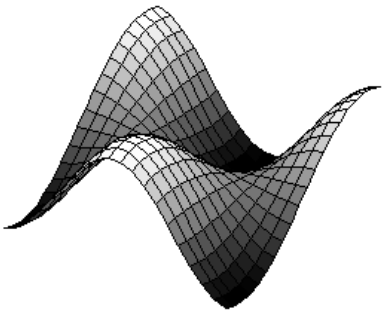


DOT

DESIGN
OPTIMIZATION
TOOLS

USERS MANUAL

Version 5.X

Vanderplaats Research & Development, Inc.

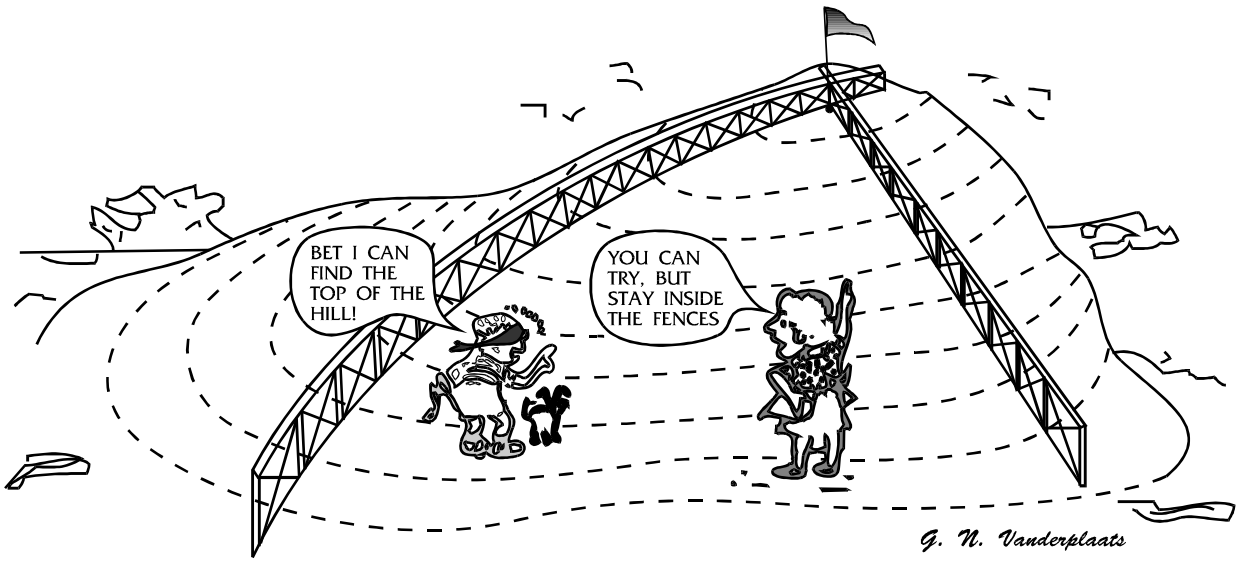
1767 S. 8th Street
Colorado Springs, CO 80906
http://www.vrand.com

Phone (719) 475-4998 FAX (719) 473-4638

OTHER VR&D SOFTWARE PRODUCTS

VisualDOC is a main ***D***esign ***O***ptimization ***C***ontrol program which replaces the DOT user's main program to greatly simplify coupling your analysis with optimization. The design problem is defined in the windows environment for convenient pre- and post-processing. VisualDOC provides additional features such as multi-objective and discrete variable optimization, approximations based on curve fits (response surface approximations), and much more.

GENESIS is a fully integrated finite element analysis and optimization program. Analysis capabilities include statics, normal modes, heat transfer, dynamic response and system buckling. Optimization capabilities include topology, member sizing and shape optimization. GENESIS uses the latest approximation techniques to gain exceptional efficiency in structural optimization. Typically, an optimum design is achieved using about ten detailed finite element analyses, even for design problems with thousands of variables and millions of constraints.



CONSTRAINED OPTIMIZATION

COPYRIGHT NOTICE

WARNING

DISCLAIMER

VR&D makes no representations or warranties with respect to the contents hereof and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose. Further, VR&D reserves the right to revise this publication and to make changes from time to time in the content hereof without obligation of VR&D to notify any person or organization of such revision or change.

TRADEMARKS MENTIONED IN THIS MANUAL

VisualDOC and DOT are trademarks of Vanderplaats Research & Development, Inc. All other trademarks are the property of their respective corporations.

Contents

WHAT'S NEW

DOT Version 5.X

DOT Version 5 represents a significant expansion and enhancement of Versions 4.x. Following is a partial list of enhancements contained in version 5.0:

- A few minor bugs have been removed to make the basic optimization capability more robust. Most of these fixes are transparent to the average user, but are important to the overall reliability of the program. We have solved thousands of optimization problems in our quest to make DOT as robust as possible.
- Significant enhancements have been made to the Sequential Quadratic Programming algorithm (METHOD=3) to improve efficiency and robustness.
- We have made major changes in the memory allocation routines to allow for the solution of larger problems. In the past, DOT allocated storage for two arrays, A and B. The key parameters defining this storage were NCOLA and NRB. The A matrix is used to store gradients of constraints, while the B matrix is used in the direction-finding sub-problem. The required dimensions are a function of the number of active and violated constraints, as well as the number of active side constraints. Because there is no way, in advance, to know the combination of general and side constraints that may be active, excessive storage was sometimes allocated. This sometimes limited the size of optimization tasks that could be solved. Version 5.xx overcomes this problem by allocating a single block of storage for both arrays and then internally adjusting storage needs as the optimization proceeds.
- Subroutine DOT510, which estimates memory requirements, has been greatly expanded. Now, DOT510 provides minimum, desired and maximum memory needs. The interactive program, DTSTOR, includes this new information.
- If NRWK is input to DOT and contains more storage than the "desired amount," the additional storage will be allocated to gradient storage and direction-finding calculations.
- The default value of IGRAD has changed. Now if IGRAD=IPRM(7)=0, DOT will choose IGRAD for maximum efficiency.
- We have modified the manual to describe new capabilities and changes.

- If the number of active and violated constraints exceeds available memory, DOT will reduce the retained set and attempt to continue with the optimization process.
- DOT is now distributed in object form only, due to numerous cases of theft of the source code in the past. We apologize for the inconvenience.
- An example has been added to Appendix A that describes how to call DOT from a C program.
- Estimates of the Lagrange multipliers are now provided at the end of the optimzation. These are stored in the first NCON locations of the WK array.
- This manual is also provided on the distribution media in electronic form.
- DOT is distributed on CD-ROM, together with VisualDOC. You are free to load VisualDOC also, and evaluate it at no cost.

Lagrange Multipliers

Lagrange multipliers, λ_j, are coefficients in the Kuhn-Tucker conditions,

$$\nabla F(\mathbf{X}) + \sum_{j=1}^{NCON} \lambda_j \nabla g_j(\mathbf{X}) = \mathbf{0}$$

These may be used to estimate the change in the optimum objective function due to a change in constraint g_j.

Assume we have a constraint R < S, which we formulate for DOT, in normalized form, as

$$g_j(X) = \frac{R}{S} - 1 \le 0$$

The Lagrange multiplier, λ_j, can now be used to estimate

$$\delta F = -\lambda_j\left(\frac{\delta R}{S}\right)$$

For example, if the optimum F = 100 for R = S = 5000, and $\lambda_j = 15$, then if we allow R to change to 6000 we get

$$\delta F = -15\left(\frac{1000}{5000}\right) = -3$$

Thus, for this simple example, an increase in the bound on R of 20% results in a 3% reduction in the objective function, F.

CHAPTER 1

Introduction

- Introduction
- What You Will Find in this Manual
- DOT System Requirements
- Installing DOT on Your Computer
- Getting Started
- Getting Familiar with DOT
- What DOT Does
- The General Optimization Problem
- Equality Constraints
- Special Notes

1.1 Introduction

Welcome to VR&D's Design Optimization Tools, DOT. The DOT program is intended to help you solve a wide variety of nonlinear constrained or unconstrained optimization problems.

Optimization concepts and methods are not new. Indeed, optimization is fundamental to most of what we do. Whether we are engineers, athletes, or businessmen, our goal is to be best in some way. Numerical optimization, which is the basis for the DOT program, helps us for those cases where we are able to define the optimization problem in a consistent mathematical or numerical way.

The DOT manual is intended for the new user of optimization, as well as the experienced user. In this first chapter, we start by defining the computer requirements and identifying the files that you have received. In Chapter 2, we discuss the use of DOT with your own application programs. This is the real power of DOT, and you are shown how to couple it with your own “analysis” programs to solve sophisticated design tasks. Chapter 3 describes a variety of options available to help you “tune” DOT to work efficiently for your particular application. Finally, Chapter 4 offers a variety of examples to help you gain familiarity with DOT and to insure that it is working properly. Chapter 5 lists several references for further study.

Appendices are provided to assist you with specialized questions and capabilities provided. Appendix E may be particularly useful if you are unfamiliar with optimization. This appendix provides an overview of the concepts as well as a reasonably detailed description of the mathematical algorithms contained in DOT.

1.2 What You Will Find in this Manual

This manual has been written with the first time user in mind. As such, the first two chapters are intended to assist in using the program right away.

This chapter first defines system requirements and lists the distribution files that you have received. The remainder of this chapter is an introduction to optimization itself. Chapter 2 is all about interfacing DOT with user-supplied application programs for powerful optimization capabilities. Chapter 3 discusses advanced uses of DOT such as over-riding internal parameters, directly supplying gradients of the objective function and constraints, interrupting and restarting DOT, and writing output to a special file for later use. Chapter 4 presents examples of optimization problems taken from a variety of disciplines. Chapter 5 is a list of references which may be useful to those seeking a better understanding of numerical optimization.

Appendix A gives a main calling program that may be used as a prototype for using DOT. Appendix B discusses the program DTSTOR and Subroutine DOT510, which calculates the minimum required working storage values of NRWK and NRIWK. Appendix C makes suggestions about what to do if you have trouble solving a particular optimization problem. Appendix D is a more detailed discussion of internal DOT parameters than is found in Chapters 2 and 3. Finally, Appendix E provides a somewhat detailed description of the optimization methods contained in DOT.

For detailed study of optimization methods and applications, the textbook, “Numerical Optimization Techniques for Engineering Design” by Dr. Vanderplaats is now available directly from VR&D. The third edition is significantly revised from previous editions. Please contact VR&D for details and pricing.

1.3 DOT System Requirements

Versions of DOT are available for all levels of computers. The system requirements are as follows:

1.3.1 Personal Computers

Operating System:

Windows 95/98, Windows NT 4.0 (Service Pack 3 or higher) or Windows 2000.

Memory:

At least 16 Mbytes of RAM and 10 Mbytes of hard disk storage. Additional memory and hard disk requirements depend on the size of your application program.

Compiler:

Digital Visual Fortran 5.0/6.0, Microsoft Powerstation Fortran 4.0, and/or Microsoft Visual C++ 5.0/6.0. DOT is written in FORTRAN 77. If you are using a C/C++ compiler to compile and link your application to the DOT library, you will need both FORTRAN and C/C++ compilers.

1.3.2 Workstations, Mini-Super and Super Computers

Operating systems:

SUN machines running Solaris 2.5 or above

SGI machines running IRIX 5.3 or above

HP machines running UX 10.2

IBM/RS6000 machines running AIX 4.0 or above

Cray supercomputers. Contact VR&D for details.

Memory:

At least 16 Mbytes of RAM and 10 Mbytes of hard disk storage. Additional memory and hard disk requirements depend on the size of your application program.

Compiler:

FORTRAN 77, FORTRAN 90, C/C++. If you are using a C/C++ compiler and link your applications to the DOT library, you will need both FORTRAN and C/C++ compilers.

1.4 Installing DOT on Your Computer

The CD-ROM you have received contains both DOT and VisualDOC. You are welcome to install VisualDOC also. It will operate in a very restricted mode immediately. If you provide VR&D with the appropriate machine information, you will receive an authorization file for limited, but less restrictive, use. Finally, on request, we can provide you with full DOT and/or VisualDOC capabilities for a short term evaluation.

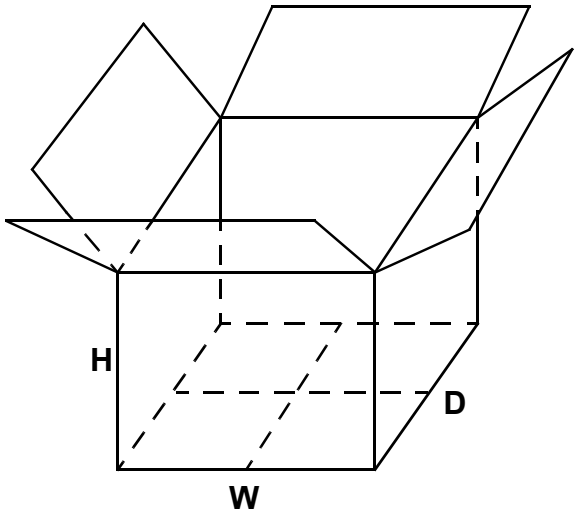
See the readme.txt file on the CD-ROM for the latest installation information.

1.5 Getting Started

DOT is a computer program for optimization. Specifically, it is used to automatically adjust parameters to maximize or minimize a calculated quantity, while satisfying a multiple number of constraints. Functions considered by DOT may be linear or nonlinear and may be very complicated implicit functions of the design variables. That is, you do not need to provide explicit equations to define the responses in terms of the design variables.

DOT is coupled with your application program by writing a small interface. DOT is a tool to solve system optimization problems in engineering, business, social science and any other applications where system responses are analyzed and evaluated numerically.

For example, suppose you received a large quantity order to produce cardboard boxes. The order specifies that the volume of the boxes must be greater or equal to 2 cubic feet, and both top and bottom surfaces have double flaps, as shown in the figure below.



Since one of the large cost items is the amount of cardboard, we need to decide the box dimensions to minimize the amount of cardboard used to make each box. A minimum material box will also be the lightest weight design, thus helping to reduce shipping costs.

This problem can be expressed in the following form.

Find the dimensions W, D, and H which will

Minimize the surface area, S, where

$$S = 2(HW + HD + 2WD) \quad \text{Objective Function} \tag{1-1}$$

Subject to:

$$\text{Volume,} \quad WHD \geq 2.0 \quad \text{Inequality Constraint} \tag{1-2}$$

$$W, H, D \geq 0.0 \quad \text{Side Constraints} \tag{1-3}$$

In this problem, the surface area is the “objective function”. The condition that the volume must be greater than or equal to 2.0 is called a “constraint”. The requirements that W, H, and D be greater than 0.0 are called “side constraints”. This description of an objective function and constraints constitutes the formal optimization problem.

At this point you may realize that to find the optimum design you have to adjust three parameters simultaneously, which is rather difficult to visualize. With DOT, you can solve this problem almost instantly.

Now let's make the problem a bit more realistic. The boxes will be cut from a sheet of cardboard, then folded and glued along one vertical corner. To do this requires an extra 1.25 inch of material of height H. Now the problem becomes

Find the dimensions W, D and H to

Minimize

$$S = 2(HW + HD + 2WD) + \frac{1.25H}{12.0} \tag{1-4}$$

Subject to:

$$\text{Volume,} \qquad HWD \geq 2.0 \tag{1-5}$$

$$W, H, D \geq 0.0 \tag{1-6}$$

Note that we have just added 1.25H/12.0 to the objective function where we divide by 12 to convert inches to feet. By doing so the problem has been made more difficult to solve by hand, although DOT solves it just as easily as before.

The method for solving general problems is to write a FORTRAN (or C/C++) program that calls DOT. The form of this program is defined in this manual and any experienced programmer will be able to quickly write such a program for most problems. Appendix A provides a simple program that can be used as a ‘template’ for using DOT.

The methods implemented in DOT are numerical search techniques known as mathematical programming. DOT represents the culmination of many years of research in system optimization by theoretical and applied mathematicians. Basic theories of mathematical programming are well known and many textbooks have been published. However, implementation of theoretical methods into reliable software has required a great deal of research and development. The principal author of DOT has been actively involved in software development as well as applications of system optimization methods since 1969.

Numerical optimization offers a number of improvements over the traditional approach to decision process and engineering design. Among the advantages of numerical optimization methods are:

- Perform system parameter adjustment far beyond human perception.
- Reduce the time required to make decisions.
- Provide a logical, systematic decision-making procedure.

- Virtually always improve system response, even if not arriving at the absolute optimal state.
- Not biased by intuition or experience. Hence it may produce new, non-traditional results.

The following is a brief list of engineering design projects to which serious application efforts have been made.

- Structural design for minimum weight. The GENESIS program from VR&D's partner company, VMA Engineering, is a fully integrated finite element analysis and optimization program. GENESIS uses advanced approximation techniques to solve the optimization task (performed by DOT) using only about ten detailed finite element analyses, even for very large optimization tasks.
- Aerodynamic design for maximum performance.
- Mechanical parts design for minimum material or for maximum performance.
- Conceptual aircraft, spacecraft and ship design.

Of course, applications are not limited to engineering design. Any system design or management problem involving numerical decisions may be cast in a form where numerical optimization is useful. Applications are limited only by the creativity of the user.

CAUTION

DOT solves the nonlinear optimization problem iteratively. It is designed to get a "near optimum" solution quickly, since in most practical problems a precise optimum (which takes much more computational effort) is not that meaningful. Therefore, you should not expect precise mathematical solutions. Chapter 4 gives examples of the differences between the theoretical optimum and that calculated by DOT for several cases. Usually, the difference is minor.

1.6 Getting Familiar with DOT

Now let's solve the simple box design of Section 1.5 using DOT. For convenience, the basic problem is restated here;

Minimize the surface area, S, where

$$S = 2(HW + HD + 2WD) \qquad \text{Objective Function} \qquad (1\text{-}7)$$

Subject to:

$$\text{Volume,} \quad WHD \geq 2.0 \qquad \text{Inequality Constraint} \qquad (1\text{-}8)$$

$$W, H, D \geq 0.0 \qquad \text{Side Constraints} \qquad (1\text{-}9)$$

The design variables, H, W and D are stored in the "Vector of Design Variables," **X**, so;

$$\mathbf{X} = \begin{Bmatrix} H \\ W \\ D \end{Bmatrix} \qquad (1\text{-}10)$$

We have three design variables, so NDV=3.

The objective function is, OBJ=S, so;

OBJ = 2(HW + HD + 2WD)

The inequality constraint is stored in the "Constraint Vector" **G** so;

G(1) = 1.0 - 0.5*X(1)*X(2)*X(3)

Note that the constraint G(1) is normalized by dividing by the bound (2.0) and converting to a non-positive inequality.

For this example, there is only one constraint, so NCON=1.

The side constraints are stored in the "Lower Bound Vector," **XL**, so;

$$\mathbf{XL} = \begin{Bmatrix} 0.0 \\ 0.0 \\ 0.0 \end{Bmatrix}$$

Because no upper bounds are prescribed, we will simply set the upper bounds on the design variables as 100.0 so the "Upper Bound Vector," **XU**, becomes;

$$\mathbf{XU} = \begin{Bmatrix} 100.0 \\ 100.0 \\ 100.0 \end{Bmatrix}$$

It is necessary to provide DOT with an initial design, so we will initially set the entries of vector **X** to unity for this example;

$$\mathbf{X} = \begin{Bmatrix} 1.0 \\ 1.0 \\ 1.0 \end{Bmatrix}$$

Note that this gives an initial volume of 1.0 ft.^{3}, which violates the minimum volume constraint of 2.0. This is acceptable in DOT and DOT will proceed to find a "feasible" optimum.

The vectors **X**, **G**, **XL** and **XU** are the basic arrays used by DOT to contain the user design information.

In this example, we could dimension **X**, **XL** and **XU** to 3 and **G** to 1, but will dimension each of them to 5. This is just to demonstrate that the required dimensions given in this manual are minimum dimensions. You may dimension arrays larger than the required value to allow for increasing the problem size in the future.

Additionally, the arrays **RPRM**(20) and **IPRM**(20) contain various control parameters. By initializing the contents to these arrays to zero, DOT will use all default parameters (defined in Chapter 3).

Also, arrays **WK** and **IWK** are used to store internal real and integer tables. These arrays are normally dimensioned rather large to insure sufficient memory is allocated. An interactive program called DTSTOR is provided to calculate the needed dimensions of these arrays, but normally we should just make them as large as possible.

Finally, we must tell DOT the number of design variables (NDV), the number of constraints (NCON), a print control parameter (IPRINT), whether to minimize or maximize (MINMAX=0 or -1 to minimize; +1 to maximize), the dimension of array **WK** (NRWK) and **IWK** (NRIWK) and an information parameter INFO. Initially, INFO=0. On return from DOT, if INFO=0, optimization is complete and **X** and **G** contain the optimum values of the design variables and constraints. If INFO=1, we calculate the value of the objective, OBJ, and constraints, **G**(I), I=1,NCON and call DOT again. In special cases, if we are providing gradient information to DOT, INFO=2 will be returned (if we use all default parameters, INFO will never equal 2, and DOT will calculate the gradients by finite difference methods). The parameter, METHOD, defines the optimization method to be used in DOT. Presently three methods are available for constrained optimization; METHOD=0 or 1 says use the Modified Method of Feasible Directions(MMFD), METHOD=2 says use the Sequential Linear Programming (SLP) method and METHOD=3 says use the Sequential Quadratic Programming (SQP) method. Two methods are available for unconstrained optimization (NCON=0). METHOD=0 or 1 says use the BFGS method and METHOD=2 says use the Fletcher-Reeves method.

It is required that we provide a main program to define the various information and call DOT, as well as a subroutine to calculate the objective and constraint functions in terms of **X**. The overall process is defined by the following 7 steps.

1. Dimension the required arrays and define the dimensions of **WK** and **IWK**.
2. Initialize the control parameters contained in **RPRM** and **IPRM** (normally set these to zero to use the default values; see Chapter 3).
3. Define the number of design variables and constraints, print control, method to be used, and whether to minimize or maximize.
4. Set the initial values of the design variables in **X**, the lower bounds, **XL**, and the upper bounds, **XU**.
5. Initialize the information parameter, INFO=0.
6. Call DOT to proceed with optimization.
7. On return from DOT, if INFO=0, terminate; the optimization process is complete. Otherwise, evaluate the objective and constraint functions and call DOT again. Eventually, DOT will return a value of INFO=0 to indicate that optimization is complete.

The FORTRAN listing for the box design problem is given below, along with a subroutine called EVAL which evaluates the functions (the function values could be calculated in the main program if you prefer). Note that arrays **X**, **XL**, **XU** and **G** are dimensioned larger than needed. It is only necessary to dimension these arrays large enough for the problem, but we can make them larger to allow for future expansion.

In this example, we used METHOD=3 (the SQP method), and print control, IPRINT=1. The resulting output from DOT is also presented below.

From this simple example, we see that it is extremely easy to use DOT for optimization. Of course, the number of design variables and constraints, as well as the routine for evaluating the functions, may be quite large. If you simply couple DOT with your analysis, we suggest limiting the number of design variables to the range of 50-100, although much larger problems have been solved. The number of constraints may be quite large (especially with METHOD=1 for constrained problems). Problems with over a million constraints have been solved with DOT.

BOX DESIGN FORTRAN PROGRAM

```
C     SAMPLE PROGRAM. BOX DESIGN.
      DIMENSION X(5),XL(5),XU(5),G(5),
     *WK(800),IWK(200),RPRM(20),IPRM(20)
C     DEFINE NRWK, NRIWK.
      NRWK=800
      NRIWK=200
C     ZERO RPRM AND IPRM.
      DO 10 I=1,20
         RPRM(I)=0.0
10       IPRM(I)=0
C     DEFINE METHOD,NDV,NCON.
C     SEQUENTIAL QUADRATIC PROGRAMMING METHOD.
      METHOD=3
C     THREE DESIGN VARIABLES.
      NDV=3
C     ONE CONSTRAINT
      NCON=1
C     DEFINE BOUNDS AND INITIAL DESIGN.
      DO 20 I=1,NDV
C     INITIAL VALUES.
         X(I)=1.0
C     LOWER BOUNDS.
         XL(I)=0.0
C     UPPER BOUNDS
20       XU(I)=100.
C     DEFINE IPRINT, MINMAX, INFO.
C     PRINT CONTROL.
      IPRINT=1
C     MINIMIZE
      MINMAX=-1
C     INITIALIZE INFO TO ZERO.
      INFO=0
C     OPTIMIZE.
100   CALL DOT (INFO,METHOD,IPRINT,NDV,NCON,X,XL,XU,
     *OBJ,MINMAX,G,RPRM,IPRM,WK,NRWK,IWK,NRIWK)
C     FINISHED?
      IF(INFO.EQ.0)STOP
C     EVALUATE OBJECTIVE AND CONSTRAINT.
      CALL EVAL(OBJ,X,G)
C     GO CONTINUE WITH OPTIMIZATION.
      GO TO 100
      END
      SUBROUTINE EVAL (OBJ,X,G)
C        SUBROUTINE TO EVALUATE THE OBJECTIVE FUNCTION AND CONSTRAINTS
C     FOR THE BOX DESIGN PROBLEM.
      DIMENSION X(*),G(*)
      OBJ=2.0*X(2)*X(1)+2.0*X(3)*X(1)+4.0*X(2)*X(3)
      G(1)=1.0-0.5*X(1)*X(2)*X(3)
      RETURN
      END
```

DOT OUTPUT

```
DDDDD         OOOOO        TTTTTTT
D    D       O     O          T
D    D  ==  O  *  O  ==      T
D    D       O     O          T
DDDDD         OOOOO           T

   DESIGN OPTIMIZATION TOOLS


     (C) COPYRIGHT, 1999

       VANDERPLAATS R&D

ALL RIGHTS RESERVED, WORLDWIDE

        VERSION 5.00
```

```
CONTROL PARAMETERS

OPTIMIZATION METHOD,                METHOD =       3
NUMBER OF DECISION VARIABLES,          NDV =       3
NUMBER OF CONSTRAINTS,                NCON =       1
PRINT CONTROL PARAMETER,             IPRINT =       1
GRADIENT PARAMETER,                   IGRAD =       0
  FORWARD DIFFERENCE GRADIENTS ARE CALCULATED BY DOT
THE OBJECTIVE FUNCTION WILL BE MINIMIZED


-- INITIAL VARIABLES AND BOUNDS

LOWER BOUNDS ON THE DECISION VARIABLES (XL-VECTOR)
     1)   0.00000E+00   0.00000E+00   0.00000E+00

DECISION VARIABLES (X-VECTOR)
     1)   1.00000E+00   1.00000E+00   1.00000E+00

UPPER BOUNDS ON THE DECISION VARIABLES (XU-VECTOR)
     1)   1.00000E+02   1.00000E+02   1.00000E+02


-- INITIAL FUNCTION VALUES
```

```
OBJ =   8.0000

CONSTRAINT VALUES (G-VECTOR)
     1)   5.00000E-01




-- OPTIMIZATION IS COMPLETE

NUMBER OF CONSTRAINED MINIMIZATIONS =    5

CONSTRAINT TOLERANCE, CT =-3.00000E-02

THERE ARE       1 ACTIVE CONSTRAINTS AND       0 VIOLATED CONSTRAINTS
CONSTRAINT NUMBERS
         1

THERE ARE       0 ACTIVE SIDE CONSTRAINTS

TERMINATION CRITERIA

RELATIVE CONVERGENCE CRITERION WAS MET FOR  2 CONSECUTIVE ITERATIONS

MAXIMUM S-VECTOR COMPONENT = 0.00000E+00 IS LESS THAN 1.00000E-04


-- OPTIMIZATION RESULTS


OBJECTIVE, F(X) =    1.20020E+01


DECISION VARIABLES, X

   ID         XL             X            XU
    1     0.00000E+00   1.99150E+00   1.00000E+02
    2     0.00000E+00   1.00225E+00   1.00000E+02
    3     0.00000E+00   1.00225E+00   1.00000E+02


CONSTRAINTS, G(X)

     1)  -2.37162E-04

FUNCTION CALLS =      29
```

1.7 What DOT Does

The DOT program uses numerical search methods to seek a minimum or maximum value of one function subject to limits on others. Numerical optimization methods such as this are formally known as Mathematical Programming Techniques. The functions involved must be calculated as functions of the "design variables." If you wish to maximize a function, DOT internally does this by minimizing the negative of that function, but this will not be apparent from the printed output.

The user must specify an initial set of design variables (also called decision variables), and must provide the necessary code(s) to evaluate the objective and constraint functions each time the design variables are changed by DOT. Section 1.8 gives the general form of the optimization problem and Chapters 2 and 3 provide the detailed information needed to use DOT to solve a particular optimization problem.

The search algorithms used by DOT are described in detail in Appendix E. However, the basic concept is to solve the problem in two steps. The first is to determine a "Search Direction" which defines how the design variables will be changed. The key idea is that all variables will be changed simultaneously in a fashion that will improve the design. The second part is to determine how far to move in this direction, and this is called a "One-Dimensional Search." This process of finding a search direction and then searching is called an "iteration" and is repeated until it converges to the optimum. This is the basic approach used by the Modified Method of Feasible Directions contained in DOT for constrained minimization. If no constraints are imposed, the minimization is called Unconstrained and DOT uses the Broydon-Fletcher-Goldfarb-Shanno (BFGS) algorithm if METHOD=1 and the Fletcher-Reeves (F.R.) algorithm if METHOD=2. These algorithms also proceeds in the two step process of finding a search direction and performing a one-dimensional search.

When using the Sequential Linear Programming (SLP) or the Sequential Quadratic Programming (SQP) method contained in DOT, the process is modified somewhat, but the basic concept of moving from one design to an improved design is the same.

Appendix E provides a more descriptive outline of the optimization process, as well as a detailed discussion of the algorithms contained in DOT. Also, Chapter 5 provides a list of references for further study.

1.8 The General Optimization Problem

The optimization problem is formally stated as follows [1]:

Minimize or Maximize

$$F(\mathbf{X}) \qquad \text{Objective Function} \qquad (1\text{-}11)$$

Subject to;

$$g_j(\mathbf{X}) \le 0 \qquad j = 1,\text{NCON} \qquad \text{Inequality Constraints} \qquad (1\text{-}12)$$

$$X_i^L \le X_i \le X_i^U \qquad i = 1,\text{NDV} \qquad \text{Side Constraints} \qquad (1\text{-}13)$$

NDV is the number of design (decision) variables. DOT is designed to be a robust numerical optimizer for problems of (typically) up to 50 design variables. However, there is nothing magic about this number. Problems in excess of two thousand design variables have been solved. Just be aware that optimization problems with more than 20-30 design variables may be more difficult to solve, not to mention more time consuming.

It should be noted that structural optimization applications, which use DOT as a subroutine in conjunction with a structural analysis code, often involve hundreds of design variables and thousands of nonlinear inequality constraints. Special methods exist for solving large structural optimization problems [2]. The GENESIS program combines the most advanced methods for structural optimization for design of structures [15].

NCON is the number of constraints in a particular problem. The number of constraints tends to get high for many problems. For example, consider a truss design problem (such as an electrical transmission tower) where each member has stress and buckling constraints, and displacement constraints are imposed at each joint. Also, the truss must support many independent load cases. For a large truss, there would be a great many constraints. Nevertheless, there is no maximum number of constraints to keep in mind.

X_i^L and X_i^U are called side constraints. These are lower and upper bounds on the design variables. A common use of lower bounds is to prevent the design variables from going below zero. For example, it would make no sense to design a mechanical part that has a negative thickness.

It is necessary to formulate optimization problems in this standard form. In Chapter 4, several examples from engineering and management are presented. These examples may be used as a guide to properly formulating optimization problems.

1.9 Equality Constraints

The formal problem statement of Section 1.8 includes only inequality constraints, which require a set of functions $g_j(\mathbf{X})$ to be less than or equal to zero. Suppose you want to specify that a function (or functions) must be equal to zero at the optimum. This is done by defining two separate inequality constraints for each function. One constraint requires the function to be less than or equal to zero and the other constraint requires the function to be greater than or equal to zero (in standard form, the negative of the function be less than or equal to zero). The only function value that satisfies both constraints is zero, which is just what an equality constraint requires. A simple example will demonstrate this.

Suppose you want the following relation to be true

$$-X_1^3 - X_2^2 + X_3 = 2.0$$

or, in the form that an optimizer understands,

$$-X_1^3 - X_2^2 + X_3 - 2.0 = 0.0$$

This requirement can be satisfied by imposing two equal and opposite inequality constraints as

$$g_1 = -X_1^3 - X_2^2 + X_3 - 2.0 \le 0.0$$

$$g_2 = -(-X_1^3 - X_2^2 + X_3 - 2.0) \le 0.0$$

That is;

$$g_2 = -g_1 \le 0.0$$

At the optimum, g_1 and g_2 should both equal zero within a small tolerance.

NOTE: In this case, an alternative would be to treat X_3 as a dependent variable and reduce the total number of independent design variables by one. Thus

$$X_3 = 2.0 + X_1^3 + X_2^2$$

Now, only X_1 and X_2 are the design variables. Whenever the optimizer requests function values, you would first calculate X_3 and then evaluate the functions in the usual way. This would reduce the number of design variables by one as well as eliminating both of the constraints. However, this is only possible when we have an explicit relationship such as this. In the general case, we just provide two equal and opposite constraints as we did above.

1.10 Special Notes

- Remember that optimization is iterative and usually nonlinear. You should try to formulate the problem so that the variables and functions are of the same general order of magnitude.
- Always normalize the constraints. For example,

  $Q \leq 20000.0$

  should be normalized as

  $Q/20000.0 - 1.0 \leq 0.0$

- DOT scales the design variables, **X** in an effort to improve the numerical conditioning of the optimization task. While there is no good theory for scaling, it usually works well. If DOT seems not to be working, try turning the scaling off by setting ISCAL = -1. [ISCAL is the internal parameter IPRM(2); see Chapter 3].

CHAPTER 2

DOT with Application Programs

- Introduction
- Methods Used by DOT
- Calling Statement
- Parameters in the Calling Statement
- Compiling and Linking
- A Simple Example

2.1 Introduction

Extremely powerful system optimization capabilities can be realized by interfacing DOT with application programs. With DOT, you can turn your system analysis or simulation programs into system performance optimization programs. You might have put a lot of effort into developing a program to analyze a system, and are quite satisfied with its performance, so that you can input various data and do trade-off studies and simulations. With DOT you can direct your computer to find values of parameters which will optimize the system performance while meeting any criteria that you specify. If you specify criteria that cannot be met, DOT will satisfy them as nearly as it can before terminating.

Interfacing DOT with your program is simple, as explained in this chapter. A part of the application program where all of the parameters and responses are available (usually in the main program) is modified to call DOT in the manner described below.

A simple main program that calls DOT is provided in Appendix A. All that needs to be provided are the parameters and functions. The parameters that must be provided are defined in the following two sections. An example is presented in section 2.6.

2.2 Methods Used by DOT

DOT solves both unconstrained and constrained optimization problems. For unconstrained problems, side constraints (lower and upper bounds on the design variables) are allowed, but there are no general constraints (NCON = 0).

The following table identifies the methods available. The choice of method is made with the METHOD parameter in the DOT calling statement.

Unconstrained Minimization (NCON = 0)	
METHOD	**DESCRIPTION**
0,1*	Broydon-Fletcher-Goldfarb-Shanno (BFGS) variable metric method. This method is usually considered best on theoretical grounds.
2	Fletcher-Reeves (F.R.) conjugate gradient method. This method uses very little computer memory and has been found to be reliable.
Constrained Minimization (NCON > 0)	
METHOD	**DESCRIPTION**
0,1*	Modified Method of Feasible Directions (MMFD). This method is reliable and uses the least computer memory.
2	Sequential Linear Programming (SLP). This method is often most efficient for general applications in terms of the number of function evaluations required, especially if there are as many active constraints as there are design variables at the optimum.
3	Sequential Quadratic Programming (SQP). This method is considered theoretically best if the optimization problem is "well conditioned." This method should always be tried for your problem. If it works successfully, it is almost always the most efficient.

* Default Method.

It is generally agreed that there is no "Best" method for all applications. The user is encouraged to try all available methods on several applications. The method that performs best will usually be good for many optimization problems in this class.

2.3 Calling Statement

DOT is invoked by the following FORTRAN calling statement in the user's program:

```
CALL DOT (INFO, METHOD, IPRINT, NDV, NCON, X, XL, XU, OBJ,
* MINMAX, G, RPRM, IPRM, WK, NRWK, IWK, NRIWK)
```

All information needed by DOT is passed via the parameter list. Also, when DOT requires the values of the objective function and constraints, it returns to the calling program instead of calling a user-supplied subroutine. This gives the user considerable flexibility in using DOT, allowing for restarting the optimization process or for calling DOT from the user's analysis subroutine(s) to perform sub-optimization tasks.

If you wish to call DOT froma C/C++ program, see Appendix A for a sample program.

2.4 Parameters in the Calling Statement

Table 2-1 lists the parameters in the calling statement to DOT. Where arrays are defined, the required dimension size is given as the array argument. These are minimum dimensions. The arrays can be dimensioned larger than this to allow for program expansion.

Table 2-1: Parameters in the DOT Argument List

PARAMETER	DEFINITION
INFO	Information parameter. Before calling DOT the first time, set INFO=0. When control returns from DOT to the calling program, INFO will normally have a value of 0 or 1. If INFO= 0, the optimization is complete (or terminated with an error message). If INFO= 1, the user must evaluate the objective, OBJ, and constraint functions, G(j), j=1,NCON, and call DOT again. A third possibility, INFO= 2, exists also. In this case, the user must provide gradient information. This is an advanced feature and is described in Chapter 3. NOTE: If IPRM(18)>0 on return from DOT, a Fatal Error has occurred (See Chapter 3).
METHOD	Optimization method to be used. METHOD = 0 or 1 means use the modified method of feasible directions. METHOD = 2 means use the sequential linear programming method. METHOD = 3 means use the sequential quadratic programming method. If the problem in unconstrained (NCON=0), the BFGS algorithm will be used if METHOD=0 or 1 and the Fletcher-Reeves algorithm will be used if METHOD=2.

IPRINT	Print control parameter. IPRINT = 0 no output. IPRINT = 1 internal parameters, initial information and results. IPRINT = 2 same plus objective function and X-vector at each iteration. IPRINT = 3 same plus G-vector and critical constraint numbers. IPRINT = 4 same plus gradients. IPRINT = 5 same plus search direction. IPRINT = 6 same plus set IPRM(11)=1 and IPRM(12)=1 IPRINT = 7 same except set IPRM(12)=2. NOTE: The IPRM Array contains additional print options. See Chapter 3.
NDV	Number of design (decision) variables contained in vector **X**.
NCON	Number of constraint values contained in array **G**. NCON=0 is allowed.
X(NDV)	Vector containing the design variables. On the first call to DOT, this is the user's best guess of the design. On the final return from DOT (INFO=0 is returned), the vector **X** contains the optimum design and vector **G** contains the corresponding constraint values.
XL(NDV)	Array containing lower bounds on the design variables, **X**. If no lower bounds are imposed on one or more of the design variables, the corresponding component(s) of **XL** must be set to a large negative number, say -1.0E+15. Be sure it's -1.0E**+**15 and not -1.0E**-**15 (+15, not -15 exponent).
XU(NDV)	Array containing upper bounds on the design variables, **X**. If no upper bounds are imposed on one or more of the design variables, the corresponding component(s) of **XU** must be set to a large positive number, say 1.0 E+15.
OBJ	Value of the objective function corresponding to the current values of the design variables contained in **X**. On the first call to DOT, OBJ need not be defined. DOT will return a value of INFO=1 to indicate that the user must evaluate OBJ and call DOT again. Subsequently, any time a value of INFO=1 is returned from DOT, the objective, OBJ, must be evaluated for the current design and DOT must be called again. OBJ has the same meaning as F(**X**) in the mathematical problem statement given in Chapter 1.

MINMAX	Integer parameter specifying whether the minimum (MINMAX=0,-1) or maximum (MINMAX=1) of the objective function is to be found.
G(NCON)	Array containing the NCON inequality constraint values corresponding to the current design contained in X. On the first call to DOT, the constraint values need not be defined. On return from DOT, if INFO=1, the constraints must be evaluated for the current **X** and DOT must be called again. If NCON=0, array **G** must be dimensioned to 1 or larger, but no constraint values need to be provided.
RPRM(20)	Array containing the real (floating point numbers) control parameters. Initialize the entire array to 0.0 to use all default values. If you use other values than the defaults, set the corresponding entries to the desired values. Chapter 3 describes how to change the value of these parameters.
IPRM(20)	Array containing the integer control parameters. As with the **RPRM** array, set the array to zero to use the default values, or set the proper entries to the desired values. Chapter 3 describes how to change the value of these parameters.
WK(NRWK)	User provided work array for real (floating point) variables. Array **WK** is used to store internal scalar variables and arrays used by DOT. If the user has not provided enough storage, DOT will print the appropriate message and terminate the optimization.
NRWK	Dimensioned size of work array **WK**. NRWK should be set quite large, starting at about 1000 for a small problem. If NRWK has been given too small a value, an error message will be printed and the optimization will be terminated.
IWK(NRIWK)	User provided work array for integer (fixed point) variables. Array **IWK** is used to store internal scalar variables and arrays used by DOT. If the user has not provided enough storage, DOT will print the appropriate message and terminate the optimization.
NRIWK	Dimensioned size of work array **IWK**. A good estimate is 300 for a small problem. Increase the size of NRIWK as the problem grows larger. If NRIWK is too small, an error message will be printed and the optimization will be terminated.

Note: The minimum required values of NRWK and NRIWK are defined in Appendix B. Those values are only minimums. The actual dimensions may be larger than this. DOT uses a large number of internal arrays. The arrays **WK** and **IWK** are used to store these and the internal data management allocates the appropriate locations to store the internal arrays.

A program called DTSTOR is provided with DOT. If you compile and run this program interactively, the minimum, desired and maximum required values of NRWK and NRIWK are calculated for you. See Appendix B for more information on this option.

2.5 Compiling and Linking

DOT is supplied as object code. When DOT is to be called by an application program, this DOT object code must be linked to a main program like the ones in Chapter 1, Chapter 4, or Appendix A.

You may use DOT in single precision (dot.a or dot*.lib files), or double precision (dot2.a or dot2*.lib files). The only difference between the two versions is that the double precision version contains IMPLICIT DOUBLE PRECISION (A-H,O-Z) statements at the beginning of each routine. Both single and double precision versions of example problems are provided.

Directions given here assume that you are using the single precision version. If you use the double precision version, the approach is the same.

Specific instructions for how to link DOT object code to the main program vary from system to system. Consult the manual that accompanies your compiler for detailed assistance.

An example of how some systems would compile and link a calling program MAIN.FOR to the DOT object code is as follows:

UNIX Systems:

```
f77 -o MAIN MAIN.FOR dot.a
```

An executable file called MAIN will be created. To run DOT, simply execute file MAIN. This assumes that “f77” is the command to invoke FORTRAN 77 compiler on your system

PC (windows 95/98/NT) with Digital FORTRAN 5.0:

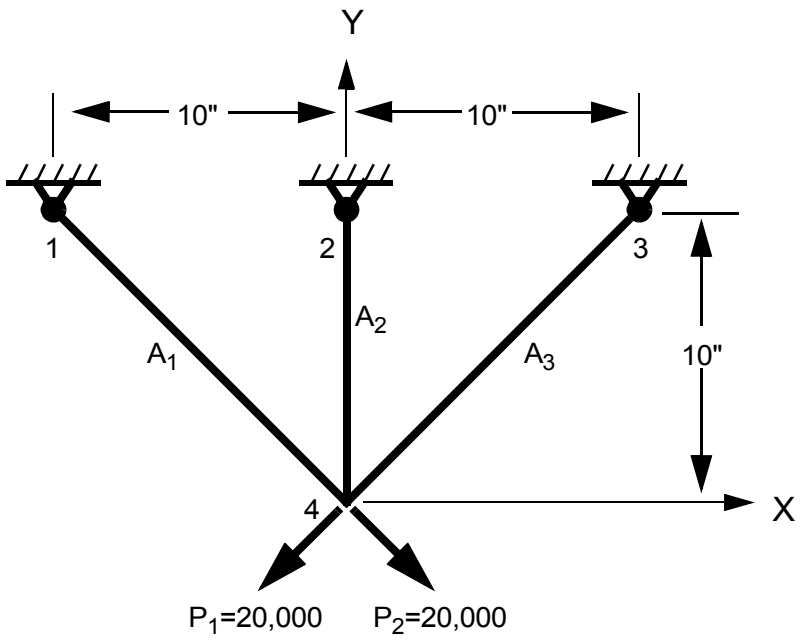
```
f77 /exe:MAIN.EXE MAIN.FOR dot_pc5.lib
```

An executable file called MAIN.EXE will be created. To run DOT, simply execute the file MAIN.EXE.

Sections 1.6 and 2.6, as well as Chapter 4 present example MAIN programs that the user can try immediately after receiving DOT. Appendix A provides a “generic” MAIN program which you may use as a “template” for your own applications.

2.6 A Simple Example

This is the optimization of the 3-bar truss shown below, which is a classical example in structural synthesis.



The objective is to minimize the total volume of the material of the members. The decision variables X_1 and X_2 correspond to the areas of member 1 (and 3) and member 2, respectively. The area of member 3 is "linked" to be the same as member 1 for symmetry. The constraints are tensile stress constraints in members 1 and 2 under load P_1. The loads, P_1 and P_2, are applied separately. This problem, in standard form for optimization, is given below. The original problem actually consists of 12 constraints, being the stress limit in each of the three members under each of the 2 loading conditions. The problem has been abbreviated here for clarity.

$$\text{Minimize} \quad OBJ = 2\sqrt{2}X_1 + X_2 \tag{2-1}$$

Subject to:

$$g_1 = \frac{2X_1 + \sqrt{2}X_2}{2X_1(X_1 + \sqrt{2}X_2)} - 1.0 \le 0.0 \tag{2-2}$$

$$g_2 = \frac{1.0}{X_1 + \sqrt{2}X_2} - 1.0 \le 0.0 \tag{2-3}$$

$$0.01 \le X_i \le 100.0 \qquad i = 1,2 \tag{2-4}$$

The program used to solve this problem and the results are presented below. Please note that, depending on your computer precision, the results may differ slightly from those given here. This is to be expected since optimization is a nonlinear iterative process. However, your results should be close to those given here. The following pages provide results for METHOD = 1, METHOD = 2 and METHOD = 3. Note that the final values of the design variables are significantly different, although the value of the objective function is very nearly the same for each case. This is because the design space is quite "flat" so that many nearby designs are equally acceptable.

LISTING 2-1: PROGRAM ILLUSTRATING HOW TO USE DOT WITH A SIMPLE CALLING PROGRAM

```
C
C     SAMPLE PROGRAM. THREE BAR TRUSS.
C
      DIMENSION X(2),XL(2),XU(2),G(2),
     *WK(800),IWK(200),RPRM(20),IPRM(20)
C     DEFINE NRWK, NRIWK
      NRWK=800
      NRIWK=200
C     ZERO RPRM AND IPRM
      DO 10 I=1,20
         RPRM(I)=0.0
10       IPRM(I)=0
C     DEFINE METHOD,NDV,NCON
      METHOD=1
      NDV=2
      NCON=2
C     DEFINE BOUNDS AND INTIAL DESIGN
      DO 20 I=1,NDV
         X(I)=1.0
         XL(I)=0.1
20       XU(I)=100.
C     DEFINE IPRINT, MINMAX, INFO
      IPRINT=3
      MINMAX=-1
      INFO=0
100   CALL DOT (INFO,METHOD,IPRINT,NDV,NCON,X,XL,XU,
     *OBJ,MINMAX,G,RPRM,IPRM,WK,NRWK,IWK,NRIWK)
      IF(INFO.EQ.0)STOP
      CALL EVAL(OBJ,X,G)
      GO TO 100
      END

      SUBROUTINE EVAL (OBJ,X,G)
C     SUBROUTINE TO EVALUATE THE OBJECTIVE FUNCTION AND CONSTRAINTS
C     FOR THE THREE BAR TRUSS PROBLEM.
      DIMENSION X(*),G(*)
      OBJ=2.0*SQRT(2.)*X(1)+X(2)
      G(1)=(2.*X(1)+SQRT(2.)*X(2))/(2.*X(1)*(X(1)+
     *SQRT(2.0)*X(2)))-1.
      G(2)=1./(2.*(X(1)+SQRT(2.)*X(2)))-1.
      RETURN
      END
```

DOT OUTPUT - METHOD=1

```
DDDDD        OOOOO       TTTTTTT
D    D      O     O         T
D    D  ==  O  *  O  ==     T
D    D      O     O         T
DDDDD        OOOOO          T

   DESIGN OPTIMIZATION TOOLS


     (C) COPYRIGHT, 1999

       VANDERPLAATS R&D

ALL RIGHTS RESERVED, WORLDWIDE

        VERSION 5.00
```

```
CONTROL PARAMETERS

OPTIMIZATION METHOD,                METHOD =       1
NUMBER OF DECISION VARIABLES,          NDV =       2
NUMBER OF CONSTRAINTS,                NCON =       2
PRINT CONTROL PARAMETER,             IPRINT =       3
GRADIENT PARAMETER,                   IGRAD =       0
  FORWARD DIFFERENCE GRADIENTS ARE CALCULATED BY DOT
THE OBJECTIVE FUNCTION WILL BE MINIMIZED


-- SCALAR PROGRAM PARAMETERS

REAL PARAMETERS
  1) CT     = -3.00000E-02            8) DX2    =  2.00000E-01
  2) CTMIN  =  3.00000E-03            9) FDCH   =  1.00000E-03
  3) DABOBJ =  3.82843E-04           10) FDCHM  =  1.00000E-04
  4) DELOBJ =  1.00000E-03           11) RMVLMZ =  4.00000E-01
  5) DOBJ1  =  1.00000E-01           12) DABSTR =  3.82843E-04
  6) DOBJ2  =  7.65685E-01           13) DELSTR =  1.00000E-03
  7) DX1    =  1.00000E-02

INTEGER PARAMETERS
  1) IGRAD  =      0    6) NGMAX  =      2   11) IPRNT1 =      0
  2) ISCAL  =      2    7) IGMAX  =      0   12) IPRNT2 =      0
  3) ITMAX  =    100    8) JTMAX  =     50   13) JWRITE =      0
  4) ITRMOP =      2    9) ITRMST =      2   14) MAXINT =  2000000000
  5) IWRITE =      6   10) JPRINT =      0   15) NSTORE =     39
```

```
     STORAGE REQUIREMENTS
 ARRAY  DIMENSION   MINIMUM   DESIRED   MAXIMUM    USED
   WK       800       115       154       154       113
  IWK       200        79                           79


 -- INITIAL VARIABLES AND BOUNDS

 LOWER BOUNDS ON THE DECISION VARIABLES (XL-VECTOR)
      1)   1.00000E-01   1.00000E-01

 DECISION VARIABLES (X-VECTOR)
      1)   1.00000E+00   1.00000E+00

 UPPER BOUNDS ON THE DECISION VARIABLES (XU-VECTOR)
      1)   1.00000E+02   1.00000E+02


 -- INITIAL FUNCTION VALUES

 OBJ =   3.8284

 CONSTRAINT VALUES (G-VECTOR)
      1)  -2.92893E-01  -7.92893E-01


 -- BEGIN CONSTRAINED OPTIMIZATION: MFD METHOD


 -- ITERATION NUMBER    1

 THERE ARE       0 ACTIVE CONSTRAINTS AND       0 VIOLATED CONSTRAINTS

 THERE ARE       0 ACTIVE SIDE CONSTRAINTS

 OBJECTIVE = 2.79967E+00

 DECISION VARIABLES (X-VECTOR)
      1)   6.76695E-01   8.85682E-01

 CONSTRAINT VALUES (G-VECTOR)
      1)  -1.94558E-03  -7.40831E-01
```

```
-- ITERATION NUMBER    2

THERE ARE       1 ACTIVE CONSTRAINTS AND       0 VIOLATED CONSTRAINTS
CONSTRAINT NUMBERS
         1

THERE ARE       0 ACTIVE SIDE CONSTRAINTS

OBJECTIVE = 2.63268E+00

DECISION VARIABLES (X-VECTOR)
     1)   7.99436E-01   3.71530E-01

CONSTRAINT VALUES (G-VECTOR)
     1)   2.83932E-03  -6.22602E-01


-- ITERATION NUMBER    3

THERE ARE       1 ACTIVE CONSTRAINTS AND       0 VIOLATED CONSTRAINTS
CONSTRAINT NUMBERS
         1

THERE ARE       0 ACTIVE SIDE CONSTRAINTS

OBJECTIVE = 2.63268E+00

DECISION VARIABLES (X-VECTOR)
     1)   7.99436E-01   3.71530E-01

CONSTRAINT VALUES (G-VECTOR)
     1)   2.83932E-03  -6.22602E-01


-- ITERATION NUMBER    4

THERE ARE       1 ACTIVE CONSTRAINTS AND       0 VIOLATED CONSTRAINTS
CONSTRAINT NUMBERS
         1

THERE ARE       0 ACTIVE SIDE CONSTRAINTS

OBJECTIVE = 2.63268E+00

DECISION VARIABLES (X-VECTOR)
     1)   7.99436E-01   3.71530E-01

CONSTRAINT VALUES (G-VECTOR)
     1)   2.83932E-03  -6.22602E-01
```

```
 -- OPTIMIZATION IS COMPLETE

 NUMBER OF ITERATIONS =    4

 CONSTRAINT TOLERANCE, CT = -3.00000E-03

 THERE ARE       1 ACTIVE CONSTRAINTS AND       0 VIOLATED CONSTRAINTS
 CONSTRAINT NUMBERS
          1

 THERE ARE       0 ACTIVE SIDE CONSTRAINTS

 TERMINATION CRITERIA

 RELATIVE CONVERGENCE CRITERION WAS MET FOR  2 CONSECUTIVE ITERATIONS

 ABSOLUTE CONVERGENCE CRITERION WAS MET FOR  2 CONSECUTIVE ITERATIONS


 -- OPTIMIZATION RESULTS


 OBJECTIVE, F(X) =    2.63268E+00


 DECISION VARIABLES, X

    ID          XL             X            XU
     1     1.00000E-01   7.99436E-01   1.00000E+02
     2     1.00000E-01   3.71530E-01   1.00000E+02


 CONSTRAINTS, G(X)

      1)   2.83932E-03  -6.22602E-01

 FUNCTION CALLS =      29
```

DOT OUTPUT - METHOD=2

```
DDDDD        OOOOO       TTTTTTT
D    D      O     O         T
D    D  ==  O  *  O  ==     T
D    D      O     O         T
DDDDD        OOOOO          T

   DESIGN OPTIMIZATION TOOLS


      (C) COPYRIGHT, 1999

        VANDERPLAATS R&D

ALL RIGHTS RESERVED, WORLDWIDE

        VERSION 5.00
```

```
CONTROL PARAMETERS

OPTIMIZATION METHOD,                METHOD =       2
NUMBER OF DECISION VARIABLES,          NDV =       2
NUMBER OF CONSTRAINTS,                NCON =       2
PRINT CONTROL PARAMETER,             IPRINT =       3
GRADIENT PARAMETER,                   IGRAD =       0
  FORWARD DIFFERENCE GRADIENTS ARE CALCULATED BY DOT
THE OBJECTIVE FUNCTION WILL BE MINIMIZED


-- SCALAR PROGRAM PARAMETERS

REAL PARAMETERS
  1) CT     = -3.00000E-02            8) DX2    =  2.00000E-01
  2) CTMIN  =  3.00000E-03            9) FDCH   =  1.00000E-03
  3) DABOBJ =  3.82843E-04           10) FDCHM  =  1.00000E-04
  4) DELOBJ =  1.00000E-03           11) RMVLMZ =  4.00000E-01
  5) DOBJ1  =  1.00000E-01           12) DABSTR =  3.82843E-04
  6) DOBJ2  =  7.65685E-01           13) DELSTR =  1.00000E-03
  7) DX1    =  1.00000E-02

INTEGER PARAMETERS
  1) IGRAD  =      0    6) NGMAX  =      2   11) IPRNT1 =      0
  2) ISCAL  =      2    7) IGMAX  =      0   12) IPRNT2 =      0
  3) ITMAX  =    100    8) JTMAX  =     50   13) JWRITE =      0
  4) ITRMOP =      2    9) ITRMST =      2   14) MAXINT =  2000000000
  5) IWRITE =      6   10) JPRINT =      0   15) NSTORE =     67
```

```
     STORAGE REQUIREMENTS
 ARRAY  DIMENSION   MINIMUM   DESIRED   MAXIMUM    USED
   WK       800       127       194       194       155
  IWK       200        83                           83


 -- INITIAL VARIABLES AND BOUNDS

 LOWER BOUNDS ON THE DECISION VARIABLES (XL-VECTOR)
      1)   1.00000E-01   1.00000E-01

 DECISION VARIABLES (X-VECTOR)
      1)   1.00000E+00   1.00000E+00

 UPPER BOUNDS ON THE DECISION VARIABLES (XU-VECTOR)
      1)   1.00000E+02   1.00000E+02


 -- INITIAL FUNCTION VALUES

 OBJ =   3.8284

 CONSTRAINT VALUES (G-VECTOR)
      1)  -2.92893E-01  -7.92893E-01


 -- BEGIN CONSTRAINED OPTIMIZATION: SLP METHOD

 -- BEGIN SLP ITERATION    1     RELATIVE MOVE LIMIT = 0.40000

 OBJ = 2.29706E+00

 DECISION VARIABLES (X-VECTOR)
      1)   6.00000E-01   6.00000E-01

 CONSTRAINT VALUES (G-VECTOR)
      1)   1.78511E-01  -6.54822E-01

 GMAX=  1.7851E-01

 -- BEGIN SLP ITERATION    2     RELATIVE MOVE LIMIT = 0.40000

 OBJ = 2.40586E+00

 DECISION VARIABLES (X-VECTOR)
      1)   8.15244E-01   1.00000E-01

 CONSTRAINT VALUES (G-VECTOR)
      1)   1.35962E-01  -4.77351E-01

 GMAX=  1.3596E-01
```

```
-- BEGIN SLP ITERATION    3     RELATIVE MOVE LIMIT = 0.40000

OBJ = 2.43400E+00

DECISION VARIABLES (X-VECTOR)
     1)   6.83774E-01   5.00000E-01

CONSTRAINT VALUES (G-VECTOR)
     1)   9.07204E-02  -6.40516E-01

GMAX=  9.0720E-02

-- BEGIN SLP ITERATION    4     RELATIVE MOVE LIMIT = 0.40000

OBJ = 2.58966E+00

DECISION VARIABLES (X-VECTOR)
     1)   8.09516E-01   3.00000E-01

CONSTRAINT VALUES (G-VECTOR)
     1)   2.29113E-02  -5.94742E-01

GMAX=  2.2911E-02

-- BEGIN SLP ITERATION    5     RELATIVE MOVE LIMIT = 0.40000

OBJ = 2.62851E+00

DECISION VARIABLES (X-VECTOR)
     1)   7.76112E-01   4.33333E-01

CONSTRAINT VALUES (G-VECTOR)
     1)   4.22476E-03  -6.40013E-01

GMAX=  4.2248E-03

-- BEGIN SLP ITERATION    6     RELATIVE MOVE LIMIT = 0.40000

OBJ = 2.64110E+00

DECISION VARIABLES (X-VECTOR)
     1)   8.27704E-01   3.00000E-01

CONSTRAINT VALUES (G-VECTOR)
     1)   3.45199E-03  -6.00629E-01

GMAX=  3.4520E-03
```

```
 -- BEGIN SLP ITERATION    7     RELATIVE MOVE LIMIT = 0.40000

 OBJ = 2.62872E+00

 DECISION VARIABLES (X-VECTOR)
      1)   7.76188E-01   4.33333E-01

 CONSTRAINT VALUES (G-VECTOR)
      1)   4.14194E-03  -6.40032E-01

 GMAX=  4.1419E-03

 CONSTRAINT VIOLATION IS INCREASED.  MOVE HALF WAY

 OBJ = 2.63491E+00

 DECISION VARIABLES (X-VECTOR)
      1)   8.01946E-01   3.66667E-01

 CONSTRAINT VALUES (G-VECTOR)
      1)   2.13071E-03  -6.21353E-01

 GMAX=  2.1307E-03

 -- BEGIN SLP ITERATION    8     RELATIVE MOVE LIMIT = 0.20000

 OBJ = 2.64022E+00

 DECISION VARIABLES (X-VECTOR)
      1)   7.80253E-01   4.33333E-01

 CONSTRAINT VALUES (G-VECTOR)
      1)  -2.64937E-04  -6.41083E-01

 GMAX= -2.6494E-04
```

```
-- BEGIN SLP ITERATION    9     RELATIVE MOVE LIMIT = 0.20000

OBJ = 2.63083E+00

DECISION VARIABLES (X-VECTOR)
     1)   8.47645E-01   2.33333E-01

CONSTRAINT VALUES (G-VECTOR)
     1)   1.44521E-02  -5.75418E-01

GMAX=  1.4452E-02

CONSTRAINT VIOLATION IS INCREASED.  MOVE HALF WAY

OBJ = 2.63553E+00

DECISION VARIABLES (X-VECTOR)
     1)   8.13949E-01   3.33333E-01

CONSTRAINT VALUES (G-VECTOR)
     1)   3.28728E-03  -6.11002E-01

GMAX=  3.2873E-03

CONSTRAINT VIOLATION IS INCREASED.  MOVE HALF WAY

OBJ = 2.63788E+00

DECISION VARIABLES (X-VECTOR)
     1)   7.97101E-01   3.83333E-01

CONSTRAINT VALUES (G-VECTOR)
     1)   6.25912E-04  -6.26647E-01

GMAX=  6.2591E-04

-- BEGIN SLP ITERATION   10     RELATIVE MOVE LIMIT = 0.05000

OBJ = 2.63938E+00

DECISION VARIABLES (X-VECTOR)
     1)   7.88795E-01   4.08333E-01

CONSTRAINT VALUES (G-VECTOR)
     1)  -1.60534E-04  -6.34039E-01

GMAX= -1.6053E-04
```

```
 -- OPTIMIZATION IS COMPLETE

 NUMBER OF CONSTRAINED MINIMIZATIONS =   10

 CONSTRAINT TOLERANCE, CT =-3.00000E-02

 THERE ARE       1 ACTIVE CONSTRAINTS AND       0 VIOLATED CONSTRAINTS
 CONSTRAINT NUMBERS
          1

 THERE ARE       0 ACTIVE SIDE CONSTRAINTS

 TERMINATION CRITERIA

 RELATIVE CONVERGENCE CRITERION WAS MET FOR  2 CONSECUTIVE ITERATIONS


 -- OPTIMIZATION RESULTS


 OBJECTIVE, F(X) =    2.63938E+00


 DECISION VARIABLES, X

    ID          XL             X            XU
     1     1.00000E-01   7.88795E-01   1.00000E+02
     2     1.00000E-01   4.08333E-01   1.00000E+02


 CONSTRAINTS, G(X)

      1)  -1.60534E-04  -6.34039E-01

 FUNCTION CALLS =      34
```

DOT OUTPUT - METHOD=3

```
DDDDD        OOOOO       TTTTTTT
D    D      O     O         T
D    D  ==  O  *  O  ==     T
D    D      O     O         T
DDDDD        OOOOO          T

   DESIGN OPTIMIZATION TOOLS


     (C) COPYRIGHT, 1999

       VANDERPLAATS R&D

ALL RIGHTS RESERVED, WORLDWIDE

       VERSION 5.00
```

```
CONTROL PARAMETERS

OPTIMIZATION METHOD,                METHOD =       3
NUMBER OF DECISION VARIABLES,          NDV =       2
NUMBER OF CONSTRAINTS,                NCON =       2
PRINT CONTROL PARAMETER,             IPRINT =       3
GRADIENT PARAMETER,                   IGRAD =       0
  FORWARD DIFFERENCE GRADIENTS ARE CALCULATED BY DOT
THE OBJECTIVE FUNCTION WILL BE MINIMIZED


-- SCALAR PROGRAM PARAMETERS

REAL PARAMETERS
  1) CT     = -3.00000E-02           8) DX2    =  2.00000E-01
  2) CTMIN  =  3.00000E-03           9) FDCH   =  1.00000E-03
  3) DABOBJ =  3.82843E-04          10) FDCHM  =  1.00000E-04
  4) DELOBJ =  1.00000E-03          11) RMVLMZ =  4.00000E-01
  5) DOBJ1  =  1.00000E-01          12) DABSTR =  3.82843E-04
  6) DOBJ2  =  7.65685E-01          13) DELSTR =  1.00000E-03
  7) DX1    =  1.00000E-02

INTEGER PARAMETERS
  1) IGRAD  =      0    6) NGMAX  =      2   11) IPRNT1 =      0
  2) ISCAL  =   1000    7) IGMAX  =      0   12) IPRNT2 =      0
  3) ITMAX  =    100    8) JTMAX  =     50   13) JWRITE =      0
  4) ITRMOP =      2    9) ITRMST =      2   14) MAXINT =  2000000000
  5) IWRITE =      6   10) JPRINT =      0   15) NSTORE =     67
```

```
     STORAGE REQUIREMENTS
 ARRAY  DIMENSION   MINIMUM   DESIRED   MAXIMUM    USED
   WK       800       136       203       203       164
  IWK       200        81                           81


 -- INITIAL VARIABLES AND BOUNDS

 LOWER BOUNDS ON THE DECISION VARIABLES (XL-VECTOR)
      1)   1.00000E-01   1.00000E-01

 DECISION VARIABLES (X-VECTOR)
      1)   1.00000E+00   1.00000E+00

 UPPER BOUNDS ON THE DECISION VARIABLES (XU-VECTOR)
      1)   1.00000E+02   1.00000E+02


 -- INITIAL FUNCTION VALUES

 OBJ =   3.8284

 CONSTRAINT VALUES (G-VECTOR)
      1)  -2.92893E-01  -7.92893E-01


 -- BEGIN CONSTRAINED OPTIMIZATION: SQP METHOD


 -- BEGIN SQP ITERATION    1

 OBJ = 2.68203E+00

 DECISION VARIABLES (X-VECTOR)
      1)   7.18853E-01   6.48809E-01

 CONSTRAINT VALUES (G-VECTOR)
      1)   1.09941E-03  -6.94453E-01

 GMAX =  1.0994E-03


 -- BEGIN SQP ITERATION    2

 OBJ = 2.61241E+00

 DECISION VARIABLES (X-VECTOR)
      1)   7.66087E-01   4.45586E-01

 CONSTRAINT VALUES (G-VECTOR)
      1)   1.07722E-02  -6.41896E-01

 GMAX =  1.0772E-02
```

```
-- BEGIN SQP ITERATION    3

OBJ = 2.63894E+00

DECISION VARIABLES (X-VECTOR)
     1)   7.85773E-01   4.16438E-01

CONSTRAINT VALUES (G-VECTOR)
     1)   3.07062E-05  -6.36285E-01

GMAX =  3.0706E-05


-- BEGIN SQP ITERATION    4

OBJ = 2.63883E+00

DECISION VARIABLES (X-VECTOR)
     1)   7.88525E-01   4.08539E-01

CONSTRAINT VALUES (G-VECTOR)
     1)   5.03988E-05  -6.34045E-01

GMAX =  5.0399E-05


-- BEGIN SQP ITERATION    5

Q.P. SUB-PROBLEM GAVE NULL SEARCH DIRECTION.  CONVERGENCE ASSUMED.



-- OPTIMIZATION IS COMPLETE

NUMBER OF CONSTRAINED MINIMIZATIONS =    5

CONSTRAINT TOLERANCE, CT =-3.00000E-02

THERE ARE       1 ACTIVE CONSTRAINTS AND       0 VIOLATED CONSTRAINTS
CONSTRAINT NUMBERS
         1

THERE ARE       0 ACTIVE SIDE CONSTRAINTS

TERMINATION CRITERIA

MAXIMUM S-VECTOR COMPONENT = 0.00000E+00 IS LESS THAN 1.00000E-04
```

```
-- OPTIMIZATION RESULTS


OBJECTIVE, F(X) =    2.63883E+00


DECISION VARIABLES, X

   ID          XL             X            XU
    1     1.00000E-01   7.88525E-01   1.00000E+02
    2     1.00000E-01   4.08539E-01   1.00000E+02


CONSTRAINTS, G(X)

     1)   5.03988E-05  -6.34045E-01

FUNCTION CALLS =      22
```

CHAPTER 3

Advanced Use of DOT

- **Introduction**
- **Over-Riding DOT Default Parameters**
- **Directly Supplying Gradients**
- **Interrupting and Restarting DOT**
- **Output to a Postprocessing Data File**

3.1 Introduction

This chapter discusses advanced uses of DOT. These include over-riding the internal default parameters, directly supplying the function gradients, interrupting and restarting the optimization, and output to a graphics file. These features can be invoked by simple modifications to the user-supplied main program.

3.2 Over-Riding DOT Default Parameters

DOT contains a variety of internal parameters that effect the efficiency and reliability of the optimization process. Each of these is assigned a "default" value to be used unless the user explicitly changes it. Occasionally, the user may wish to over-ride some of the internal parameters of DOT. The default parameters include constraint tolerance, and the maximum finite difference step for gradient calculations, among others. The default parameters can be changed by simply setting the proper element of the RPRM or IPRM array to the desired value before calling DOT the first time. A sample program is included in this section. The figure below is a block diagram of the program to over-ride DOT default parameters. Tables 3-1 through 3-4 identify and define the internal parameters. Additional explanation of the parameters is given in Appendix D. Listing 3-1 shows an example FORTRAN file where the constraint tolerances CT and CTMIN, as well as the scaling parameter, ISCAL, are modified.

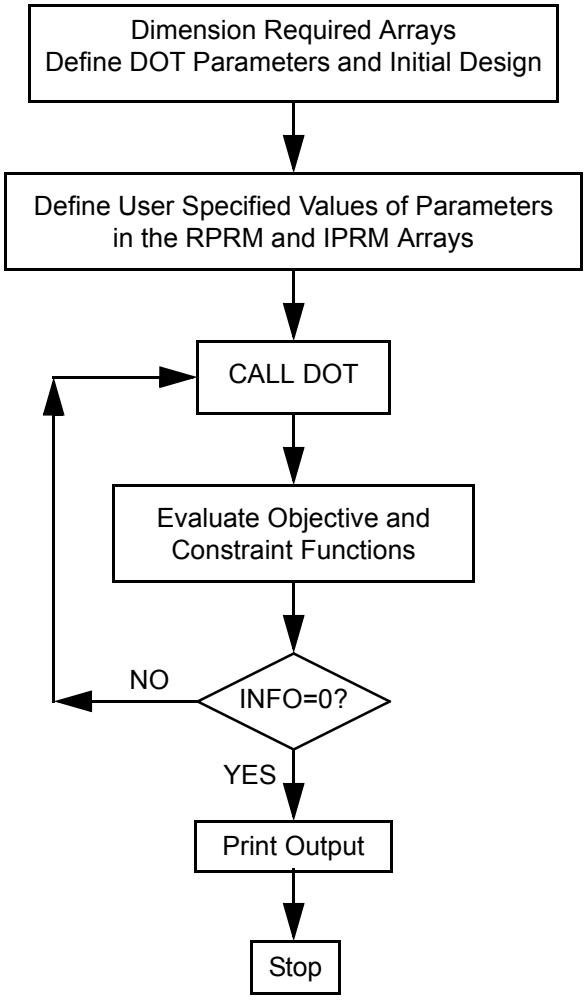


Table 3-1: Scalar Parameters Stored in the RPRM Array

LOCATION	NAME	DEFAULT VALUE
RPRM(1)	CT	-0.03
RPRM(2)	CTMIN	0.003
RPRM(3)	DABOBJ	MAX[0.0001*ABS(F0),1.0E-20]
RPRM(4)	DELOBJ	0.001
RPRM(5)	DOBJ1	0.1
RPRM(6)	DOBJ2	0.2*ABS(F0)
RPRM(7)	DX1	0.01
RPRM(8)	DX2	0.2*ABS[X(I)]
RPRM(9)	FDCH	0.001
RPRM(10)	FDCHM	0.0001
RPRM(11)	RMVLMZ	0.4
RPRM(12)	DABSTR	MAX[0.0001*ABS(F0),1.0E-20]
RPRM(13)	DELSTR	0.001
RPRM(14)-RPRM(15)	RESERVED FOR INTERNAL USE	
RPRM(16)	XSAV	Internally defined. XSAV = X(1)
RPRM(17)-RPRM(20)	RESERVED FOR INTERNAL USE	

NOTE: F0 = The value of the objective function at the start of optimization (for the initial values of **X**).

Table 3-2: Definitions of Parameters Contained in the RPRM Array

LOCATION	PARAMETER	DEFINITION
1	CT	A constraint is active if its numerical value is more positive than CT. CT is a small negative number.
2	CTMIN	A constraint is violated if its numerical value is more positive than CTMIN.
3	DABOBJ	Maximum absolute change in the objective between ITRMOP consecutive iterations to indicate convergence in optimization.
4	DELOBJ	Maximum relative change in the objective between ITRMOP consecutive iterations to indicate convergence in optimization.
5	DOBJ1	Relative change in the objective function attempted on the first optimization iteration. Used to estimate initial move in the one-dimensional search. Updated as the optimization progresses.
6	DOBJ2	Absolute change in the objective function attempted on the first optimization iteration.
7	DX1	Maximum relative change in a design variable attempted on the first optimization iteration. Used to estimate the initial move in the one-dimensional search. Updated as the optimization progresses.
8	DX2	Maximum absolute change in a design variable attempted on the first optimization iteration. Used to estimate the initial move in the one-dimensional search. Updated as the optimization progresses.
9	FDCH	Relative finite difference step when calculating gradients.
10	FDCHM	Minimum absolute value of the finite difference step when calculating gradients. This prevents too small a step when X(I) is near zero.
11	RMVLMZ	Maximum relative change in design variables during the first approximate subproblem in the Sequential Linear Programming Method. That is, each design variable is initially allowed to change by $\pm$40%. This move limit is reduced as the optimization progresses.
12	DABSTR	Maximum absolute change in the objective between ITRMST consecutive iterations of the Sequential Linear Programming and Sequential Quadratic Programming methods to indicate convergence to the optimum.
13	DELSTR	Maximum relative change in the objective between ITRMST consecutive iterations of the Sequential Linear Programming method to indicate convergence to the optimum.
16	XSAV	When NEWITR [(IPRM(19)] changes to indicate the beginning of a new iteration, XSAV stores X(1) during finite difference gradient calculations.

Table 3-3: Parameters Stored in the IPRM Array

LOCATION	NAME	DEFAULT VALUE
IPRM(1)	IGRAD	0
IPRM(2)	ISCAL	NDV
IPRM(3)	ITMAX	100
IPRM(4)	ITRMOP	2
IPRM(5)	IWRITE	6
IPRM(6)	NGMAX	NCON, but not more than 2*NDV
IPRM(7)	IGMAX	0
IPRM(8)	JTMAX	50
IPRM(9)	ITRMST	2
IPRM(10)	JPRINT	0
IPRM(11)	IPRNT1	0
IPRM(12)	IPRNT2	0
IPRM(13)	JWRITE	0
IPRM(14)	MAXINT	2000000000
IPRM(15)	NSTORE	Internally defined.
IPRM(16)-IPRM(17)	RESERVED FOR FUTURE USE	
IPRM(18)	IERROR	INTERNALLY DEFINED FATAL ERROR FLAG
IPRM(19)	NEWITR	INTERNALLY DEFINED ITERATION COUNTER
IPRM(20)	NGT	INTERNALLY DEFINED NUMBER OF NEEDED CONSTRAINT GRADIENTS

Table 3-4: Definitions of Parameters Contained in the IPRM Array

LOCATION	PARAMETER	DEFINITION
1	IGRAD	Specifies whether the gradients are calculated by DOT (IGRAD=-1 or 0) or by the user (IGRAD=1). If IGRAD=0, gradients are calculated by first forward finite difference. If IGRAD=-1, gradients are calculated by central finite difference. Note: If IGRAD=-1, the gradients are more accurate, but the number of function evaluations is almost doubled.
2	ISCAL	Design variables are rescaled every ISCAL iterations. Set ISCAL = -1 to turn off scaling.
3	ITMAX	Maximum number of iterations allowed at the optimization level.
4	ITRMOP	The number of consecutive iterations for which the absolute or relative convergence criteria must be met to indicate convergence at the optimizer level.
5	IWRITE	File number for printed output.
6	NGMAX	Number of retained constraints used for METHOD=2 or 3. Also, the maximum number of constraints retained for gradient calculations when METHOD=1.
7	IGMAX	If IGMAX=0, only gradients of active and violated constraints are calculated. If IGMAX>0, up to NGMAX gradients are calculated, including active, violated, and near active constraints.
8	JTMAX	Maximum number of iterations allowed for the Sequential Linear Programming and Sequential Quadratic Programming methods. This is the number of linearized subproblems solved.
9	ITRMST	The number of consecutive iterations for which the absolute or relative convergence criteria must be met to indicate convergence in the Sequential Linear Programming and Sequential Quadratic Programming methods.
10	JPRINT	Sequential Linear Programming and Sequential Quadratic Programming subproblem print. If JPRINT>0, IPRINT is turned on during approximate subproblem. This is for debugging only.
11	IPRNT1	If IPRNT1=1, print scaling factors for the X vector.
12	IPRNT2	If IPRNT2=1, print miscellaneous search information. If IPRNT2=2, turn on print during one-dimensional search process. This is for debugging only.
13	JWRITE	File number to write iteration history information to. This is useful for using postprocessing programs to plot the iteration process. This is only used if JWRITE>0.
14	MAXINT	Maximum integer number that can be defined.

15	NSTORE	Storage allocated for constraint gradients and solution of the direction finding sub-problem. NSTORE is internally calculated by DOT.
16-17	Not presently used	
18	IERROR	Fatal error parameter returned by DOT. Normally = 0. IERROR = 1 indicates WK or IWK dimension is too small. IERROR = 2 indicates some XL(I) is greater than XU(I). IERROR = 3 or 4 indicates that storage for constraint gradients is too small. IERROR = 5 indicates that a violated constraint has a zero gradient. Therefore, no feasible design can be found.
19	NEWITR	Normally = -1. Set = n at the start of a new iteration, where n is the number of the iteration just completed. At the beginning of optimization, n=0 will be returned to indicate that the initial analysis is being done. If METHOD=0,1, for constrained problems, or if METHOD=0,1,2 for unconstrained problems, this is after each one-dimensional search. If METHOD=2,3, for constrained problems, this is after each approximate optimization. If JWRITE>0, the optimization information will have just been written to that file. If you wish to stop after each iteration (or after a particular iteration) and then re-start later, NEWITR is a flag to do this. NEWITR is defined internally by DOT.
20	NGT	The number of constraint gradients needed. If the user supplies gradients to DOT, this will be needed. The constraint numbers for which gradients are needed are contained in position 1 through NGT of the IWK array. NGT is defined internally by DOT.

LISTING 3-1: OVER-RIDING DEFAULT PARAMETERS: THE 3-BAR TRUSS.

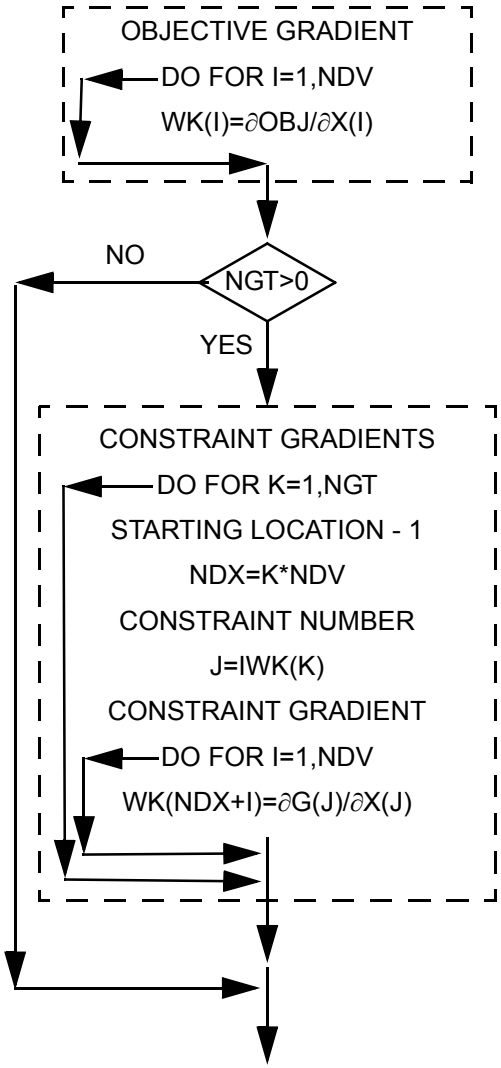
```
C     REQUIRED ARRAYS.  WK and IWK are dimensioned very large.
      DIMENSION X(2),XL(2),XU(2),G(2),WK(1000),
     *IWK(500),RPRM(20),IPRM(20)
C     DIMENSIONS OF WK AND IWK
      NRWK=1000
      NRIWK=500
C     ZERO RPRM AND IPRM
      DO 10 I=1,20
        RPRM(I)=0.0
10    IPRM(I)=0
C * * OVER-RIDE THE DEFAULT FOR CT AND CTMIN
C * * CT VALUE
      RPRM(1)=-0.1
C * * CTMIN VALUE
      RPRM(2)=0.002
C * * TURN OFF SCALING (ISCAL = -1)
      IPRM(2)=-1
C     DEFINE METHOD, NDV, NCON, IPRINT, MINMAX, X, XL, XU
      METHOD=1
      NDV=2
      NCON=2
      IPRINT=1
      MINMAX=-1
      X(1)=1.0
      X(2)=1.0
      XL(1)=0.1
      XL(2)=0.1
      XU(1)=1.0E+20
      XU(2)=1.0E+20
C     READY TO OPTIMIZE
      INFO=0
20    CALL DOT (INFO,METHOD,IPRINT,NDV,NCON,X,XL,XU,OBJ,
     *MINMAX,G,RPRM,IPRM,WK,NRWK,IWK,NRIWK)
C     EVALUATE OBJECTIVE AND CONSTRAINTS
      OBJ=2.*SQRT(2.)*X(1)+X(2)
      G(1)=(2.*X(1)+SQRT(2.)*X(2))/(2.*X(1)*(X(1)+
     *SQRT(2.)*X(2)))-1.
      G(2)=1./(X(1)+SQRT(2.)*X(2))-1.
      IF(INFO.GT.0)GO TO 20
C     INFO=0. OPTIMIZATION IS COMPLETE. PRINT RESULTS.
      WRITE(6,40)OBJ,X(1),X(2),G(1),G(2)
      STOP
40    FORMAT(//5X,'OPTIMUM',5X,'OBJ =',E12.5//5X,'
     *X(1) =',E12.5,5X,'X(2) =',E12.5/5X,
     *'G(1) =',E12.5,5X,'G(2) =',E12.5)
      END
```

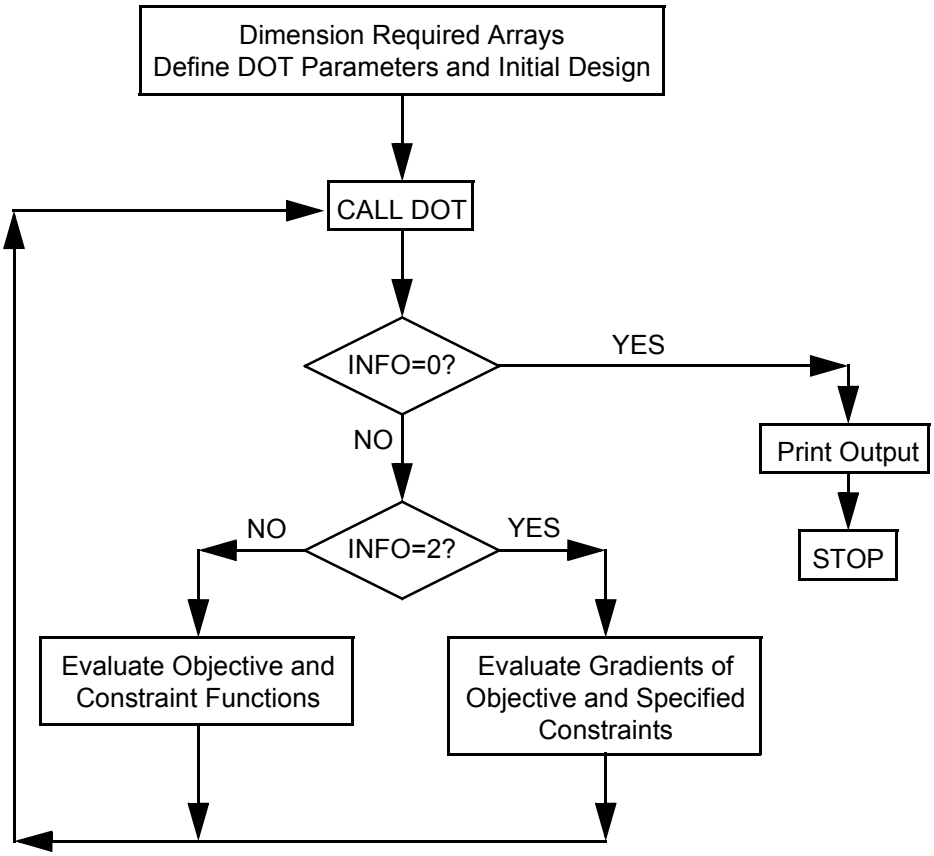
3.3 Directly Supplying Gradients

The default option of DOT is to use finite difference methods to calculate the gradients of the objective function and constraints. This approach can become time consuming for large problems. The user may wish to directly evaluate the gradients in the FORTRAN or C/C++ program and send them to DOT. This is done by setting IPRM(1) = 1 before calling DOT.

The gradients of the objective function are stored in the first NDV locations of the **WK** array. DOT requires gradients of the active and near active constraints only. The active and near active constraints are identified in the first NGT elements of the **IWK** array, where NGT is given in **IPRM**(20) and is the number of active and near active constraints. The gradients of these constraints are stored in the next NDV*NGT elements of the **WK** array (following the gradient of the objective function).

The general outline of the code to provide gradients is shown in following figures. The second figure is a block diagram of the standard calling program for DOT with user-supplied gradients. The logic described there is used in the block where gradients are calculated in the first figure. Listing 3-2 gives an example FORTRAN program for supplying gradients while optimizing the three-bar truss problem of Section 2.6.


OBJECTIVE GRADIENT
DO FOR I=1,NDV
WK(I)=∂OBJ/∂X(I)
NO
NGT>0
YES
CONSTRAINT GRADIENTS
DO FOR K=1,NGT
STARTING LOCATION - 1
NDX=K*NDV
CONSTRAINT NUMBER
J=IWK(K)
CONSTRAINT GRADIENT
DO FOR I=1,NDV
WK(NDX+I)=∂G(J)/∂X(J)


Dimension Required Arrays
Define DOT Parameters and Initial Design
CALL DOT
INFO=0?
YES
NO
Print Output
STOP
NO
INFO=2?
YES
Evaluate Objective and Constraint Functions
Evaluate Gradients of Objective and Specified Constraints

LISTING 3-2: THREE-BAR TRUSS. USER-SUPPLIED GRADIENTS.

```
C     USER-SUPPLIED GRADIENTS: THE THREE-BAR TRUSS.
C     REQUIRED ARRAYS.
C
      DIMENSION X(2),XL(2),XU(2),G(2),
     *WK(1000),IWK(500),RPRM(20),IPRM(20),AA(2,2),BB(2,2)
C
C     DIMENSIONS OF WK AND IWK
      NRWK=1000
      NRIWK=500
C     ZERO RPRM AND IPRM
      DO 10 I=1,20
        RPRM(I)=0.0
10    IPRM(I)=0
C     SPECIFY THAT GRADIENTS ARE TO BE PROVIDED
      IPRM(1)=1
C     DEFINE METHOD, NDV, NCON, IPRINT, MINMAX
      METHOD=1
      NDV=2
      NCON=2
      IPRINT=1
      MINMAX=-1
C     DEFINE X,XL,XU
      X(1)=1.0
      X(2)=1.0
      XL(1)=0.1
      XL(2)=0.1
      XU(1)=1.0E+20
      XU(2)=1.0E+20
C     READY TO OPTIMIZE
      INFO=0
20    CALL DOT (INFO,METHOD,IPRINT,NDV,NCON,X,XL,XU,OBJ,
     *MINMAX,G,RPRM,IPRM,WK,NRWK,IWK,NRIWK)
C     EXIT IF CONVERGENCE IS OBTAINED
      IF(INFO.EQ.0)GO TO 70
C     PROVIDE GRADIENTS IF DOT IS REQUESTING THEM
      IF(INFO.EQ.2)GO TO 30
      OBJ=2.*SQRT(2.)*X(1)+X(2)
      G(1)=(2.*X(1)+SQRT(2.)*X(2))/(2.*X(1)*(X(1)+
     *SQRT(2.)*X(2)))-1.
      G(2)=1./(X(1)+SQRT(2.)*X(2))-1.
      GO TO 20
C     ------------------------------------------------
30    CONTINUE
C     GRADIENT OF OBJECTIVE
      WK(1)=2.*SQRT(2.)
      WK(2)=1.0
      NGT=IPRM(20)
      IF(NGT.EQ.0)GO TO 20
```

```
C     CONSTRAINT GRADIENTS. USE ARRAY BB FOR TEMPORY
C     STORAGE
      D1=(X(1)+SQRT(2.)*X(2))**2
C     GRADIENT OF G(1)
      BB(1,1)=-(2.*X(1)*X(1)+2.*SQRT(2.)*X(1)*X(2)+
     *SQRT(2.)*X(2)*X(2))/(2.*X(1)*X(1)*D1)
      BB(2,1)=-1./(SQRT(2.)*D1)
C     GRADIENT OF G(2)
      BB(1,2)=-0.5/D1
      BB(2,2)=SQRT(2.)*BB(1,2)
C     STORE APPROPRIATE GRADIENTS IN ARRAY AA
      DO 40 K=1,NGT
      J=IWK(K)
      AA(1,K)=BB(1,J)
40    AA(2,K)=BB(2,J)
C     PUT THE GRADIENTS IN THE WK ARRAY
      N1=NDV
      DO 60 K=1,NGT
      DO 50 I=1,NDV
50       WK(I+N1)=AA(I,K)
      N1=N1+NDV
60    CONTINUE
      GO TO 20
C     ------------------------------------------------
C     INFO = 0. OPTIMIZATION IS COMPLETE. TERMINATE.
C     PRINT RESULTS
70    WRITE(6,80)OBJ,X(1),X(2),G(1),G(2)
      STOP
80    FORMAT(//5X,'OPTIMUM',5X,'OBJ =',E12.5//5X,'X(1) =',
     1E12.5,5X,'X(2) =',E12.5/5X,'G(1) =',E12.5,5X,'G(2) =',
     2E12.5)
      END
```

3.4 Interrupting and Restarting DOT

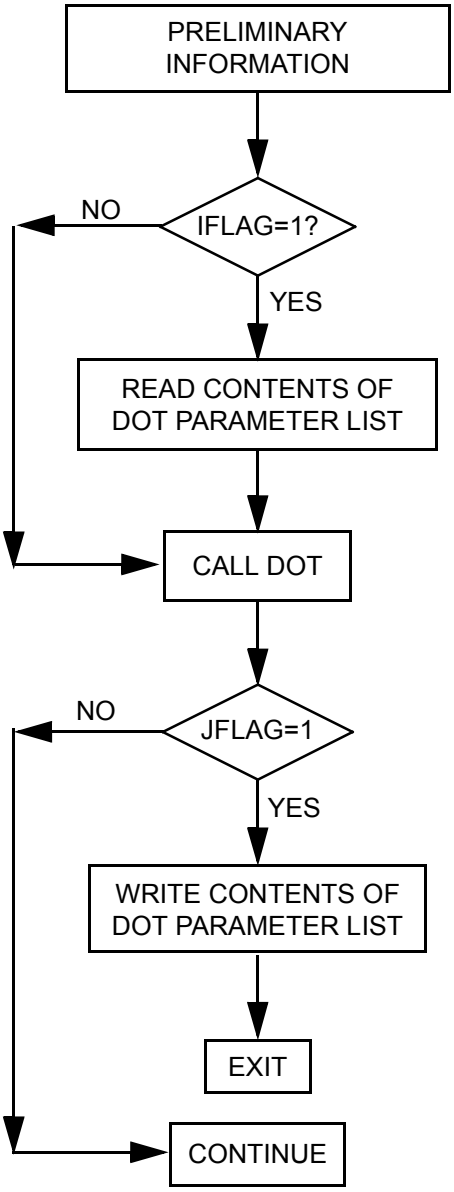
It is a simple matter to stop the optimization when you wish and then restart from that point at a later time. All that is required is that, on return from DOT, you write the entire contents of the parameter list to a file and then exit. Upon restarting, you simply read this information again and continue from the point where you exited. All internal parameters of DOT will have the values that they had at the time you exited. This provides you with the flexibility to review the optimization progress before continuing, and is particularly useful if computer resources are limited or if you just want to insure that the optimization is performing as expected.

The basic program flow is shown in the figure below. It is assumed here that you have defined parameters called IFLAG and JFLAG to indicate what you wish to do. The values of IFLAG and JFLAG are assumed to have the following meanings;

IFLAG = 1 - This is a restart. Read saved parameters and continue.

JFLAG = 1 - Save information and exit.

It is perhaps more common to interrupt the optimization process at the end of an iteration. This is easily accomplished by checking the value of NEWITR which is contained in **IPRM**(19). If NEWITR=n, iteration n+1 has begun. If you interrupt at this point, you can review the progress of the optimization before proceeding. If JWRITE>0, (JWRITE is stored in location 13 of **IPRM**), the current design will have been written to file JWRITE just before this return to the calling program. The modification to interrupt after each iteration is simply to check if JFLAG=1 and NEWITR=n, where n is the iteration number after you wish to exit. If both are true, write the information to a file and exit. Note that NEWITR will be set to 0 at the beginning of the first iteration, after the analysis has been performed for the initial design. Therefore, when NEWITR=0, you have only evaluated the functions for the initial design, but have not changed the design.


PRELIMINARY INFORMATION
IFLAG=1?
NO
YES
READ CONTENTS OF DOT PARAMETER LIST
CALL DOT
JFLAG=1
NO
YES
WRITE CONTENTS OF DOT PARAMETER LIST
EXIT
CONTINUE

3.5 Output to a Postprocessing Data File

By opening a file and setting JWRITE [**IPRM**(13)] to the value of that file number, the user can output useful design iteration history information. This can be used to make decisions based on the progress of the optimization, as well as for plotting the iteration history during or after the optimization process is complete.

If JWRITE is greater than zero, the following information is written to this file during the optimization process. This information is written as an ASCII file using the FORTRAN FORMATS shown;

INFORMATION	FORMAT
ITER, NDV, NCON	1X,3I10
OBJ	1X,E15.8
X-VECTOR	1X,5E15.8
G-VECTOR (if NCON > 0)	1X,5E15.8

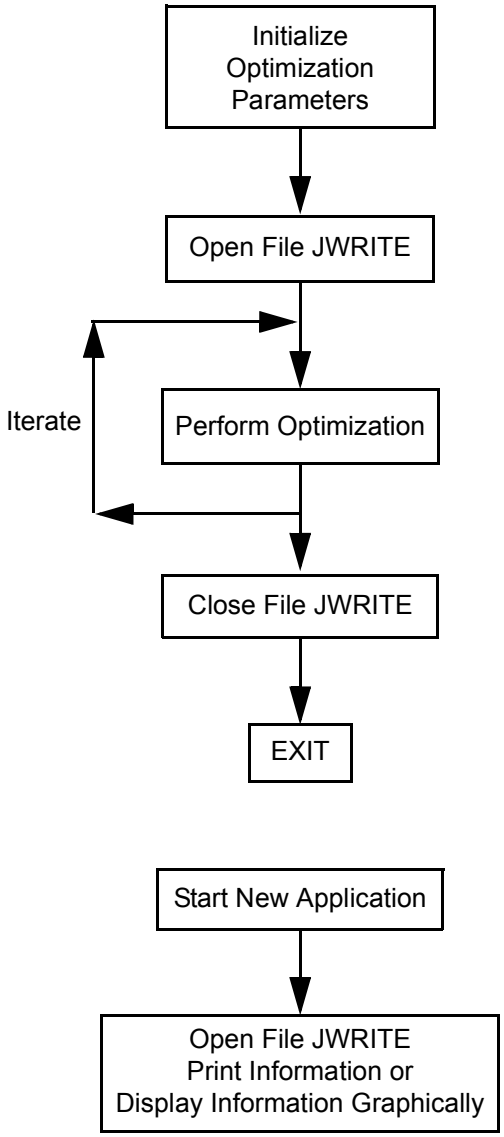
Note that the X-Vector and G-Vector are written in groups of 5 entries.

If METHOD = 0 or 1, or if METHOD = 2 and NCON = 0, this information is written after each one-dimensional search. If NCON > 0 and METHOD = 2 or 3, this information is written after each approximation to the problem is solved.

In addition to the information written after each iteration, the initial optimization information (ITER = 0) is also output.

It is the user's responsibility to position the file according to his/her needs. The usual application here is to open the file before optimization begins and then access it after the optimization process ends. This is shown in the figure below.

During the iteration loop of optimization, file JWRITE may be accessed. However, it is important to keep track of the file position so that all desired information is saved.


Initialize Optimization Parameters
Open File JWRITE
Iterate
Perform Optimization
Close File JWRITE
EXIT
Start New Application
Open File JWRITE
Print Information or
Display Information Graphically

CHAPTER 4

Examples

- Introduction
- Box Design
- Three-Bar Truss
- Cantilevered Beam
- Equilibrium of a Spring System
- Construction Management
- Piston Design
- Portfolio Selection
- Equality Constraints

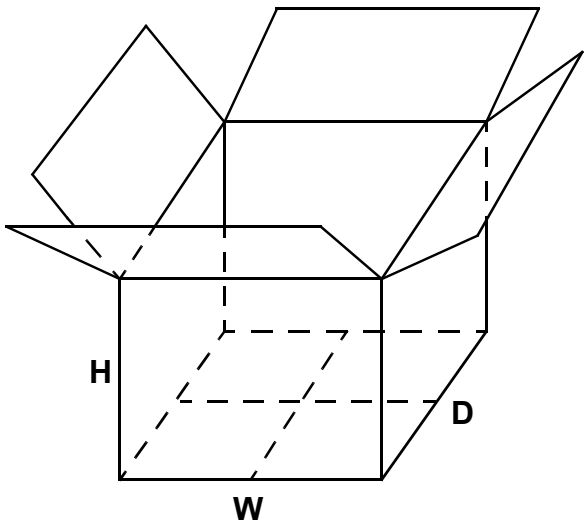
4.1 Introduction

This section presents examples of optimization problems taken from a variety of disciplines. There is a FORTRAN calling program included at the end of each example. You can begin using DOT immediately by solving these example problems. Refer to Section 2.5 for instructions on compiling the programs and linking them with DOT. The examples are as follows.

- Box Design (Material Minimization)
- Three-Bar Truss (Volume Minimization)
- Cantilevered Beam (Volume Minimization)
- Spring System Equilibrium (Nonlinear Structural Analysis Problem)
- Construction Management (Cost Minimization)
- Piston Oil Minimization (Volume Minimization)
- Portfolio Selection (Yield Maximization)
- Equality Constraint Example

4.2 Box Design

This is the very simple box design problem that described in Chapter 1. The goal is to determine dimensions H, W, and D to minimize the carton surface area required to enclose a volume of at least 2 cubic feet. The box is drawn as follows



The problem is presented in standard form as

Minimize Surface Area, S, where

$$S = 2.0*(W*H + D*H + 2.0*W*D) \quad (4\text{-}1)$$

Subject to;

$$2.0 - H*D*W \leq 0.0 \quad (4\text{-}2)$$

$$H, W, D \geq 0.0 \quad (4\text{-}3)$$

The theoretical optimum design for this problem is

$S = 12.00\ ft^2$, $V = 2.00\ ft^3$, $H = 2.00$ ft, $W = 1.00$ ft, $D = 1.00$ ft

The DOT solutions are

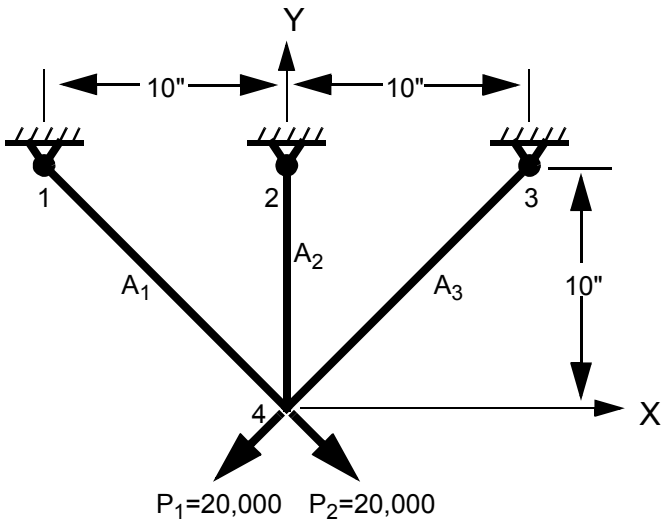
PARAMETER	INITIAL VALUE	OPTIMUM Method=1	OPTIMUM Method=2	OPTIMUM Method=3
H	1.0	2.057	2.011	2.136
W	1.0	0.985	0.997	0.969
D	1.0	0.985	0.997	0.969
OBJECTIVE	8.0	11.980	12.002	12.011
MAX G	0.5	0.0028	-2.2E-4	3.0E-4
FUNCTIONS		36	34	25

LISTING 4-1: BOX DESIGN FORTRAN PROGRAM.

```
C     SAMPLE PROGRAM. BOX DESIGN.
      DIMENSION X(5),XL(5),XU(5),G(5),
     *WK(800),IWK(200),RPRM(20),IPRM(20)
C     DEFINE NRWK, NRIWK.
      NRWK=800
      NRIWK=200
C     ZERO RPRM AND IPRM.
      DO 10 I=1,20
         RPRM(I)=0.0
10       IPRM(I)=0
C     DEFINE METHOD,NDV,NCON.
      METHOD=1
C     THREE DESIGN VARIABLES.
      NDV=3
C     ONE CONSTRAINT
      NCON=1
C     DEFINE BOUNDS AND INITIAL DESIGN.
      DO 20 I=1,NDV
C     INITIAL VALUES.
         X(I)=1.0
C     LOWER BOUNDS.
         XL(I)=0.0
C     UPPER BOUNDS
20       XU(I)=100.
C     DEFINE IPRINT, MINMAX, INFO.
C     PRINT CONTROL.
      IPRINT=1
C     MINIMIZE
      MINMAX=-1
C     INITIALIZE INFO TO ZERO.
      INFO=0
C     OPTIMIZE.
100   CALL DOT (INFO,METHOD,IPRINT,NDV,NCON,X,XL,XU,
     *OBJ,MINMAX,G,RPRM,IPRM,WK,NRWK,IWK,NRIWK)
C     FINISHED?
      IF(INFO.EQ.0)STOP
C     EVALUATE OBJECTIVE AND CONSTRAINT.
      CALL EVAL(OBJ,X,G)
C     GO CONTINUE WITH OPTIMIZATION.
      GO TO 100
      END
      SUBROUTINE EVAL (OBJ,X,G)
C     SUBROUTINE TO EVALUATE THE OBJECTIVE FUNCTION AND CONSTRAINTS
C     FOR THE BOX DESIGN PROBLEM.
      DIMENSION X(*),G(*)
      OBJ=2.0*X(2)*X(1)+2.0*X(3)*X(1)+4.0*X(2)*X(3)
      G(1)=1.0-0.5*X(1)*X(2)*X(3)
      RETURN
      END
```

4.3 Three-Bar Truss

This is the optimization of the 3-bar truss solved in Chapter 2 and shown in the figure below. This is a classical example in structural design. The objective function is the total volume of the material of the members. The decision variables X_1 and X_2 correspond to the cross-sectional areas of member 1 (and 3) and member 2, respectively. The area of member 3 is "linked" to be the same as member 1 for symmetry. The constraints are tensile stress constraints in members 1 and 2 under load P1. The loads, P_1 and P_2, are applied separately and the material specific weight is 0.1 lb. per cubic inch. The allowable stress in the members is 20,000 psi. This problem is formulated into the form given below. The original problem actually consists of 12 constraints, being the stress limit in each member under each of the 2 loading conditions. The complete problem is solved in the VisualDOC reference manual [16.]. Here, the problem has been abbreviated for clarity.



The problem is formally stated as follows

$$\text{Minimize} \quad OBJ = 2\sqrt{2}X_1 + X_2 \tag{4-4}$$

Subject to:

$$g_1 = \frac{2X_1 + \sqrt{2}X_2}{2X_1(X_1 + \sqrt{2}X_2)} - 1.0 \le 0.0 \tag{4-5}$$

$$g_2 = \frac{1.0}{X_1 + \sqrt{2}X_2} - 1.0 \le 0.0 \tag{4-6}$$

$$0.01 \le X_i \le 100.0 \quad i = 1,2 \tag{4-7}$$

The DOT solutions are

PARAMETER	INITIAL VALUE	OPTIMUM Method=1	OPTIMUM Method=2	OPTIMUM Method=3
A1	1.00	0.799	0.789	0.790
A2	1.00	0.372	0.408	0.406
A3	1.00	0.799	0.789	0.790
OBJECTIVE	3.828	2.633	2.639	2.640
MAX G	-0.293	0.0028	-0.0002	-0.0002
FUNCTIONS		29	34	22

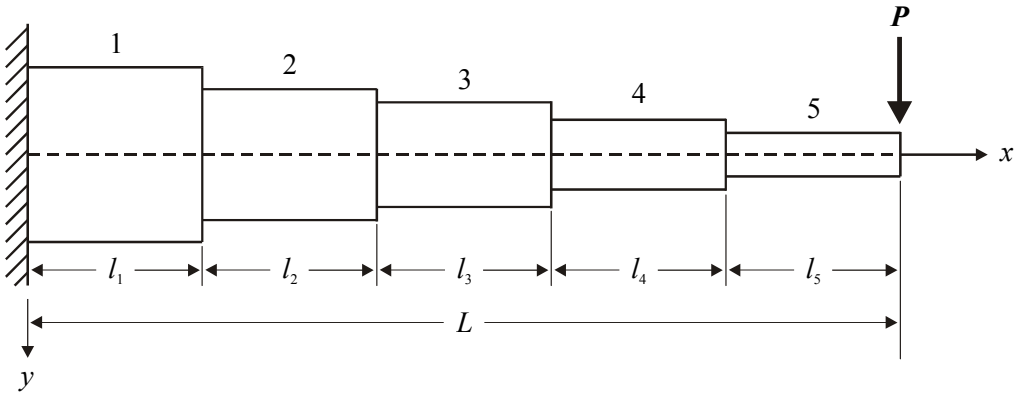
LISTING 4-2: THREE BAR TRUSS FORTRAN PROGRAM.

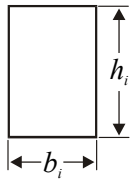
```
C     SAMPLE PROGRAM. THREE BAR TRUSS.
      DIMENSION X(2),XL(2),XU(2),G(2),
     *WK(800),IWK(200),RPRM(20),IPRM(20)
C     DEFINE NRWK, NRIWK.
      NRWK=800
      NRIWK=200
C     ZERO RPRM AND IPRM.
      DO 10 I=1,20
         RPRM(I)=0.0
10       IPRM(I)=0
C     DEFINE METHOD,NDV,NCON.
      METHOD=1
      NDV=2
      NCON=2
C     DEFINE BOUNDS AND INITIAL DESIGN.
      DO 20 I=1,NDV
         X(I)=1.0
         XL(I)=0.1
20       XU(I)=100.
C     DEFINE IPRINT, MINMAX, INFO.
      IPRINT=1
      MINMAX=-1
      INFO=0
100   CALL DOT (INFO,METHOD,IPRINT,NDV,NCON,X,XL,XU,
     *OBJ,MINMAX,G,RPRM,IPRM,WK,NRWK,IWK,NRIWK)
      IF(INFO.EQ.0)STOP
      CALL EVAL(OBJ,X,G)
      GO TO 100
      END

      SUBROUTINE EVAL (OBJ,X,G)
C     SUBROUTINE TO EVALUATE THE OBJECTIVE FUNCTION AND CONSTRAINTS
C     FOR THE SPRING EQUILIBRIUM PROBLEM.
      DIMENSION X(*),G(*)
      OBJ=2.0*SQRT(2.)*X(1)+X(2)
      G(1)=(2.*X(1)+SQRT(2.)*X(2))/(2.*X(1)*(X(1)+
     *SQRT(2.0)*X(2)))-1.
      G(2)=1./(2.*(X(1)+SQRT(2.)*X(2)))-1.
      RETURN
      END
```

4.4 Cantilevered Beam

The cantilevered beam shown below is to be designed for minimum material volume. The design variables are the width b and height h at each of N segments. We wish to design the beam subject to limits on stress (calculated at the left end of each segment), deflection under the load, and the geometric requirement that the height of any segment does not exceed twenty times the width.





Cross section

P = 50,000 N
E = 2.0x10^7 N/cm^2
L = 500 cm
$\overline{\sigma}$ = 14,000 N/cm^2
$\overline{y}$ = 2.5 cm

The deflection y_i at the right end of segment i is calculated by the following recursion formulas:

$$y_0 = y'_0 = 0 \tag{4-8}$$

$$y'_i = \frac{P l_i}{EI_i}\left[L + \frac{l_i}{2} - \sum_{j=1}^{i} l_j\right] + y'_{i-1} \tag{4-9}$$

$$y_i = \frac{P l_i^2}{2EI_i}\left[L - \sum_{j=1}^{i} l_j + \frac{2l_i}{3}\right] + y'_{i-1} l_i + y_{i-1} \tag{4-10}$$

where the deflection y is defined as positive downward, y'_i is the derivative of y with respect to x (the slope), and l_i is the length of segment i. Young's modulus E is the same for all segments, and the moment of inertia for segment i is

$$I_i = \frac{b_i h_i^3}{12} \tag{4-11}$$

The bending moment at the left end of segment *i* is calculated as

$$M_i = P\left[L + l_i - \sum_{j=1}^{i} l_j\right] \tag{4-12}$$

and the corresponding maximum bending stress is

$$\sigma_i = \frac{M_i h_i}{2I_i} \tag{4-13}$$

The design task is now defined as

Minimize:
$$V = \sum_{i=1}^{N} b_i h_i l_i \tag{4-14}$$

Subject to:

$$\frac{\sigma_i}{\bar{\sigma}} - 1 \le 0 \qquad i = 1, N \tag{4-15}$$

$$h_i - 20 b_i \le 0 \qquad i = 1, N \tag{4-16}$$

$$\frac{y_N}{\bar{y}} - 1 \le 0 \tag{4-17}$$

$$b_i \ge 1.0 \qquad i = 1, N \tag{4-18}$$

$$h_i \ge 5.0 \qquad i = 1, N \tag{4-19}$$

Here $\bar{\sigma}$ is the allowable bending stress and $\bar{y}$ is the allowable displacement. This is a design problem in $n = 2N$ variables. There are $N + 1$ nonlinear constraints defined by Eqs. (4-15) and (4-17), N linear constraints defined by Eq. (4-16), and $2N$ side constraints on the design variables defined by Eqs. (4-18) and (4-19). The side constraints are treated here as general inequality constraints. Additionally, lower bounds of 0.1 are imposed explicitly on b_i and h_i, $i = 1, N$ within the optimization program to ensure that the design remains physically meaningful.

This problem was solved using five segments (10 design variables).

The DOT solutions are

PARAMETER	INITIAL VALUE	OPTIMUM Method=1	OPTIMUM Method=2	OPTIMUM Method=3
b_1	5.00	3.11	3.14	3.15
b_2	5.00	2.88	2.89	2.88
b_3	5.00	2.62	2.58	2.57
b_4	5.00	2.20	2.20	2.21
b_5	5.00	1.75	1.92	1.75
h_1	40.00	61.20	62.71	63.02
h_2	40.00	57.55	57.70	57.54
h_3	40.00	52.31	51.64	51.32
h_4	40.00	44.09	44.09	44.10
h_5	40.00	35.00	33.42	35.01
OBJECTIVE	100,000	65,368	65,775	65,433
MAX G	0.5625	0.0024	0.00034	-0.0001
FUNCTIONS		194	160	140

Note that at the beginning of the program listing a parameter called NSEG is defined. In the above example, NSEG=5. You may change NSEG to increase or decrease the problem size. The program is dimensioned to allow a value of NSEG up to 50, to yield 100 design variables. If you wish to solve even larger problems, be sure to increase the array sizes.

This problem was solved again using fifty segments (100 design variables).

The DOT solutions are

PARAMETER	INITIAL VALUE	OPTIMUM Method=1	OPTIMUM Method=2	OPTIMUM Method=3
OBJECTIVE	100,000	63,825	63,908	63,763
MAX G	0.5625	4.2E-5	9.5E-7	1.9E-7
FUNCTIONS		4,194	2,225	1,029

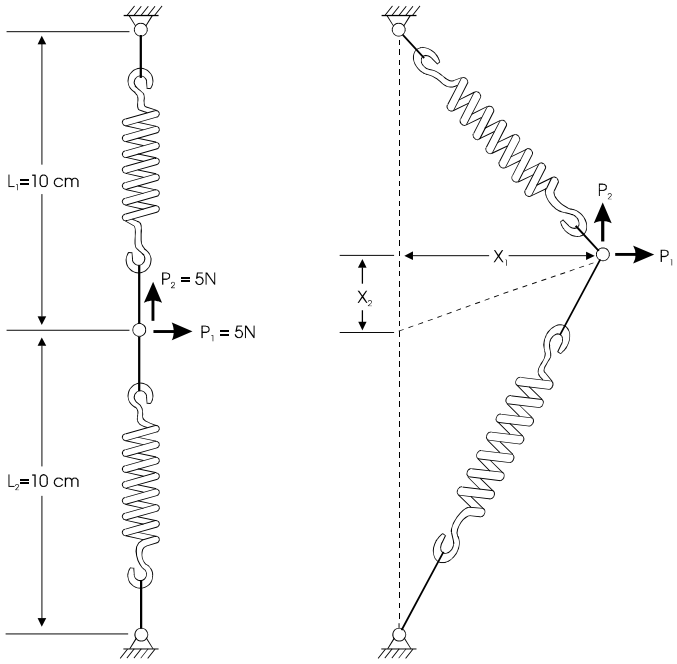
LISTING 4-3: CANTILEVER BEAM ANALYSIS FORTRAN PROGRAM.

```
C
C     SAMPLE PROGRAM.  NSEG ELEMENT BEAM DESIGN.
C
      DIMENSION X(1000),XL(1000),XU(1000),G(1001),
     *WK(10000000),IWK(10000),RPRM(20),IPRM(20)
C     DEFINE NRWK, NRIWK.
      NRWK=10000000
      NRIWK=10000
C
C     NUMBER OF BEAM SEGMENTS
C
      NSEG=5
C     ZERO RPRM AND IPRM.
      DO 10 I=1,20
         RPRM(I)=0.0
10       IPRM(I)=0
C     DEFINE METHOD,NDV,NCON.
      METHOD=3
C     TWO TIMES NSEG DESIGN VARIABLES.
      NDV=2*NSEG
C     TWO TIMES NSEG + 1 CONSTRAINTS.
      NCON=2*NSEG+1
C     DEFINE BOUNDS AND INITIAL DESIGN.
      DO 20 I=1,NSEG
C     INITIAL VALUES.
         X(I)=5.0
         X(I+NSEG)=40.0
C     LOWER BOUNDS.
         XL(I)=1.0
         XL(I+NSEG)=5.0
C     UPPER BOUNDS
         XU(I)=100.
         XU(I+NSEG)=100.
20    CONTINUE
C     DEFINE IPRINT, MINMAX, INFO.
C     PRINT CONTROL.
      IPRINT=1
C     MINIMIZE
      MINMAX=-1
C     INITIALIZE INFO TO ZERO.
      INFO=0
C     OPTIMIZE.
100   CALL DOT (INFO,METHOD,IPRINT,NDV,NCON,X,XL,XU,
     *OBJ,MINMAX,G,RPRM,IPRM,WK,NRWK,IWK,NRIWK)
C     FINISHED?
       if(info.eq.0) write(*,*)' final obj =',obj
      IF(INFO.EQ.0) STOP
C     EVALUATE OBJECTIVE AND CONSTRAINT.
      CALL EVAL(OBJ,X,G,NSEG)
C     GO CONTINUE WITH OPTIMIZATION.
      GO TO 100
      END
```

```
      SUBROUTINE EVAL (OBJ,X,G,NSEG)
      DIMENSION X(*),G(*)
C     5-ELEMENT BEAM.
C     NDV=10, NCON=11.
      P=50000.
      E=2.0E+7
      AL=500.
      ALI=AL/FLOAT(NSEG)
      SIG=14000.
      YMX=2.5
C     VOLUME, STRESS CONSTRAINTS, H/B CONSTRAINTS, DISPL. CONSTRAINT.
      VOL=0.
      ALA=0.
      Y=0.
      YP=0.
      DO 22 I=1,NSEG
      VOL=VOL+ALI*X(I)*X(I+NSEG)
      AI=X(I)*(X(I+NSEG)**3)/12.
      ALA=ALA+ALI
      AM=P*(AL+ALI-ALA)
      SIGI=.5*AM*X(I+NSEG)/AI
C     STRESS CONSTRAINTS.
      G(I)=SIGI/SIG-1.
C     H/B CONSTRAINTS.
      G(I+NSEG)=X(I+NSEG)-20.*X(I)
      G(I+NSEG)=.1*G(I+NSEG)
      Y=Y+.5*P*ALI*ALI*(AL-ALA+2.*ALI/3.)/(E*AI)+YP*ALI
      YP=YP+P*ALI*(AL+.5*ALI-ALA)/(E*AI)
22    CONTINUE
      G(2*NSEG+1)=Y/YMX-1.
      OBJ=VOL
      RETURN
      END
```

4.5 Equilibrium of a Spring System

The figure below shows a simple spring for which we wish to find the equilibrium position under the applied loads [1]. This is a nonlinear analysis problem. Loads P_1 and P_2 are applied to a two-spring system as shown. The equilibrium position is calculated by minimizing the potential energy (PE).



(a) Undeformed (b) Deformed

The total potential energy of the system is defined as;

$$PE = \frac{1}{2}K_1\delta L_1^2 + \frac{1}{2}K_2\delta L_2^2 - P_1X_1 - P_2X_2$$

$$= \frac{1}{2}K_1[\sqrt{X_1^2 + (L_1 - X_2)^2} - L_1]^2 + \frac{1}{2}K_2[\sqrt{X_1^2 + (L_2 + X_2)^2} - L_2]^2 \quad (4\text{-}20)$$

$$- P_1X_1 - P_2X_2$$

This equation is solved with unconstrained minimization (NCON = 0).

The DOT solutions are

PARAMETER	INITIAL VALUE	OPTIMUM Method=1	OPTIMUM Method=2
X1	0.00	8.631	8.631
X2	0.00	4.530	4.530
OBJECTIVE	0.00	-41.81	-41.81
FUNCTIONS		68	68

LISTING 4-4: SPRING EQUILIBRIUM ANALYSIS FORTRAN PROGRAM.

```
C
C     SAMPLE PROGRAM.  SPRING EQUILIBRIUM ANALYSIS.
C
      DIMENSION X(2),XL(2),XU(2),G(1),
     *WK(800),IWK(200),RPRM(20),IPRM(20)
C     DEFINE NRWK, NRIWK.
      NRWK=800
      NRIWK=200
C     ZERO RPRM AND IPRM.
      DO 10 I=1,20
         RPRM(I)=0.0
10       IPRM(I)=0
C     DEFINE METHOD,NDV,NCON.
      METHOD=2
      NDV=2
      NCON=0
C     DEFINE BOUNDS AND INITIAL DESIGN.
      DO 30 I=1,NDV
         X(I)=0.0
         XL(I)=-100.0
30       XU(I)=100.0
C     DEFINE IPRINT, MINMAX, INFO
      IPRINT=1
      MINMAX=-1
      INFO=0
100   CALL DOT (INFO,METHOD,IPRINT,NDV,NCON,X,XL,XU,
     *OBJ,MINMAX,G,RPRM,IPRM,WK,NRWK,IWK,
     *NRIWK)
      IF(INFO.EQ.0)STOP
      CALL EVAL(OBJ,X,G)
      GO TO 100
      END
      SUBROUTINE EVAL (OBJ,X,G)
C
C     SUBROUTINE TO EVALUATE THE OBJECTIVE FUNCTION AND CONSTRAINTS
C     FOR THE SPRING EQUILIBRIUM PROBLEM.
C
      DIMENSION X(*),G(*)
      OBJ=4.*(SQRT(ABS(X(1))**2 +
     * ABS(10.-X(2))**2 )-10.)**2
     *+.5*(SQRT(ABS(X(1))**2 +ABS(10.+
     *X(2))**2 )-10.)**2 -5.*X(1)-5.*X(2)
      RETURN
      END
```

4.6 Construction Management

A contractor is considering two gravel pits from which he may purchase material to supply a project [3]. The unit cost to load and deliver the material to the project site is \$5.00/yd.^{3} from pit 1 and \$7.00/yd.^{3} from pit 2. He must deliver a minimum of 10,000 yd.^{3} to the site.

The mix that he delivers must consist of at least 50 percent sand, no more than 60 percent gravel, nor more than 8 percent silt. (Note that there is some redundancy in the requirements.) The material at pit 1 consists of 30 percent sand and 70 percent gravel. The material at pit 2 consists of 60 percent sand, 30 percent gravel, and 10 percent silt. Determine how much material should be taken from each pit.

Since the gravel from pit 1 does not contain the minimum amount of sand to meet project requirements, the contractor may not utilize the cheaper material exclusively. He must mix the material from pits 1 and 2 to produce the required proportions.

We define the decision variables to be

X_1 = amount of material taken from pit 1 (in cubic yards)

X_2 = amount of material taken from pit 2 (in cubic yards)

The objective function is

$$\text{Minimize } C = 5X_1 + 7X_2$$

Let $X_1 + X_2$ equal the total amount of standard mix delivered to the project site. The contractor must deliver at least 10,000 cubic yards, thus the delivery constraint is

$$X_1 + X_2 \geq 10{,}000$$

The mixture must contain at least 50 percent sand. The contractor may obtain the desired amount of sand by combining the materials from each pit.

$$0.3X_1 + 0.6X_2 \geq 0.5(X_1 + X_2)$$

The products $0.3X_1$ and $0.6X_2$ are the amounts of sand taken from pits 1 and 2, respectively. The term $0.5(X_1 + X_2)$ is the amount of sand in the mix. Similarly, the constraint on the amount of gravel to be delivered is

$$0.7X_1 + 0.3X_2 \leq 0.6(X_1 + X_2)$$

Finally, the constraint equation for silt is

$$0.1X_2 \leq 0.08(X_1 + X_2)$$

Examples

The minimum cost model may be written as

Minimize $C = 5X_1 + 7X_2$

Subject to:

$X_1 + X_2 \geq 10{,}000$ (delivery)

$0.3X_1 + 0.6X_2 \geq 0.5(X_1 + X_2)$ (sand)

$0.7X_1 + 0.3X_2 \leq 0.6(X_1 + X_2)$ (gravel)

$0.1X_2 \leq 0.08(X_1 + X_2)$ (silt)

$X_1, X_2 \geq 0$

or in standard form,

Minimize $F(X_1, X_2) = 5X_1 + 7X_2$ (4-21)

Subject to:

$10{,}000 - X_1 - X_2 \leq 0$ (4-22)

$2X_1 - X_2 < 0$ (4-23)

$X_1 - 3X_2 < 0$ (4-24)

$X_2 - 4X_1 < 0$ (4-25)

$X_1, X_2 \geq 0$ (4-26)

The DOT solutions are

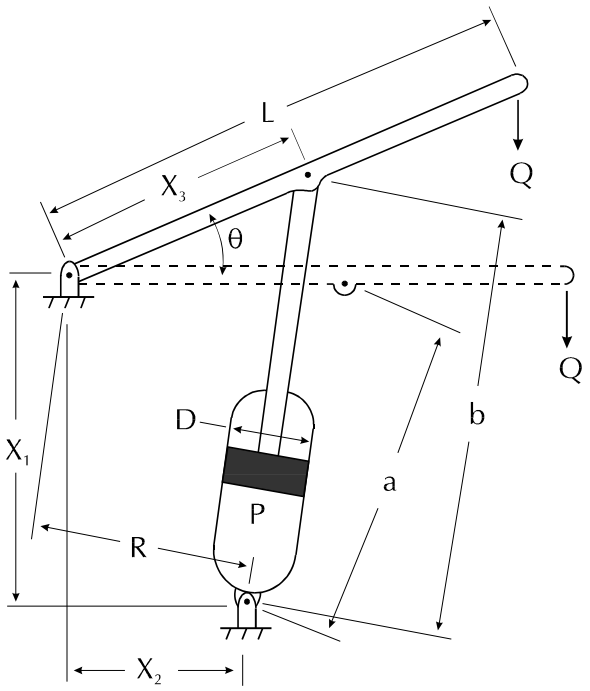
PARAMETER	INITIAL VALUE	OPTIMUM Method=1	OPTIMUM Method=2	OPTIMUM Method=3
X_1	3,000.0	3,333	3,333	3,334
X_2	3,000.0	6,667	6,696	6,668
OBJECTIVE	36,000.0	63,333	63,542	63,346
MAX G	0.40	0.000	-0.003	-5.8E-5
FUNCTIONS		13	13	12

LISTING 4-5: CONSTRUCTION MANAGEMENT FORTRAN PROGRAM.

```
C     SAMPLE PROGRAM.  CONSTRUCTION MANAGEMENT.
C
      DIMENSION X(2),XL(2),XU(3),G(4),
     *WK(800),IWK(200),RPRM(20),IPRM(20)
C     DEFINE NRWK, NRIWK.
      NRWK=800
      NRIWK=200
C     ZERO RPRM AND IPRM.
      DO 10 I=1,20
         RPRM(I)=0.0
10       IPRM(I)=0
C     DEFINE METHOD,NDV,NCON.
      METHOD=2
      NDV=2
      NCON=4
C     DEFINE BOUNDS AND INITIAL DESIGN.
      DO 20 I=1,NDV
         X(I)=3000.
         XL(I)=0.0
20       XU(I)=10000.
C     DEFINE IPRINT, MINMAX, INFO.
      IPRINT=1
      MINMAX=-1
      INFO=0
100   CALL DOT (INFO,METHOD,IPRINT,NDV,NCON,X,XL,XU,
     *OBJ,MINMAX,G,RPRM,IPRM,WK,NRWK,IWK,NRIWK)
      IF(INFO.EQ.0)STOP
      CALL EVAL(OBJ,X,G)
      GO TO 100
      END
      SUBROUTINE EVAL (OBJ,X,G)
C
C     SUBROUTINE TO EVALUATE THE OBJECTIVE FUNCTION AND CONSTRAINTS
C     FOR THE CONSTRUCTION MANAGEMENT PROBLEM.
C
      DIMENSION X(*),G(*)
      OBJ=5.*X(1)+7.*X(2)
      G(1)=(-X(1)-X(2)+10000.)/10000.
      G(2)=(2.*X(1)-X(2))/10000.
      G(3)=(X(1)-3.*X(2))/10000.
      G(4)=(-4.*X(1)+X(2))/10000.
      RETURN
      END
```

4.7 Piston Design

A piston lifts a load Q, as depicted below.



The specifications for the problem are

Payload (Q)= 10,000 lbs.

Beam Length (L)= 120 in.

Oil Pressure (P)= 1,500 psi

Maximum Allowable Bending Moment of the Beam $= 1.8 \times 10^6$ lbs-in.

The objective is to minimize the volume of oil required to lift the load from 0 to 45 degrees. There are constraints on force equilibrium, maximum bending moment of the beam, and minimum piston stroke. The pivot position, X_3, must not exceed L/2 and the support position must be at least half the piston diameter. The design variables are the distances X_1, X_2, X_3, and D. The problem is formulated as

$$\text{Minimize Volume of Oil, } V = \frac{\pi}{4}D^2(b-a) \tag{4-27}$$

Subject to;

$$\text{Equilibrium at } \theta = 45^o \quad QL\cos\theta - R(\theta)F \le 0 \tag{4-28}$$

$$\text{Bending Moment} \quad Q(L-X_3) - 1.8\times10^6 \le 0 \tag{4-29}$$

$$\text{Piston Stroke} \quad 1.2(b-a) - a \le 0 \tag{4-30}$$

Support Location $\frac{D}{2} - X_2 \le 0$ (4-31)

where

$$R = \frac{\left| -X_2(X_3 \sin\theta + X_1) + X_1(X_2 - X_3\cos\theta) \right|}{\sqrt{(X_3\sin\theta + X_1)^2 + (X_2 - X_3\cos\theta)^2}}$$

$$F = \frac{\pi}{4} P D^2$$

$$a = \sqrt{(X_3 - X_2)^2 + X_1^2}$$

$$b = \sqrt{(X_3 \sin 45^o + X_1)^2 + (X_1^2 - X_3 \cos 45^o)^2}$$

$$\text{Volume of Oil, } V = \frac{\pi}{4} D^2 (b - a)$$

The DOT solutions are

PARAMETER	INITIAL VALUE	OPTIMUM Method=1	OPTIMUM Method=2	OPTIMUM Method=3
X1	84.0	52.17	50.90	50.90
X2	60.0	4.01	3.26	3.27
X3	84.0	120.00	120.0	120.0
D	6.0	6.42	6.52	6.52
OBJECTIVE	1584.4	1034.3	1037.1	1035.4
MAX G	-0.133	0.003	-0.0004	-7.4E-8
FUNCTIONS		106	42	76

LISTING 4-6: PISTON DESIGN FORTRAN PROGRAM.

```
C     SAMPLE PROGRAM. PISTON DESIGN.
      DIMENSION X(4),XL(4),XU(4),G(5),
     *WK(800),IWK(200),RPRM(20),IPRM(20)
C     DEFINE NRWK, NRIWK.
      NRWK=800
      NRIWK=200
C     ZERO RPRM AND IPRM.
      DO 10 I=1,20
         RPRM(I)=0.0
10       IPRM(I)=0
C     DEFINE METHOD,NDV,NCON.
      METHOD=1
      NDV=4
      NCON=5
C     DEFINE BOUNDS AND INTIAL DESIGN.
      DO 20 I=1,NDV
         XL(I)=.05
20       XU(I)=1000.
      XU(3)=120.
      X(1)=84.0
      X(2)=60.0
      X(3)=84.0
      X(4)=6.0
C     DEFINE IPRINT, MINMAX, INFO.
      IPRINT=1
      MINMAX=-1
      INFO=0
100   CALL DOT (INFO,METHOD,IPRINT,NDV,NCON,X,XL,XU,
     *OBJ,MINMAX,G,RPRM,IPRM,WK,NRWK,IWK,
     *NRIWK)
      IF(INFO.EQ.0)STOP
      CALL EVAL(OBJ,X,G)
      GO TO 100
      END
      SUBROUTINE EVAL (OBJ,X,G)
C     SUBROUTINE TO EVALUATE THE OBJECTIVE FUNCTION AND CONSTRAINTS
C     FOR THE PISTON DESIGN PROBLEM.
      DIMENSION X(*),G(*)
C     DEFINE LOAD, LENGTH, PRESSURE, AND ALLOWABLE BENDING MOMENT.
      Q=10000.
      AL=240.
      P=1500.
      BNDMAX=1.8E+6
```

```
C     CALCULATE THE OBJECTIVE FUNCTION AND CONSTRAINTS.
      PIO4=3.1415927/4.0
      BO=SQRT(ABS(X(3)-X(2))**2+X(1)*X(1))
      B45=SQRT(ABS(X(3)*SIN(PIO4)+X(1))**2+ABS(X(3)*COS(PIO4)-X(2))**2)
      RNO=-X(2)*X(1)+X(1)*(X(2)-X(3))
      RDO=SQRT(X(1)**2+(X(2)-X(3))**2)
      RO=ABS(RNO/RDO)
      RN45=-X(2)*(X(3)*SIN(PIO4)+X(1))+X(1)*(X(2)-X(3)*COS(PIO4))
      RD45=SQRT((X(3)*SIN(PIO4)+X(1))**2+(X(2)-X(3)*COS(PIO4)**2))
      R45=ABS(RN45/RD45)
      OBJ=((PIO4)*X(4)*X(4))*(B45-BO)
      F=PIO4*P*X(4)*X(4)
      G(1)=(Q*AL/(RO*F)-1.0)
      G(2)=(Q*AL*COS(PIO4)/(R45*F)-1.0)
      G(3)=Q*(AL-X(3))/BNDMAX-1.
      G(4)=1.2*(B45-BO)/BO-1.
      G(5)=.5*X(4)-X(2)
      RETURN
      END
```

4.8 Portfolio Selection

A bank has $10 million to invest. Securities available for purchase are [4].

Bond Name	Bond Type	Quality	Years To Mature	Yield	After Tax Yield
A	Municipal	2	9	4.3%	4.3%
B	Agency	2	15	5.4%	2.7%
C	Government	1	4	5.0%	2.5%
D	Government	1	3	4.4%	2.2%
E	Municipal	5	2	4.5%	4.5%

Restrictions on the investments are:

1. Government and agency bonds must total at least $4 million.
2. Average quality must be 1.4 or less.
3. Average years to maturity must be 5 years or less.

The bank wants to maximize after tax earnings.

Decision variables can be defined as follows:

X_a = Amount invested in A (in $millions)

X_b = Amount invested in B (in $millions)

X_c = Amount invested in C (in $millions)

X_d = Amount invested in D (in $millions)

X_e = Amount invested in E (in $millions)

The after tax earnings can be expressed as follows:

Objective = $0.043X_a + 0.027X_b + 0.025X_c + 0.022X_d + 0.045X_e$

The constraints are expressed as follows:

Total investments should not exceed $10 million

$$X_a + X_b + X_c + X_d + X_e \leq 10.0$$

At least $4 million must be invested in government and agency bonds

$$X_b + X_c + X_d \geq 4.0$$

Average quality (total quality / total volume) must not exceed 1.4

$$\frac{2X_a + 2X_b + X_c + X_d + 5X_e}{X_a + X_b + X_c + X_d + X_e} \leq 1.4$$

$$0.6X_a + 0.6X_b - 0.4X_c - 0.9C_d + 3.6X_e \leq 0$$

Average maturity must not exceed 5 yrs.

$$\frac{9X_a + 15X_b + 4X_c + 3X_d + 2X_e}{X_a + X_b + X_c + X_d + X_e} \leq 5.0$$

$$4X_a + 10X_b - X_c - 2X_d - 3X_e \leq 0.0$$

The problem can now be formally stated as

$$\text{Maximize } 0.043X_a + 0.027X_b + 0.025X_c + 0.022X_d + 0.045X_e \quad (4\text{-}32)$$

Subject to

$$X_a + X_b + X_c + X_d + X_e - 10.0 \leq 0.0 \quad (4\text{-}33)$$

$$4.0 - (X_b + X_c + X_d) \leq 0.0 \quad (4\text{-}34)$$

$$0.6X_a + 0.6X_b - 0.4X_c - 0.4X_d - 3.6X_e \leq 0.0 \quad (4\text{-}35)$$

$$4X_a + 10X_b - X_c - 2X_d - 3X_e \leq 0.0 \quad (4\text{-}36)$$

The DOT solutions are

PARAMETER	INITIAL VALUE	OPTIMUM Method=1	OPTIMUM Method=2	OPTIMUM Method=3
Xa	5.0	2.18	2.19	2.79
Xb	5.0	0.00	0.00	0.00
Xc	5.0	7.36	7.37	3.50
Xd	5.0	0.00	0.00	3.42
Xe	5.0	0.45	0.45	0.30
OBJECTIVE	0.810	0.298	0.299	0.296
MAX G	0.40	1.0E-6	7.1E-5	6.4E-5
FUNCTIONS		63	61	27

LISTING 4-7: PORTFOLIO SELECTION FORTRAN PROGRAM.

```
C     SAMPLE PROGRAM. PORTFOLIO SELECTION.
      DIMENSION X(5),XL(5),XU(5),G(4),
     *WK(800),IWK(200),RPRM(20),IPRM(20)
C     DEFINE NRWK, NRIWK.
      NRWK=800
      NRIWK=200
C     ZERO RPRM AND IPRM.
      DO 10 I=1,20
         RPRM(I)=0.0
10       IPRM(I)=0
C     DEFINE NDV, NCON, METHOD.
      NDV=5
      NCON=4
      METHOD=1
C     DEFINE BOUNDS AND INITIAL DESIGN.
      DO 20 I=1,NDV
         X(I)=5.0
         XL(I)=0.0
20       XU(I)=10.
C     DEFINE IPRINT, MINMAX, INFO.
      IPRINT=1
      MINMAX=1
      INFO=0
100   CALL DOT (INFO,METHOD,IPRINT,NDV,NCON,X,XL,XU,
     *OBJ,MINMAX,G,RPRM,IPRM,WK,NRWK,IWK,
     *NRIWK)
      IF(INFO.EQ.0)STOP
      CALL EVAL(OBJ,X,G)
      GO TO 100
      END
      SUBROUTINE EVAL (OBJ,X,G)
C     SUBROUTINE TO EVALUATE THE OBJECTIVE FUNCTION AND CONSTRAINTS
C     FOR THE PORTFOLIO SELECTION PROBLEM.
      DIMENSION X(*),G(*)
      OBJ=0.043*X(1)+0.027*X(2)+0.025*X(3)+0.022*X(4)+0.045*X(5)
      G(1)=(X(1)+X(2)+X(3)+X(4)+X(5)-10.)/100.
      G(2)=(4.-X(2)-X(3)-X(4))/100.
      G(3)=(0.6*X(1)+0.6*X(2)-0.4*X(3)-0.4*X(4)+3.6*X(5))/100.
      G(4)=(4.*X(1)+10.*X(2)-X(3)-2.*X(4)-3.*X(5))/100.
      RETURN
      END
```

4.9 Equality Constraints

This is an example of how equality constraints are formulated. Consider the following mathematical programming problem

$$\text{Minimize OBJ} = (X_1 + X_2)^2 + (X_2 + X_3)^2$$

$$\text{Subject to: } h_1 = X_1 + 2X_2 + 3X_3 - 1.0 = 0.0$$

Note that the constraint in this case is an equality constraint. The constraint function must equal zero. This problem is set up for DOT as follows

$$\text{Minimize} \quad \text{OBJ} = (X_1 + X_2)^2 + (X_2 + X_3)^2 \tag{4-37}$$

Subject to;

$$g_1 = X_1 + 2X_2 + 3X_3 - 1.0 \leq 0.0 \tag{4-38}$$

$$g_2 = -(X_1 + 2X_2 + 3X_3 - 1.0) \leq 0.0 \tag{4-39}$$

or

$$g_1 = -g_2 \leq 0$$

The DOT solutions are

PARAMETER	INITIAL VALUE	OPTIMUM Method=1	OPTIMUM Method=2	OPTIMUM Method=3
X1	-4.00	0.475	0.454	0.554
X2	1.00	-0.469	-0.500	-0.588
X3	2.00	0.488	0.515	0.541
OBJECTIVE	18.00	3.8E-4	0.0024	0.0034
MAX G	3.00	8.4E-5	0.0004	2.7E-6
FUNCTIONS		43	203	34

LISTING 4-8: EQUALITY CONSTRAINTS FORTRAN PROGRAM.

```
C
C     SAMPLE PROGRAM.  EQUALITY CONSTRAINTS.
C
      DIMENSION X(3),XL(3),XU(3),G(2),
     *WK(800),IWK(200),RPRM(20),IPRM(20)
C     DEFINE NRWK, NRIWK.
      NRWK=800
      NRIWK=200
C     ZERO RPRM AND IPRM.
      DO 10 I=1,20
         RPRM(I)=0.0
10       IPRM(I)=0
C     DEFINE METHOD,NDV,NCON.
      METHOD=3
      NDV=3
      NCON=2
C     DEFINE BOUNDS AND INTIAL DESIGN.
      DO 20 I=1,NDV
         XL(I)=-100.
20       XU(I)=100.
       X(1)=-4.
       X(2)=1.
       X(3)=2.
C     DEFINE IPRINT, MINMAX, INFO.
      IPRINT=1
      MINMAX=-1
      INFO=0
100   CALL DOT (INFO,METHOD,IPRINT,NDV,NCON,X,XL,XU,
     *OBJ,MINMAX,G,RPRM,IPRM,WK,NRWK,IWK,NRIWK)
      IF(INFO.EQ.0)STOP
      CALL EVAL(OBJ,X,G)
      GO TO 100
      END
      SUBROUTINE EVAL (OBJ,X,G)
C
C     SUBROUTINE TO EVALUATE THE OBJECTIVE FUNCTION AND CONSTRAINTS
C     FOR THE EQUALITY CONSTRAINT PROBLEM.
C
      DIMENSION X(*),G(*)
      OBJ=(X(1)+X(2))**2 + (X(2)+X(3))**2
      G(1)=X(1)+2.*X(2)+3.*X(3)-1.
      G(2)=-G(1)
      RETURN
      END
```

CHAPTER 5

References

- **Introduction**
- **References**

5.1 Introduction

There is much to be gained from a review of some of the basic optimization literature. While DOT can often be satisfactorily used by the optimization novice, a better understanding of the theory of optimization can lead to more effective use of the program. The following is a list of publications which may be useful to those seeking a better understanding of numerical optimization.

5.2 References

1. Vanderplaats, G. N., **Numerical Optimization Techniques for Engineering Design: With Applications**, 3rd Edition, Vanderplaats Research & Development, Inc., Colorado Springs, CO, 1999.
2. Haftka, R. T., and Kamat, M. P., **Elements of Structural Optimization**, Nijhoff Publishers, Netherlands, 1985.
3. Ossenbruggen, P. J., **Systems Analysis for Civil Engineers**, John Wiley and Sons, N. Y., 1984.
4. Bradley, S. P., Hax, A. C. and Magnanti, T. L., **Applied Mathematical Programming**, Addison-Wesley, Mass. 1977.
5. Zangwill, W. I., **Nonlinear Programming: A Unified Approach**, Prentice-Hall, Englewood Cliffs, N.J., 1969.
6. Fletcher, R. and Reeves, C. M., "Function Minimization by Conjugate Gradients," Br. Computer J., Vol. 7, No. 2, pp. 149-154, 1964.
7. Broyden, C. G., "The Convergence of a Class of Double Rank Minimization Algorithms, Parts I and II," J. Inst. Math. Appl., Vol. 6, pp. 76-90, 222-231, 1970.
8. Fletcher, R., "A New Approach to Variable Metric Algorithms," Computer J., Vol. 13, pp. 317-322, 1970.
9. Goldfarb, D., "A Family of Variable Metric Methods Derived by Variational Means," Math. Comput., Vol. 24, pp. 23-26, 1970.
10. Shanno, D. F., "Conditioning of Quasi-Newton Methods for Function Minimization," Math. Comput., Vol. 24, pp. 647-656, 1970.
11. Kelly, J. E., "The Cutting Plane Method for Solving Convex Programs," J. SIAM, Vol. 8, pp. 702-712, 1960.
12. Dantzig, G. B., **Linear Programming and Extensions**, Princeton University Press, Princeton, N.J., 1963.
13. Powell, M. J. D., "Algorithms for Nonlinear Constraints that use Lagrangian Functions," Math. Prog., Vol. 14, No. 2, pp. 224-248, 1978.
14. Vanderplaats, G. N. and Sugimoto, H., "Application of Variable Metric Methods to Structural Synthesis," Engineering Computations, Vol. 2, No. 2, June, 1985.
15. GENESIS User's Manuals, VMA Engineering, Colorado Springs, CO, 1999.
16. DOC User's Manual, Vanderplaats Research & Development, Inc., 1995.
17. Thomas, H., Vanderplaats, G. and Shyy, Y-K, "A Study of Move Limit Adjustment Strategies in the Approximation Concepts Approach to Structural Synthesis," Proc. 4th AIAA/USAF/NASA/OAI Symposium on Multidisciplinary Analysis and Optimization, Holiday Inn, Cleveland, OH, Sept. 21-23, 1992, AIAA-92-4839.
18. Zoutendijk, K. G., **Methods of Feasible Directions**, Elsevier Publishing Co., Amsterdam, Netherlands, 1960.
19. Vanderplaats, G. N., "An Efficient Feasible Direction Algorithm for Design Synthesis," AIAA Journal, Vol. 22, No. 11, Nov. 1984.

References

APPENDIX A

Structure of Program Calling DOT

- **Introduction**
- **Basic Program Organization**
- **Structure of Program Interfacing with DOT**
- **Box Design Program in C Language Interfacing DOT Object Code Compiled in FORTRAN 77**

A.1 Introduction

The program given here may be used as a prototype as a main calling program for using DOT. All default parameters are defined prior to calling DOT. The defaults are contained in the **RPRM** and **IPRM** arrays. These are normally initialized to zero. Then, any over-ride values are set in the proper locations. For detailed information about using DOT with application programs, see Chapters 2 and 3 of this manual.

A.2 Basic Program Organization

NOTE: In the following outline of the program, a question mark (**?**) means that a value or values must be provided. Also, the parameters, such as NDV, given in the dimension statement are required values and must be replaced by actual numbers. Each place where a parameter must be supplied by the user is highlighted in ***italic***. Remember that the arrays can be dimensioned larger than the required values to allow for future expansion. Thus, the parameter BIG for **WK** and **IWK**, means that these arrays must be dimensioned at least large enough to solve the problem, but may be dimensioned larger. It is good practice to dimension these arrays as large as possible to allow for future expansion of the number of design variables and constraints.

A.3 Structure of Program Interfacing with DOT

FORTRAN Example.

```
      DIMENSION X(NDV),XL(NDV),XU(NDV),G(NCON),WK(BIG),IWK(BIG),
     * RPRM(30),IPRM(20)
C     DIMENSIONS OF WK AND IWK.
      NRWK=?
      NRIWK=?
C     ZERO RPRM AND IPRM
      DO 10 I=1,20
        RPRM(I)=0.0
10    IPRM(I)=0
C     AT THIS POINT SET ANY ENTRIES OF RPRM AND IPRM
C     TO THEIR DESIRED VALUES IF THE DEFAULTS ARE
C     TO BE OVER-RIDDEN.
C     E.G.
C     RPRM(1)=-0.01
C     DEFINE NDV, NCON, IPRINT, MINMAX, METHOD
      NDV=?
      NCON=?
      IPRINT=?
      MINMAX=?
      METHOD=?
C     DEFINE X, XL, XU
      X(I)=?, I=1,NDV
      XL(I)=?, I=1,NDV
      XU(I)=?, I=1,NDV
C     READY TO OPTIMIZE
      INFO=0
20    CALL DOT (INFO,METHOD,IPRINT,NDV,NCON,X,XL,XU,
     *OBJ,MINMAX,G,RPRM,IPRM,WK,NRWK,IWK,NRIWK)
C     EVALUATE OBJECTIVE AND CONSTRAINTS.  YOU MAY CALL ONE
C     OR MORE SUBROUTINES TO DO THIS.
      OBJ=?
      G(I)=?, I=1,NCON
      IF(INFO.GT.0) GO TO 20
C     OPTIMIZATION IS COMPLETE. OUTPUT RESULTS.
      STOP
      END
```

A.4 Box Design Program in C Language Interfacing DOT Object Code Compiled in FORTRAN 77

```
/* BOX DESIGN C MAIN PROGRAM */
/* MACHINE DEPENDENT DECLARATIONS */
/* PC */
#if defined (PC)
#define DOT DOT
extern void __stdcall DOT(int *INFO, int *METHOD, int *IPRINT,
            int *NDV, int *NCON, double *X, double *XL, double *XU,
            double *OBJ, int *MINMAX, double *G, double *RPRM, int *IPRM,
            double *WK, int *NRWK, int *IWK, int *NRIWK);
#endif
/* SUN and SGI Unix machines  */
#if defined (SUN) || defined (SGI)
#define DOT dot_
extern void DOT(int *INFO, int *METHOD, int *IPRINT,
            int *NDV, int *NCON, double *X, double *XL, double *XU,
            double *OBJ, int *MINMAX, double *G, double *RPRM, int *IPRM,
            double *WK, int *NRWK, int *IWK, int *NRIWK);
#endif
/*  IBM and HP Unix machines  */
#if defined (IBM) || defined (HP)
#define DOT dot
extern void DOT(int *INFO, int *METHOD, int *IPRINT,
            int *NDV, int *NCON, double *X, double *XL, double *XU,
            double *OBJ, int *MINMAX, double *G, double *RPRM, int *IPRM,
            double *WK, int *NRWK, int *IWK, int *NRIWK);
#endif
int main()
{
    int info, method, iprint, ndv, ncon, minmax, iprm[20], nrwk, iwk[200], nriwk;
       float x[5], xl[5], xu[5], obj, g[5],rprm[20],wk[800];
     int i;
/* define nrwk, nriwk */
            nrwk  = 800;
            nriwk = 200;
/* zero rprm and iprm arrays */
            for (i = 1; i <= 20; ++i) {
                    rprm[i - 1] = 0.0;
                    iprm[i - 1] = 0;
                }
/* modified method of feasible directions */
            method = 1;
/* three design variables */
            ndv = 3;
```

Structure of Program Calling DOT

```
/* one constraint */
                ncon = 1;
/* define bounds and initial design */
                for (i = 1; i <= ndv; ++i) {
                        x [i-1] = 1.0;
                        xl[i-1] = 0.01;
                        xu[i-1] = 100.0;
                  }
/* print control */
                iprint = 1;
/* minimize */
                minmax  =-1;
/* initialize infor to zero */
                 info = 0;
/* call DOT optimizer */
do
{
   dot_(&info, &method, &iprint, &ndv, &ncon, x, xl, xu, &obj, &minmax, g,
   rprm, iprm, wk, &nrwk, iwk, &nriwk);
/* evaluate objective and constraint */
                 eval(&obj, x, g);
 }while ( info != 0);
return 0;
}
/* function to evaluate the objective function and constraints */

int eval(obj, x, g)
float *obj, *x, *g;
{
   --g;
   --x;
                *obj = x[2] * 2. * x[1] + x[3] * 2. * x[1] + x[2] * 4. * x[3];
                g[1] = 1. - x[1] * .5 * x[2] * x[3];
                return 0;
} /* end of program */
```

Compiling and Linking:

The C main program must be compiled using your C/C++ compiler. You should preferabley link the object code of the C main program to DOT object code using a FORTRAN 77 linker. If you want to link using C linker, you must also link the FORTRAN 77 libraries, which means you must have FORTRAN 77 installed on your computer.

A Example of how to compile and link the box design program (boxdot.c) on a SGI workstation:

```
> cc -DSGI -c boxdot.c
> f77 -o boxdot boxdot.o dot.a
```

This will create the executable file “boxdot” which you may execute to perform optimization.

APPENDIX **B**

Calculating DOT Array Sizes

- **Introduction**
- **Unconstrained Problems (NCON=0)**
- **Constrained Problems (NCON > 0)**
- **Interactive Storage Calculations**

B.1 Introduction

Arrays **WK** and **IWK** must be dimensioned in any program that calls DOT. The minimum required dimensions, NRWK and NRIWK, can be calculated using the formulas given here:

B.2 Unconstrained Problems (NCON=0)

NGMAX = 0

METHOD = 1

NRWK = 11*NDV+NDV*(NDV+1)/2+40

NRIWK = NDV+71

METHOD = 2

NRWK = 11*NDV+40

NRIWK = NDV+71

B.3 Constrained Problems (NCON > 0)

The parameters NCOLA and NRB are used to determine internal storage needs for storing constraint gradients and solving the direction finding problem. Beginning with Version 5.0 of DOT, a single block of storage is used to provide the needed space. Three estimates are made, minimum, desired and maximum.

The needed information is calculated using the following formulas

```
IF METHOD=1 THEN

  NSIDE=0
  DO FOR I=1,NDV
    XLI=XL(I)
    XUI=XU(I)
    IF(ABS(XLI).GE.0.9E+30.AND.ABS(XUI).GE.0.9E+30) GO TO END DO
    IF(ABS(XLI).LT.0.9E+30.OR.ABS(XUI).LT.0.9E+30) THEN
      NSIDE=NSIDE+1
      DX=(XUI-XLI)/(MAX(ABS(XLI),ABS(XUI))+0.001)
      IF(DX.LE.0.001) NSIDE=NSIDE+1
    ENDIF
  END DO

ELSE

  NSIDE=2*NDV

ENDIF

NCOLMN=MIN(NCON,3,2*NDV)
NRBMN=NCOLM+MIN(3,(NSIDE+1))

NCOLA=MIN(NCON,2*NDV)
NGMAX=NCOLA
NRB=NCOLA+NSIDE+1

NCOLMX=NCON
NRBMX=NCOLMX+NSIDE+1

NSTRMN=NRBMN*(NRBMN+2)+NDV*NCOLMN

NSTORE=NRB*(NRB+2)+NDV*NCOLA

NSTRMX=NRBMX*(NRBMX+2)+NDV*NCOLMX

NSAVE=11*NDV+40

NRIWK=NDV+2*MAX(NDV,NGMAX)+NCON+71

IF METHOD = 2 THEN
  NSAVE=NSAVE+NDV*(3+NGMAX)

  NRIWK=NRIWK+NDV+NGMAX

ENDIF
```

```
IF METHOD=3 THEN
  NSAVE=NSAVE+NDV*(4+NGMAX)+NCON+NDV*(NDV+1)/2
    +MAX(NDV,NCON)
  NRIWK=NRIWK+NGMAX
ENDIF
NRWKMN=NSTRMN+NSAVE
NRWKD=NSTORE+NSAVE
NRWKMX=NSTRMXNSAVE
```

If the value of NRWK provided to DOT is less than NRWKMN, or if NRIWK is less than the value calculate here, the optimization will be terminated with an error message.

If the value of NRWK provided to DOT is greater than NRWKD, the parameter NSTORE will be increased to take advantage of the available storage.

B.4 Interactive Storage Calculations

Alternatively, NRWK and NRIWK can be calculated interactively using a program called DTSTOR. This program is provided with DOT and contains Subroutine DOT510. After compiling and linking DTSTOR, you can run it interactively to determine needed array sizes. This program requests the values of NDV, NCON, XL and XU and allows a value of NGMAX to be specified. DTSTOR then prints the required values of NRWK and NRIWK on the terminal.

This program calls SUBROUTINE DOT510, which may be directly called from a user program. The parameter list of SUBROUTINE DOT510 is as follows

```
SUBROUTINE DOT510 (NDV,NCON,METHOD,NRWK,NRWKMN,NRIWD,
*NRWKMX,NRIWK,NSTORE,NGMAX,XL,XU,MAXINT,IERR)
```

where NDV is the number of design variables, NCON is the number of constraints, **XL** and **XU** are arrays containing the lower and upper bounds on the design variables, respectively. NRWK is the number of rows in the **WK** array, and NRIWK is the number of rows in the **IWK** array. On input, DOT510 requires NDV, NCON, METHOD, **XL**, **XU** and MAXINT. MAXINT is the largest integer number that can be represented. DOT510 outputs the minimum, desired and maximum theoretical required values of NRWK, as NRWKMN, NRWKD and NRWKMX, respectively. Also, DOT510 outputs the required integer storage NRIWK, as well as the maximum number of gradients evaluated, NGMAX.

Note that, if you are doing dynamic memory allocation in your program, you may call DOT510 directly to calculate the needed memory for the **WK** and **IWK** arrays.

APPENDIX C

In Case of Difficulty

- **Introduction**
- **Debugging Procedure**

C.1 Introduction

DOT is a robust numerical optimizer, and there should seldom be a case where no progress is made during the optimization. Also, numerous internal checks are made to avoid exponent overflows, divide by zero, and similar run-time errors.

Usually, when something seems wrong, it can be traced to the basic setup of the optimization problem or (more often) simple programming errors. Thus, while it is difficult to project all possible errors of this sort, some are common enough to be able to offer a short list of items to check.

C.2 Debugging Procedure

It is suggested that the following steps be followed in order to try to isolate problems:

1. Check all array dimension statements. Be sure the values of NRWK and NRIWK are correct. If your code is written in double precision, be sure you are using the double precision version of DOT.
2. Check the parameter list for calling DOT. Be sure that all parameters are present and in the proper order. A common error is to create a program with an editor that allows 80 column lines, while using a compiler that ignores all characters after column 72.
3. Turn off the automatic scaling and try again. Sometimes the scaling actually makes the conditioning of the problem worse. If the difficulties still exist, leave the scaling turned off during further testing.
4. Set the print control, IPRINT, to 5. This will cause the gradient information to be printed during the optimization. If the gradient of the objective or any constraint function has all zeroes, this parameter is not a function of the design variables. While it is theoretically possible to have a zero gradient, it is extremely rare on a digital computer. Check the problem formulation.
5. Check the order of magnitude of the components of the gradients. A well conditioned problem will have roughly the same order of magnitude values (within a factor of 100). If one term is several orders of magnitude greater that the others, it may help to scale this design variable by dividing by a number of that order of magnitude. If the components of the gradient of a constraint are all very large it will probably help to divide that constraint by a large number. A common error in problem formulation is to have a function, say Q, that must be less than QQ, where QQ is on the order of 10,000. In creating the constraint (which is required to be less than or equal to zero) we may write **G**(I) = Q - QQ. This will make the constraint very difficult to deal with by DOT, because Q must equal about 9,999.97 before the constraint is considered active. Therefore, it is important to normalize the constraint as **G**(I) = Q/QQ - 1.0. Now a constraint value of -0.01 will identify the constraint as being within one percent of being critical.
6. As a last resort, set IPRINT equal to 7 or IPRNT2 equal to 2. This will turn on print in the one-dimensional search. Plot the objective and constraint functions versus the move parameter, ALPHA. If one or more are extremely nonlinear, reformulation of the problem by dividing that function by a large number is indicated. Another possibility here is that the finite difference gradient parameters, FDCH and FDCHM are either too large or too small. If the analysis is iterative, it often helps to try FDCH = 0.02 or larger and FDCHM = 0.01 or larger. This will mask the inaccuracies in the analysis. On the other hand, if the analysis is calculated very precisely as functions of the design variables, an order of magnitude smaller than the default value is indicated.
7. If the last resort fails, please contact VR&D for assistance. If possible, provide some printed output. Set IPRINT = 7 so all of the information will be there. We will do our best to help.

APPENDIX D

Internal Parameters in DOT

- **Introduction**
- **Parameters Contained in RPRM**
- **Parameters Contained in IPRM**

D.1 Introduction

In this Appendix, a description of the DOT internal parameters is given. While this is necessarily brief, it is somewhat more detailed than the cryptic information contained in Chapter 3. Appendix E also refers to these parameters and discusses how they are used in the mathematics of the optimization process.

The parameters are listed in the order they appear in the **RPRM** and **IPRM** arrays. If it is unlikely that the parameter should be changed from its default value, this is stated. In addition to Appendix E, Reference 1 describes the algorithms contained in DOT, and may be referred to for a more detailed description of how a parameter is used in a given algorithm.

D.2 Parameters Contained in RPRM

1. CT - This is the constraint tolerance. This parameter defines when a constraint is considered active.

   One of the key issues in constrained optimization is determining when a constraint is numerically "critical". If a constraint, **G**(I) is numerically greater than CT (CT is a negative number), it is considered critical for purposes of finding a new search direction or deciding if the optimum has been found. This is also why the constraint should be normalized to order of magnitude unity. Thus if **G**(I) is numerically greater than CT (say -0.03) then it is assumed to be within 3 percent of being critical. Numerically, this is considered to be an "active" constraint.

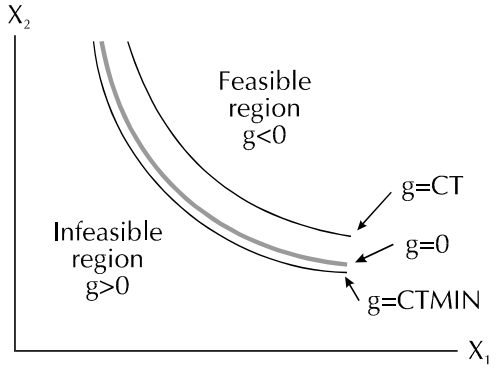
   For highly nonlinear problems, it is often helpful to make CT more negative, say -0.05 or -0.10. By this method, the constraint is "trapped" sooner and the optimization process will direct the design away from this constraint. On the other hand, if the constraints are nearly linear, it may help to make CT closer to zero, say -0.01. Then, when interpolating for **G**(I) = 0.0, a more precise value of **G**(I) is obtained. In either case, the value of CT is progressively reduced during optimization to a value of -CTMIN, which is the value at which a constraint becomes strongly critical. In fact, if **G**(I) exceeds CTMIN (a positive number), the constraint is considered to be violated. (See the definition of CTMIN)

   If one or more constraints repeatedly become active on one iteration and inactive on the next, CT should be increased in magnitude (say try CT = -0.05 or -0.10), or the offending constraint should be divided by a factor of ten to reduce its sensitivity.

2. CTMIN -This is a constraint tolerance for defining when inequality constraints are violated. CTMIN is a small positive number. A constraint is considered inactive if its value is more negative than CT. If the constraint value is more positive than CTMIN, it is considered violated.

   Since, mathematically a constraint is violated any time its value is greater than zero, there may be a temptation to set CTMIN=0. However, this should not be done because the optimization algorithms interpolate on zero and some numerical bandwidth should be provided to allow for inaccuracies. The default value allows for about a half of a percent constraint violation for normalized constraints.

   The geometric relationship between a constraint **G** and the parameters CT and CTMIN is shown in the following figure



3. DABOBJ - This is the absolute convergence criterion for optimization. If the objective function is changed by less than this value for ITRMOP consecutive iterations, the optimization will terminate. If the objective function changes by more than one order of magnitude during optimization, the default value for DABOBJ will probably cause premature convergence. In this case, it is usually desirable to set DABOBJ to a small number, say 0.001, and let the optimization process converge based on the relative change criterion defined by DELOBJ.

4. DELOBJ -This parameter is used in conjunction with DABOBJ. Here the convergence is tested on the relative change in the objective function. The combination of DABOBJ and DELOBJ work together to form the diminishing returns convergence criteria in DOT. Here by relative change we mean the fractional change in the value of the objective function between successive iterations.

   If the objective function is quite small in magnitude, a relative change of, say, one percent may not be meaningful. The absolute convergence criterion are relied on to detect convergence. On the other hand, for large values of the objective function, the absolute change is considered of lesser importance and the relative criterion tends to control the optimization convergence.

5. DOBJ1 -This is used in the one-dimensional search. On the first search, it is difficult to estimate a desirable move parameter, ALPHA, because the optimization process has no history. DOBJ1 is used to estimate the ALPHA which will reduce the objective function by this fraction, based on a linear approximation to the problem. Thus, for DOBJ1 = 0.1, the first step in the one-dimensional search will attempt to reduce the objective by ten percent.

   If the problem is highly nonlinear, so that the calculated ALPHA is consistently less than the proposed ALPHA, efficiency will be improved by reducing DOBJ1. Alternatively, if the calculated ALPHA is consistently greater than the proposed ALPHA, it is desirable to increase DOBJ1.

6. DOBJ2 -If the objective function is quite large in magnitude, a move to reduce the objective by the fraction DOBJ1 may be too large. In this case, DOBJ2 is used to limit the change in the objective function to the magnitude of DOBJ2. In other words, DOBJ1 is a fractional change and DOBJ2 is an absolute change. As with DOBJ1, if the proposed moves are too large, DOBJ2 may be reduced. If the proposed moves are too small, DOBJ2 may be increased.

   Both DOBJ1 and DOBJ2 are updated during the optimization process by keeping track of progress. Therefore, their initial values are usually not too critical except for highly nonlinear problems where no progress can be made due to very large estimates for ALPHA.

7. DX1, DX2 -These are used in the one-dimensional search. These parameters have an equivalent meaning to DOBJ1 and DOBJ2, but here are applied to each component of the **X** vector. The same general rules apply. The purpose of DX1 and DX2 is to prevent very large initial changes in the components of the X vector. DX1 and DX2 are also updated during the optimization process.

8. FDCH -Used if **IPRM**(1) = 0 for internal gradient calculations by DOT. Gradients are calculated by first forward finite difference unless a variable is at its upper bound. In this case, a first backwards finite difference step is taken and no check is made to insure that the resulting design variable is above its lower bound. FDCH is the finite difference step size as a fraction of the design variable being perturbed.

If high precision is available and required in evaluating the objective and constraint functions, FDCH should be reduced. If the analysis is iterative, with its own internal convergence parameters, FDCH may have to be increased. For iterative analysis, a value of FDCH up to 0.05 may be appropriate for constrained problems, but FDCH = 0.02 is a more reasonable limit for unconstrained problems. The reason for this is that DOT seeks the point where the gradient is zero for unconstrained problems, and if FDCH is large, this is numerically difficult and will lead to false gradient information. On the other hand, for constrained problems, the gradients of the objective and critical constraints are usually non-zero at the optimum. Hence, precision is less important for constrained problems.

9. FDCHM -This is used if **IPRM**(1) = 0 for internal gradient calculations by DOT. This is the minimum absolute step size for gradient calculations. This is used if the component of **X** is near zero since a fractional change may not be meaningful. The same general rules apply as with FDCH.

10. RMVLMZ - This is the relative change allowed in each design variable during the first iteration of the Sequential Linear Programming and Sequential Quadratic Programming Methods. Because the objective and constraint functions are approximated by a first order Taylor Series expansion, the information is only valid in the region of the approximation. Thus, we do not allow the design variables to change too much for each approximation. Depending on the progress of the optimization, this parameter will be updated. If the approximate optimization does not lead to significant constraint violations, this parameter is unchanged. However, if after an approximate optimization, one or more constraints are found to be violated more than they were at the beginning of the approximate optimization, RMVLMZ is reduced by 50% (this is only done after the second and subsequent iterations of the SLP method since on the first iteration it is common to go into the infeasible region).

    If the problem is known to be linear, RMVLMZ should be increased to a large value, say 100. If the problem is found to be quite nonlinear (the more common case), RMVLMZ should be reduced to 0.2 or even 0.1.

    Move limits on the individual design variables are internally adjusted, based on whether the design variable is consistently changing in one direction or if its value is oscillating [17.].

11. DABSTR - This parameter has the same meaning as DABOBJ, but here it is applied to convergence of the Sequential Linear Programming and Sequential Quadratic Programming Methods. That is, if the absolute value of the objective does not change by more than this amount between two consecutive solutions of the approximate optimization problem, this criterion is considered satisfied. Normally this parameter does not need to be changed.

12. DELSTR - This parameter has the same meaning as DELOBJ, but here it is applied to convergence of the Sequential Linear Programming and Sequential Quadratic Programming Methods. That is, if the relative value of the objective does not change by more than this amount between two consecutive solutions of the approximate optimization problem, this criterion is considered satisfied. Normally this parameter does not need to be changed.

D.3 Parameters Contained in IPRM

1. IGRAD -Specifies whether the gradients of the objective function and the constraints are calculated by DOT using finite difference methods (IGRAD = 0) or are supplied by the user (IGRAD = 1). If the gradients are readily available then directly providing gradients can save much computer time.
2. ISCAL -Specifies whether the design variables, objective and constraint functions, and gradients are to be scaled by DOT (ISCAL = 0 or greater) or if the problem is to remain unscaled (ISCAL = -1). The problem is actually re-scaled every ISCAL iterations where the default (if the user sets ISCAL = 0) is NDV. While there has not been much research into the theory of scaling, practical experience shows that scaling often serves to improve the conditioning of many problems. It should be noted, however, that sometimes scaling actually makes a problem worse. In this case the scaling should be turned off by setting ISCAL = -1 (ISCAL = 0 is the default).
3. ITMAX -Maximum number of iterations in the optimizer. If function evaluations are extremely expensive, reduce ITMAX. In the extreme case ITMAX = 1 or 2 is justified because the first few iterations are where most progress is made. If function evaluations are not expensive and the optimization terminates by reaching ITMAX, it should be increased.
4. ITRMOP -The number of consecutive iterations that must satisfy the absolute or relative convergence criteria before optimization is terminated in the Modified Method of Feasible Directions or the BFGS or Fletcher-Reeves Methods. Usually ITRMOP should be at least 2 because it is common to make little progress on one iteration, only to make major progress on the next. Therefore, the default of ITRMOP = 2 will allow a second try before terminating. If progress toward the optimum seems slow, but consistent, and function evaluations are not too expensive, it may improve the solution to increase ITRMOP to a value of 3 to 5.
5. IWRITE - File number to which DOT output will be sent. The default value is 6.
6. NGMAX - This is the maximum number of columns in the A-Matrix. This matrix is used to store the gradients of all active, violated, and near active constraints. The A-Matrix is stored in the **WK** array and so storage must be allocated for it. Ideally, NGMAX = NCON since it is conceivable that all constraints are active or violated. On the other hand, theoretically there should never be more than NDV active or violated constraints since this would define a “fully constrained” problem. However, this rule is often violated at the start because we do not have a good feasible starting point.

   The reason that the value of NGMAX is a concern is that we may have only a few design variables but thousands of constraints. Therefore, if we set NGMAX = NCON, this would require a large amount of storage. Normally, it is best do dimension **WK** and **IWK** as large as possible and then let DOT use the default value for NGMAX. If a storage error occurs, NGMAX (and if necessary, the dimensions of **WK** and **IWK**) should be increased if possible. Otherwise, choose a different starting design in an attempt to reduce the number of critical constraints.

7. JTMAX - This is the maximum number of iterations allowed in the Sequential Linear Programming and Sequential Quadratic Programming Methods. It is equivalent to ITMAX for the Modified Method of Feasible Directions. This is only used if METHOD=2 or 3 and NCON>0. When using these methods, ITMAX should be set to it's default value since solution of the approximate sub-problem is not usually expensive. If function values are very expensive, it is suggested to set JTMAX to a value of 1 or 2 to judge the progress of the optimization before continuing.

8. ITRMST - The number of consecutive iterations that must satisfy the absolute or relative convergence criteria before optimization is terminated in the Sequential Linear Programming and Sequential Quadratic Programming Methods. This is equivalent to ITRMOP, but now defines the number of approximate problems that will be solved before exiting. Usually ITRMST should be at least 2 because it is common to make little progress on one iteration, only to make major progress on the next. Therefore, the default of ITRMOP = 2 will allow a second try before terminating. If progress toward the optimum seems slow, but consistent, and function evaluations are not too expensive, it may improve the solution to set ITRMOP to a value of 3 or 4.

9. JPRINT - This is a debugging print control with the Sequential Linear Programming and Sequential Quadratic Programming Methods. Normally it should remain at its default value of 0. If JPRINT>0, IPRINT will be turned on during the sub-optimization problem. This will generate a considerable amount of output and should be used only for debugging purposes. Normally, this sub-problem is solved reliably and so it is not desirable to monitor its progress.

10. IPRNT1, IPRNT2 - These parameters just provide additional print that is used for debugging purposes. Normally, they should not be changed from their default values of 0. A value if IPRINT of 6 or 7 will turn on these print controls.

11. JWRITE - This is a file number for outputting information about the optimization history. If JWRITE=0, no output will be provided. IF JWRITE>0, it is assumed that the user has opened the file before calling DOT. DOT will then output the iteration history to this file. This is useful if you wish to create a graphics file showing the optimization progress or if you wish to output a summary of the optimization process in tabular form. In version 1.xx of DOT, this information was difficult to get because when DOT returned to the calling program, at the beginning of a new iteration, the values of **X**, **G** and OBJ were often different than they were on the last return. This is because, during the one-dimensional search, the last design investigated may be rejected in favor of an earlier design. The information on file JWRITE is consistent and includes the initial design information as well as the results at the end of each design iteration.

    If the Modified Method of Feasible Directions, BFGS or Fletcher-Reeves method is used, this information is written at the end of each one-dimensional search. If the Sequential Linear Programming or Sequential Quadratic Programming Method is used, this information is written at the end of each approximate optimization.

12. NEWITR - This parameter is not input to DOT, but is calculated internally. NEWITR will have a value of -1 or n. If it's value is n, optimization iteration number 'n' has just ended and a new iteration has begun (or if INFO=0, the optimization is complete). The value of n = 0 is returned to indicate that the initial design has just been evaluated (this is the result of iteration 0).

    If finite difference methods are used to calculate gradients, the first term in the **X** vector will already be changed when NEWITR = -1 (NEWITR = -1 will be returned NDV times during gradient calculations). Therefore, it is difficult for the user to interact to create graphics files or to monitor the optimization progress. NEWITR provides a means of determining exactly what is happening and acting accordingly.

    If JWRITE>0, the current optimization information has just been written to that file. Therefore, this is a good place to either interrogate that file or to interrupt and later restart the program.

13. NGT - This parameter is not input to DOT, but is calculated internally. NGT is the number of constraints for which gradients are needed and it is only meaningful if the user provides gradients. NGT will usually be much less than NCON since gradients of all constraints are usually not needed. The first NGT locations of the **IWK** array identify the constraint numbers for which gradients are required. The user must, in turn, store these gradients in the **WK** array, following the gradient of the objective function.

APPENDIX **E**

Optimization Algorithms

- **Introduction**
- **Basic Optimization Concepts**
- **Unconstrained Minimization**
- **Constrained Minimization**
- **Summary**

E.1 Introduction

The user should not normally require a detailed understanding of what the optimization routines in DOT actually do. However, some knowledge of this process will be invaluable as an aid to properly formulating the design task for efficient solution.

Here we offer a general overview of the numerical optimization process and the algorithms presently used in DOT. It is assumed here that the user is familiar with matrix algebra. Also, it is assumed that the user understands that the collection of design variables may be viewed as a vector so that any change in the design is also viewed as a move in vector space.

E.2 Basic Optimization Concepts

Mathematical programming provides a very general framework for scarce resource allocation and the basic algorithms originate in the operations research community. Engineering applications include chemical process design, aerodynamic optimization, nonlinear control system design, mechanical component design, structural design and a variety of others. Because the statement of the numerical optimization problem is so close to the traditional statement of engineering design problems, the variety of tasks to which it can be applied is inexhaustible.

In the most general sense, numerical optimization solves the nonlinear, constrained problem; Find the set of design variables, X_i, i=1,N, contained in vector **X**, that will

$$\text{Minimize} \quad F(\mathbf{X}) \tag{E-1}$$

Subject to;

$$g_j(\mathbf{X}) \le 0 \qquad j = 1, M \tag{E-2}$$

$$h_k(\mathbf{X}) = 0 \qquad k = 1,L \tag{E-3}$$

$$X_i^L \le X_i \le X_i^U \qquad i = 1,N \tag{E-4}$$

Equation E-1 defines the objective function which depends on the values of the design variables, **X**. Equations E-2 and E-3 are inequality and equality constraints respectively, and equation E-4 defines the region of search for the minimum. The bounds defined by equation E-4 are referred to as side constraints. A clear understanding of the generality of this formulation makes the breadth of problems that can be addressed apparent.

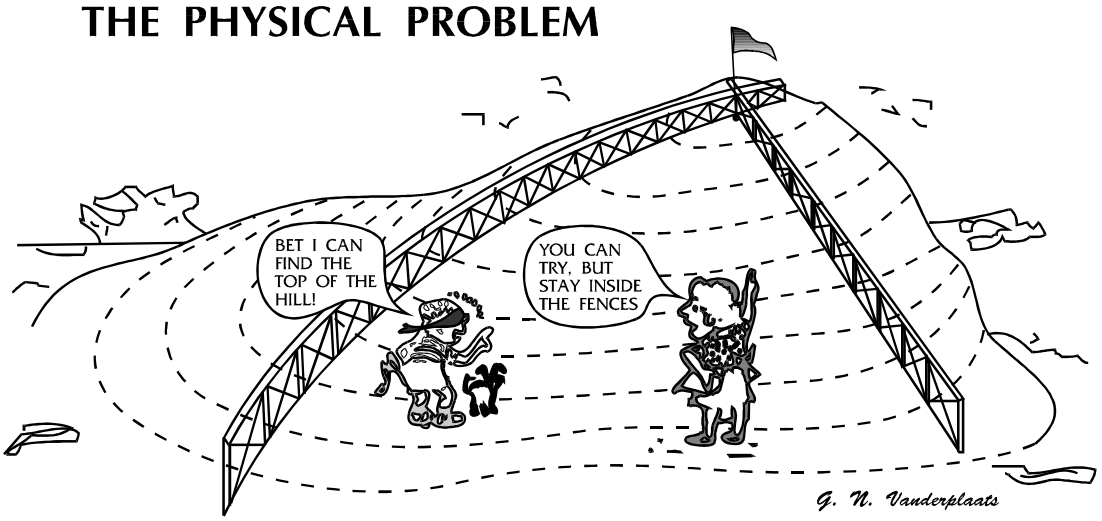
The DOT program does not directly deal with the equality constraints given in equation E-3. Instead, if such constraints are needed (which is seldom true for engineering design) it is only necessary to add additional inequality constraints of the form;

$$-g_j(\mathbf{X}) \le 0 \tag{E-5}$$

In other words, two equal and opposite inequality constraints will force the function to become an equality. Therefore, in this discussion, we will consider only inequality constraints of the form of equation E-2, remembering that equality constraints can be converted to a set of two opposite inequality constraints.

E.2.1 A Physical Example

Now consider how we may solve such a "design" problem. Imagine that you are standing on the side of a hill and are blindfolded. The objective function is your elevation on the hill, which you wish to maximize. Since we wish to think in terms of minimization, we will minimize the negative of the elevation so F(**X**) = -Elevation. Remembering this, we can define all mathematics here assuming we will minimize F(**X**). Also, you must stay inside of several fences on the hill. These represent the inequality constraints of equation E-2.



Mathematically, the negative of the distance from each fence is the amount by which you satisfy the constraint. If you are touching a fence, the constraint value is zero. Remember that you are blindfolded so you can't see the highest point on the hill that is inside the fences. You must somehow search for this point. One approach would be to take a small step in the north-south direction and another in the east-west direction and from that estimate the slope of the hill (assuming you are well inside of the fences). What you have done is to calculate the gradient of the objective function (-Elevation) by finite difference, defined as

$$\nabla F(\mathbf{X}) = \left\{ \begin{array}{c} \dfrac{F(\mathbf{X}+\delta X_1)-F(\mathbf{X})}{\delta X_1} \\ \dfrac{F(\mathbf{X}+\delta X_2)-F(\mathbf{X})}{\delta X_2} \\ \cdots \\ \cdots \\ \dfrac{F(\mathbf{X}+\delta X_N)-F(\mathbf{X})}{\delta X_N} \end{array} \right\} \qquad \text{(E-6)}$$

where the symbol ∇ is called the gradient operator. Equation E-6 is an approximation to the true mathematical gradient, which is the vector of partial derivatives of the function with respect to the independent design variables;

$$\nabla F(\mathbf{X}) = \begin{Bmatrix} \frac{\partial F(\mathbf{X})}{\partial X_1} \\ \frac{\partial F(\mathbf{X})}{\partial X_2} \\ \cdots \\ \cdots \\ \frac{\partial F(\mathbf{X})}{\partial X_N} \end{Bmatrix} \tag{E-7}$$

where, in general, F(**X**) can be the objective or any constraint function.

This is a vector direction. The slope is the direction you might chose to search since this will move you up the hill at the fastest rate. This we call the "search" direction. Mathematically, this gradient of the objective is referred to as a direction of "steepest ascent." Because we wish to minimize F(**X**), we would move in the negative gradient, or "steepest descent" direction. You can now move in this direction until you reach the crest of the hill or encounter a fence.

Note that the number of steps you take in this direction is a scalar parameter (partial steps are allowed so it will usually not be an integer number). We will call the number of steps in a given search direction α. Now define the point at which you started as $\mathbf{X}^0$. In this case, $\mathbf{X}^0$ contains two entries, being the longitude and latitude of your starting point. Assuming you choose to move straight up hill, you moved in a vector search direction we will call **S**. Also, this is the first iteration in the process of maximizing your elevation so it is iteration 1. In general we will use the letter "q" to indicate the iteration number. Finally, you moved in a steepest descent direction so, mathematically, $\mathbf{S} = -\nabla F(\mathbf{X})$. Remember that, because F(**X**) is the **negative** of the elevation, this is actually the steepest **ascent** direction up the hill.

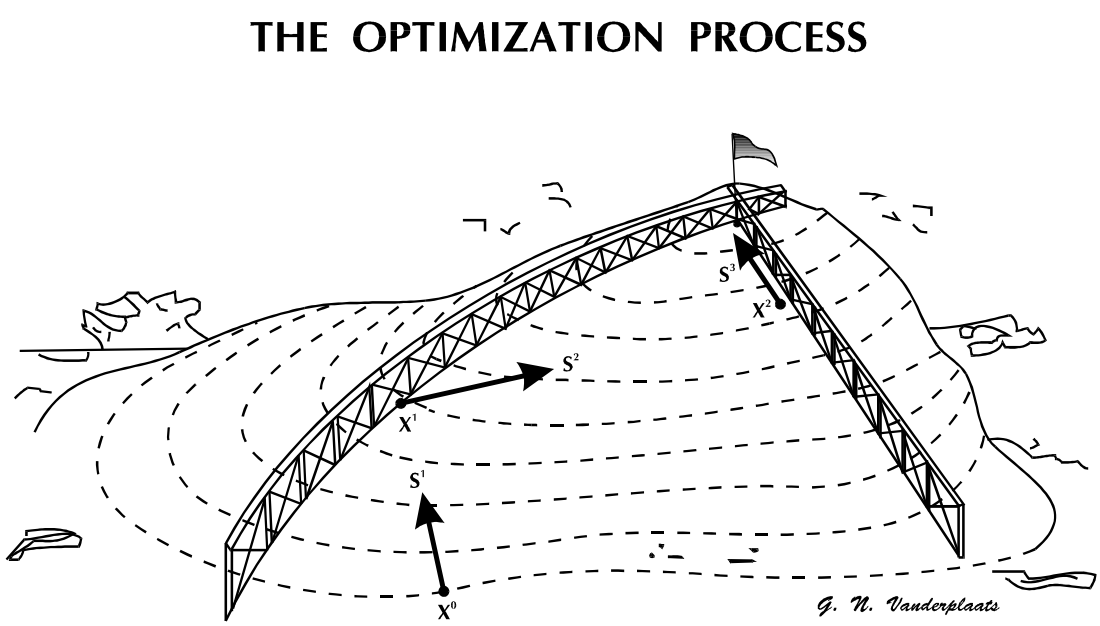
Since this is the first iteration, the direction you move is designated as $\mathbf{S}^1$. Upon encountering a fence or the crest of the hill in direction $\mathbf{S}^1$, we can update the description of your location on the hill by the simple mathematical expression;

$$\mathbf{X}^1 = \mathbf{X}^0 + \alpha^* \mathbf{S}^1 \tag{E-8}$$

where the asterisk on α denotes the "optimum" move parameter.

This completes the first iteration in the "design" process. If you are at the crest of the hill, we could just repeat the process of finding a new steepest descent direction and moving again. In practice we will see that, in this case, there is a better choice of directions, called a conjugate direction. However, the concept is the same. You find a search direction up the hill and move in that direction as far as possible.

Now, assume instead that you have encountered a fence, as shown in the figure below.



In order to make any further improvement in the "design," we must find a new search direction, $\mathbf{S}^2$, that will continue to increase your elevation on the hill, but will keep you inside of the fence. Here we seek a "usable-feasible" direction, where a usable direction is one that moves you up hill (decreases the objective function, F(**X**)) and a feasible direction is one that will keep you inside of the fence. The mathematical definition of a usable search direction is

$$\nabla F(\mathbf{X})^T \mathbf{S} \le 0 \qquad \text{(E-9)}$$

Equation E-9 is just the scalar product (dot product) of the gradient of the objective function with the search direction. This is the magnitude of $\nabla F(\mathbf{X})$ times the magnitude of **S** times the cosine of the angle between the two vectors. Thus, the cosine of the angle determines the sign of the product since the magnitudes are positive numbers. For the cosine to be zero or negative, the angle between the vectors must be between 90 and 270 degrees. If the cosine is zero, the search direction would be at an angle of 90 or 270 degrees from the gradient of the objective function. A move in this direction would follow a contour on the hill and (for a small move) would neither increase or decrease the function. If the cosine is -1.0, the direction would be opposite the direction of $\nabla F(\mathbf{X})$ and would be the direction of steepest descent (steepest ascent up the hill). Thus, we wish to find a search direction that will make the left hand side of equation E-9 as negative as possible. However, we must also require that this direction not move outside of an active constraint (fence). This is the feasibility requirement which is stated similar to the usability requirement, but now with respect to the critical constraint;

$$\nabla g_j(\mathbf{X})^T \mathbf{S} \le 0 \qquad \text{(E-10)}$$

Just as for the objective function, the angle between the search direction, **S**, and the gradient of the constraint must be between 90 and 270 degrees. If the angle is exactly 90 or 270 degrees, the search direction will be tangent to the constraint boundary (fence). To find the search direction that will make the greatest possible improvement in the objective function, but will follow or move inside of the fences, we combine the usability and feasibility requirements to create the following sub-optimization task; Find the components of the search direction, **S**, that will

$$\text{Minimize} \quad \nabla F(\mathbf{X})^T \mathbf{S} \tag{E-11}$$

Subject to;

$$\nabla g_j(\mathbf{X})^T \mathbf{S} \le 0 \qquad j \varepsilon J \tag{E-12}$$

$$\mathbf{S}^T \mathbf{S} \le 1 \tag{E-13}$$

where J is the set of constraints whose values are zero within some numerical tolerance. In other words, J includes any fences we are against, while those fences somewhere else do not matter because we can move towards them at least some distance without going outside. The purpose of equation E-13 is simply to prevent an unbounded solution to the problem defined by equations E-11 and E-12. In the case of a simple two variable problem, finding the appropriate search direction is quite easy and may be done graphically. In the more general case, where there are numerous design variables as well as several active constraints, this becomes a subproblem that is solved as part of the optimization. This sub-problem is linear in **S**, except for the quadratic constraint of equation E-13.

Assuming we can find a usable-feasible search direction, we can now search in this direction. If we are moving tangent to the fence, as is the case here, we must make corrections as we go to stay inside, because the fence is curved. That is, the constraint is mathematically nonlinear. We continue searching and correcting in this direction until we can make no further improvement. In the figure above, we actually moved away from the fence so we could move in a straight line. This is what the original feasible directions method does [18]. The modified method of feasible directions method used in DOT actually follows the curved fence [19]. The sub-problem of finding a new usable-feasible search direction is repeated, followed by continued search until no search direction can be found that will improve the design without violating one or more constraints. We call this point the “optimum.” In the present example, you began inside of the fences and have sequentially improved the design until the optimum is reached. In practice, you may start outside of one or more fences, in which case the initial design is called infeasible. One feature of the optimization algorithms used in DOT is that, if this is the case, we modify the algorithm to find a search direction back toward the feasible region, and then proceed from there.

The question now arises, How do we know that we have reached the optimum? The answer lies in what are known as the Kuhn-Tucker conditions [5]. In the case of an unconstrained problem [omit equations E-2-E-4], this simply is the condition where the gradient of the objective function vanishes (equals zero), as is well known to all undergraduate calculus students. In the case of the constrained optimization problem considered here, the conditions of optimality are more complex. Now the governing equation is the stationary condition of the Lagrangian;

$$L(\mathbf{X}, \lambda) = F(\mathbf{X}) + \sum_{j=1}^{M} \lambda_j g_j(\mathbf{X}) \tag{E-14}$$

The Kuhn-Tucker conditions dictate that the Lagrangian function, $L(\mathbf{X},\lambda)$, must have a vanishing gradient at the optimum design denoted by $\mathbf{X}^*$. However, we must additionally remember the original optimization problem and the inequality condition of equation E-2 [we have already agreed to omit equation E-3]. When all of these conditions are considered, it leads to the statement of the Kuhn-Tucker necessary conditions for optimality;

1. $\mathbf{X}^*$ is feasible $(\text{all} g_j(\mathbf{X}^*) \leq 0)$ (E-15)

2. The product of λ_j and $g_j(\mathbf{X}^*)$ equals zero

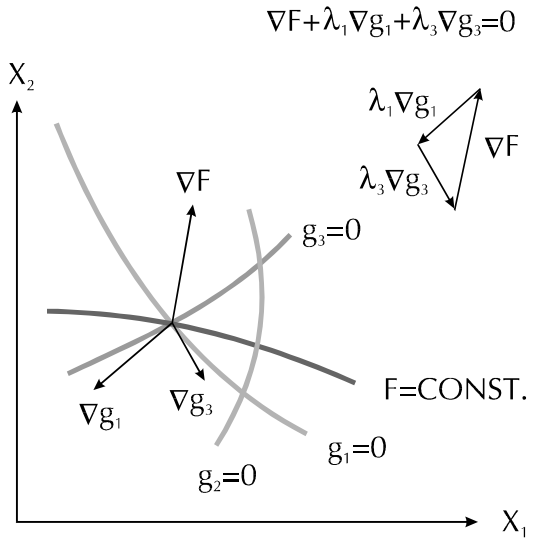
$$\lambda_j g_j = 0 \quad j = 1, M \tag{E-16}$$

3. The gradient of the Lagrangian vanishes.

$$\nabla F(\mathbf{X}) + \sum_{j=1}^{M} \lambda_j \nabla g_j(\mathbf{X}) = 0 \tag{E-17}$$

$$\lambda_j \geq 0 \quad j = 1, M \tag{E-18}$$

The physical interpretation of these conditions is that the sum of the gradient of the objective and the scalars, λ_j times the associated gradients of all active constraints must vectorially add to zero, as shown in the figure below. This is much like the statement that for equilibrium, the sum of all forces at a point must vectorially add to zero.



These are only necessary conditions and the definition here is actually a bit more restrictive that it need be in some cases. However, it provides the essential idea. In practice, it is difficult to find a design that precisely satisfies the Kuhn-Tucker conditions. Also, numerous designs may satisfy these conditions since there may be more than one constrained minimum. The importance of the Kuhn-Tucker conditions is that an understanding of the necessary conditions for optimality gives us some understanding of what is needed to achieve an optimum.

The simple example of climbing a hill by the approach outlined above is the physical interpretation of a class of optimization methods known as Feasible Direction techniques. A multitude of other numerical search algorithms are available to solve the general optimization problem [1]. Most of the more powerful methods update the design by the relationship identified above;

$$\mathbf{X}^q = \mathbf{X}^{q-1} + \alpha^* \mathbf{S}^q \qquad \text{(E-19)}$$

where q is the iteration number, $\mathbf{S}^q$ is a vector search direction and α^* is a scalar move parameter determining the amount of change in the design. Equation E-19 is the same as equation E-8 except now we have replaced the superscripts by the iteration counter, q. Just as in the example of searching up the hill, the product $\alpha^*\mathbf{S}$ is the design modification at the current step. Note that this is very similar to the traditional engineering approach of modifying a current best design, instead of trying a completely new (random?) design. An initial design $\mathbf{X}^0$ must be provided, but need not satisfy all of the constraints (indeed, one of the most powerful uses of optimization is to find a feasible solution to a complicated problem). In order to determine a search direction, $\mathbf{S}^q$, that will improve the design, gradients of the objective and critical constraints must

be supplied. Ideally, these are computed analytically. This dramatically increases the size of the problem that can be efficiently solved. Finally, a "one-dimensional search" is performed by trying several values of α and interpolating for the one that gives a minimum objective while satisfying the constraints. During this process, the objective and constraints must be repeatedly evaluated.

In the following sections we will define the overall optimization process and identify the steps taken to reach the optimum. The outline here will be necessarily brief and the reader is referred to reference 1 and the further references contained there for more details.

E.3 Unconstrained Minimization

DOT contains two algorithms for solving unconstrained minimization problems. Here a problem is defined as unconstrained if the set of constraints $g_j(\mathbf{X})$ does not exist. That is, NCON=0 in the calling statement to DOT. Lower and upper bounds can be imposed on the design variables, and these bounds will be respected.

Is assumed that we are provided with an objective function, $F(\mathbf{X})$ as well as lower and upper bounds on the design variables. Also, the gradient of the objective function is assumed to be available (DOT will calculate gradients by finite difference if necessary). Thus, we are solving the general problem defined by Equations E-1 and E-4, which are repeated here for easy reference;

Find the set of design variables, X_i, i=1,N, contained in vector $\mathbf{X}$, that will

$$\text{Minimize} \quad F(\mathbf{X}) \tag{E-20}$$

Subject to;

$$X_i^L \le X_i \le X_i^U \qquad i = 1,N \tag{E-21}$$

The optimizer does not know what kind of problem it is solving. It is simply minimizing a function. We are given an initial $\mathbf{X}$-vector, $\mathbf{X}^0$, and we will update the design according to equation E-19, which is also repeated here;

$$\mathbf{X}^q = \mathbf{X}^{q-1} + \alpha^*\mathbf{S}^q \tag{E-22}$$

The overall optimization process now proceeds in the following steps.

1. Start, q = 0, $\mathbf{X} = \mathbf{X}^0$
2. q = q + 1
3. Evaluate $F(\mathbf{X}^{q-1})$
4. Calculate $\nabla F(\mathbf{X}^{q-1})$
5. Determine a search direction, $\mathbf{S}^q$
6. Perform a one-dimensional search to find α^*
7. Set $\mathbf{X}^q = \mathbf{X}^{q-1} + \alpha^*\mathbf{S}^q$
8. Check for convergence to the optimum. If satisfied, exit. Otherwise go to step 2.

The critical parts of the optimization task consist of the following;

1. Find a usable search direction, $\mathbf{S}^q$
2. Find the scalar parameter, α^*, that will minimize $F(\mathbf{X}^{q-1} + \alpha\mathbf{S}^q)$.
3. Test for convergence to the optimum and terminate if convergence is achieved.

We will discuss each of these in turn.

E.3.1 Finding the Search Direction

The first step in finding the search direction is to calculate the gradient of the objective function, F(**X**). This may be provided by the user or (by default) will be calculated by DOT using finite difference methods.

On the first iteration, we use the steepest descent direction. Therefore, the search direction is simply

$$\mathbf{S}^q = -\nabla F(\mathbf{X}^{q-1}) \tag{E-23}$$

If we always use the negative gradient of the objective function for our search direction, it would be what is called the steepest descent method. In the steepest descent method, the search direction is always perpendicular to the previous direction. However, this method is notoriously inefficient and should never be used as a general algorithm.

We only use the steepest descent direction if this is the beginning of the optimization (q=1), or of the optimization progress indicates that our search direction is poor due to nonlinearity or numerical reasons.

On subsequent iterations, we will use the BFGS variable metric method if METHOD = 1 and the Fletcher-Reeves conjugate direction method if METHOD = 2 in the DOT calling statement.

The BFGS Search Direction

If METHOD=1 and NCON=0, the DOT program uses what is called the BFGS method [7-10] to determine the search direction.

The BFGS method is called a quasi-Newton method because it creates an approximation to the inverse of the Hessian matrix (matrix of second derivatives of the objective function). The matrix H contains the (inverse of) the second derivatives. Initially, H is set to the identity matrix, I. The search direction is defined as;

$$\mathbf{S}^q = -\mathbf{H}\nabla\mathbf{F}(\mathbf{X}^{q-1}) \tag{E-24}$$

so that the first search direction is the direction of steepest descent.

After the first iteration, H is updated using the following formula;

$$\mathbf{H}^{q+1} = \mathbf{H}^q + \mathbf{D}^q \tag{E-25}$$

where

$$\mathbf{D}^q = \left(\frac{\sigma+\tau}{\sigma^2}\right)pp^T - \frac{1}{\sigma}[\mathbf{H}^q yp^T + p(\mathbf{H}^q y)^T]$$

$$\sigma = p^T y$$

$$\tau = y^T \mathbf{H}^q y$$

The change vectors p and y are defined as;

$$\mathbf{p} = \mathbf{X}^q - \mathbf{X}^{q-1}$$

$$\mathbf{y} = \nabla \mathbf{F}(\mathbf{X}^q) - \nabla \mathbf{F}(\mathbf{X}^{q-1})$$

This method is considered to be theoretically best, but requires significant memory to store the H matrix (actually just the upper half, since H is symmetric). This method is considered to be less sensitive to the accuracy of the one-dimensional search than the Fletcher-Reeves method, though experience shows that the two methods are roughly comparable in efficiency and reliability as they are coded in DOT.

This method can be proven to converge in N or fewer iterations for strictly quadratic functions. However, since most optimization problems of interest are not quadratic, the method may have to be restarted by setting H to the identity matrix as the optimization progresses. This restarting is automatic in DOT.

The Fletcher-Reeves Search Direction

The Fletcher-Reeves conjugate directions method [6] was added to DOT Version 4. If METHOD=2 and NCON=0, this method is used. This method is very simple. We define the search direction as;

$$\mathbf{S}^q = -\nabla \mathbf{F}(\mathbf{X}^{q-1}) + \beta \mathbf{S}^{q-1} \tag{E-26}$$

where

$$\beta = \frac{\left|\nabla \mathbf{F}(\mathbf{X}^{q-1})\right|^2}{\left|\nabla \mathbf{F}(\mathbf{X}^{q-2})\right|^2} \tag{E-27}$$

The advantage to using the conjugate search direction is that it requires very little computer storage. Therefore, for large numbers of design variables, this method is preferred if computer memory is limited. This method can also be shown to converge in N or fewer iterations for truly quadratic problems and, in general, must be restarted with a steepest descent direction when the optimization progress slows down.

E.3.2 The One-Dimensional Search

Once we have chosen the search direction, $\mathbf{S}^q$, we search in this vector direction to find the value of α that will minimize $F(\mathbf{X}^{q-1} + \alpha \mathbf{S}^q)$. When no improvement can be found, a steepest descent direction will be attempted to see if this will improve the objective. If still no improvement can be found, the optimization is complete.

For a general description of the one-dimensional search process used in DOT, please see the following sections which describe the constrained optimization process. The only difference here is that no constraints are included in the one-dimensional search, except the lower and upper bounds on the design variables.

E.3.3 Convergence to the Optimum

A key part of the overall optimization process is deciding when to stop. DOT uses several criteria to make this decision. These include a maximum number of iterations, "reasonable" satisfaction of the Kuhn-Tucker conditions and dimensioning returns. These criteria are described in some detail in the following sections on constrained optimization algorithms.

E.3.4 A Simple Example

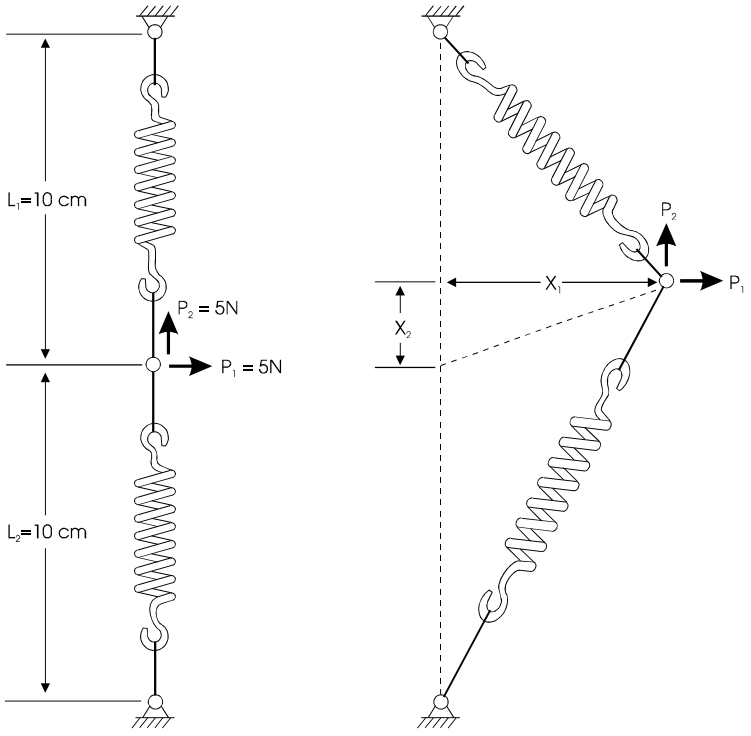
The figure below shows a simple spring for which we wish to find the equilibrium position under the applied loads. This is a nonlinear analysis problem and is the same as the example presented in Section 4.5. Loads P_1 and P_2 are applied to a two-spring system as shown. The equilibrium position is calculated by minimizing the total potential energy (PE).

The total potential energy of the system is defined as;

$$PE = \frac{1}{2}K_1\delta L_1^2 + \frac{1}{2}K_2\delta L_2^2 - P_1X_1 - P_2X_2$$

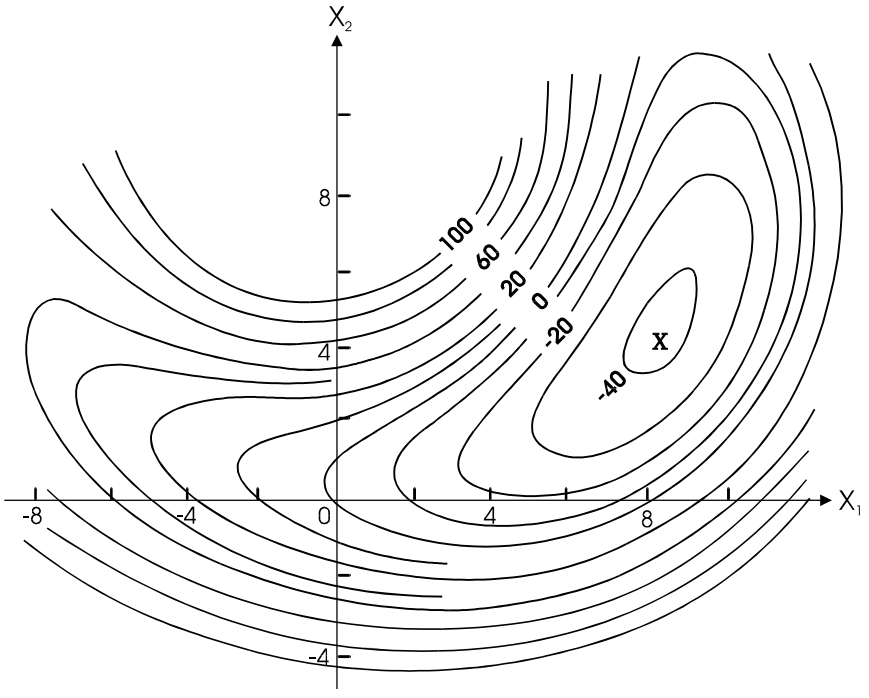
$$= \frac{1}{2}K_1[\sqrt{X_1^2 + (L_1 - X_2)^2} - L_1]^2 + \frac{1}{2}K_2[\sqrt{X_1^2 + (L_2 + X_2)^2} - L_2]^2 - P_1X_1 - P_2X_2$$

The design variables are X_1 and X_2.

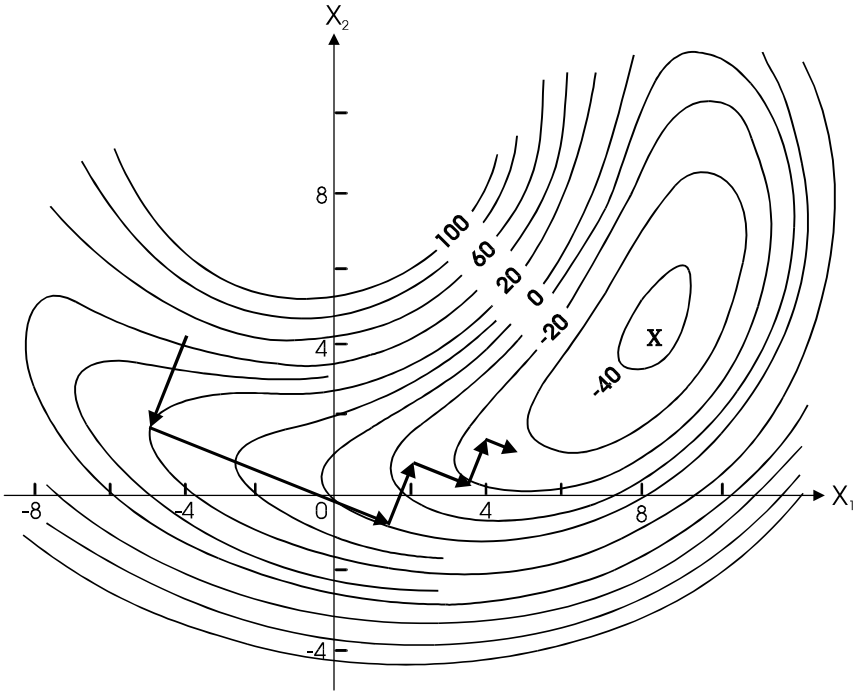


(a) Undeformed (b) Deformed

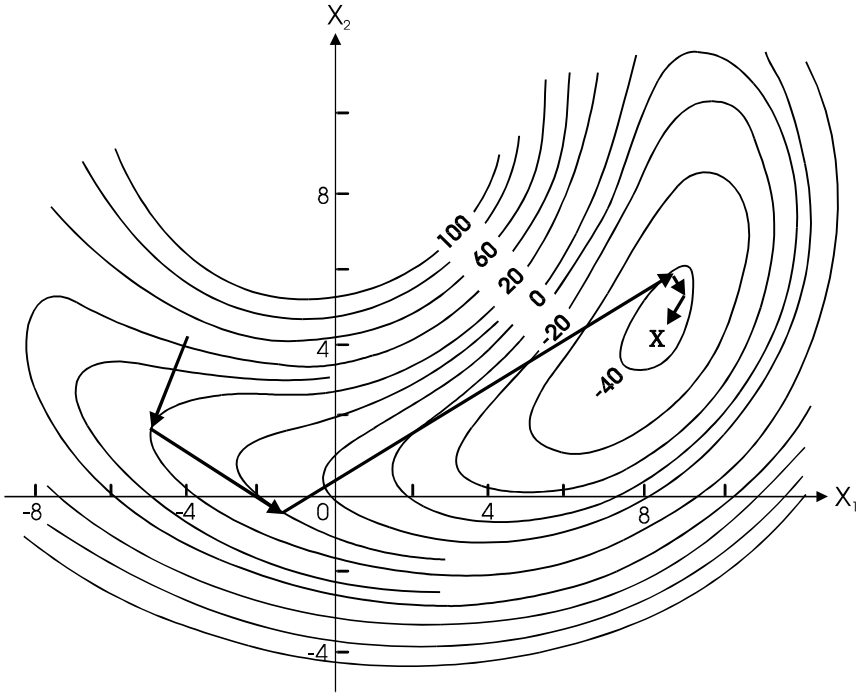
The figure below shows the "two-variable function space" for this problem, which is a series of contour lines for constant values of the objective function in terms of X_1 and X_2.



It was noted above that the steepest descent method is a poor method and should not be used. To demonstrate this, the figure below shows the iteration history using the steepest descent method. Only a few iterations are shown here since the method quickly deteriorates to make very small moves.



On the other hand, the BFGS and Fletcher-Reeves methods provide the iteration history shown below, which is clearly much more efficient.



E.4 Constrained Minimization

Three methods are available in DOT for solving the constrained minimization problem. These include the Modified Method of Feasible Directions (METHOD = 1), Sequential Linear Programming (METHOD = 2) and Sequential Quadratic Programming (METHOD = 3).

E.4.1 The Modified Feasible Directions Algorithm

Now we turn to the task of solving the constrained optimization problem. At this point it is assumed that we are provided with an objective function, F(**X**) and constraints, $g_j(\mathbf{X}) \le 0$, j = 1, M, as well as lower and upper bounds on the design variables. Also, gradients of the objective and constraints are assumed to be available. Thus, we are solving the general problem defined by Equations E-1, E-2 and E-4, which are repeated here for easy reference;

Find the set of design variables, X_i, i=1,N, contained in vector **X**, that will

$$\text{Minimize} \quad F(\mathbf{X}) \tag{E-28}$$

Subject to;

$$g_j(\mathbf{X}) \le 0 \qquad j = 1, M \tag{E-29}$$

$$X_i^L \le X_i \le X_i^U \qquad i = 1,N \tag{E-30}$$

The optimizer does not know what kind of problem it is solving. It is simply minimizing a function subject to inequality constraints. We are given an initial **X**-vector, $\mathbf{X}^0$, and we will update the design according to equation E-19, which is also repeated here;

$$\mathbf{X}^q = \mathbf{X}^{q-1} + \alpha^*\mathbf{S}^q \tag{E-31}$$

The overall optimization process now proceeds in the following steps.

1. Start, q = 0, $\mathbf{X} = \mathbf{X}^0$
2. q = q + 1
3. Evaluate $F(\mathbf{X}^{q-1})$ and $g_j(\mathbf{X}^{q-1})$, j = 1,M
4. Identify the set of critical constraints, J
5. Calculate $\nabla\mathbf{F}(\mathbf{X}^{q-1})$ and $\nabla\mathbf{g}_j(\mathbf{X}^{q-1})$ $j\varepsilon J$
6. Determine a search direction, $\mathbf{S}^q$
7. Perform a one-dimensional search to find α^*

8. Set $\mathbf{X}^q = \mathbf{X}^{q-1} + \alpha^* \mathbf{S}^q$
9. Check for convergence to the optimum. If satisfied, exit. Otherwise go to step 2.

The critical parts of the optimization task consist of the following;

1. Find a usable-feasible search direction, $\mathbf{S}^q$
2. Find the scalar parameter, α^*, that will minimize $F(\mathbf{X}^{q-1} + \alpha \mathbf{S}^q)$ subject to the constraints.
3. Test for convergence to the optimum and terminate if convergence is achieved.

We will discuss each of these in turn.

Finding the Search Direction

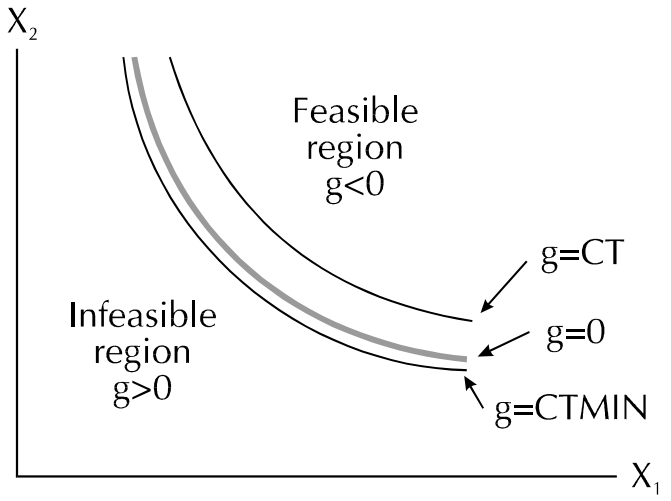
The first step in finding the search direction is to determine which constraints, if any, are active or violated. Here we define an active constraint as one with a value between CT and CTMIN, where CT is a small negative number and CTMIN is a small positive number. Remember that constraints must be negative to be feasible, so if $g_j(\mathbf{X})$ is less (more negative) than CT, it is not considered active. Also, when we approach a constraint boundary, it makes sense to begin "pushing" away from that "fence," so we initially choose a relatively large value for CT, say -0.03 (within three percent of being mathematically active). On the other hand, any constraint with a positive value is mathematically violated. However, trying to achieve a precise zero on a digital computer is not meaningful. Also, loads, material allowables, etc. are not really known precisely. Furthermore, the responses calculated by the analysis are usually only approximate by the nature modeling a real process. Therefore, we allow a small positive constraint value before we identify a constraint as violated. This is the value of CTMIN, typically taken as 0.005 (one-half percent violation). Thus, the governing definitions are;

$$g_j(\mathbf{X}) < \text{CT} \qquad \text{Inactive} \qquad \text{(E-32)}$$

$$\text{CT} \le g_j(\mathbf{X}) \le \text{CTMIN} \qquad \text{Active} \qquad \text{(E-33)}$$

$$g_j(\mathbf{X}) > \text{CTMIN} \qquad \text{Violated} \qquad \text{(E-34)}$$

This is shown graphically in the figure below.



This underscores the importance of normalizing constraints. There is no direct way for DOT to determine what a reasonable normalization factor might be. For example, assume a constraint is given in the following form;

$$g = A^2 - R \leq 0.0 \tag{E-35}$$

Now if the parameter R is of the order of 20,000 (such as a stress limit in a structure), the value of g would be the difference between two large numbers. For this constraint to be active, this difference must be between CT and CTMIN. In a numerical search process such as optimization, this would probably never happen. The net result would be considerable numerical ill-conditioning leading to poor or premature convergence. There is no way for the program to detect the order of magnitude of the components of this equation in advance, so the user must be responsible for this. A better form of the equation would be

$$g = \frac{A^2 - R}{20,000} \leq 0.0 \tag{E-36}$$

Alternatively, we could divide by the absolute value of R, but since R is probably a function of the design variables, this would increase the nonlinearity of the problem. The actual value of the scale factor is not critical (10,000 or 30,000 would do as well), but its order of magnitude is important. The function should be normalized in this fashion so that the order of magnitude of the result is between -1 and 1, or at least between -10 and 10.

Using the active constraint criteria, we first sort all constraints and identify those which are active or violated. Having done this, we calculate the gradient of the objective function and all active and violated constraints. Now we find a usable-feasible search direction. Here, there are three possibilities; 1) there are no active or violated constraints, 2) there are active constraints but no violated constraints and 3) there are one or more violated constraints. Each of these cases is handled differently.

No Active or Violated Constraints

Very often at the beginning of the optimization process there are no active or violated constraints. In this case the feasibility requirement is automatically met since we can move in any direction, at least a short distance, without violating any constraints. Thus, we only need to find a usable direction, being one which reduces the objective function. It does not have to be the steepest descent direction, but to start the process, this is the preferred choice. Therefore, the search direction is simply

$$\mathbf{S}^q = -\nabla F(\mathbf{X}^{q-1}) \tag{E-37}$$

However, we only use the steepest descent direction if this is the beginning of the optimization (q=1), or if the last iteration had active or violated constraints. In other words, we use a steepest descent direction for the first search direction at any time there are no active or violated constraints.

Now assume the last search direction was in the steepest descent direction and there are still no active constraints (if there were no violated constraints before, there will not be any now). We could again search in the steepest descent direction, and this is commonly done. Such a direction would be perpendicular to the previous direction. However, there are very solid theoretical reasons not to do this, and experience attests to the accuracy of the theory. Instead, we move in what is called a "conjugate" search direction, or more precisely, an A-conjugate direction, where A is the matrix of second partial derivatives of the objective function. We do not actually have the A matrix, but we have methods that offer a guaranteed convergence rate for problems where the objective is a quadratic function, whereas the successive use of steepest descent directions will probably never converge.

Here we use the Fletcher-Reeves conjugate direction method [6], described above, and the algorithm is very simple. We define the search direction as;

$$\mathbf{S}^q = -\nabla \mathbf{F}(\mathbf{X}^{q-1}) + \beta \mathbf{S}^{q-1} \tag{E-38}$$

where

$$\beta = \frac{\left|\nabla \mathbf{F}(\mathbf{X}^{q-1})\right|^2}{\left|\nabla \mathbf{F}(\mathbf{X}^{q-2})\right|^2} \tag{E-39}$$

The advantage to using the conjugate search direction is that it represents a very simple modification to the basic steepest descent algorithm, but provides a major improvement in efficiency. In the steepest descent method, the search direction is always perpendicular to the previous direction. On the other hand, when using conjugate directions, each search direction uses the steepest descent direction plus some fraction of the previous search direction. This makes physical sense. If I am making progress in direction $\mathbf{S}^1$, it is reasonable to bias direction $\mathbf{S}^2$ a bit in that direction, instead of simply using a steepest descent direction. This is exactly what Equation E-38 does.

This algorithm is extremely simple but it dramatically improves the rate of convergence to the optimum. Remember that for completely unconstrained functions (M = NCON = 0), the DOT program uses what is called the BFGS method [7-10] if METHOD = 1 and the Fletcher-Reeves method [6] if METHOD = 2.

Active Constraints, but no Violated Constraints

The usual search direction problem where the initial design is feasible and we have encountered a constraint is to find a search direction that will improve the design, but will move parallel to, or away from a constraint. This requires that we find a search direction, $\mathbf{S}^q$, that will reduce the objective function without violating any currently active constraints. This is stated mathematically as; Find the search direction, $\mathbf{S}^q$, that will

$$\text{Minimize } \nabla \mathbf{F}(\mathbf{X}^{q-1})^T \mathbf{S}^q \tag{E-40}$$

Subject to;

$$\nabla \mathbf{g}_j(\mathbf{X}^{q-1})^T \mathbf{S}^q \leq \quad j \varepsilon J \qquad \text{Feasibility Condition} \tag{E-41}$$

$$(\mathbf{S}^q)^T \mathbf{S}^q \leq 1 \qquad \text{Bounds on } \mathbf{S} \tag{E-42}$$

This is the same problem as was posed before for finding a usable-feasible search direction in the physical example of searching for the highest point on the hill.

Recall that the scalar product is the magnitude of the two vectors times the cosine of the angle between them. Thus, the objective of this sub-optimization problem is

$$\left|\nabla \mathbf{F}(\mathbf{X}^{q-1})\right| |\mathbf{S}| \cos\theta \tag{E-43}$$

Since we are minimizing this function, we will try to make the cosine of the angle between the two vectors as large negative as possible, but with the restriction of equation E-40. Alternatively, for any angle θ between 90 and 270 degrees, we can drive equation E-41 more negative by increasing the magnitude of $\mathbf{S}^q$. Also, if direction $\mathbf{S}^q$ satisfies the requirements of equation E-41, any increase in the magnitude of $\mathbf{S}$ will also satisfy this equation. Therefore, it is necessary to bound the magnitude of $\mathbf{S}$, and this is done by equation E-42.

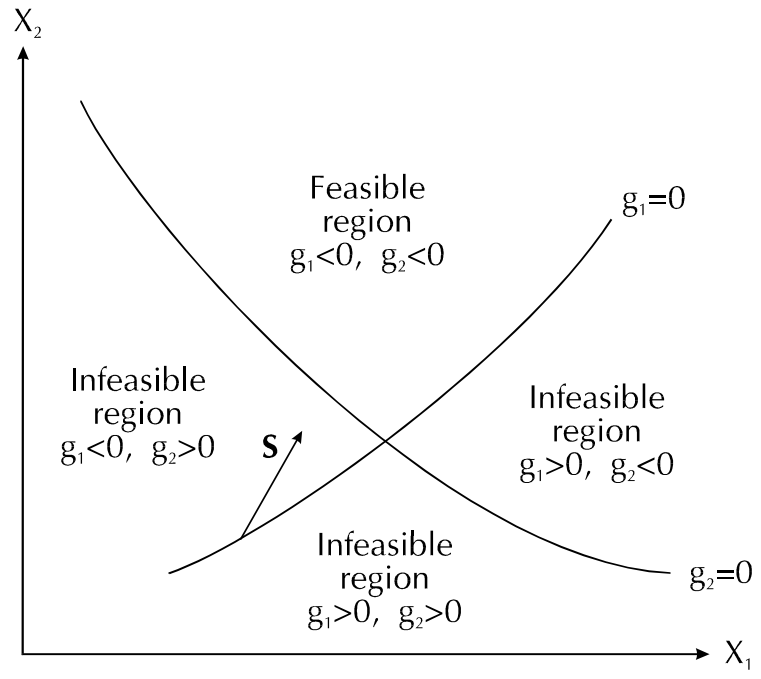
Assuming the resulting objective function from this sub-problem is negative, we will have satisfied the usability requirement. If we cannot drive the objective function defined by equation E-40 negative, then it follows that no direction exists that will reduce the objective function while staying inside of the constraints. This is the case where the Kuhn-Tucker conditions are satisfied and so the optimization process may be terminated.

In practice, the decision is not quite this simple because equation E-41 may include constraints that are within a tolerance of CT, but are not really critical. Therefore, we reduce CT and delete any constraints from the active set, J, that are not inside of the new tolerance. We then solve this direction finding problem again to see if we can move to a more precise satisfaction of the constraints. If the value of CT has been reduced in magnitude to CTMIN (CT is negative), and no negative objective can be found for the direction finding problem, we define this as satisfying the Kuhn-Tucker conditions and end the optimization process.

The question now becomes, How do we solve this sub-problem of finding the usable-feasible search direction, $\mathbf{S}^q$? The details are beyond the scope of this discussion. For our purposes it is sufficient to note that the problem can be converted to an iterative form of solving a set of simultaneous equations with some conditions on the variables. The actual algorithm is similar to that used in traditional linear programming. The details of this sub-optimization problem are given in reference 1.

One or More Violated Constraints

Now consider the case where one or more constraints are violated. Such a case is shown in the figure below, where constraint $g_1(\mathbf{X})$ is active and $g_2(\mathbf{X})$ is violated. Now we must find a search direction back toward the feasible region, even if it is necessary to increase the objective function to do so. To achieve this, we augment our direction finding problem of equations E-40-E-42 with a new variable, W, which we can call an artificial variable.



It has no direct physical significance to the problem, except as a measure of the constraint violation. The new direction finding problem is now; Find the search direction, $\mathbf{S}^q$, and artificial variable, W, that will

$$\text{Minimize } \nabla \mathbf{F}(\mathbf{X}^{q-1})^T \mathbf{S}^q - \Phi W \tag{E-44}$$

Subject to;

$$\nabla \mathbf{g}_j(\mathbf{X}^{q-1})^T \mathbf{S}^q + \theta_j W \leq \quad j \varepsilon J \tag{E-45}$$

$$(\mathbf{S}^q)^T \mathbf{S}^q + W^2 \leq 1 \tag{E-46}$$

For discussion, assume that the parameter, θ_j, in equation E-45 is equal to 1.0 and the parameter, Φ, is a very large positive number. Then the second term will dominate the minimizing of the function defined by equation E-44 so any increase in the variable, W, will drive the objective more and more negative (reduce the objective of the direction finding problem). However, for W to increase, the first term in equation E-44 must become more and more negative. Since $\mathbf{S}^q$ is bounded by equation E-46, the cosine of the angle between $\nabla \mathbf{g}_j(\mathbf{X}^{q-1})$ and $\mathbf{S}^q$ must be driven closer and closer to -1.0. For this to happen, $\mathbf{S}^q$ must point in a direction opposite to $\nabla \mathbf{g}_j(\mathbf{X}^{q-1})$. That is, $\mathbf{S}^q$ must point straight back toward the feasible region. Including the first term in equation E-44 is simply a means to reduce the true objective function if possible while we find a feasible design. It is actually the second term of the equation that controls the sub-optimization task.

Now consider the value of θ_j in equation E-45. This is referred to as a "push-off" factor, since its value determines how hard we push away from this violated constraint. If $\theta_j = 0.0$ even increasing W will not require that we move anywhere except tangent to the constraint, and this will probably not point us back to the feasible region. Therefore, some positive value is needed. In the DOT program, the philosophy is used that the more the constraint is violated, the harder we should push. Thus, θ_j is chosen using the following equation;

$$\theta_j = \theta_0 \left[1.0 - \frac{g_j(\mathbf{X}^{q-1})}{CT} \right]^2 \tag{E-47}$$

$$\theta_j \leq 50.0 \tag{E-48}$$

This is a quadratic function of the constraint value, chosen such that $\theta_j = 0.0$ at $g_j(\mathbf{X}^{q-1}) = CT$. That is, just as a constraint becomes active, we begin to include it in the direction finding process. But as the constraint becomes more critical, particularly as it becomes violated, we push harder and harder. We limit the value of θ_j to 50.0 as an arbitrary numerical limit based on the experience that this is a large enough value to drive the design back to the feasible region without causing too much numerical ill-conditioning in the algorithm.

Finally, consider the value of Φ. This parameter is initially chosen as a small number, say 5.0. Also, we always normalize $\nabla F(\mathbf{X}^{q-1})$ and $\nabla g_j(\mathbf{X}^{q-1})$ so that the first term in equations E-44 and E-45 is near unity. Thus, the second term in equation E-44 will dominate, but not too strongly, allowing for the possibility of reducing the objective function while overcoming the constraint violation. If this iteration does not overcome the violation, we will increase Φ by a factor of 10.0, but limit it to an upper bound of about 1000, again to avoid numerical ill-conditioning. Usually, this has the result of bringing the design back to the feasible region in a few iterations. If 20 iterations pass without overcoming the constraint violations, the optimization process is terminated on the assumption that no feasible solution exists.

The One-Dimensional Search

Having determined a usable-feasible search direction, the problem now becomes one of determining how far we can move in that direction. Again, we have a variety of possibilities to contend with, depending on the starting point, $\mathbf{X}^{q-1}$. However, in each case, we will make use of polynomial approximations to the objective and constraint functions. The basic concept is to try some initial value for α^* in equation E-46 and evaluate the corresponding objective and constraint functions. Therefore, the first question is to ask, What is a good first estimate for α^*? At the beginning of the optimization process, we have very little information available, except the function values and their derivatives with respect to α. In fact, at first look, it is not obvious that the derivative of the objective and constraints with respect to α is available. However, consider the objective function (the same algebra applies to constraints) and create a first order Maclaurin series approximation to $F(\alpha^*)$ in terms of α^*. Remember that

$$F(\mathbf{X}^q) = F(\mathbf{X}^{q-1} + \alpha^* \mathbf{S}^q) \tag{E-49}$$

Thus, an approximation to $F(\mathbf{X}^q)$ is

$$F(\mathbf{X}^q) \approx F(\mathbf{X}^{q-1}) + \sum_{i=1}^{N} \frac{\partial F(\mathbf{X}^{q-1})}{\partial X_i} \left\{ \frac{\partial X_i}{\partial \alpha^*} \right\} \alpha^* \tag{E-50}$$

or

$$F(\mathbf{X}^q) \approx F(\mathbf{X}^{q-1}) + \frac{dF(\mathbf{X}^{q-1})}{d\alpha^*}\alpha^* \tag{E-51}$$

But $\frac{\partial F(\mathbf{X}^{q-1})}{\partial X_i}$ is just the i-th entry into $\nabla \mathbf{F}(\mathbf{X}^{q-1})$. Also, from equation E-49,

$X_i^q = X_i^{q-1} + \alpha^* S_i^q$ so $\frac{\partial X_i}{\partial \alpha^*} = S_i$.

Therefore

$$\frac{dF(\mathbf{X}^{q-1})}{d\alpha^*} = \nabla \mathbf{F}(\mathbf{X}^{q-1})^T \mathbf{S}^q \tag{E-52}$$

Since both terms in equation E-52 are available, we have the slope of the function at $\alpha = 0$ for any function (objective or constraint) for which the gradient is available.

Now consider how we might use this information. First, since we are beginning the optimization process, we might optimistically expect to reduce the objective function by some fraction, say 10%. Now, using a simple linear approximation,

$$\begin{aligned} F(\mathbf{X}^q) &\approx F(\mathbf{X}^{q-1}) + \frac{dF(\mathbf{X}^{q-1})}{d\alpha^*}\alpha^* \\ &= F(\mathbf{X}^{q-1}) - 0.1\left|F(\mathbf{X}^{q-1})\right| \end{aligned} \tag{E-53}$$

from which we obtain a proposed α^*;

$$\alpha_{est}^* = \frac{-0.1\left|F(\mathbf{X}^{q-1})\right|}{\left[\frac{dF(\mathbf{X}^{q-1})}{d\alpha^*}\right]} \tag{E-54}$$

Now we have an estimate of α^* which will reduce the objective function by 10%. However, since we probably have the gradients of some constraints available, we can calculate other possible moves as well. Remember that equation E-52 applies to a constraint by simply substituting the constraint gradient for the objective gradient. Now assume we have some gradients for constraints that are not critical and we wish to estimate how far to move to make one of them critical. That is (instead of reducing its' value by 10% as in the case of the objective) we wish to drive $g_j(\mathbf{X}^q)$ to zero. Thus, a linear approximation to find $g_j(\mathbf{X}^q) = 0.0$ is

$$g_j(\mathbf{X}^q) \approx g_j(\mathbf{X}^{q-1}) + \frac{dg_j(\mathbf{X}^{q-1})}{d\alpha^*}\alpha^* = 0.0 \tag{E-55}$$

and an estimate for α^* is

$$\alpha_{est}^* = \frac{-g_j(\mathbf{X}^{q-1})}{\left[\frac{dg_j(\mathbf{X}^{q-1})}{d\alpha^*}\right]} \qquad \text{(E-56)}$$

Therefore, even at the beginning of the one-dimensional search, we have considerable information available to direct the process. Using the estimates for α^* given by equations E-54 and E-56 (for each non-critical constraint), we take the smallest positive proposed α^* as our first estimate of how far to move.

If constraints are currently violated, we can also use equation E-56. Applying this approximation to all violated constraints, we can pick the largest proposed value of α^* as an estimate of how far to move to overcome all constraint violations. If the projected move to a new constraint is smaller than this, then we would only move to the new active constraint as a first estimate.

Using similar approximations, we can estimate an upper bound on α^* such that all design variables will be driven to their lower or upper bounds, and this would provide a maximum value for α^* to be allowed.

While this brief discussion is not detailed enough to understand all of the intricacies of estimating the first step in the one-dimensional search, it does give a sense of the complexity of the decision processes. Also, it should be noted that on subsequent search directions, this will be modified to account for knowledge gained on previous iterations. For example, if we obtain an average of 5% improvement on the last two one-dimensional searches, we will attempt another 5% improvement on this one.

The one-dimensional search process now proceeds to first find bounds on the α^* that contain the solution and then interpolate within those bounds to find the constrained minimum in direction $\mathbf{S}^q$. Because we have chosen $\mathbf{S}^q$ as a direction of improving design, we can limit the search to positive values of α^*.

Finding Bounds on α^*

At the beginning of this process, we have the values of the objective and constraints at $\alpha = 0$ and $\alpha = \alpha_1$, where α_1 is our initial estimate for α^*. Now we have three cases to consider in deciding if α_1 is the upper bound, α_u.

Case 1 - All constraints are initially satisfied [all $g_j(\alpha=0) \le CTMIN$]

In this case, the search direction will be one that reduces the objective. Therefore, if at α_1, the objective is greater than at $\alpha_0(\alpha=0)$, we know that α_1 is an upper bound on α^*. Also, since the initial constraints are all satisfied, if any $g_j(\alpha_1) > CTMIN$, we know that α_1 is an upper bound because we have moved into the infeasible design space. If either the objective function has begun to increase or some constraint has become violated, α_1 is chosen as the upper bound, α_u.

Case 2 - One or more constraints are violated (greater than CTMIN) at $\alpha = 0$, and the objective function is increasing $\frac{dF(\mathbf{X}^{q-1})}{d\alpha^*} > 0$.

Here, we have chosen a search direction that should reduce the constraint violation. Thus, at α_1, if the maximum constraint value is greater than the maximum constraint value at $\alpha = 0$, we know that α_1 is the upper bound, α_u. This usually occurs when some new constraint becomes violated at α_1. If this happens, it is probably not possible to overcome the constraint violations in this direction, but we may at least reduce the maximum constraint violation. Alternatively, we may find a value of α_1 where all constraints are satisfied. If this happens, we have found a feasible design and so, again, α_1 is an upper bound on α^*. Here, we do not worry that the objective function is increasing since our first priority is to overcome the constraint violations. If we are able to overcome the constraint violations, we only wish to do so with a minimum increase in the objective function.

Case 3 - One or more constraints are violated (greater than CTMIN) at $\alpha = 0$, and the objective function is decreasing $\frac{dF(\mathbf{X}^{q-1})}{d\alpha^*} < 0$

This is the same as Case 2, except we are able to reduce the objective function. Therefore, everything is the same, except that we do not stop if we overcome the constraint violation(s). If we are able to find a feasible design and still reduce the objective function further, we will continue to move until some new constraint becomes active or the objective begins to increase. The corresponding α is the upper bound, α_u.

During the process of finding the upper bound, α_u, on α^* we may be required to move further than the initial step α_1. If α_1 does not yield an upper bound, we take another step, $\alpha_2 = 2\alpha_1$ and try again. In practice, it may take many steps before a bound is found on the solution. If this is the case, we retain the best four solutions, calling them α_L, α_1, α_2 and α_u. If more than three steps are required, the lower bound, α_L will not equal zero. Instead, we carry a lower bound forward to retain the nearest four

proposed values of α^*. If fewer than four values are available (including $\alpha = 0$), we retain them all. For each retained value of α, we also store the value of the objective and all constraint functions. These are now used to interpolate for an improved solution to the value of α^*.

Interpolation for α^*

Now that we have bounds on the solution to the one-dimensional search problem, it is desirable to refine the solution as much as we reasonably can. To achieve this, we use polynomial interpolation of the objective and constraint functions. The basic tool that we use here is a polynomial curve fit. This may be linear, quadratic, or cubic, depending on the amount of information available. If more than four pieces of information are available for a function of one variable, such as this, we could use higher order approximations. However, experience has shown that a cubic is adequate to approximate the functions without introducing too much numerical error.

Whether we are approximating the objective function or a constraint, the basic tool is the same. We will demonstrate the process using a quadratic fit to three function values. The other approximations are similar and are given in some detail in reference 1.

A quadratic polynomial is defined as

$$F = a_0 + a_1X + a_2X^2 \tag{E-57}$$

where a_0, a_1 and a_2 are constants and X is the variable. We use this nomenclature to be consistent with the literature. In the present case, X corresponds to the one-dimensional search parameter, α.

Now assume we have three values of F for three values of X, as given in the table below;

X	F
0.5	-1.25
1.0	-3.00
2.0	-2.00

Substituting the values of F and X into equation E-57, we get three equations in three unknowns;

$$a_0 + 0.5a_1 + 0.25a_2 = -1.25 \tag{E-58}$$

$$a_0 + 1.0a_1 + 1.00a_2 = -3.00 \tag{E-59}$$

$$a_0 + 2.0a_1 + 4.00a_2 = -2.00 \tag{E-60}$$

Solving for the values of a_0, a_1 and a_2 we get

$$a_0 = 2.0,\ a_1 = -8.0,\ a_2 = 3.0 \tag{E-61}$$

Substituting this into equation E-57,

$$F = 2.0 - 8.0X + 3.0X^2 \qquad \text{(E-62)}$$

Now, depending on whether the function, F, is the objective or a constraint, we will use equation E-62 in different ways. First, assume this is the objective function and we wish to find the value of X that will minimize it. The function, F, will have a minimum or maximum when its' derivative with respect to X is zero, so differentiating equation E-62,

$$dF/dX = -8.0 + 6.0X \qquad \text{(E-63)}$$

Setting this to zero and solving for X gives a proposed solution, X = 1.33. Substituting this into equation E-62 gives an estimated value of the function of F = -3.33. However, this is not sufficient to indicate that F is a minimum. We must also consider the second derivative which is

$$d^2F/dX^2 = 6.0 \qquad \text{(E-64)}$$

Since this is positive, we know that our proposed solution represents a minimum of F instead of a maximum.

Now, assume the function we are approximating is a constraint. In this case, we wish to find the value of X that will make the function equal zero (drive the design to a constraint boundary). To achieve this, we find the value of X that will make equation E-62 equal zero. Solving for this we get two values for X;

$$X = \frac{4.0 \pm \sqrt{10.0}}{3.0} = 0.2792, 2, 3874 \qquad \text{(E-65)}$$

For each of these values of X, the estimated value of F is zero. The question is, Which one do we choose? In this case, we note that the initial value (X=0) is positive. Therefore, X = 0.2792 is a proposed lower bound on α^* since this will overcome the constraint violation (to a quadratic approximation). Therefore, this is a candidate lower bound on α^*, α_L. However, we have more information available. The value of the variable, X = 2.3874 is the point at which the constraint will again become violated. Therefore, this value of X is a candidate upper bound on α^*, α_u. We can verify this by calculating the gradient of F at α_L and α_u to see that it is negative (driving g_j more negative) at α_L and positive (driving g_j violated) at α_u. To see that this is true, simply substitute the values of $X(\alpha)$ into equation E-63.

We now have an estimate of α^* for the minimum of the objective and the zero of one constraint. In this case, the constraint is initially violated so we have both a lower and upper bound on α^*. However, this is not the whole story. We must estimate values of α^* for all constraints. The maximum lower bound and the minimum upper bound will bracket the proposed solution to the one-dimensional search. It may happen that the

proposed upper bound is less than the proposed lower bound. If this happens, it usually means that for each proposed α^* there are one or more violated constraints. In this case, we use equation E-57 to minimize the maximum constraint violation and this decision over-rides other conditions.

Assuming we use the polynomial approximation for the objective and all constraints and we conclude with a lower and upper bound on the constraint values such that the upper bound is greater than the lower bound (if no constraints are violated at $\alpha = 0$, this will be the usual case), we must now chose what value of α to pick as α^*. If we start with violated constraints and the lower bound is less than the upper bound, we either pick the lower bound or we pick the value that will minimize the objective, assuming that is between the lower and upper bounds. If no constraints are violated at $\alpha = 0$, we pick the minimum proposed α^* between the one that minimized the objective and the upper bound.

In summary, as indicated by the difficulty in describing the one-dimensional search algorithm, this is a particularly complex part of the optimization process. The description given here is only the most simple outline of the process. In practice, we must make decisions based on how many proposed values of α^* are available, using linear, quadratic and cubic curve fits to estimate the best value of the move parameter. Because it is impossible to predict all possible combinations of situations that may arise, this is necessarily a process that is based on experience and intuition. The key is that we wish to consistently improve the design. If we do not get the absolute best solution during this one-dimensional search, we know that we will get another chance, until the convergence criteria are satisfied. This is why we do not use more complex methods (with their associated large number of function evaluations and cost) during the one-dimensional search. In the Modified Feasible Directions algorithm used in the DOT optimization program the one-dimensional search process is basically as outlined here but, as noted, is somewhat more complex.

At this point, we have determined the best search direction to improve the design and have searched in that direction. The question now is “How do we know if we have the optimum design?” The answer to this lies in the decisions on when to terminate the optimization process.

Convergence to the Optimum

Because optimization is an iterative process, one of the most critical and difficult tasks is determining when to stop. The DOT software uses several criteria to decide when to end the iterative search process, and these are described here.

Maximum Iterations

As with any iterative process, a maximum iteration counter is included. The default for this is 100 iterations (search directions). Usually, an optimum is found much sooner than this, so the maximum is mainly intended to avoid excessive computations.

No Feasible Solution

If the initial design is infeasible (constraints are violated) our first priority is to overcome this and find a feasible solution. However, if there are conflicting constraints, this may not be possible. Therefore, if a feasible design has not been achieved in 20 iterations, the optimization process is terminated.

During the optimization process the constraint tolerance, CT, is sequentially reduced. If CT is reduced in magnitude to CTMIN and either the absolute or relative convergence criteria given below is satisfied for ITRM iterations, the optimization will be terminated before 20 iterations.

Point of Diminishing Returns

Probably the most common situation is where the optimum is approached asymptotically. Therefore, while some progress is still being made, continued iterations are not justified. Here, two criteria are used. The first is that the relative change in the objective between iterations is less than a specified tolerance, DELOBJ. Thus, the criteria is satisfied if

$$\frac{\left|F(\mathbf{X}^q)-F(\mathbf{X}^{q-1})\right|}{\left|F(\mathbf{X}^{q-1})\right|} \le \text{DELOBJ} \tag{E-66}$$

The default value for DELOBJ is 0.001.

The second criteria is that the absolute change in the objective between iterations is less than a specified tolerance, DABOBJ. This criteria is satisfied if

$$\left|F(\mathbf{X}^q)-F(\mathbf{X}^{q-1})\right| \le \text{DABOBJ} \tag{E-67}$$

The default value for DABOBJ is the maximum of $0.001*|F(\mathbf{X}^0)|$ and 0.0001.

The reason for the two criteria is that if the objective function is large, the relative change between two successive iterations is an indication of convergence. However if $F(\mathbf{X})$ is a very small number, a relative change will not be meaningful and so the absolute change will control convergence.

Finally, it is recognized that no progress may be made on one iteration, only to make significant progress on the next. Therefore, we require that one of these criteria be satisfied on ITRMOP consecutive iterations, where the default value of ITRMOP is 2.

Satisfaction of the Kuhn-Tucker Conditions

The Kuhn-Tucker necessary conditions for optimality are given in Section E.2.1. In the uncommon case where we do not have any constraints, these conditions degenerate to the familiar case of the vanishing gradient of the objective. In practice, we say this is true if all components of the gradient are less than 0.001.

The usual optimization task is where many constraints are imposed. In the process of finding a usable-feasible search direction, we are able to detect if the Kuhn-Tucker conditions are satisfied. If they are, we should terminate the optimization process. However, in practice, this is not the whole story. Remember that we define a constraint as active if it is numerically less than the parameter CT. The optimization starts with

CT=-0.03 (any constraint within three percent of being satisfied is called critical). Because a constraint with this value will be considered in the Kuhn-Tucker calculations, these conditions may be satisfied even though we are up to three percent away from the constraint boundary. Therefore, we also check the value of CT. If it is larger than CTMIN in magnitude, we reduce its value by some fraction (typically 50%) and try again to find a usable-feasible direction. If we have reduced CT in magnitude to a value of CTMIN and still satisfy the Kuhn-Tucker conditions, we terminate the optimization process.

E.4.2 Sequential Linear Programming

The Sequential Linear Programming Method [11] was added to DOT Version 2. While this method is considered to be a "poor" method by many theoreticians, experience has shown that it can be very powerful when carefully coded.

The basic concept is quite simple. First, we create a Taylor Series approximation to the objective and constraint functions. Then, we use this approximation for optimization, instead of the original nonlinear functions. Now, when the optimizer requires the values of the objective and constraint functions, these are very easily and inexpensively calculated from the linear approximation. Also, since the approximate problem is linear, the gradients of the objective and constraints are available directly from the Taylor Series expansion.

Traditionally, the SIMPLEX algorithm[12] is used for solving the linear approximate problem. However, DOT uses the Modified Method of Feasible Directions, since it solves linear problems nicely and works well for the size of problems normally considered by DOT.

The general algorithm proceeds in the following steps:

1. For the current values of the design variables, sort the constraints and retain the most critical for use during this cycle. Typically, we may retain 5*NDV constraints. The reason that we do not retain all constraints is that there may be thousands, and most are far from critical. Therefore, it is wasted effort to calculate their gradients.
2. Create a first order Taylor Series expansion of the objective and retained constraints with respect to the design variables.
3. Define move limits on the design variables. Typically, during one cycle, the design variables will be allowed to change by 20-40%, but this is adjusted during later cycles.
4. Solve the linear approximate optimization problem.
5. Check for convergence. If satisfied, EXIT. Otherwise, repeat the process from step 1.

In step 1, DOT creates the Taylor Series expansion in the form;

$$\tilde{F}(\mathbf{X}) = F(\mathbf{X}^{q-1}) + \nabla \mathbf{F}(\mathbf{X}^{q-1})^{T} \delta \mathbf{X} \qquad \text{(E-68)}$$

$$\tilde{g}_j(\mathbf{X}) = g_j(\mathbf{X}^{q-1}) + \nabla \mathbf{g}_j(\mathbf{X}^{q-1})^T \delta \mathbf{X} \quad j \varepsilon J \tag{E-69}$$

where

$$\delta \mathbf{X} = \mathbf{X}^q - \mathbf{X}^{q-1} \tag{E-70}$$

and J is the set of retained constraints.

Note that everything in Equations E-68 and E-69 is constant except the values of the design variables, $\mathbf{X}^q$. Therefore, we can rewrite these equations as;

$$\tilde{F}(\mathbf{X}) = \tilde{F}^0 + \nabla \mathbf{F}(\mathbf{X}^{q-1})^T \delta \mathbf{X} \tag{E-71}$$

$$\tilde{g}_j(\mathbf{X}) = \tilde{g}_j^0 + \nabla \mathbf{g}_j(\mathbf{X}^{q-1})^T \delta \mathbf{X} \quad j \varepsilon J \tag{E-72}$$

where

$$\tilde{F}^0 = F(\mathbf{X}^{q-1}) - \nabla \mathbf{F}(\mathbf{X}^{q-1})^T \delta \mathbf{X}^{q-1} \tag{E-73}$$

and

$$\tilde{g}_j^0 = g_j(\mathbf{X}^{q-1}) - \nabla \mathbf{g}_j(\mathbf{X}^{q-1})^T \delta \mathbf{X}^{q-1} \quad j \varepsilon J \tag{E-74}$$

We now solve the linear approximate optimization problem;

$$\text{Minimize } \tilde{F}(\mathbf{X}^q) \tag{E-75}$$

Subject to;

$$\tilde{g}_j(\mathbf{X}^q) \le 0 \quad j \varepsilon J \tag{E-76}$$

$$X_i^L \le X_i \le X_i^U \quad i = 1, N \tag{E-77}$$

where

$$\tilde{X}_i^L = X_i^q - D\left|X_i^q\right| \tag{E-78}$$

and

$$\tilde{X}_i^U = X_i^q + D\left|X_i^q\right| \tag{E-79}$$

The multiplier D in Equations E-78 and E-79 is initially set to RMVLMZ (contained in the RPRM array). Depending on the progress of the optimization, this parameter will be sequentially reduced. Typically, the first approximate optimization produces a design that violates one or several constraints. However, as the optimization proceeds,

this constraint violation should be reduced. If it actually increases, we reduce the parameter, D. Actually, the move limit strategy in DOT is more complex than this simple description. In practice, move limits are individually adjusted, either up or down, on each design variable [17].

The convergence criteria for the SLP method are much the same as for the other methods, with the addition that the process will not terminate if there are significant constraint violations. In this case, we will continue until the constraints are satisfied or the maximum number of iterations is reached.

E.4.3 Sequential Quadratic Programming

The Sequential Quadratic Programming Method [13, 14] was added to DOT Version 3. This method is considered to be an excellent method by many theoreticians.

The basic concept is very similar to Sequential Linear Programming. First, we create a Taylor Series approximation to the objective and constraint functions. However, instead of minimizing the linearized objective, we create a quadratic approximate objective function. We use the linearized constraints with this to create a direction finding problem of the form;

$$\text{Minimize } Q(\mathbf{S}) = F^0 + \nabla \mathbf{F}^T \mathbf{S} + \frac{1}{2}\mathbf{S}^T \mathbf{B}\mathbf{S} \tag{E-80}$$

Subject to;

$$(\nabla \mathbf{g}_j)^T \mathbf{S} + g_j^0 \le 0 \quad j = 1,M \tag{E-81}$$

This sub-problem is solved using the Modified Method of Feasible Directions. The matrix B is a positive definite matrix which is initially the identity matrix. On subsequent iterations, B is updated to approach the Hessian of the Lagrangian function.

Now assume we have solved the approximate problem of minimizing Q subject to the linearized constraints. At the optimum for this problem, we calculate the Lagrange multipliers, $\lambda_j \quad j = 1,M$. We now search in direction **S** using the approximate

Lagrangian function. That is, we find α to

$$\text{Minimize } \Phi = F(\mathbf{X}) + \sum_{j=1}^{M} u_j \max[0, g_j(\mathbf{X})] \tag{E-82}$$

where

$$\mathbf{X} = \mathbf{X}^{q-1} + \alpha \mathbf{S} \tag{E-83}$$

$$u_j = |\lambda_j| \quad j = 1, M \text{ first iteration} \tag{E-84}$$

$$u_j = \max\left[|\lambda_j|, \frac{1}{2}(u'_j + |\lambda_j|)\right] \quad j = 1, M \text{ subsequent iterations} \tag{E-85}$$

and $u'_j = u_j$ from the previous iteration.

In DOT, during the one-dimensional search, approximations are made to the components of Φ because this function has discontinuous derivatives at the constraint boundaries, but the components are smooth.

After the one-dimensional search is complete, the B matrix is updated using the BFGS formula;

$$\mathbf{B}^* = \mathbf{B} - \frac{\mathbf{B}\mathbf{p}\mathbf{p}^T\mathbf{B}}{\mathbf{p}^T\mathbf{B}\mathbf{p}} + \frac{\eta\eta^T}{\mathbf{p}^T\eta} \tag{E-86}$$

where

$$\mathbf{p} = \mathbf{X}^q - \mathbf{X}^{q-1} \tag{E-87}$$

$$\eta = \theta\mathbf{y} + (1-\theta)\mathbf{B}\mathbf{p} \tag{E-88}$$

$$\mathbf{y} = \nabla_X\Phi^q - \nabla_X\Phi^{q-1} \tag{E-89}$$

$$\Phi = F(\mathbf{X}) + \sum_{j=1}^{M} \lambda_j g_j(\mathbf{X})) \tag{E-90}$$

$$\theta = \begin{cases} 1.0 & \text{if } \mathbf{p}^T\mathbf{y} \geq 0.2\mathbf{p}^T\mathbf{B}\mathbf{p} \\ \dfrac{0.8\mathbf{p}^T\mathbf{B}\mathbf{p}}{\mathbf{p}^T\mathbf{B}\mathbf{p} - \mathbf{p}^T\mathbf{y}} & \text{if } \mathbf{p}^T\mathbf{y} < 0.2\mathbf{p}^T\mathbf{B}\mathbf{p} \end{cases} \tag{E-91}$$

Now $\mathbf{B}^*$ replaces $\mathbf{B}$ and a new iteration is begun.

E.5 Summary

The purpose of this Appendix has been to provide a brief overview of the computational details of the optimization process. The techniques used in the DOT program have been chosen because they are reasonably robust and because they allow for considerable flexibility in formulating the design problem.

From the variety of decisions that must be made during the optimization process it is clear that anything the user can do to improve the numerical conditioning of the problem will have a strong influence on the efficiency and reliability of the result. It is hoped that some understanding of the actual optimization process will assist in this.

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V