or minus the constraint tolerance about the allowable value; this variation is an artifact of the optimization and is not meaningful.

Similarly, for a 2-d explore optimum it only makes sense to plot the objective. By definition each of the mesh points is feasible, so we can't show boundaries that demarcate feasible and infeasible space. Yet if we choose to plot boundaries they will often show up, in some sort of nonsensical pattern, only because the constraint tolerance allows a constraint to be slightly violated within the bounds of the tolerance. Again, this variation is an artifact of the optimization and is not meaningful. Examples of these nonsense plots are given in the Section 10, "Graph Window Reference."

9.4 Explore Files

Listings of an example 1-d and 2-d Explore file are given below.

1-d Explore file:

Twoobar 1D Explore-Diameter	←	The file title
diameter	←	The variable to be meshed.
1.00000000000000e+00 3.00000000000000e+00 10	←	The Min and Max and no. of mesh points.
height 3.00000000000000e+01	}	These are names and values of variables that were held constant during the explore (diameter is ignored in this list).
diameter 3.00000000000000e+00		
width 6.00000000000000e+01		
thickness 1.50000000000000e-01		
density 3.00000000000000e-01		
modulus 3.00000000000000e+04	}	
load 6.60000000000000e+01		
###	←	File marker
weight	}	Names and order of function values.
stress		
stress-buckling		
deflection	←	
###	←	File marker
1.1995784110239e+01	}	Function values follow next in blocks of four.
9.9034793314258e+01		
7.8010479151177e+01		
1.9806958662852e-01		
1.4661513912515e+01	}	Next set of function values.
8.1028467257120e+01		
4.9850238801145e+01		
1.6205693451424e-01		
1.7327243714790e+01		
6.8562549217563e+01		
2.5199623610114e+01		
1.3712509843513e-01		
etc.		
etc.		

2-d Explore File:

<pre> Twobar truss at optimum diameter 1.00000000000000e+00 3.00000000000000e+00 10 height 1.00000000000000e+01 3.00000000000000e+01 10 height 1.42130000000000e+01 diameter 1.69069000000000e+00 width 6.00000000000000e+01 thickness 1.50000000000000e-01 density 3.00000000000000e-01 modulus 3.00000000000000e+04 load 6.60000000000000e+01 ### weight 4.00000000000000e+01 -1 stress 1.00000000000000e+02 -1 stress-buckling 0.00000000000000e+00 -1 deflection 2.50000000000000e-01 -1 ### 8.9411295713023e+00 2.2144852998832e+02 1.8360476449478e+02 7.3816176662774e-01 9.1592379534486e+00 1.8560496065723e+02 1.4954207824573e+02 5.3119264834560e-01 9.4143003000959e+00 1.6142381900403e+02 1.2728857395974e+02 4.1298743437499e-01 etc. etc. </pre>	<p>← The file title</p> <p>← The first variable to be meshed.</p> <p>← The Min and Max and no. of mesh points for the first variable.</p> <p>← These are names and values of variables that were held constant during the explore (diameter and height are ignored in this list).</p> <p>← File marker indicating the end of the variables.</p> <p>← Names, allowable values, type of function (-1 less than, +1, greater than), and order of function values.</p> <p>← File marker</p> <p>← Function values follow next in blocks of four</p> <p>← Next set of function values.</p>
--	--

10 Graph Window Reference

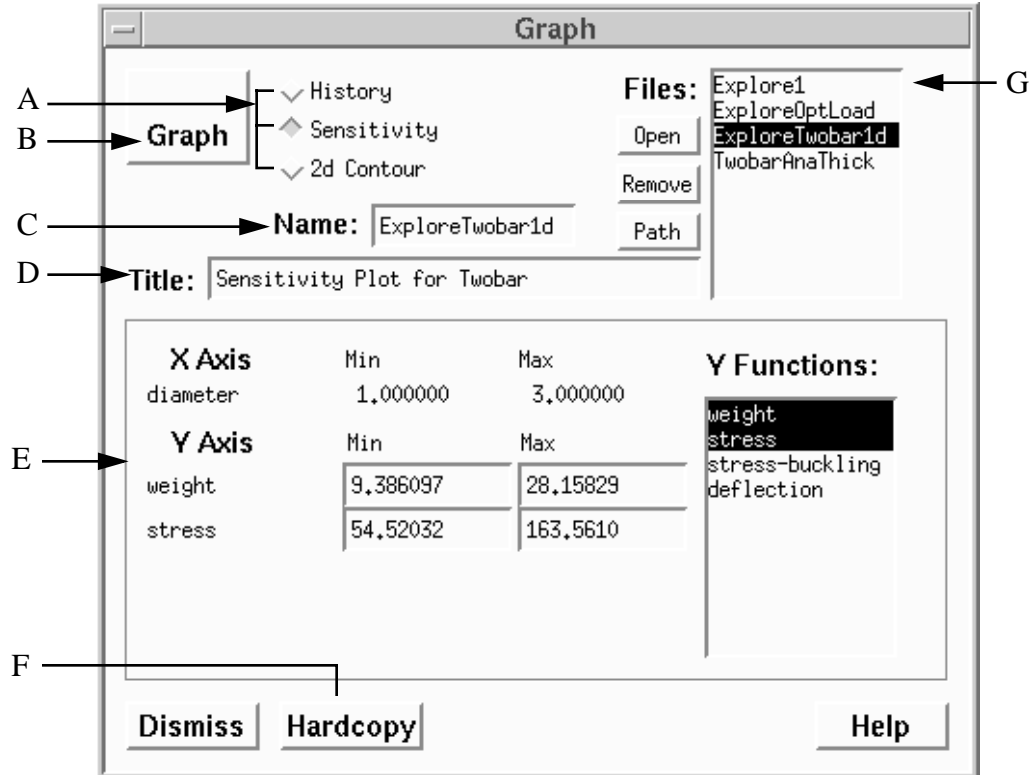
10.1 Overview

The Graph window is used to graph data files generated by other windows.

Three types of files can be graphed: History files, which show the progress of an optimization; Sensitivity files, which show how functions change with respect to one variable; and 2d Contour files, which show how functions change with respect to two variables. Sensitivity and 2d Contour files are created in the Explore window.

Depending on the file type, the Graph window is toggled between two display formats: 1-d display and 2-d display. History and Sensitivity files both use the 1-d display format. Contour files use the 2-d display format. The display format determines what data is shown in box E in the diagram.

The operation of the upper half of the window is the same for either display format. Features of this part of the window are shown below. Pressing the Graph pushbutton causes the selected data to be graphed in a new window. The button is dimmed until an appropriate selection is made. File protocol is the same as for other windows: files are selected by clicking on the file name in the Files list; the name and title of a selected file are shown in the Name and Title text fields. A file is opened by selecting it and pressing the Open pushbutton or by double clicking on it. When opened, data from the file is shown in box E.

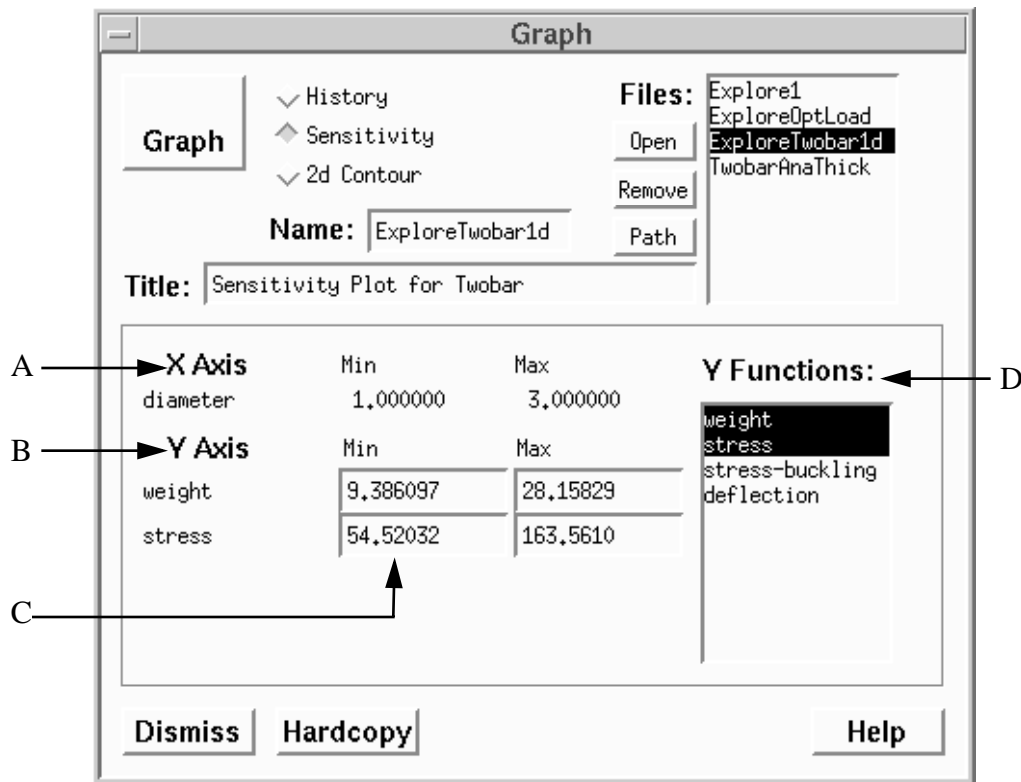


A - File type radio box. Choices are: History (one variable changing), Sensitivity (one

- variable changing), and 2d Contour (two variables changing).
- B - Graph pushbutton used to open a graphics window and plot the data from the selected file.
- C - File Name text field.
- D - File Title text field. The title is limited to 60 characters; 40 are shown. The File Title is also the title of the graph. The title is editable; editing the title changes it for the graph but does not change it in the file.
- E - File data.
- F - Hardcopy pushbutton used to obtain a hardcopy of a graph (Section 10.4).
- G - Files scrollable list displaying the available data files in the current directory for the option selected (History, Sensitivity, or 2d Contour).

10.2 Sensitivity and History Graphs

10.2.1 Reference Diagram



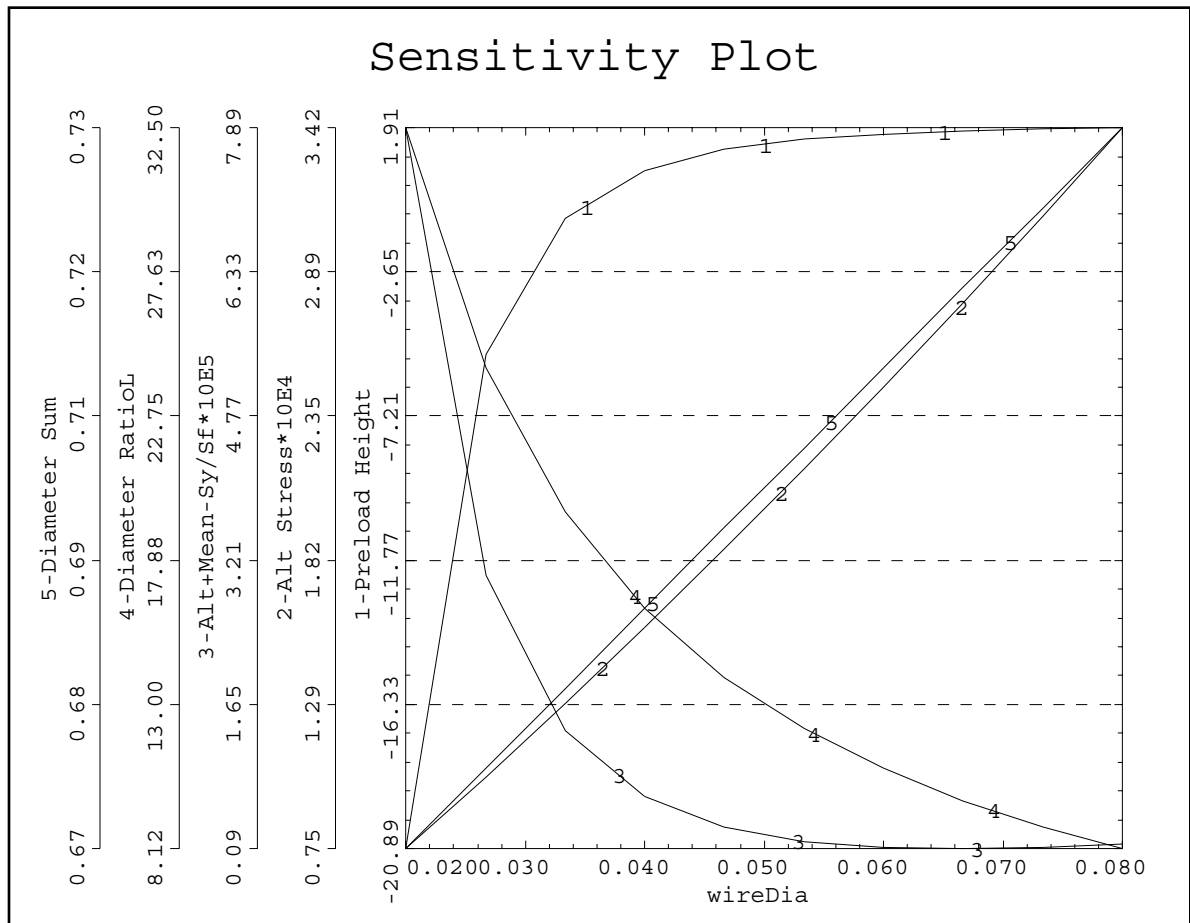
- A - X Axis information. This is read from the file.
- B - Y Axis information. For a sensitivity or history plot the Y axes are determined by the functions selected in the Y Functions list. Up to five functions can be plotted.
- C - Minimum and maximum values for the function. These set the scale on the graph; they can be edited.
- D - Y Functions scrollable list showing functions which can be plotted along the Y Axis. This is a multiple select list--clicking on an entry selects it; clicking again de-selects it.

10.2.2 Operation

General

Files can be browsed by selecting them in the Files list. When a file is opened, the file is read and X Axis and Y Axis data are displayed. You can then select up to five functions to be plotted. As the functions are selected, they appear in the Y Axis list with their mins and maxes. You can edit these mins and maxes to adjust the scale of the graph; the default is for the scale to go from the min to the max.

Pressing the Graph pushbutton causes the graph to be displayed in a new window. The graph can be resized by placing the cursor on any corner and dragging. OptdesX will adjust the resized window such that the same proportions are maintained. An example sensitivity plot is shown below.



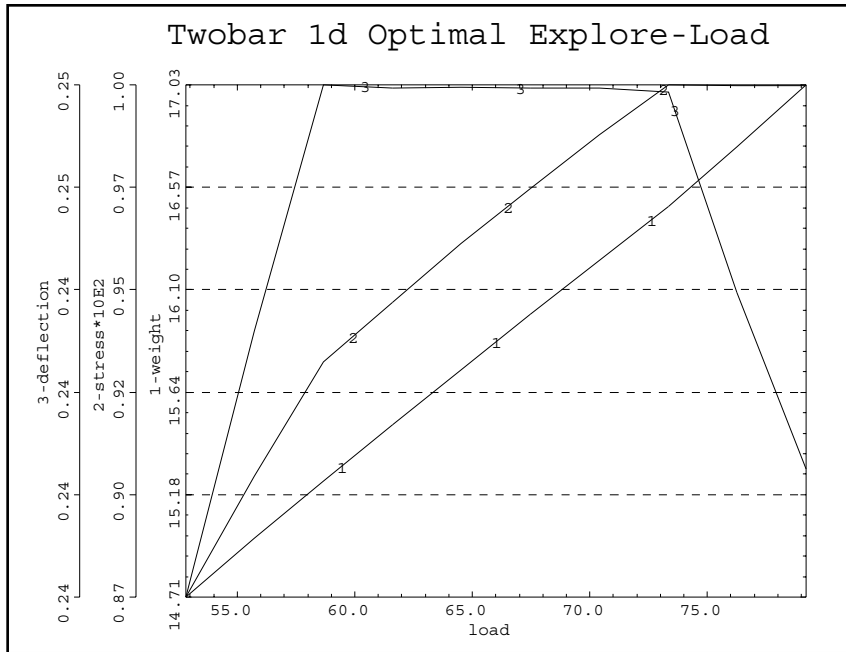
A separate scale is drawn for each function plotted. Tic marks on the scales are forced to line up; usually this means the scale divisions are not multiples of 1, 2, or 5 (unless only 1 function is plotted, in which case they are). Scale divisions can be indirectly set by adjusting the Min and Max value for the scale; see C above.

Graphing Data from an Explore Optimum

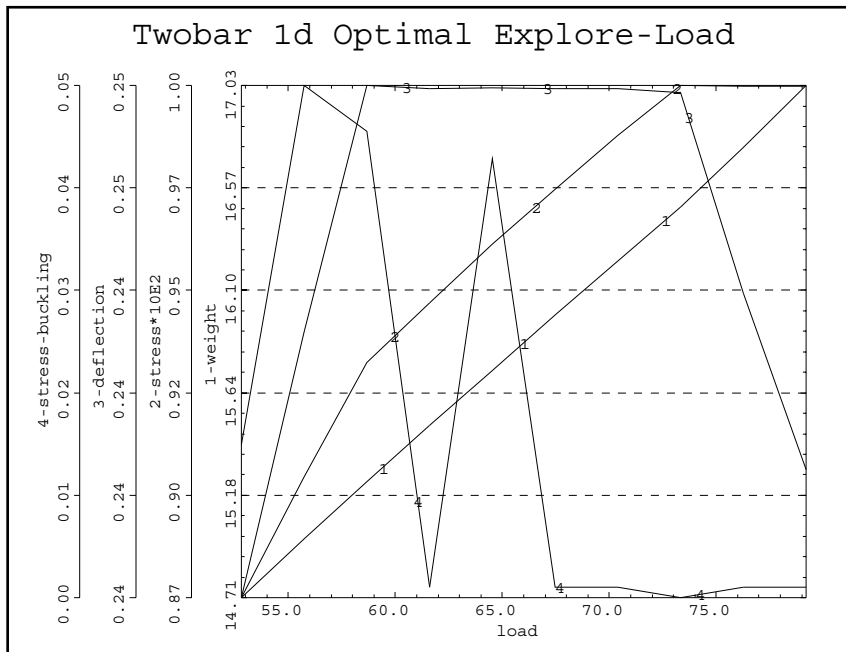
When graphing data from an 1-d explore optimum, it does not usually make sense to plot a constraint that is always binding. Such a constraint only varies by plus or minus the constraint tolerance about the allowable value; this variation is not meaningful. For example, in the tuto-

rial in Section 3.3 we graphed a 1-d explore optimum file but we did not graph stress–buckling which was always binding. (A good question at this point is, how did we know it was always binding? You can ascertain this by examining the min and max values, label C in the reference diagram, when the function is selected.)

The plot given in the tutorial in Section 3.3 is shown again below. It is followed by the same plot with stress–buckling plotted as well. Note that the scale for stress–buckling shows it varies by about 1 part in 1000, which is the same order as the constraint tolerance. Stress–buckling should be deleted--its variation is an artifact of the optimizations, and it clutters the plot.



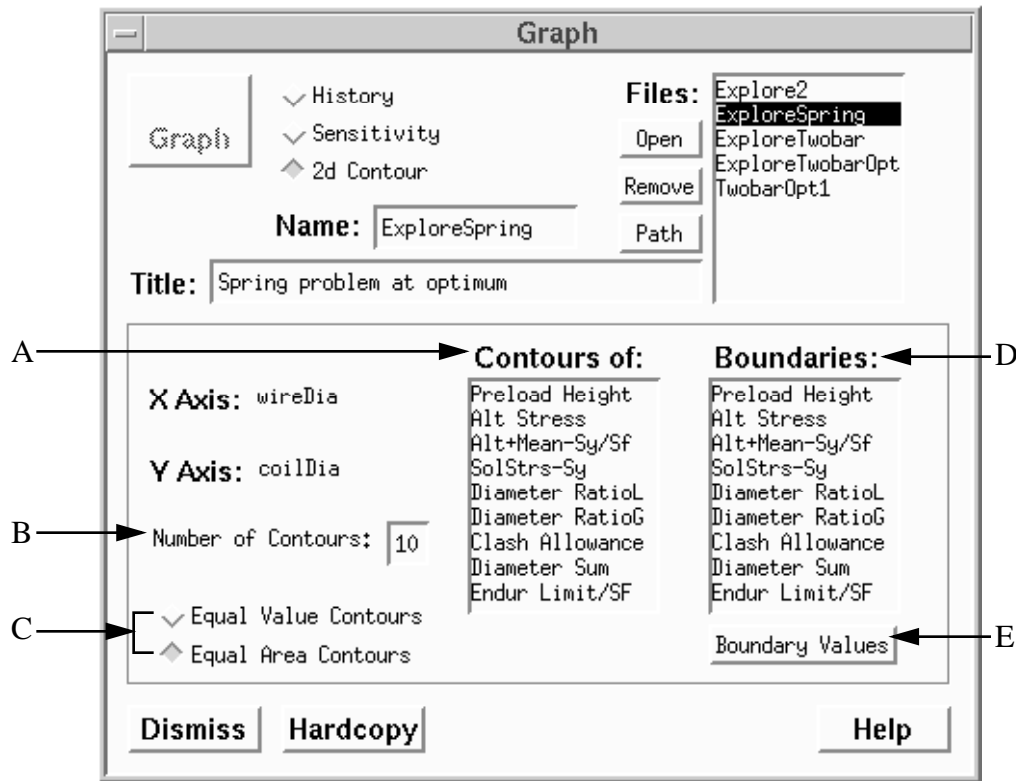
Two-bar 1-d Optimal Explore without stress–buckling



Two-bar 1-d Optimal Explore with stress–buckling graphed

10.3 Contour Plots

10.3.1 Reference Diagram



A - Contours scrollable list of design functions. Only one function can be contoured.

B - Number of Contours value field.

C - Contour Type radio box used to select the spacing of the contours. Equal Value Contours sets the spacing to be numerically equal intervals; Equal Area Contours sets the spacing such that the contours are spread approximately equally across the graph.

D - Boundaries scrollable list. As many functions as desired can be selected. Typically the constraints in the problem are selected. This is an extended-select list--items are selected by clicking and dragging. An item can be de-selected by pressing the control key while clicking on the label.

E - Boundary Value pushbutton used to open the Boundary Values window. This allows you to change the boundary values for each function. Typically the allowable values for the constraints are made the boundary values.

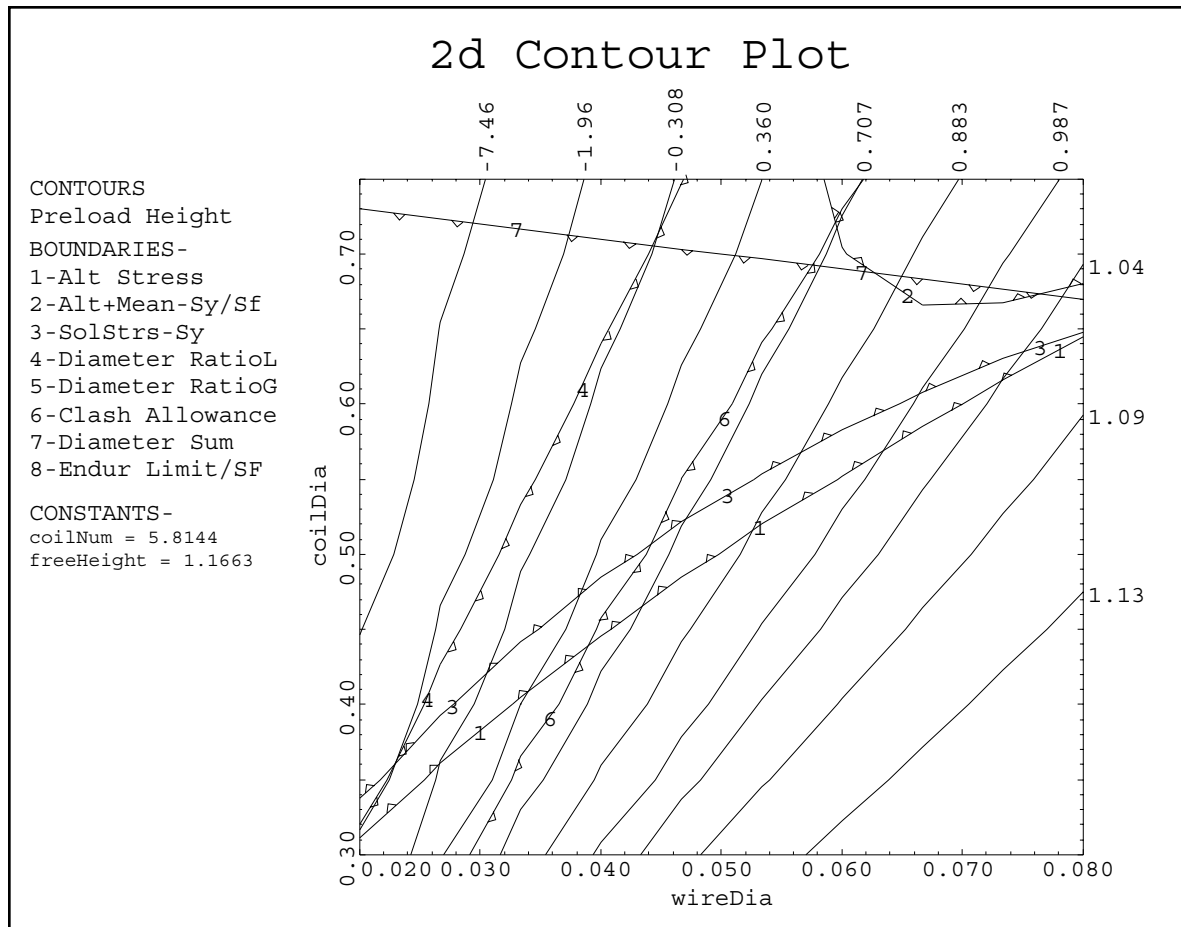
10.3.2 Operation

General

When a 2d Contour file is opened, the file is read and the “Contours Of” and “Boundaries” lists are filled. You can then select one function to be contoured, and as many functions as you wish to have boundaries plotted.

Pressing the Graph pushbutton causes the graph to be displayed in a new window. The graph

can be resized by placing the cursor on any corner and dragging. OptdesX will adjust the resized window such that the same proportions are maintained. An example contour plot is shown below.



Contour Plot with Equal Area Contours

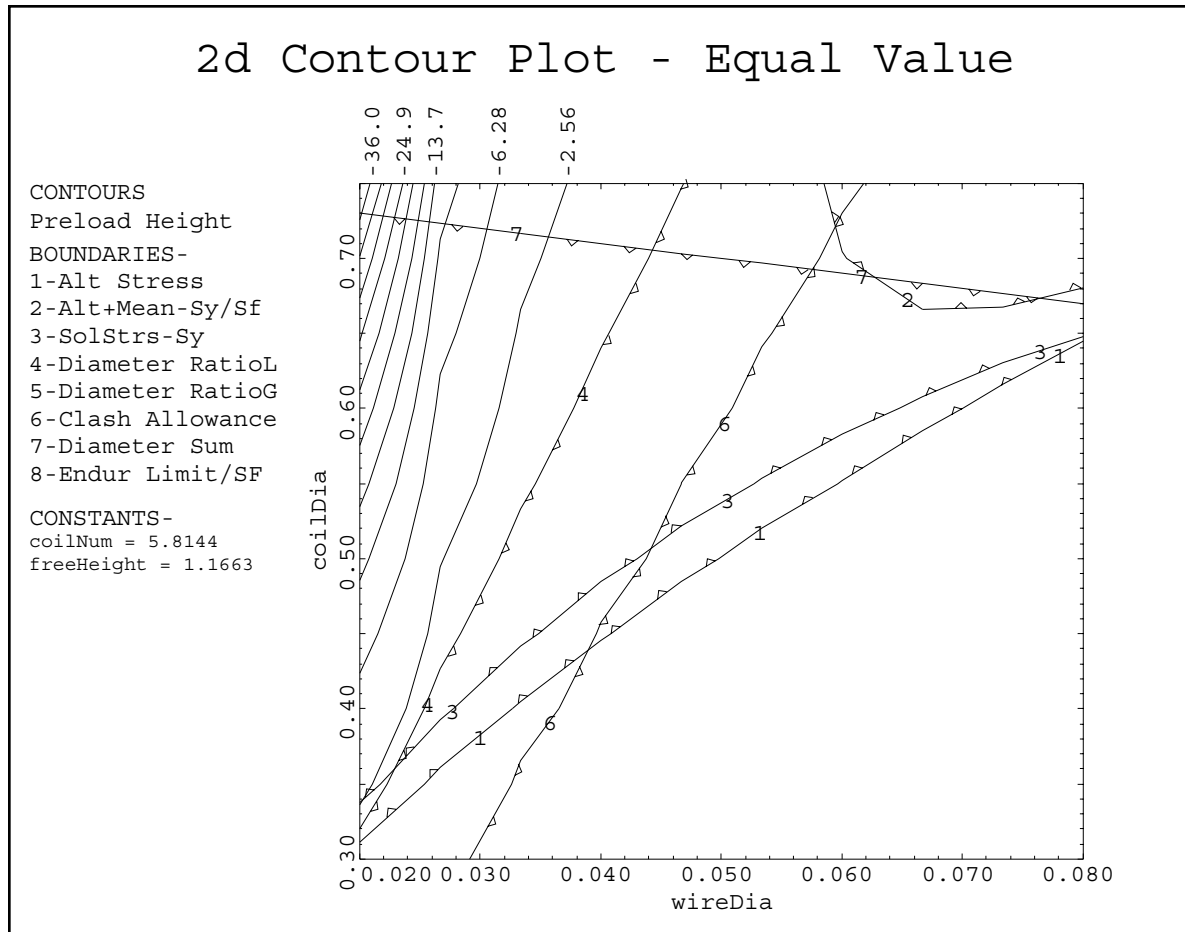
The solid lines in the plot are contour lines: they are lines of constant value for the function being contoured. Lines with triangular markers are boundaries. The markers point to the feasible side of the boundary (for equality constraints, however, neither side is feasible so the marker direction is irrelevant). For this problem, the feasible region is the small triangular area defined by the intersections of constraints 2 and 7 in the upper right hand corner.

The default values for boundaries are taken from the allowable values in the Functions window. These values can be edited in the Boundaries window (Section 10.3.2.4). Boundaries are labeled with numbers that correspond with the list on the left side of the graph.

Equal Value Contours

In the preceding example the contours were drawn with the “Equal Area Contours” option set (C in the Reference diagram). With this option OptdesX attempts to spread the contours evenly across the graph. Note however, that the numerical interval between contours changes. The “Equal Value Contours” option forces the contours to be equally spaced *numerically*.

When this option is selected the above plot looks like:

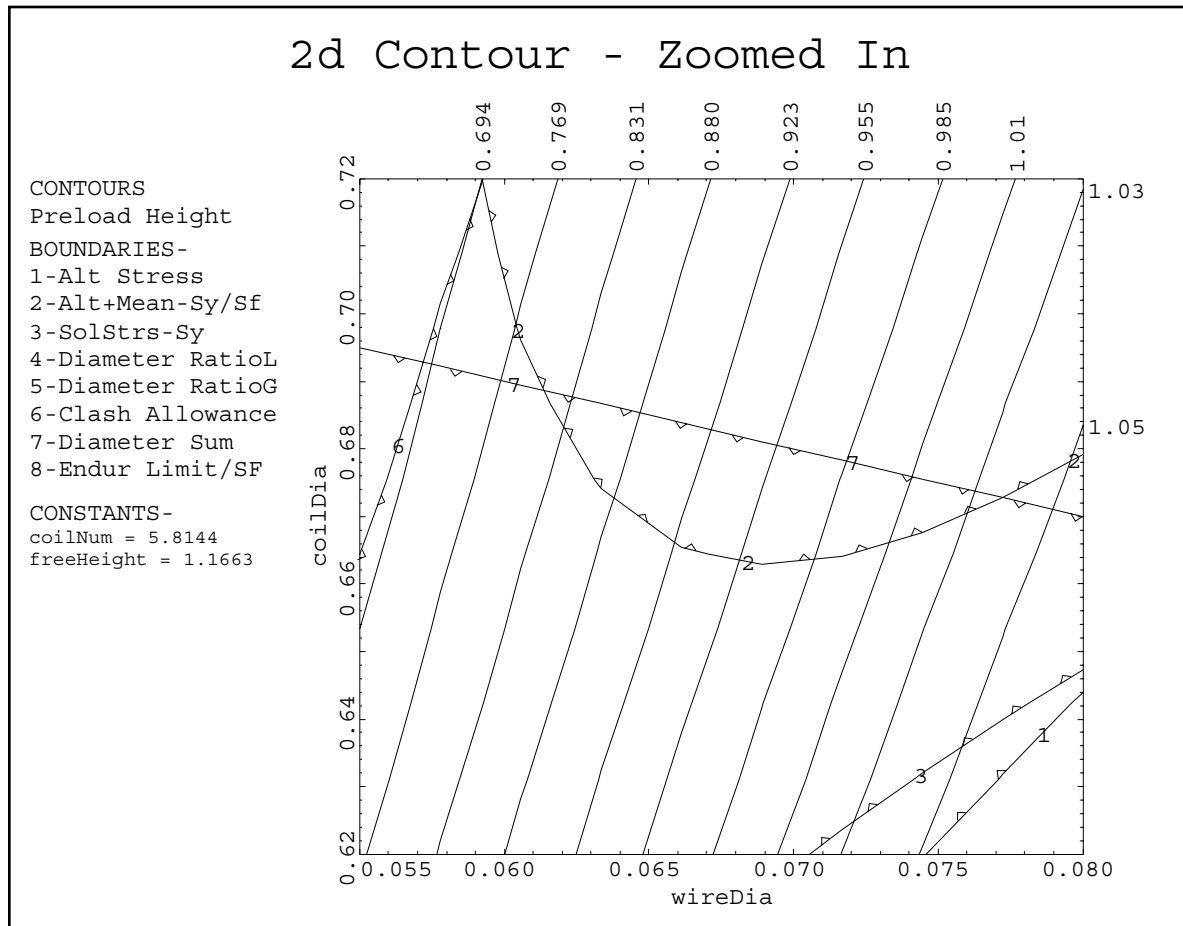


Contour Plot with Equal Value Contours

The numerical interval between all contours is the same. Because the function surface becomes relatively steep in the upper left corner, the contours are stacked up there. Equal Value contours gives a better sense of the slope of the surface; Equal Area Contours gives a better sense of the shape of the surface across the entire range. If the slope does not change too much, the graph will be nearly the same regardless of which option is selected.

Zooming In

In the above example the feasible region as a percentage of the graph is quite small. We may wish to “zoom in” on this region to see it better. We can easily do this by generating a new Explore file that has mins and maxes for the meshed variables collapsed around this region. Instead of a range for “coilDia” of 0.30-0.75, we will set it to be 0.62-0.72. Similarly, we will collapse the range for “wireDia” from 0.020-0.080 to 0.055-0.080. The plot generated from this new Explore file is shown on the next page.



We have zoomed in on the feasible region (the area enclosed by boundaries 2 and 7) and can see it much better.

Appropriate Mesh Density

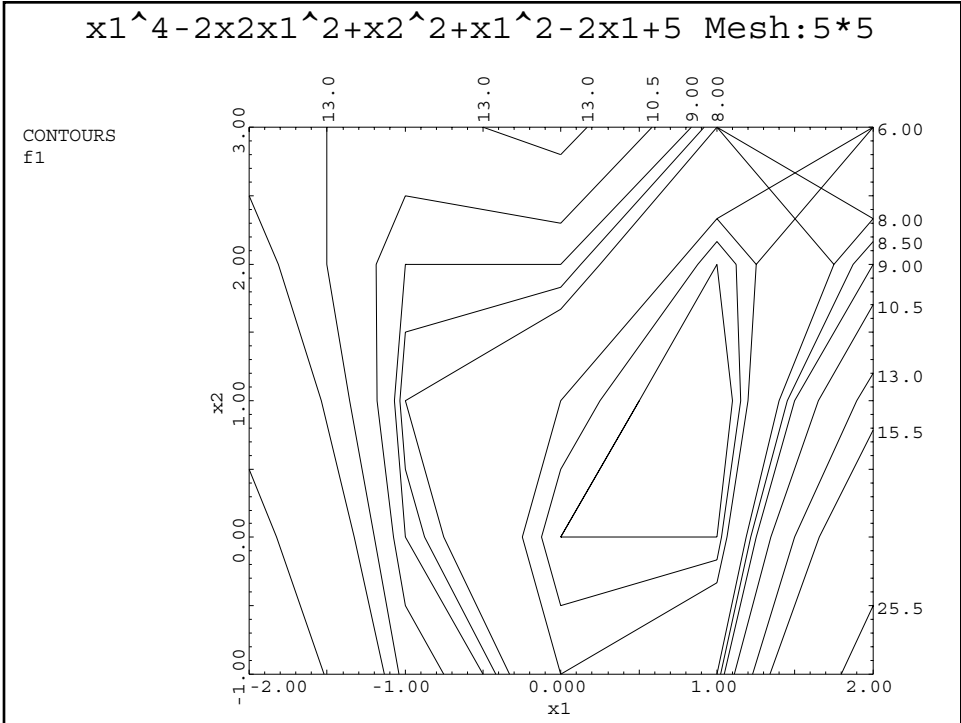
Contours and boundaries are linearly interpolated from the underlying mesh. If the mesh is too coarse for the functions being plotted, the interpolation will be in error. In general, the more nonlinear the contours are, the finer the mesh that is required to adequately represent the surface. As an example, consider the function,

$$f_1 = x_1^4 - 2x_2x_1^2 + x_2^2 + x_1^2 - 2x_1 + 5$$

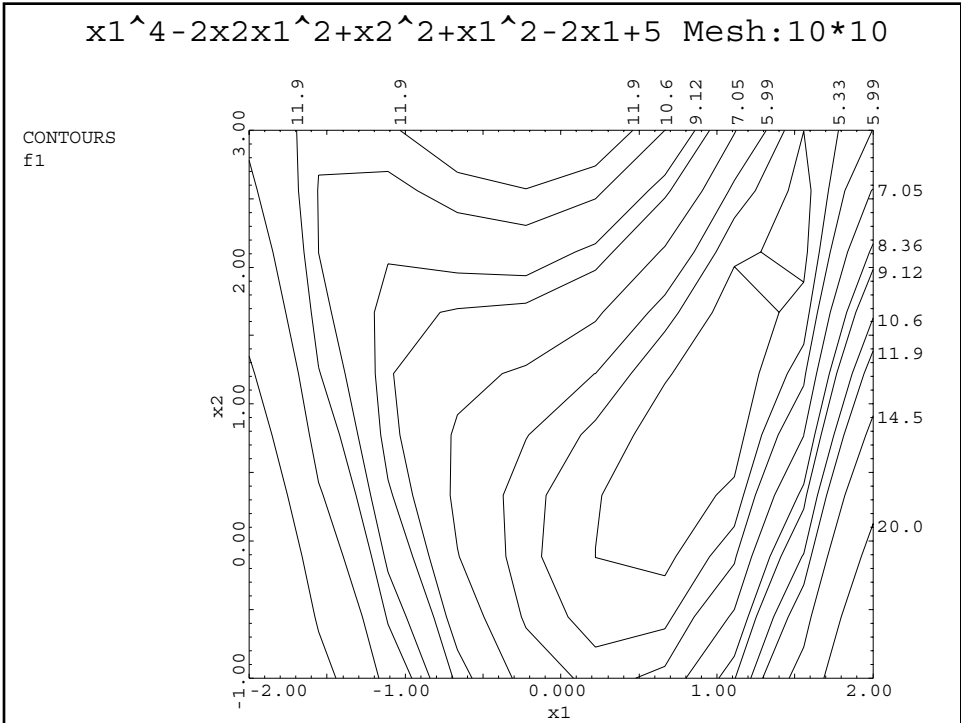
Contours for this function are given in three figures, which each have a different mesh density but the same range in the x_1 and x_2 variables. The first figure shows contours for a 5 by 5 mesh (25 analyses). The length of one grid element is 25% of the X or Y axis, as is evident by the length of the straight line segments. Clearly the mesh is inadequate.

The second figure shows contours for a 10 by 10 mesh (100 analyses). One grid element is now 12.5% of the length of the X or Y axis. The true nature of the surface can now be discerned, but the contours are still quite jagged. The contour that forms a loop has some extra segments because the spacing between the sides of the loop is less than the grid size.

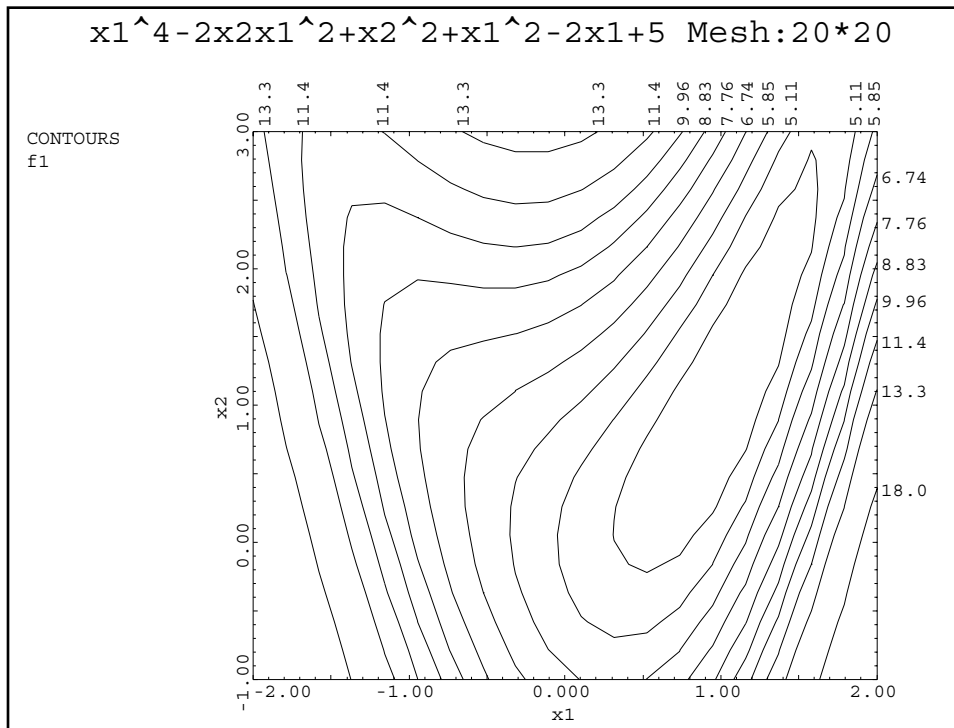
The final figure shows contours for a 20 by 20 mesh (400 analyses). One grid element is 6.25% of the length of the X or Y axis. The contours are smoother and the extra segments for the loop contour have disappeared.



Contours for a 5 by 5 mesh.



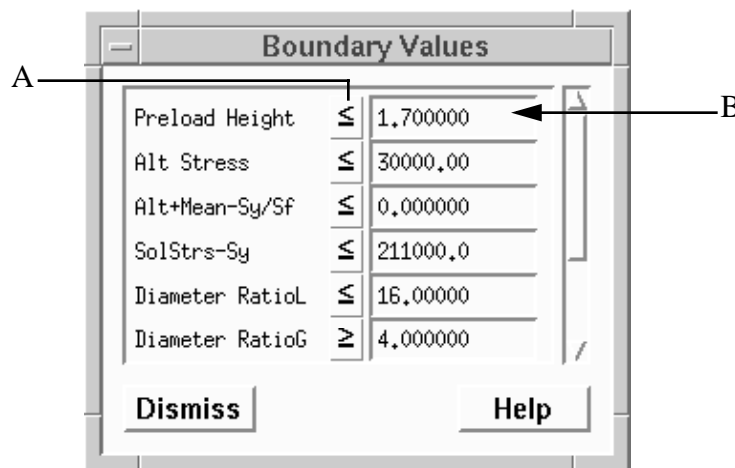
Contours for a 10 by 10 mesh.



Contours for a 20 by 20 mesh.

Boundary Values

The Boundary Values pushbutton, labeled in the Reference diagram in Section 10.3.1 as “E,” allows you to edit the boundary values that are plotted.

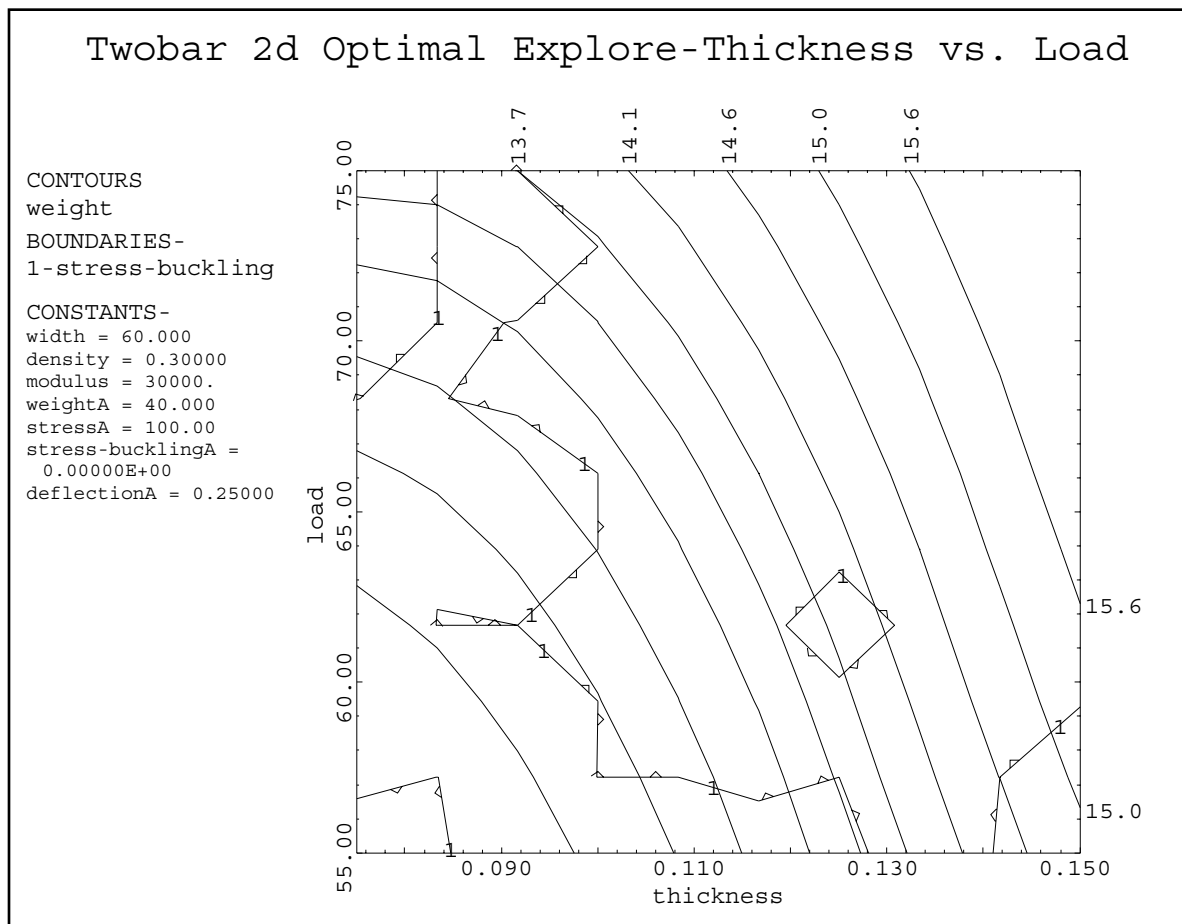


The column of buttons pointed to by label A sets the direction of the boundary markers. A boundary is just a contour level for a function. Therefore, on one side of the boundary the function decreases, and on the other it increases. For \leq boundaries, the markers point towards decreasing function values; for \geq boundaries, the markers point towards increasing function values. Clicking this button changes it from “ \leq ” to “ \geq ” and vice versa.

The Boundary Values field indicated by label B allows you to change the boundary values for the functions. This can assist you in visually understanding the effect of changing, for example, the allowable value for a constraint.

Graphing Data from an Explore Optimum

When graphing a 2-d Explore Optimum file, no boundary functions should be selected. By definition each of the mesh points is feasible (at least they better be, if we want a useful plot!), so we can't show boundaries that demarcate feasible and infeasible space. For example, in the Tutorial in Section 3.2, we plotted the optimal weight vs. thickness and load. If we show the same plot with *any* of the constraint boundaries, we get nonsense (or nothing at all):

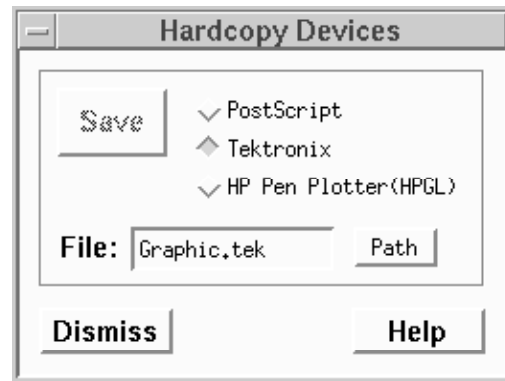


Two-bar 2-d Optimal Explore with stress-buckling boundary graphed

If all of the designs are feasible, why do boundaries show up at all? The boundaries show up as an artifact of the optimization process. During optimization a function is considered satisfied if it is less than or equal to the allowable value plus the constraint tolerance. Thus a constraint can be slightly violated within the bound of this tolerance, but this violation is not meaningful. For this reason constraint boundaries should not be plotted.

10.4 Hardcopy

The Hardcopy facility is used to obtain hardcopy of graphs. The following window is opened when the Hardcopy pushbutton is pressed:



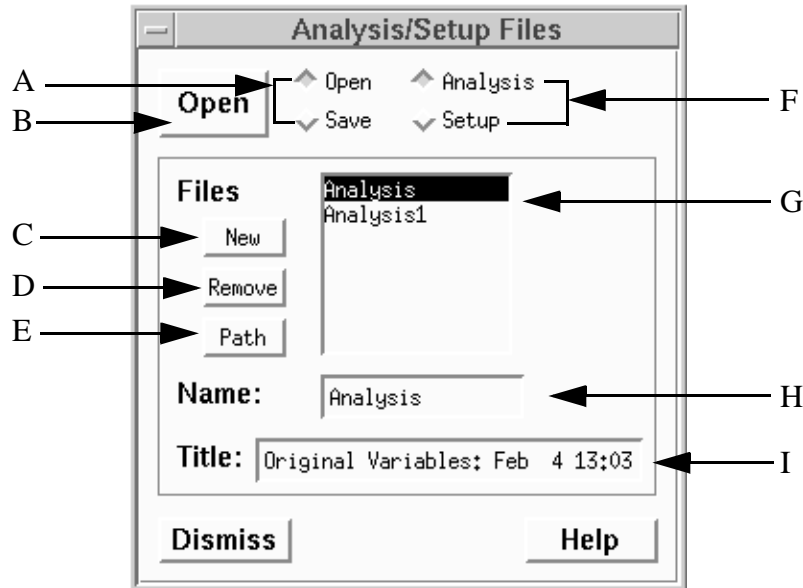
The facility supports three file formats: PostScript, Tektronix and HP pen plotter. When the Save pushbutton is pressed, the graph that would normally be drawn on the screen is dumped to a file with the name specified here. The file must then be processed by a hardcopy device that can interpret the file format.

The normal mode of operation is to view the graph on the screen, and if it is acceptable, open the Hardcopy window and save the graph to a file.

11 File Window Reference

11.1 Overview

The File Window is used to open and save Analysis and Setup files. A reference diagram is given below.



A - Open-Save Radio box.

B- File pushbutton used to open or save an Analysis or Setup file.

C- New pushbutton used to create a new Analysis file.

D- Remove pushbutton used to delete the selected file(s).

E- Path pushbutton used to change the current directory by opening the Path window.

F- Analysis-Setup Radio box.

G - Scrollable list of Analysis or Setup files (depending on the state of the Analysis-Setup radio box) in the current directory. A "<No Match>" message indicates there are no Analysis files in the current directory.

H - Name text field used for the Analysis or Setup file name. File names are limited to 16 characters.

I - Title text field used for the Analysis or Setup file title. Titles are limited to 60 characters. Typing the special characters <Date> will cause the current date and time to be inserted into the title.

File operations in all windows in OptdesX follow the same protocol: one click on the file name selects it. Its name and title are then shown in the Name and Title text fields. After selecting a file, it can be opened by clicking the Open pushbutton, or deleted using the Remove button. Alternatively, two clicks will select and open a file.

11.2 Opening Files

Files are opened by setting the Open-Save radio box to “Open,” and then either double clicking on the file or clicking on the file name and pressing the Open pushbutton.

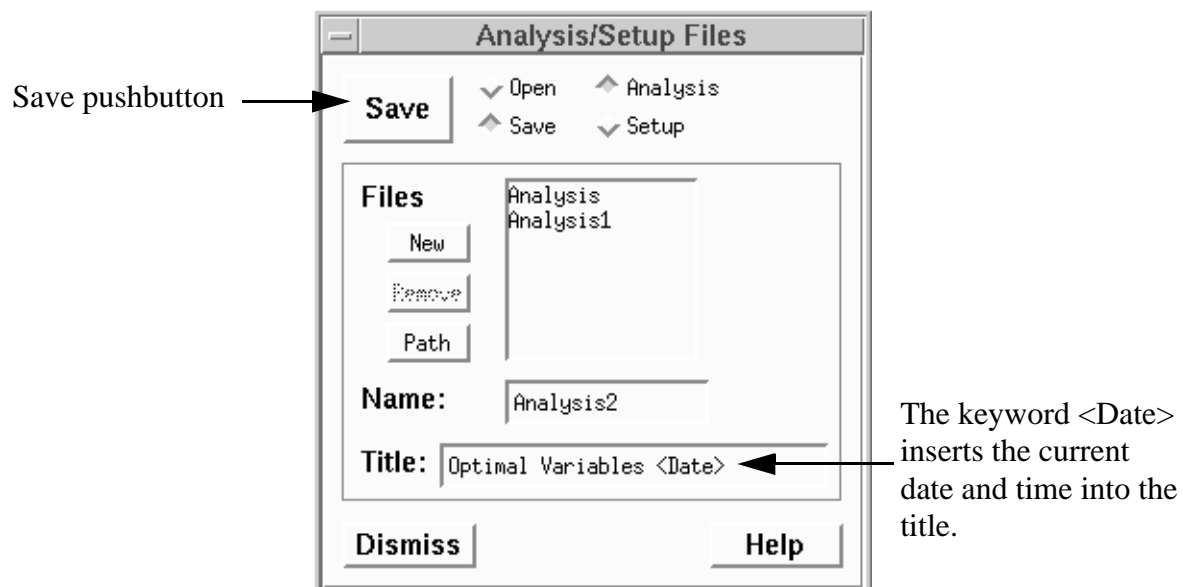
Analysis or Setup files can be selected by setting the Analysis-Setup radio box to the desired choice. An Analysis file restores the variable names and values; a Setup file restores the mapping (i.e. the specification of the objectives and constraints) for the design problem. Any Analysis file may be restored with any Setup file and vice-versa. When OptdesX is first started, you normally first restore an Analysis file and then restore a Setup file. If a Setup file is restored before any Analysis file has been opened, the most recent Analysis file is opened. OptdesX will display a message similar to:



The Files window is designed to facilitate “browsing.” Often you will have several Analysis and Setup files. In order to help you remember which is which, you can click on a file name; its name and title will then be displayed. The title is limited to 60 characters, only 30 of which are shown (by placing the cursor at the right of the text field and dragging, you can see it all). If you save files with descriptive titles, you can browse the list and, by reading titles, differentiate among them.

11.3 Saving Files

Analysis and Setup files can be saved at any time. Files are saved by toggling the Open-Save radio box to “Save,” typing the file name and title, and pressing the Save pushbutton.



OptdesX will suggest a default name and title. The default name is “Analysis” for Analysis files. If you type something else, it becomes the default for future files. A numeric extension is added on the assumption you do not wish to overwrite previous files. The default title is “Analysis <Date>” for Analysis files. When typed as part of the title, the keyword <Date> will cause the current date and time to be written when the file is saved. If you replace the default with something else, it becomes the default from that point on.

Default names and titles for Setup files are the same as those for Analysis files, with the word “Setup” exchanged for “Analysis.”

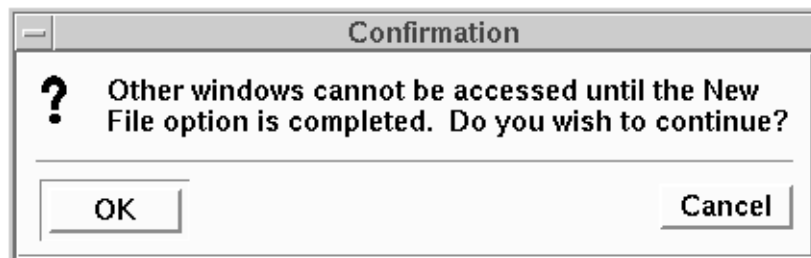
Analysis files are given the extension “.ana” by OptdesX. Setup files are given the extension “.opt.” These extensions are not shown when the files are displayed in OptdesX windows.

11.4 New, Remove, Path

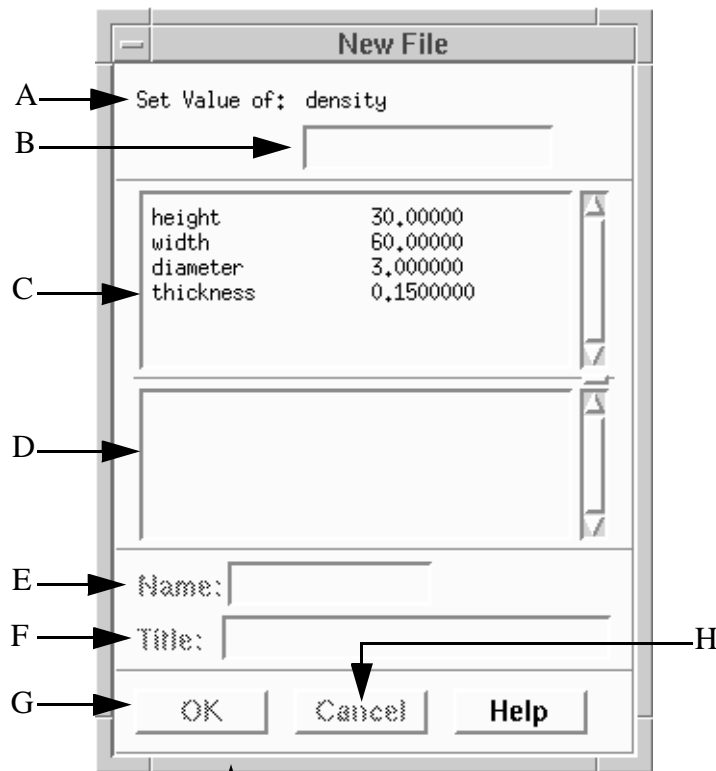
11.4.1 The New Button

The New File option allows you to create a new Analysis file. The option may be invoked at any time, but is usually only used the first time OptdesX is executed with a particular model and no Analysis files exist. Afterwards Analysis files are more easily created by changing the design in the Variables window, computing the functions, and pushing the “Save” button.

The New File window is “modal,” meaning that once this option is selected it must be completed before anything else can be done. A Confirmation box allows you to confirm that this is what you wish to do.



The New File window will open on your screen:



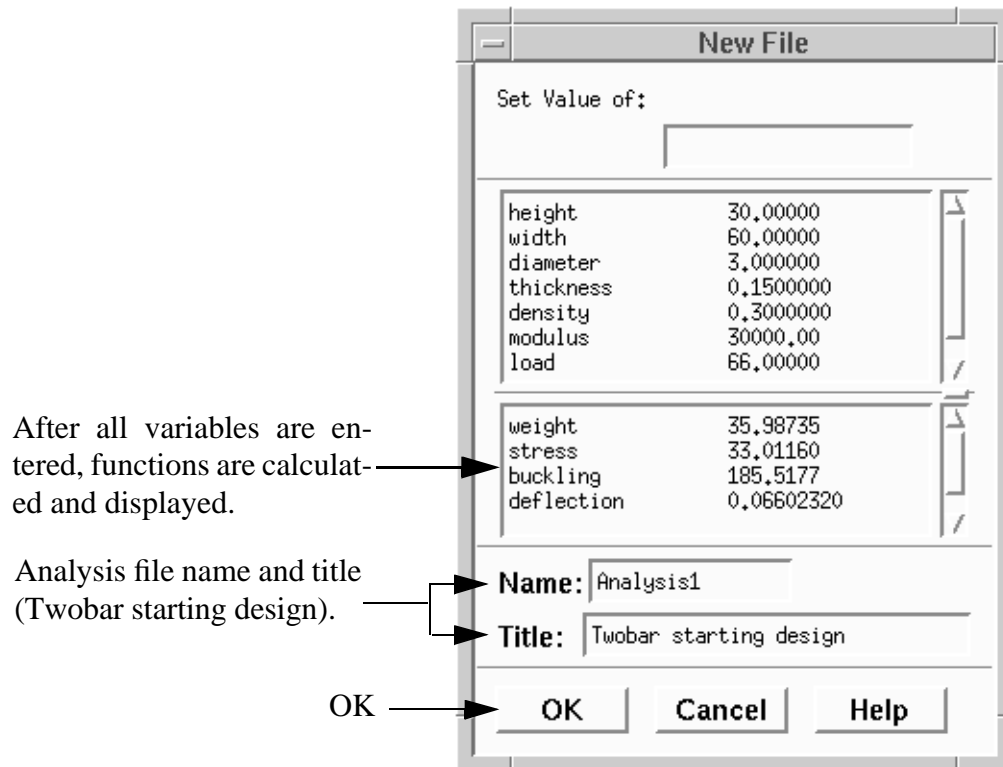
The window can be made longer by grabbing the top or bottom border and dragging.

- A - Name text field. This field acts as a prompt--you are to type the value for the variable in the value field below it.
- B - Analysis variables value field.
- C - Scrolled window of analysis variables already entered.
- D - Scrolled window of analysis function values. These are computed after all variables are entered.
- E - Analysis file name. The file name is limited to 16 characters.
- F - Title text field. The title is limited to 60 characters (only 30 are displayed--to see the rest of a long title, place the cursor at the right boundary and drag).
- G - OK pushbutton used to save the Analysis file after all values have been entered.
- H - Cancel pushbutton used to close the New File window without saving the analysis file. This option cannot be selected until all the variable values have been entered.

The New Window opens with only the value field for the first analysis variable displayed. After typing and registering the value with a carriage return, the variable will be displayed in the scrolled window below it, and you will be prompted for a new variable value. If you make a mistake but do not notice it until after the carriage return is entered, you cannot correct the error. You must enter values for the rest of the variables and then cancel and start over again.

Once all the variable values have been entered, the functions are calculated and displayed in

the middle box of the New File window as shown below.



After all variables are entered, functions are calculated and displayed.

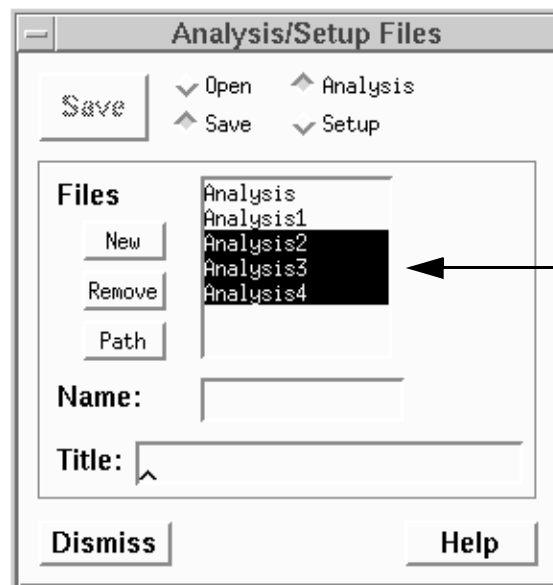
Analysis file name and title (Twobar starting design).

OK

You then enter a file name and title and press the OK button to save the file. If you do not wish to save the file, you push Cancel.

11.4.2 The Remove Button

The Remove pushbutton allows you to remove files from your directory. You can remove files one at a time by selecting them with a single click, or you can remove a block of files by clicking and dragging (individual files within a block can be de-selected by holding down the option key while clicking):



You can select a block of files by clicking and dragging.

11.4.3 The Path Button

The Path button allows you to set the directory where files will be found. Several windows have path buttons, and changing the path in any one of them changes it globally.



The OK button changes the path and closes the window; the Apply button changes the path and leaves the window open; the Reset button resets the path to what it was when OptdesX was first started; Cancel closes the window without taking any action.

11.5 Example Analysis and Setup Files

An example Analysis file is given below. Analysis files are not initially saved with function values in them. Only after they have been restored once are function values written. If function values are not present, OptdesX calculates them when the file is restored. If function values are present, they are marked by a “#” character in the file.

Original Variables: Feb 4 13:03	←	File title
Two-bar Truss	←	Model Name
d 0 0 height 30.000000000000		Variable type, name and value
d 0 0 width 60.000000000000		
d 0 0 diameter 3.000000000000		
d 0 0 thickness 0.150000000000		
d 0 0 density 0.300000000000		
d 0 0 modulus 30000.00000000		
d 0 0 load 66.000000000000		The # is a marker separating functions and variables.
#		
d 0 0 weight 35.98735233072		Function type, name and value
d 0 0 stress 33.01159777142		
d 0 0 buckling 185.5177257080		
d 0 0 deflection 0.06602319554284		

A Setup file contains mapping information. A Setup file for the Twobar Truss from Tutorial 1 is given below. A Setup file has many descriptors or codes for variables and functions to completely describe the setup. It is not expected that you will want to manually write this file so we do not present a complete explanation of it. If you need to have more information about the Setup file, please contact Design Synthesis.

```

Twobar truss starting setup
Twobar Truss
c 0 0 label dv1
height
d 0 0 heightL
10.0000
d 0 0 heightU
30.0000
i 0 0 height1
1
c 0 0 height2
height
c 0 0 label dv2
diameter
d 0 0 diameterL
1.00000
d 0 0 diameterU
3.00000
i 0 0 diameter1
1
c 0 0 diameter2
diameter
i 0 0 number dvs
2
c 0 0 label df1
weight
d 0 0 weightA
40.0000
d 0 0 weightI
10.0000
i 0 0 weightD
0
d 0 0 weightW
1.0000
i 0 0 weight3
1
c 0 0 weight4
weight
c 0 0 label df2
stress
d 0 0 stressA
100.000
d 0 0 stressI
50.0000
i 0 0 stressD
1
i 0 0 stress3

```

File title

Model Name

Variable descriptors for height.

Function descriptors for weight.

File Window Reference 11-8

```
1
c 0 0 stress4
stress
c 0 0 label df3
stress-buckling
d 0 0 stress-bucklingA
0.00000
d 0 0 stress-bucklingI
-50.0000
i 0 0 stress-bucklingD
1
i 0 0 stress-buckling3
2
c 2 0 stress-buckling4
stress
buckling
i 0 0 stress-bucklingT
1
i 2 0 stress-buckling+
1
-1
c 0 0 label df4
deflection
d 0 0 deflectionA
0.250000
d 0 0 deflectionI
0.0500000
i 0 0 deflectionD
1
i 0 0 deflection3
1
c 0 0 deflection4
deflection
i 0 0 number dfs
4
i 0 0 number sets
0
i 0 0 number combos
0
```

12 Error Reference

Errors which are reported by OptdesX are listed in alphabetical order. Explanations follow, except for errors deemed to be self-explanatory. Fatal errors are errors from which the software cannot recover, resulting in termination of the program. Naturally, the customer should never see a fatal error. However, in the event that a fatal error should occur, contact Design Synthesis for support.

A positive number of iterations must be chosen.

Allocation failure in ...

Fatal Error. OptdesX was not able to allocate memory. Either the problem size must be reduced or more memory must be made available.

An optimization cannot be performed until the following errors are resolved:

The optimization problem setup is incomplete. The listed problems must be resolved before optimization can proceed.

Analysis files cannot be opened until the following errors are resolved:

The listed problems must be resolved before optimization can proceed.

AVLoseFocusCB: can't find the AV node for that operation.

An error has occurred in the linked list of analysis variables. This should never happen. Contact Design Synthesis if this error occurs.

Bad Call to Attach.

Contact Design Synthesis.

Bad Call to CreateStdPixmap.

Contact Design Synthesis.

Bad case--PrepareTitleString

A title has a bad keyword inserted into it. The most common keyword is <Date>.

Bad file format.

There are two reasons for this error: 1) the file has somehow become corrupted so that things are not where they are supposed to be, or 2) the file contains bad data, e.g. "Infinity," "NaN," "?", etc. which cannot be graphed. The second reason is the most common. To check a file for bad data, use your system editor to search for bad strings. Bad strings must be replaced with good data before the file can be read. Usually bad data results from doing an explore with ranges for variables that are too wide. Collapse the ranges and test the corner points of the ranges manually (doing a "Compute Functions") to make sure all designs for an explore are defined.

Best method can not be computed because the Allowable and Indifference values for <label> are the same.

The allowable and indifference values are used to scale the functions. See section 2.4 in the user's manual for more information about function scaling. If the allowable and indifference values were the same, a divide by zero error would result when scaling the functions.

Cannot find af: <label>

Cannot find av: <label>

Cannot find df: <label>

Cannot find dv: <label>

Cannot find ds. <label>

Fatal errors. OptdesX could not find a label in its database. These errors should not happen unless Analysis or Setup files have been edited and are no longer consistent with each other or with the labels used in the ANAFUN routines. All of these labels must match, including leading or trailing blanks.

Cannot find node: <node name>

An error has occurred in a linked list of variables or functions. This should never happen. Contact Design Synthesis if this error occurs.

ChangeFunMap: can't find the DF node for that operation.

ChangeMapTypeCB: can't find the MF node for that operation.

ChangeSignCB: can't find the MF node for that operation.

ChangeVarMap: can't find the DV node for that operation.

Fatal errors. An error has occurred in searching a linked list. This should never happen. Contact Design Synthesis if any of these errors occurs.

Color matrix cannot be displayed for 1x1 matrix.

The matrix to be displayed only contains one element. This piece of data can only be displayed as a Value matrix; it cannot be displayed as a Color matrix, which requires at least two pieces of data.

CreateWeightWidget: cannot locate the first objective function.

Fatal error. This should never happen. Contact Design Synthesis if any of these errors occurs.

DCloseFocusCB: can't find the DC node for that operation.

DFCloseFocusCB: can't find the DF node for that operation.

DPLoseFocusCB: can't find the DP node for that operation.

DVCloseFocusCB: can't find the DC/MC node for that operation.

DVCloseFocusCB: can't find the DV node for that operation.

Fatal errors. An error has occurred in searching a linked list. This should never happen. Contact Design Synthesis if any of these errors occurs.

Derivatives could not be checked with <label> Please make another variable the first design variable

The Determine Best Method feature uses the design variable at the top of the window, i.e. the first variable in the list, to check for model noise. If all derivatives with respect to this variable are zero, another variable must be used. Make another variable--preferably one that you think

is associated with noise, the variable at the top of the list.

- * **Design function name <label> is not unique. It conflicts with an AF.**
- * **Design function name <label> is not unique. It conflicts with another DF.**
- * **Design variable name <label> is not unique. It conflicts with an AV**
- * **Design variable name <label> is not unique. It conflicts with another DV**

Design variable and function labels must be unique. Particularly when mapping analysis variables or functions together, it is possible to have two variables or functions with the same name. Names with frames around them can be edited. Edit conflicting names so that they are unique.

DestroyWeightWidget: cannot locate the first objective function.

Fatal error. This should never happen. Contact Design Synthesis if any of these errors occurs.

Duplicate af: <label>

Duplicate av: <label>

Fatal error. A duplicate label has been inserted into the OptdesX database. This should never happen. Contact Design Synthesis.

e too large, IMAX too small in routine gser.

This fatal error occurs during the calculation of tolerances. Contact Design Synthesis.

ERROR: cannot normalize zero-length vector.

Fatal error, Contact Design Synthesis.

Error: Attempt to invert singular matrix

This error is generated inside the optimization algorithms. The algorithms have strategies built in to handle singular matrices; nevertheless sometimes these fail. Moving to another point in design space will sometimes fix this problem.

Error: Double precision storage exceeded, <integer>

The SQP and GRG algorithms in OptdesX are written in Fortran and do not allocate memory dynamically. This message means the algorithms have run out of double precision storage. The only way to recover is to make your optimization problem smaller. If you need a version of the software with more storage, contact Design Synthesis.

Error in Grgiji, Singular.

Fatal error. Contact Design Synthesis.

Error: Integer storage exceeded, <integer>

The SQP and GRG algorithms in OptdesX are written in Fortran and do not allocate memory dynamically. This message means the algorithms have run out of integer storage. The only way to recover is to make your optimization problem smaller. If you need a version of the software with more storage, contact Design Synthesis.

Error: No solution found to Exhaustive Search

Error: No solution found to Branch and Bound

No solution could be found for any of the discrete values given in the discrete files. You can verify this by setting the variables to correspond with discrete values and then computing the functions. You should then see that some of the constraints are not feasible.

Error reading file: <filename>. The first line of the file must contain the number of rows and columns (both integers).

The first line of a discrete file must have the number of rows (integer) followed by the number of columns (integer). These can be separated by white space (tabs or blanks) but not commas.

Error reading file: <filename>. The number of columns must be ≥ 1 .

The first line of a Discrete file contains two integer numbers which give the number of rows and number of columns. Both must be positive values..

Error reading file: <filename>. An error occurred in trying to read an entry for Row <row number>, Column <column number>

OptdesX was not able to read a discrete file entry. Check the given row and column numbers to locate the bad data.

Exploring at the analysis level cannot be done until the following errors are resolved:

An Explore Analysis cannot be performed until the listed problems are resolved.

Exploring at the optimization level cannot be done until the following errors are resolved:

An Explore Optimum cannot be performed until the listed problems are resolved.

File extension not recognized.

This error usually occurs if trying to read a setup file which does not have a .opt extension.

File: <filename> contains bad data.

Usually this means the file contains strings such as “NaN,” “Infinity,” or “?” which are generated on some systems when a overflow, etc. occurs. Check the file using your system editor to see if you can locate the bad data. This data must be removed (usually meaning the file is re-generated with better ranges) before it can be read in or graphed.

File: <filename> is not compatible with Analysis model name

The model name (which comes from the name in ANAPRE) in the file does not match the model name read from ANAPRE. This means the Analysis or Setup files belong to a different problem, i.e. a different set of ANASUB routines.

File: <filename> is not compatible with Analysis model name. Please open another Analysis file before opening a Setup file.

A Setup file was opened before an Analysis file, so the most recent Analysis file was selected to be opened; however, this Analysis file was not compatible with the model name in ANAPRE. You need to open another Analysis file before opening a Setup file.

Files cannot be saved until the following errors are resolved:

An Analysis or Setup file cannot be saved until errors in the variables or functions are resolved.

Functions cannot be computed until the following errors are resolved:

Errors must be resolved (usually with non-unique names) before analysis functions can be computed.

Gradients at the optimum cannot be computed until the following errors are resolved:

Gradients cannot be computed until the following errors are resolved:

Gradients cannot be computed until the listed errors are resolved.

GRG--Error in Grgpr3, zero denom

This error should not happen. Contact Design Synthesis.

GRG--Error in line search

This error should not happen. Contact Design Synthesis.

GRG--Error in QP solver--Hessian not pos. definite.

Usually this error can be overcome by starting the algorithm from a different point in design space.

GRG--Error in QP solver--No solution found

This error can occur in two ways: a solution exists to the actual problem, but no solution exists to the approximate problem given the QP solver, or no solution exists to the actual problem. Check to see if you can find a feasible starting point and restart the algorithm from this point. A feasible point is guaranteed to solve this problem.

GRG--Error1 in Grgiji, Singular

GRG--Error1 in Grgint

GRG--Error2 in Grgint

GRG--Stpdel error in Grgsat, bnd

These errors should not happen. Contact Design Synthesis.

History file: <filename> is corrupt and will be deleted.

The usual cause for this error is bad data such as “NaN,” “Infinity,” or “?” or other such strings inserted into the History file.

Include must be <= Total.

The number of discrete values for a neighborhood must be less than or equal to the total discrete values in the file.

Label greater than <max label length> characters af: <label>

Label greater than <max label length> characters av: <label>

Names for analysis variables and functions are limited to 16 characters.

Lagrange multipliers do not exist.

Derivatives at the optimum cannot be computed unless you have just finished running the GRG

or SQP algorithms so that Lagrange multipliers are available.

ListCB: cannot locate the selected variable in the database.

Fatal error. This should never happen. Contact Design Synthesis.

Maximum errors reached.

Contact Design Synthesis.

MaxSummarySize too small to hold current message.

The Summary window can hold a message of approximately 500,000 characters, at which point the beginning of the message is overwritten.

Negative neighborhood not allowed.

The number of discrete values for a neighborhood must be positive.

No curves selected.

Graphing cannot proceed until some curves are selected.

No design functions exist.

No design variables exist.

Gradients of “Design Functions with respect to Design Variables” cannot be taken unless design variables and functions exist. You can take derivatives of All Functions with respect to All Variables.”

No file selected.

A file must be selected before an operation can be performed.

No functions exist.

Design functions must be defined before an optimization can be performed.

No inequality constraint functions exist.

This error occurs when trying to compute shifts with the tolerances option. Shifts are applied to inequality constraint functions, and this error will occur anytime you try to optimize or compute shifts with tolerances on when no inequality constraints are specified.

No variables exist.

Design variables must be defined before an optimization can be performed.

No unmapped analysis variables exist.

Gradients of the optimum with respect to unmapped analysis variables cannot be computed unless some unmapped analysis variables exist.

Not available for multiple objectives

Derivatives of the optimum cannot be computed for problems with multiple objectives.

Not enough space in af: <label>

Not enough space in av: <label>

Not enough space in database for af: <label>

These errors only occur when vectors are being used as analysis variables or functions. You have requested that OptdesX load a vector from its database into a vector that is not long enough. The number of elements must be the same.

Number of contours must be a positive number.

Specify a positive number of contours for the contour plot.

Out of range af: <label>

Out of range av: <label>

These errors only occur when vectors are being used as analysis variables or functions. You have requested that OptdesX find elements of a vector that do not exist, i.e. you have sent a number of elements that is too large. The number of elements must be the same.

Path too long. Unable to change directory.

The path must be less than 256 characters, or OptdesX will not be able to use it.

Please open an Analysis file before opening a Setup file.

A Setup file was opened before an Analysis file was opened. OptdesX attempted to open the most recent Analysis file, but for some reason was unsuccessful. Open an Analysis file before opening a Setup file.

Random number generator error.

This error should never happen. Contact Design Synthesis.

Shifts cannot be computed until the following errors are resolved:

Fix the errors listed before continuing.

Spaces are illegal in file names.

*** Some of the discrete variable files have not been read in. Press the OK button to the right of each discrete variable file name.**

SQP: Double precision storage exceeded, <integer>

SQP: Integer storage exceeded, <integer>

The SQP algorithm is written in Fortran and does not allocate storage dynamically. The default double precision storage is set to 40000. The only way to overcome these errors is to make your problem smaller or obtain a version of the software from Design Synthesis with larger storage limits.

*** The allowable value equals the indifference value for design function <label>.**

The difference in these two values is used as a range for scaling; it must not be zero.

*** The number of combinations for the Branch and Bound algorithm must be greater**

than zero.

The number of combinations is set in the Parameters window--this window is opened by pressing the Parameters pushbutton in the Optimize window.

The following variables and/or functions have scaling differences greater than three orders of magnitude:

Poor scaling can significantly effect the ability of the algorithms to make progress. The minimum and maximum values for variables and the allowable and indifference values for functions can be adjusted to remedy this error. See section 2.4 of the user's manual for more information about variable and function scaling.

*** The number of combinations for the Exhaustive Search algorithm must be greater than zero.**

The number of combinations is set in the Parameters window--this window is opened by pressing the Parameters pushbutton in the Optimize window.

The software license expired <number> days ago. Please issue a purchase order immediately or return the software to

**Design Synthesis
390 West 800 North Suite 103-C
Orem, UT 84057
(801) 223-9525**

The license key delivered to you with the software was encoded with a license expiration date. After this date passes, you are no longer able to use the software until a new license key code has been received from Design Synthesis. A new key code can be generated and faxed to you as soon as a purchase order for the license extension fee is received by Design Synthesis.

*** There are no design variables selected.**

Optimization cannot proceed until some design variables have been defined.

*** There are no discrete variables.**

A discrete optimization algorithm has been selected but no discrete variables have been defined.

*** There are NULL FUNCTIONS in the Functions window.**

NULL FUNCTIONS result when #Map is greater than one; the next function(s) mapped will fill the NULL FUNCTIONS slot. The NULL FUNCTION can be deleted by making the #Map equal to 1.

*** There are NULL VARIABLES in the Discrete Variables window.**

NULL VARIABLES result when there is more than one column in the Discrete file; the next discrete variable(s) mapped will fill the NULL VARIABLES slot.

*** There are NULL VARIABLES in the Variables window.**

NULL FUNCTIONS result when #Map is greater than one; the next variable(s) mapped will fill the NULL VARIABLES slot. The NULL VARIABLE can be deleted by making the #Map

equal to 1.

*** There is no objective function selected.**

There is a space in the path name. Path name not applied.

TolAVLoseFocusCB: can't find the AV node for that operation.

TolDVLoseFocusCB: can't find the DV node for that operation.

Fatal errors. Contact Design Synthesis.

Unable to list selected path. Apply aborted.

OptdesX cannot obtain a directory of the files in the current path.

Uninitialized node: <name>

UnmapDscComCB: can't find the DV node for that operation.

UnmapDesVarCB: can't find the DC/MC node for that operation.

Fatal errors. An error has occurred in searching a linked list. This should never happen. Contact Design Synthesis.

Warning: There is a space in the path name.

Index

Symbols

#map, functions 3-18, 6-3

#map, variables 5-4

A

afdsca 3-3, 4-3

algorithms

branch and bound

description 8-12

history plot 8-16

optimization summary 8-15

parameters 8-14

stopping messages 8-15

storage 8-14

crashes 8-4

essential information 8-3

exhaustive search

description 8-22

history plot 8-25

optimization summary 8-24

parameters 8-23

stopping messages 8-24

storage 8-23

failure to make progress 8-4

general information 8-2

generalized reduced gradient

description 8-5

history plot 8-9

optimization summary 8-8

parameters 8-6

stopping messages 8-8

storage 8-6

sequential quadratic programming

description 8-10

history plot 8-12

optimization summary 8-12

parameters 8-11

stopping messages 8-11

storage 8-11

simulated annealing

description 8-18

history plot 8-21

optimization summary 8-20

parameters 8-19

stopping messages 8-20

storage 8-19

stopping messages 8-5

allowable value 3-17, 6-5

Analysis file

example 3-8, 11-6

restoring 3-10, 11-2

saving 3-26, 11-2

analysis functions

definition 2-2

display 6-3

analysis variables

definition 2-2

display 5-3

ANASUB routines

example, Fortran 3-2, 4-2

anasubC routines

example, C 4-7

avdsca 3-3, 4-3

B

bound icon 3-13

branch and bound

description 8-12

history plot 3-69, 8-16

linearization 3-65

optimization summary 8-15

parameters 3-62, 8-14

stopping messages 8-15

storage 8-14

C

colors 1-8

compute functions pushbutton 5-8

constraint pushbutton 3-16, 6-3

continuous discrete pushbutton 5-3

contour graphs 10-5

conventional interface

C 4-5

fortran 4-2

D

design functions 6-4

#map value field 6-3

allowable value 6-5

constraint pushbutton 6-3

- definition 2-3
- display 6-3
- indifference value 6-5
- multiple objectives, reference 6-6
- null function 6-4
- objective pushbutton 6-3
- operation pushbutton 6-4
- design trial-and-error 3-10
- design variables
 - #map value field 5-4
 - adding 3-32
 - at bounds icon 5-3
 - continuous discrete pushbutton 5-3
 - definition 2-3
 - display 5-3
 - minimum and maximum values 5-5
 - null variable 5-4
- determine best method 7-9
 - example 3-23
- Discrete file 5-8
 - examples 3-53, 3-73, 5-8
 - reading 3-57
- discrete variables
 - filename 5-6
 - reference 3-56, 5-6
 - related 3-58, 5-7
 - row # value field 5-6

E

- error reference 12-1
- exhaustive search
 - description 8-22
 - history plot 8-25
 - optimization summary 8-24
 - parameters 3-71, 8-23
 - stopping messages 8-24
 - storage 8-23
- explore analysis
 - 1d, example 3-40
 - 1d, reference 9-2
 - 2d, example 3-42
 - 2d, reference 9-3
- Explore file 9-7
- explore optimum
 - 1d, example 3-46
 - 2d, example 3-49

- reference 9-3
- Explore window
 - 1d explore analysis 9-2
 - 2d explore analysis 9-3
 - explore optimum 9-3
 - reference 3-39, 9-1

F

- File window
 - new, remove, path 11-3
 - opening files 11-2
 - reference 3-5, 11-1
 - saving files 11-2
- files
 - Analysis
 - description 11-6
 - restoring 3-10
 - saving 3-26
 - Discrete 5-8
 - examples 3-53, 3-73, 5-8
 - Explore
 - 1d 9-7
 - 2d 9-8
 - History
 - reference 3-28
 - selection 2-11
 - Setup
 - example 11-7
 - restoring 3-37
 - saving 3-21

fonts 1-8

- Functions window
 - enlarging 6-1
 - paning 6-1
 - reference 3-16, 6-2

G

- generalized reduced gradient
 - description 8-5
 - history plot 8-9
 - optimization summary 8-8
 - parameters 8-6
 - stopping messages 8-8
 - storage 8-6
- gradients
 - all functions wrt all variables 3-88, 7-3

- color matrix 3-83, 7-5
- definition 3-80, 7-1
- design functions wrt design variables 3-82
- lagrange multipliers 3-91, 7-12
- numerical methods 7-6
- optimum 3-90, 7-12
- optimum, unmapped analysis variables 3-92, 7-12
- scaled 3-83, 3-84, 7-5
- unscaled 3-83, 7-5
- value matrix 3-83, 7-4
- Gradients window
 - determine best method 3-23, 7-9
 - display options 7-4
 - gradients box 7-3
 - numerical options box 7-6
 - optimum 7-12
 - reference 3-22, 3-82, 7-2
- graph
 - 2d contour
 - example 3-45, 3-51
 - reference 10-5
 - history
 - branch and bound 3-70, 8-17
 - exhaustive search 8-25
 - generalized reduced gradient 8-9
 - reference 10-2
 - sequential quadratic programming 3-30, 8-12
 - simulated annealing 8-21
 - sensitivity
 - example 3-41, 3-48
 - reference 10-2
- Graph window
 - 2d contour graphs
 - boundary values button 10-10
 - effect of mesh density 10-8
 - equal area contours 10-6
 - equal value contours 10-7
 - graphing data from an explore optimum 10-11
 - reference 3-44, 10-5
 - zooming in 10-7
 - hardcopy button 10-12
 - history graphs 10-2
 - reference 3-29, 10-1
 - sensitivity graphs 10-2
 - graphing data from an explore optimum 10-4
- H**
 - hardcopy for graph 10-12
 - hardcopy for variables 5-9
 - Help window 2-10
 - History file
 - reference 3-28
 - history graph 10-2
 - branch and bound 3-70, 8-17
 - exhaustive search 8-25
 - generalized reduced gradient 8-9
 - sequential quadratic programming 3-30, 8-12
 - simulated annealing 3-79, 8-21
- I**
 - indifference value 3-17, 6-5
 - installing OptdesX
 - DEC Ultrix, IBM AIX, HP UX 1-1
 - Silicon Graphics Irix 1-1
 - Sun OS 1-6
 - VAX/VMS 1-4
- L**
 - linking to OptdesX 3-1, 4-1
 - conventional interface, C 4-5
 - conventional interface, fortran 4-2
 - stand alone interface, C 4-16
 - stand alone interface, fortran 4-9
 - list widget 2-7
- M**
 - mapping, definition 2-3
 - maximum value 3-13, 5-5
 - minimum value 3-13, 5-5
 - multiple objectives
 - example 3-34
 - reference 6-6
 - solution method 6-7
 - weighting coefficients 3-35, 6-6
- N**
 - new file pushbutton 11-3

- example 3-5
- New File window
 - reference 3-7
- null function 3-19, 6-4
- null variable 5-4
- #map, functions 3-18, 6-3
- #map, variables 5-4

- O**
- objective pushbutton 3-16, 6-3
- objectives, multiple
 - example 3-34
 - reference 6-6
 - solution method 6-7
 - weighting coefficients 3-35, 6-6
- opening files 11-2
- operation pushbutton 6-4
- Optimization Options window
 - during optimization 8-26
 - files 8-27
 - reference 3-24, 8-1, 8-26
 - summary window options 8-27
- Optimization Summary window 8-28
- Optimize window
 - algorithms, general information 8-2
 - reference 3-24, 8-1
 - start button 8-2
- optimizing
 - branch and bound 3-61, 8-12
 - continuous variables 3-26
 - crashes 8-4
 - essential information 8-3
 - exhaustive search 3-71
 - failure to make progress 8-4
 - generalized reduced gradient 3-26, 8-5
 - global vs. local optima 8-4
 - noisy functions 8-4
 - sequential quadratic programming 8-10
 - simulated annealing 3-74, 8-18
 - stopping messages 8-5

- P**
- paning windows 2-8
- parameters
 - branch and bound 3-62, 8-14
 - exhaustive search 3-71

- generalized reduced gradient 8-6
- sequential quadratic programming 8-11
- simulated annealing 3-75, 8-19
- path pushbutton 2-11, 11-6
- postprocess pushbutton 5-8
- pushbutton widget 2-6

- R**
- radio box widget 2-6
- restoring an Analysis file 3-10

- S**
- saving files 11-2
- scale slider widget 2-8
- scaling
 - effects on algorithms 8-3
 - functions 2-5
 - using derivatives to check 7-6
 - variables 2-4
- sensitivity graphs 10-2
- sequential quadratic programming
 - description 8-10
 - history plot 8-12
 - optimization summary 8-12
 - parameters 8-11
 - stopping messages 8-11
 - storage 8-11
- Setup file
 - example 11-7
 - restoring 3-37
 - saving 3-21
 - saving discrete 3-59
- simulated annealing
 - description 8-18
 - history 3-79
 - history plot 8-21
 - optimization summary 8-20
 - parameters 3-75, 8-19
 - stopping messages 8-20
 - storage 8-19
- stand alone interface
 - C 4-12
 - fortran 4-5
- system command for VMS 4-12

- T**
- tabbing through value fields 2-9

text field widget 2-7
toggle button widget 2-6

V

Variables window
 compute functions 5-8
 discrete reference 3-56, 5-6
 enlarging 5-1
 hardcopy 5-8
 paning 5-1
 post process 5-8
 reference 3-13, 5-2
vector data types in ANAFUN 4-15

W

weighting coefficients 3-35, 6-6
widgets
 lists 2-7
 pushbutton 2-6
 radio box 2-6
 scale slider 2-8
 text field 2-7
 toggle button 2-6