The OptdesX design optimization software

A.R. Parkinson and R.J. Balling

Abstract This paper describes the OptdesX design optimization software. The software provides a design environment for optimization of engineering problems. The software supports interactive variable and function selection, optimization with continuous and discrete algorithms, design space graphics, tolerance analysis, and control of noise in numerical derivatives, as well as numerous other features. The software is described and illustrated in terms of a small example problem. The software is available on Unix platforms only.

Key words design optimization software, continuous and discrete optimization

1 Introduction

1.1 Description

The OptdesX (Optimal design for Xwindows) software provides an interactive design environment for exploration and optimization of an engineering model. Figure 1 shows the software interface for an example design session. Many of these windows will be explained in more detail in following sections.

Received December 30, 2000

A.R. Parkinson[1] and R.J. Balling[2]

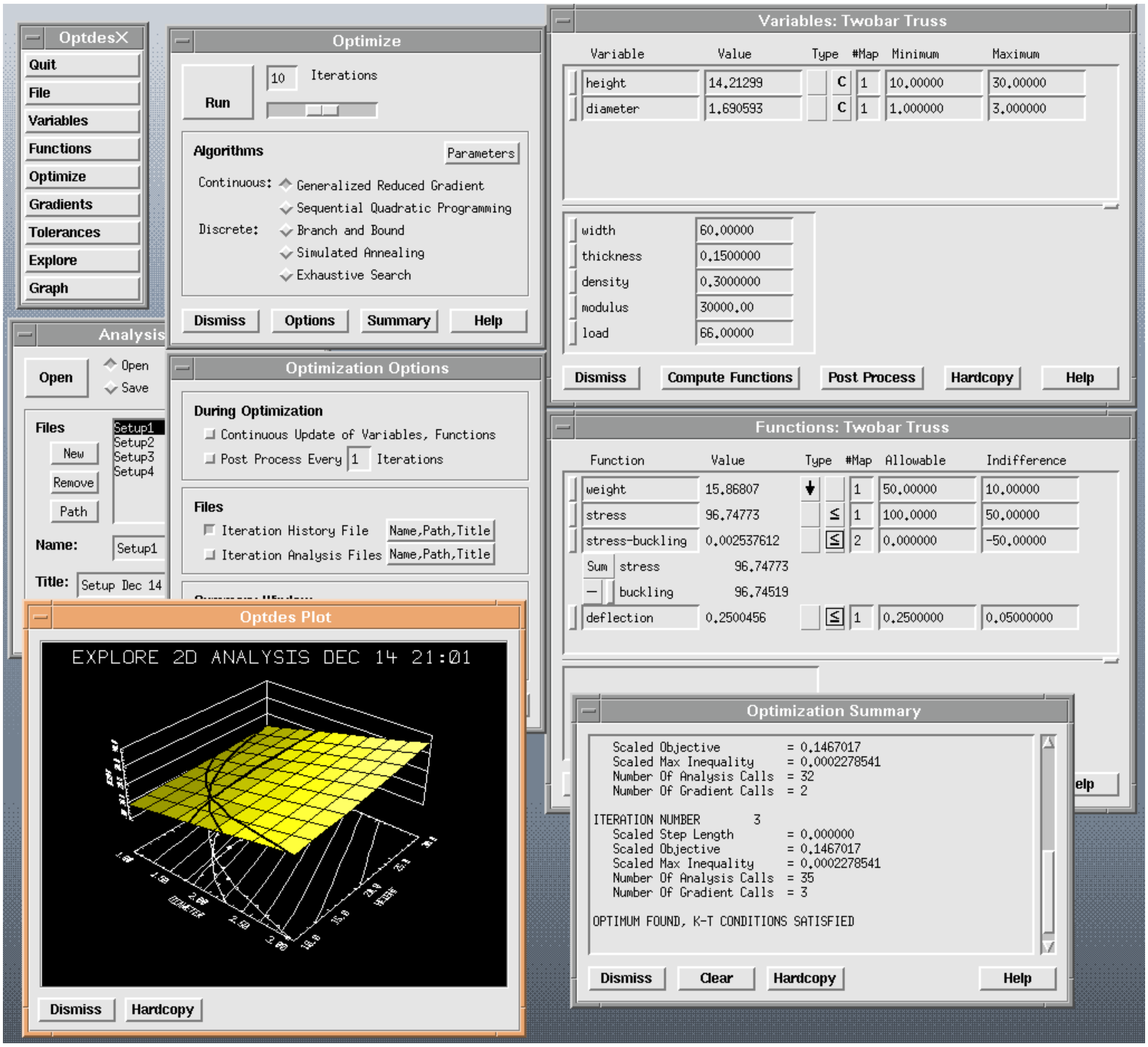
[1] Department of Mechanical Engineering, Brigham Young University, 435 CTB, BYU Provo, UT 84602, USA
e-mail: `parkinson@byu.edu`
[2] Department of Civil Engineering,Brigham Young University, 368 CB, BYU Provo, UT 84602, USA
e-mail: `balling@byu.edu`

The software allows a designer to do the following:

- adjust design variables to find a feasible design;
- adjust design variables to find an optimal design;
- optimize with multiple objectives and examine tradeoffs among competing objectives;
- optimize with both continuous and discrete variables, including components from catalogs;
- define design problems that include sophisticated mappings from analysis to design space;
- redefine a design problem 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;
- develop a design that is robust with respect to tolerances on model variables or parameters;
- minimize variation in the design as a constraint or objective; and
- periodically save or recall a promising design for further reference or optimization.

1.2 History

The OptdesX software was developed at Brigham Young University. The primary authors of the software are Alan Parkinson, Richard Balling and Joseph Free (retired), all professors of Mechanical or Civil engineering at Brigham Young, and numerous students. The first version of the software, Optdes 1.0, was created in 1983. This gave way to versions 2.0–4.0, which all had a text-based user interface. OptdesX 1.0, with a graphical user interface, was introduced in 1989. It was followed by OptdesX 2.0 in 1993 and version 2.0.4, the current version of the software, in 1995. The software is available from Brigham Young University.



Optimize: 10 Iterations

Algorithms
Continuous: Generalized Reduced Gradient; Sequential Quadratic Programming
Discrete: Branch and Bound; Simulated Annealing; Exhaustive Search

Analysis: Open / Save
Files: Setup1, Setup2, Setup3, Setup4
Name: Setup1
Title: Setup Dec 14

Optimization Options
During Optimization
Continuous Update of Variables, Functions
Post Process Every 1 Iterations
Files
Iteration History File — Name,Path,Title
Iteration Analysis Files — Name,Path,Title

Variables: Twobar Truss

Variable	Value	Type	#Map	Minimum	Maximum
height	14.21299	C	1	10.00000	30.00000
diameter	1.690593	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	15.86807	↓	1	50.00000	10.00000
stress	96.74773	≤	1	100.0000	50.00000
stress-buckling	0.002537612	≤	2	0.000000	-50.00000
Sum stress	96.74773				
— buckling	96.74519				
deflection	0.2500456	≤	1	0.2500000	0.05000000

Optimization Summary

```
   Scaled Objective            = 0.1467017
   Scaled Max Inequality       = 0.0002278541
   Number Of Analysis Calls    = 32
   Number Of Gradient Calls    = 2

ITERATION NUMBER         3
   Scaled Step Length          = 0.000000
   Scaled Objective            = 0.1467017
   Scaled Max Inequality       = 0.0002278541
   Number Of Analysis Calls    = 35
   Number Of Gradient Calls    = 3

OPTIMUM FOUND, K-T CONDITIONS SATISFIED
```

Fig. 1 The OptdesX design environment

Licenses to the software have been sold to several hundred companies and universities in the United States and Europe. Feedback from users and the results from research conducted by the authors have been used to define the current version of the software.

1.3 Terminology

For the discussion that follows, it will be useful to define some terminology. The engineering model to be optimized is referred to as the *analysis* model. The inputs to this model are *analysis variables*. The outputs are *analysis functions*. For example, consider the design of a simple truss. The analysis variables for the truss are height, width, diameter, thickness, density, modulus of elasticity, and load. When these are all specified, the analysis model can compute the analysis functions, which in this case are the weight, stress, buckling stress and deflection.

From among the set of analysis variables, we select some to be optimized. We refer to these as *design variables*. From among the analysis functions, we select objectives and constraints. These are *design functions*. The specification of design variables and design functions is referred to as *the mapping between analysis space and design space*.

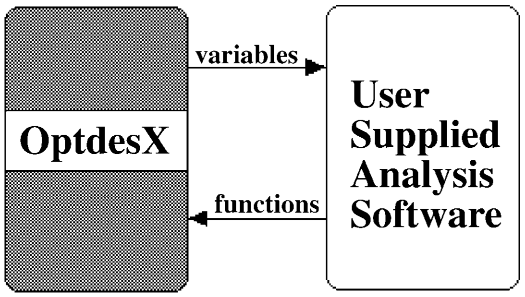


Fig. 2 Relationship between OptdesX and analysis software

2 Linking to the users analysis software

To optimize the analysis model, it must be linked to OptdesX software. The relationship between the model and OptdesX is illustrated in Fig. 2. As OptdesX optimizes, it changes the values of the analysis variables; the model calculates the corresponding functions and passes these back to the software.

The user supplied analysis routine is supplied in a routine called anafun. This can be coded in either C or Fortran; a C version of the anafun routine for the truss problem is given in Table 1

Table 1 Analysis routine for the simple truss problem

```
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 scalar AF values */
   wght = 2. * dens * area * leng;
   strs = load * leng / 2. / area / hght;
   buck = (PI * PI * modu * iOverA ) /
    (leng * leng);
   defl = load * pow(leng,3.) / 2. / modu / area /
    (hght * hght);

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

}
```

The function calls at the beginning of the routine, e.g. `avdsca(&hght, height)`, extract the variable values from the OptdesX database and make them available to the analysis routine. The acronym “avdsca” stands for “analysis variable double scalar.” After all the variables have been extracted, the calculations of the model are completed and the analysis functions are passed back to OptdesX in statements such as
`afdscaC(wght, ‘‘weight’’)`.

OptdesX communicates with the analysis model using Interprocess Communication (IPC). Support routines for IPC are included with the software, this communication is transparent to the user. One consequence of IPC is that OptdesX and the analysis model are separate executable programs that run concurrently. Besides Fortran or C, the model may also be developed in commercial software. In the case of commercial software a special dummy analysis program is used which executes the commercial code. Detailed instructions for this situation are given in the users manual.

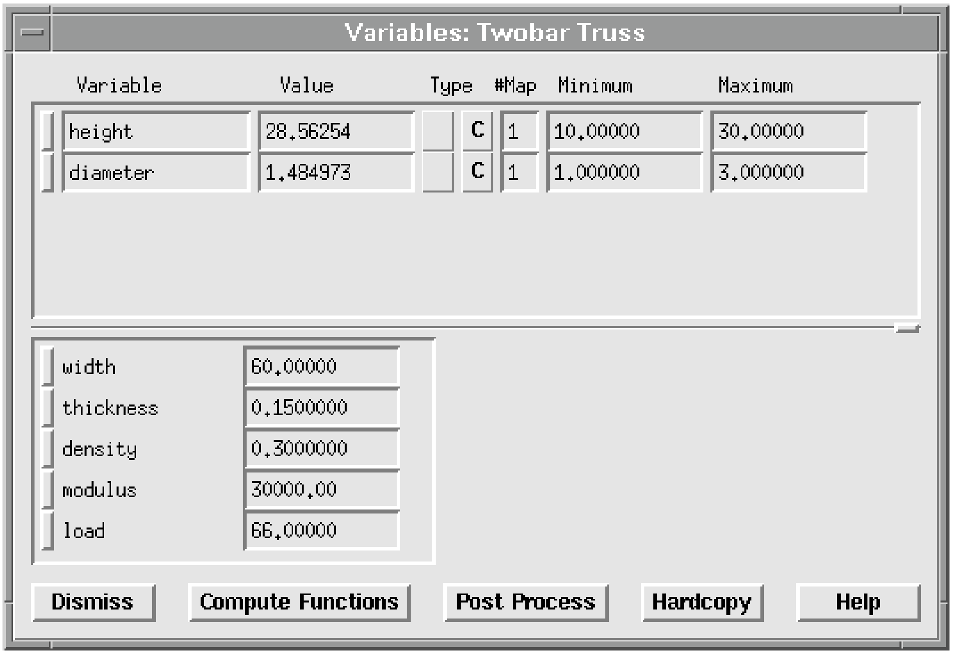
3 Problem setup

3.1 Introduction

Once the analysis model has been linked to OptdesX, the user is ready to begin problem set-up. This involves selecting the design variables (optimization variables) and design functions (objectives and constraints). This set-up is accomplished using point and click operations in the variables and functions windows, shown in Figs. 3 and 4. All analysis variables initially appear in the bottom half of the variables window; by clicking on the mapping button (small button to the left of a label) the variable appears in the top half of the window as a design variable. In a similar manner, design functions are chosen from among the analysis functions in the functions window. Further mouse clicks select a function to be an objective, a constraint or both. In Figs. 3 and 4, height and diameter have been chosen as design variables, weight and stress have been selected as objectives, and stress and deflection have been made constraints. Obviously many different optimization problems can be set-up from the same model, i.e. there can be many different mappings. OptdesX allows the user to store and restore different problem set-ups.

3.2 Scaling

As is shown in Fig. 3, for design variables the user specifies a minimum and maximum value. Besides being con-

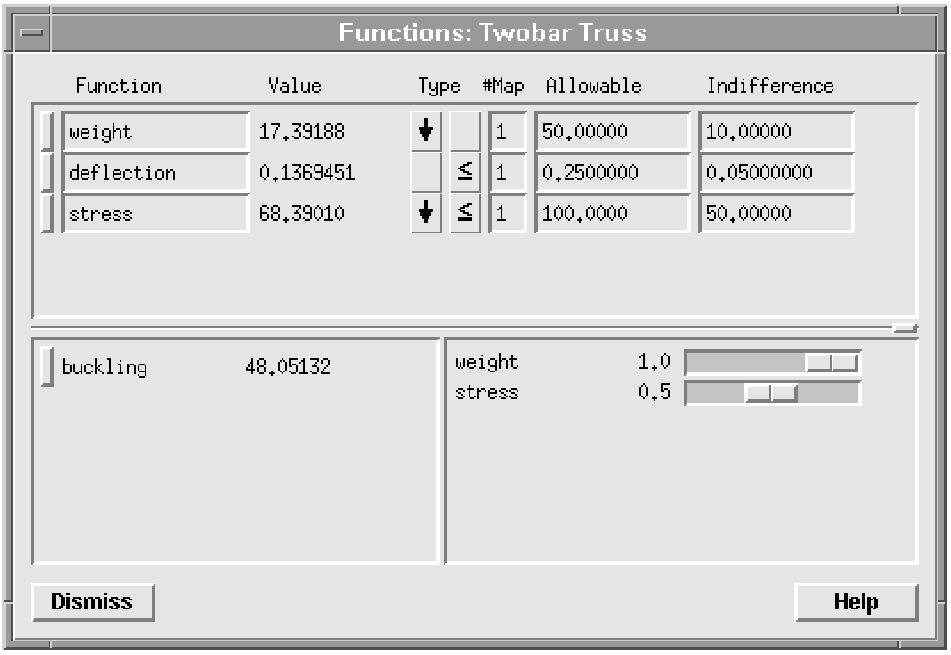


Variables: Twobar Truss

Variable	Value	Type	#Map	Minimum	Maximum
height	28.56254	C	1	10.00000	30.00000
diameter	1.484973	C	1	1.000000	3.000000

width	60.00000
thickness	0.1500000
density	0.3000000
modulus	30000.00
load	66.00000

Fig. 3 The variables window in OptdesX. Design variables are height and diameter. Width, thickness, density, modulus, and load are analysis variables and will remain constant during optimization



Functions: Twobar Truss

Function	Value	Type		#Map	Allowable	Indifference
weight	17.39188	↓		1	50.00000	10.00000
deflection	0.1369451		≤	1	0.2500000	0.05000000
stress	68.39010	↓	≤	1	100.0000	50.00000

buckling	48.05132

weight	1.0
stress	0.5

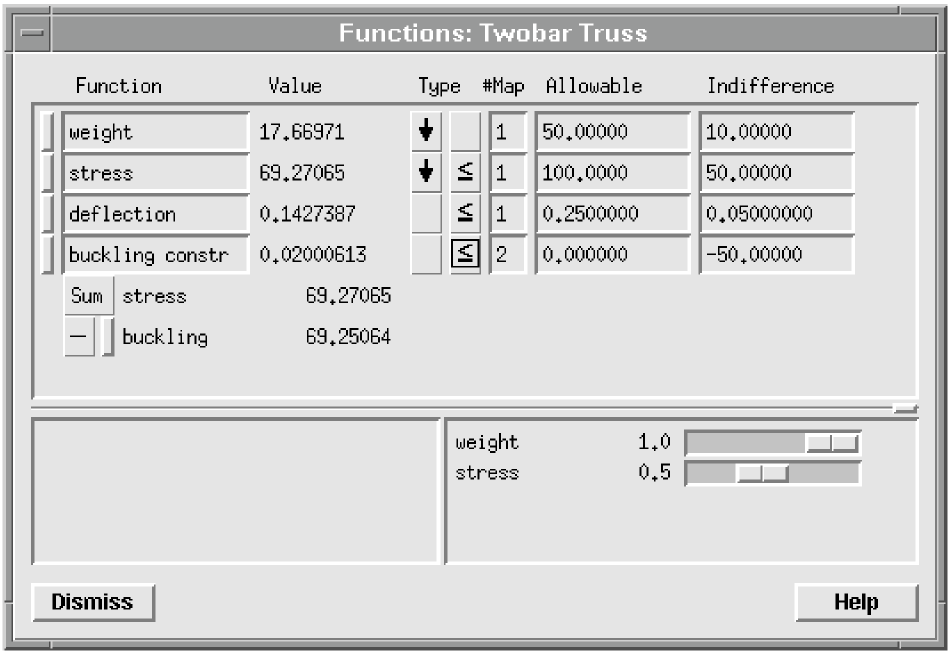
Fig. 4 The functions window in OptdesX. Weight and stress are objectives. Deflection and stress are constraints. Buckling stress is not mapped in this example

sidered to be firm bounds, these values are used to scale the design variables internally to be between 1.0 and +1.0. In like manner, the user specifies allowable and indifference values for design functions. The allowable value is the limiting value of a function, i.e. it is the right-hand side of a constraint if the function is made a constraint; the indifference value is the "best" value the user feels can be achieved; if there are multiple objectives the indifference value is the value at which the user is indifferent to further improvement and would prefer to see other objectives improved instead. The allowable and indifference values are also used to scale functions internally to be on the order of one. We have found that good scaling can make a significant difference in the ability of the algorithms to reach an optimum for engineering problems.

3.3 Sophisticated set-up features

The software contains a number of special set-up features. Several analysis variables can be mapped or linked to one design variable. This makes it possible to take advantage of symmetry or other constraints and reduce problem size. Combinations of functions can be made one design function. This is illustrated in Fig. 5 below, where the difference in stress and buckling stress have been made one constraint. The analysis function stress now appears in two constraints.

Of special note is the method for specifying discrete variables. In the same way an analysis variable is made a design variable, a design variable can be declared to be a discrete variable. (In this paper we define discrete to mean the values may be integer or selected from a finite set of real numbers.) Further, several design variables



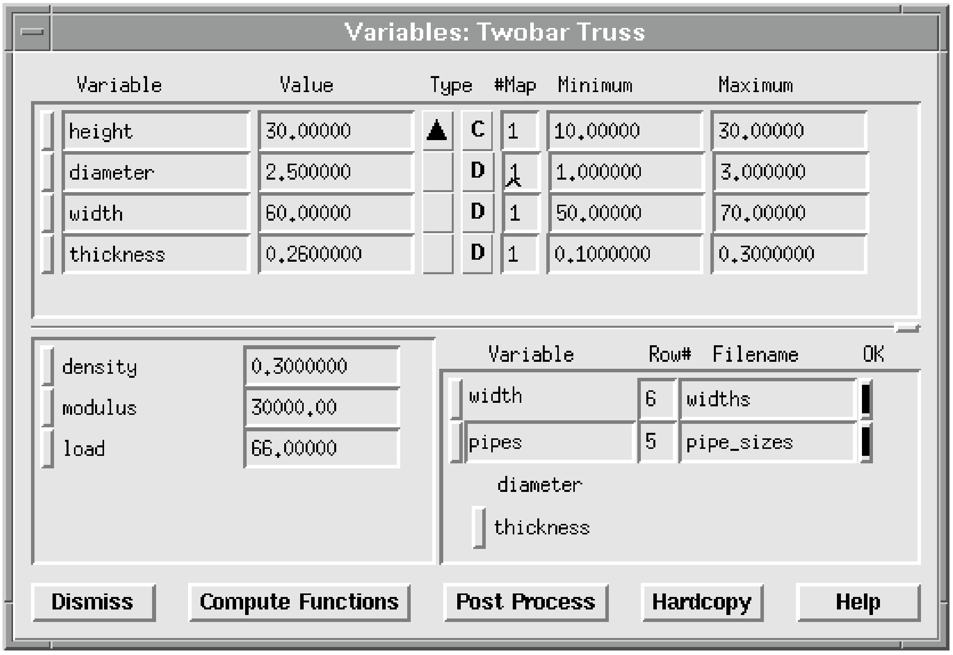
Functions: Twobar Truss

Function	Value	Type		#Map	Allowable	Indifference
weight	17.66971	↓		1	50.00000	10.00000
stress	69.27065	↓	≤	1	100.0000	50.00000
deflection	0.1427387		≤	1	0.2500000	0.05000000
buckling constr	0.02000613		≤	2	0.000000	-50.00000

Sum	stress	69.27065
−	buckling	69.25064

weight	1.0
stress	0.5

Fig. 5 Function set-up showing two analysis functions mapped together to create the buckling constraint



Variables: Twobar Truss

Variable	Value	Type		#Map	Minimum	Maximum
height	30.00000	▲	C	1	10.00000	30.00000
diameter	2.500000		D	1	1.000000	3.000000
width	60.00000		D	1	50.00000	70.00000
thickness	0.2600000		D	1	0.1000000	0.3000000

density	0.3000000
modulus	30000.00
load	66.00000

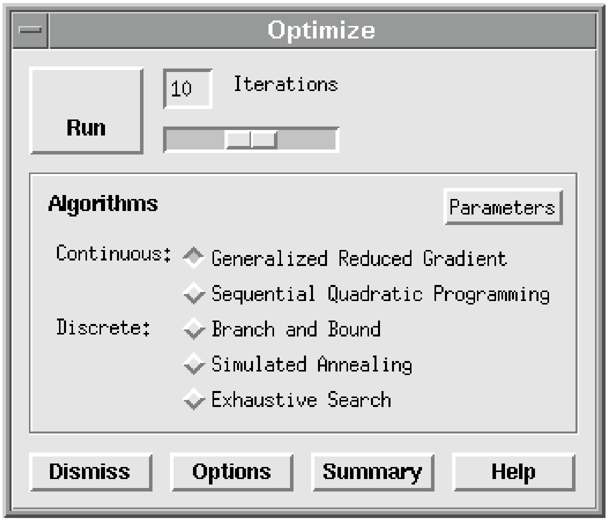
Variable	Row#	Filename	OK
width	6	widths	▌
pipes	5	pipe_sizes	▌

diameter
thickness

Fig. 6 A problem set-up showing discrete variables, "width" and "pipes". Diameter and thickness have been made a "related discrete variable" called "pipes"

can be mapped together to form a *related discrete variable*. An example of a related discrete variable is shown in Fig. 6. Here diameter and thickness have been mapped to be one discrete variable, "pipes", the values for which are found in the file "pipe_sizes". For example, if we wished to use standard size pipes, we would find that diameter and thickness for standard pipes would not only be discrete, but would also be related to each other; as the diameter of standard size pipes increases so do the available thicknesses. Thus each discrete value of pipes maps a value to diameter and thickness.

Related discrete variables make it possible to optimize with values from vendor catalogs. It is very common in engineering design to take advantage of off-the-shelf components such as pipes, springs, motors, valves, bearings, gears, I-beams, etc. The results of a continuous optimization are often impractical because they require the design to be custom-made. A feature such as related discrete variables helps to bridge the gap between the practical needs of the designer and the theoretical results of algorithms. Further information about optimizing with related discrete variables can be found in the paper by Hager and Balling (1988).



Iterations: 10

Algorithms

Continuous: [✓ Generalized Reduced Gradient]
Sequential Quadratic Programming

Discrete: Branch and Bound
Simulated Annealing
Exhaustive Search

Fig. 7 OptdesX algorithm selection box

4
Algorithms

4.1
Introduction

OptdesX supports optimization of continuous, mixed and pure discrete problems. The algorithm selection box is shown in Fig. 7. The software is designed to solve small to medium-sized highly nonlinear problems. The algorithms do not use any sparse matrix methods. A practical upper limit on problem size, governed somewhat by the interface, would be a problem of about 100 variables and 100 functions.

4.2
Algorithms for continuous problems

As is shown in Fig. 7, the main algorithms for continuous optimization are generalized reduced gradient (GRG) and sequential quadratic programming (SQP). The GRG algorithm is actually a GRG-SQP hybrid developed at BYU (Parkinson and Wilson 1988). The algorithm relies on many of the features described by Lasdon *et al.* (1978). The algorithm partitions the design variables into two sets: independent variables adjusted to improve the objective and dependent variables adjusted to stay feasible. At each step of the line search, the dependent variables are adjusted to maintain feasibility using a Newton-Raphson method. The reduced Hessian of the independent variables is updated with a BFGS update. The algorithm has been tuned to reduce the number of times the analysis model is evaluated and has been found to use up to 80% fewer function calls than other GRG algorithms. The algorithm can start from an infeasible point, using a search strategy to reduce the most violated constraint and adding it to the feasible constraint set when it becomes feasible. When all constraints have been made feasible, it reverts to optimization of the original problem. GRG is more tolerant of noise in the users analysis model than SQP and therefore is the first choice in OptdesX for optimizing engineering models. The SQP algorithm is based directly on Powell's description (Powell 1978), with the QP solver coded according to the method given by Goldfarb and Idnani (1978). Both algorithms require derivatives which are obtained numerically. Both only guarantee finding a local optimum.

4.3
Algorithms for mixed or discrete problems

There are three algorithms available for mixed or discrete problems: branch and bound (BNB), simulated annealing (SAN), and exhaustive search (EXS). All three algorithms can be used on pure discrete problems or on mixed problems; however, BNB requires that the problem be able to be modelled as continuous since a continuous algorithm (GRG) is used to obtain the lower bound on cost required by BNB.

The BNB algorithm is based on the classical description of the algorithm. It uses a best first search strategy for evaluating nodes. It has two modifications made to increase its efficiency. The user can select a "neighbourhood" of discrete values around the continuous optimum rather than considering all discrete values, and the continuous subproblems can be linearized so that no analysis calls to the model are required. These efficiencies can reduce the computation associated with the BNB tree by more than 99% on realistic-sized problems.

Simulated Annealing (SAN) (Kirkpatrick *et al.* 1983) can be used to solve continuous, mixed or discrete problems. A main advantage of SAN is that is does not require derivatives and so can be used when discrete variables cannot be modelled as continuous variables or if the model contains a lot of discontinuities. Simulated annealing rejects any designs which are infeasible. It also has a feature to reduce the number of designs it generates which are infeasible, according to predictions from a linear model.

Exhaustive search of course enables the user to consider all possible discrete combinations. It relies on GRG for continuous optimization if the problem is mixed.

4.4 Optimization procedure

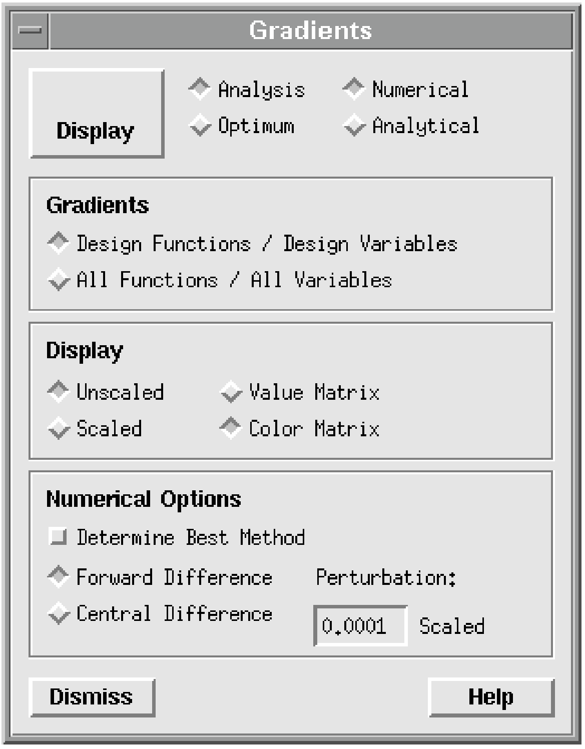
OptdesX was developed to be an interactive optimization environment. After problem set-up has been completed, the user selects the algorithm to be used, adjusts any algorithm parameters, and then selects the run button shown in Fig. 7. An optimization summary window opens which provides information about the run, and the values in the variables and functions windows change in real time as the optimization proceeds.

4.5 Parameters and termination criteria

Each algorithm has parameters which the user may adjust, although most users elect to use the defaults. These parameters are shown in Table 2.

5 Gradients

OptdesX computes all derivatives numerically. The derivatives can be displayed as either a “colour matrix” or “value matrix”. Figure 8 shows the options available in the gradients window. (An example value matrix for a case study is shown in Fig. 12.)



[✓ Analysis] Optimum
[✓ Numerical] Analytical

Gradients
[✓ Design Functions / Design Variables]
All Functions / All Variables

Display
[✓ Unscaled] Scaled
Value Matrix [✓ Color Matrix]

Numerical Options
Determine Best Method
[✓ Forward Difference]
Central Difference
Perturbation: 0.0001 Scaled

Fig. 8 Gradients display window

Engineering models often suffer from noisy functions. Noise can be defined as a lack of significant figures in

Table 2 Parameters and termination criteria

Algorithm	Parameters
GRG	Constraint tolerance, Objective tolerance, Initial step length, Minimum step length, Maximum iterations
SQP	Converge tolerance, Maximum step length, Maximum iterations
BNB	Linearization option, Neighborhood selection of discrete values
SAN	Maximum perturbation, Initial and Final probability of selecting a worse design, Constraint tolerance, Feasibility filter tolerance, Number of cycles
EXS	Linearization option, Neighbourhood selection of discrete values, Number of combinations
Termination criteria are as follows:	
Algorithm	Termination Criteria
GRG	Kuhn-Tucker conditions satisfied, Objective function not changing, Minimum step size reached, Maximum iterations reached
SQP	Convergence tolerance satisfied (see above for explanation), Maximum iterations reached
BNB	Branch and bound search completed (all non-pruned branches examined)
SAN	Number of specified cycles completed
EXS	Search completed (all possibilities within neighborhoods examined)

the functions, regardless of how many values are reported. This noise is amplified in numerical derivatives and may result in very large errors which can be fatal for derivative-based algorithms. These errors can be reduced by determining the optimal numerical method and perturbation value. A unique feature of the software is the "Determine Best Method" option, shown in Fig. 8, which attempts to estimate the noise in the analysis model and determine the best perturbation and method. This feature uses a five point derivative to estimate error in forward or central difference derivatives and to suggest the optimum method and perturbation.

Derivatives of the optimum, such as Lagrange multipliers, can also be displayed.

6 Graphics

The software supports several different kinds of graphics, including optimization history plots, 1-D sensitivity plots, 2-D contour plots, and 3-D contour plots. An example 2-D contour plot is given in Fig. 9 below (other design variables, besides the two being plotted, are held constant for the construction of the graph). Although contour plots can be expensive to generate in terms of calls to the analysis model, it is our experience that designers always wish to see pictures of the design space if they can afford the computational expense.

Besides contour plots, it is possible to generate "explore optimum" plots. These are graphs showing how the optimum changes as a function of two analysis variables. Whereas for contour plots the analysis model is called at each grid point of the underlying mesh, for an explore optimum plot an optimization is performed at each grid point. Thus these plots can be an order of magnitude more expensive to create than regular sensitivity or contour plots.

7 Tolerances

Another feature of the software directed specifically towards engineering design is the ability to determine a robust optimum, i.e. to determine how much we must back off the optimum design in order to take into account tolerances on variables. Both statistically distributed and worst case tolerances can be considered. Further information about the robust design capability of the software is given by Parkinson (1995), Emch and Parkinson (1994), Lewis and Parkinson (1994).

8 Example applications

The OptdesX software has been applied to the design of turbine engine parts, heat sinks, heat exchangers, linkages, cams, compliant mechanisms, composite lay-up, structures, hydroelectric dams, control systems, automobile suspensions, chemical processes, aerodynamic lifting bodies, and weapons design, to name a few examples.

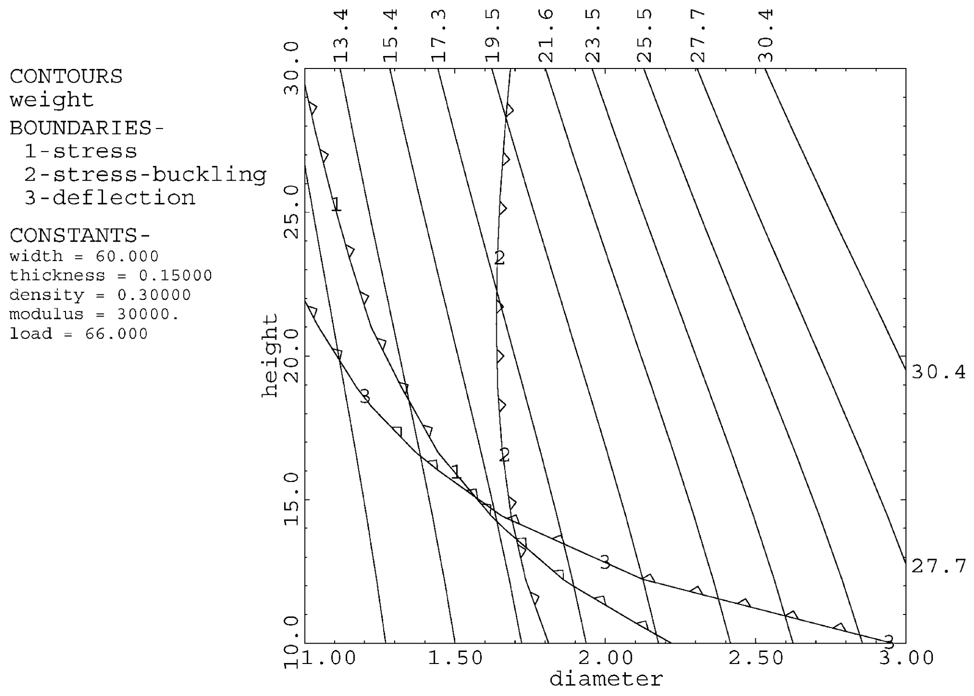


Fig. 9 Example 2D contour plot. Arrowheads on constraint boundaries point to feasible space

Table 3 Spring design example

Objective	Maximize force of spring at preload height
Design variables	coil diameter, wire diameter, number of coils, free height
Other analysis variables	preload height, shear modulus, endurance limit, safety factor, other material properties
Constraints	Stress at solid height must be less than yield strength
	Alternating stress must be less than endurance limit divided by safety factor
	Mean plus alternating stress must be less than yield strength divided by safety factor
	A clash allowance must be provided to insure spring does not reach solid height in service
	Ratio of coil diameter to wire diameter must be between 4 and 16
	Overall width of spring is limited

9 Case study

We will present a simple case study using the software for the design of a spring. To make the process clear, we will note each step. The actual modelling equations will not be given since they are not important in understanding the process of linking to OptdesX and executing the software.

Table 4 Subroutine ANAFUN Fortran model for the spring problem

```
      SUBROUTINE ANAFUN

      implicit double precision  (a-z)

cget values from OptdesX

      call avdsca(wd,''Wire Diameter'')
      call avdsca(cd,''Coil Diameter'')
      call avdsca(cnum,''Number of Coils'')
      call avdsca(hf,''Free Height'')
      call avdsca(ho,''Preload Height'')
      call avdsca(def,''Deflection'')
      call avdsca(g,''Shear Modulus'')
      call avdsca(se,''Endurance Limit'')
      call avdsca(sf,''Safety Factor'')
      call avdsca(q,''Q'')
      call avdsca(w,''W'')

c...calculate model equations

   .
   .
   .

csend function values back to OptdesX

     call afdsca(fpre,''Preload Force'')
     call afdsca(ta,''Alt Stress'')
     call afdsca(tm,''Mean Stress'')
     call afdsca(tsol,''Solid Stress'')
     call afdsca(dratio,''Diameter Ratio'')
     call afdsca(delh,''Clash Allowance'')
     call afdsca(sy,''Yield Strength'')
     call afdsca(ssy,''YS/SF'')
     call afdsca(sse,''Endur Limit/SF'')
     call afdsca(dsum,''Diameter Sum'')

RETURN
END
```

We wish to determine the design of a spring which results in maximum force. The design variables, other analysis variables, objective, and constraints are summarized in Table 3.

9.0.1 Step 1. Code the model

In this case the analytical model is coded as a Fortran subroutine. Calls are provided to allow communication with OptdesX. These are statements at the beginning `''call avdsca''` to receive variable values from OptdesX and at the end `''call afdsca''` to send function values to OptdesX. A partial listing of the Fortran model is given in Table 4.

9.0.2 Step 2. Compile and link the model with OptdesX IPC routines

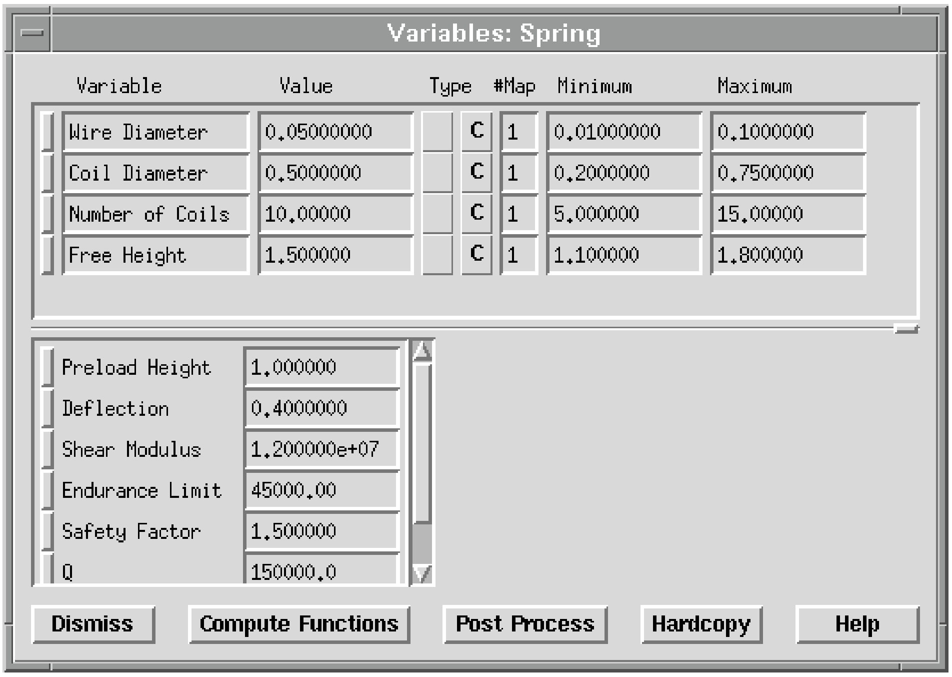
A Unix Makefile is provided for this purpose. The IPC routines allow OptdesX to communicate with the model while both run as independent programs.

9.0.3 Step 3. Start OptdesX and begin problem definition

Using point and click operations, the design variables (optimization variables) and design functions (objective and constraints) are mapped. The Variables and Functions windows are shown in Figs. 10 and 11 after the optimization problem is defined. The initial design is infeasible, as indicated by the constraint marker for the "mean stress" constraint which is white on black instead of black on white. That constraint is currently violated.

9.0.4 Step 4. Check the scaling and gradients

Scaled gradients are checked to make sure the scaling of the problem (as governed by the minimum and max-

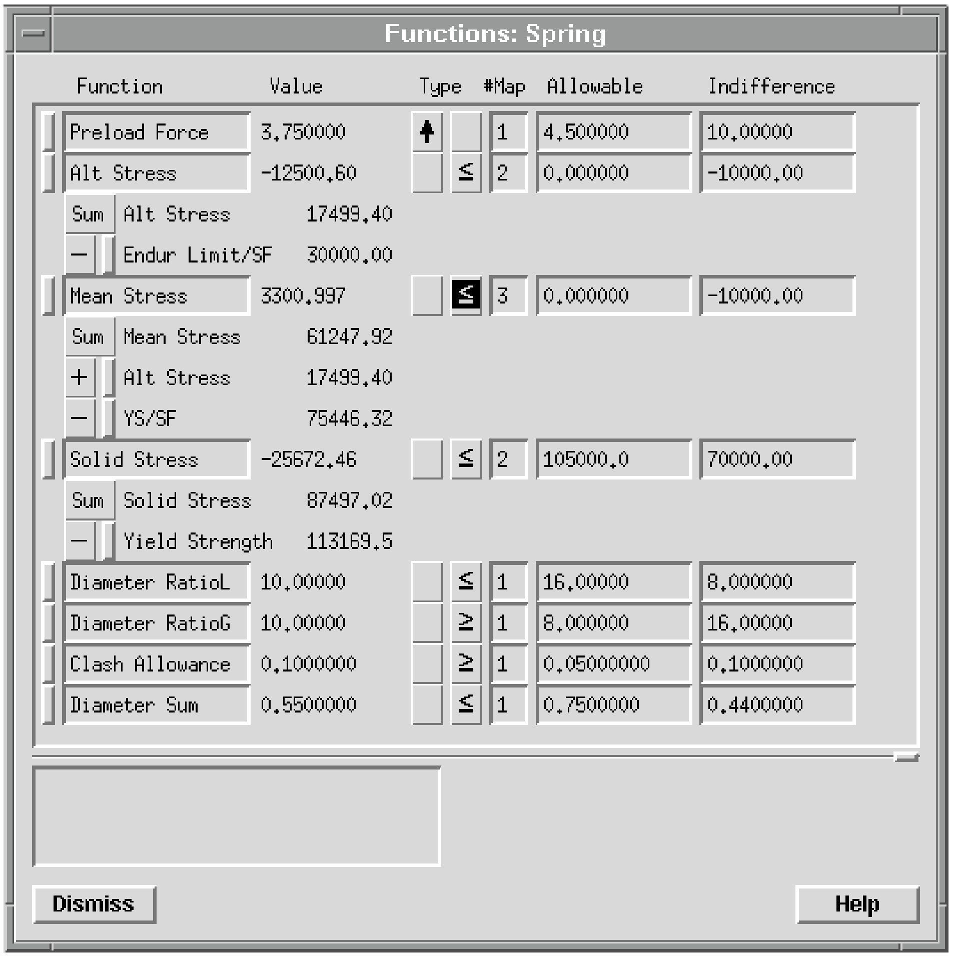


Variables: Spring

Variable	Value	Type	#Map	Minimum	Maximum
Wire Diameter	0.05000000	C	1	0.01000000	0.1000000
Coil Diameter	0.5000000	C	1	0.2000000	0.7500000
Number of Coils	10.00000	C	1	5.000000	15.00000
Free Height	1.500000	C	1	1.100000	1.800000

Preload Height	1.000000
Deflection	0.4000000
Shear Modulus	1.200000e+07
Endurance Limit	45000.00
Safety Factor	1.500000
Q	150000.0

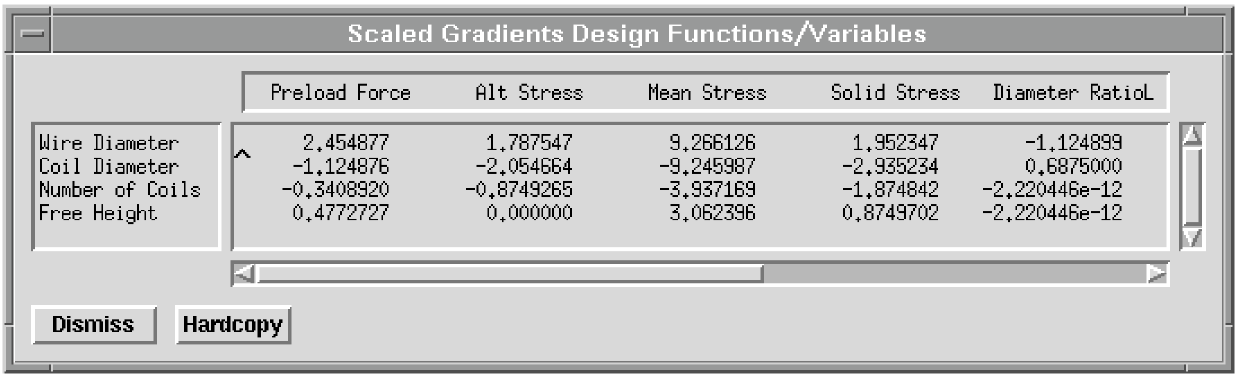
Fig. 10 Variables window after problem setup. Four design variables, in the upper portion of the window, have been selected



Functions: Spring

Function	Value	Type	#Map	Allowable	Indifference
Preload Force	3.750000	↑	1	4.500000	10.00000
Alt Stress	-12500.60	≤	2	0.000000	-10000.00
Sum Alt Stress	17499.40				
− Endur Limit/SF	30000.00				
Mean Stress	3300.997	≤	3	0.000000	-10000.00
Sum Mean Stress	61247.92				
+ Alt Stress	17499.40				
− YS/SF	75446.32				
Solid Stress	-25672.46	≤	2	105000.0	70000.00
Sum Solid Stress	87497.02				
− Yield Strength	113169.5				
Diameter RatioL	10.00000	≤	1	16.00000	8.000000
Diameter RatioG	10.00000	≥	1	8.000000	16.00000
Clash Allowance	0.1000000	≥	1	0.05000000	0.1000000
Diameter Sum	0.5500000	≤	1	0.7500000	0.4400000

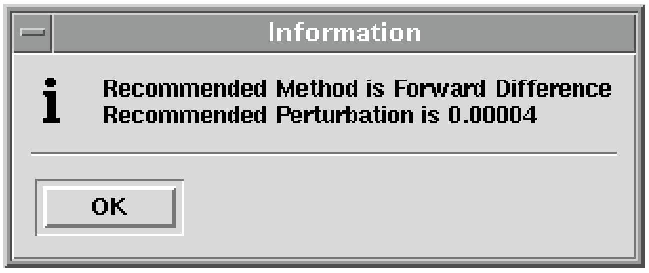
Fig. 11 Functions window after problem setup. The objective is to maximize preload force. Seven constraints have been defined. Mean stress is currently violated



	Preload Force	Alt Stress	Mean Stress	Solid Stress	Diameter RatioL
Wire Diameter	2.454877	1.787547	9.266126	1.952347	-1.124899
Coil Diameter	-1.124876	-2.054664	-9.245987	-2.935234	0.6875000
Number of Coils	-0.3408920	-0.8749265	-3.937169	-1.874842	-2.220446e-12
Free Height	0.4772727	0.000000	3.062396	0.8749702	-2.220446e-12

Fig. 12 Scaling of the problem is checked by examining the gradients in scaled form

imum values for variables and allowable and indifference values for functions) is adequate. These are shown in Fig. 12. Gradients should be the same order of magnitude when the problem is well-scaled. The gradient method and perturbation are found using the "determine best method" feature of the software. This information is given in Fig. 13.



Recommended Method is Forward Difference
Recommended Perturbation is 0.00004

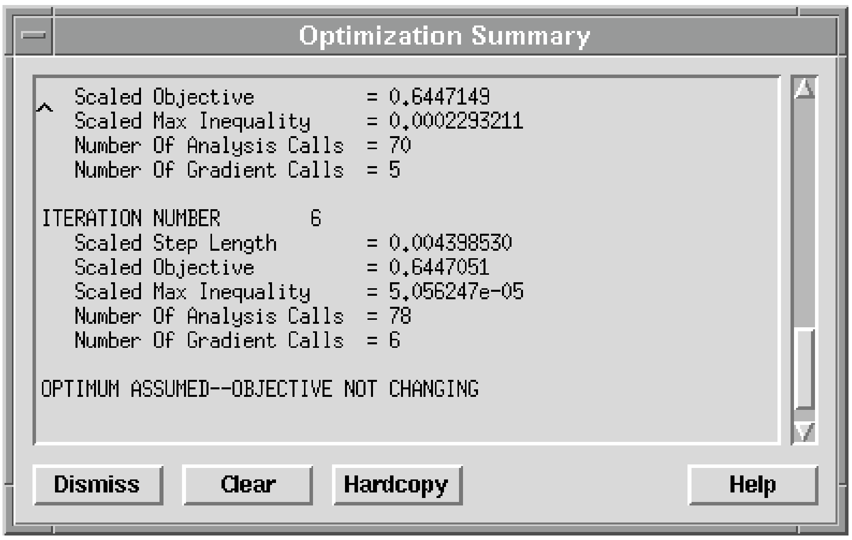
Fig. 13 The model is checked for noise and the gradient method and optimum perturbation are determined

9.0.5 Step 5. Optimize the model

We are now ready to begin optimization. We select the GRG algorithm since we know the model is smooth and continuous. GRG is the most robust of the algorithms. As we click on run (in the window shown in Fig. 7) we see the values change in real time. A summary window opens and provides information about the progress of the optimization, as given in Fig. 14. Preload force has increased and all constraints are satisfied (three constraints are binding).

9.0.6 Step 6. Explore "what if" questions and re-optimize

Using point and click operations, we can change the definition of the problem and see the effect on the design. We explore the design space. Optimization history and vari-

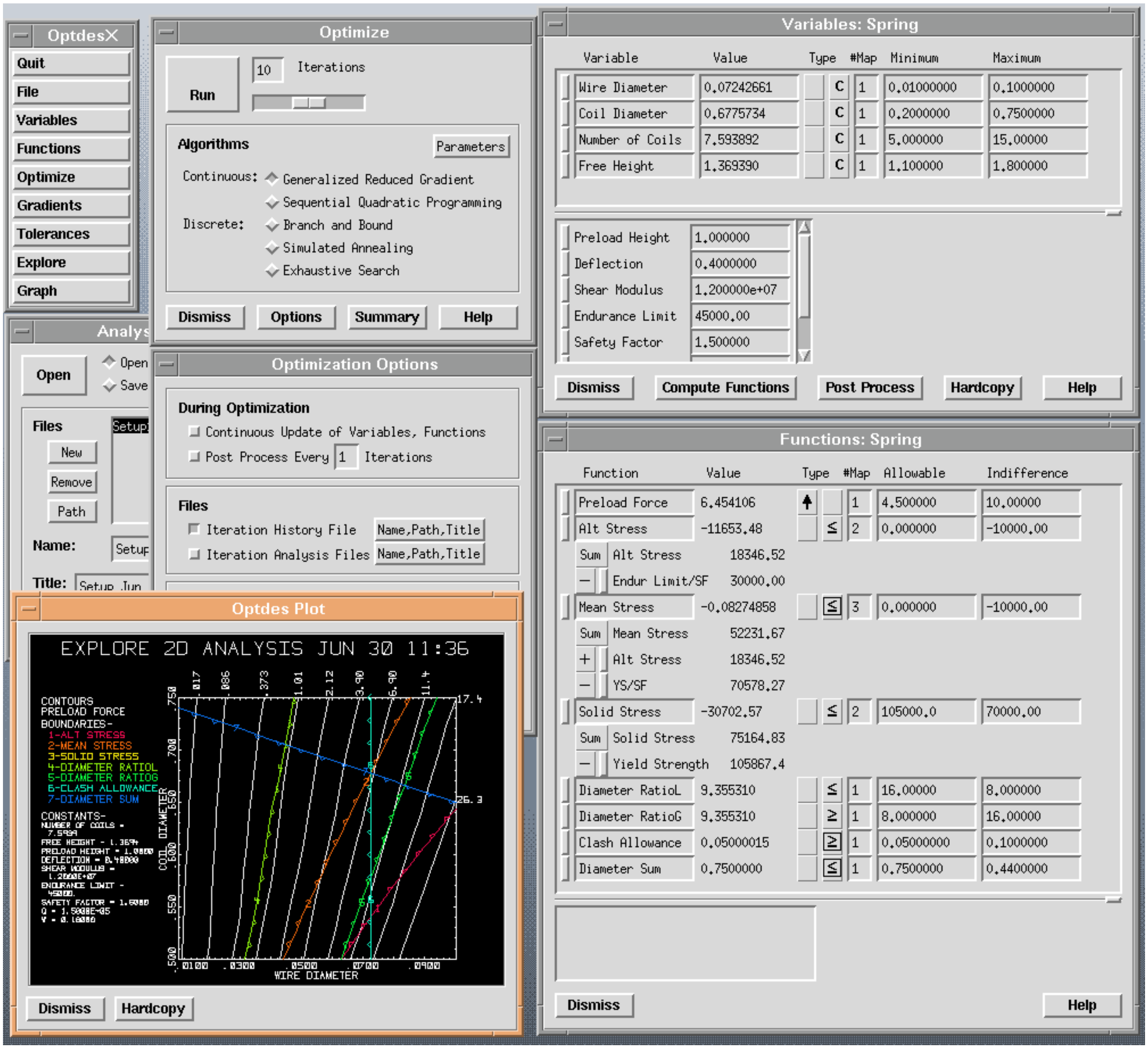


```
   Scaled Objective         = 0.6447149
   Scaled Max Inequality    = 0.0002293211
   Number Of Analysis Calls = 70
   Number Of Gradient Calls = 5

ITERATION NUMBER       6
   Scaled Step Length       = 0.004398530
   Scaled Objective         = 0.6447051
   Scaled Max Inequality    = 5.056247e-05
   Number Of Analysis Calls = 78
   Number Of Gradient Calls = 6

OPTIMUM ASSUMED--OBJECTIVE NOT CHANGING
```

Fig. 14 Optimization summary window



10 Iterations

Algorithms — Parameters

Continuous: ◆ Generalized Reduced Gradient
◇ Sequential Quadratic Programming
Discrete: ◇ Branch and Bound
◇ Simulated Annealing
◇ Exhaustive Search

During Optimization
☐ Continuous Update of Variables, Functions
☐ Post Process Every 1 Iterations

Files
☐ Iteration History File — Name,Path,Title
☐ Iteration Analysis Files — Name,Path,Title

Variable	Value	Type	#Map	Minimum	Maximum
Wire Diameter	0.07242661	C	1	0.01000000	0.1000000
Coil Diameter	0.6775734	C	1	0.2000000	0.7500000
Number of Coils	7.593892	C	1	5.000000	15.00000
Free Height	1.369390	C	1	1.100000	1.800000

Preload Height	1.000000
Deflection	0.4000000
Shear Modulus	1.200000e+07
Endurance Limit	45000.00
Safety Factor	1.500000

Function		Value	Type	#Map	Allowable	Indifference
Preload Force		6.454106	↑	1	4.500000	10.00000
Alt Stress		-11653.48	≤	2	0.000000	-10000.00
Sum	Alt Stress	18346.52				
–	Endur Limit/SF	30000.00				
Mean Stress		-0.08274858	$\boxed{\le}$	3	0.000000	-10000.00
Sum	Mean Stress	52231.67				
+	Alt Stress	18346.52				
–	YS/SF	70578.27				
Solid Stress		-30702.57	≤	2	105000.0	70000.00
Sum	Solid Stress	75164.83				
–	Yield Strength	105867.4				
Diameter RatioL		9.355310	≤	1	16.00000	8.000000
Diameter RatioG		9.355310	≥	1	8.000000	16.00000
Clash Allowance		0.05000015	$\boxed{\ge}$	1	0.05000000	0.1000000
Diameter Sum		0.7500000	$\boxed{\le}$	1	0.7500000	0.4400000

Fig. 15 The OptdesX design environment for the spring problem

ous representations of design space can be displayed. An example graph is shown in Fig. 15.

It should be emphasized that all of the foregoing windows are on the screen at the same time. Most operations can be executed at any time in virtually any order. Thus the design environment looks like Fig. 15.

10
Summary

The OptdesX software provides an interactive design environment that facilitates exploration and optimization of an engineering model. It is structured to allow a designer to quickly ask and answer many "what if" questions. Rarely, for example, is a designer content with the optimum to a particular problem. Rather the designer wishes to explore and understand the trade-offs which drive the design, the effects of adding or deleting constraints or changing allowable values, and the graphical nature of the design space. OptdesX enables a designer to do this very rapidly. By allowing the designer to explore a broad range of possibilities in a short time, OptdesX helps the designer to make good design decisions.

References

Emch, G.; Parkinson, A. 1994: Robust optimal design for worst-case tolerances. *ASME J. Mech. Des.* **116**, 1019

Goldfarb, D.; Idnani, A. 1978: A numerically stable dual method for solving strictly convex quadratic programs. *Math. Prog.* **27**, 1–33

Hager, K.; Balling, R.J. 1988: A new approach for discrete structural optimization. *ASCE J. Struct. Eng.* **114**, 1120–1134

Kirkpatrick, S.; Gelatt, D.D.; Vecchi, M.P. 1983: Optimization by simulated annealing. *Science* **220**, 671

Lasdon L.S.; Waren, A.D.; Jain, A.; Ratner, M. 1978: Design and testing of a generalized reduced gradient code for nonlinear programming. *ACM Trans. Math. Software* **4**, 34–50

Lewis, L.; Parkinson, A. 1994: Robust optimal design with a second order tolerance model. *Research Eng. Des.* **6**, 25–37

Parkinson, A. 1995: Robust mechanical design using engineering models. *J. Mech. Des.* **117**, 48–54

Parkinson, A.; Wilson, M. 1988: Development of a hybrid SQP-GRG algorithm for constrained nonlinear programming. *ASME J. Mech., Trans., Auto. Des.* **110**, 308

Powell, M.J.D. 1978: Algorithms for nonlinear functions that use Lagrangian functions. *Math. Prog.* **14**, 224–248