UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

A User-Friendly Wrapper for DSIDES (Decision Support in the Design of Engineering Systems)

A THESIS

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirements for the

Degree of

MASTER OF SCIENCE

By

SARA HAJIHASHEMI

Norman, Oklahoma

2023

A User-Friendly Wrapper for DSIDES
(Decision Support in the Design of Engineering Systems)

A THESIS APPROVED FOR THE

SCHOOL OF INDUSTRIAL AND SYSTEMS ENGINEERING

BY THE COMMITTEE CONSISTING OF

I extend my heartfelt gratitude to my parents for being my constant motivation and to my supportive husband and children for their unwavering support in my pursuit of graduate school. Their encouragement and belief in me have been invaluable throughout my academic journey.

Acknowledgements

I express my heartfelt gratitude to several individuals who have played instrumental role in shaping my academic journey:

Professors Farrokh Mistree and Janet K. Allen have been unwavering pillars of support and guidance throughout my master’s program. Their mentorship and profound wisdom have been invaluable in shaping my academic and research pursuits. Their unwavering belief in my abilities, their nurturing guidance, and their encouragement have been instrumental in my academic growth. They have not only imparted their knowledge but also instilled in me the art of research, mentorship, and the invaluable skill of critical thinking. Their recognition of my latent potential and their relentless efforts to help me improve and acquire competencies have had a transformative impact on my academic journey.

Siavash Mandegari and Dr. Rajlakshmi Nayak have been essential in validating the effectiveness and usability of my Wrapper. Their support and contributions have been pivotal in ensuring the success of this project.

I extend special thanks to Gehendra Sharma, Mathew Baby, and Dr. Lin Guo for providing problems that allowed me to rigorously test the functionality of the Wrapper.

Your collective contributions have been indispensable to my academic success, and I am deeply appreciative of your guidance, mentorship, and unwavering support.

I gratefully acknowledge the financial support that I received from the :John and Mary Moore Chair and the L.A. Comp Chair held by Professors Allen and Mistree, respectively.

Finally, and most importantly, I thank my family members, who mean everything to me.

Table of Contents

Contents

List of Figures

List of Tables

Abstract:

When dealing with complex systems, we need to consider that these systems have behaviors that are hard to predict or control, and uncertainties are always present since computational models are abstracts of reality. It is recognized that in many situations, it may not be possible to simultaneously optimize all objectives due to inherent conflicts, resource limitations, or uncertainty. As George E.P. Box said: "All models are wrong, but some are useful." The consequences of these observations are significant. We need to accept that our models might not capture everything and that uncertainties are a part of the picture. Hence, we must accept and deal with uncertainty instead of ignoring it and find solutions that are relatively insensitive to the uncertainties.

When choosing a method to work with, we need to consider the quality of our data. To make this all work, we need a method to find solutions that achieve a reasonable compromise or balance among the objectives and identify a set of solutions that are relatively insensitive to uncertainties. Also, be able to facilitate the exploration of solution space to support human decision-making. This ties into the problems we face in supporting decisions for complex systems. These problems involve choosing between options and making compromises.

The compromise Decision Support Problem (cDSP) construct and the Adaptive Linear Programming algorithm has been developed as a result, which was first introduced by Mistree and co-authors (1993). It is a domain-independent, multiobjective decision model based on mathematical and goal programming. They effectively deal with multiobjective problems involving bounds, linear and nonlinear constraints, goals, and consisting of Boolean and continuous variables. The requirements for this construct are:

1) Identify a set of solutions that are relatively insensitive to uncertainties

2) Facilitate the exploration of solution space to support human decision-making

Mistree and co-authors also designed a computer program to implement cDSP construct. It has been written in FORTRAN to identify robust satisficing solutions to design problems when the models are abstractions of reality. It is called DSIDES (Decision Support in the Design of Engineering Systems).

DSIDES is a software tool developed to help engineers and designers make better decisions in the design of complex engineering systems and provides decision support for the design of complex engineering systems.

In this thesis, our primary objective is to enhance the accessibility and user-friendliness of DSIDES by designing a user-friendly wrapper. Three key areas of focus are included in this thesis:

1) **Exploration of cDSP Construct:** In this part, the examination of the cDSP (Compromise Decision Support Problem) construct, including its structural components and the formulation of problem statements within the cDSP framework, has been discussed.
2) **Comprehensive Analysis of the DSIDES Wrapper**: A detailed exploration of the DSIDES wrapper and a step-by-step walkthrough of the wrapper's functionalities are covered.
3) **DSIDES Software Program Manuals:** Program manuals for the DSIDES software has been created. These manuals are helpful resources for individuals seeking to enhance, expand, or modify the software.

Based on these key areas of focus, there are three different parts to this thesis:

1) **Part One: DSIDES Software and cDSP Construct: An Introduction.**
2) **Part Two: Designing the User-Friendly Wrapper for DSIDES.**

3) **Part Three: Program Manuals and Improvement of DSIDES.**

In the following sections, all three parts and their related details are discussed, respectively.

A User-Friendly Wrapper for DSIDES (Decision Support in the Design of Engineering Systems)

Part One: DSIDES Software and cDSP Construct: An Introduction

I.1 Overview of Part 1

In the first part of this thesis, enhanced information about the compromise Decision Support Problem (cDSP) and formulation of a problem in cDSP, including Archimedean and preemptive forms in detail with some examples, a short description about DSIDES software and platform of DSIDES are provided. Upon reading the first part, the reader will have learned the necessary information to formulate a problem in cDSP, allowing them to start using the software effectively.

I.2 On the Realization of Complex Systems:

When dealing with complex systems, it is important to consider that these systems have behaviors that are difficult to predict or control. Additionally, uncertainties are always present since computational models are abstracts of reality. It is recognized that in many situations, it may not be possible to simultaneously optimize all objectives due to inherent conflicts, resource limitations, or uncertainty. Also, as George E.P. Box said: "All models are wrong, but some are useful." The consequences of these observations are significant. We need to accept that our models might not capture everything and that uncertainties are a part of the picture. When choosing a method to work with, it is important to consider the quality of our data. To make this all work, we need a method to find solutions that achieve a reasonable compromise or balance among the objectives and identify a set of solutions that are relatively insensitive to uncertainties. Also, be able to facilitate the exploration of solution space to support human decision-making. This ties into the problems when faced with supporting decisions for complex systems. These problems involve choosing between options and making compromises.

The compromise Decision Support Problem (cDSP) construct and the Adaptive Linear Programming algorithm were developed by Mistree and co-authors (1993). It is a domain-independent, multiobjective decision model based on mathematical and goal programming. These tools effectively deal with multiobjective problems involving bounds, linear and nonlinear constraints, goals, and consisting of Boolean and continuous variables.

In the next section, the compromise decision support problem (cDSP) construct will be discussed in detail to learn more about it and understand how to formulate the problem statement in the cDSP construct to find solutions that are relatively insensitive to uncertainty. In Part Two of this thesis, in Sections II.4.1, II.4.2, and II.4.3, three different examples are used that start from the problem statement and are followed by steps to convert it to the cDSP construct and, finally, how to implement them in the user-friendly wrapper. They could be good sources to practice and ensure learning this construct.

I.3 A Brief History of The Compromise Decision Support Problem (cDSP) Construct

In 1981, Mistree, Hughes, and Phuoc presented an algorithm titled SLIP2 (Sequential Linear Programming 2nd Generation). Since then, this stand-alone version has been significantly improved and is now a major component of the DSIDES (Decision Support In the Design of Engineering Systems) system. The SLIP2 algorithm was extended to solve multilevel, hierarchical problems and was then called SLIPML (Sequential Linear Programming Multilevel). Refinements to the SLIPML resulted in the Adaptive Linear Programming (ALP) algorithm and the compromise

Decision Support Problem formulation. Multiobjective problems involving bounds, both linear and nonlinear constraints and goals, and consisting of Boolean and continuous variables are now effectively dealt with by the compromised DSP. Goal Programming and Sequential Linear Programming respectively are the basis for the compromise Decision Support Problem and the Adaptive Linear Programming algorithm for multiobjective decision support problems.

The Compromise DSP involves the improvement of an alternative through modification. It is a class of decision problems in which multiple conflicting objectives or criteria must be considered simultaneously to find satisfactory compromise solutions in engineering design. Compromise DSP is taken into account when there are trade-offs between different design objectives or performance measures, and a decision-maker seeks to balance them. In other words, compromise DSP is used to determine values of design variables that satisfy a set of constraints and simultaneously achieve, as well as possible, a trade-off between a set of conflicting goals.

The goal is to assist decision-makers in exploring the design space, understanding the trade-offs between conflicting objectives, and ultimately selecting compromise solutions that align with their preferences and constraints.

It is worth noting that the cDSP construct is closely related to the field of multiobjective decision support problems and decision-making under uncertainty. The need to consider multiple objectives, manage conflicts, and handle uncertainty in decision-making is recognized by cDSP and applied in various domains.

In the following section, the process of formulating the problem into the cDSP construct will be discussed.

I.4 Formulating Compromise Decision Support Problems[1]

The preceding formulation of a compromise DSP is a hybrid formulation in that it incorporates concepts from traditional mathematical programming and goal programming (GP) and uses some new ones. It is similar to goal programming in that the multiple objectives are transformed into system goals (involving both system and deviation variables), and the deviation function is solely a function of the goal deviation variables. (This is in contrast to traditional mathematical programming, where multiple objectives are modeled as a weighted function of the system variables only.) However, the concept of system constraints is retained from the traditional constrained optimization formulation. Special emphasis is placed on the bounds of the system variables, unlike in traditional mathematical programming and goal programming. In effect, the traditional formulation is a subset of the compromise DSP - an indication of the generality of the compromise formulation.

The Compromise DSP - involves improvement of an alternative through modification. It is stated in words as follows:

Given

- An alternative that is to be improved through modification. Assumptions used to model the domain of interest.
- The system parameters.
- The goals for the design.

[1] Learning How to Design: A Minds-On, Hands-On, Decision-Based Approach.
Farrokh Mistree, Janet K. Allen, Harshavardhan Karandikar, Jon A. Shupe, and Eduardo Bascaran, 1995. It is available on Research Gate
https://www.researchgate.net/publication/2818887_Learning_How_to_Design_A_Minds-On_Hands-On_Decision-Based_Approach

Find

- The values of the independent system variables (they describe the attributes of an artifact).
- The values of the deviation variables (they indicate the extent to which the goals are achieved).

Satisfy

- The system constraints that must be satisfied for the solution to be feasible.
- The system goals that must achieve a specified target value as much as possible.

Bounds

- The lower and upper bounds on the system variables.

Minimize

- The deviation function which is a measure of the deviation of the system performance from that implied by the set of goals and their associated priority levels or relative weights.

The preceding formulation of a compromise DSP is a hybrid formulation in that it incorporates concepts from both traditional mathematical programming and goal programming (GP) and makes use of some new ones. It is similar to goal programming in that the multiple objectives are transformed into system goals (involving both system and deviation variables) and the deviation function is solely a function of the goal deviation variables. (This is in contrast to traditional mathematical programming where multiple objectives are modeled as a weighted function of the system variables only.) The concept of system constraints, however, is retained from the traditional constrained optimization formulation. Special emphasis is placed on the bounds on the system variables unlike in traditional mathematical programming and goal programming. In effect the traditional formulation is a subset of the compromise DSP - an indication of the generality of the compromise formulation.

I.5 Descriptors of the Compromise DSP Formulation

System descriptors are used to define a compromise DSP.

Parameters - are used to complete the modeling of the compromise DSP. For example, in the case of the design of a structure, the material properties are invariably treated as parameters, that is, their values are needed to enable solution but they are not affected by the solution process itself. Parameters are sometimes called "fixed variables".

Variables

- ❑ System variables.
- ❑ Deviation variables.

System constraints

(Equivalent to rigid goals in the GP formulation).

System goals

(Equivalent to soft goals in the GP formulation).

Bounds

- ❑ On system variables (formulated as rigid goals in the GP formulation).

Deviation function

In this section, the system descriptors for a compromise DSP are described. The descriptors are illustrated in Figure I.1 for a two dimensional compromise DSP.

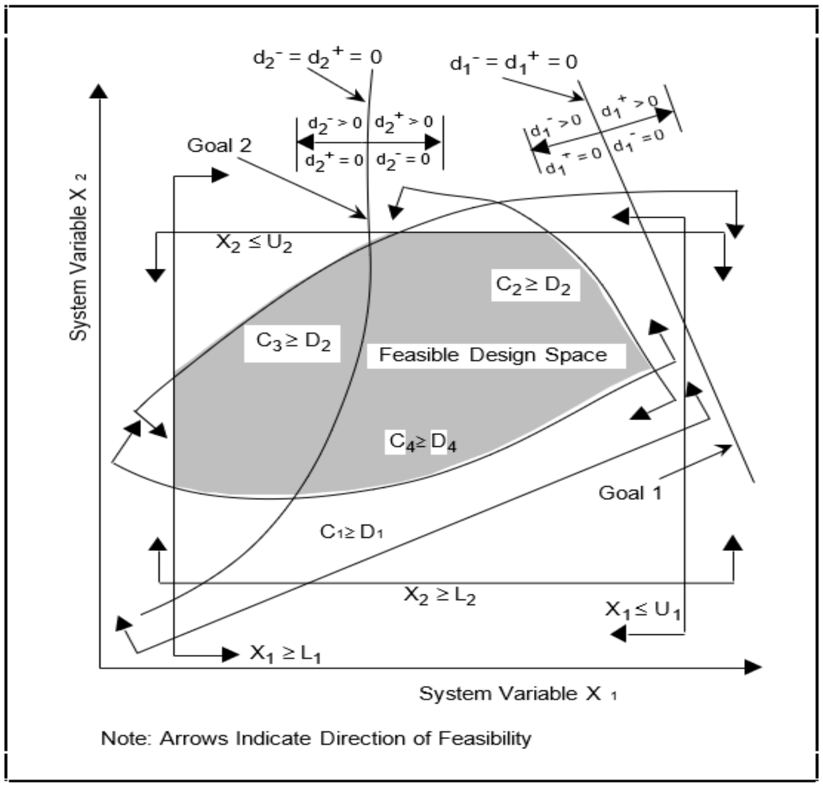


Figure I. 1: Typical Design Space For A Two Variable Compromise DSP

I.5.1 System Variables and System Constraints

System Variables

$$X = (X_1, X_2, ..., X_n), \ X_i \geq 0.$$

System Constraints

$$C_i(X) \ \geq, \text{or} = D_i(X); \ i = 1, 2, 3 ..., m.$$

Compromise DSPs have a minimum of two system variables. Consider a set of 'n' design variables represented by *X*. The vector of variables includes continuous variables and Boolean variables (1 if TRUE, 0 if FALSE). System variables are, by their nature, independent of the other descriptors and can be changed as required by the designer to alter the state of the system.

System variables that define the physical attributes of an artifact are always nonzero and positive. In Figure I.1 the system variables X_1 and X_2, being independent, are represented by the abscissa and ordinate, respectively. Each member of the set *X* represents an axis of an 'n' dimensional space.

A *system constraint* is a constraint placed on the design. The set of system constraints must be satisfied for the feasibility of the design. Mathematically, system constraints are functions of system variables only. They are rigid and no violations are allowed. They relate the demand placed on the system *D(X)* to the capability of the system *C(X)* to meet the demand. The region of feasibility defined by the system constraints is called the *feasible design space.*

The set of system constraints may be all linear, nonlinear or consist of both linear and nonlinear functions. In engineering problems the system constraints are invariably inequalities. However, occasions requiring equality constraints may arise. All system constraints shown in Figure I.1 are inequalities.

I.5.2 Deviation Variables and System Goals

A set of system goals is used to model the aspiration a designer has for the design. It relates the goal (aspiration level), G_i, of the designer to the actual attainment, $A_i(X)$, of the goal. Three conditions need to be considered:

1. $A_i(X) \leq G_i$; we wish to achieve a value of $A_i(X)$ that is equal to or less than G_i.
2. $A_i(X) \geq G_i$; we wish to achieve a value of $A_i(X)$ that is equal to or greater than G_i.
3. $A_i(X) = G_i$; we would like the value of $A_i(X)$ to equal G_i.

We will now introduce the concept of a deviation variable. Consider the third condition, namely, we would like the value of $A_i(X)$ to equal G_i. The deviation variable is defined as: $d = G_i - A_i(X)$. The deviation variable d can be negative or positive. Considerable simplification of the solution algorithm is effected if one can assert that all the variables in the problem being solved are positive. Therefore, the deviation variable d is replaced by two variables:

$$d = d_i^- - d_i^+$$

where

$$d_i^- \cdot d_i^+ = 0$$

and

$$d_i^- , d^+ \geq 0 . \qquad i$$

The preceding ensures that the deviation variables never take on negative values.

The system goal becomes:

$$A_i(X) + d_i^- - d_i^+ = G_i ; \qquad i = 1, 2, ..., m \qquad (1.1)$$

where

$$d_i^- , \; d_i^+ \geq 0 \qquad \text{and} \qquad d_i^- \,.\, d_i^+ = 0$$

The product condition ensures that one of the deviation variables will always be zero. If the problem is solved using an algorithm that provides a vertex solution as a matter of course then the condition is automatically satisfied, making its inclusion in the formulation redundant. Since, the solution scheme described in this book and the software that is available for solution makes use of an algorithm that provides a vertex solution we will assume that this condition is satisfied. For completeness we include this condition as a constraint in the mathematical forms of the compromise DSP given later in this chapter and for brevity will omit this constraint from all subsequent formulations.

Note that a system goal is always expressed as an equality. It is possible that the designer's aspiration levels are inordinately high, or the system constraints are much too restrictive to attain the desired levels of achievement. The deviation variables d_i^- and d_i^+ are used to allow the designer a certain degree of latitude in making decisions. The deviation variables therefore relate the actual performance of the design to the aspired level of performance. These variables serve to "anchor" the aspiration levels to realistic achievement levels. When considering Equation 1.1, the following will be true:

IF $A_i \leq G_i$ (underachievement) THEN $d_i^- > 0$ and $d_i^+ = 0$.

IF $A_i \geq G_i$ (overachievement) THEN $d_i^- = 0$ and $d_i^+ > 0$,

IF $A_i = G_i$ (exact achievement) THEN $d_i^- = 0$ and $d_i^+ = 0$

How do we model the three conditions listed earlier using Equation 1.1?

1 To satisfy $A_i(X) \leq G_i$, we must ensure that the positive deviation d_i^+ is zero. The negative deviation d_i^- will measure how far is the performance of the actual design from the goal.

2 To satisfy $A_i(X) \geq G_i$, the negative deviation d_i^- must be made equal to zero. In this case, the degree of overachievement is indicated by the positive deviation d_i^+.

3 To satisfy $A_i(X) = G_i$, both deviations, d_i^- and d_i^+ must be zero.

The difference between a system variable and a deviation variable is that the former represents a distance in the ith dimension from the origin of the design space, whereas the latter originates on the surface of the system goal. This is illustrated in Figure I.2. The value of the

i^{th} deviation variable is determined by the degree to which the i^{th} goal is achieved. It depends upon the value of $A_i(X)$ alone (since G_i is fixed by the designer). $A_i(X)$ in turn is dependent upon the system variables X. The set of deviation variables can be all continuous, all Boolean or some can be Boolean and others continuous. Obviously, both the deviation variables associated with a particular system goal will be of the same type.

The system goal represents an equation for a family of either parallel linear or nonlinear functions. In Figure I.2, goal i (represented by line A) is the target goal to be achieved. Assume that lines B and C represent the maximum acceptable excursion that is possible from the target goal. In other words, the system variables can achieve any value in the shaded region. Three representations for lines B and C are shown in the figure, as follows,

1. In terms of system variables.
2. In terms of the system variables and the nonzero deviation variable.
3. In terms of the system variables and both the deviation variables.

In 1 (see Figure I.2) the right hand sides for the equations for A, B and C are different. In 2 and 3 the right hand sides for both B and C are the same (b_1) however the deviation variables are different. In 3 both B and C are expressed in terms of the system variables and the two deviation variables. For B, the underachievement d_1^- is nonzero and the overachievement d_1^+ is zero. For C it is the other way around. Since only one deviation variable, by definition, can be nonzero we are able to write the equation for the family of system goals B through C. This is analogous to Equation 1.1.

I.5.3 Range of Values for Deviation Variables

The objective of a traditional single objective optimization problem requires the maximization or minimization of a certain function. This function is in terms of the system variables. In a compromise DSP formulation, each of the objectives is converted into a goal (using Equation 1.1) with its corresponding deviation variables. The resulting formulation is similar to a single objective optimization problem but with the following differences:

- The objective is always to minimize a function.
- The objective function is expressed using deviation variables only.

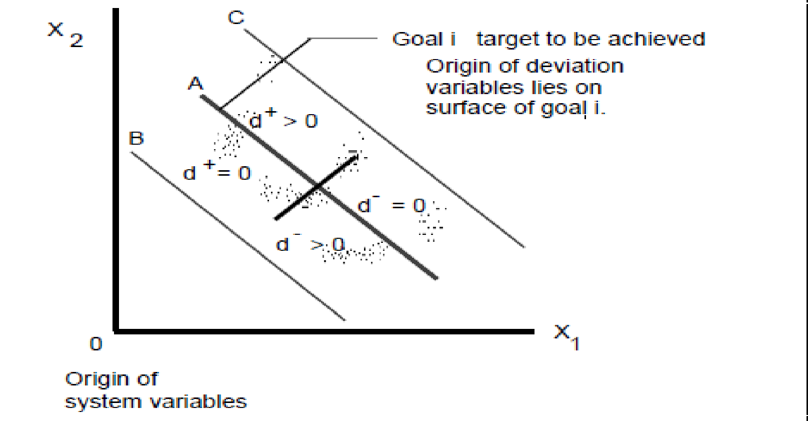
System Variables: X_1, X_2, … Form the axes of the design space.

Deviation Variables:

d^- Represents underachievement of the goal.

d^+ Representsoverachievement of the goal.

Note: Goals are represented by the parallel lines A, B and C.



X_2
C
A
B
Goal i target to be achieved
Origin of deviation variables lies on surface of goal i.
$d^+ > 0$
$d^+ = 0$
$d^- = 0$
$d^- > 0$
0
X_1
Origin of system variables

Equation for Goal (Line) A	*Numerical Example*
1. $a_1X_1 + a_2X_2 = b_1$	Say .... $8X_1 + 7X_2 = 13$
Equation for Goal (Line) B	
1. $a_1X_1 + a_2X_2 = b_2$; $b_2 < b_1$	Say $8X_1 + 7X_2 = 10$
2. $a_1X_1 + a_2X_2 + d_1^- = b_1$; $d_1^- \neq 0$	$8X_1 + 7X_2 + d_1^- = 13$; $d_1^- = 3$
3. $a_1X_1 + a_2X_2 + d_1^- - d_1^+ = b_1$	$8X_1 + 7X_2 + d_1^- - d_1^+ = 13$
with $d_1^- > 0$, $d_1^+ = 0$	with $d_1^- = 3$, $d_1^+ = 0$
Equation for Goal (Line) C	
1. $a_1X_1 + a_2X_2 = b_3$; $b_3 > b_1$	Say $8X_1 + 7X_2 = 15$
2. $a_1X_1 + a_2X_2 - d_1^+ = b_1$; $d_1^+ \neq 0$	$8X_1 + 7X_2 - d_1^+ = 13$;$d_1^+ = 2$
3. $a_1X_1 + a_2X_2 + d_1^- - d_1^+ = b_1$	$8X_1 + 7X_2 + d_1^- - d_1^+ = 13$
with $d_1^- = 0$, $d_1^+ > 0$	with $d_1^- = 0$, $d_1^+ = 2$
Equation for a Family of Goals B Through C	
3. $a_1X_1 + a_2X_2 + d_1^- - d_1^+ = b_1$	$8X_1 + 7X_2 + d_1^- - d_1^+ = 13$
with d_1^- , $d_1^+ \geq 0$	with d_1^- , $d_1^+ \geq 0$

Figure I. 2: The System Goal

The objective in the compromise DSP formulation is called the deviation function. As indicated earlier, the deviation variables are associated with system goals and therefore their ranges of values depend on the goal itself. Goals are not equally important to a designer. Therefore to solve the problem, given a designer's preferences, the goals are rank ordered into priority levels. Within a priority level it is imperative that the deviation variables are of the same order of magnitude. This is achieved by normalizing the goals. If this is not done the deviation variable with the larger numeric value will dominate the solution process without regard to the designer- established preference for the set of goals.

A solution to the order of magnitude problem is to normalize the achievement $A_i(X)$ with respect to the target value G_i before the deviation variables are introduced. The following rules are used to formulate the system goals in a way that ensures that all the deviation variables will range within the same values (0 and 1 in this case).

a. To maximize the achievement, $A_i(X)$, choose a target value G_i greater or equal to the maximum expected value of $A_i(X)$, so that the ratio $A_i(X)/G_i$ is always less or equal than 1. For example, if $A_i(X)$ is the reference stress then G_i could be the yield stress. Consider the following:

$$A_i(X) \leq G_i \qquad \Rightarrow \qquad A_i(X)/G_i \leq 1$$

Transform the expression into a system goal by adding and subtracting the corresponding deviation variables (which in this case will range between zero and one).

$$A_i(X) / G_i + d_i^- - d_i^+ = 1 \qquad (1.2)$$

In this case, the overachievement variable, d_i^+, will always be zero, as indicated in Section 4.2.2.

Then minimize the underachievement deviation, d_i^-, to ensure that the performance of the design will be as close as possible to the desired goal.

b. To minimize the achievement, $A_i(X)$, the following steps are in order:

i. Choose a target value, Gi, less than or equal to the minimum expected value of Ai(X). In this case, the ratio $G_i / A_i(X)$ will be less than or equal to one.

$$A_i(X) \geq G_i \qquad \Rightarrow \qquad G_i / A_i(X) \leq 1$$

Transform the expression into a system goal (note the inversion of G and A) and flip the signs of the deviation variables (to account for the inversion). The deviation variables will vary between 0 and 1.

$$G_i / A_i(X) - d_i^- + d_i^+ = 1 \qquad (1.3)$$

The underachievement deviation, d_i^-, will be zero as indicated in Section I.5.2. Minimizing the overachievement deviation, d_i^+, will ensure that the performance of the design is as close as possible to the desired goal.

i. If the target value, G_i, is taken as zero, get an estimate of the maximum value that the achievement, $A_i(X)$, can obtain within the bounds set for the system variables, $A_i^{max}(X)$.Then divide the inequality by this maximum value and convert into a system goal with the following result:

$$A_i(X) / A_i^{max}(X). + d_i^- - d_i^+ = 0 \qquad (1.4)$$

The deviation variables will now vary between 0 and 1. Note that the signs of the deviation variables remain as in the original Equation 1.1. In this case, the underachievement deviation d_i^-

will always be zero. Minimize then the overachievement deviation d_i^+ to ensure that the performance of the design will be as close as possible to the desired value of zero.

c. If it is desired that $A_i(X) = G_i$, and

i. If the target value G_i is approached from below by $A_i(X)$, use Equation 1.2 and minimize the sum $(d_i^- + d_i^+)$.

ii. If the target value G_i is approached from above by $A_i(X)$, use Equation 1.3 and minimize the sum $(d_i^- + d_i^+)$.

iii. If the target value G_i is equal to zero, use Equation 1.4 and minimize the sum $(d_i^- + d_i^+)$.

I.5.4 Bounds on System and Deviation Variables

Bounds are specific limits placed on the magnitude of each of the variables. Each variable has associated with it a lower and an upper bound. Bounds are important for modeling real-world problems because they provide a means to include the experience-based judgment of a designer in the mathematical formulation. Unfortunately, in most engineering design textbooks that encourage the notion of using optimization techniques in design there has been a tendency to ignore bounds. Bounds on the system variables take the form

$$L \leq X \leq U$$

where L and U represent the set of lower and upper bounds, respectively. The bounds on the system variables demarcate the region in which a search is to be made for a feasible solution. In engineering design, the lower bounds are always nonzero and positive, reflecting physical limitations.

Deviation variables are by definition nonnegative (see Section I.5.2) and therefore a lower bound of zero is always associated with them.

I.5.5 The Deviation Function

In the compromise DSP formulation, the aim is to minimize the difference between that which is desired and that which can be achieved. This is done by minimizing the deviation function, $Z(d^-, d^+)$, which is always written in terms of the deviation variables.

A designer sets an aspiration level for each of the goals. It may be impossible to obtain a design that satisfies all the levels of aspiration. Therefore, a compromise solution must be accepted by the designer. It is desirable, however, to obtain a design whose performance matches the aspiration levels as closely as possible. This in essence is the objective of a compromise solution. The difference between the goals and achievement is expressed by a combination of appropriate deviation variables, $Z(d^-, d^+)$. This deviation function provides an indication of the extent to which specific goals are achieved.

All goals may not be equally important to a designer and the formulations are classified as Archimedean or Preemptive - based on the manner in which importance is assigned to satisficing the goals.

In the following section, we will delve into both of these forms, examining their applications in optimization and cDSP construct , and ultimately, we will compare their respective outcomes.

I.6 Archimedean and Preemptive Form : Comparing Solutions

Archimedean In the context of multiobjective optimization: In the context of multiobjective optimization, a mathematical framework that allows the combination of multiple objectives into a unified objective function has been provided in the Archimedean form. With the Archimedean form, compromise solutions can be identified by decision-makers, taking into account conflicting objectives.

The Archimedean form is characterized by the aggregation of individual objectives using an Archimedean aggregation operator. This operator incorporates the relative weights or importance assigned to each objective by the decision-maker, enabling the expression of preferences and the reflection of the relative significance of different objectives in the optimization process.

One of the commonly used Archimedean aggregation operators is the weighted sum operator. It involves the linear combination of individual objectives by multiplying each objective with its corresponding weight and summing the results. The weighted sum operator is expressed as follows:

The aggregated objective function, F(x), is given by the equation:

$$F(x) = w_1 * f_1(x) + w_2 * f_2(x) + ... + w_n * f_n(x)$$

Here, F(x) represents the aggregated objective function, while $f_1(x)$, $f_2(x)$, ..., $f_n(x)$ denote the individual objective functions. The weights w_1, w_2, ..., w_n are associated with each objective and reflect their relative importance.

Determining the weights is subjective and relies on the decision-makers preferences, priorities, and domain knowledge. Techniques such as direct elicitation, pairwise comparisons, or the analytic hierarchy process (AHP) can be employed to assign appropriate weights.

In addition to the weighted sum operator, other Archimedean aggregation operators can be utilized based on the decision-makers preferences and the nature of the objectives. For example, the weighted product operator considers the product of individual objectives raised to their corresponding weights. The weighted power mean operator computes a weighted average of individual objectives using a power

function.

Once the objectives are aggregated into a single objective function, optimization algorithms can be employed to search for optimal or near-optimal solutions. The objective is to identify solutions on the Pareto front, representing the set of non-dominated solutions. A solution on the Pareto front cannot be improved in one objective without sacrificing at least one other objective.

Decision-makers could get a structured framework for handling multiobjective optimization problems with Archimedean form. By converting multiple objectives into a single objective function, the complexity of analyzing and comparing objectives is simplified. Adjusting the weights allows decision-makers to explore different trade-offs and identify solutions that align with their preferences and requirements.

Archimedean In the context of cDSP: A significant role is played by the Archimedean form in the context of the Compromise Decision Support Problem (cDSP) by providing a mathematical framework to handle conflicting objectives and facilitate decision-making. In the cDSP, the focus is on finding compromise solutions that balance multiple objectives, considering the inherent conflicts, resource limitations, and uncertainties present in real-world decision scenarios.

The cDSP formulation is aimed at minimizing the difference between desired goals and achievable goals in the Archimedean form. In this formulation, weights can be assigned to each goal by the designer based on their level of importance and satisfaction.

In the Archimedean formulation, the deviation function, denoted as $Z(d^-, d^+)$, is used to express the difference between the desired goals and their achievement.

The underachievement and overachievement of each goal are represented by the deviation variables (d^-, d^+) in the Archimedean form.

The deviation function is defined as follows:

$$Z(d^-, d^+) = \sum(W_i(d_i^- + W_i(d_i^+)) \qquad \text{for } i = 1,\ldots,m$$

In this formulation, m represents the number of goals, and W_i denotes the weight assigned to the i-th goal. The weights reflect the level of desire to achieve each goal, and they should satisfy the following conditions:

$$\sum W_i = 1, \quad \text{and} \quad W_i \geq 0 \quad \text{for all } i.$$

The representation of aspiration levels for each goal, which signifies the desired performance, is enabled by the Archimedean formulation. However, the recognition that it may be impossible to simultaneously achieve all aspiration levels is acknowledged by the Archimedean formulation. Therefore, a compromise solution must be accepted. In the Archimedean formulation, the deviation function is minimized by considering the weighted sum of the deviation variables for each goal. The objective is to find a solution that closely matches the desired aspiration levels for all goals.

Various methods can be used to determine the appropriate weights, such as pairwise comparison techniques. These methods involve comparing the goals in pairs and expressing the preference between them. Overall, balancing multiple goals in the compromise, DSP is facilitated by the framework provided by the Archimedean formulation, allowing informed decisions to be made by the designer based on the relative importance of each goal and the extent to which they are achieved.

The formulation classifies the goals into different priority levels. The designer can determine the importance of goals either by using Archimedean weights or by rank ordering the goals in a preemptive approach. The previous discussion covered the Archimedean formulation of the compromise Decision Support Problem (cDSP). Now, let us delve into the preemptive form of the cDSP, which provides an alternative approach to solving such problems.

Preemptive In The Context Of Multiobjective Optimization: In the context of multiobjective optimization, the preemptive form involves rank-ordering goals or objectives based on their priority levels. The sequential optimization of goals is permitted by this approach, with higher priority goals being achieved before lower priority goals are addressed. The preemptive form is particularly useful when assigning weights or preferences to the goals is difficult.

In the preemptive form, goals are organized into priority levels, and the optimization process aims to minimize the deviations or underachievement of each goal at its respective priority level. Once a goal is fully satisfied, the optimization focuses on the next priority-level goal. This process continues until all goals have been addressed.

The formulation of the preemptive form involves defining deviation variables for each goal and constructing a deviation function that captures the deviations at each priority level. The deviation function is designed to minimize the deviations at each priority level while maintaining the achieved levels of higher priority goals.

A structured approach to multiobjective optimization, allowing for systematic consideration of goal priorities, has been provided by the preemptive form. It is particularly useful when there is limited information available to assign precise weights or preferences to the goals. By prioritizing higher priority goals first, a way is provided to achieve a satisfactory compromise solution while considering the relative importance of different objectives. Note that the preemptive form is just one of the approaches in multiobjective optimization, and the choice of the approach depends on the problem characteristics, available information, and the preferences of the decision-maker.

Preemptive In The Context Of cDSP: In the preemptive formulation, the emphasis is placed on rank-ordering the goals rather than assigning explicit weights to them. This approach is particularly useful when it is challenging to assign meaningful weights or when there is limited information available in the early stages of design or in an industrial environment. The preemptive form is one of the approaches in multiobjective decision support problems, and the choice of the approach depends on the problem characteristics, available information, and the preferences of the decision-maker.

In the Preemptive approach, the difficulty of assigning weights is circumvented by rank ordering the goals. The measure of achievement is obtained in terms of the lexicographic minimization of an ordered set of goal deviations, wherein within each set of goals at a particular rank, weights may be used. Goals are ranked lexicographically, and an attempt is made to achieve a more important goal before considering other goals.

The mathematical definition of lexicographic minimum follows,

LEXICOGRAPHIC MINIMUM Given an ordered array f of nonnegative elements, fk's, the solution, given by $f^{(1)}$, is preferred to $f^{(2)}$ if

$$f_k^{(1)} < f_k^{(2)}$$

and all higher-order elements (i.e., $f_1, \ldots, f_{k-1}$) are equal. If no other solution is preferred to f, then f is the lexicographic minimum.

The goals are ranked in order of priority, and the aim is to achieve the higher-ranked goals before considering the lower-ranked ones. This ranking represents the preference of one goal over another without quantifying the degree of preference or importance.

To illustrate this concept, consider a set of goals with associated deviation variables: goal 1 (d_1^-, d_1^+), goal 2 (d_2^-, d_2^+), goal 3 (d_3^-, d_3^+), and so on. The preemptive deviation function can be expressed as:

$$Z = [\ f_1(d_1^-, d_1^+), f_2(d_2^-, d_2^+), f_3(d_3^-, d_3^+), \ldots\ ]$$

In this formulation, each priority level, represented by f_1, f_2, f_3, etc., is minimized sequentially. The priority levels are determined by the rank order of the goals. The objective is to find a solution that minimizes the deviation variables at each priority level while maintaining the achieved goals at the higher priorities. **Formulation of the preemptive form in cDSP involves the following steps:**

Goal Ranking: The first step is to rank the goals in order of priority. This ranking is typically based on the preferences and importance assigned by the decision-maker. Higher-ranked goals are considered more important and should be achieved before lower-ranked goals.

Deviation Variables: Deviation variables are introduced to quantify the deviation between the desired goal values and the achieved values. For each goal, two deviation variables are defined: d_i^- (underachievement) and d_i^+ (overachievement). The extent to which a particular goal is not fully satisfied or exceeds the desired level is represented by these variables.

Deviation Function: The deviation function, denoted as $Z(d^-, d^+)$, is formulated to measure the overall deviation between the desired goals and their actual achievement. In the preemptive form, the deviation function is written as:

$$Z = [\ f_1(d_i^-, d_i^+), \ldots, f_k(d_i^-, d_i^+\ )]$$

Where k represents the number of goals or priority levels, the deviation variables corresponding to each priority level are included in the deviation function.

Minimization: The objective is to minimize the deviation function while satisfying the constraints of the problem. The solution is sought by finding a set of values for the decision variables that minimize the deviations for the highest-priority goal and then sequentially minimize the deviations for lower-priority goals.

Mathematical Representation: The preemptive form can be mathematically represented using cDSP techniques. The formulation includes the goal constraints, system constraints, and the deviation function. The goal constraints specify the desired levels of achievement for each goal, while the system constraints represent the limitations or requirements of the problem.

By solving the preemptive form of cDSP, a solution that achieves the highest-priority goal as closely as possible while considering the subsequent goals in the ranked order is obtained.

Example:

As an example, consider two solutions, $f^{(r)}$ and $f^{(s),}$ where

$$f^{(r)} = (0, 10, 400, 56)$$

$$f^{(s)} = (0, 11, 12, 20)$$

In this example, note that $f^{(r)}$ is preferred to $f^{(s).}$ The value 10 corresponding to $f_2^{(r)}$ is smaller than the value 11 corresponding to $f_2^{(s)}$. (Since the objective is to minimize or achieve smaller values for the second element, $f^{(r)}$ is preferred over $f^{(s)}$ because $f_2^{(r)}$ (10) is smaller than $f_2^{(s)}$ (11)) .Once a preference is established, then all higher-order elements are assumed to be equivalent. Hence, the deviation function for the Preemptive formulation is written as

$$Z = [\ f_1(d_i^-,d_i^+), \ldots, f_k(d_i^-,d_i^+)\ ] .$$

For a four goal problem, the deviation function may look like

$$Z(d^-, d^+) = [\ (d_1^- + d_2^-), (d_3^-), (d_4^+)\ ]$$

In this case, three priority levels are considered. The deviation variables, d_1^- and d_2^-, have to be minimized preemptively before variable d_3^- is considered, and so on. These priorities are represented by rank, indicating the preference for one goal over another. No conclusions can be drawn with respect to the amount by which one goal is preferred or is more important than another. This approach is, therefore, suitable when there is little information available. For a simple problem with only two system variables, a graphical solution can be easily found by satisficing the goals in a logical manner. This is in contrast to the Archimedean approach, in which the numerical evaluation of the deviation function is required even for the simplest case. The numerical solution of a Preemptive formulation requires the use of a special optimization algorithm developed to solve these types of problems. One such algorithm has been developed by Ignizio [J. P. Ignizio, "Multiobjective Mathematical Programming via the MULTIPLEX Model and Algorithm," European Journal of Operational Research, 22, 1985, 338-346.]. It is also possible to solve the Preemptive formulation by reformulating the deviation function into a pseudo-preemptive form, as suggested by Schniederjans [M. J. Schniederjans, Linear Goal Programming, Petrocelli Books, Princeton, N.J, 1984.]. Schniederjans' notion is to force the deviation function to satisfy the priorities by multiplying each priority level by a quantity Pi, whose numerical value is much larger than the corresponding one associated with the next priority level. The deviation function for the example problem presented earlier expressed in a pseudo-preemptive fashion looks like

$$Z(d^-, d^+) = P_1 (d_1^- + d_2^-) + P_2 (d_3^-) + P_3 (d_4^+) \qquad \text{where } P_1 >> P_2 >> P_3,$$

In the preceding, the >> implies preference, and the Pi's represent rank-ordered priorities that are modeled numerically. Lexicographic preference is modeled numerically on a computer only if the numerical values between the priorities are substantial. For example, let us try to model the following numerically:

P1 >> P2 >> P3.

Consider the following series of numbers:

3 >> 2 >> 1

300 >> 200 >> 100

1037 >> 1020 >> 1010 .

Which of the three series models is the preference the best? The correct answer is the third set of numbers.

In the subsequent section, an illustrative example is employed to provide a more detailed explanation of the contrast in solutions achieved through the utilization of preemptive and Archimedean formulations.

Comparing Solutions: Preemptive and Archimedean Formulations

The following example is presented to illustrate the difference in the solution obtained by using the preemptive and Archimedean formulations. The design space for the example problem is shown in Figure I.3.

The algorithms that have been developed to solve the compromise DSPs provide vertex solutions. Therefore, we will restrict our discussion to vertex solutions only. Further, we are seeking a solution that achieves all three goals completely.

Find

System Variables X_1, X_2

Deviation Variables $d_1^-, d_1^+, d_2^-, d_2^+, d_3^-, d_3^+$

Satisfy

System Constraints

$2X_1 + 3X_2 \leq 30$ [c1]

$6X_1 + 4X_2 \leq 60$ [c2]

System Goals (Dimensionless, Normalized)

$X_1/10 + X_2/10 + d_1^- - d_1^+ = 1$ [g1]

$X_2/ 7 + d_2^- - d_2^+ = 1$ [g2]

$X_1/8 + d_3^- - d_3^+ = 1$ [g3]

Bounds omitted for brevity.

Minimize

Case a: Using the Preemptive approach (lexicographic minimum).

$Z = [(d_1^-+ d_1^+), (d_2^-+ d_2^+), (d_3^-+ d_3^+)]$

All deviation variables are considered due to equality goals.

Case b: Using the Archimedean approach.

$Z = W_1(d_1^-+ d_1^+), + W_2(d_2^-+ d_2^+), + W_3(d_3^-+ d_3^+),$

where W1 = W2 = W3 = 1/3 (assumed values)

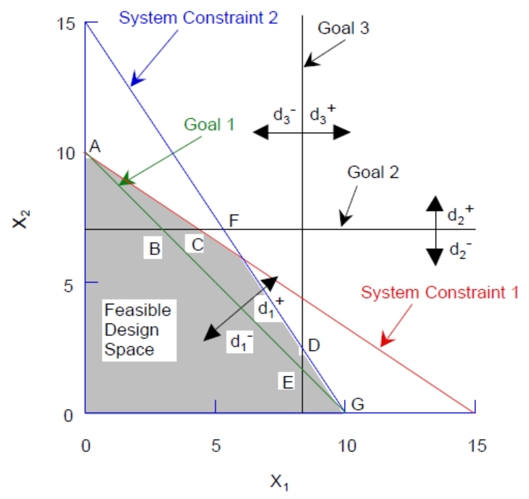


Figure I. 3: Design Space, For Example Problem

The solution to the preceding compromise DSP using both the Preemptive and Archimedean approaches follows:

Case A: Preemptive

- The goal with the highest priority is considered first (Goal 1). This goal lies completely within the feasible design space, and consequently, any point satisfying the goal is considered to be a solution, namely, vertices A, B, E, and G.
- We next move to priority level 2, which requires the minimization of d_2^- and d_2^+. Notice that in Figure 1.2, these deviations may be set to zero at point B without reducing the value of the solution obtained for priority 1. That is, d_2^- and d_2^+ may be set to zero without any increase in either d_1^- or d_1^+. Therefore, vertex A is the second preferred solution, with the first priority still being satisfied (d_1^-, $d_1^+ = 0$) and a minimum value for the overachievement of the second goal d_2^+.
- Moving to priority level 3, we attempt to minimize d_3^- and d_3^+ without degrading the solution for the other priority levels. In this case, the solution point that comes closer is once again

point B, with d_1^-, d_1^+, d_2^-, $d_2^{+,}$ and d_3^+= 0 and a minimum value of the third goal underachievement d_3^-.

- If the priorities were changed to goals 1, 3, and 2 in that order, the preferred solution would be at point E. We suggest that you solve this as an exercise.

Case B : Archimedean

In Table I.1, the values of the deviation variables and the deviation function at different vertices are summarized. It follows from the table that the best solution is at 'C' where Z is a minimum Z=0.196).

The solutions obtained in the two cases are different. The Preemptive approach is suitable when less is known about the design, and consequently, a designer can only rank-order the preferences for the goals. Using the Archimedean approach is warranted when it is possible to determine the relative importance of the goals using a pairwise comparison method.

Should all the deviation variables be included in the formulation of the deviation function? Deviation variables will be zero; consequently, we can exclude them from the deviation function. If one is interested in varying the target values to study the sensitivity of the solution, it is necessary to include all the deviation variables in the deviation function.

Table I.1: Deviation Function Values For Archimedean Solution

Acceptable Value of Solution								
Vertices	*Normalized Dev. Var.*						*Sum*	
Func.								
(coord.)	d_1^-	d_1^+	d_2^-	d_2^+	d_3^-	d_3^+	$\sum(d_i^- + d_i^+)$	
Z								
A = (0, 10)	0	0	0	0.429	1	0	1.429	0.476
B = (3, 7)	0	0	0	0	0.625	0	0.625	0.208

C = (4.5, 7)	0	0.150	0	0	0.438	0	0.588	***0***.*196*
D = (8, 3)	0	0.100	0.571	0	0	0	0.671	0.224
E = (8, 2)	0	0	0.714	0	0	0	0.714	0.238
F = (6, 6)	0	0.200	0.143	0	0.250	0	0.593	0.198
G = (10, 0)	0	0	1	0	0	0.250	1.250	0.417

Subsequent to the deliberation on the compromise decision support problem, the following section, the DSIDES software, a developed tool designed for the implementation of the cDSP construct, has been described.

I.7 Expanded cDSP Examples : The Two Coal Problem

Recent revisions of pollution control laws have had a direct influence on the running of a power station. These revisions have reduced the allowable emission of pollutants into the atmosphere from the plant's exhaust gases. To comply with these new regulations expeditiously and eliminate downtime it is desired, now, to control the emission rates by the appropriate use of coal.

Historically, coal has been bought from two sources, say A and B. Both types of coal are transported to the plant and stored in separate stockpiles. From there they are fed by a mechanical conveyer into a pulverizer, crushed into fine particles, mixed at a specified rate and burnt in a combustion chamber.

Coal from source A, Coal A, is relatively hard, clean burning, has a low sulfur content and is more expensive than Coal B which is soft, smoky when burnt, and has a high sulfur content. The thermal value, in terms of steam produced, is 24,000 lbs/ton for Coal A and 20,000 lbs/ton for Coal B. Since Coal A is hard, the pulverizer can handle 16 tons/hr of it. Since Coal B is soft, the pulverizer can handle 24 tons/hr of it. The capacity of the conveyer is 20 tons/hr for both types of coal. There is a limit to the amount of coal that can be stockpiled. This limit translates to a maximum of 25 tons/hr for any type of coal that can be burnt.

The new pollution regulations limit sulfur oxide emissions to 3,000 parts per million and the particulate emissions (smoke) to 12 kg/hr. The characteristics of the two types of coal are summarized in Table I.2.

Case A: A Linear Single Objective Optimization Problem.

Problem Statement: Determine the most efficient combination of the two types of coal to be burnt that satisfies the constraints and bounds and maximizes the rate of production of electricity. No information is given in the story about the lower bounds on the rates of consumption of the two types of coal. It is reasonable to assume, initially, that the lower bounds are zero. The implication of this assumption is that a solution that requires the burning of a single type of coal, in our case coal from a single vendor, is acceptable.

Table I.2: Coal and material handling characteristics

PROPERTIES	Coal A	Coal B	Units
Thermal value	24,000	20,000	lbs steam / ton
Sulfur oxides emission	1,800	3,800	ppm
Particulate emission	0.5	1.0	kg / ton
Pulverizer coal handling capacity	16	24	tons / hour
Conveyer coal handling capacity	20	20	tons / hour

Case B: The Linear Single Objective Optimization Problem - Revisited.

Problem Statement: The power company does not want to rely on a single source for its supply of coal. The purchasing department has determined that there is a minimum order quantity for the coal. This minimum order quantity translates to a lower bound of 5 tons of each type of coal per hour. Determine the most efficient combination of the two types of coal to be burnt that satisfies the constraints and bounds and maximizes the rate of production of electricity.

This problem is used to illustrate the algorithm for solving the most general linear single objective optimization problem. The preceding requires the introduction of two nonzero lower bounds on the system variables and the introduction of artificial variables to get an initial solution by inspection. Unlike Case A, in this case it is inappropriate to drop the lower bounds at the very outset.

Case C: A Linear Single Goal Compromise Decision Support Problem.

Problem Statement: The demand for steam is likely to increase to 432,000 lbs/hr during the summer months. Can this be achieved without violating any of the constraints and bounds?

This problem is formulated as a single goal compromise DSP, and the solution compared to that of the single objective optimization problem, Case A.

Case D: A Linear, Multigoal Compromise Decision Support Problem.

Problem Statement: The demand for steam is going to increase to at least 432,000 lbs/hr during the summer months. Management is prepared to violate some of the pollution constraints and pay a fine, if necessary, to get this amount of steam from the plant. Company executives have identified one viable scenario as putting out as much sulfur oxides and smoke as is permissible and maximizing the amount of steam produced.

This problem is used to illustrate the solution of a multigoal compromise DSP.

Case E: A Nonlinear, Multigoal Compromise Decision Support Problem.

Problem Statement: The plant is being modified and access to the warehouse is limited. For a short period the management would like to hold the stockpiling down so as not to exceed 10 tons/hr of Coal A and 5 tons/hour of Coal B. The cost for stockpiling per ton is assessed proportionally to the excess capacity, over the desired capacities, and is equal to $3/(excess ton of Coal A) and $4/(excess ton of Coal B). The prices reflect the difficulties in stockpiling the softer Coal B. An incentive

is provided in stockpiling less coal than available capacity and is equal to $3/(surplus ton of Coal A) and $4/(surplus ton of Coal B). It is desirable to limit the total stockpiling costs to $30/hr.

Linear Single Objective Optimization: Formulation and Graphical Solution

I.7.1 Case A: The Word Problem

Given

- ❑ The properties of Coal A and Coal B.
- ❑ The capacity of the conveyer unit.
- ❑ The capacity of the pulverizer unit.
- ❑ Emission limit on sulfur oxides.
- ❑ Emission limit on particulates.
- ❑ Upper limits on the amount of coal that is stockpiled.

Assumptions

- ❑ Uninterrupted supply of coal is available.
- ❑ The combustion chamber can handle any amount of coal supplied from the pulverizer.

- ❑ The maximization of the rate of electricity produced is equivalent to the maximization of the rate of steam generated.
- ❑ The coal prices are stable.

Find

Independent System Variables

The rate of consumption of Coal A: X1 [tons/hr]

The rate of consumption of Coal B: X2 [tons/hr]

Satisfy

System Constraints

- ❑ The conveyer capacity is 20 tons/hr for any type of coal.
- ❑ The pulverizer can process 16 tons of Coal A and 24 tons of Coal B per hour.
- ❑ The emission of sulfur oxides is limited to 3,000 parts per million.
- ❑ The emission of particulates (smoke) is limited to 12 kg/hr.

Bounds on the System Variables

- ❑ The system variables should be nonnegative
- ❑ The maximum of any one type of coal that can be burnt is 25 tons/hr.

Maximize

- ❑ The rate of steam generated and therefore, the electricity produced.

I.7.2 Case A: Derivation of the Constraints and the Objective Function

The system variables. In the short run the plant's facilities are fixed. It is quite appropriate that management has decided to affect the output of electricity by using the best combination of the two types of coal. Therefore, let X_1 be the number of tons of Coal A burnt per hour, and

X_2 be the number of tons of Coal B burnt per hour. These variables

have two characteristics. One, they are physical quantities and are therefore nonnegative. Two, these variables are continuous, that is, any value that is feasible is acceptable from a mathematical standpoint.

The system constraints and bounds. The system constraints are written in terms of the system variables. In engineering, system constraints are invariably inequalities. The system constraints and bounds must be satisfied for feasibility. System constraints generally model the physics of the problem. The bounds, on the other hand, are the product of experience-based insight. They represent what is acceptable to the designer without regard to the physics of the problem. A constraint invariably has two or more system variables. A bound contains only one system variable and is always parallel (geometrically) to the axis represented by the system variable. Rarely is a constraint specified in terms of a single system variable. In this case the constraint plays the same role as a bound in the design space even though it may represent the physics of the problem.

1 The constraint on conveyer capacity

The conveyer has a capacity of 20 tons/hour. This capacity is independent of the type of coal that is placed on the conveyer. Therefore, the constraint is written as:

$X_1 + X_2 \leq 20$ [tons/hr]

The constraint is shown in Figure I.4.

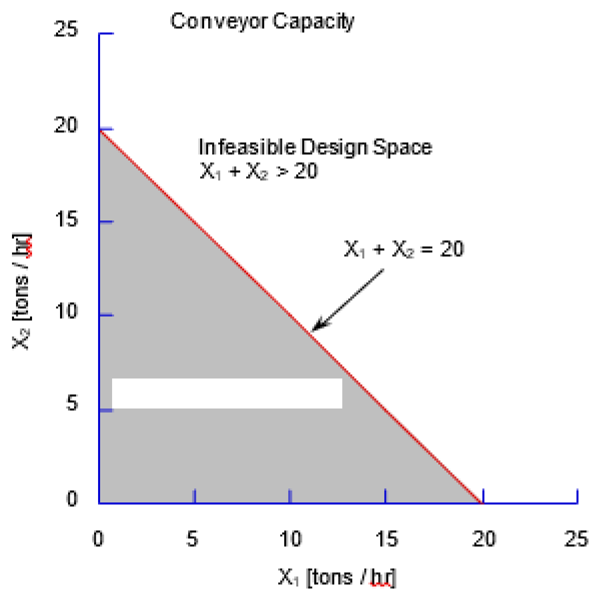


Figure I. 4: Conveyor capacity

2 The constraint on pulverizer capacity

The pulverizer capacity constraint is shown in Figure I.5. The maximum capacity of the unit is 16 tons of Coal A or 24 tons of Coal B per hour or any corresponding combination of the two. The right hand side for this constraint has not been specifically given in the problem statement. It has to be figured out. In this case, consider the amount that can be pulverized in one hour: it takes 1/16 of an hour to pulverize a ton of Coal A and 1/24 of an hour to pulverize a ton of Coal B.

Therefore, the constraint is written as:

$X_1/16 + X_2/24 \leq 1$

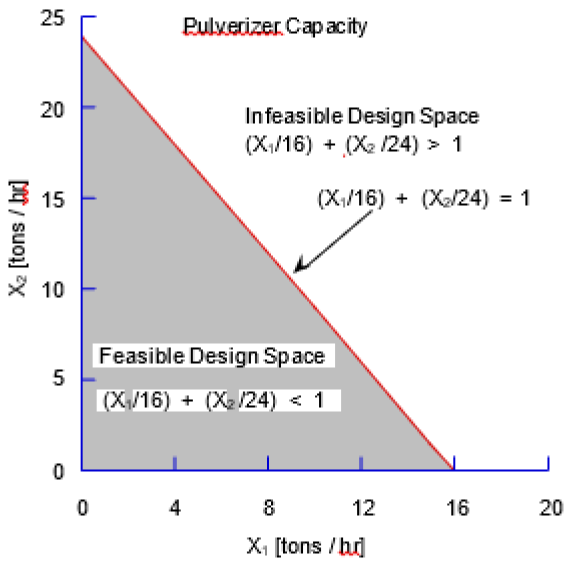


Figure I. 5: Pulverizer capacity

Therefore, the pulverizer capacity for one 24 hour day is (multiplying through by 24 [hrs/day]):

$1.5X_1 + X_2 \leq 24$ [day]

Notice the units. Normally, multiplying through does not result in meaningful units. In this case, because there are 24 hours in a day, the second form of the constraint has meaningful units.

3 The limit on sulfur oxides emission

The maximum emission of sulfur oxides is limited to 3,000 ppm. This constraint is shown in Figure I.6. There may be an urge to specify the constraint as:

$1{,}800X_1 + 3{,}800X_2 \leq 3{,}000.$

What is wrong with the constraint? The units on the left hand side and the right hand side of the equation do not match. What is to be done? The units of 1,800, 3,800 and 3,000 are parts per million. If only the X_1 and X_2 were dimensionless the preceding constraint would be acceptable. The way around this problem is to normalize X_1 and X_2 and make them dimensionless. How? Given that the two coals are burnt simultaneously, assume that a combination of X_1 tons/hr of Coal A and X_2 tons/hr of Coal B is fed into the combustion chamber as a homogeneous mixture.

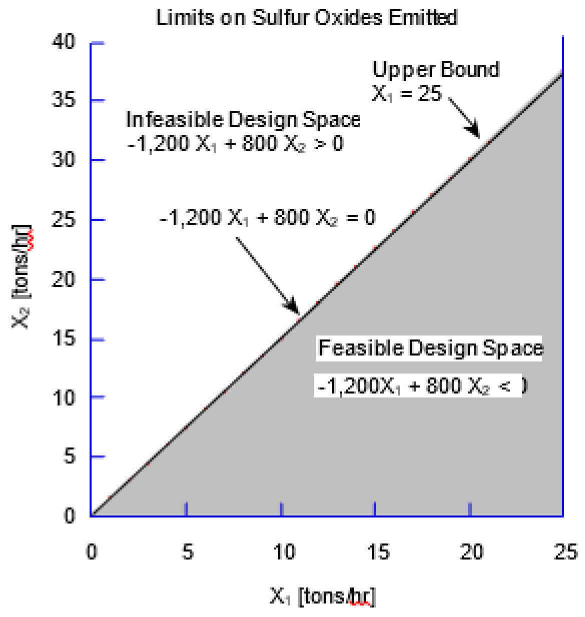


Figure I. 6: Limit on sulfur oxides

Then,

The proportion of Coal A in the total mixture is

$$X_1/(X_1 + X_2), \text{ and}$$

The proportion of Coal B in the total mixture is

$$X_2/(X_1 + X_2).$$

Now the constraint on the sulfur oxides emission level is equal to the weighted average of the individual levels, i.e.,

$$1{,}800X_1/(X_1 + X_2) + 3{,}800X_2/(X_1 + X_2) \leq 3{,}000 \qquad \text{[ppm]}$$

The preceding can be rewritten as:

$$-1{,}200X_1 + 800X_2 \leq 0 \qquad \text{[NMU]}$$

The second form of the constraint, though algebraically simpler than the first has no meaningful units (NMU). The second form is more convenient to use from a computational standpoint. Since the second form has no meaningful units associated with it, it will not be possible to gain much meaningful insight through post-solution analysis. Since the first form has meaningful units, it is preferred over the second form for the post-solution analysis.

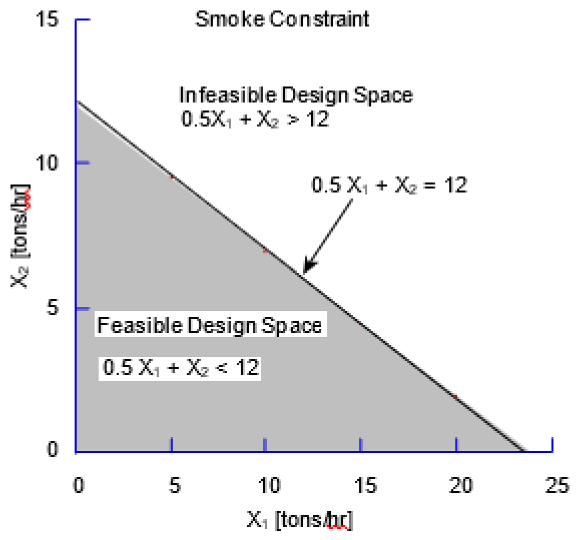


Figure I. 7: Smoke constraint

4 *The limit on particulate (smoke) emission*

According to the information given, each ton of Coal A produces 0.5 kg of smoke and each ton of Coal B, 1 kg of smoke. The amount of smoke that can be emitted per hour is limited to 12 kg. Therefore, this constraint is stated as:

$$0.5X_1 + X_2 \leq 12 \quad [\text{kg/hr}]$$

This constraint is shown in Figure I.7

5 *The lower bounds on the system variables*

Nothing is mentioned explicitly in the problem statement about lower bounds. Since this problem deals with physical quantities and they are always nonnegative, the lower bounds on the system variables are stated as follows:

$X1 \geq 0$ [tons/hr]

$X2 \geq 0$ [tons/hr]

6 The upper bounds on the system variables

The upper bounds on the system variables are explicitly stated in the problem statement and these are as follows:

$X1 \leq 25$ [tons/hr]

$X2 \leq 25$ [tons/hr]

The objective (deviation) function. The objective (see Figure 4.8) is to maximize the electricity produced at the plant. Since electricity is produced by using steam to drive the turbines, there is a direct relationship between the amount of electricity that is produced and the amount of steam that is produced in a specified length of time. What is the amount of steam produced for any arbitrary combination of coal used in any hour?

Coal	Steam (lbs/ton)	Fuel used (tons/hr)	Steam (lbs/hr)
A	24,000	X1	24,000X1
B	20,000	X2	20,000X2

The total amount of steam (lbs/hr) = 24,000X1 + 20,000X2. The objective function therefore is,

$Z = 24{,}000X1 + 20{,}000X2$ [lbs/hr]

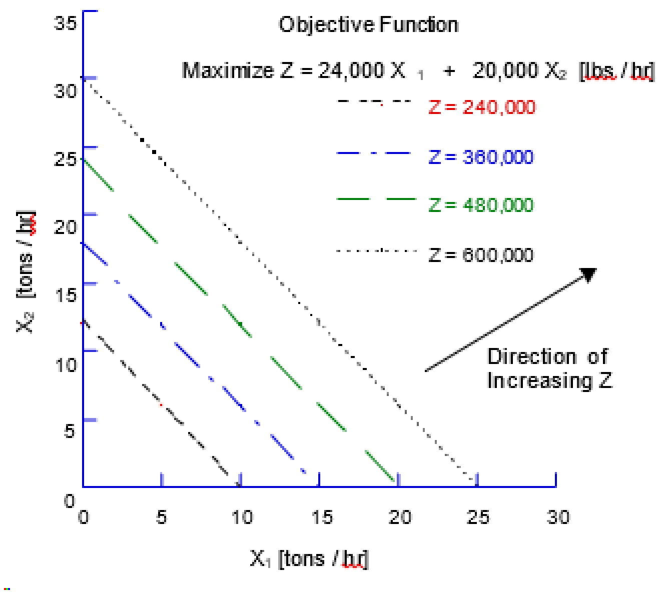


Figure I. 8: The objective function

I.7.3 Case A: The Mathematical Form of the Word Problem

Given

As stated in the word problem.

Find

System Variables

X_1 -	the rate of consumption of Coal A	[tons/hr]
X_2 -	the rate of consumption of Coal B	[tons/hr]

Satisfy

System Constraints

1. Conveyer capacity

$X_1 + X_2 \leq 20$ [tons/hr]

2. Pulverizer capacity

$X_1/16 + X_2/24 \leq 1$ [tons/hr] or

$1.5X_1 + X_2 \leq 24$ [tons/day]

3. Sulfur oxides emission

$1{,}800X_1/(X_1 + X_2) + 3{,}800X_2/(X_1 + X_2) \leq 3{,}000$ [ppm]

or

$-1{,}200X_1 + 800X_2 \leq 0$ [NMU]

4. Smoke emission

$0.5X_1 + X_2 \leq 12$ [kg/hr]

Bounds on system variables

5. Lower bounds on system variables

$X_1 \geq 0$ [tons/hr]

$X_2 \geq 0$ [tons/hr]

6. Upper bounds on system variables

$X_1 \leq 25$ [tons/hr]

$X_2 \leq 25$ [tons/hr]

Maximize

7. The rate of steam produced

$Z = 24{,}000X_1 + 20{,}000X_2$ [lbs/hr]

$= 24\,X_1 + 20\,X_2$ [1000 lbs/hr]

I.7.4 Case A: The Graphical Solution

The set of all combinations of the system variables that satisfy all constraints and bounds simultaneously is called the set of feasible solutions and the space consisting of the feasible solutions is called the feasible design space. This is shown in Figure I.9. A solution that results in the violation of any of the constraints or bounds is called an infeasible solution. A constraint or bound that does not border the feasible design space is called a redundant constraint or bound. In this example, the upper bounds and the conveyer constraint are redundant.

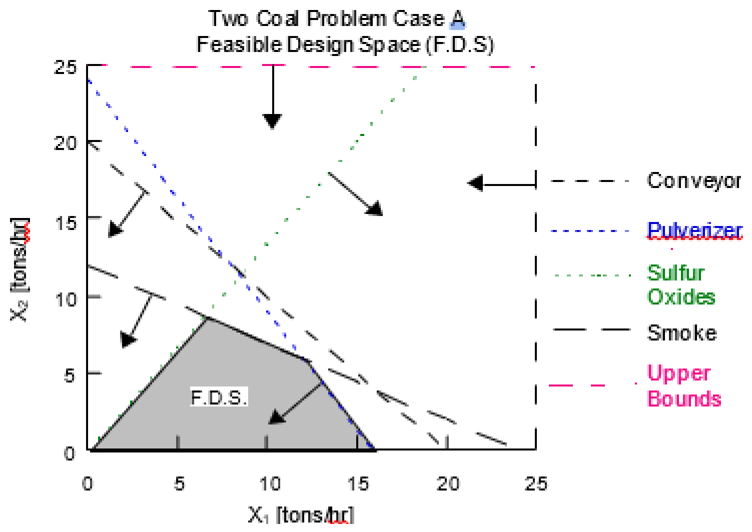


Figure I. 9: Case A feasible design space

The graphical solution is shown in Figure I.10. Pay particular attention to the following:

- ❑ The independent system variables are the axes of the design space.
- ❑ The system constraints and bounds form the feasible design space.
- ❑ The direction of feasibility is indicated, with arrows, on each constraint and bound. Experience has shown that many errors are avoided if students do not omit this simple step.
- ❑ The constraints and bounds are labelled in a way that makes it easy to refer back to the word problem and its mathematical form. A one-to-one correspondence should exist between the word problem, its mathematical form and the graphical solution. Experience has shown that the errors made by students are fewer if this is checked as a

matter of course.

- The best solution for the model, at which the objective has the highest value (when maximizing), is at a vertex of the feasible design space.
- The solution to the problem consists of not just the values of the system variables and the objective function but also the active and inactive constraints, etc. The solution is shown on the graph and a recommendation is made, as required, to management.

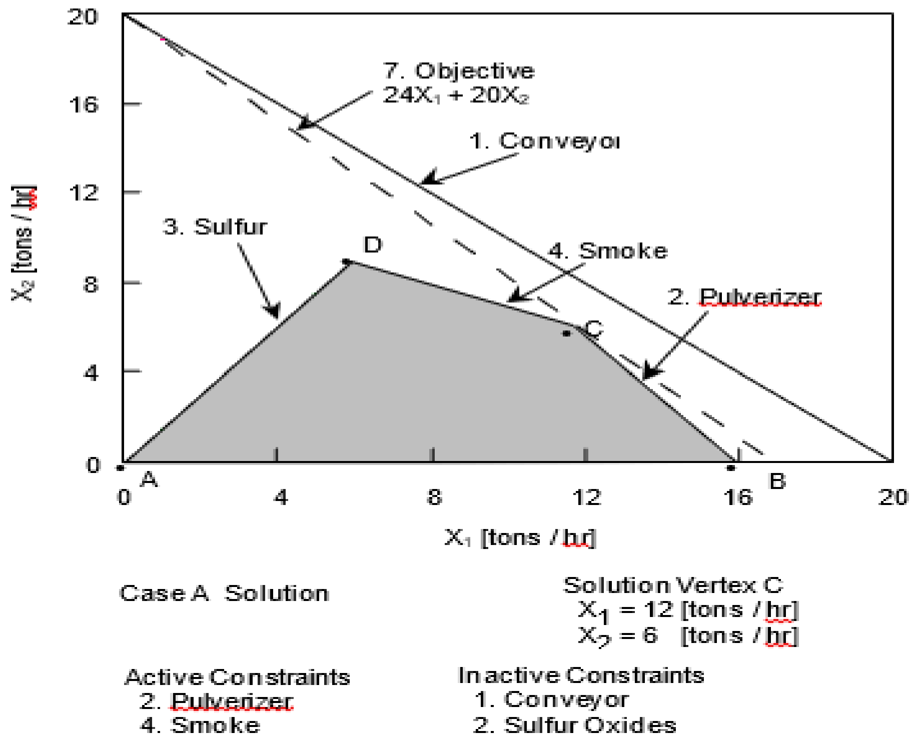


Figure I. 10: Case A solution

I.7.5 Case A: Recommendation

If 12 tons of Coal A and 6 tons of Coal B are burnt per hour, 408,000 lbs of steam will be generated per hour. This will result in the maximum amount of electricity being produced with all the constraints and bounds being satisfied.

The best solution for the model occurs at vertex C in Figure I.10. At vertex C the smoke constraint, constraint 4, and the pulverizer constraint, constraint 2, are active. The maximum amount of particulates that can be emitted into the air are being emitted and there is no reserve capacity for the pulverizer.

I.7.6 Case A: Post-solution Analysis

Post-solution analysis deals with the "What is the impact on ... if ..." questions. For example,

- ❑ What happens if there is a change in the coefficient of a variable in the objective function?
- ❑ What happens if there is a change in the right hand side of a constraint?
- ❑ What is the impact on the solution of adding a variable, i.e., another type of coal?
- ❑ What happens if one of the coefficients on the left hand side of a constraint changes?

The first three will be answered in this section.

Slack and surplus variables. For any feasible solution, the difference between the left hand side and the right hand side of the constraint is called the amount of slack (for ⍰ inequalities) or surplus (for ⍰ inequalities). In system constraints, this difference is represented by the inclusion of *slack* or *surplus* variables. For Case A, after the introduction of the slack and surplus variables, the mathematical form is as follows (note the form used for constraint 3):

Find

$X_1, X_2, S_1, S_2, S_3, S_4, S_5, S_6, S_7, S_8$

Satisfy

1. $X_1 + X_2 + S_1 = 20$ [tons/hr]
2. $1.5 X_1 + X_2 + S_2 = 24$ [hours]
3. $1{,}800 X_1/(X_1+X_2) + 3{,}800 X_2/(X_1 + X_2) + S_3 = 3{,}000$ [ppm]

4. $0.5 X_1 + X_2 + S_4 = 12$ [kg/hr]

5. $X_1 - S_5 = 0$ [tons/hr]

6. $X_2 - S_6 = 0$ [tons/hr]

7. $X_1 + S_7 = 25$ [tons/hr]

8. $X2 + S8 = 25$ [tons/hr]

Maximize

$Z = 24 X_1 + 20 X_2$ [1,000 lbs/hr]

Slack and surplus variables represent unused resource or capacity. If either the slack or surplus variable is zero for a particular constraint, then that constraint is *active.* If the slack or surplus variable for a constraint is nonzero, then the corresponding constraint is *inactive.* For Case A, with $X_1 = 12$ and $X_2 = 6$ tons/hour the slacks and surplus variables are:

Conveyer	$S_1 = 2$	[tons/hr]	Inactive
Pulverizer	$S_2 = 0$	[hours]	Active
Sulfur	$S_3 = 533.33$	[ppm]	Inactive
Smoke	$S_4 = 0$	[tons/hr]	Active

The nonzero slacks indicate the amount of reserve capacity or resources. For Case A, the amount of reserve conveyer capacity is 2 tons per hour and the additional amount of sulfur oxides that can be emitted into the atmosphere without penalty is 533 parts per million.

Change in the slope of the objective function. What happens if the values of the coefficients of the objective function change? Assume that the thermal value of Coal A is 32,000 pounds of steam per ton. The objective function changes to

$$Z = 32 X_1 + 20 X_2 \quad [1000 \text{ lbs/hr}].$$

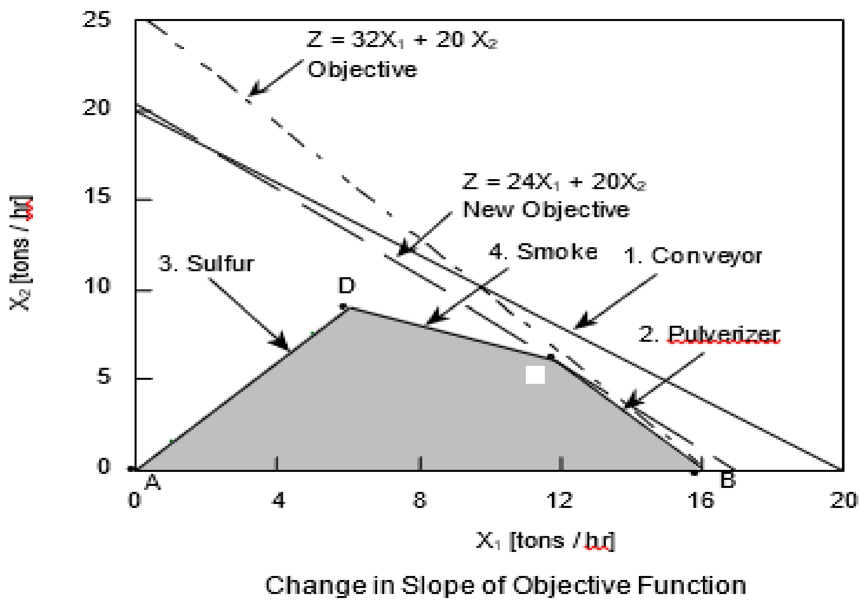


Figure I. 11: Change in slope of objective function

The change in the coefficient changes the slope of the objective function and if this slope is sufficiently large the solution will move to another vertex, Figure 4.11. This will alter the values of the system variables, the objective function and the slack and surplus variables. In Case A, the solution moves to vertex B. *Ranging* or *parametric analysis* of the objective function is the answer to the following question: By how much can we change the coefficient of the objective function and still keep the same solution? Ranging involves identifying the range of change of a coefficient for which the solution remains the same. For example, if C_1 is the coefficient of X_1, then the solution will be at vertex C or include vertex C as long as C_1 satisfies the following

$$10 \leq C_1 \leq 30$$

Change in the right hand side value of a constraint. Suppose the management is contemplating the installation of emission control equipment that would reduce smoke emission from the smoke stack by 25 percent. This would allow legal emission standards to be met by "uncontrolled" emission of smoke at the furnace of up to 15 kg/hr. How much would this be worth per hour in terms of steam output?

Assume, for the present, that the limit on particulate emission is raised by 1 kg/hr. In this case the right hand side of the smoke constraint goes from 12 to 13 and the smoke constraint becomes:

$$0.5X_1 + X_2 \leq 13.$$

As seen from Figure I.12, the solution moves from vertex C to vertex C'. The net change in the amount of steam produced is calculated as follows:

Old Solution Point C	*New Solution Point C'*	*Difference*	*Change in Z*
$X_1 = 12$	$X_1 = 11$	-1	(-1) 24
$X_2 = 6$	$X_2 = 7.5$	+1.5	(1.5) 20
			Net change in Z

The new value of the objective function is (408 + 6), i.e., 414. So, 414,000 lbs of steam is generated per hour. This change in value of the objective for a unit change in the value of the right hand side is called *imputed value, opportunity cost, shadow price, dual price or dual variable.*

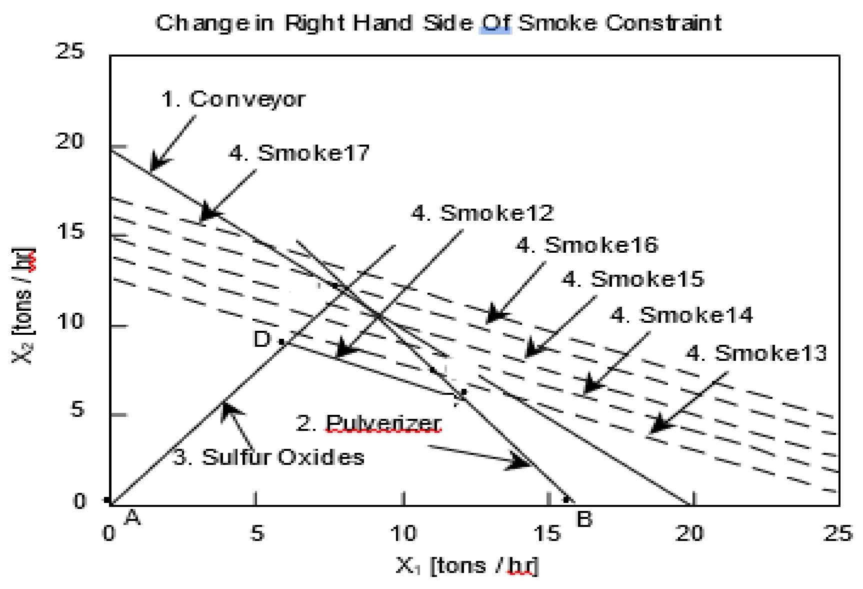


Figure I. 12: Change in right hand side coefficient

As the right hand side of the smoke constraint is further relaxed to 14, 15, etc., the value of the objective function continues to increase until a maximum steam production of 432,000 lbs. of steam per hour is reached at a right hand side value of 16. Further increase in the right hand side coefficient of the constraint has no impact on the value of the objective function since smoke constraint becomes inactive and the conveyer and sulfur constraints become active. The pulverizer constraint continues to remain active. At C", the imputed value for relaxing the smoke constraint goes to zero. The imputed value for tightening the smoke constraint is

-6.

Slacks, imputed values and insight. The imputed value for an active constraint is nonzero. For an inactive constraint it is zero and therefore the impact of the constraint on the objective, after a change in its right hand side, will remain zero. Therefore, it is adequate to compute the imputed values for the active constraints. These values provide insight into the stability of the solution from the standpoint of the active constraints. The slack or surplus variable is zero for an active constraint and nonzero for an inactive constraint. The nonzero slacks provide insight into the stability of the solution from the stand- point of the inactive constraints. For Case A, the information used to understand the stability of the solution is as follows:

Constraint	*Slack/Surplus*	*Imputed Value*	*Constraint Status*
Conveyer	2	0	Inactive
Pulverizer	0	14	Active[7]
Sulfur	533.33	0	Inactive
Smoke	0	6	Active

A constraint is said to be tightened if by changing the right hand side value of the constraint, the feasible design space is reduced. A constraint is said to be relaxed if changing the right hand side value increases the size of the feasible design space. With this by way of definition, what if it becomes necessary to reduce the design space (say because of maintenance of equipment), which constraint should be tightened? If on the other hand it is possible to increase the size of the feasible design space by investing in some equipment, which constraint should be relaxed?

The addition of another system variable.

Problem Statement: Plant management is evaluating the possible use of a third type of coal, Coal C. This coal has the following properties:

Pulverizer	1/20 hour pulverizer time/ton
Sulfur oxide emission rate	2,000 ppm
Smoke emission rate	0.8 kg/ton
Equivalent thermal value	21,000 lbs/ton.

The questions are:

- Should this coal be used? If no,

- ❑ What should be the properties of a coal that is likely to be selected?

The mathematical form for the Three Coal Problem follows.

Find

System Variables

X_1 - the rate of consumption of Coal A [tons/hr]

X_2 - the rate of consumption of Coal B [tons/hr]

X_3 - the rate of consumption of Coal C [tons/hr]

Satisfy

System Constraints

- ❑ Conveyer capacity

$X_1 + X_2 + X_3 \leq 20$ [tons/hr]

- ❑ Pulverizer capacity

$1.5 X_1 + X_2 + 1.2 X_3 \leq 24$ [hours]

- ❑ Sulfur oxides emission

$-1{,}200 X_1 + 800 X_2 - 1000 X_3 \leq 0$ [NMU]

- ❑ Smoke emission

$0.5 X_1 + X_2 + 0.8 X_3 \leq 12$ [kg/hr]

Bounds on System Variables

- ❑ Lower bounds on system variables

$X_1 \geq 0$ [tons/hr]

$X_2 \geq 0$ [tons/hr]

$X_3 \geq 0$ [tons/hr]

- ❑ Upper bounds on system variables

$X_1 \leq 25$ [tons/hr]

$X_2 \leq 25$ [tons/hr]

$X_3 \leq 25$ [tons/hr]

Maximize

- ❑ The rate of steam produced

$Z = 24 X_1 + 20 X_2 + 21 X_3$ [1000 lbs/hr]

The preceding can be solved by starting afresh or by using the imputed values from the Two Coal Problem solution, Case A.

Let us assume (arbitrarily) that 1 ton of Coal C is burnt per hour. This has the same effect of reducing the right hand sides of the system constraints as follows:

1. Conveyer capacity

$X_1 + X_2 \leq 20 - 1$ [tons/hr]

2. Pulverizer capacity

$1.5 X_1 + X_2 \leq 24 - 1.2$ [hours]

3. Sulfur oxides emission

$-1{,}200 X_1 + 800 X_2 \leq 0 + 1000$ [NMU]

4. Smoke emission

$0.5 X_1 + X_2 \leq 12 - 0.8$ [kg/hr]

The change in the value of the objective function on using 1 unit of Coal C is computed as follows:

Constraint	*Imputed value*	*Change in RHS*	*Change in Z*
1. Conveyer	0	-1	0
2. Pulverizer	14	-1.2	-16.8
3. Sulfur	0	1,000	0
4. Smoke	6	-0.8	- 4.8

Total change in steam output	-21.6
Steam produced by 1 unit of Coal C	21.0
Net change in steam output [1000 lbs/hr]	-0.6

Since the steam output decreases, Coal C is not competitive and should not be used. For Coal C to be competitive its thermal value should be greater than 21,600 lbs/hr.

I.7. 7 Case B: Formulation and Graphical Solution

The mathematical form of Case B is identical to that of Case A. In Case B, however, the lower bounds on the system variables are nonzero:

$X_1 \geq 5$ [tons/hr]

$X_2 \geq 5$ [tons/hr]

The solution space, for Case B, is shown in Figure I.13. Because of the nonzero lower bounds in Case B, the feasible design space in Figure I.13 is smaller than the feasible design space for Case A in Figure I.11. The best solution occurs at vertex C and is the same as that for Case A. The active constraints are also the same in both cases. Why then are the two cases being presented?

The reason is principally pedagogical. Case A is used to illustrate the method of *formulating* a linear single objective optimization problem so that the *pivoting* operations required for solving the problem using the pre-multiplication technique are possible. For Case B, it is assumed that a person knows how to pivot, the formulation is extended so that the pre-multiplication technique can be used to solve any linear single objective optimization problem.

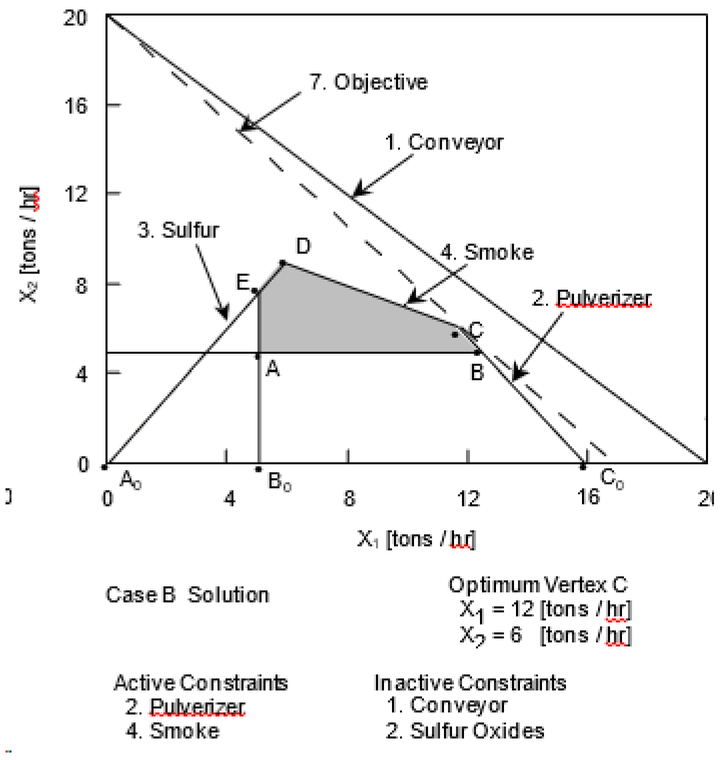


Figure I. 13: Case B solution

The Single Goal Compromise Decision Support Problem

In this section, the single objective optimization problem is reformulated and solved as a single goal compromise DSP. It will be shown that single objective optimization problems that call for the minimization or the maximization of an objective can be reformulated and solved as single goal compromise DSP.

I.7. 8 General Formulation

The linear optimization problem can be rewritten as a single goal compromise DSP. A target value is first assigned to the objective which can then be written as a goal. Then depending on the objective the appropriate deviation variable is included in the minimizing deviation function.

Find

The Independent System Variables

X1, X2

The Deviation Variables

d-, d+

Satisfy

System Constraints

$$a_{11}X_1 + a_{12}X_2 \leq b_1$$

$$a_{21}X_1 + a_{22}X_2 \leq b_2$$

$$a_{31}X_1 + a_{32}X_2 \geq b_3$$

System Goal (Normalized)

$(c_1/T)X_1 - (c_2/T)X_2 + d^- - d^+ = 1$ (T = Target value)

Bounds on System Variables

$$X_1 \geq X_1^{min}$$

$$X_2 \geq X_2^{min}$$

$$X_1 \leq X_1^{max}$$

$$X_2 \leq X_2^{max}$$

Minimize

The Deviation Function

$Z = d^- + d^+$ or d^- or d^+.

The formulation here is different from the single objective case in that it includes deviation variables and a system goal. Also the “objective” here is in terms of the deviation variables only. The target value T has to be set to an appropriate value.

I.7.9 Case C: The Word Problem

Given

Same as in Section for Case A.

Target value, T, of steam to be produced [lbs/hr]

Assumption

The maximization of the rate of electricity produced is equivalent to the maximization of the rate of steam generated.

Find

Independent System Variables

The rate of consumption of Coal A: X1 [tons/hr]

The rate of consumption of Coal B: X2 [tons/hr]

Deviations from the target amount of steam to be produced

d^- underachievement of the rate of steam production [-]

d^+ overachievement of the rate of steam production [-]

Satisfy

System Constraints

1. The capacity constraint on the conveyer unit.
2. The capacity constraint on the pulverizer unit.
3. The emission of sulfur oxides is limited.

4. The emission of particulates (smoke) is limited.

System Goal

5. It is desirable to achieve the target value of steam, T, to be produced.

Bounds on the System Variables

6. The system variables should not be less than a specified value.
7. The system variables should not exceed a specified upper limit.

Minimize

Underachievement of the steam production target, d-.

I.7.10 Case C: The Mathematical Form of the Word Problem

Given

As stated in the word problem

T= 432000 lbs/hr

Find

System Variables

X_1	- the rate of consumption of Coal A	[tons/hr]
X_2	- the rate of consumption of Coal B	[tons/hr]

Deviations from the Target Amount of Steam to be Produced

d^- underachievement of the rate of steam production

d^+ overachievement of the rate of steam production

Satisfy

System Constraints

1 Conveyer capacity

$$X_1 + X_2 \leq 20 \quad \text{[tons/hr]}$$

2 Pulverizer capacity

$$1.5X_1 + X_2 \leq 24 \quad \text{[hours]}$$

3 Sulfur oxides emission

$$-1200X_1 + 800X_2 \leq 0 \quad \text{[NMU]}$$

4 Smoke emission

$$0.5X_1 + X_2 \leq 12 \quad \text{[kg/hr]}$$

System Goal

Steam generation

$$(24{,}000/T)\ X_1 + (20{,}000/T)\ X_2 + d^- - d^+ = 1$$

Bounds on System Variables

$$X_1 \geq 0,\ X_2 \geq 0 \quad \text{[tons/hr]}$$

$$X_1 \leq 25,\ X_2 \leq 25 \quad \text{[tons/hr]}$$

Minimize

The deviation from the target rate of steam production, T.

$$Z = d^-$$

I.7.11 Case C: Graphical Solution

The solution space is shown in Figure 4.14. The following points are pertinent to the solution:

- The compromise solution is at point C in the figure.
- The rate of consumption of Coal A (X_1) is 12 tons/hour.
- The rate of consumption of Coal B (X_2) is 6 tons/hour.
- The pulverizer constraint, constraint 2, and the particulate emission constraint,

constraint 4, are active.

- The slack capacity of the conveyer constraint (S_1) is 2 tons/hour.
- The slack in the sulfur oxides emission limit is (S_3) 533.33 ppm.
- The target amount of steam cannot be generated without violating at least one of the other constraints. The shortfall of steam generated, d^-, is 24,000 lbs/hour. Therefore, only 408,000 lbs of steam can be generated without violating any of the constraints.

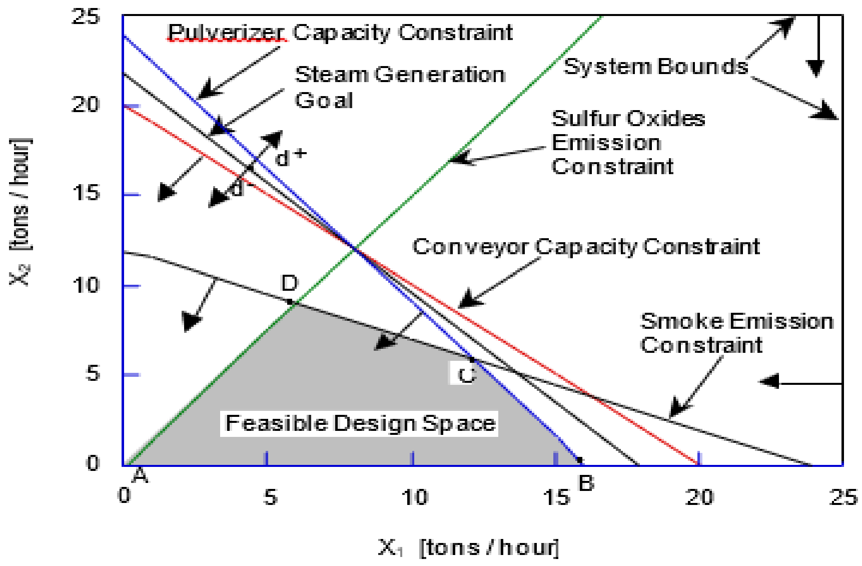


Figure I. 14: Case C design space

The Linear, Multigoal Compromise Decision Support Problem

I.7.12 General Formulation of the Linear, Multigoal Compromise DSP

Given

Same as in Section I.7. 8

Assumption

The maximization of the rate of electricity produced is equivalent to the maximization of the rate of steam generated.

Find

Independent System Variables

The rate of consumption of Coal A: X_1 [tons/hr]

The rate of consumption of Coal B: X_2 [tons/hr]

Deviations from the rate of sulfur oxides emission

d_1^- surplus capacity to emit sulfur oxides without penalty.

d_1^+ sulfur oxides emitted over specified limit.

Deviations from rate of smoke emission

d_2^- surplus capacity to emit smoke without penalty.

d_2^+ smoke emitted over limit.

Deviations from the target amount of steam to be produced

d_3^- underachievement of the rate of steam production.

d_3^+ overachievement of the rate of steam production.

Satisfy

System Constraints

1. The capacity constraint on the conveyer unit.
2. The capacity constraint on the pulverizer unit.

System Goals

3. The emission of sulfur oxides is limited.
4. The emission of particulates (smoke) is limited.
5. It is desirable to achieve the target value of steam, T, to be produced.

Bounds on the System Variables

6. The system variables should be greater than the specified lower limit.
7. The system variables should not exceed a specified upper limit.

Minimize

A function of the deviation variables. All goals have the same importance.

I.7.13 Case D: The Mathematical Form of the Word Problem

The derivation of all the constraints and the goals in the mathematical formulation has been covered in Section 4.5. Of special interest is the adjustment of the coefficients of constraint 3 to ensure that the deviation variables of all goals vary within the same range. The constraint was divided by 10 to make the coefficients on its left hand side of the same order as in the other system goals.

Find

System Variables

X_1 - the rate of consumption of Coal A [tons/hour]

X_2 - the rate of consumption of Coal B [tons/hour]

Deviations from the Target Rate of Sulfur Oxides Emission (normalized)

d_1^- surplus capacity to emit sulfur oxides without penalty

d_1^+ sulfur oxides emitted above limit

Deviations from the Target Rate of Smoke Emission (normalized)

d_2^- surplus capacity to emit smoke without penalty

d_2^+ smoke emitted above limit

Deviations from the Target amount of steam to be generated (normalized)

d_3^- underachievement of the rate of steam production

d_3^+ overachievement of the rate of steam production

Satisfy

System Constraints

Conveyer capacity $X_1 + X_2 \leq 20$ [tons/hour]

Pulverizer capacity $1.5X_1 + X2 \leq 24$ [hours]

System Goals

Sulfur oxides emission

$$-0.1X_1 + 0.0667X_2 + d_1^- - d_1^+ = 0$$

Smoke emission

$$(0.5/12)X_1 + (1/12)X_2 + d_2^- - d_2^+ = 1$$

or

$$0.0417X_1 + 0.0833X_2 + d_2^- - d_2^+ = 1$$

Steam generation (Target value, T = 432,000)

$$(24{,}000/T)\ X_1 + (20{,}000/T)\ X_2 + d_3^- - d_3^+ = 1$$

or

$$0.0556X_1 + 0.0463X_2 + d_3^- - d_3^+ = 1$$

Bounds on System Variables

$X_1 \geq 0$, $X_2 \geq 0$ [tons/hour]

$X_1 \leq 25$, $X_2 \leq 25$ [tons/hour]

Minimize

The deviation function

$Z = W_1 d_1^+ + W_2 d_2^+ + W_3 d_3^-$

where $W_1 + W_2 + W_3 = 1$ and $W_1 = W_2 = W_3 = W$.

The value of W = 0.33 is used in this case.

I.7.14 Case D: Graphical Solution

The design space for Case D is shown in Figure I.15. The feasible design space has been identified. In this case the solution lies at point P on the boundary of the feasible design space. The solution is the same as the one obtained in Cases A, B and D. The smoke emission goal is exactly satisfied at this point. The sulfur oxide emission and steam generation target values are underachieved. Note that because of fewer system constraints the feasible design space is larger.

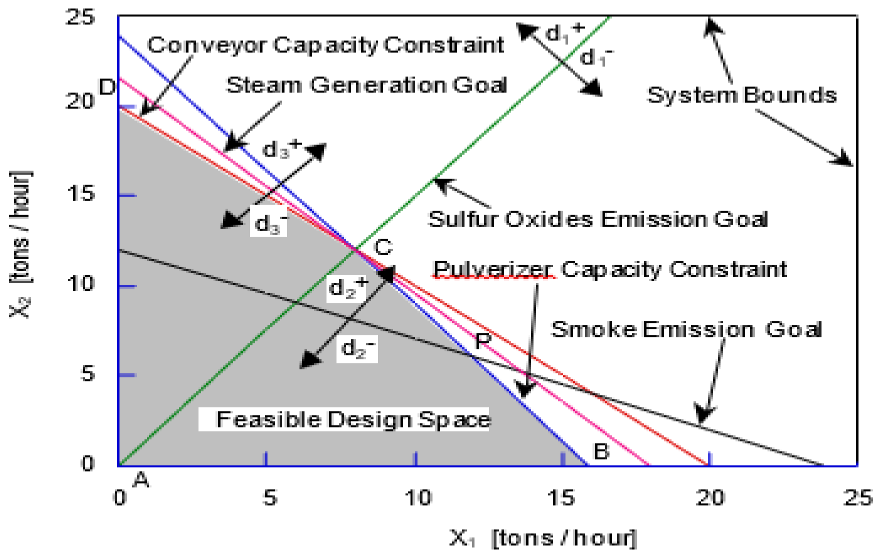


Figure I. 15: Case D solution space

The Nonlinear, Multigoal Compromise Decision Support Problem

I.7.15 Case E: Mathematical Formulation

The modification to the story introduces a new, nonlinear, constraint in the original formulation. The constraint deals with the upper limit on the stockpiling costs.

It reads:

$3X_1(X_1 - 10) + 4X_2(X_2 - 5) \leq 30$ (\$/hour)

The mathematical formulation for the nonlinear problem is given next. An additional system constraint is added and the lower bounds on the system variables are taken as zero.

Given

Same as for Cases A and B.

The penalties (gains) from stockpiling coal above (below) the desired limits. A maximum stockpiling cost of \$30/hour.

Assumption

The maximization of the rate of electricity produced is equivalent to the maximization of the rate of steam generated.

Find

System Variables

X_1	The rate of consumption of Coal A	[tons/hour]
X_2	The rate of consumption of Coal B	[tons/hour]

Deviations from the target rate of sulfur oxides emission (normalized)

d_1^-	Surplus capacity to emit sulfur oxides without penalty
d_1^+	Sulfur oxides emitted above limit

Deviations from the target rate of smoke emission (normalized)

d_2^- Surplus capacity to emit smoke without penalty

d_2+ Smoke emitted above limit

Deviations from the target amount of steam to be generated (normalized)

d_3^- underachievement of the rate of steam production

d_3^+ overachievement of the rate of steam production

Satisfy

System Constraints

Conveyor capacity

$$X_1 + X_2 \leq 20 \qquad \text{[tons/hour]}$$

Pulverizer capacity

$$1.5\, X_1 + X_2 \leq 24 \qquad \text{[hours]}$$

Stockpiling cost

$$3\, X_1 (X_1 - 10) + 4\, X_2 (X_2 - 5) \leq 30 \qquad \text{[\$/hour]}$$

System Goals

Sulfur oxides emission

$$-0.1X_1 + 0.0667X_2 + d_1^- - d_1^+ = 0$$

Smoke emission

$$0.0417X_1 + 0.0833X_2 + d_2^- - d_2^+ = 1$$

Steam generation

$$0.0556X_1 + 0.0463X_2 + d_3^- - d_3^+ = 1$$

Bounds on system variables

$X_1 \geq 0, \quad X_2 \geq 0$ [tons/hour]

$X_1 \leq 25, \quad X_2 \leq 25$ [tons/hour]

Minimize

The deviation function

$$Z = W_1 d_1^+ + W_2 d_2^+ + W_3 d_3^-$$

$W1 + W2 + W3 = 1$ and $W1 = W2 = W3 = W$ (= 0.33)

I.7.16 Case E: Graphical Solution

The design space for this problem is shown in Figure 4.16. The feasible design space is shown by hatched lines. The new constraint, constraint 3, is also shown. The graphical solution is obtained by linearizing constraint 3. The method for linearizing equations is described in greater detail in Volume 2.

Step 1

Rewrite constraint as $f(X) \geq 0$

$$f(X) = 30 - 3X1(X1 - 10) - 4X2(X2 - 5) \geq 0$$

Step 2

Choose an initial starting point, X^o

$$X^o = \{ X_1^o = 0, X_2^o = 0 \}$$

Step 3

Evaluate the following coefficients at X^o

$A = f(X) = 30$

$B1 = \partial f(X)/\partial X_1 \;|_{\underline{x}^0} = 30$ $\qquad B2 = \partial f(\underline{X})/\partial X_2 \;|_{\underline{x}^0} = 20$

$C_1 = \partial^2 f(X)/\partial X_1^2 \;|_{\underline{x}^0} = -6$ $\qquad C_2 = \partial f(X)/\partial X_2^2 \;|_{\underline{x}^0} = -8$

Step 4

Evaluate secant plane derivatives

$a_1 = (AC_1/B_1)/(1 - (1-2AC_1/B_1^2)^{0.5}) = 32.748$

$a_2 = (AC_2/B_2)/(1 - (1-2AC_2/B_2^2)^{0.5}) = 24.832$

Step 5

This step is skipped because the roots are real.

Step 6

Evaluate the right hand side of the linearized constraint.

$b = a_1X_1^o + a_2X_2^o - A = -30$

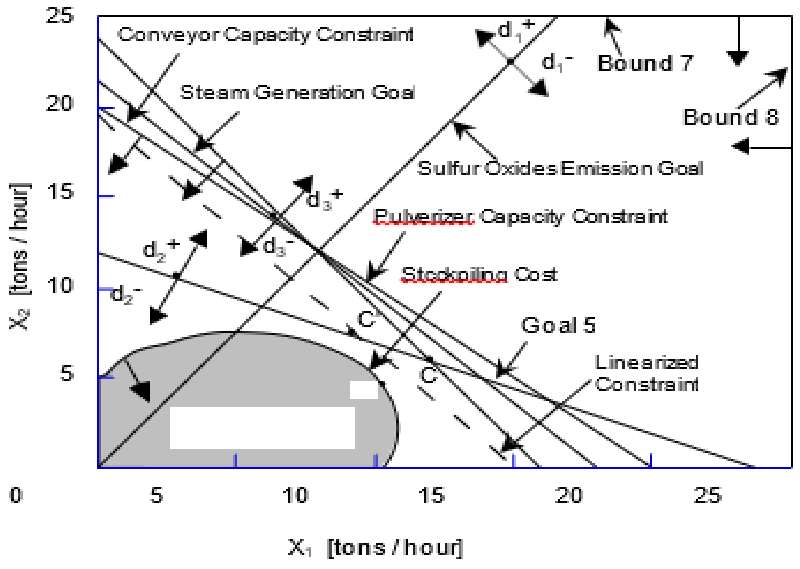


Figure I. 16: Case E design space

Step 7

Establish the linearized constraint

$$a_1X_1 + a_2X_2 \geq b$$

i.e.,

$$32.748\ X_1 + 24.832\ X_2 \geq -30$$

This constraint, when plotted in the design space makes the entire first quadrant of the design space feasible and therefore is redundant.

Step 8

Choose C, Figure I.16, as the next initial point.

$$X^o = \{\ X_1 = 12,\ X_2 = 6\ \}$$

Step 9

Evaluate the following coefficients.

$$A = -66$$

$$B1 = -42 \qquad B2 = -28$$

$$C1 = -6 \qquad C2 = -8$$

Step 10

Evaluate the secant plane derivatives.

$$a_1 = -36.588$$

$$a_2 = -18.857/(1- (-0.3469)^{0.5})$$

Step 11

Are a_1 and a_2 real?

a_2 is imaginary Set $a_2 = B_2 = -28$

Step 12

Evaluate the right hand side of the linearized constraint. Hence,

$$b = -541.06$$

Step 13

Establish the linearized constraint, viz.,

$$36.588\ X_1 - 28\ X_2 \geq -541.06 \quad \text{or}$$

$$36.588\ X_1 + 28\ X_2 \leq 541.06$$

This constraint is plotted, line 3', in Figure 4.16. As determined by the set of linearized constraints, the optimum is found to be at C' = {X_1 = 9.08, X_2 = 7.46}. However, this solution is approximate. It is in the vicinity of the optimum. To obtain a more accurate solution, a new starting point needs to be chosen and steps 1 through 8 repeated to obtain solutions close to the actual optimal. Point C'' {X_1 = 10, X_2 = 5.75} is determined to be the true optimum. The algorithm is cumbersome when calculations are done by hand.

I.8 DSIDES Software

I.8.1 History of DSIDES 1976 to 2023

The acronym DSIDES stands for Decision Support In the Design of Engineering Systems. It was developed to provide a comprehensive decision support environment for all decision makers, whether engineers or not. The foundation for DSIDES is the Decision Support Problem Technique (DSPT) as developed under the direction of Farrokh Mistree at the Systems Design Laboratory, University of Houston[1] . In turn, the DSPT is itself built on the fundamental paradigm that design is a decision-based activity. Much has been written about the DSPT and Decision-Based Design (DBD) . [2,3,4]

In 1974, Farrokh Mistree started as a Post-doctoral research fellow under the mentorship of the late Professor Owen F. Hughes at the University of New South Wales, Sydney, Australia. The Sequential Linear Programming 2nd order (SLIP2) algorithm was jointly developed by Owen Hughes and Farrokh Mistree during their years of research collaboration (1974-1980) at the University of New South Wales in Sydney, Australia.

The SLIP2 algorithm was specially developed for a ship structural optimization program called AUSTROSHIP. The development of AUSTROSHIP was funded by the Department of Defence (NAVY), Canberra (1974-1976). The American Bureau of Shipping (ABS) continued support for the ship structural optimization work (1977-1980). The ABS sponsored structural optimization program is called SHIPOPT. In 1979 a grant was received from the Australian Research Grants Committee to develop a stand-alone version of the optimization program. H.B. Phuoc was supported by these funds and he collaborated with Farrokh Mistree in the development of the stand-alone version of SLIP2 (which now stood for Sequential Linear Programming 2nd generation).

Many people have contributed to the development of SLIP2 over the years. The first attempt to develop this algorithm was made by Brian Morley an undergraduate student at the University of New South Wales under the supervision of Dr. Owen Hughes in 1973-74. Morley, in his undergraduates honors thesis, identified the problems that needed to be solved. Rigby Gilbert, in his honors thesis (1976) provided insight which lead to the use of a "convex approximation" for modeling the nonlinear constraint function. Dr. Vedran Zanic (University of Zagreb) contributed significantly to overcoming some major obstacles.

- particularly in the area of data management. He also contributed the routines to formulate the Simplex tableau and parts of the input routine (1976).

The multi-objective feature was introduced by Farrokh Mistree in 1980. This work was started and completed at the University of New South Wales in 1980. Some refinements were introduced by Farrokh Mistree after he moved to the University of Houston in 1981. In 1983, Tim Lyon (Department of Defense (Navy), Canberra), assisted in the validation of this feature and also introduced the feature for post-solution analysis, while he was a graduate student at the University of Houston. In 1985, the feature to solve hierarchical decision support problems was introduced and the SLIPML (Sequential Linear Programming: Multi-Level) algorithm was born. The multilevel feature was validated by Warren Smith (Department of Defense (Navy), Canberra), Azim Jivan and Jon Shupe.

In 1979 Andrew Cawsey, for his undergraduate honors thesis at the University of New South Wales under the supervision of Farrokh Mistree, developed a stand-alone program (Program SELECT) for solving selection decision support problems. Saiyid Kamal was responsible for the integration of DSIDES into a single unit. In 1987, Eduardo Bascaran and Harshavardhan Karandikar implemented the feature for automatically determining a suitable starting solution. This feature was refined by Ravi Reddy in 1988. In 1988 Eduardo Bascaran wrote the code for the multiplex algorithm and integrated it into DSIDES [5] . This algorithm is used to solve the true-Preemptive formulation (not the pseudo- preemptive formulation) of the compromise and coupled DSPs.

Adaptive linear programming was introduced by Bert Bras in 1989 [6] . This was based on some work done at MARIN, (Maritime Research Institute Netherlands, Wageningen, Netherlands). Bert Bras and Harshavardhan Karandikar made a number of modifications to improve the robustness and convergence speed of the ALP algorithm. In 1990 Stan Abeln created a graphic post-processor for the SUN based on the X-Windows system.

The UH team transformed the card based input to terminal based input, provided options for various levels of output, and included XPLORE [7] to find a good starting solution for DSIDES. The UH team standardized the code to FORTRAN 77 and ensured that the code ran on the mainframe at the University of Houston and also on the Vax/VMS and later on the SUN/OS/UNIX mini computers. The UH team made it possible to solve not only the compromise Decision Support Problem (cDSP) but also the selection DSP (sDSP), the heuristic DSP (hDSP), and the coupled Decision Support Problems using the Archimedean and Preemptive formulations of the cDSP in a synergistic manner.

In April 1991, it was decided to undertake a major rewrite of the entire program. This represented the first major restructuring of the program since its inception in 1976. The changes implemented in this rewrite were the creation of simpler formats for user files, the modularization of the output, the use of the MULTIPLEX algorithm as the primary solver, the incorporation of a robust adaptive reduced move limit scheme for improved convergence, and a feature for preliminary exploration of the design space. This process was coordinated by Warren Smith, with Bert Bras serving as a consultant to the group. The rewrite was carried out by Warren Smith and Ravi Reddy, with Srinivas Vadde creating the new output routines. The new data input modules were designed and created by Ravi Reddy. The adaptive reduced move limit algorithm was implemented by Ravi Reddy. The option for creating tab-delimited text files (for exporting to spreadsheet programs) was created by Warren Smith. The new interface for coupled problems was created by Warren Smith. The sensitivity analysis for selection problems was added by Ravi Reddy. Uwe Lautenschlager performed a lot of testing during this phase.

In March 1992, a manual for DSIDES 4.0 was released. This manual represents the work of many people. Ravi Reddy coordinated its creation. The authors and contributors in alphabetical order are Bert Bras, Wei Chen, Stein Erikstad, Samir Karandikar, Badrinath Krishnakumar, Uwe Lautenschlager, Aashish Malhotra, Farrokh Mistree, Sangram Mudali, Bharat Patel, Rama Pakala, Ravi Reddy, Warren Smith, Srinivas Vadde.

In the late nineties, Mathew Marston (graduate student at the Georgis Institute of Technology) created a stripped down version of DSIDES in Java to run on an IBM PC. Lin Guo (University of Oklahoma 2021), in her doctoral dissertation, added the following functionalities to DSIDES [8] :

- Adaptive Linear Programming with Parameter Learning (ALPPL) to replace heuristics with knowledge about the behavior of the system.
- Adaptive Leveling-Weighting Clustering Algorithm (ALWC) a method to facilitate subsystem reorganization.
- Adaptive Scenario Planning a method to capture emergent properties of a complex system while it is being designed.

She also created videos to help novices learn how to prepare input files.

In 2023, Sara Hajihashemi, for her MS thesis to facilitate ease of input created a wrapper for DSIDES. To be consistent, DSIDES is used by Sara Hajihashemi in her thesis to refer to various versions and manifestation of SLIPML.

Upon translating the problem statement into a mathematical formulation within the cDSP construct, this mathematical representation can be seamlessly incorporated into the DSIDES software to obtain results. Notably, DSIDES is coded in FORTRAN. In this thesis, we design a user-friendly wrapper for DSIDES, aiming to harness the computational speed advantages of FORTRAN while ensuring user-friendliness akin to Excel. Additionally, we are establishing a connection between Excel and MATLAB for post-processing analysis; all details are discussed in Part 2 of this thesis.

To gain a clearer understanding of my thesis contribution and to compare the initial DSIDES version with the DSIDES Wrapper I have developed, it is essential to delve more into the history of DSIDES.

I.8.2 DSIDES: A Historic Compiled Program in FORTRAN

The Age of Batch Processing

DSIDES was developed in FORTRAN in the early 70 on a CDC 6600 mainframe with 120k RAM. The program was written for batch processing. On account of 120k RAM the program was heavily overlayed. The program was written on cards, complied and the complied version was executed. The data input was also via cards. The CDC 6600 was in use in the 1960s and early 1970s. It was one of the fastest computers of its time. The picture of CDC 6600 is shown in Figure I.17.

Figure I. 17: Computer CDC 6600 | Old computers, Old technology
https://www.pinterest.com/pin/10907224075942812

In this context:

Compilation: DSIDES, like many programs of its time, was compiled. The source code, written in FORTRAN, was transformed into an executable file using a compiler. By this process, the computer's processor could execute the program without the need for interpreting the source code each time it ran.

Memory Limitations: DSIDES operated under severe memory constraints. Mistree and Hughes[2] had a mere 120 K (kilobytes) of memory at their disposal, and through determined effort, they

[2] In 1974, Farrokh Mistree started as a Post-doctoral research fellow under the mentorship of the late Professor Owen F. Hughes at the University of New South Wales, Sydney, Australia. The early version of DSIDES was called SLIPML and was co-developed by Hughes and Mistree. SLIPML was an integral part of AUSEVAL a ship structural design program. AUSEVAL has morphed into MAESTRO a commercial naval ship and submarine program being used by several navies around the world. In the early eighties, SLIPML was extracted from AUSEVAL and transformed into DSIDES by Warren F. Smith and Bert A. Bras (PhD students at the University of Houston). To be consistent DSIDES is used in this thesis to refer to various versions of SLIPML.

managed to secure an extra 4 K for overnight use. By modern standards, this amount of memory is extraordinarily limited, emphasizing the ingenuity needed to make the most of it.

Compilation Overlays: Overlay techniques were employed by the compiled version of DSIDES to operate efficiently within these memory limitations. This involved loading different segments of the program into memory as needed, swapping them in and out to make the best use of the available space. This was a challenging and meticulous process.

Handling the Physical Medium: Punch Cards

DSIDES operated during an era where computer input and output were managed through punched cards. A figure of punch card machine is presented in Figure I.18.

FigureI. 18: Punch Card in Punch Card Machine
(https://www.computerhope.com/jargon/p/punccard.htm)

This presented its own set of unique challenges:

Punched Card Deck: To run DSIDES, a deck of punched cards was prepared as the program input. Each card contained specific instructions or data, and the order of the cards was crucial for the compilation and correct execution of the program. A stack of IBM FORTRAN punch cards is represented in Figure I.19.

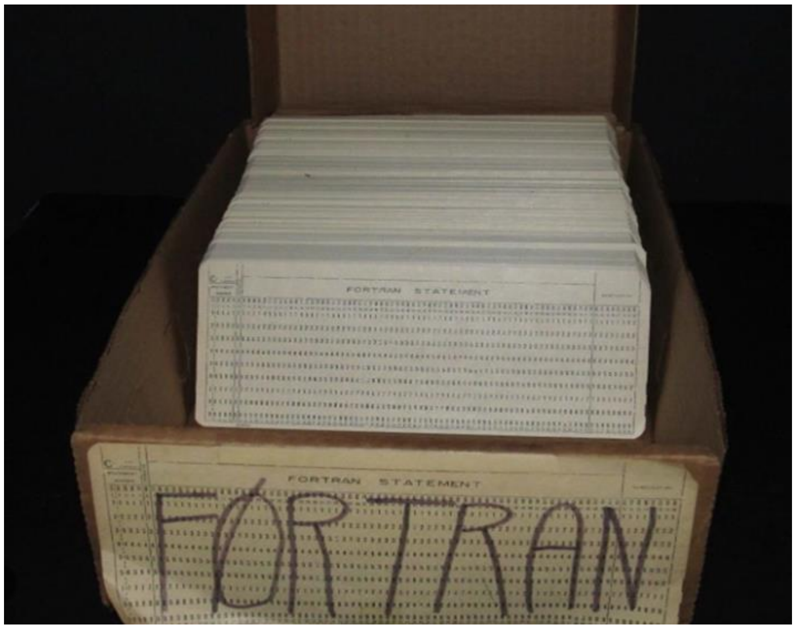


Figure I. 19: IBM FORTRAN Punch Cards (https://www.are.na/block/1023150)

Fragility: Punched cards were fragile and required careful handling. Dropping or mis-ordering the cards could lead to errors during program execution, making the process laborious and error-prone.

As previously mentioned, DSIDES, originating in the 1970s, was initially coded in the punch card format. Let Us Let us delve into the world of programming with punched cards to gain a deeper understanding of this unique and historical method of software development.

I.8.3 Programming with Punched Cards

In the early days of computer programming, especially in the 1960s and 1970s, programmers used punched cards for writing and storing computer programs. The historic method of coding and inputting computer programs using physical cards with holes punched in them is referred to as programming with punch cards. These punched cards contained holes that represented instructions or data and were used to input programs and data into early computer systems.

Punched Card Format

Punched cards were typically 80 columns wide and 12 rows tall. Each column could represent a character, digit, or a control character. FORTRAN program statements had this format:

Column 1 – Comment - usually marked with a "C"

Column 1-5 - Line Number – usually used with format statements

Column 6 - Continuation of statement on previous card

Column 7-72 - the actual FORTRAN code (latter systems used 7-80)

Column 72-80 – Sequence or Identification number

Creating Punched Cards

- **Writing Code:** Programmers wrote their code on coding sheets, with each line of code corresponding to a single punch card. A figure of punch card is shown in Figure I.20.
- **Punching the Cards:** After writing the code, it was transferred to punch cards using keypunch machines. Programmers created holes in the cards, representing binary data. The pattern of holes encoded the instructions and data.

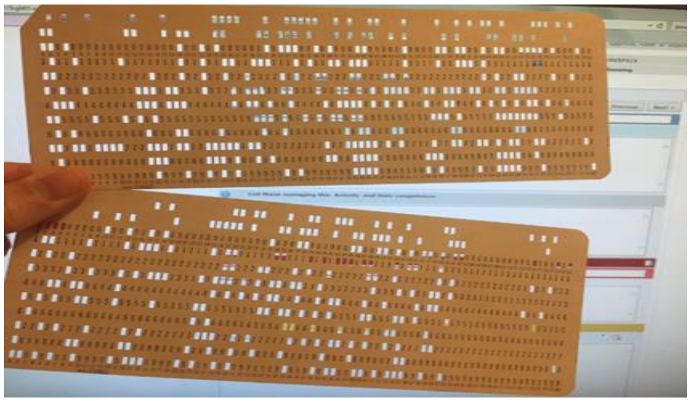


Figure I. 20: Punch Card (https://twitter.com/dannsimmons/status/1063412986189094912)

Compilation and Execution

Once the punch cards were ready, they were fed into card readers, and the code was compiled or interpreted by the computer. This process turned the punched holes into executable instructions.

Programming Process

- **Card Sorting:** Programs were typically written on stacks of punched cards, with each card representing a line of code. The sequence of cards determined the program's flow. To make changes to the program, programmers created new cards or updated existing ones and re-sorted the entire deck of cards.
- **Card Numbering:** It was common practice to number each card to maintain the correct sequence. Card numbers were typically in columns 73-80, used for debugging and version control.
- **Languages Used:** Various programming languages were used with punch cards, including assembly language, FORTRAN, COBOL, and others, depending on the computer and the application. FORTRAN was a popular language used with punched cards.

Challenges and Handling

- **Error Handling:** If a card were damaged, misplaced, or dropped, it could result in errors when running the program. Therefore, careful handling of the cards was essential.
- **Deck Management:** Managing a deck of punched cards was a critical aspect of programming. Large programs could result in stacks of cards that needed to be kept in order, making it easy to introduce errors when making changes.

- **Batch Processing:** Early computers operated in batch mode. Programmers submitted their card decks to a computer operator who fed the cards into the computer for processing. The results, often printed output, were returned to the programmer at a later time.
- **Debugging:** Debugging programs was a laborious process. If an error was discovered, the programmer had to locate the erroneous card, correct it, and then recompile the program, often by generating a new deck of cards.

The limitations and the unique challenges associated with punched card programming, such as the specific card format and the need to avoid card mishandling, were characteristic of this era. However, punched card systems were eventually replaced by text-based programming on terminals and personal computers, significantly simplifying the programming process and eliminating many of the inconveniences associated with punched cards.

I.8.4 Comparing the DSIDES Versions: Evolution and User Experience

Following the initial release of DSIDES, a series of iterative improvements were made. The culmination of these enhancements, which introduced the Preemptive form into the software, occurred in 1982. The DSIDES wrapper has been crafted based on DSIDES 1982. In this section, a concise comparison between the DSIDES 1982 and the modern DSIDES Wrapper 2023 is provided. The significant improvements in user experience and functionality offered by the DSIDES Wrapper are underscored by this comparison.

Input File Preparation and Interface

In the DSIDES 1982, users faced the challenge of preparing input files within plain Notepad documents. This process lacked descriptions and required users to refer to multiple documents to ensure they considered all necessary data blocks. In contrast, this process has been streamlined by the DSIDES Wrapper through the adoption of an Excel-based interface. All required data blocks are readily available, and each cell is accompanied by a description, making it easier for users to understand and input data accurately.

Execution and User-Friendliness

Running the DSIDES 1982 was a multi-step process. Users had to save Notepad files in specific folders, obtain file paths, navigate through command prompts, and adhere to specific file structures. This complex process is simplified by the DSIDES Wrapper through the implementation of a user-friendly interface. Users can now achieve the same results by merely clicking a "Run" button, eliminating the need for intricate command-line operations.

Normalization and Post-Processing

In the DSIDES 1982, after obtaining results, users faced additional challenges. They had to manually calculate normalized values for each scenario (including 13 different scenarios) and then switch to MATLAB to import data and create ternary plots for post-processing analysis. In contrast, the normalization process is automated by the DSIDES Wrapper, and a seamless connection to MATLAB is established. With just one click, users can generate colorful or black-and-white ternary plots, providing a more efficient and user-friendly post-processing experience.

In summarizing, a substantial enhancement is represented by the DSIDES Wrapper over the original DSIDES 1982 version. Notably, the following changes are deemed pivotal in this transformation:

Reducing the Number of Steps in the Process:

DSIDES 1982:

1) Generate a plain-text document with linear information, ensuring adherence to the

specified block structure and input data order crucial for the software. Save the file at the designated location in the ".DAT" format.

2) Generate a text document containing nonlinear information and save it to a specified location with the file extension ".f."
3) Navigate to the specific address on your server (Computer/LocalDisk/MinGW/msys).
4) Select the file named 'msys.'
5) In the popped-up command window, type 'cmd' and press 'Enter.'
6) Navigate to your problem's directory by typing 'cd XXXX' and pressing 'Enter' (where 'XXXX' is the path to your problem).
7) Run DSIDES to solve the model by typing 'runalp XXXX' and pressing 'Enter' (where 'XXXX' is your model's name).
8) Return to the main folder to find the output file.

DSIDES Wrapper 2023:

1) Open the main folder and select the Excel file located in this folder.
2) Fill out the Linear sheet and click the RUN button. All necessary information and descriptions are provided in the sheet.
3) Fill in the Nonlinear sheet and click the RUN button. The output file will appear on the screen.

In summary, the steps to obtain the final result for cDSP have been reduced from 8 in DSIDES 1982 to 3 in DSIDES 2023.

Minimizing the Occurrence of Errors:

In DSIDES 1982, users were prone to errors during command prompt operations, file imports, and when filling out the linear and nonlinear sheets due to specific formatting requirements. These challenges are addressed by DSIDES 2023, with potential mistakes being eliminated through the provision of a user-friendly interface that standardizes input formats and is accompanied by detailed instructions.

Enhancing Time Efficiency:

The reduction of steps and the addition of information and descriptions to the linear and nonlinear sheets in DSIDES 2023 result in a significant enhancement of time efficiency. Users can now navigate through the streamlined process more swiftly, reducing the time required for input preparation and execution.

Facilitating Post-Process Analysis:

DSIDES 2023 streamlines post-process analysis by automating normalization and simplifying connectivity to MATLAB. The generation of ternary plots is facilitated with a single click, providing a more efficient and user-friendly experience for users engaged in post-processing tasks.

Overall, data input, execution, and post-processing tasks are simplified, resulting in an enhanced overall user experience and making it a valuable tool for engineering systems in the modern era. Having highlighted the significant enhancements in the DSIDES Wrapper, let us now delve into my key contributions as “I statement” in making these improvements a reality.

I Statement

Learning FORTRAN Language

One of the foundational steps in understanding DSIDES was learning the FORTRAN programming language. Since the program was originally coded in FORTRAN, I needed to familiarize myself with its syntax and structure. I acquired this knowledge, allowing me to comprehend the different subroutines and intricacies of the program.

Understanding Punch Card Logic

To bridge the gap between the archaic punch card era and modern computing, I had to schedule the logic of the program. This involved understanding how programming was executed through punch cards. Importantly, I did not modify the punch cards themselves; instead, I created a front-end interface designed to seamlessly integrate with this legacy format.

Learning VBA Language

Visual Basic for Applications (VBA) was another essential skill I had to acquire. I developed the user-friendly wrapper within Microsoft Excel, enhancing the program's accessibility and ease of use with VBA.

Connecting Excel and FORTRAN

DSIDES is compiled, making it impossible to make direct alterations to the original code. To overcome this challenge, I had to learn how to establish a connection between Excel and FORTRAN. I enabled data transfer and interaction between the two platforms, preserving the integrity of the original DSIDES code.

Integrating Excel and MATLAB

In addition to connecting Excel and FORTRAN, I learned how to integrate Excel with MATLAB for post-process analysis. I further extended the capabilities of DSIDES by enabling the analysis and interpretation of results.

In summary, my work on the DSIDES Wrapper encompassed an adequate learning journey, from mastering the FORTRAN language to understanding the logic of punch cards and effectively connecting Excel with FORTRAN and MATLAB. A more user-friendly and accessible version of DSIDES has been achieved through these efforts, granting engineers access to a powerful tool for addressing complex design challenges in the modern era.

I.9 PDSIDES

DSIDES will fit in the PDSIDES platform. PDSIDES, the **P**latform for **D**ecision **S**upport **i**n the **D**esign of **E**ngineering **S**ystems, is a smart system designed to help designers work faster and create better, cost-effective designs. The tricky part of making decisions during the design process, especially when dealing with complex engineering systems, is addressed by PDSIDES. Previous work that employed templates and ontologies to capture decision-related knowledge is built upon PDSIDES. But now, it takes these templates and turns them into a handy computer tool. This tool is useful for different users involved in design work, whether creating, editing, or implementing templates. Various design scenarios, like starting from scratch, adapting existing designs, or making variations, are covered by it.

Its worth is demonstrated by PDSIDES through its successful assistance in the design of a hot rod rolling system (HRRS), illustrating how design work can be made more efficient and top-notch results can be ensured without breaking the bank.

An overview of PDSIDES is illustrated in Figure I.21. PDSIDES is divided into three parts: knowledge, users, and decision-based design. What follows is the description of the platform from the bottom up, including how these three parts are connected to enable the functionalities. (Ming, Z., Nellippallil, A. B., Yan, Y., Wang, G., Goh, C. H., Allen, J., & Mistree, F. July 2018, PDSIDES—A Knowledge-Based Platform for Decision Support in the Design of Engineering Systems. Journal of Computing and Information Science in Engineering, DOI: 10.1115/1.4040461.

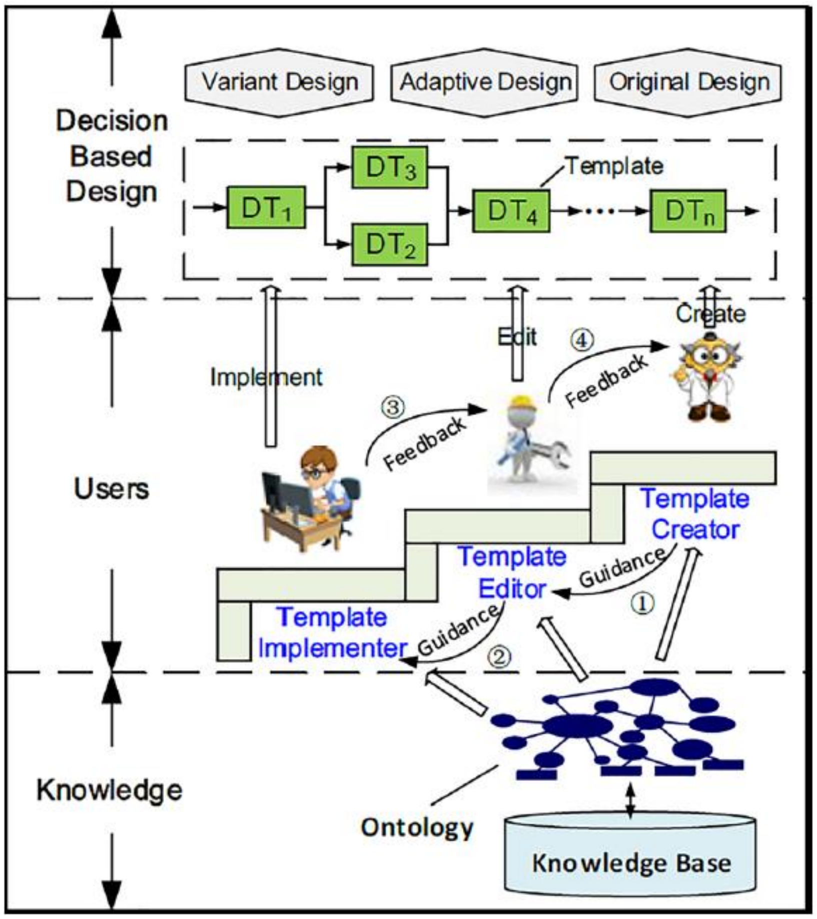


Figure I. 21: PDSIDES Overview

I.10 The Path Forward: Future Plans for Wrapper Improvement

In response to the inherent limitations of traditional visualization methods, including Ternary Plots, which are characterized by a restriction in representing conflicting goals to only three variables, a commitment has been made to advancing the capabilities of the wrapper, extending to addressing challenges associated with other prominent techniques. Different types of visualization techniques and their limitation has been represented in Figure I.22.

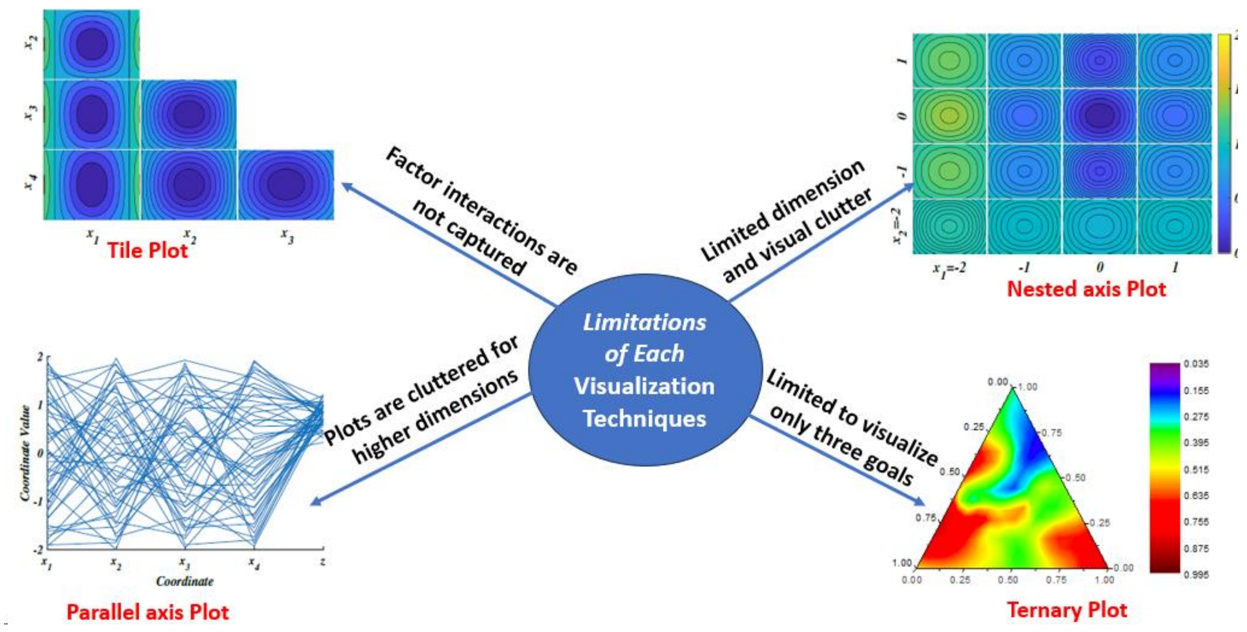

Tile Plot
Nested axis Plot
Parallel axis Plot
Ternary Plot
Limitations of Each Visualization Techniques
Factor interactions are not captured
Limited dimension and visual clutter
Plots are cluttered for higher dimensions
Limited to visualize only three goals
Coordinate Value
Coordinate

Figure I. 22: Different Types of Visualization Techniques

Notably, the relationship between two categorical variables is displayed using rectangular tiles in a tile plot, also known as a mosaic plot, where the proportion of data in each category is represented by the area of each tile. However, there are some limitations, such as challenges in interpretation when there are numerous categories, and it may not be well-suited for visualizing relationships between more than two categorical variables.

Also, the Parallel Axis Plot, a method employed for visualizing multivariate data through multiple parallel axes, has limitations in the form of potential clutter when faced with numerous variables. The resulting complexity may make interpretation difficult, especially in the context of complex datasets.

Similarly, challenges are faced as the number of nested variables increases in the Nested Axis Plot, despite its effectiveness in showcasing relationships between multiple variables through nested axes. Increased visual complexity may be introduced, potentially complicating the interpretation, particularly in scenarios with a substantial number of nested variables.

Moreover, the Ternary Plot, specifically designed for three variables, has constraints when applied to datasets exceeding this dimensionality. This limitation has restricted its utility for scenarios involving more than three conflicting goals, thereby underscoring the necessity for a more adaptive and encompassing visualization approach.

In light of these considerations, a machine learning-based visualization technique, specifically the interpretable self-organizing maps (ISOM), is incorporated into the future path. Positioned as an unsupervised neural network structure, excelling at transforming high-dimensional data into a two-dimensional space is a characteristic of ISOM. Through this strategic integration, users are granted access to a robust and interpretable visualization platform capable of accommodating and elucidating multiple conflicting goals simultaneously, with the aim of overcoming the limitations posed by traditional methods. A proactive response to the challenges posed by existing visualization techniques is indicated by the adoption of ISOM, ensuring a more comprehensive exploration of complex solution spaces within the wrapper.

I.11 Summary and Way Forward

In Part 1 of this thesis, We focus on the introduction of the compromise Decision Support Problem (cDSP) construct and DSIDES. Moving forward, in Part 2 of this thesis, our focus is on the practical implementation of a user-friendly interface for DSIDES. The implementation includes the provision of a user manual, guiding users through the functionalities and features of the wrapper. Users can interact seamlessly with the DSIDES wrapper and effectively utilize it to address complex decision problems.

Furthermore, the interpretation of output results from DSIDES is discussed, enabling users to analyze and comprehend the solutions provided by the DSIDES wrapper. The post-process analysis in the decision-making process is explained to enhance users' understanding of the results generated by DSIDES. To validate the effectiveness of the wrapper and demonstrate its capabilities, three example problems are presented.

In conclusion, in Part 1, we learn how to formulate the problem in the form of mathematics in cDSP construct; in Part 2, we implement and convert math formulation from Part 1 to the template. Initially, we focus on mastering the skill of translating problem statements into the cDSP construct. Subsequently, we apply this understanding within the DSIDES wrapper for implementation.

Part Two: Designing the User-Friendly Wrapper for DSIDES

Overview of Part 2

In the second part of this thesis, the main focus is on introducing a user-friendly wrapper that has been designed and implemented for DSIDES. In Part 2, detailed information about how to work with the DSIDES's Wrapper effectively, including the Front-end that has been designed to prepare the input for DSIDES and the Back end that is designed to support the exploration of the solution space to find solutions that are relatively insensitive to uncertainty, is provided. There is information on interpreting the results and verifying and validating the DSIDES's Wrapper in Part 2. To this end, three problems are presented that demonstrate how to start from scratch, formulate the problem in cDSP, and finally implement the problem in the wrapper. This section is a useful document for users to practice and learn how to use the software effectively.

Glossary

DSP: The Decision Support Problem (DSP) is a framework used to help make better decisions when different goals conflict with each other. Solutions that balance these goals fairly are sought by the DSP. A structured way to analyze and choose the best option from different choices is provided by the DSP. Complex situations are guided by decision support systems, which utilize data, models, and criteria, facilitating informed choices that align with our needs and priorities.

cDSP: The compromise Decision Support Problem (cDSP) is a hybrid model combination of mathematical programming and goal programming techniques that are provided to offer structure and support for decision-making that involves multiple conflicting goals or objectives. The objective of cDSP is to enhance the design of a system by modifying its variables, taking into account system constraints, goals, and bounds. The goal of cDSP is to minimize the deviation between the system's performance and the desired target values specified in the problem requirements list.

DSIDES: (Decision Support in the Design of Engineering Systems). It is written in FORTRAN to implement the cDSP construct to identify robust satisficing solutions to design problems when the models are abstractions of reality.

Wrapper: Wrapper is a protective layer. The underlying functionality of a component or system has been simplified with the use of a wrapper. The core functionality is wrapped around, and a simplified interface for interacting with the underlying front-end or back-end systems is provided. In our work, we have designed a user-friendly Wrapper for DSIDES.

Interface: A set of rules or specifications is defined by an interface, which determines how different front-end or back-end components can communicate and interact with each other. Compatibility is ensured, and seamless communication between different parts of the system is facilitated by the interface, which acts as a bridge.

Front-End: The user-facing part of a software application or website, known as the Front-end, is where content is presented, user input is facilitated, and an intuitive and engaging user experience is provided. It includes visual elements, user interface, and interactive features directly interacted with by users in Front-End. In our work, Front-end has been designed to prepare the input for DSIDES.

Back-End: In general, the server-side of a software application or website, known as the back end, is where data processing, business logic, and communication with the Front-end are handled. It includes components, databases, and logic that are responsible for managing and storing data, executing complex calculations or operations, and providing the necessary information and functionality to the Front-end for a seamless user experience. In our work, Back-end is designed to support exploring the solution space to find solutions relatively insensitive to uncertainty.

II . Introduction To the Wrapper for DSIDES

The second part of this thesis is about the user-friendly wrapper of DSIDES, which is designed to implement the formulated problem and find solutions that are relatively insensitive to uncertainty. Building upon the material presented in Part 1 of the thesis, in Part 2, detailed instructions on effectively using the wrapper, including working with the interface, interpreting the output results, and verifying and validating the software, have been provided. To illustrate the process, three problems are presented that demonstrate how to start from scratch, formulate the problem in cDSP, and ultimately implement it in the wrapper ace. By combining the theoretical foundations established in the first part with practical examples and step-by-step guidance, this section is designed to serve as a valuable resource for users to become proficient in utilizing DSIDES for their research purposes.

II.1 Understanding Front-End, Back-End, and Wrapper: Enhancing User Experience and Functionality

Front-end and back-end are terms used by programmers and computer professionals to illustrate the layers that make up hardware, a computer program, or a website, which are defined based on how accessible they are to a user. The Front-end and back-end can also be used to describe situations where the customer has access to one view and employees have access to another. Front-end components are customer-facing, while rights to the back end are exclusively for authenticated users. In other words, Professionals usually handle the front-end aspect of a project, while engineers and developers handle the back end. Integration Diagram: Front-end and Back-end Connections have been represented in Figure II.1. In this figure, specifically, the DSIDES’s wrapper has been considered. (In Figure II.1, "HTC" stands for "Human-Technology Collaboration. The collaboration and interaction between humans (users or designers) and technology (software or systems) to achieve a specific goal or outcome are signified by HTC. In the context of wrapper design and programming, the combined efforts of designers and programmers working together to create a wrapper that meets the needs and expectations of users are referred to as HTC, which means the importance of considering both the human aspect (user experience, usability, visual design, etc.) and the technological aspect (programming, functionality, data processing, etc.)).

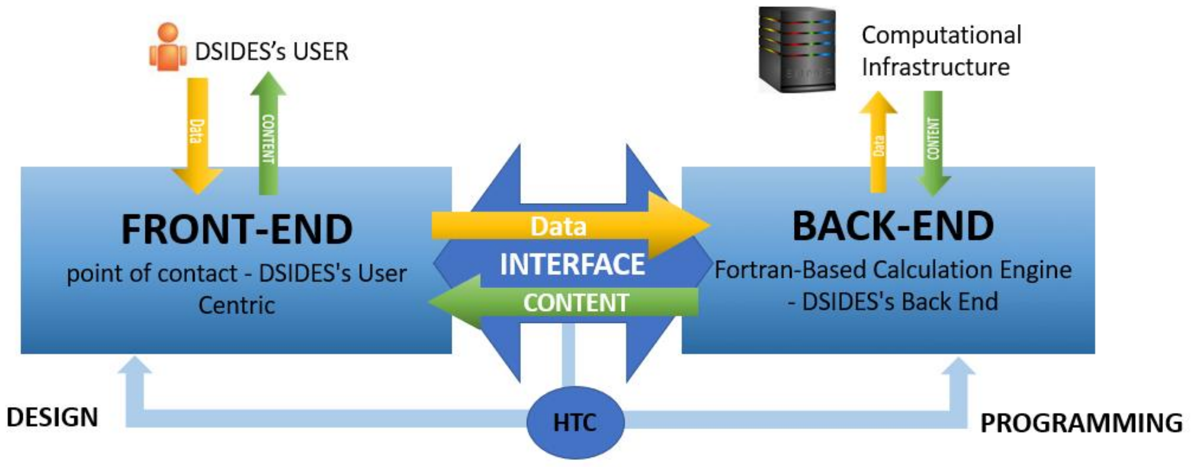

DSIDES’s USER
Data
CONTENT
Computational Infrastructure
Data
CONTENT
FRONT-END
point of contact - DSIDES's User Centric
Data
INTERFACE
CONTENT
BACK-END
Fortran-Based Calculation Engine - DSIDES's Back End
DESIGN
HTC
PROGRAMMING

Figure II. 1: Integration Diagram: Front-end and Back-end Connection in DSIDES

In this section, we explore what front-end and back-end are and their importance and explain the wrapper concept and role in integrating the front and back-end.

The Front-end refers to the user-facing parts of an application, while the back end refers to the parts of an application that operate without user accessibility. Typically, front-end development involves designing and implementing the elements that users interact with, such as buttons, menus, forms, and other graphical elements.

Front-End: Front-end development is a critical part of software development as it directly impacts an application's usability and overall user experience. Designing a good user interface is one of the essential aspects of front-end development. The designer must consider the application's purpose, target audience, and other factors that affect the user experience when creating the user interface. A good user interface should be intuitive and easy to navigate, allowing users to perform tasks quickly and efficiently.

Front-end development has evolved significantly in recent years, with new technologies, tools, and techniques emerging every day. However, the basic principles of front-end development remain the same. The primary goal of front-end development is to create an attractive, intuitive, and user-friendly interface that engages users and meets their needs.

Front-end development also involves optimizing the application's performance, such as reducing the size of files and images, compressing code, and minimizing the number of server requests. By doing so, the application can load faster, providing a better user experience.

Several popular front-end frameworks, such as React, Angular, and Vue.js, provide developers with pre-built components and structures, making it easier to create complex user interfaces quickly. Additionally, they often have a strong community of developers who contribute to their development, providing resources and support to other developers.

Front-end development also involves integrating the user interface with back-end systems, which requires an understanding of how data is stored, retrieved, and processed. In many cases, front-end developers work closely with back-end developers to ensure the user interface and back-end systems are compatible and work seamlessly together.

Testing is another critical aspect of front-end development. Front-end developers must test their code thoroughly to ensure that the user interface functions correctly and provides a positive user experience. This involves testing the application on different devices and browsers and testing for accessibility and performance.

To illustrate the importance of front-end development, let us consider an example. Imagine that you are building a website for an e-commerce store. The website's Front-end must be intuitive and easy to use, with clear navigation and a simple checkout process. If the Front-end is poorly designed or difficult to use, users may abandon their shopping carts and go to a competitor's website instead. In contrast, a well-designed and user-friendly Front-end can increase user engagement, improve customer satisfaction, and ultimately lead to more sales and revenue for the store.

In conclusion, front-end development is a constantly evolving field critical to the success of software applications. It is an essential part of web development that involves designing and implementing a website's visual and interactive elements. It requires combining technical skills, design knowledge, and communication skills. In addition, a good front-end developer must understand user interface design, accessibility, and performance optimization. Software applications can provide a seamless and enjoyable user experience by considering these key aspects of front-end development.

Back-End: The back end refers to parts of a computer application or a program's code that let it operate Without user accessibility. In other words, it refers to any system that supports backoffice applications and manages orders, inventory, and supply processing. The back end gets input from front-end applications, supports user services, and interfaces with required resources. It may interact directly with the Front-end or be called from an intermediate program that intercedes front-end and back-end activities.

Most data and operating syntax are saved and accessed in the back end of a computer system. Usually, the code is comprised of one or more programming languages. It is also called the data access layer of software or hardware and comprises any functionality that requires to be accessed and navigated to by digital means.

The processing and management of data, execution of core functionality, and performance of calculations are handled by the back end of a software system. The back end operates on the server-side and collaborates with the Front-end to ensure users are provided with a comprehensive and interactive experience.

Tasks and operations involved in back-end processing are executed behind the scenes to process data and perform complex calculations. This includes the handling of user inputs, interaction with databases, execution of computations, and generation of results. Achieving efficient and accurate execution to attain the desired functionality of the software system is the main focus.

Data management within the back end encompasses the software system's organization, storage, retrieval, and manipulation of data. Activities such as database management, data validation, integration, and security are included. Data integrity is ensured by the back end, suitable storage solutions are implemented, and efficient mechanisms for data access and retrieval are provided.

A seamless and efficient user experience is delivered by collaborating the back end with the

Front-end. Behind the scenes, the back-end processes data and executes complex operations necessary for the functioning of the software application. Data is retrieved and processed, computations are performed, and results are presented to the users by the Front-end through communication with the back end. Back-end processing takes place on the server-side and remains concealed from direct user interaction. It is responsible for handling the heavy lifting of data processing, calculations, and core functionality execution. Efficient data processing and generation of accurate results are ensured by the back end for the smooth operation of the software system.

Maintaining the integrity of the system's data and organizing it is crucial, and this is accomplished through data management in the back end. It involves tasks such as managing databases, validating data inputs, integrating data from various sources, and implementing security measures to safeguard sensitive information. Effective data manipulation and access by the Front-end and other system components are enabled by providing efficient mechanisms for data storage and retrieval by the back end.

Overall, the back end of a web application is responsible for managing data and providing the necessary functionality to the Front-end. It is a complex system that requires a range of skills and expertise to build and maintain.

Wrapper: In programming, a wrapper is a piece of code or software designed to simplify or enhance the use of an existing functionality or component. It wraps around the original code or component and provides a more user-friendly or convenient interface to work with.

Think of it like a gift wrapper. The gift inside may have its own shape, size, and packaging, but the wrapper makes it easier to handle and present. Similarly, a programming wrapper makes it easier for developers to interact with complex code by providing a simpler and more intuitive interface. It hides the complexity of the underlying code and exposes a more straightforward way to use it. A wrapper can also add extra features or functionality to the original code. It may include additional methods or functions that extend the capabilities of the wrapped code, making it more versatile and adaptable for different purposes. In other words, a wrapper in programming is a convenient and user-friendly layer that simplifies the use of existing code or components, making them easier to work with and enhancing their functionality if needed.

The wrapper concept in software development is a piece of code or software component that is acted upon as an intermediary layer between the front and back-end of an application. Its primary role is played by facilitating the integration and communication between these two components.

The bridge is served by the wrapper between the user interface and the underlying functionality or data processing logic (back-end). It is acted upon as an encapsulator of complex operations and is provided with a simplified and standardized interface for the Front-end to be interacted with by the back end.

Some Key Roles of a Wrapper in Integrating the Front-End and Back-End:

Abstraction and Simplification: The underlying complexity of the back end is abstracted by the wrapper, as it is provided with a simplified and user-friendly interface for interaction with the Front-end. A set of functions or methods are presented by it, which can be utilized by the Front-end without needing to understand the intricacies of the back-end implementation.

Data Transformation and Validation: The tasks of data transformation and validation are handled by the wrapper, ensuring that the data passed between the Front-end and back-end is in the correct format and meets the required specifications. The data is converted and prepared from the format used by the Front-end to the format expected by the back end and vice versa.

Communication and Interfacing: Communication channels are established between the Front-end and back-end by the wrapper, thus facilitating the exchange of data and commands. The transmission of requests from the Front-end to the back end is handled by it, and the corresponding responses are received, ensuring smooth interaction and data flow between the two components.

Error Handling and Exception Management: Mechanisms for handling errors and exceptions that may occur during the interaction between the Front-end and back-end are included within the wrapper. Errors or exceptions raised by the back end are captured and processed by it, providing appropriate feedback or error messages to the Front-end.

Security and Access Control: Security measures and access control policies can be enforced by the wrapper, ensuring that only authorized requests and data are passed between the Front-end and back-end. User credentials can be authenticated and validated, access to specific functionalities or data can be authorized, and data privacy and protection can be enforced.

Overall, the integration of the Front-end and back-end is facilitated by the wrapper, providing a simplified and standardized interface, handling data transformation and validation, managing communication, handling errors, and ensuring security. Seamless collaboration between the user interface and the underlying functionality is enabled by it, contributing to the overall efficiency, usability, and robustness of the software application.

After learning about the front-end, back-end, and Wrapper, we can now delve into the detailed workings of the wrapper and provide users with a detailed user manual. In these next sections, we will explore the various features and functionalities of the wrapper step by step, guiding users on how to navigate and utilize its capabilities effectively.

Before proceeding to the next section, we will present the steps of the DSIDES wrapper in the form of a flowchart in Figure II.2. This will allow the user to easily and quickly review the process, ensuring a better understanding of the DSIDES wrapper workflow. As represented in Figure II.2, the user needs to follow the following steps:

Step 1: Formulate the problem in the cDSP construct. (Elephant Stand Problem has been represented as an example in part II.2)

Step 2: Open the main folder.

1) It could have any name based on user preference. (Suggestion is to select the project's name.)
2) This folder includes 2 folders (OutPut_Files and TernaryPlot) and one excel file (DSIDES_Wrapper.xlsm).

Step 3: Is the problem in Archimedean form?

1) If yes: Import the required information into the LinearArchimedean sheet. (Section II.2.1)
2) If No: Import the required information into the LinearPreemptive sheet. (Section II.2.2)

Step 4: After importing the data in the proper sheet, Save the data and press the Run button. After this step, DSIDES.dat will be built in the Output_Files folder."

Step 5: Import the required information into the NonLinear sheet. (Section II.2.3).

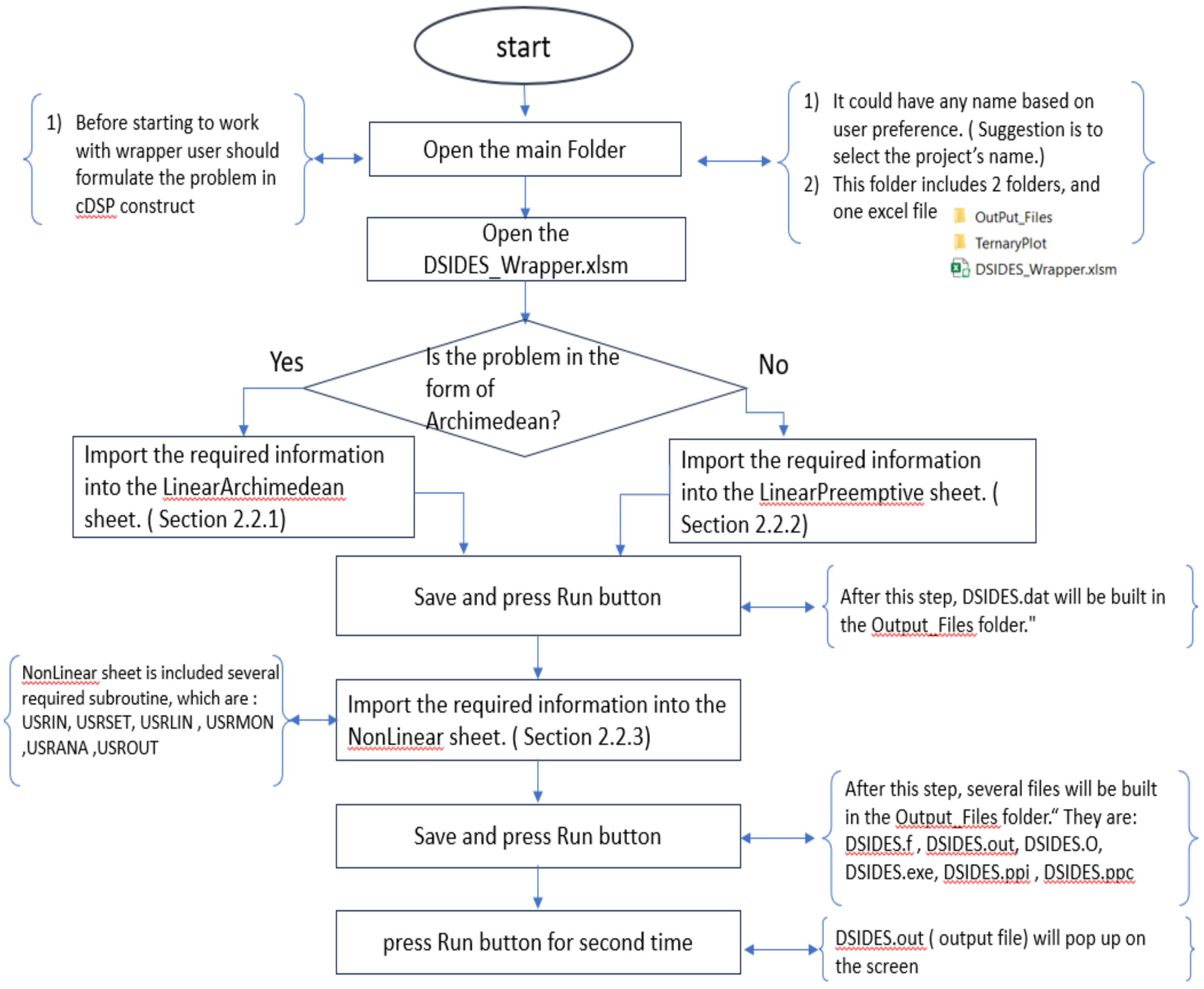
Step 6: Save the imported information and press the Run button. After this step, several files will be built in the Output_Files folder." They are DSIDES.f, DSIDES.out, DSIDES.O, DSIDES.exe, DSIDES.ppi, DSIDES.ppc

Note: if you press the Run button for the second time, the output file (DSIDES.out) will pop up on the screen

Step 7: Ternary Plot for Goal 1: Import the required input into the TernaryPlot1 sheet to generate the ternary plot for the first goal (Section II.2.4)

Step 8: Ternary Plot for Goal 2: Import the required input into the TernaryPlot2 sheet to generate the ternary plot for the second goal (Section II.2.4)

Step 9: Ternary Plot for Goal 3: Import the required input into the TernaryPlot3 sheet to generate the ternary plot for the third goal (Section II.2.4)


start
1) Before starting to work with wrapper user should formulate the problem in cDSP construct
Open the main Folder
1) It could have any name based on user preference. (Suggestion is to select the project's name.)
2) This folder includes 2 folders, and one excel file
OutPut_Files
TernaryPlot
DSIDES_Wrapper.xlsm
Open the DSIDES_Wrapper.xlsm
Is the problem in the form of Archimedean?
Yes
No
Import the required information into the LinearArchimedean sheet. (Section 2.2.1)
Import the required information into the LinearPreemptive sheet. (Section 2.2.2)
Save and press Run button
After this step, DSIDES.dat will be built in the Output_Files folder."
NonLinear sheet is included several required subroutine, which are : USRIN, USRSET, USRLIN , USRMON ,USRANA ,USROUT
Import the required information into the NonLinear sheet. (Section 2.2.3)
Save and press Run button
After this step, several files will be built in the Output_Files folder." They are: DSIDES.f , DSIDES.out, DSIDES.O, DSIDES.exe, DSIDES.ppi , DSIDES.ppc
press Run button for second time
DSIDES.out (output file) will pop up on the screen

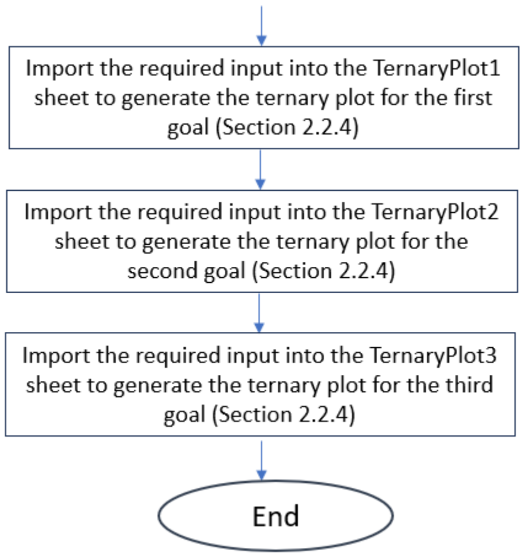

Import the required input into the TernaryPlot1 sheet to generate the ternary plot for the first goal (Section 2.2.4)
Import the required input into the TernaryPlot2 sheet to generate the ternary plot for the second goal (Section 2.2.4)
Import the required input into the TernaryPlot3 sheet to generate the ternary plot for the third goal (Section 2.2.4)
End

Figure II. 2: Flowchart of DSIDES Wrapper

II.2 Preparing Input for DSIDES in Different Sheets in the Wrapper

As outlined in Part 1 and shown in Figure II.2, it is necessary to formulate the problem as a Compromise Decision Support Problem (cDSP) construct before utilizing the DSIDES wrapper. To demonstrate the functionality of the DSIDES wrapper, the Elephant Stand problem, as an illustrative example, has been represented. Consequently, various steps to address this example have been presented, including defining the problem statement, formulating the cDSP (both in word and mathematical form), and finally implementing it within the DSIDES wrapper.

Elephant Stand Problem in Archimedean Form

Problem Statement:

In the late 1800s, Ringling Bros and Barnum and Baily Circus were looking to establish dimensions of a new pedestal for their circus elephant Jumbo. They would play a trick that involved a support pedestal where Jumbo would perform a one-legged hand stand. The cost of manufacturing must be minimized, which depends on its thickness, width, and the amount of material it consumes.

And it must be as tall as possible for a wow factor. And finally, the pedestal must be wide enough to ensure Jumbo has enough room to safely stand on one foot. This means *the goals of the design are to minimize the manufacturing cost, maximize the height, and maximize the outer radius.* A material of 2014 Aluminum with a modulus of 10600 ksi and a density of 0.1 lb/in^3 has been selected.

Jumbo's foot is approximately 25" in diameter, so the pedestal must also be greater than 25". Jumbo weighs 13,560lb and stands 13.5ft tall. Use a factor of safety of 1.5.

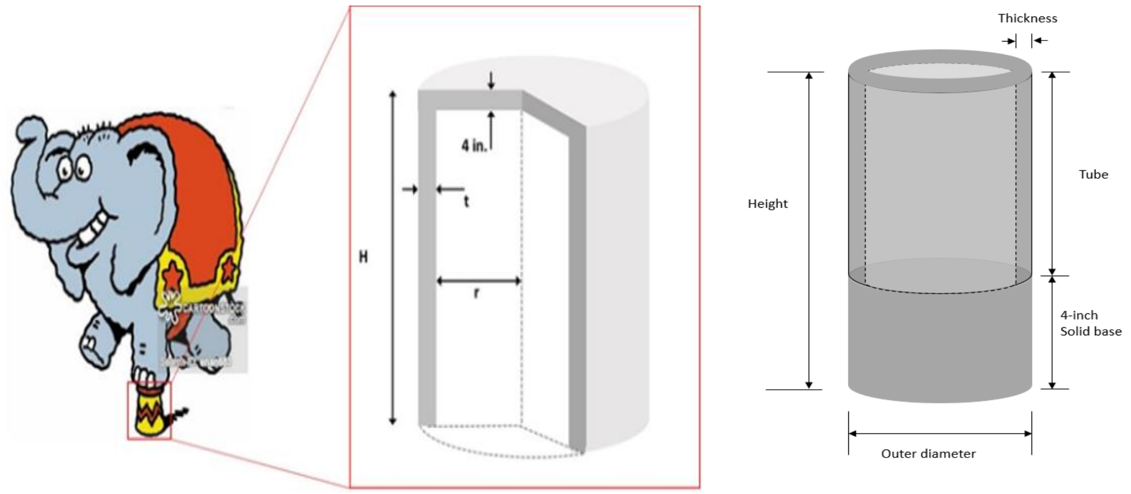


Figure II. 3: Overview of Elephant stand problem.

Given a certain type of material, design a cylinder (the “elephant stand”). The cylinder has two parts joined together. The upper half is a tube, and the designer’s interest is to determine its thickness, radius, and height that best satisfies the goals identified. The lower half is a 4-inch-height solid base. The goals identified by the designer include Minimizing the manufacturing cost, maximizing the height, and maximizing the outer diameter. Requirements include the upper and lower limit of the parameters that the cylinder can physically reach. The overview of the elephant stand problem is represented in Figure II.3.

cDSP Formulation:

Word Formulation:

Given

System parameters

Material – 2014 Aluminum

The elastic modulus for the material

Safety factor

Yield stress for the material

Density of the material

Load (elephant's weight)

Moment of inertia for the cylindrical section

Maximum normal stress

Maximum buckling stress for a fixed free column

Cost target

Height target

OR target

Find

System variables

Radius, Thickness, Height

Satisfy

System constraints

Reaching the minimum outer diameter.

Not exceeding a certain height-to-width ratio.

Reaching the stress requirement.

Not exceeding the maximum weight.

Not exceeding the maximum load in stand.

System goals

Goal 1: reaching minimum cost target.

Goal 2: reaching maximum height target.

Goal 3: reaching maximum outer radius target.

Bounds

The upper and lower limit of Radius, Thickness and Height

Minimize

The deviation function.

cDSP Math Formulation:

Given

E = 10600000.

OR=R+T

SF   = 1.5

SIGY = 11000.

P   = 12000.

PI   = 2*ACOS(0.0)

RHO = 0.1

$$I = \frac{\pi}{4} * [(R + T)^4 - R^4]$$

$$W1 = \pi * [(R + T)^2 - R^2] * (H - 4)$$

$W2 = \pi * [(R + T)^2] * 4$

W=(W1+W2)∗RHO

STR=P/(π∗[(R+T)^2−R^2])

PCR=(π^2∗E∗I)/ 4∗(H^2)

COST=e^(2.5/T)∗e^(3/R)∗W

Cost target = 5000

Height target = 180

OR target = 15

Find

System variables

- R Radius

- T Thickness

- H Height

Deviation variables

- d_i^+ over achievement of Goal i, where i=1,2,3

- d_i^- under-achievement of Goal i, where i=1,2,3

Satisfy

System constraints

- minOD: minimum outer diameter

$2 * R + 2 * T \geq 6$ (CO1)

- heiwid: height to width ratio

$R + T - 0.03 * H \geq 0$ (CO2)

- stress : minimum stress in stand

$$\frac{\mathrm{SIGY}}{\mathrm{SF}} - \mathrm{STR} \geq 0 \qquad \text{(CO3)}$$

- weight: maximum weight

$$1000 - \mathrm{W} \geq 0 \qquad \text{(CO4)}$$

- buckle: maximum load in stand

$$\frac{\mathrm{PCR}}{\mathrm{F}} - \mathrm{P} \geq 0 \qquad \text{(CO5)}$$

System goals

- Goal 1: minimum cost

$$\frac{\text{cost target}}{\text{cost}} + d_1^- - d_1^+ = 1 \qquad \text{(G1)}$$

- Goal 2: maximum height

$$\frac{\text{height}}{\text{height target}} + d_2^- - d_2^+ = 1 \qquad \text{(G2)}$$

- Goal 3: maximum outer radius

$$\frac{\text{OR}}{\text{OR target}} + d_3^- - d_3^+ = 0 \qquad \text{(G3)}$$

Bounds

$3 \leq R \leq 45$

$0.5 \leq T \leq 2.5$

$100 \leq H \leq 120$

Minimize

The deviation function.

$$Z = \sum_{i=1}^{3} w_i \cdot (di^- + di^+), \sum_{i=1}^{3} w_i = 1$$

Implementation in DSIDES Wrapper:

In this section, a detailed description of the wrapper is provided. The work is initiated with a main folder that contains two subfolders: TernaryPlot and Output_Files. Additionally, an Excel file named DSIDES_Wrapper will be included in this folder.

The TernaryPlot folder encompasses all the necessary files for generating the ternary plot, eliminating the need for user intervention. The second folder, named OutPut_Files, contains all the required files for establishing the connection with the DSIDES software and executing the program to obtain the final solution. Following the execution of the Linear sheet, the DSIDES.dat file is generated within this folder. Subsequently, upon executing the Nonlinear sheet, six additional files are generated in this folder, all named DSIDES. These files are of different types, including Fortran Source (.f), Out File, PPI File, PPC File, O file, and Application. The OUT file contains all output information described in Section II.3, "Interpret the Output File."

Regarding the name of the main folder, users can choose the naming of this folder based on their preference. For instance, they may use the project name for clarity in future reference. As illustrated in Figure II.4, the folder is referred to as DSIDES_ElephantStand.

Note: It is important to note that the folder name should consist of a single word without any spaces between alphabets.

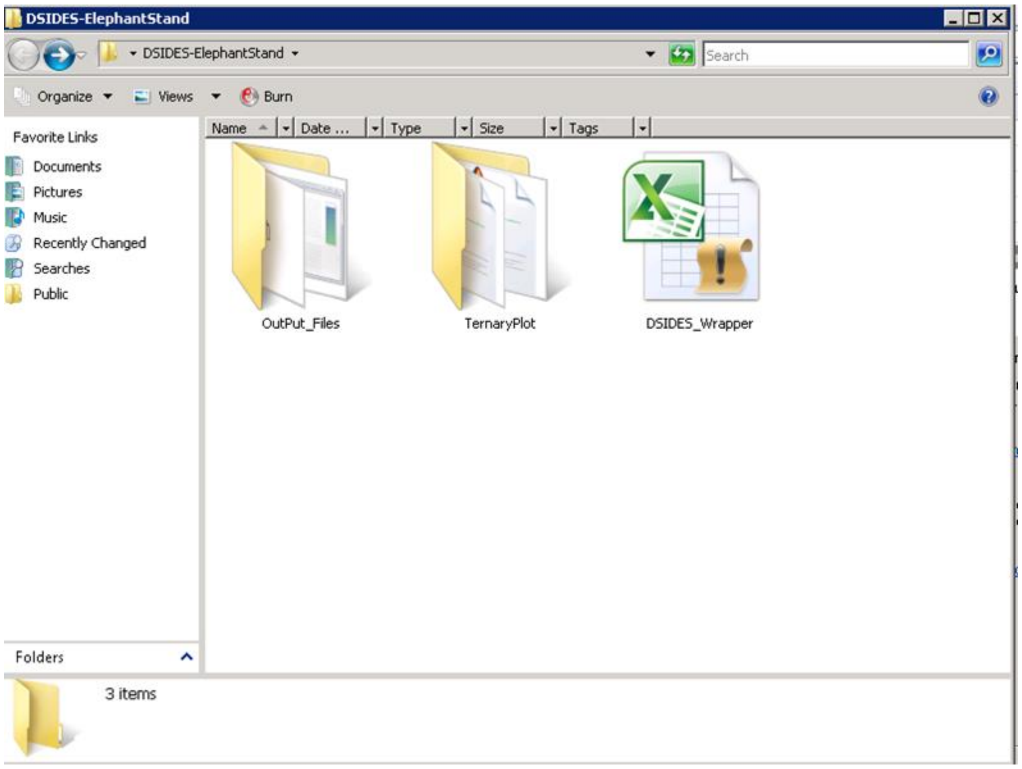


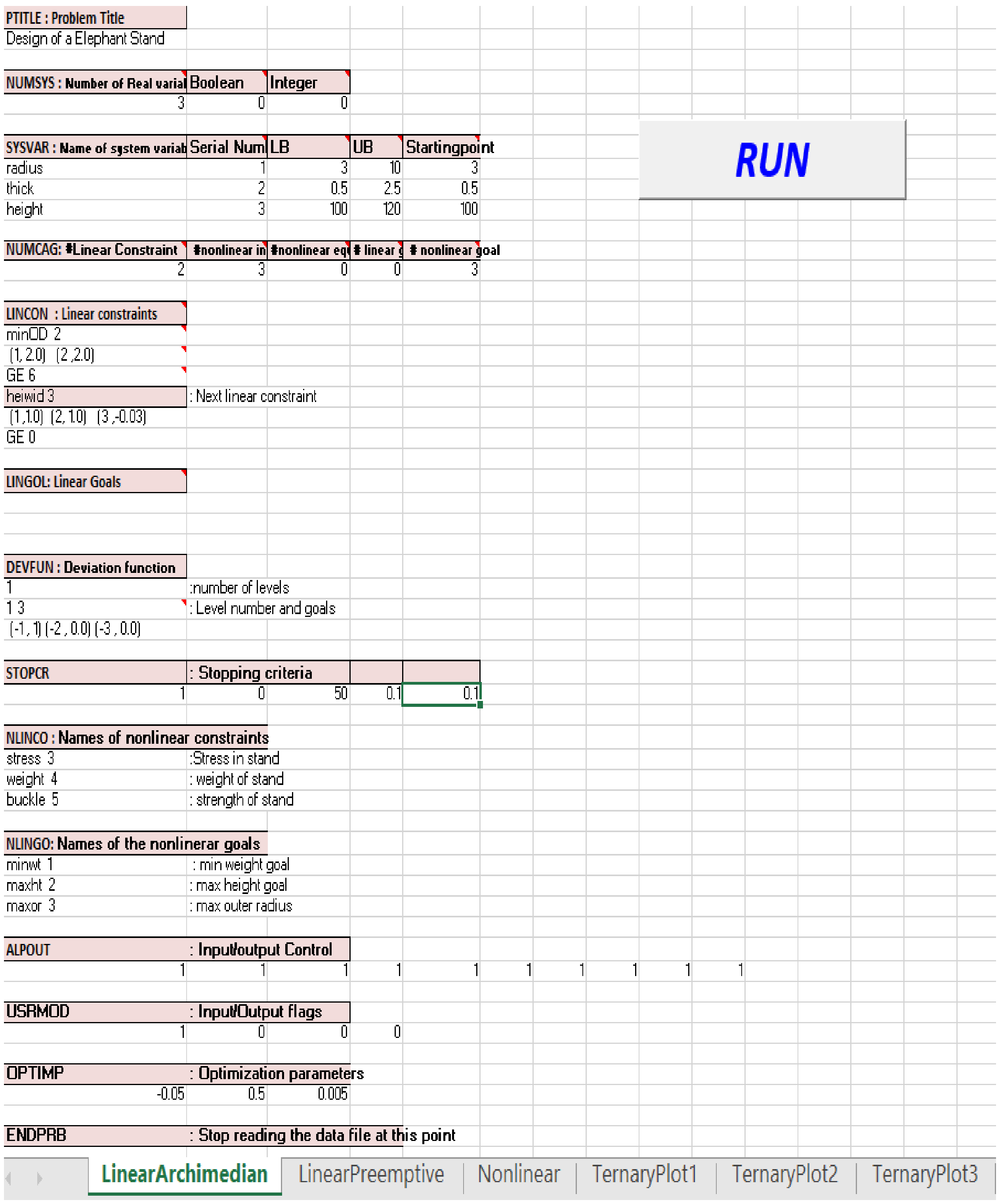
Figure II. 4: Files Included in the Main Folder

In the folder, there is an Excel sheet named "DSIDES_Wrapper." This file is the primary working file for our Wrapper, and it contains six different sheets, namely LinearArchimedian, LinearPreemptive, NonLinear, TernaryPlot1, TernaryPlot2, and TernaryPlot3.

In the subsequent sections, a detailed description of each sheet, outlining the necessary steps for importing the required information into each sheet to enable efficient use of the software based on the Elephant stand problem, is provided.

II.2.1 Linear Archimedean Sheet

As mentioned in Figure II.2 (Flowchart of DSIDES Wrapper), the third step after defining the problem in cDSP form and opening the DSIDES_Wrapper.xlsm file, the initial sheet is titled "Linear Archimedean Sheet." We will go through this sheet if our problem is Archimedean form. The general layout of this sheet is displayed in Figure II.5. To solve an Archimedean problem (Such as Elephant Stand Problem), the user must incorporate the linear information of the compromise decision support problem from this source. Therefore, this sheet will be discussed in detail in this section, thoroughly describing its contents and functionalities (the Elephant Stand Problem has been implemented as an example in this section).



PTITLE : Problem Title									
Design of a Elephant Stand									
NUMSYS : Number of Real varial	Boolean	Integer							
3	0	0							
SYSVAR : Name of system variab	Serial Num	LB	UB	Startingpoint					
radius	1	3	10	3					
thick	2	0.5	2.5	0.5					
height	3	100	120	100					
NUMCAG: #Linear Constraint	#nonlinear in	#nonlinear eq	# linear g	# nonlinear goal					
2	3	0	0	3					
LINCON : Linear constraints									
minOD 2									
(1, 2.0) (2 ,2.0)									
GE 6									
heiwid 3	: Next linear constraint								
(1 ,1.0) (2, 1.0) (3 ,-0.03)									
GE 0									
LINGOL: Linear Goals									
DEVFUN : Deviation function									
1	:number of levels								
1 3	: Level number and goals								
(-1 , 1) (-2 , 0.0) (-3 , 0.0)									
STOPCR	: Stopping criteria								
1	0	50	0.1	0.1					
NLINCO : Names of nonlinear constraints									
stress 3	:Stress in stand								
weight 4	: weight of stand								
buckle 5	: strength of stand								
NLINGO: Names of the nonlinerar goals									
minwt 1	: min weight goal								
maxht 2	: max height goal								
maxor 3	: max outer radius								
ALPOUT	: Input/output Control								
1	1	1	1	1	1	1	1	1	1
USRMOD	: Input/Output flags								
1	0	0	0						
OPTIMP	: Optimization parameters								
-0.05	0.5	0.005							
ENDPRB	: Stop reading the data file at this point								

Figure II. 5: Overview of LinearArchimedian sheet

The data file for compromise Decision Support problems is divided into several blocks. Each block has a six-character name, e.g., PTITLE, NUMSYS, etc. The user is allowed to organize the problem-related information into logically arranged chunks of information by the blocks. Certain blocks are mandatory, i.e., they have to be specified for any problem. These blocks constitute the bare minimum amount of information required for a problem. Other optional blocks provide the user with further control of the solution procedure. Advanced users are expected to use more optional blocks than novice users. The various blocks are listed in this section. Mandatory blocks and Optional blocks have been represented in Table II.1 and Table II.2, respectively.

Table II.1: Mandatory Blocks in LinearSheet

PTITLE	1	Problem title
NUMSYS	2	Number of System Variables
SYSVAR	3	Description of System Variables - name, type, bounds, and guess value
NUMCAG	4	Number of Constraints and Goals
LINCON	5	Linear Constraints - names and data (if specified in NUMCAG)
LINGOL	6	Linear Goals- names and data (if specified in NUMCAG)
DEVFUN	7	Deviation Function - number of levels and weights of deviation variables
STOPCR	8	Stopping Criteria (run and principal print flags, NITER, EPSZ, EPSX)

Table II.2: Optional Blocks in LinearSheet

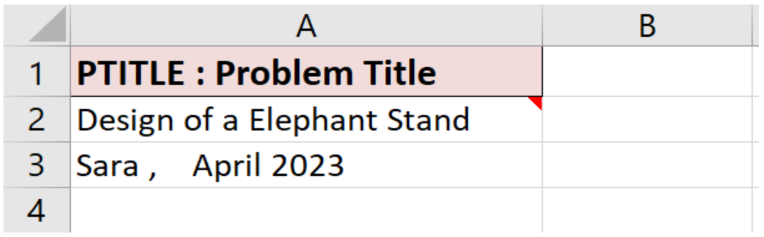
NLINCO	9	Names of Nonlinear Constraints (default names: NLCO##)
NL I NGO	10	Names of Nonlinear Goals (default names: NLGO##)
ALP OUT	11	Flags for Output Level, Post Processor, and Time Statistics
USRMOD	12	Flags for User Modules (USRINP, USROUT, USRMON, USRLIN)
OPTIMP	13	Optimization Parameters (VIOLIM, REMO, STEP)
INITFS	14	Automatic Generation of Initial Feasible Solution
USRDAT	15	User Data Block for Access From USRINP
ADPCTL	16	Nonlinear Inequality Constraint Adaption Flag (LADAP)
US ERAN	17	Information for USRANA (maximum cycles - NANCY, NSYCY)
FIXVAR	18	Fixing of Variables
SUPCON	19	Suppression of Nonlinear Constraints
PVALFX	20	Particular Values for Stationarity of System Variables
PVALFZ	21	Particular Values for Stationarity of Deviation Function Levels
PVSTEP	22	Particular Values for STEP

PVCVIL	23	Particular Values for VIOLIM
PVREMO	24	Particular Values for REMO
ADREMO	25	Adaptive Reduced Move Parameters
XPLORE	26	Explore the design space for best initial points
ENDPRB	27	End of Problem Definition

DESCRIPTION OF DATA BLOCKS:

Subsequent to familiarizing oneself with the significance of the various data blocks, including mandatory and optional fields present in different cells of the Excel sheet, In the subsequent paragraphs, a comprehensive overview of each data block will provide.

PTITLE (Mandatory):
Purpose: Define the problem title and other user information (name, date, etc.)
Format:
Textline1 (80 characters maximum)
textline2 (80 characters maximum)
Variables:
Textline1: string (80 characters maximum)
textline2: string (80 characters maximum)
Example:



	A	B
1	**PTITLE : Problem Title**	
2	Design of a Elephant Stand	
3	Sara , April 2023	
4		

As depicted in the figure above, a small red indicator is visible in a specific cell. Placing the mouse cursor over the red indicator will reveal a pop-up text box that contains the description of the cell.

NUMSYS (Mandatory):

Purpose: Define the number of system variables - real and Boolean.

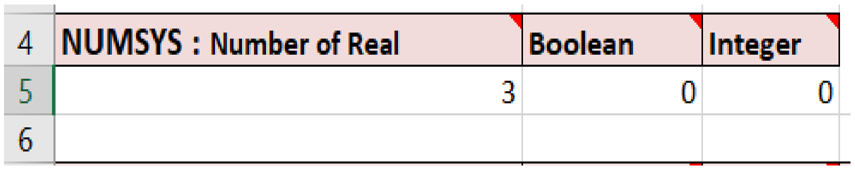
Format:

i k j

Variables:

i: integer	Number of Real variables
k: integer	Number of Boolean (selection) variables
j: integer	Number of Integer variables

Example:



4	NUMSYS : Number of Real	Boolean	Integer
5	3	0	0
6			

Notes:

If you do not have any variables of one type, you must indicate this by specifying a value of zero. In other words, three integers (i, k, and j) must be specified on the second line of the block. (Line 5)

SYSVAR (Mandatory):

Purpose: Define system variable information.

Format:

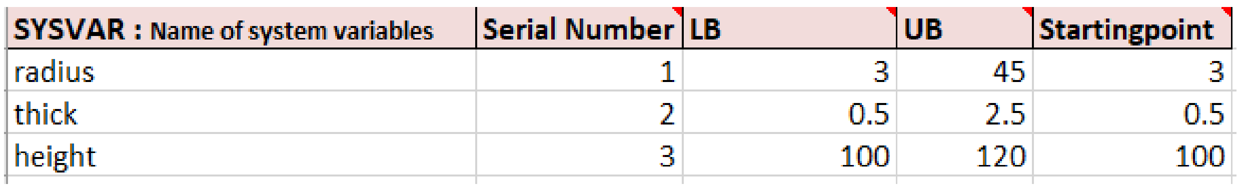
Name Serial number LB UB guess

Variables:

Name: string	Name of variable (6 characters long)
Serial number: integer	Serial number of variable
LB: (real/integer)	Lower bound for variable
UB: (real/integer)	Upper bound for variable

guess: (real/integer) Initial guess value for variable (Starting point)

Example:



SYSVAR : Name of system variables	Serial Number	LB	UB	Startingpoint
radius	1	3	45	3
thick	2	0.5	2.5	0.5
height	3	100	120	100

Notes:

- Real variables must precede the integer variables, and the Boolean variables should follow the integer variables.
- If the variable name is not given, a default name is assigned to the variable. This is of the form X## where## is the serial number of the variable, e.g., Xl, X45, etc.
- Variable types are assigned based on the serial number of the variable and are shown in the output file.
- Lower and upper bounds for Boolean variables are O and 1, respectively. The guess value can be either O or 1.
- If the initial guess value is out of the specified bounds, a default value of[min max] is assumed, and a warning is printed in the output file.
- It is possible to add as many variables as necessary. If the available space in a spreadsheet is insufficient, additional rows can be inserted to accommodate new variables. This can be done by right-clicking on the empty cell and selecting the "Insert" option to add a new row. In this way, the data set can be expanded to include any number of variables required for the problem at hand.

4. NUMCAG (Mandatory):

Purpose: Define the number of constraints and goals.

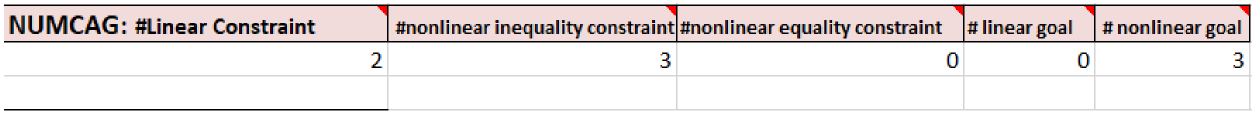
Format:

i j k m n

Variables:

i: integer	Number of linear constraints
j: integer	Number of nonlinear inequality constraints
k: integer	Number of nonlinear equality constraints
m: integer	Number of linear goals
n: integer	Number of nonlinear goals

Example:



NUMCAG: #Linear Constraint	#nonlinear inequality constraint	#nonlinear equality constraint	# linear goal	# nonlinear goal
2	3	0	0	3

Notes:

- The absence of a particular type of constraint or goal must be explicitly specified by using a zero for the corresponding argument. For instance, zero indicates that there are no nonlinear equality constraints in the above example.
- If linear constraints are specified (i.e., i* 0), then the LINCON block has to be specified.
- If linear goals are specified (i.e., m* 0), then the LINGOL block has to be specified.
- It is assumed that all the nonlinear constraints and nonlinear goals are defined in the user-supplied FORTRAN module USRSET.
- This block must be specified before any of the other blocks relating to constraints and goals.

• The default names for the deviation variable names are assigned based on the total number of goals (=m+n) specified in this block. These names are printed in the output file.

5. LINCON (Mandatory if Required by NUMCAG):

Purpose: Define linear constraints.

Format: (for each linear constraint)

Name n

($ivar_1$,cof_1) (... , ...) ... ($ivar_n$,cof_n)

Lc rhs

Variables:

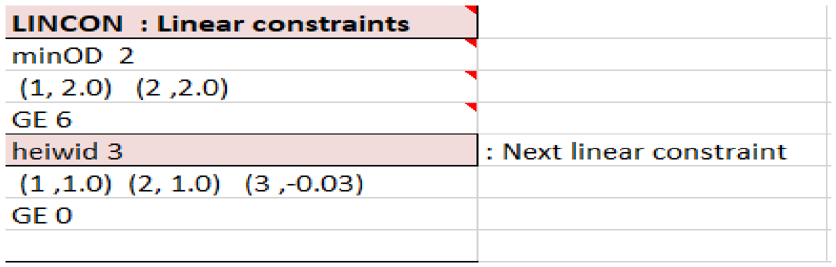
name: string	Name of linear constraint (6 characters long)
n: integer	Number of terms to follow / Number of variables in this constraint.
ivar: integer	System variable number / Serial number of Variable
cof: real	Coefficient for system variable
Lc: chr*2	Logical connector, any one of the following: le, LE, <=, ge, GE, >=, eq, EQ, ==
rhs: real	Right hand side value for constraint

Example:

The example shows the following two linear constraints.

minOD: minimum outer Diameter : 2*R +2*T >= 6

heiwid : height to width ratio : R+T -0.03 *H >= 0



LINCON : Linear constraints	
minOD 2	
(1, 2.0) (2 ,2.0)	
GE 6	
heiwid 3	: Next linear constraint
(1 ,1.0) (2, 1.0) (3 ,-0.03)	
GE 0	

Notes:

- The inequality constraints must precede all equality constraints.
- If the name is not specified, a default name of the form LICO## is assigned to the constraint, e.g., LICO1, LICO37 etc.
- The order of the terms does not matter as can be seen in the specification of the second constraint.
- All other coefficients for the linear constraint are set equal to zero.
- It is possible to add as many constraints as necessary. If the available space in a spreadsheet is insufficient, additional rows can be inserted to accommodate new constraints. This can be done by right-clicking on the empty cell and selecting the "Insert" option to add a new row. In this way, the data set can be expanded to include any number of constraints required for the problem at hand.

6. LINGOL (Mandatory if required by NUMCAG):

Purpose: Define linear goals.

Format: (for each linear goal)

Name k n

($ivar_l$,cof_l) (..,..) … ($ivar_n$,cof_n)

rhs

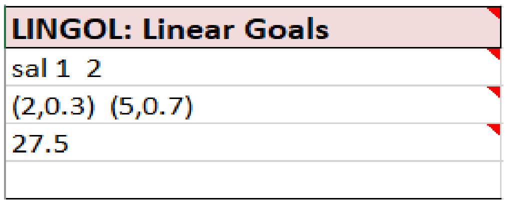
Variables:

name: string	Name of linear goal (6 characters long)
k: int	Serial number of goal.
n: int	Number of terms to follow.
ivar: int	System variable number
cof: real	Coefficient for system variable
rhs: real	Right-hand side value for goal

Example:

Since in Elephant Stand Problem, there is no linear goal; another example has been represented. The example shows the following linear goal :

$0.3\ X_2 + 0.7\ X_5 + d^- - d^+ = 27.5$



LINGOL: Linear Goals
sal 1 2
(2,0.3) (5,0.7)
27.5

Notes:

- The serial number of the goal is necessary for the specification of the achievement function.
- If the name is not specified, a default name of the form LIGO## is assigned to the goal, e.g., LIGO1, LIGO37, etc.
- The order of the terms does not matter.
- All unspecified coefficients for the linear goal are set equal to zero.
- It is possible to add as many goals as necessary. If the available space in a spreadsheet is insufficient, additional rows can be inserted to accommodate new goals. This can be done by

right-clicking on the empty cell and selecting the "Insert" option to add a new row. In this way, the data set can be expanded to include any number of goals required for the problem at hand.

7. DEVFUN (Mandatory):

Purpose: Define Deviation Function - number of levels and weights of deviation variables. It is used in Archimedean form. In Preemptive form, ACHFUN will be used.

Format: (for each linear goal)

K

L n

(m_1,w_1) (... , ...) ... (m_k,W_k)

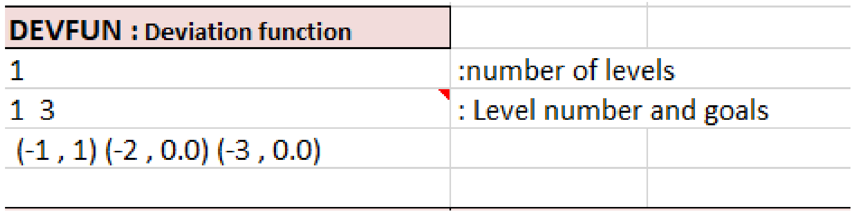
Variables:

k: int	Number of levels
L: int	Level number.
n : int	Number of goals in this level.
m: integer	signed integer indicating deviation variable. (+m for overachievement and -m for underachievement)
w: real	weight for deviation variable

Example:

One priority level.

$Z1 = 0.3d_1^- + 0.6\, d_2^- + 0.1d_3^+$



DEVFUN : Deviation function		
1	:number of levels	
1 3	: Level number and goals	
(-1 , 1) (-2 , 0.0) (-3 , 0.0)		

8. STOPCR(Mandatory):

Purpose: Define the stopping criteria.

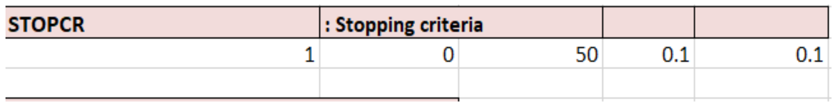
Format:

i j k a b

Variables:

i: integer	= 1, perform calculations
	= 0, dry run only, no calculations performed
j: integer	= 1, print final solution only, no intermediate results printed.
	= 0, print intermediate results
k: integer	Maximum number of synthesis cycles (NITER)
a: real	Desired stationarity of deviation function (EPSZ)
b: real	Desired stationarity of system variables (EPSX) { Recommended value: 0.05}

Example:



STOPCR	: Stopping criteria			
1	0	50	0.1	0.1

Notes:

- The iterations are stopped if the change in any level of the deviation function is less than EPSZ.

Values for individual levels can be changed using the block PVEPSZ.

- The iterations are stopped if the following condition is met by the system variables,

$$| X_{i,old} - X_{i,new} | <= FRAC_i$$

Where:

$$FRAC_i = EPSX * [X_{i,max} - X_{i,min}]$$

- Particular values for FRAC can be set in the block PVALFX.

9. NLINCO (Optional):

Purpose: Define the names of nonlinear constraints (inequality constraints first).

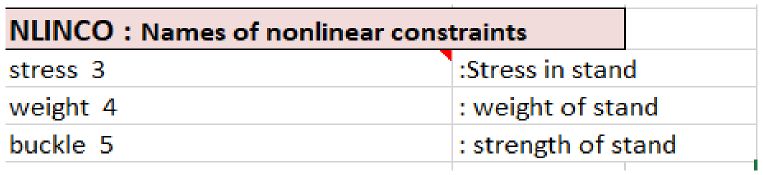
Format: (for each constraint)

name k

Variables:

name: string	Name of nonlinear constraint (6 characters long)
k : integer	Number of the nonlinear constraint

Example:



NLINCO : Names of nonlinear constraints	
stress 3	:Stress in stand
weight 4	: weight of stand
buckle 5	: strength of stand

Notes:

- If the name is not specified (i.e., six spaces are specified as the name) or if the NLINCO block is not specified, a default name of the form NLCO## is assigned to the nonlinear constraint, e.g., NLCO1, NLC037 etc.
- The user has the responsibility of matching the numbering scheme for the constraints in the data file with the numbers used in the FORTRAN file (in module USRSET).
- It is possible to add as many names as necessary. If the available space in a spreadsheet is insufficient, additional rows can be inserted to accommodate new names. This can be done by right-clicking on the empty cell and selecting the "Insert" option to add a new row. In this way,

the data set can be expanded to include any number of names for nonlinear constraints required for the problem at hand.

10. NLINGO (Optional):

Purpose: Define the names of nonlinear goals.

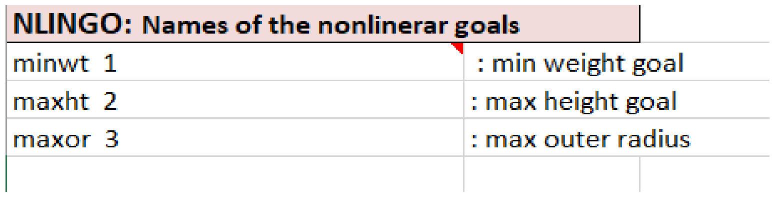
Format: (for each goal)

name k

Variables:

name: string	name of nonlinear goal(6 characters long)
k: integer	number of the nonlinear goal

Example:



NLINGO: Names of the nonlinerar goals	
minwt 1	: min weight goal
maxht 2	: max height goal
maxor 3	: max outer radius

11. ALPOUT (Optional):

Purpose: Output control.

Format:

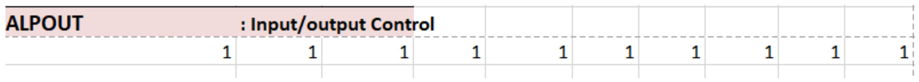
kl k2 k3 k4 k5 k6 k7 k8 j m

Variables:

kl: integer	= 1, print system variables (Default)
	= 0, do not print system variables
k2: integer	= 1, print deviation variables(Default)
	= 0, do not print deviation variables

k3: integer	= 1, print deviation function(Default)
	= 0, do not print deviation function
k4: integer	= 1, print bound information
	= 0, do not print bound information(Default)
k5: integer	= 1, print linear constraint information
	= 0, do not print linear constraint information(Default)
k6: integer	= 1, print nonlinear constraint information
	= 0, do not print nonlinear constraint information(Default)
k7: integer	= 1, print linear constraint activity for a linear solution
	= 0, do not print linear constraint activity for linear solution(Default)
k8: integer	= 1, print nonlinear constraint activity for a linear solution
	=0, suppress nonlinear constraint activity for linear solution(Default)
j: integer	= 1, print postprocessor information
	= 0, do not print postprocessor information(Default)
m: integer	= 1, print time statistics
	= 0, do not print time statistics(Default)

Example:



ALPOUT	: Input/output Control								
1	1	1	1	1	1	1	1	1	1

12. USRMOD (Optional):

Purpose: User module-related flags.

Format:

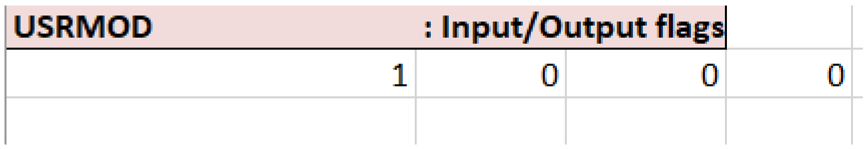
i j k l

Variables:

i: integer	= 1, User module USRINP to be used.
	= 0, USRINP not used (Default)
j: integer	= 1, User module USROUT to be used
	= 0, USROUT not used (Default)
k: integer	= 1, User module USRMON to be used
	= 0, USRMON not used (Default)
l: integer	= 1, User module USRLIN to be used
	= 0, USRLIN not used (Default)

Example:

Call USRINP only:



USRMOD	: Input/Output flags		
1	0	0	0

Notes:

- The user module USRINP is called before any computations are performed. It can be used to read problem specific data.
- The user module USROUT is called after all computations are performed. It can be used to perform customized output.
- The user module USRMON is called after every synthesis cycle. It can be used to obtain extra problem-specific output after each cycle.

• The user module USRLIN is called when the linear goals or coefficients have variable coefficients.

13. OPTIMP (Optional):

Purpose: Define optimization parameters.

Format:

a b c

Variables:

a: real Acceptable violation of nonlinear constraints (VIOLIM).

Negative real number.{ Default value: -0.001}

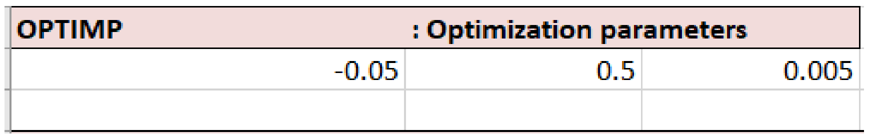
b: real Reduced move limit (REMO)

Real value between O and 1(Default value: 0.5)

c:real Perturbation step for linearization (STEP) specified as a fraction of the variable range.

Real value between O and 1(Default value: 0.005)

Example:



OPTIMP	: Optimization parameters	
-0.05	0.5	0.005

Notes:

VIOLIM defines the acceptable range by which a nonlinear constraint may be violated and yet be considered as being satisfied by SLIPML.

A larger value of REMO (e.g., REMO=l) will usually lead to faster convergence.

If the solution shows large fluctuations between cycles, use a small value of reduced move (e.g., 0.1).

The value of VIOLIM is used for all nonlinear constraints. To specify individual values for constraints, use the PVCVIL block.

The value of the reduced move is used for all system variables (except Boolean variables). To specify individual values for variables, use the PVREMO block. The optional PVREMO block is designed to provide users with the ability to select specific values for REMO when utilizing the advanced model feature. Due to its optional nature, this block has not been included in the main wrapper, and therefore no specific cell has been allocated for it. To utilize this feature, users will need to manually add a cell to the end of the sheet before the ENDPRB block. This can be done by clicking on the cell and selecting the "insert" option, followed by selecting the "entire row" to add a new line. Once the new row has been added, users can import the relevant data based on the instructions provided in PVREMO blocks.

The value of the perturbation step is used for all nonlinear functions. To specify individual values for functions, use the PVSTEP block.

Additional Optional Blocks:

The additional blocks described below are optional and may be useful for advanced users who require greater customization of their model. As such, these blocks have not been included in the main wrapper, and no specific cell has been allocated for them. If users wish to utilize these advanced features, they will need to manually add a cell to the end of the sheet before the ENDPRB block. This can be done by clicking on the desired cell and selecting the "insert" option, followed by selecting the "entire row" to add a new line. Users can then import the relevant data based on the instructions provided. We hope this information is helpful to our users who require greater flexibility and customization in their modeling. Since the following blocks are optional and we did not consider them in the Elephant Stand problem, some other examples have been presented.

14.. INITFS:

Purpose: Automatic generation of initial feasible solution.

Format:

n a b c d e

Variables:

n: integer	Maximum allowable number of calls to USRSET.
a: real	Expansion factor for pattern search > 1 (Recommended value 2.0)
b: real	Contraction factor for pattern search < 1 (Recommended value 0.5)
c: real	Acceptable value for the sum of violations (Small value)
d: real	Step size as a fraction of range (0 <d < 1)
e: real	Minimum allowable fraction of the initial step.

Notes:

• The very presence of the INITFS block acts as a flag for the pattern search to be executed. If you need to tum this option off, remove the block from your data file or inactivate it by placing it after the ENDPRB block.

• The initial feasible solution is generated using the Hook-Jeeves pattern search algorithm with ridge adaptation.

• The pattern search will call the user module USRSET at most n times. When the analysis is time-consuming, this number can be kept small (e.g., 40). Otherwise, use a large number like 400.

• The pattern search is terminated if the sum of absolute violations is smaller than c.

• The pattern search is initialized with a step size based on the bounds and the parameter d.

$STEP_i$ = Step size* (Max_i - Min_i)

• The pattern search is terminated if the step size multiplier becomes smaller than e. The initial multiplier is taken as unity (=1). This parameter governs the accuracy of the final solution of the pattern search.

15. USRDAT (Optional):

Purpose: This block contains data that can be read from USRINP.

Format:

K

$textline_1$

…

$textline_k$

Variables:

k: integer	Number of lines of text to follow.
textline	80 characters (maximum) of text

Note:

• This should only be used when a relatively small amount of data needs to be read by USRINP and when the effort of maintaining an additional data file is considered unnecessary.

16. ADPCTL (Optional):

Purpose: Nonlinear inequality constraint adaption flag.

Format:

j

Variables:

j: integer	= 1, Use nonlinear constraint/goal adaptation
	= 0, No adaptation used (Default)

17. USERAN (Optional):

Purpose: Parameters related to the user analysis module USRANA.

Format:

1

k_1 k_2 ... k_i

Variables:

i: integer	Number of analysis cycles (NANCY)
k: integer	Number of synthesis cycles within each analysis cycle (NYSCY)

Example:

USERAN	: USRANA related parameters
3	: NASC # of synthesis cycles
20 30 15	: NANSYC(I)

18. FIXVAR (Optional):

Purpose: Fix system variables to initial guess values.

Format:

1

k1 k2 ... ki (free format list)

Variables:

i: integer	Number of system variables to be fixed.
k: integer	Serial number of variable to be fixed.

Example:

FIXVAR : Fixing Variables
2 1 4 Fix X1 and X4 to constant Values.

19. SUPCON (Optional):

Purpose: Suppress nonlinear constraints.

Format:

i

k1 k2 ... ki (free format list)

Variables:

i: integer	Number of system constraints to be fixed.
k: integer	Serial number of constraint to be suppressed.

Example:

SUPCON : Suppressing constraints.
1 2 : suppress the stress constraint.

20. PAVALFX (Optional):

Purpose: Particular values for stationarity of system variables.

Format:

k

(i_1, r_1) (... , ...) ... (i_k, r_k)

Variables:

k: integer	Number of values to follow.
i: integer	Serial number of variable

r: real Particular value of FRACX as a fraction of variable range defined by the bounds.

21. PVALFZ (Optional):

Purpose: Particular values for stationarity of deviation function level.

Format:

k

(i1, ri) (... , ...) ... (ik, fJc)

Variables:

k: integer	Number of values to follow
i: integer	Priority level number
r:real	Particular value of EPSZ for this level

Example:

PVALFZ	Stationarity of deviation function
1	
(1,0.001)	Need accurate convergence for first level goals

22. PVSTEP (Optional):

Purpose: Perturbation step size for particular set of variables.

Format:

k

(f_1,s_1) (... , ...) ... (f_k,S_k)

Variables:

k: integer	Number of values to follow.
f: integer	Serial number of system variable being perturbed.
s: real	Perturbation step size specified as a fraction of the variable range.

Example:

PVSTEP	: particular values for steps
1	
(2, 0,02)	: step size for variable 2

23. PVCVIL (Optional):

Purpose: Particular values for nonlinear constraint satisfaction (VIOLIM).

Format:

k

(m_1,s_1) (..., ...) ... (m_k,S_k)

Variables:

k: integer	Number of values to follow.
m: integer	Serial number of nonlinear constraint.
s: real	acceptable violation for nonlinear constraint (< 0.0)

Example:

PVCVIL	: Nonlinear constraint violations
1	
(3, -0.005)	: strict control for constraint #3

24. PVREMO (Optional):

Purpose: Particular values for the reduced move.

Format:

k

(i_1, r_i) (... , ...) ... (i_k, f_k)

Variables:

k: integer	Number of values to follow.

i: integer	Serial number of variable.
r: real	Reduced move size.

Example:

PVREMO	**:particular values for reduced move**
(1,0.1) (3,0.1)	: Reduced move for variables 1 and 3

Note:

• This can be useful when a small number of variables are not converging, but most of the variables are converging. Set the reduced move for these variables at a small value.

25. ADREMO (Optional):

Purpose: Use adaptive reduced move algorithm.

Format:

k a

Variables:

k: integer	Maximum number of calls to USRSET in each cycle
a: real	Convergence criterion for the reduced move, R. { Recommended value: 0.05}

Example:

ADREMO	: adaptive reduced move
10 0.05	: Max . calls, deltaR

Notes:

• The adaptive reduced move is performed using a combination of linear search and a Golden section Search algorithm to find the value of R_{best} ($0\leq R \leq 1$) which will minimize the deviation function along the line joining the old solution point, X_{old}, to the new solution point, X_{new}.

$X_{best} = X_{old} + R_{best} [X_{new} - X_{old}]$ where $O \leq R_{best} \leq 1$

If point X_{new} is infeasible, the search range is contracted to lie between O and 0.5. This contraction is repeated until a feasible value of R is found or the interval becomes smaller than the convergence criterion given by a.

• The maximum number of calls to USRSET, to achieve convergence to a given value of a, can be calculated from the following equation

$N_{max} = 4.82 \log10 (1/a) + 2$

The maximum number of times that USRSET is actually called is the smaller of the two numbers - k and $N_{max.}$

26. XPLORE (Optional):

Purpose: Explore the solution space to find a feasible starting point.

Format:

j_1 j_2 j_3 j_4

i

k_1 k_2 ... k_i (free format list)

Variables:

j1: integer	Number of points to be generated (NPTGEN)
j_2: integer	Number of best points to be selected.
j_3: integer	Flag to print best points in standard output file.
	= 0, do not print best points
	= 1, print best points
$j_{4:}$ integer	Integer seed number for pseudo-random number generation (positive integer less than 2531)

i: integer Number of system variables to be fixed at initial values during the search procedure.

k: integer Serial number of variable to be fixed.

Example:

```
 XPLORE       : Exploring Design space
3000  20  1  1233
3
1   4                  : fix X1 and X4 to constant values during search
```

Notes:

- The points in the design space are generated using two different methods depending on the number of points requested. If NPTGEN > 2^m, then we use the systematic search algorithm of Aird and Rice (1977) to generate the points. Otherwise, we generate points using random numbers. Here mis the number of variables which are not fixed.
- The best points are stored in a file named ALPPTS.DAT.

At this stage, all the necessary information regarding the Linear Archimedean sheet has been described. The focus will now shift to the Linear Preemptive sheet, which shares many similarities with the previous sheet, but with some minor differences to be discussed in the subsequent section. The forthcoming section will provide a comprehensive overview of the Linear Preemptive sheet, highlighting its distinct features and functionalities.

II.2.2 Linear Preemptive Sheet

In this sheet we are importing linear information regarding the preemptive form. As we described in Section 1.3.6, in the Preemptive approach, the goals are rank ordered to avoid the difficulty of assigning weights. The measure of achievement is obtained in terms of the lexicographic minimization of an ordered set of goal deviations. Weights may be used within each set of goals at a particular rank, but the goals are ranked lexicographically, and an attempt is made to achieve a more important goal before considering other goals. The deviation function in the Preemptive formulation is written as an ordered array of nonnegative elements. In other words, to define the formulation, everything is exactly the same; the only difference is in defining goals. We need to define levels for preemptive since, in Archimedean, levels are equal to 1.

To implement the formulation in this sheet, follow all the rules mentioned in the LinearArchimedian sheet. The difference is to define the deviation function. In the LinearArchimedian sheet, there is a DEVFUN option, which includes only one level, but there is an ACHFUN section in this sheet.

ACHFUN:

Purpose: Define Deviation Function - number of levels and weights of deviation variables. It is used in Preemptive form.

Format: (for each linear goal)

K

L n

(m_1,w_1) (... , ...) ... (m_k,W_k)

Variables:

k: int	Number of levels
L: int	Level number.
n: int	Number of goals in this level.
m: integer	signed integer indicating deviation variable.
	(+m for overachievement and -m for underachievement)

w: real weight for deviation variable

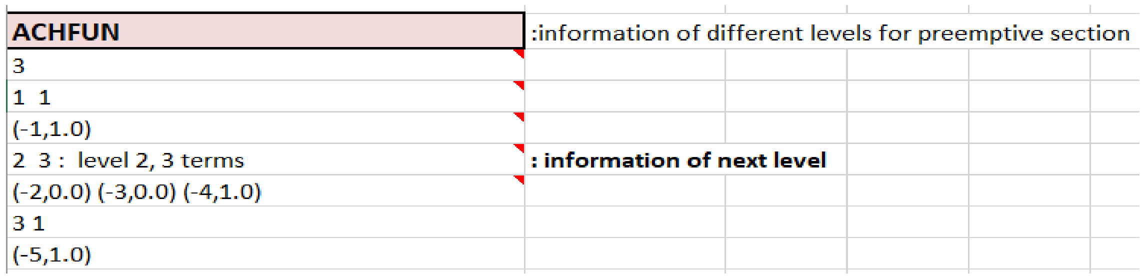
Example:

In the following example, the 3-level preemptive form has been represented, which includes one goal in the first level, three goals in level two (all weights are assigned to the third goal), and one goal in level three.

Deviation functions (pre-emptive form):

$z = [\, e_1^- , \sum_{i=1}^{3} wi \,.\, (d_i^- + d_i^+), d_4^-], \sum_{i=1}^{3} wi = 1$ (w_i are weights of the various compromise goals)

where deviation variables are: $e_1^-, e_1^+, d_1^+, d_1^-, d_2^+, d_2^-, d_3^+, d_3^-, d_4^-, d_4^+$



ACHFUN	:information of different levels for preemptive section
3	
1 1	
(-1,1.0)	
2 3 : level 2, 3 terms	**: information of next level**
(-2,0.0) (-3,0.0) (-4,1.0)	
3 1	
(-5,1.0)	

Note:

• It is possible to add as many levels as necessary. If the available space in a spreadsheet is insufficient, additional rows can be inserted to accommodate new levels. This can be done by right-clicking on the empty cell and selecting the "Insert" option to add a new row. In this way, the data set can be expanded to include any number of levels and the information required for the problem at hand.

All other information would be the same as LinearArchimedian and follow the same rules.

At this stage, all the necessary input information for the linear aspect of the problem has been described. The subsequent section will shift the focus toward the nonlinear portion of the problem, wherein its intricate details will be thoroughly examined. This forthcoming section will provide an in-depth analysis of the nonlinear component, covering its various intricacies and complexities.

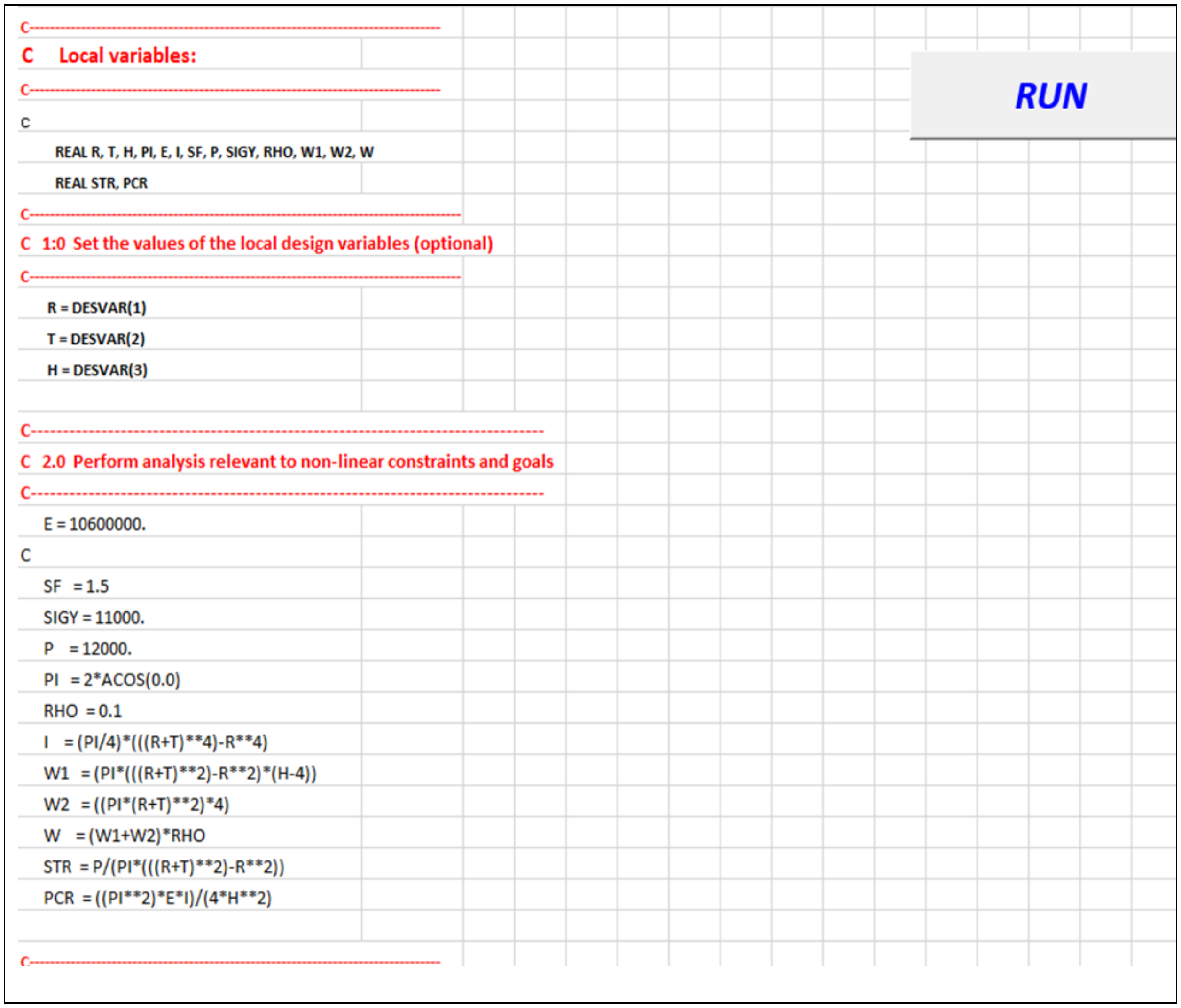
II.2.3 Nonlinear Sheet

To formulate and solve a Compromise DSP, in addition to the LinearArchimedian or LinearPreemptive sheet that is described in Sections II.2.1 and II.2.2, we need to input the required information in the third sheet, which is the Nonlinear sheet. In this part of the program, the following six user specified subroutines are required:

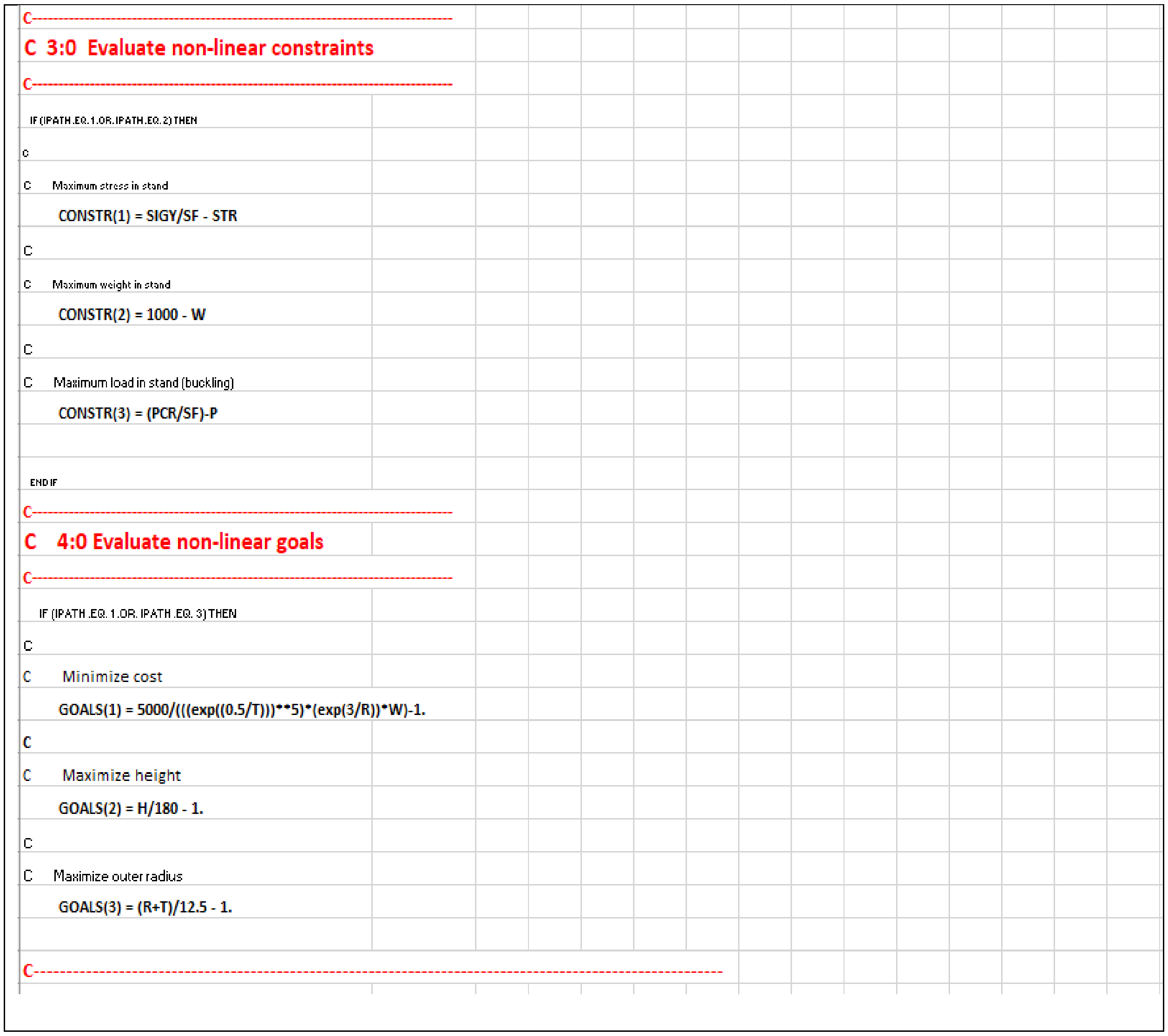
- USRINP (for user specific input),
- USRSET (for evaluating nonlinear constraints and nonlinear goals),
- USRLIN (for updating linear constraint and linear goal coefficients),
- USRMON (for user specific monitoring of the solution process),
- USRANA (for relevant analysis cycle calculations), and
- USROUT (for user specific output).

In order to streamline the user experience and facilitate quick identification of the necessary input information, the NonLinear sheet incorporates distinct visual elements. These include specific sections highlighted in larger font sizes and colored in red. Within these prominent red sections, users are required to import essential information such as local variables, details pertaining to nonlinear constraints, and specifications for nonlinear goals (those are related to the Subroutine USRSET). This design approach aims to enhance user efficiency and ensure that the crucial input parameters are readily apparent for the user's attention and completion.

To provide users with a comprehensive understanding of the NonLinear sheet, I have included visual representations in the form of images for this sheet in Figure II.6. With this image, users could have a holistic view of the layout to grasp the appearance and structure of the NonLinear sheet at a glance.



```
C-----------------------------------------------------------------
C    Local variables:
C-----------------------------------------------------------------
C
      REAL R, T, H, PI, E, I, SF, P, SIGY, RHO, W1, W2, W
      REAL STR, PCR
C-----------------------------------------------------------------
C  1:0  Set the values of the local design variables (optional)
C-----------------------------------------------------------------
     R = DESVAR(1)
     T = DESVAR(2)
     H = DESVAR(3)

C-----------------------------------------------------------------
C  2.0  Perform analysis relevant to non-linear constraints and goals
C-----------------------------------------------------------------
    E = 10600000.
C
    SF   = 1.5
    SIGY = 11000.
    P    = 12000.
    PI   = 2*ACOS(0.0)
    RHO  = 0.1
    I    = (PI/4)*(((R+T)**4)-R**4)
    W1   = (PI*(((R+T)**2)-R**2)*(H-4))
    W2   = ((PI*(R+T)**2)*4)
    W    = (W1+W2)*RHO
    STR  = P/(PI*(((R+T)**2)-R**2))
    PCR  = ((PI**2)*E*I)/(4*H**2)

C-----------------------------------------------------------------
```



```
C-----------------------------------------------------------------------
C  3:0  Evaluate non-linear constraints
C-----------------------------------------------------------------------
 IF (IPATH.EQ.1.OR.IPATH.EQ.2) THEN
C
C      Maximum stress in stand
        CONSTR(1) = SIGY/SF - STR
C
C      Maximum weight in stand
        CONSTR(2) = 1000 - W
C
C      Maximum load in stand (buckling)
        CONSTR(3) = (PCR/SF)-P

 END IF
C-----------------------------------------------------------------------
C   4:0 Evaluate non-linear goals
C-----------------------------------------------------------------------
  IF (IPATH .EQ. 1.OR. IPATH .EQ. 3) THEN
C
C       Minimize cost
        GOALS(1) = 5000/(((exp((0.5/T)))**5)*(exp(3/R))*W)-1.
C
C       Maximize height
        GOALS(2) = H/180 - 1.
C
C      Maximize outer radius
        GOALS(3) = (R+T)/12.5 - 1.

C-----------------------------------------------------------------------------------------------------
```

Figure II. 6: Overview of Nonlinear sheet

Important Note:

When importing input data into the sheet, it is crucial to adhere to the formatting rules, which follow the convention of Fortran, the primary language used in the DSIDES program. According to this convention, it is necessary to begin each line with 8 empty spaces before adding the content.

To illustrate, let us consider a scenario where we need to define integer variables named "a" and "b." To properly adhere to the formatting rules, we would begin by inserting 8 empty spaces at the start of each line. Following that, we would write "Integer" to indicate the variable type and then import "a" and "b" on separate lines. Adhering to this specific formatting ensures compatibility with the DSIDES program and maintains consistency with the Fortran language conventions. By incorporating 8 empty spaces at the beginning of each line before adding content, we can effectively import the input data in a manner that aligns with the requirements of the system.

It is essential to know that the absence of any of these six subroutines will result in a warning and/or error message at link time, depending on the system being utilized. The consequence of not providing a subroutine is system dependent. In order to avoid problems, the user should provide at least a "dummy" subroutine within a DSP template such that all external calls are satisfied. The recommended "dummy" routines are defined in Section II.2.3.7.

Since these subroutines are essentials to be in this sheet, we will go through each of them to describe how they will work in the main program.

II.2.3.1 SUBROUTINE USRINP

Subroutine USRINP provides the user with a data input wrapper. By defining a user common block, the relevant data read through this wrapper can then be made available to other user specified subroutines (i.e., the other 5 ALP mandatory user supplied routines or any appropriate additional analysis routines). The desire to actively call subroutine USRINP is flagged through the optional ALP Data Block USRMOD that has been described in Section II.2.1. The first Boolean flag (0/1) specified in USRMOD accommodates the USRINP switch.

To minimize the number of files a user has to provide and maintain, the data to be read by USRINP can be written into the ALP Data Block USRDAT . Again, for details of USRDAT, see II.2.1 in part

12. If USRDAT is included in the active portion of the standard compromise data file, the lines of "text" are written to a temporary file. This file is then rewound, and the unit number of this temporary file is passed in the USRINP argument list . It is then necessary for the user to write in USRINP a set of matching read statements for these lines of "text".

Alternatively, if the ALP Data Block USRDAT is not to be used, all other legal FORTRAN 1/0 constructs could be used in USRINP. For example, USRINP could be written to query the user interactively during runtime or to access some other file or set of files.

```
C******************************************************************
C  Subroutine USRINP
C
C   Purpose:     To facilitate the input of domain dependent information
                          which can be stored in a COMMON (e.g., COMMON/USER/ ...)
C                     and shared with other user specified routines as required.
c-------------------------------------------------------------------

c Arguments                Name       Type              Description
C ---------
C  Input:                   NDESV     int            number of design variables
C                           NINP      int            unit number of the input data file
C                           NOUT      int            unit number of the output data file
C                           DESVAR    real           vector of design variables

C  Output :                 none

C Input/Output:               none
c-------------------------------------------------------------------
C  Common Blocks:        none
C
C  Include Files:        none
C
C  Called from:      ALPCTL
C
C  Calls to:         none
c-------------------------------------------------------------------
C Development History
C
C Author:
```

```
C Date:
C
C Modifications:
C
C***********************************************************************
c
      SUBROUTINE USRINP (NDESV, NINP, NOUT, DESVAR)
C
c--------------------------------------
c         Arguments:
c--------------------------------------
      INTEGER NDESV, NINP, NOUT
      REAL DESVAR(NDESV)
C--------------------------------------
c         Local variables:
c--------------------------------------
C
COMMON/USER/ ...
C
C
      Any legal FORTRAN statements.
C
     RETURN
     END
```

II.2.3.2 SUBROUTINE USRSET

The following subroutine is implemented within the NonLinear sheet of the wrapper, allowing users to import essential data. Its purpose is to evaluate the nonlinear constraints and goals effectively. This subroutine is used to evaluate the nonlinear constraints and goals. It is important that in specifying the constraints, the inequality constraints precede the equality constraints. Failure to do so will result in misinterpretation of the evaluation results by the main program. As far as possible, it is recommended to formulate the constraints so that an increase in x represents an increase in feasibility. All nonlinear constraints must be specified in the form LHS > 0. 0 or LHS = 0. 0. That is, the returned value (CONSTR) should be greater equal than 0.0 for feasibility.

Numbering of the indices for constraints and goals within Subroutine USRSET should begin with 1 and run to NNLCON and NNLGOA, respectively.

In this subroutine, there are specific sections highlighted in larger font sizes and colored in red on the sheet. Within these prominent red sections, users are required to import essential information such as local variables, Set the values of the local design variables (optional), Perform analysis relevant to nonlinear constraints and goals, evaluate nonlinear constraints, and evaluate nonlinear goals.

Once the necessary information has been imported into the NonLinear sheet, the user can initiate the computation process by clicking the "Run" button. This action triggers the program to execute the calculations based on the provided input. Subsequently, an output file is generated, encompassing the values of all variables involved in the analysis. This output file is automatically saved in the main folder of the software. In the event that the user decides to run the analysis again by clicking the "Run" button for the second time, the output file is conveniently displayed on the screen. This enables the user to access and review the results promptly, facilitating a seamless and iterative workflow.

```
C*********************************************************************
C Subroutine USRSET
c
C Purpose:   Evaluate nonlinear constraints and goals.
C          NOTE - Do not specify the deviation variables
c--------------------------------------------------------------------
c Arguments                  Name       Type              Description
C -------------------------------------------------------------------------------------------
C  Input:                     IPATH        int          = 1        evaluate constraints and goals
C                                                      = 2        evaluate constraints only
C                                                      = 3        evaluate goals only

C                            NDESV       int          number of design variables
C                            MNLNCG     int          maximum  number  of  nonlinear  constraints
```

```
and goals
C                                   NOUT          int          unit number of output data file
C                                   DESVAR     real           vector of current system variables

C  Output :                         CONSTR     real           vector of constraint values
C                               GOALS            real         vector of goal values
C Input/Output:                        none
c--------------------------------------------------------------------
c Common Blocks:             USER
C Include Files:               none
 C Called from:            GCALC
c Calls to:                none
c--------------------------------------------------------------------
c Development History
C
C Author:
c Date:
C
C Modifications:
C*******************************************************************
c-
      SUBROUTINE USRSET (IPATH, NDESV, MNLNCG, NOUT, DESVAR,
      &                              CONSTR, GOALS)
C
c--------------------------------------
c         Arguments:
c--------------------------------------

       INTEGER  IPATH, NDESV, MNLNCG, NOUT
       REAL DESVAR(NDESV)
       REAL CONSTR(MNLNCG), GOALS(MNLNCG)
C
c--------------------------------------
c         Local variables:
c--------------------------------------
C
      COMMON/USER/ ...
C
C         1.0 Set the values of the local design variables (optional)
C
      Any legal FORTRAN statements.
C
C         2.0 Perform analysis relevant to nonlinear constraints and goals
C
```

```
      Any legal FORTRAN statements
C
C
C        3.0 Evaluate nonlinear constraints
C
        IF (IPATH .EQ. 1 .OR. IPATH .EQ. 2) THEN
C
      Any legal FORTRAN statements.
      Perform analysis relevant to nonlinear constraints.
      Specify CONSTR(J) for each constraint where J = 1, NNLCON
C
      END IF
C
C
C        4.0 Evaluate nonlinear goals
C
      IF (IPATH .EQ. 1 .OR. IPATH .EQ. 3) THEN
C
        Any legal FORTRAN statements.
C
      Perform analysis relevant to nonlinear goals
C
       Specify GOALS( J) for each goal where J = 1, NNLGOA
C
       END IF
C
C        5.0 Return to calling routine
C
       RETURN
        END
```

II.2.3.3 SUBROUTINE USRLIN

The Purpose of this subroutine is to facilitate the modification of linear constraint and linear goal coefficients that are functions of system variables. One example of use is the revision of merit function values in coupled problems involving selection.

```
C*******************************************************************
C Subroutine USRLIN
C
```

```
C Purpose:       To facilitate the modification of linear constraints and
C                  linear goal coefficients that are functions of the system
C                  variables. One example of use is the revision of merit
 C                 function values in coupled problems involving selection.
c---------------------------------------------------------------------
C  Arguments    Name      Type                  Description
C Input:           MLINCG      int                    maximum total number of linear constraints and
goals
C                   NLINCO      int                    number of linear constraints
C                   NDESV                int           number of design variables
C                   NLINGO     int          number of linear goals
C                   NOUT      int              unit number of the output data file
C                    DESVAR      real            vector of current system variables

C   Output:            none
C Input/Output:           COFLIN             real   matrix of coefficients of linear  constraints and
goals
C                   RHSLIN     real vector of RHS values for linear constraints and goals
c---------------------------------------------------------------------
C Common Blocks:    USER
C Include Files:        none
C Called from       DFCALC
C Calls to             none
c---------------------------------------------------------------------
C Development History
C
C Author:
C Date:
C
C Modifications:
C*********************************************************************
c-
      SUBROUTINE USRLIN (MLINCG, NDESV, NLINCO, NLINGO, NOUT,
&         DESVAR, COFLIN, RHSLIN)
C
c--------------------------------------
c          Arguments:
c--------------------------------------
c
      INTEGER MLINCG, NDESV, NLINCO, NLINGO, NOUT
      REAL DESVAR(NDESV), COFLIN(MLINCG,NDESV), RHSLIN(NLINCO+NLINGO)
C
c--------------------------------------
c          Local variables:
```

```
c--------------------------------------
c
      COMMON/USER/ ...
C
      Any legal FORTRAN statements.
C
      RETURN
      END
```

II.2.3.4 SUBROUTINE USRMON

```
C*******************************************************************
C Subroutine USRMON
C
C Purpose:      To facilitate monitoring of the solution process and
C                 to provide the opportunity to generate additional output
C                (Called from within ALPMOD prior to calling linear solver)
c--------------------------------------------------------------------

c Arguments                Name       Type              Description
C ---------
C Input:                   NDESV      int              number of design variables
C                          NMPRI         int           number of priority levels
C                          NNLCON        int           number of nonlinear constraints
C                          NNLGOA        int           number of nonlinear goals
C                          NOUT       int                 unit number of output data file
C                          CONDEV        real                 total constraint violation
C                          DESVAR        real                 vector of current system variables
C                          DEVFUN     real             vector of deviation function values
C                          DEWAR      real             vector of deviation variables
C                          GVAL       real             vector of nonlinear constraint and goal
values

C Output :                 none

C Input/Output:         none
c----------------------------------------------------·---------------
Common Blocks:      USER
C Include Files:         none
C Called from:      ALPMOD
C Calls to:            none
c--------------------------------------------------------------------
C Development History
```

```
C
C Author:
C Date:
C
C Modifications:
C*********************************************************************
c-
      SUBROUTINE USRMON (NDESV, NDEVAR, NMPRI, NNLCON, NNLGOA, NOUT,
&         DESVAR, DEVVAR, CONDEV, DEVFUN, GVAL)
C
c--------------------------------------
c         Arguments:
c--------------------------------------
c
            INTEGER        NOUT, NDESV, NDEVAR, NMPRI, NNLCON, NNLGOA
            REAL           DESVAR(NDESV), DEVVAR(NDEVAR),
      & CONDEV, DEVFUN(NMPRI), GVAL(NNLCON+NNLGOA)
C
c--------------------------------------
c         Local variables:
c--------------------------------------
c
COMMON/USER/ ...
C
```
Any legal FORTRAN statements.
```
C
        RETURN
        END
```

II.2.3.5 SUBROUTINE USRANA

```
C*********************************************************************
C Subroutine USRANA
C
C Purpose:      To facilitate access to analysis packages too costly to C       call within a synthesis
cycle
c--------------------------------------------------------------------
 c Arguments                Name       Type             Description

C  Input:                   NDESV       int             number of design variables
                            NOUT        int             unit number of output data file
                            DESVAR      real            vector of current system variables
```

```
C Output:                    none

C Input/Output:        none
c--------------------------------------------------------------------
C Common Blocks:    USER
C Include Files:        none
C Called from:         ALPCTL
C Calls to:             none
c--------------------------------------------------------------------
C************* *******************************************************
SUBROUTINE USRANA (NDESV, NOUT, DESVAR)
C
c-------------------------------------
c         Arguments:
c-------------------------------------
            INTEGER     NDESV, NOUT
             REAL          DESVAR(NDESV)
C
c-------------------------------------
c         Local variables:
c-------------------------------------
c
C
COMMON/USER/ .....
C
     Any legal FORTRAN statements.
C
      RETURN
      END
```

II.2.3.6 SUBROUTINE USROUT

```
C*******************************************************************
C Subroutine USROUT
C
C Purpose:      To facilitate the customization of the final output
c--------------------------------------------------------------------

c Arguments                     Name       Type          Description
C Input:                        NOUT          int         unit number of output data file
C                               DESVAR        real        vector of design variables
```

```
C                              LCONDF        lgcl          TRUE if deviation function converged.
C                              LCONSV        lgcl          TRUE if design variables converged
C                              LXFEAS        lgcl          TRUE if vector of design variables represents
a feasible design
C Output:                       none
C Input/Output:                 none
C
C Common Blocks:               none
Include Files:                 none
 Called from:                  ALPCTL
Calls to:                      none

c------------------------------------------------------------------------
C***********************************************************************
c
      SUBROUTINE USROUT (NDESV, NOUT, DESVAR, LCONDF, LCONSV, LXFEAS)
C
c----------------------------------------
c         Arguments:
c----------------------------------------
      INTEGER   NDESV, NOUT
     REAL   DESVAR(NDESV)
     LOGICAL  LCONDF, LCONSV, LXFEAS
C
c-------------------.------------------
c         Local variables:
c----------------------------------------
C
COMMON/USER/ .....
C
C
```
Any legal FORTRAN statements.
```
C
      RETURN
       END
```

II.2.3.7 MINIMUM DUMMY SUBROUTINES

The minimum user specified "dummy" subroutines are defined as given below.

Minimum USRINP "dummy" subroutine

```
      SUBROUTINE USRINP (NDESV, NINP, NOUT, DESVAR)
C
C*** DUMMY ROUTINE. Not used.
C
      INTEGER NDESV, NINP, NOUT
      REAL DESVAR(NDESV)
C
      RETURN
      END
```

Minimum USRSET "dummy" subroutine

```
      SUBROUTINE USRSET (IPATH, NDESV, MNLNCG, NOUT, DESVAR,
&        CONSTR, GOALS)
C
C*** DUMMY ROUTINE. Not used in the formulation
C
      INTEGER IPATH, NDESV, MNLNCG, NOUT
      REAL DESVAR(NDESV)
      REAL CONSTR(MNLNCG), GOALS(MNLNCG)
C
      RETURN
      END
```

Minimum USRLIN "dummy" subroutine

```
      SUBROUTINE USRLIN (MLINCG, NDESV, NLINCO, NLINGO, NOUT,
&        DESVAR, COFLIN, RHSLIN)
C
C*** DUMMY ROUTINE. Not used in the formulation
C
      INTEGER MLINCG, NDESV, NLINCO, NLINGO, NOUT
      REAL DESVAR(NDESV), COFLIN(MLINCG,NDESV), RHSLIN(NLINCO+NLINGO)
C
      RETURN
      END
```

Minimum USRMON "dummy" subroutine

```
      SUBROUTINE USRMON (NDESV, NDEVAR, NMPRI, NNLCON, NNLGOA, NOUT,
&        DESVAR, DEVVAR, CONDEV, DEVF.UN, GVAL)
C
```

```
C*** DUMMY ROUTINE. Not used in the formulation
C
      INTEGER NOUT, NDESV, NDEVAR, NMPRI, NNLCON, NNLGOA REAL DESVAR(NDESV),
DEVVAR(NDEVAR),
&         CONDEV, DEVFUN(NMPRI), GVAL(NNLCON+NNLGOA)
C
      RETURN
      END
```

Minimum USRANA "dummy" subroutine

```
      SUBROUTINE USRANA (NDESV, NOUT, DESVAR)
C
C*** DUMMY ROUTINE. Not used in the formulation
C
      INTEGER NDESV, NOUT REAL DESVAR(NDESV)
C
      RETURN
      END
```

Minimum USROUT "dummy" subroutine

```
       SUBROUTINE USROUT (NDESV, NOUT, DESVAR, LCONDF, LCONSV, LXFEAS)
C
C*** DUMMY ROUTINE. Not used in the formulation
C
      INTEGER NDESV, NOUT
      REAL DESVAR(NDESV)
      LOGICAL LCONDF, LCONSV, LXFEAS
C
      RETURN
      END
```

II.2.3.7 LIST OF RESERVED NAMES

The following is a list of symbols (SUBROUTINE names, FUNCTION names and COMMON block names) which are reserved for use by DSIDES and should not be used in the user's FORTRAN program.

ADNUPT	GETNAM	PRBNDS	SELPRl

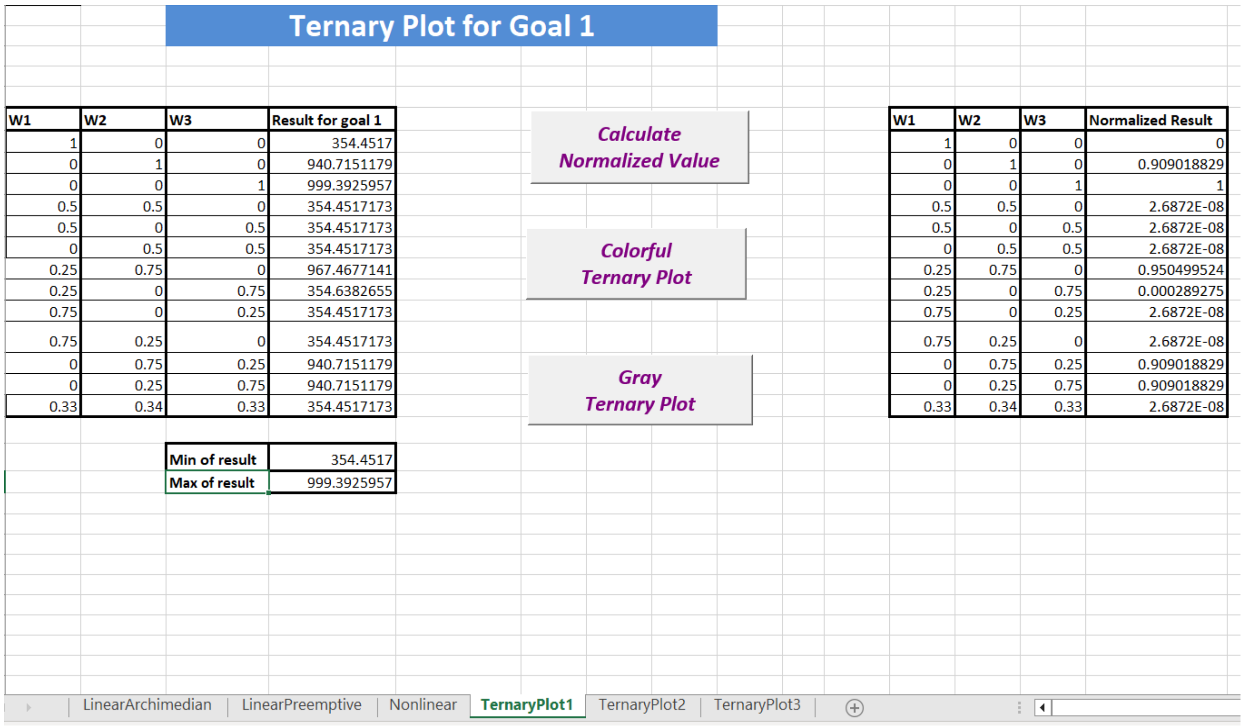
ADREMO	HJLIMS	PRDESV	SELPRT
ADVUSl	HJMAIN	PRDEW	SELPRT
ALPDAT	HJTEMP	PRDEW	SELREA
ALPMOD	IARFIL	PRDFUN	SELSEN
BLKCHK	INITAB	PRFACT	SPLITC
CGRAPH	INITSL	PRICEV	SRAN
CHKROW	INJFND	PRLINA	SRCH
COEFF2	INTMUL	PRLINC	STRNEW
CONACC	INVERS	PRNLNA	STROLD
CONCHK	IQSIGN	PRNLNC	TIMER
CONCOR	LEXSML	PRPNT	TRFORM
CONLIN	LUBKSB	PRPPCF	UPDTAB
CONVER	LUDCMP	PRPPIF	VECMAX
CONVIL	LVCEQU	PTRAND	VECSUM
COPYTX	MATMUL	PTSRCH	XPLORE
DCOMP	MKNAME	RANCOM	ZERONE
DELACC	MLINOP	RDCOST	ZSRCH
DERIV	MPLEX	REINVR	SELDAT
DEVCAL	MPLEX	RTEST·	SELCYC
DFCALC	MSETUP	SEIGHT	PRADPC
ECOMP	MULTI1	SELCAL	PCIMPR
EIGEN	MULTI2	SELCHR	NEWVAR

ENTVAR	ETAVEC	EXPTAB	NEWRHS
FDATE	GCALC	NEWINV	SELINT
SELLOG	SELMER		

At this juncture, all the necessary information and sheets have been described, enabling the smooth execution of the DSIDES software. Upon running the linear and nonlinear sheets, an output file is generated, either in the form of a popup Notepad file displayed on the screen to or saved as an outfile type in the main folder. A comprehensive description of the output file can be found in Section II.3, which elucidates how to interpret its contents. Before delving into the intricacies of the output file, it is imperative to address the remaining three sheets in Section II.2.4, which revolve around ternary plots. These forthcoming sections will provide a thorough exploration of ternary plots, encompassing their functionalities and guiding users on effectively utilizing them.

II.2.4 Ternary Plot Sheets

Within the DSIDES's wrapper, there are six sheets available to users: LinearArchimedean, LinearPreemptive, NonLinear, TernaryPlot1, TernaryPlot2, and TernaryPlot3. In this section, our focus will be on the last three sheets, specifically dedicated to the plotting of ternary diagrams for each individual goal. Figure II.7 is the general looking of Ternary Plot sheet.



Ternary Plot for Goal 1

W1	W2	W3	Result for goal 1
1	0	0	354.4517
0	1	0	940.7151179
0	0	1	999.3925957
0.5	0.5	0	354.4517173
0.5	0	0.5	354.4517173
0	0.5	0.5	354.4517173
0.25	0.75	0	967.4677141
0.25	0	0.75	354.6382655
0.75	0	0.25	354.4517173
0.75	0.25	0	354.4517173
0	0.75	0.25	940.7151179
0	0.25	0.75	940.7151179
0.33	0.34	0.33	354.4517173

Min of result	354.4517
Max of result	999.3925957

W1	W2	W3	Normalized Result
1	0	0	0
0	1	0	0.909018829
0	0	1	1
0.5	0.5	0	2.6872E-08
0.5	0	0.5	2.6872E-08
0	0.5	0.5	2.6872E-08
0.25	0.75	0	0.950499524
0.25	0	0.75	0.000289275
0.75	0	0.25	2.6872E-08
0.75	0.25	0	2.6872E-08
0	0.75	0.25	0.909018829
0	0.25	0.75	0.909018829
0.33	0.34	0.33	2.6872E-08

Figure II. 7: Overview of TernaryPlot sheet

In order to facilitate human-based decision-making, users would be able to consider various scenarios and have weights for each goal imported based on their preferences.

Each sheet contains two different tables. The first table on the left side of the sheet consists of four columns and 13 rows (the user can define up to 13 variations by assigning different weights to the goals). In the first three columns the user could assign different weights to each goal, ensuring that the sum of the weights for the three goals is equal to one. $\sum_{i=1}^{3} wi = 1$

The last column calculates the goal value based on the defined weights and the value of variables that calculated and represented in the output file output file has been interpret in the Section II.3. After importing the weights and the corresponding results for each goal, the user can save the data. Then by pressing the "Calculate the Normalized Value" button, the system calculates the normalized value for each goal based on the defined scenarios. The results are displayed in the second table on the right-hand side of the sheet.

Once the necessary information for plotting the ternary plots is ready, the user is presented with two options: a colorful plot or a gray plot. By simply pressing the respective buttons, the program automatically connects to MATLAB, retrieves the input data from the Excel sheet, and generates the chosen plot. The user can then save the plot for further analysis and documentation. (User could follow same rules for all three sheets for goal 1 and goal 2 and Goal 3.)

Using DSIDES's wrapper engineers and designers are able to visualize the trade-offs among multiple goals in a decision-making process. A graphical representation of the relationships between the goals is offered by the ternary plots, providing valuable insights into the decision space. These plots are used as a tool for understanding the impact of different weight assignments and scenarios, aiding users in making informed decisions based on their preferences.

A visually appealing representation is provided by the colorful plot option, with distinct colors assigned to each region within the ternary diagram and users could quickly identify the dominant goal for specific scenarios or combinations of weights. On the other hand, a simplified, monochromatic representation is offered by the gray plot option, which may be preferred in cases where color differentiation is not necessary or desired.

DSIDES is integrated with MATLAB, utilizing its power for data analysis and visualization. Efficient data transfer and accurate plotting of ternary diagrams are enabled by the connection between DSIDES and MATLAB. The user experience is enhanced, and the advanced analytical features provided by MATLAB are leveraged through this integration.

Once the ternary plot is generated, the next step in the DSIDES workflow involves identifying satisficing solutions. However, it is important to note that manual intervention is required to obtain the satisficing space from the ternary plot. This manual process is thoroughly explained in Section II.5.1.4 of the software documentation, which provides detailed step-by-step instructions for users to follow.

In Section II.5.1.4, users are guided through a comprehensive procedure to identify and define the satisficing space within the ternary plot. This process involves visually analyzing the plotted data points and determining the desired range or region that represents satisfactory solutions based on the specific project requirements and constraints.

The step-by-step instructions provided in this section aim to ensure that users effectively navigate through the identification of the satisficing space. The manual intervention required in this stage allows users to exercise their expertise and judgment to determine which solutions within the ternary plot align with their desired criteria for success.

By carefully following the detailed instructions outlined in Section II.5.1.4, users can confidently identify the subset of solutions that meet their predefined thresholds and objectives. This manual approach empowers users to make informed decisions based on their domain knowledge and experience, further refining the solution space to align with their specific project goals.

Before proceeding with the verification and validation section and implementing all the information provided, it is important to evaluate the overall content of the output file. A thorough review of the file will help ensure its accuracy, professionalism, and relevance to the intended purpose.

II.3 Interpret the Output File

Before proceeding with the verification and validation section and implementing all the information provided, it is important to evaluate the overall content of the output file. This output file provides a comprehensive summary of the analysis, results, and key findings obtained through the software's various features and processes. By carefully examining the contents of the output file, users can gain a holistic understanding of the decision-making outcomes and validate the effectiveness of their design choices. The output file consolidates all relevant information, including imported data, calculated values, and other pertinent details, providing a comprehensive record of the user's decision-making journey. By evaluating the output file, users can assess the accuracy, consistency, and reliability of the software's calculations, ensuring that the obtained results align with their expectations and meet the desired project objectives. Additionally, the output file serves as a valuable resource for documentation purposes, allowing users to track and communicate the rationale behind their design decisions. Users could find the output file in the main folder that we have talked about in Section II.2 after running the linear and nonlinear sheet. The file would be in the format of notepad with the name of "DSIDES.out" and the type is Out file. The snapshot of the file has been attached here.

DSIDES.out 5/9/2023 10:10 PM OUT File

The output file begins by specifying the problem size limits, such as the maximum number of system variables, linear and nonlinear constraints and goals, accumulated constraints, goal priority levels, synthesis and analysis cycles, and other relevant parameters.

The subsequent sections of the output file provide detailed information about the problem being solved. This includes the title of the design problem and the system variable details, such as the number of real variables, discrete variables, and selection (Boolean) variables.

The file also presents information about linear constraints, nonlinear constraints, linear goals, and nonlinear goals. It provides data on the number of terms in each constraint or goal, their respective equations, and any associated deviation variables.

Additionally, the output file includes information on the stop criteria for synthesis cycles, values for stationarity of design variables, names of nonlinear constraints and goals, and other parameters related to optimization. (The additional details originate from the user-provided data in the Linear and Nonlinear sheets.)

Furthermore, the file mentions the output information that will be printed, such as system variables, deviation variables, deviation function, bound information, linear and nonlinear constraint information, and activity for linear and nonlinear solutions.

The output file also provides details about the current point's feasibility, design variables and their values, deviation variables and their values, deviation function value, and bounds on design variables.

It further includes information on linear constraints and goals, nonlinear constraints and goals, and the coefficients and RHS values associated with them.

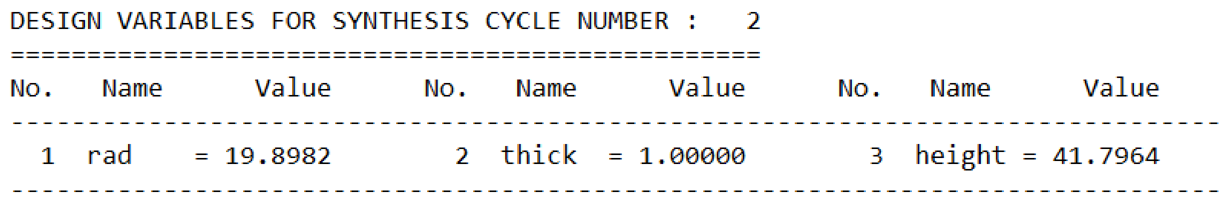
Also, the file lists any accumulated constraints, suppressed constraints, and additional notes or warnings that may be relevant to the decision base problem process.

Also, as mentioned in Section II.2.1, the Linear sheet user could select that wants to see the final result or result for each iteration. The output typically includes the following information in each iteration:

Design Variables:

The values of design variables for the current synthesis cycle number.

Example:

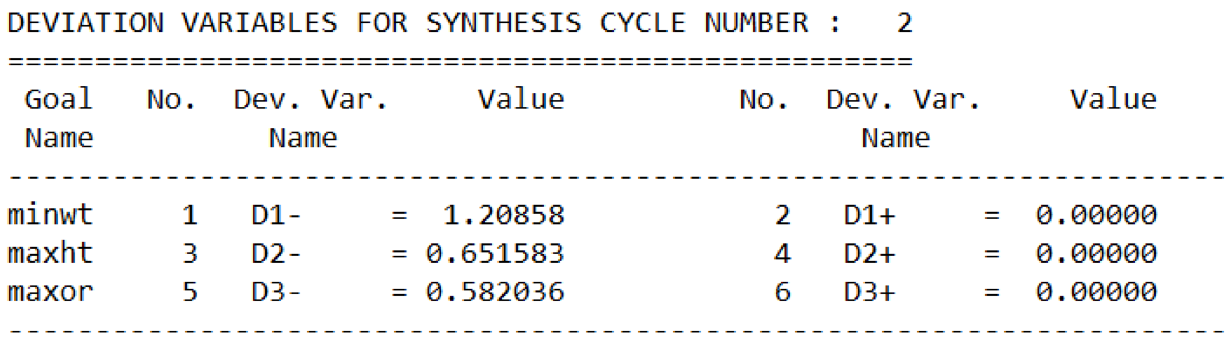


```
DESIGN VARIABLES FOR SYNTHESIS CYCLE NUMBER :   2
==================================================
No.   Name      Value      No.   Name      Value      No.   Name      Value
--------------------------------------------------------------------------------
  1  rad    = 19.8982        2  thick  = 1.00000        3  height = 41.7964
--------------------------------------------------------------------------------
```

Deviation Variables:

The values of deviation variables associated with the design. These variables quantify the deviations from the desired goals or constraints.

Example:

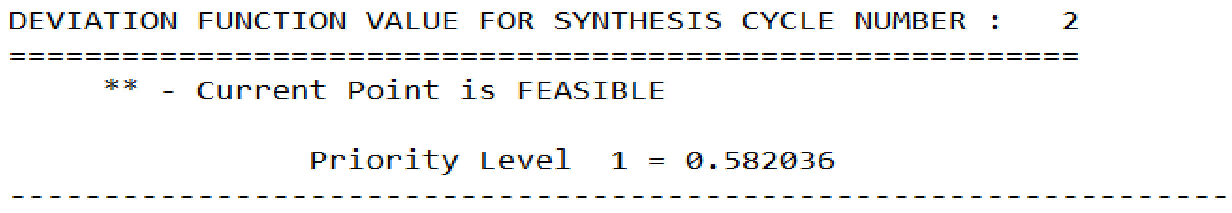


```
DEVIATION VARIABLES FOR SYNTHESIS CYCLE NUMBER :   2
====================================================
 Goal   No.  Dev. Var.     Value          No.  Dev. Var.     Value
 Name            Name                               Name
----------------------------------------------------------------------
minwt     1   D1-     =  1.20858            2   D1+     =  0.00000
maxht     3   D2-     = 0.651583            4   D2+     =  0.00000
maxor     5   D3-     = 0.582036            6   D3+     =  0.00000
----------------------------------------------------------------------
```

Deviation Function Value:

The value of the deviation function for the current point. This function represents the overall measure of how well the design satisfies the specified goals and constraints. A lower value indicates a better solution.

Example:



```
DEVIATION FUNCTION VALUE FOR SYNTHESIS CYCLE NUMBER :   2
=========================================================
     ** - Current Point is FEASIBLE

                    Priority Level  1 = 0.582036
------------------------------------------------------------------
```

Bounds on Design Variables:

The lower and upper bounds for each design variable. These bounds define the acceptable range of values for the variables during the process.

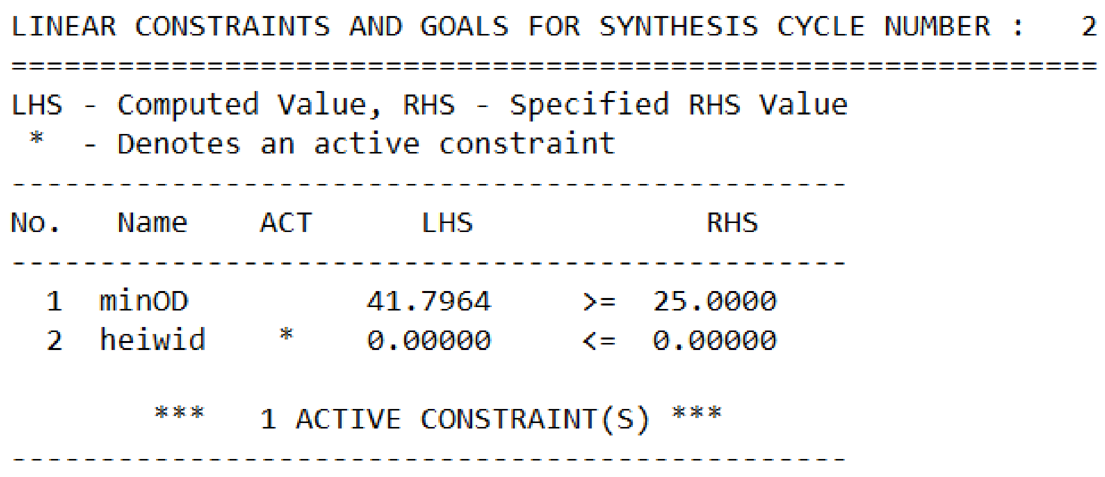
Example:

```
BOUNDS ON DESIGN VARIABLES FOR SYNTHESIS CYCLE NUMBER :    2
===========================================================
* - Denotes Active Lower or Upper Bounds
-------------------------------------------------------------------------
No.    Name  Active      Minimum          Value         Maximum   Active
-------------------------------------------------------------------------
  1   rad               5.00000     <=  19.8982     <=  45.0000
  2   thick       *     1.00000     <=  1.00000     <=  12.5000
  3   height            12.0000     <=  41.7964     <=  120.000
-------------------------------------------------------------------------
SUMMARY OF ACTIVE BOUNDS
------------------------

   1   Lower bound of variable thick  =  1.00000
-------------------------------------------------------------------------
```

Linear Constraints and Goals:

The linear constraints and goals for the current synthesis cycle. These constraints and goals represent the linear relationships among the design variables and are specified with corresponding coefficients and RHS (right-hand side) values.

Example:

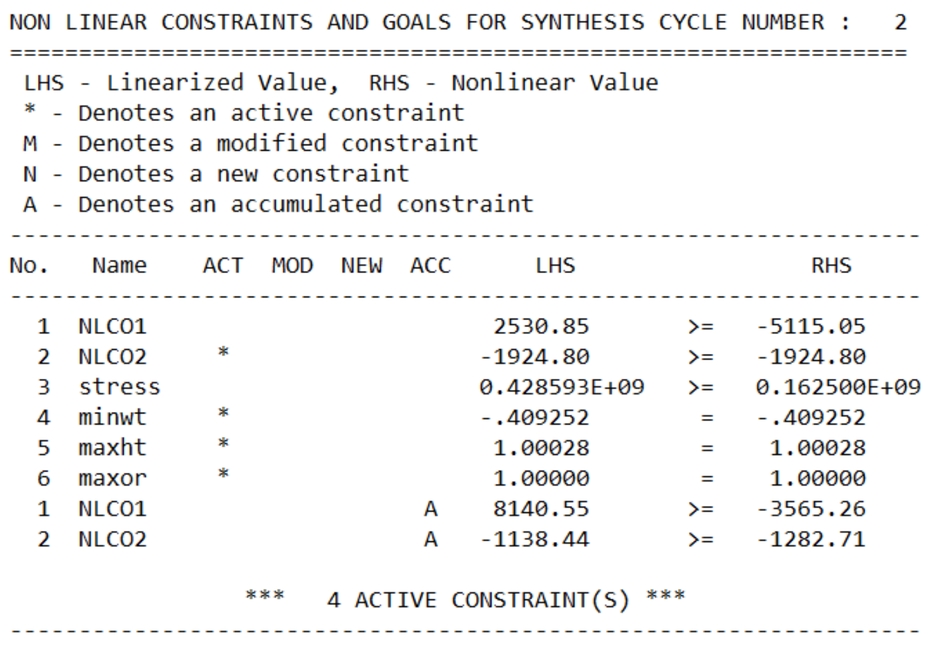


```
LINEAR CONSTRAINTS AND GOALS FOR SYNTHESIS CYCLE NUMBER :   2
==============================================================
LHS - Computed Value, RHS - Specified RHS Value
 *  - Denotes an active constraint
-----------------------------------------------
No.   Name    ACT      LHS             RHS
-----------------------------------------------
  1  minOD          41.7964     >=  25.0000
  2  heiwid    *    0.00000     <=  0.00000

        ***   1 ACTIVE CONSTRAINT(S) ***
-----------------------------------------------
```

Nonlinear Constraints and Goals:

The nonlinear constraints and goals for the current synthesis cycle. These constraints and goals involve nonlinear relationships among the design variables and are often represented with linearized values and nonlinear values.

Example:



```
NON LINEAR CONSTRAINTS AND GOALS FOR SYNTHESIS CYCLE NUMBER :   2
=================================================================
 LHS - Linearized Value,  RHS - Nonlinear Value
 * - Denotes an active constraint
 M - Denotes a modified constraint
 N - Denotes a new constraint
 A - Denotes an accumulated constraint
------------------------------------------------------------------
No.   Name    ACT  MOD  NEW  ACC      LHS                RHS
------------------------------------------------------------------
  1  NLCO1                           2530.85      >=   -5115.05
  2  NLCO2     *                    -1924.80      >=   -1924.80
  3  stress                         0.428593E+09  >=   0.162500E+09
  4  minwt     *                    -.409252       =   -.409252
  5  maxht     *                     1.00028       =    1.00028
  6  maxor     *                     1.00000       =    1.00000
  1  NLCO1                      A    8140.55      >=   -3565.26
  2  NLCO2                      A   -1138.44      >=   -1282.71

                ***   4 ACTIVE CONSTRAINT(S) ***
------------------------------------------------------------------
```

Note:

"Active constraints" refer to the constraints that are currently influencing the decision support process. These constraints are taken into account during the synthesis cycle and can impact the solution.

The term "active" is used to indicate that these constraints are affecting the current iteration and may be restricting the values or relationships of the design variables. Active constraints are typically those that are not satisfied or are close to being violated.

In the given output, the line "4 ACTIVE CONSTRAINT(S)" indicates that there are four constraints that are currently active in synthesis cycle number 2. These constraints are influencing the process and may need to be considered when evaluating the feasibility and quality of the design solution.

The specific constraints mentioned in the output, such as NLCO1, NLCO2, stress, minwt, maxht, and maxor, have corresponding LHS (linearized value) and RHS (nonlinear value) values. The LHS represents the computed or calculated value based on the current design variables, while the RHS represents the desired or specified value for that constraint.

The asterisk (*) next to NLCO2, minwt, maxht, and maxor indicates that these constraints are active, meaning they are currently affecting the decision support process. The "A" in the last two rows for NLCO1 and NLCO2 indicates that these constraints are accumulated constraints, which means they were active in previous synthesis cycles and are still influencing the current cycle.

Overall, the presence of active constraints in the output file indicates that the algorithm is considering and working to satisfy these constraints in order to find a feasible and satisficing design solution.

Dual Variables (Shadow Prices):

The section provides an overview of the dual variables or shadow prices associated with each constraint or goal in the decision support problem. These Lagrangian multipliers indicate the sensitivity and impact of the constraints and goals on the overall problem. They quantify the rate of change of the objective function in response to small perturbations in the corresponding constraints or goals. The number of constraint multipliers is categorized according to the solver's A matrix, distinguishing between linear and nonlinear goals, linear and nonlinear constraints, and various types of nonlinear constraints such as inequality, equality, accumulated, and new (adapted) constraints. These dual variables offer valuable insights into the direction, magnitude, and influence of each constraint or goal on the current solution, with a value of 0 indicating minimal influence.

```
DUAL VARIABLES (SHADOW PRICES) FOR SYNTHESIS CYCLE NUMBER :   2
===============================================================
   The Lagrangian Multiplier is equal to the Dual
   Variable that is assossiated with each constraint
   or goal.  Constraint multipliers are listed as
   arranged for the solvers A matrix:
     Linear goals                        - NLINGO =   0
     Nonlinear goals                     - NNLGOA =   3
     Linear constraints                  - NLINCO =   2
     Nonlinear inequality constraints    - NNLINQ =   3
     Nonlinear equality constraints      - NNLEQU =   0
     Nonlinear accumulated constraints   - NACCUM =   2
     Nonlinear new (adapted) constraints - NNUCON =   0
--------------------------------------------------------------------------

                  minwt           maxht           maxor           minOD
  CONLEV        0.00000         0.00000         0.00000        -1.00000
  PLEV1         0.00000         0.00000         0.00000         0.00000

                  heiwid          NLCO1           NLCO2           stress
  CONLEV        0.00000        -1.00000        -1.00000        -1.00000
  PLEV1        -.443355E-02     0.00000        0.246419E-03     0.00000

                NLCO1  :A01     NLCO2  :A02
  CONLEV       -1.00000        -1.00000
  PLEV1         0.00000         0.00000

--------------------------------------------------------------------------
```

Note:

For example, let us focus on the first constraint "minwt" in the table:

The "minwt" constraint has dual variable or shadow price values of 0.00000 for the design variables "CONLEV," "PLEV1," This indicates that the objective function is not significantly affected by small changes in these variables with respect to the "minwt" constraint.

However, the "minOD" constraint has a dual variable value of -1.00000 for the design variable "CONLEV." This suggests that a small increase or decrease in the "CONLEV" variable would have a significant impact on the objective function, resulting in changes in the shadow price of the "minOD" constraint. The same interpretation applies to other constraints and goals listed in the table.

The provided information corresponds to the final solution of a design problem obtained after the specified synthesis cycle. The convergence criteria have been achieved based on the stationarity of variables and deviation function.

Final Solution:

Bounds:

The section begins by presenting the bounds on the design variables for the given synthesis cycle. It includes the active lower and upper bounds for each variable, denoted by an asterisk (*). The variables are listed along with their respective minimum and maximum bounds.

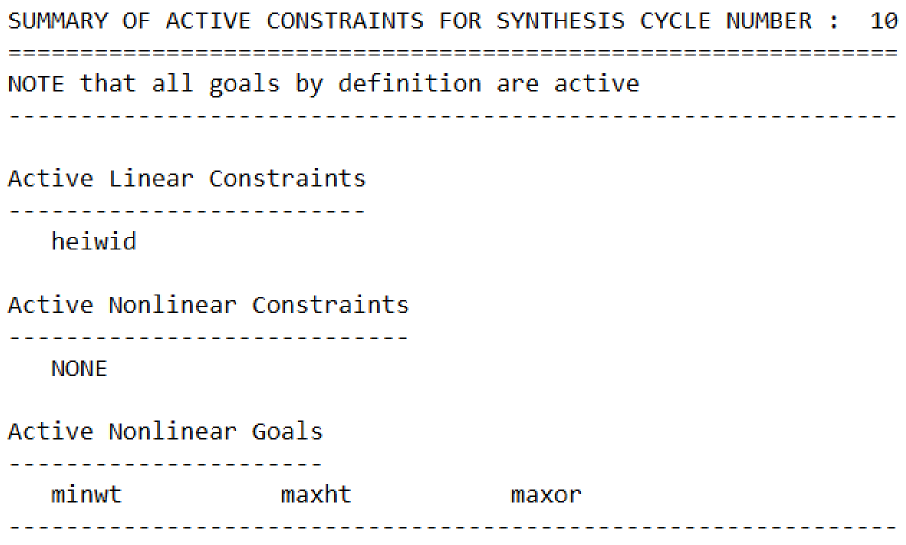
Example:

```
F I N A L   S O L U T I O N
----------------------------

output of: SYNTHESIS CYCLE: 10

CONVERGENCE ACHIEVED
(based on variable and deviation function stationarity)

BOUNDS ON DESIGN VARIABLES FOR SYNTHESIS CYCLE NUMBER :  10
===========================================================
* - Denotes Active Lower or Upper Bounds
-------------------------------------------------------------------------
No.    Name  Active      Minimum          Value          Maximum   Active
-------------------------------------------------------------------------
  1   rad               5.00000     <=  19.5593     <=  45.0000
  2   thick       *     1.00000     <=  1.00000     <=  12.5000
  3   height            12.0000     <=  41.1186     <=  120.000
-------------------------------------------------------------------------
SUMMARY OF ACTIVE BOUNDS
------------------------

   1   Lower bound of variable thick  =  1.00000
-------------------------------------------------------------------------
```

Active constraints:

A summary of the active constraints follows, indicating the constraints that are currently active. This section distinguishes between active linear constraints, active nonlinear constraints, and active nonlinear goals. It provides insights into which constraints or goals are affecting the current solution.

Example:

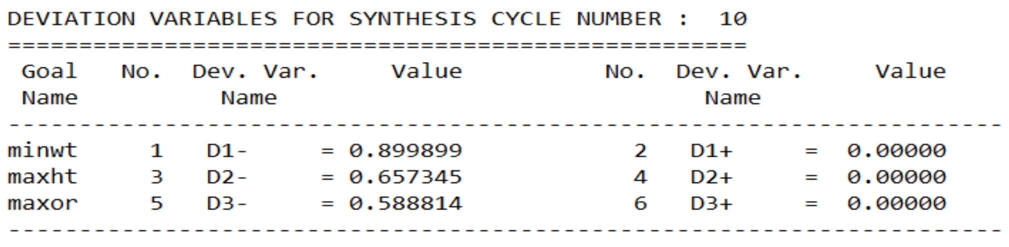


```
SUMMARY OF ACTIVE CONSTRAINTS FOR SYNTHESIS CYCLE NUMBER :   10
==============================================================
NOTE that all goals by definition are active
--------------------------------------------------------------

Active Linear Constraints
-------------------------
   heiwid

Active Nonlinear Constraints
----------------------------
   NONE

Active Nonlinear Goals
----------------------
   minwt           maxht           maxor
--------------------------------------------------------------
```

Deviation variables:

Next, the deviation variables for the synthesis cycle are provided. These variables represent the deviation from the desired values for each goal or objective. The deviation variables are labeled and associated with specific goals or objectives, and their corresponding values are given.

Example:



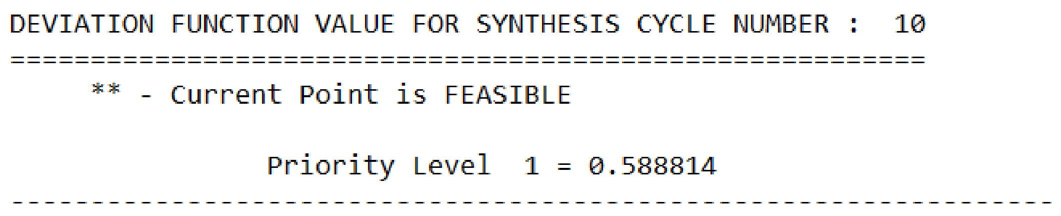
```
DEVIATION VARIABLES FOR SYNTHESIS CYCLE NUMBER :   10
====================================================
 Goal    No.  Dev. Var.     Value          No.  Dev. Var.     Value
 Name           Name                              Name
----------------------------------------------------------------------
minwt      1    D1-      = 0.899899          2    D1+      =  0.00000
maxht      3    D2-      = 0.657345          4    D2+      =  0.00000
maxor      5    D3-      = 0.588814          6    D3+      =  0.00000
----------------------------------------------------------------------
```

Deviation function:

The deviation function value for the synthesis cycle is presented, indicating the level of deviation or objective function value.

This value provides an indication of the solution's performance with respect to the desired goals or objectives.

Example:



```
DEVIATION FUNCTION VALUE FOR SYNTHESIS CYCLE NUMBER :   10
============================================================
     ** - Current Point is FEASIBLE

                Priority Level  1 = 0.588814
--------------------------------------------------------------------
```

Finally, the section concludes with any additional relevant information, such as the total time required for the problem and the completion details of the job, including the date and time. This information helps assess the computational effort and provides a sense of completion for the problem-solving process.

II.4 Verification and Validation

The developed wrapper is a powerful tool for facilitating various tasks, and it is crucial to ensure its accuracy and reliability. A comprehensive verification process has been undertaken to confirm its performance as intended and ensure that the specified requirements have been met. The wrapper's functionality and performance have been evaluated against predefined benchmarks and user expectations through thorough testing and analysis. The verification process is carried out to validate the implemented features' correctness and ensure that all input and output interactions are functioning as expected. Our goal of this verification step is to ensure that users' needs are effectively addressed by the wrapper, productivity is enhanced, and meaningful solutions are provided for the intended problems. Through successful verification and validation, reliability is established by the wrapper, and it could be used as a reliable platform for users to efficiently and effectively accomplish their tasks. A robust approach has been adopted to verify and validate the system, ensuring its accuracy, reliability, and user-friendliness. Three examples have been presented in preemptive and Archimedean forms, each consisting of a problem statement, cDSP formulation, and wrapper implementation. To ensure thorough evaluation, multiple individuals execute these examples, rigorously assessing the system's performance in terms of accuracy, reliability, and user-friendliness. There are two important purposes for this verification and validation. Firstly, for validation of the functionality and effectiveness of the wrapper, confirming its ability to meet the intended objectives and providing reliable solutions. Secondly, it is a valuable learning opportunity for users to practice and gain proficiency in thesis Parts 1 and 2, enhancing their understanding of the system and its capabilities. The Elephant Stand Problem example, which was used in Section II.2 to demonstrate how different sheets would function and how input should be prepared for DSIDES, is presented in the next section.

II.4.1 Elephant Stand Problem in Archimedean Form

The provided example was utilized to illustrate the wrapper section within the introduction of various sheets. However, given its relevance to the post-solution analysis, it becomes necessary to reiterate it in this context.

II.4.1.1 Problem statement

In the late 1800s, Ringling Bros and Barnum and Baily Circus were looking to establish dimensions of a new pedestal for their circus elephant Jumbo. They would play a trick that involved a support pedestal where Jumbo would perform a one-legged hand stand. The cost of manufacturing must be minimized, which depends on its thickness, width, and the amount of material it would consume. And it must be as tall as possible for a wow factor. And finally, the pedestal must be wide enough to ensure Jumbo has enough room to safely stand on one foot. This means *the goals of the design are to minimize the manufacturing cost, maximize the height, and maximize the outer radius.* A material of 2014 Aluminum with a modulus of 10600 ksi and a density of 0.1 lb/in^3 has been selected.

Jumbo's foot is approximately 25" in diameter, so the pedestal must also be greater than 25". Jumbo weighs 13,560lb and stands 13.5ft tall. Use a factor of safety of 1.5.

Given a certain type of material, design a cylinder (the “elephant stand”). The cylinder has two parts joined together. The upper half is a tube, and the designer’s interest is to determine its thickness, radius, and height that best satisfies the goals identified. The lower half is a 4-inch-height solid base. The goals identified by the designer include Minimizing the manufacturing cost, maximizing the height, and maximizing the outer diameter. Requirements include the upper and lower limits of the parameters that the cylinder can physically reach. The overview of the elephant stand problem is represented in figure II.6.

II.4.1.2 cDSP Formulation

Word Formulation:

Given

System parameters

Material – 2014 Aluminum

Elastic modulus for the material

Safety factor

Yield stress for the material

Density of the material

Load (elephant’s weight)

Moment of inertia for the cylindrical section

Maximum normal stress

Maximum buckling stress for a fixed free column

Cost target

Height target

OR target

Find

System variables

Radius, Thickness, Height

Satisfy

System constraints

Reaching the minimum outer diameter.

Not exceeding a certain height-to-width ratio.

Reaching the stress requirement.

Not exceeding the maximum weight.

Not exceeding the maximum load in the stand.

System goals

Goal 1: reaching minimum cost target.

Goal 2: reaching maximum height target.

Goal 3: reaching maximum outer radius target.

Bounds

The upper and lower limit of Radius, Thickness, and Height

Minimize

The deviation function.

cDSP Math Formulation:

Given

E = 10600000.

OR=R+T

SF = 1.5

SIGY = 11000.

P = 12000.

PI = 2*ACOS(0.0)

RHO = 0.1

$$I = \frac{\pi}{4} * [(R + T)^4 - R^4]$$

$$W1 = \pi * [(R + T)^2 - R^2] * (H - 4)$$

$$W2 = \pi * [(R + T)^2] * 4$$

W=(W1+W2)∗RHO

STR=P/(π∗[(R+T)^2−R^2])

PCR=(π^2∗E∗I)/ 4∗(H^2)

COST=e^(2.5/T)∗e^(3/R)∗W

Cost target = 5000

Height target = 180

OR target = 15

Find

System variables

- R Radius

- T Thickness

- H Height

Deviation variables

- d_i^+ over achievement of Goal i, where i=1,2,3

- d_i^- under-achievement of Goal i, where i=1,2,3

Satisfy

System constraints

- minOD: minimum outer diameter

$2 * R + 2 * T \geq 6$ (CO1)

- heiwid: height to width ratio

$R + T - 0.03 * H \geq 0$ (CO2)

- stress : minimum stress in stand

$$\frac{\mathrm{SIGY}}{\mathrm{SF}} - \mathrm{STR} \geq 0 \qquad \text{(CO3)}$$

- weight: maximum weight

$$1000 - \mathrm{W} \geq 0 \qquad \text{(CO4)}$$

- buckle: maximum load in stand

$$\frac{\mathrm{PCR}}{\mathrm{F}} - \mathrm{P} \geq 0 \qquad \text{(CO5)}$$

System goals

- Goal 1: minimum cost

$$\frac{\text{cost target}}{\text{cost}} + d_1^- - d_1^+ = 1 \qquad \text{(G1)}$$

- Goal 2: maximum height

$$\frac{\text{height}}{\text{height target}} + d_2^- - d_2^+ = 1 \qquad \text{(G2)}$$

- Goal 3: maximum outer radius

$$\frac{\text{OR}}{\text{OR target}} + d_3^- - d_3^+ = 0 \qquad \text{(G3)}$$

Bounds

$3 \leq R \leq 45$

$0.5 \leq T \leq 2.5$

$100 \leq H \leq 120$

Minimize

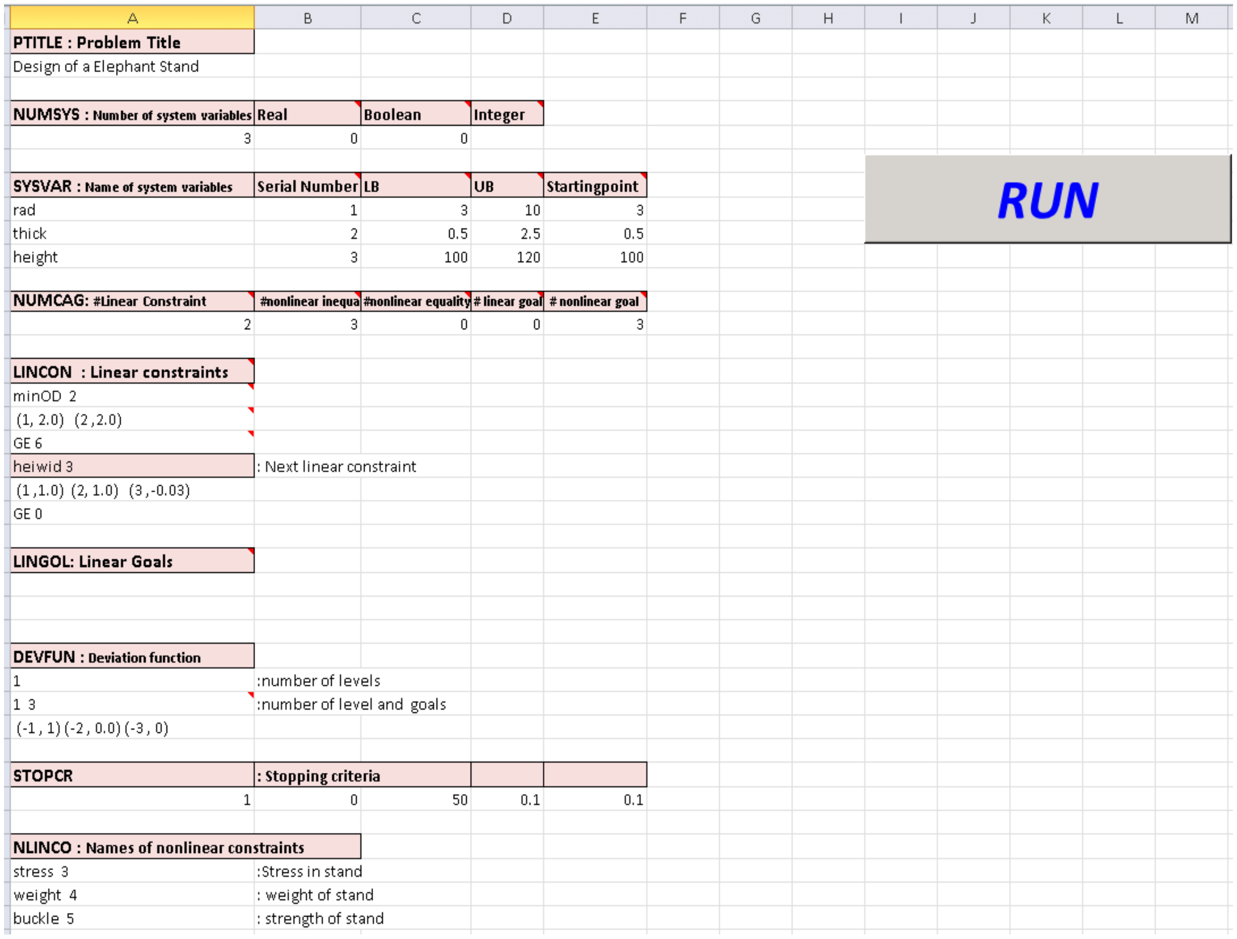
The deviation function.

$$Z = \sum_{i=1}^{3} w_i \cdot (di^- + di^+), \sum_{i=1}^{3} w_i = 1$$

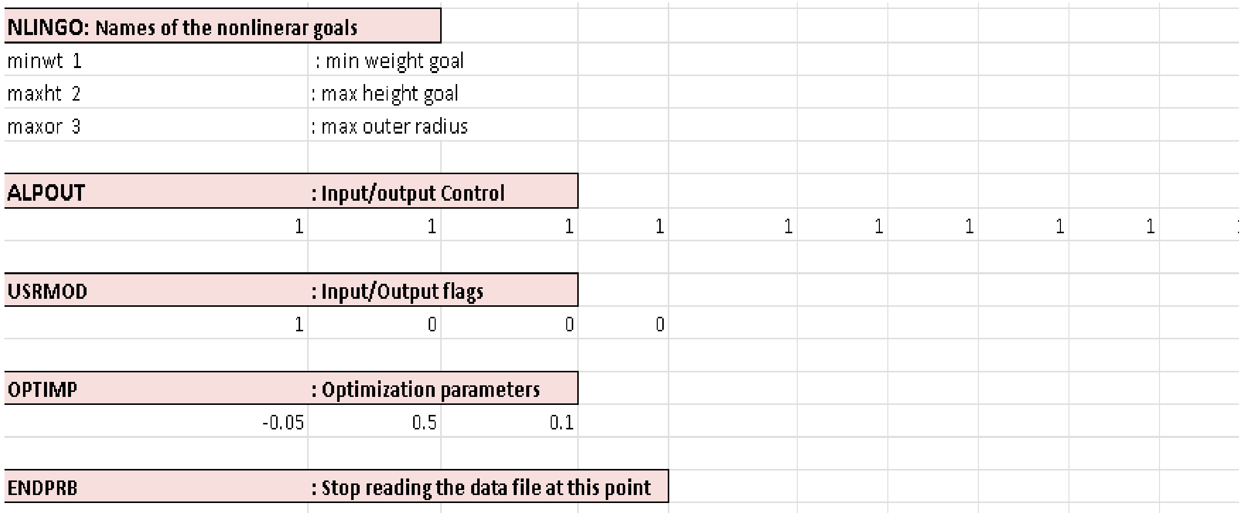
II.4.1.3 Implementation in DSIDES wrapper

1) Linear sheet

Once the mathematical cDSP formulation is ready, the next step is to finalize the Linear sheet. In this particular problem, which is Archimedean and consists of a single level, we will choose the LinearArchimedean sheet to complete the process. Once the LinearArchimedean sheet is filled with the necessary information, we proceed to press the "Run" button. Executing this command will generate a file named "DSIDES" in the form of a ".dat" file. This file will be automatically saved in the main folder of our project. The overview of the Linear sheet for the Elephant stand problem is represented in Figure II.8



A	B	C	D	E
PTITLE : Problem Title				
Design of a Elephant Stand				
NUMSYS : Number of system variables	**Real**	**Boolean**	**Integer**	
3	0	0		
SYSVAR : Name of system variables	**Serial Number**	**LB**	**UB**	**Startingpoint**
rad	1	3	10	3
thick	2	0.5	2.5	0.5
height	3	100	120	100
NUMCAG: #Linear Constraint	**#nonlinear inequa**	**#nonlinear equality**	**# linear goal**	**# nonlinear goal**
2	3	0	0	3
LINCON : Linear constraints				
minOD 2				
(1, 2.0) (2 ,2.0)				
GE 6				
heiwid 3	: Next linear constraint			
(1 ,1.0) (2, 1.0) (3 ,-0.03)				
GE 0				
LINGOL: Linear Goals				
DEVFUN : Deviation function				
1	:number of levels			
1 3	:number of level and goals			
(-1 , 1) (-2 , 0.0) (-3 , 0)				
STOPCR	**: Stopping criteria**			
1	0	50	0.1	0.1
NLINCO : Names of nonlinear constraints				
stress 3	:Stress in stand			
weight 4	: weight of stand			
buckle 5	: strength of stand			

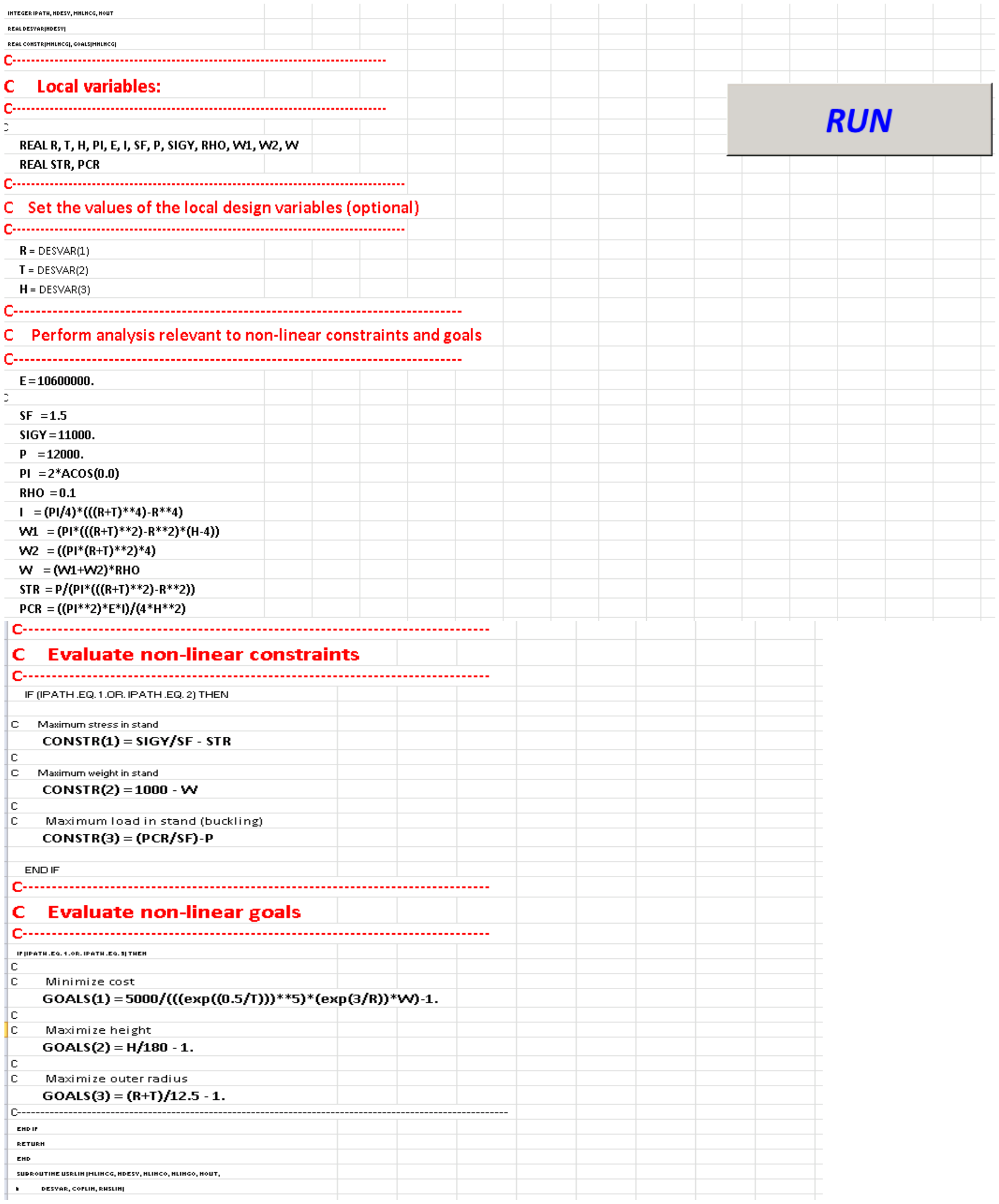


NLINGO: Names of the nonlinerar goals								
minwt 1	: min weight goal							
maxht 2	: max height goal							
maxor 3	: max outer radius							
ALPOUT	**: Input/output Control**							
1	1	1	1	1	1	1	1	1
USRMOD	**: Input/Output flags**							
1	0	0	0					
OPTIMP	**: Optimization parameters**							
-0.05	0.5	0.1						
ENDPRB	**: Stop reading the data file at this point**							

Figure II. 8: Overview of Linear sheet in Elephant stand problem.

2) NonLinear sheet

After this step, we need to fill the Nonlinear sheet form. Once we have completed the previous step, the next task is to complete the Nonlinear sheet form. After filling out the required information in the form, we need to proceed by pressing the "run" button. This action will initiate a process that generates several files, such as "DSIDES.f" and DSIDES.out. The "DSIDES.f" file is in the Fortran format and serves a specific purpose within the system. On the other hand, the DSIDES.out contains all the output information that we require. Upon pressing the "run" button again, the system will display the " DSIDES.out " on the screen, making it easily accessible for further analysis or utilization. The overview of the NonLinear sheet for the Elephant stand problem is represented in Figure II.9.

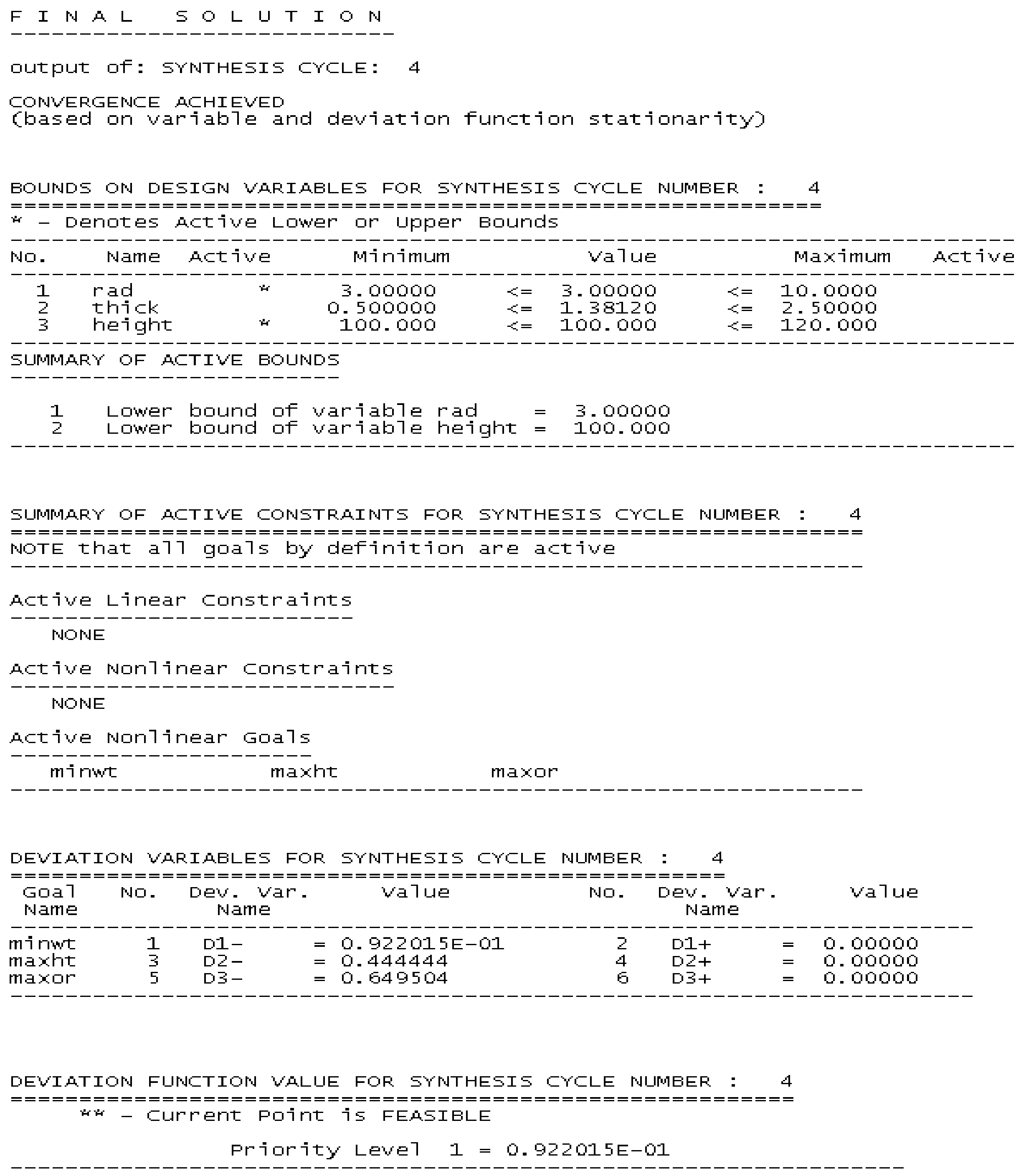


```
INTEGER IPATH, NDESV, MNLNCG, NOUT
REAL DESVAR(NDESV)
REAL CONSTR(MNLNCG), GOALS(MNLNCG)
C--------------------------------------------------------------------------
C    Local variables:
C--------------------------------------------------------------------------
C
  REAL R, T, H, PI, E, I, SF, P, SIGY, RHO, W1, W2, W
  REAL STR, PCR
C-----------------------------------------------------------------------------
C   Set the values of the local design variables (optional)
C-----------------------------------------------------------------------------
  R = DESVAR(1)
  T = DESVAR(2)
  H = DESVAR(3)
C------------------------------------------------------------------------------
C   Perform analysis relevant to non-linear constraints and goals
C------------------------------------------------------------------------------
  E=10600000.
C
  SF  =1.5
  SIGY=11000.
  P   =12000.
  PI  =2*ACOS(0.0)
  RHO =0.1
  I   =(PI/4)*(((R+T)**4)-R**4)
  W1  =(PI*(((R+T)**2)-R**2)*(H-4))
  W2  =((PI*(R+T)**2)*4)
  W   =(W1+W2)*RHO
  STR =P/(PI*(((R+T)**2)-R**2))
  PCR =((PI**2)*E*I)/(4*H**2)
C--------------------------------------------------------------------------
C    Evaluate non-linear constraints
C--------------------------------------------------------------------------
   IF (IPATH .EQ. 1 .OR. IPATH .EQ. 2) THEN

C      Maximum stress in stand
     CONSTR(1) = SIGY/SF - STR
C
C      Maximum weight in stand
     CONSTR(2) = 1000 - W
C
C       Maximum load in stand (buckling)
     CONSTR(3) = (PCR/SF)-P

   END IF
C--------------------------------------------------------------------------
C    Evaluate non-linear goals
C--------------------------------------------------------------------------
 IF (IPATH .EQ. 1 .OR. IPATH .EQ. 3) THEN
C
C       Minimize cost
     GOALS(1) = 5000/(((exp((0.5/T)))**5)*(exp(3/R))*W)-1.
C
C       Maximize height
     GOALS(2) = H/180 - 1.
C
C       Maximize outer radius
     GOALS(3) = (R+T)/12.5 - 1.
C-----------------------------------------------------------------------------------------------
 END IF
 RETURN
 END
 SUBROUTINE USRLIN (MLINCG, NDESV, NLINCO, NLINGO, NOUT,
 &       DESVAR, COFLIN, RHSLIN)
```

Figure II. 9: Overview of NonLinear sheet in Elephant stand problem.

3) Output file

The final result of these formulations has been represented in Figure II.10



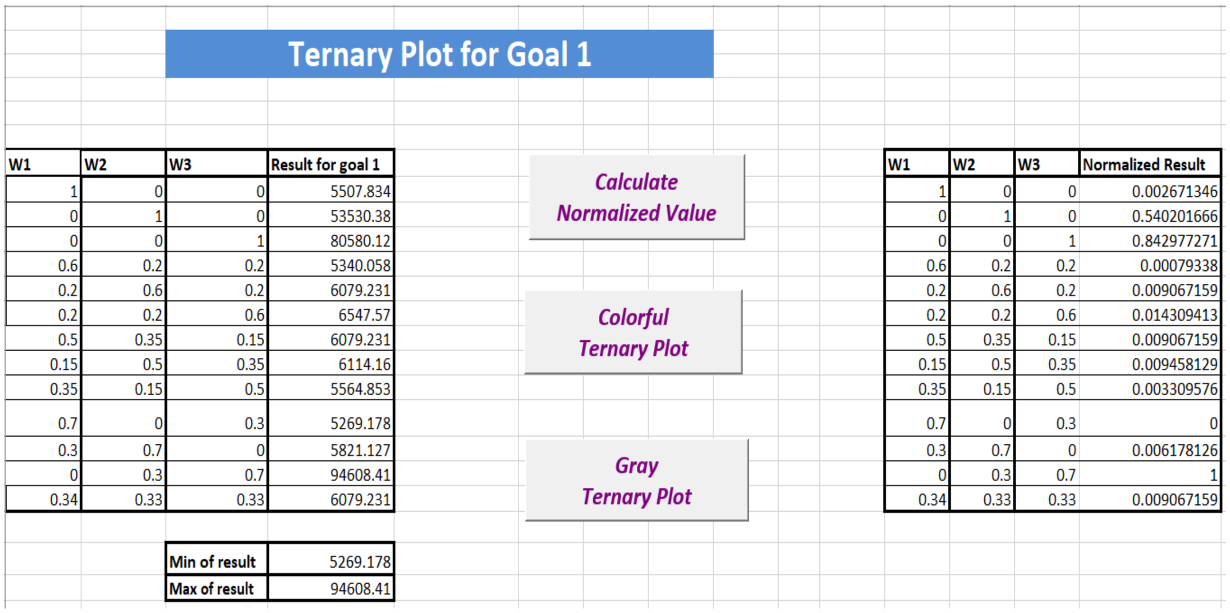
```
F I N A L   S O L U T I O N
----------------------------

output of: SYNTHESIS CYCLE:  4

CONVERGENCE ACHIEVED
(based on variable and deviation function stationarity)


BOUNDS ON DESIGN VARIABLES FOR SYNTHESIS CYCLE NUMBER :   4
===========================================================
* - Denotes Active Lower or Upper Bounds
-------------------------------------------------------------------------
No.    Name  Active     Minimum          Value           Maximum   Active
-------------------------------------------------------------------------
  1   rad         *     3.00000     <=  3.00000     <=  10.0000
  2   thick            0.500000     <=  1.38120     <=  2.50000
  3   height      *     100.000     <=  100.000     <=  120.000
-------------------------------------------------------------------------
SUMMARY OF ACTIVE BOUNDS
------------------------

   1   Lower bound of variable rad    =  3.00000
   2   Lower bound of variable height =  100.000
-------------------------------------------------------------------------


SUMMARY OF ACTIVE CONSTRAINTS FOR SYNTHESIS CYCLE NUMBER :   4
==============================================================
NOTE that all goals by definition are active
--------------------------------------------------------------

Active Linear Constraints
-------------------------
   NONE

Active Nonlinear Constraints
----------------------------
   NONE

Active Nonlinear Goals
----------------------
   minwt            maxht            maxor
--------------------------------------------------------------


DEVIATION VARIABLES FOR SYNTHESIS CYCLE NUMBER :   4
====================================================
 Goal   No.  Dev. Var.     Value          No.  Dev. Var.     Value
 Name          Name                              Name
----------------------------------------------------------------------
minwt     1   D1-      = 0.922015E-01      2    D1+      =  0.00000
maxht     3   D2-      = 0.444444          4    D2+      =  0.00000
maxor     5   D3-      = 0.649504          6    D3+      =  0.00000
----------------------------------------------------------------------



DEVIATION FUNCTION VALUE FOR SYNTHESIS CYCLE NUMBER :   4
=========================================================
     ** - Current Point is FEASIBLE

                Priority Level  1 = 0.922015E-01
-----------------------------------------------------------------
```

Figure II. 10: Overview of final result in Elephant stand problem

4) Ternary Plot

After performing the linear and nonlinear analysis, an output file is generated. The comprehensive description of this file can be found in Section II.3, titled "Interpretation of the Output File." In this section, users are instructed to carefully examine the output file, taking into account the values of variables, deviation variables, and the weights assigned based on the designer's preferences. By utilizing these factors, users can calculate the results for each goal.

Three separate sheets are provided to aid the analysis, each presenting a ternary plot for an individual goal. These ternary plots are elaborated upon in Section II.2.4, titled "Ternary Plot." They visually represent the results of each goal and facilitate a clearer understanding of the data. Furthermore, the overview of each sheet has been represented in Figure II.11, II.13, and II.15 that illustrating the calculated results in a graphical format. The ternary plot for goals 1, 2, and 3 is represented in Figures II.12 , II.14, and II.16, respectively.



W1	W2	W3	Result for goal 1
1	0	0	5507.834
0	1	0	53530.38
0	0	1	80580.12
0.6	0.2	0.2	5340.058
0.2	0.6	0.2	6079.231
0.2	0.2	0.6	6547.57
0.5	0.35	0.15	6079.231
0.15	0.5	0.35	6114.16
0.35	0.15	0.5	5564.853
0.7	0	0.3	5269.178
0.3	0.7	0	5821.127
0	0.3	0.7	94608.41
0.34	0.33	0.33	6079.231

Min of result	5269.178
Max of result	94608.41

W1	W2	W3	Normalized Result
1	0	0	0.002671346
0	1	0	0.540201666
0	0	1	0.842977271
0.6	0.2	0.2	0.00079338
0.2	0.6	0.2	0.009067159
0.2	0.2	0.6	0.014309413
0.5	0.35	0.15	0.009067159
0.15	0.5	0.35	0.009458129
0.35	0.15	0.5	0.003309576
0.7	0	0.3	0
0.3	0.7	0	0.006178126
0	0.3	0.7	1
0.34	0.33	0.33	0.009067159

Figure II. 11: Overview of TernaryPlot1 sheet in Elephant Stand Problem

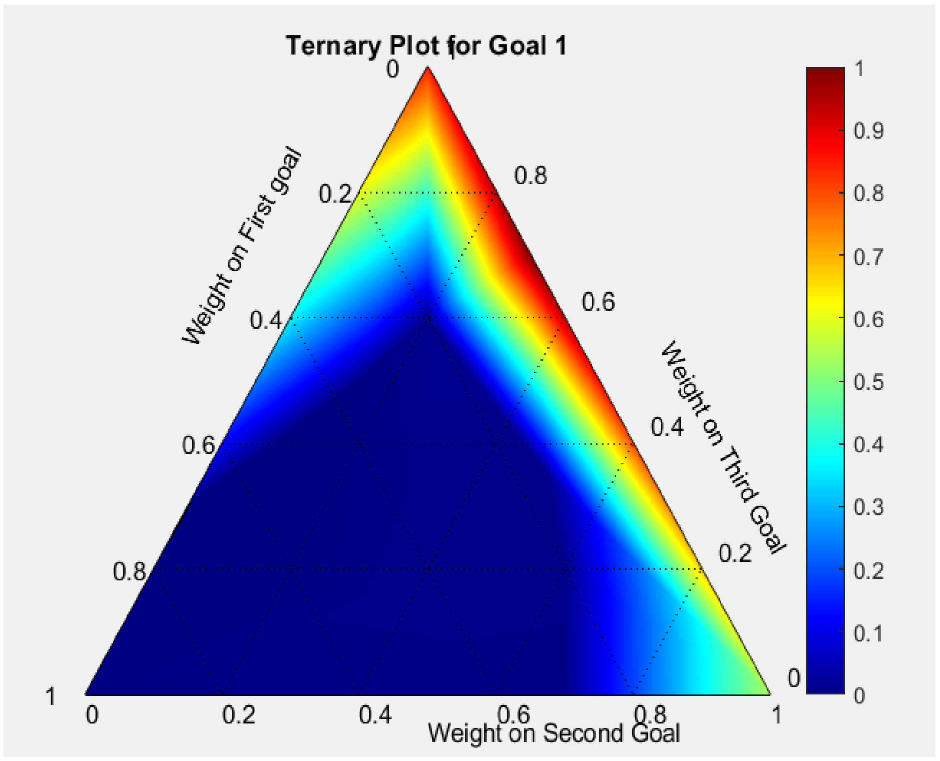
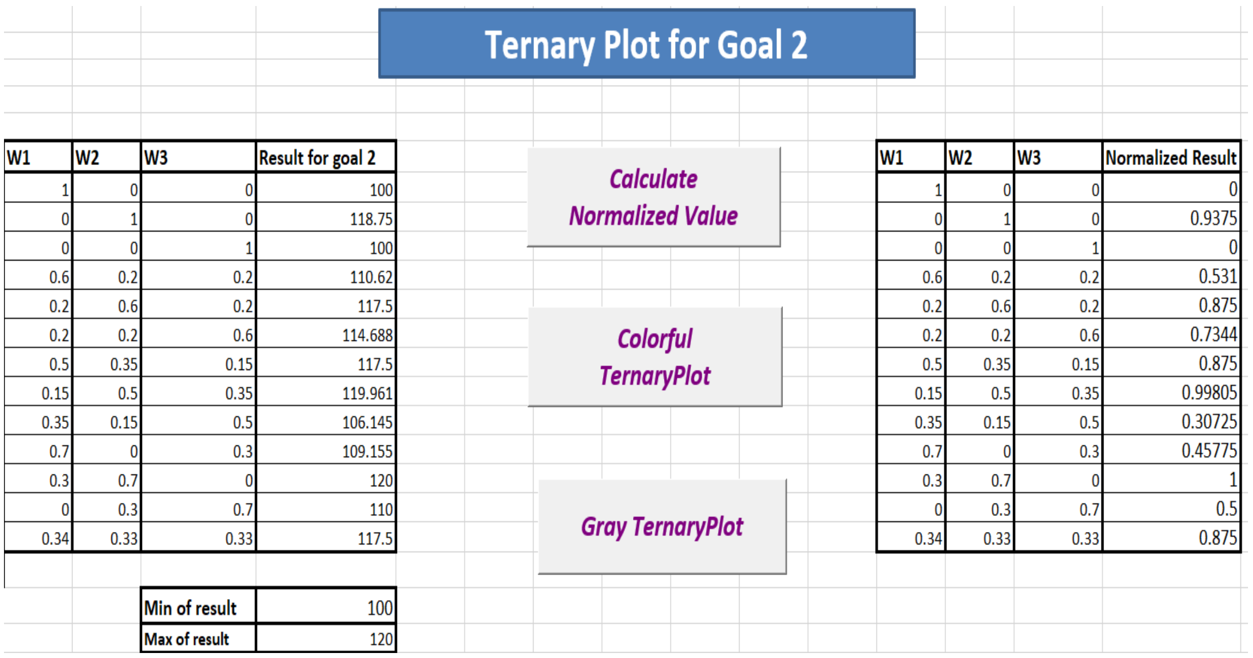


Figure II. 12: Ternary plot for goal 1 in Elephant Stand Problem


Ternary Plot for Goal 2

W1	W2	W3	Result for goal 2
1	0	0	100
0	1	0	118.75
0	0	1	100
0.6	0.2	0.2	110.62
0.2	0.6	0.2	117.5
0.2	0.2	0.6	114.688
0.5	0.35	0.15	117.5
0.15	0.5	0.35	119.961
0.35	0.15	0.5	106.145
0.7	0	0.3	109.155
0.3	0.7	0	120
0	0.3	0.7	110
0.34	0.33	0.33	117.5

Min of result	100
Max of result	120

W1	W2	W3	Normalized Result
1	0	0	0
0	1	0	0.9375
0	0	1	0
0.6	0.2	0.2	0.531
0.2	0.6	0.2	0.875
0.2	0.2	0.6	0.7344
0.5	0.35	0.15	0.875
0.15	0.5	0.35	0.99805
0.35	0.15	0.5	0.30725
0.7	0	0.3	0.45775
0.3	0.7	0	1
0	0.3	0.7	0.5
0.34	0.33	0.33	0.875

Figure II. 13: Overview of TernaryPlot2 sheet in Elephant Stand Problem

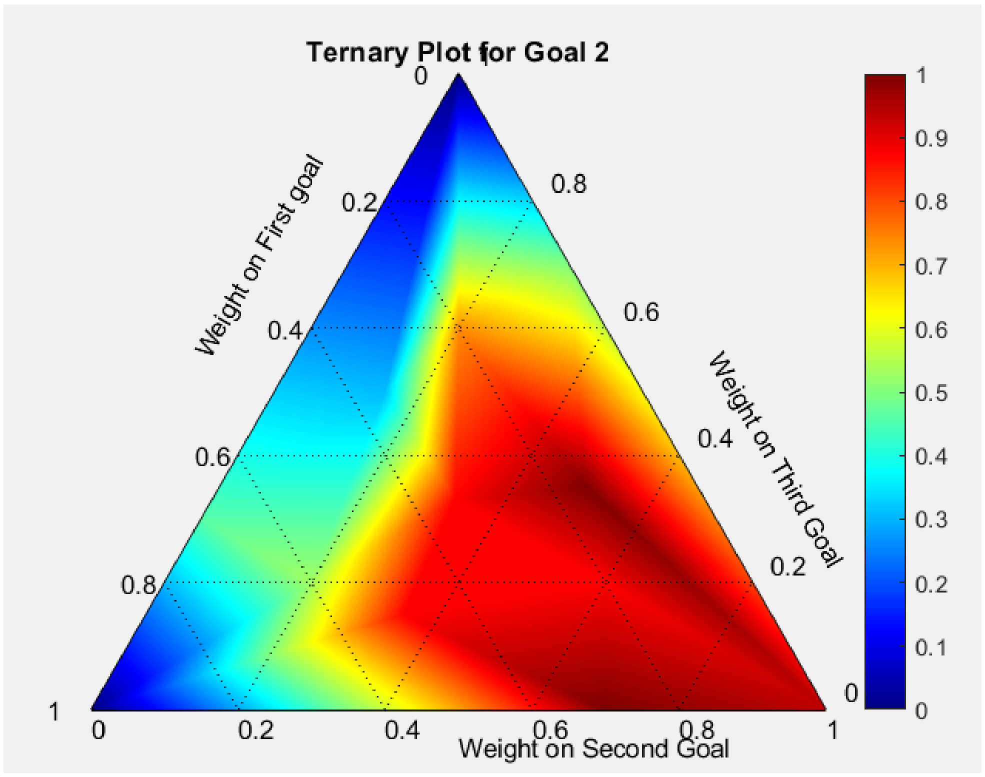
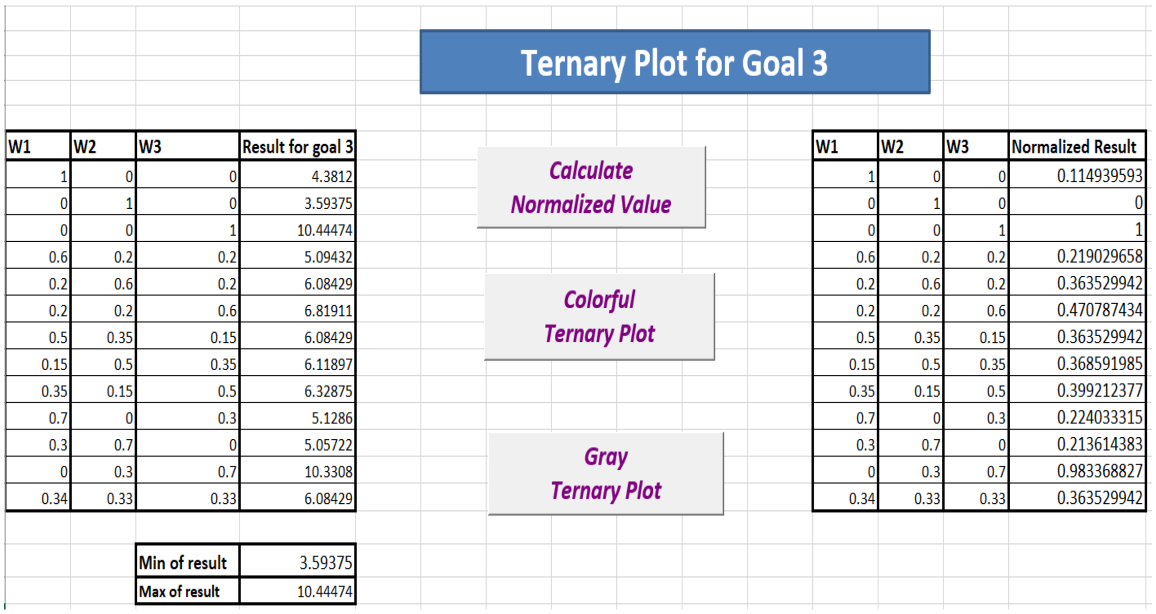


Figure II. 14: Ternary plot for goal 2 in Elephant Stand Problem



Ternary Plot for Goal 3

W1	W2	W3	Result for goal 3
1	0	0	4.3812
0	1	0	3.59375
0	0	1	10.44474
0.6	0.2	0.2	5.09432
0.2	0.6	0.2	6.08429
0.2	0.2	0.6	6.81911
0.5	0.35	0.15	6.08429
0.15	0.5	0.35	6.11897
0.35	0.15	0.5	6.32875
0.7	0	0.3	5.1286
0.3	0.7	0	5.05722
0	0.3	0.7	10.3308
0.34	0.33	0.33	6.08429

Min of result	3.59375
Max of result	10.44474

W1	W2	W3	Normalized Result
1	0	0	0.114939593
0	1	0	0
0	0	1	1
0.6	0.2	0.2	0.219029658
0.2	0.6	0.2	0.363529942
0.2	0.2	0.6	0.470787434
0.5	0.35	0.15	0.363529942
0.15	0.5	0.35	0.368591985
0.35	0.15	0.5	0.399212377
0.7	0	0.3	0.224033315
0.3	0.7	0	0.213614383
0	0.3	0.7	0.983368827
0.34	0.33	0.33	0.363529942

Figure II. 15: Overview of TernaryPlot3 sheet in Elephant Stand Problem

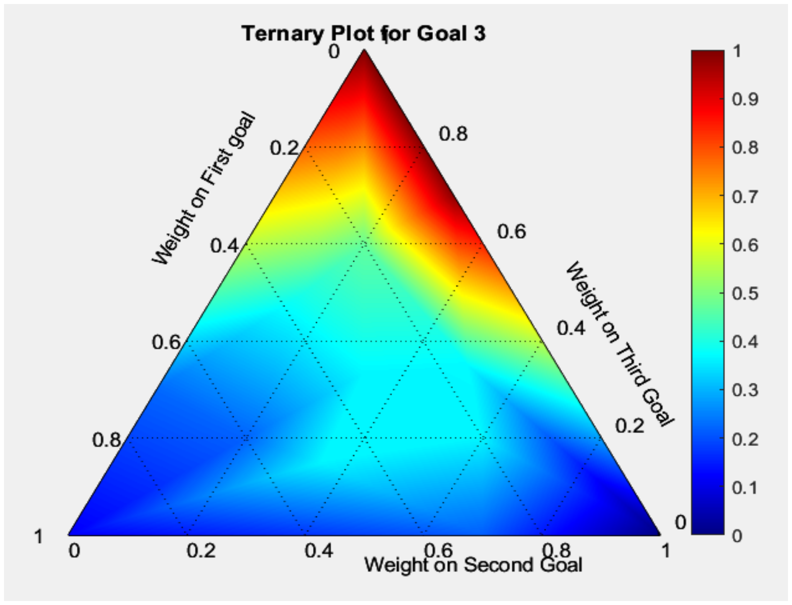


Figure II. 16: Ternary plot for goal 3 in Elephant Stand Problem

II.4.1.4 Find Satisficing Space in Elephant Stand Problem Based on the Ternary Plots

To obtain the satisficing space from the ternary plot, manual intervention is required. The following steps outline the process in a detailed and understandable manner:

- **Identify Acceptable Values**: Drawing upon their experience and knowledge, the designer should determine the acceptable value range for each goal. These values have represented the desired outcomes for the design problem. While there might be some variations among different designers' perspectives, the focus here is on determining a range of values that represent the desired outcomes rather than seeking a single optimal solution. Given the nature of the satisficing space analysis, where the goal is to find feasible combinations of variables that meet or exceed the acceptable values for all goals, a broader range of acceptable values can be considered. Flexibility and acknowledgment that multiple solutions within the acceptable range can still be considered satisfactory are achieved by this approach.

- **Normalize Acceptable Values:** Normalize the acceptable values to bring them to a consistent scale. This step ensures that all goals are treated equally during the analysis.
- **Draw Contour Lines:** Plot the contour lines of the normalized acceptable values on each ternary plot. The boundaries of the acceptable values for each goal are represented with these contour lines. Connect the points with equal normalized values to form the contour lines.
- **Identify Feasible Space:** The region on each ternary plot that reaches or exceeds the acceptable value of a particular goal is considered the feasible space for that goal. Combinations of variables that satisfy or surpass the designer's expectations for that specific goal are represented by this region.
- **Determine Satisficing Space:** Superimpose the feasible spaces of all three goals on the ternary plot. The overlapping region, where the feasible spaces intersect, is the satisficing space of the design problem. Combinations of variables that simultaneously satisfy or exceed the acceptable values for all three goals are represented by overlapping region.
- **Iterate if Necessary:** If there is no superimposed region initially, we could realize that the acceptable values set in the first step may need adjustment. Revisit the first step and redefine the acceptable values until a superimposed region is formed in the ternary plot. With this iterative process, the designer ensures that the design goals are adequately balanced and achievable.

Following the outlined steps, designers can manually analyze the ternary plot to identify the satisficing space, representing the feasible and desirable design solutions. However, it is important to acknowledge that there might be situations where no satisficing space is found initially.

Suppose no superimposed region is observed in the ternary plot, indicating that there is no overlap between the feasible spaces of the individual goals. In that case, the initially defined acceptable values for the goals may need adjustment.

The described steps for identifying the satisficing space were implemented for the elephant stand problem. The ternary plots corresponding to each goal were created, and the results of the analysis were visually presented in Figures II.17, II.18, and II.19.

The ternary plot illustrating the feasible space for the first goal has been presented in Figure II.17, while the ternary plot for the feasible space for the second goal has been represented in Figure II.18. Similarly, the ternary plot of the feasible space for the third goal has been shown in Figure II.19.

The superimposed region of the feasible spaces from the three goals is presented in Figure II.20.

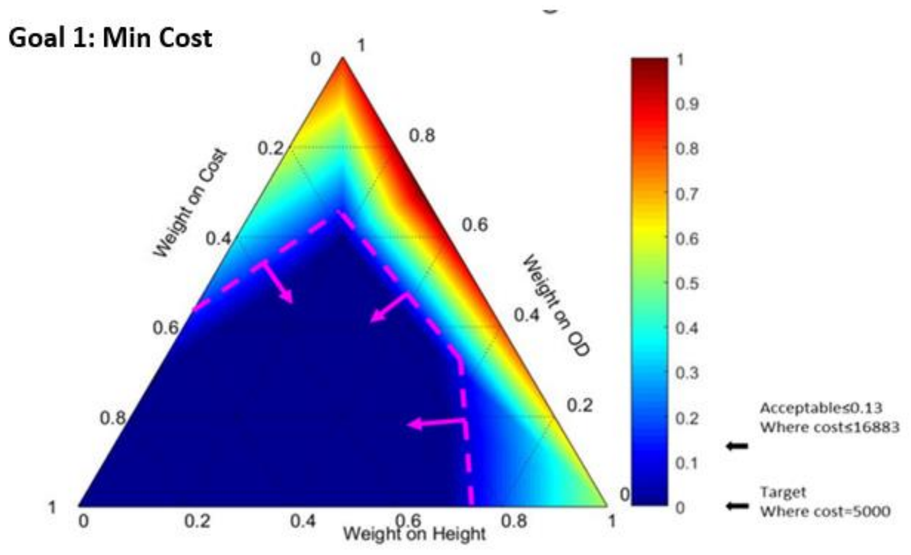


Figure II. 17: Satisficing space for goal 1 in the Elephant Stand Problem

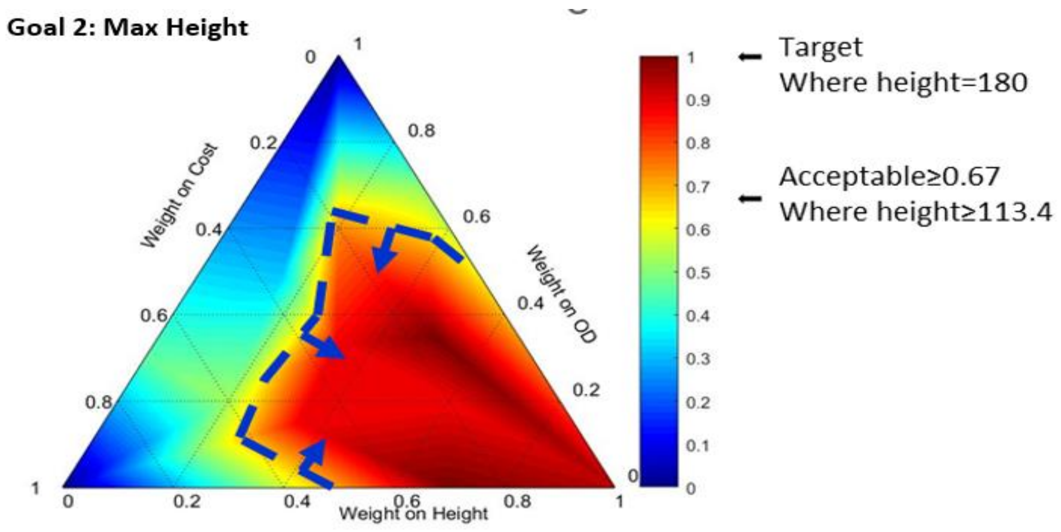


Figure II. 18: Satisficing space for goal 2 in Elephant Stand Problem

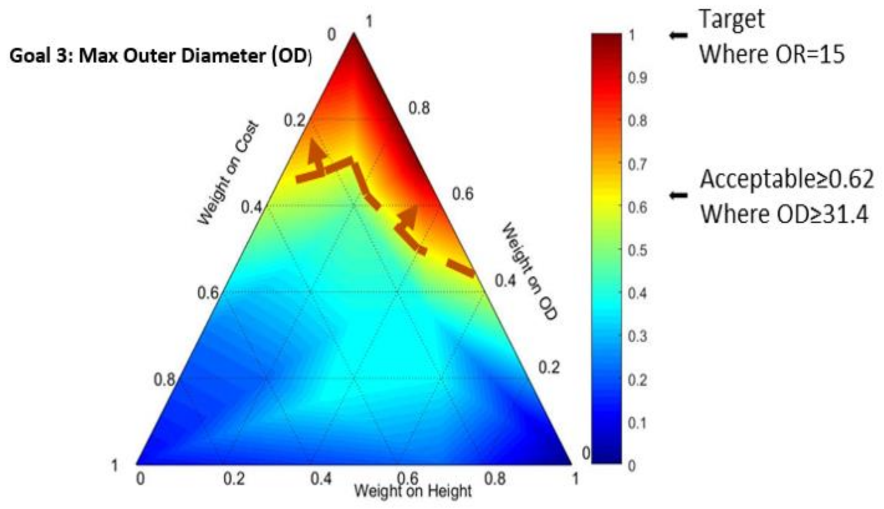


Figure II. 19: Satisficing space for goal 3 in Elephant Stand Problem

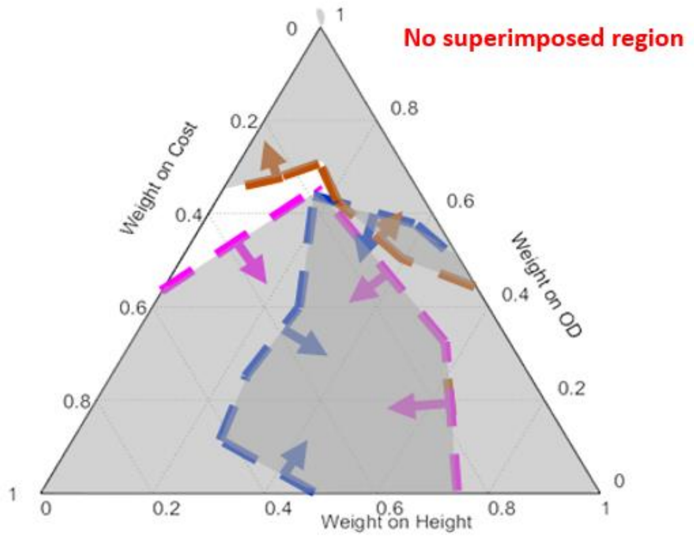


Figure II. 20: No Superimposed region of three goals in the Elephant Stand Problem

NO Satisficing Space in This Case:

In the case where no superimposed region is observed in Figure II.20, the absence of a satisficing space that satisfies all three goals simultaneously is indicated. There are two potential approaches to consider in order to expand or identify a satisficing space:

Altering Designer's Preferences: One approach is reassessing the designer's preferences regarding goal achievements. The range is selected based on the designer’s experience/knowledge/preference. It is not a pure “eye-balled” value; instead, it should be due to the combination of stakeholders’ requirements and the ternary space generated. A human designer uses his/her domain knowledge and experience to make the final decision based on preference. Then, the designer carries out the relaxations on the acceptable values based on his/her judgment.

Through this iterative process, designers can potentially identify a larger satisficing space that aligns with their revised preferences.

Modifying Problem Specifications/Formulation: Another approach is about the reevaluating the problem specifications or formulation. By revisiting the problem constraints, variables, or functional relationships, designers can discover opportunities to redefine the problem to get a larger, satisfying space. This may involve considering alternative design criteria, relaxing certain constraints, or introducing new variables. In addition, designers could have the opportunity to explore different design possibilities and expand the potential solution space by Modifying the problem formulation.

It is important to note that both approaches require careful consideration and analysis. Altering the designer's preferences or modifying the problem formulation should be guided by a comprehensive understanding of the design problem, its requirements, and the potential implications of any changes made. Iterative refinement and experimentation may be necessary to determine the most suitable adjustments that result in an expanded or identified satisficing space.

By exploring these two approaches in detail, designers can gain insights into potential avenues for expanding or identifying a satisficing space, allowing for a more comprehensive exploration of design solutions that meet their goals and requirements.

Altering Designers' Preferences for Elephant Stand Problem:

When considering the option of altering the designer's preference, it is important to emphasize that the selection of acceptable value ranges is not based solely on subjective judgment or "eyeballing." Instead, it involves a combination of stakeholders' requirements, the ternary space generated, and the designer's domain knowledge and experience. Various factors and considerations are taken into account in the process of defining acceptable values for each goal. The designer draws upon their expertise and understanding of the design problem, incorporating stakeholders' requirements and constraints. The acceptable values are determined through a thoughtful analysis that considers the design's feasibility, practicality, and desired outcomes.

For example, let us consider Goal 2. The designer assigns an initial normalized value of "0.67" based on the original value of "113.4". However, due to their experience and judgment, the designer may choose to relax the acceptable value to "112.8", resulting in a normalized value of "0.64". This relaxation is made with careful consideration of the impact on the design, balancing the desired outcome with practical constraints. (Figure II.22)

Similarly, the designer can apply similar relaxations to the acceptable values of all other goals. These relaxations are carried out based on the designer's judgment, considering the trade-offs, constraints, and requirements of the design problem.

It is crucial to highlight that the designer's role in adjusting the acceptable values is critical to the design process. Their domain knowledge, experience, and understanding of the project context enable them to make informed decisions that align with the stakeholders' requirements while considering the feasible design space. By incorporating these relaxations and adjustments to the acceptable values, designers can explore alternative design possibilities and potentially expand the satisficing space. That the design solution is well-informed, balanced, and aligned with the goals and preferences of the designer and stakeholders is ensured by this iterative process.

The steps described earlier have been successfully implemented and visualized in Figures II.21, II.22, and II.23. The adjustments made to the acceptable value ranges for each goal in order to explore a potentially expanded satisficing space are illustrated by these figures.

In Figure II.21, we depict the updated ternary plot representing the feasible space for the first goal after the relaxation of acceptable values. Similarly, in Figure II.22, the ternary plot of the modified feasible space for the second goal, reflecting the adjustments made based on the designer's judgment, has been shown. In Figure II.23, the ternary plot illustrating the revised feasible space for the third goal, incorporating the relaxations made to the acceptable values, has been represented.

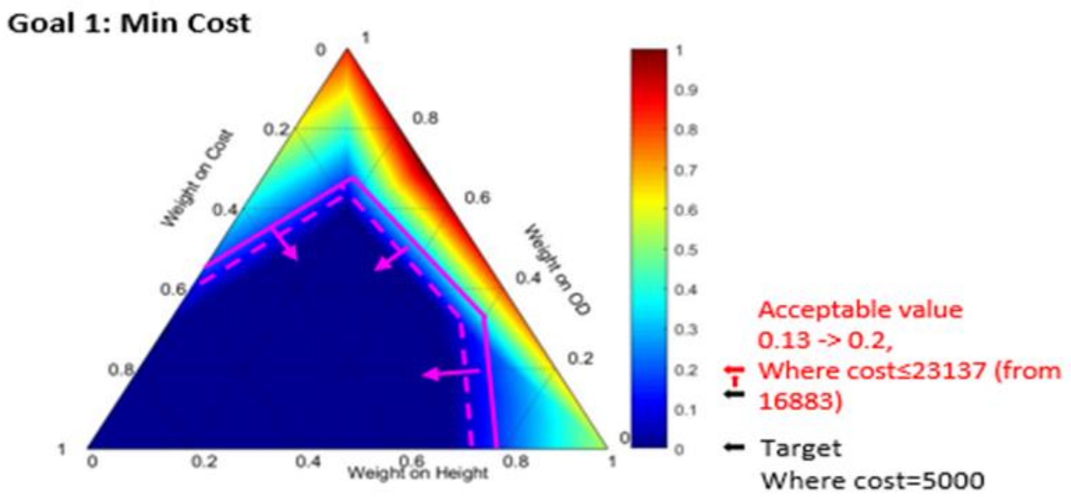


Figure II. 21: Satisficing space for goal 1 in the Elephant Stand Problem after implementing the Altering Designer's Preferences method.

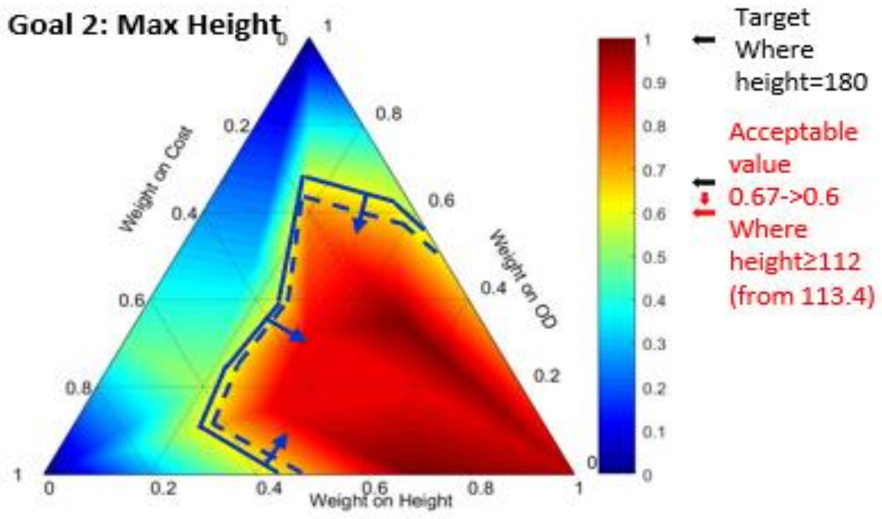


Figure II. 22: Satisficing space for goal 2 in the Elephant Stand Problem after implementing the Altering Designer's Preferences method.

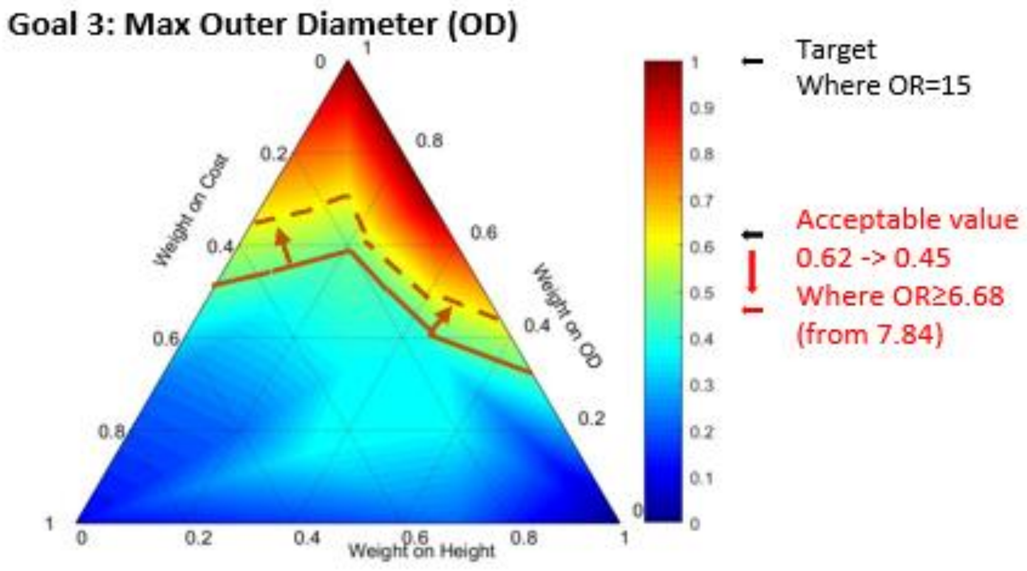


Figure II. 23: Satisficing space for goal 3 in the Elephant Stand Problem after implementing the Altering Designer's Preferences method.

To provide a comprehensive overview, the superimposed region of the updated feasible spaces for all three goals is presented in Figure II.24. In this figure, the intersecting region that defines the expanded satisficing space, where all three goals are simultaneously satisfied or exceeded, is demonstrated. By examining these figures, designers gain valuable insights into the effects of adjusting the acceptable value ranges on the feasible solution space. In addition, the clear identification and comprehension of the expanded satisficing space by designers, enabling the exploration of design solutions that better align with their preferences and stakeholder requirements, is facilitated by the visual representation of the ternary plots and their superimposition in Figure II.24.

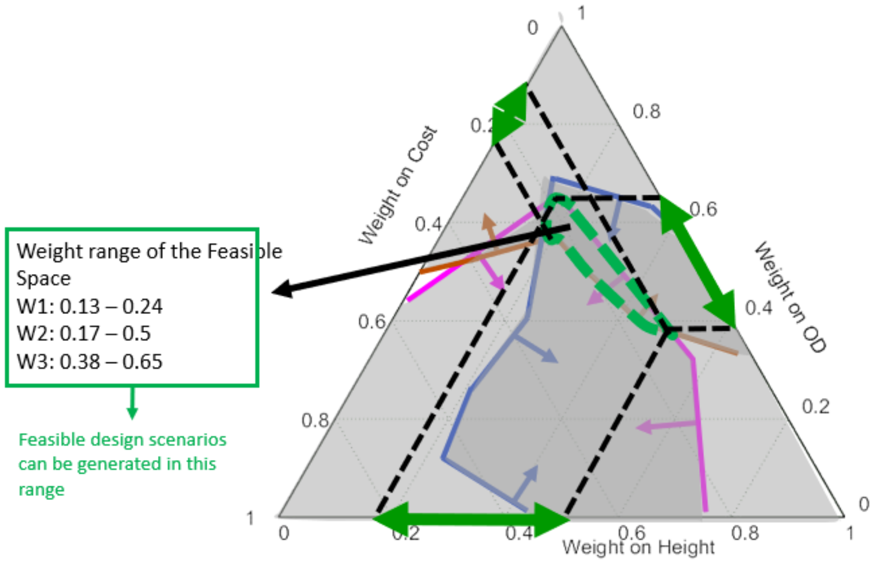
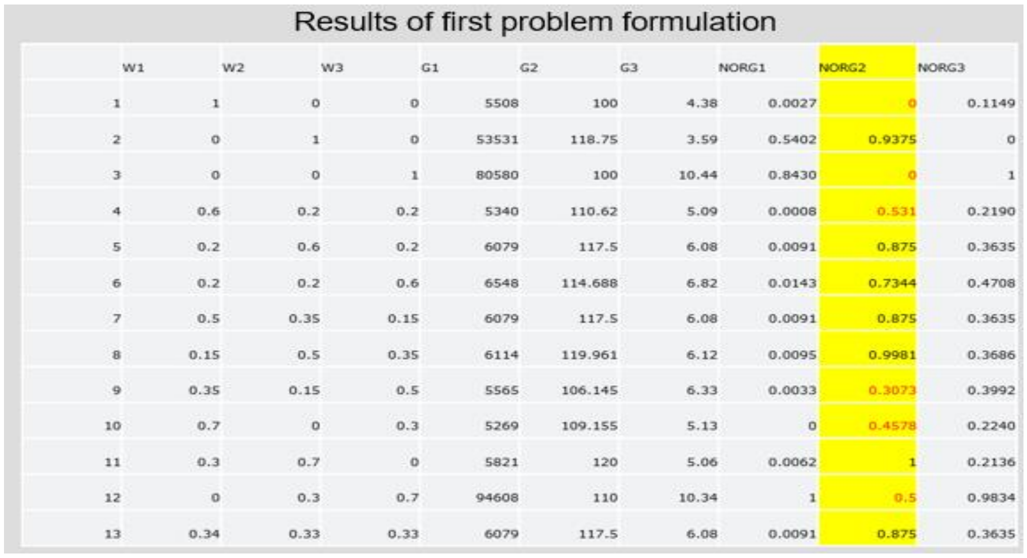


Figure II. 24: Satisficing space for all goal in Elephant Stand Problem , after implementing Altering Designer's Preferences method.

Figures II.21, II.22, II.23, and II.24 are presented as crucial tools for decision-making, offering designers a visual representation of the adjusted feasible spaces and the resulting satisficing space. With these figures, designers can gain insights into the design problem and support the exploration of improved design solutions within an expanded solution space, facilitating a deeper understanding of the design problem and supporting the exploration of improved design solutions within an expanded solution space.

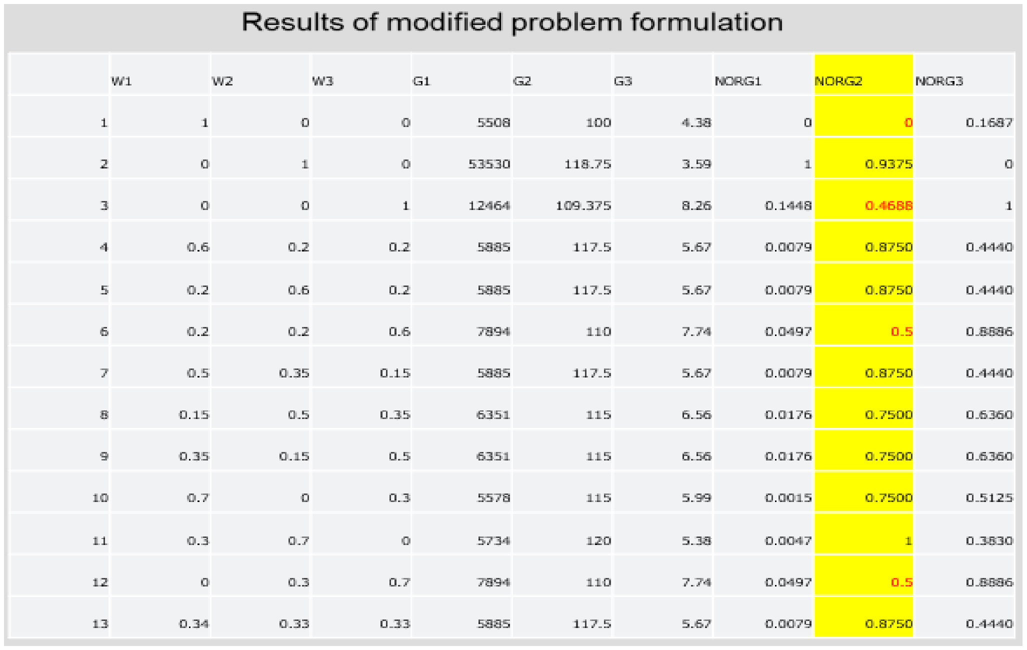
Modifying Problem Specifications/Formulation for Elephant Stand Problem.

The second approach to obtaining a satisficing space is modifying the problem specifications. A practical example is demonstrated through the comparison of two result tables representing various design scenarios in Figure II.25



Results of first problem formulation

	W1	W2	W3	G1	G2	G3	NORG1	NORG2	NORG3
1	1	0	0	5508	100	4.38	0.0027	0	0.1149
2	0	1	0	53531	118.75	3.59	0.5402	0.9375	0
3	0	0	1	80580	100	10.44	0.8430	0	1
4	0.6	0.2	0.2	5340	110.62	5.09	0.0008	0.531	0.2190
5	0.2	0.6	0.2	6079	117.5	6.08	0.0091	0.875	0.3635
6	0.2	0.2	0.6	6548	114.688	6.82	0.0143	0.7344	0.4708
7	0.5	0.35	0.15	6079	117.5	6.08	0.0091	0.875	0.3635
8	0.15	0.5	0.35	6114	119.961	6.12	0.0095	0.9981	0.3686
9	0.35	0.15	0.5	5565	106.145	6.33	0.0033	0.3073	0.3992
10	0.7	0	0.3	5269	109.155	5.13	0	0.4578	0.2240
11	0.3	0.7	0	5821	120	5.06	0.0062	1	0.2136
12	0	0.3	0.7	94608	110	10.34	1	0.5	0.9834
13	0.34	0.33	0.33	6079	117.5	6.08	0.0091	0.875	0.3635



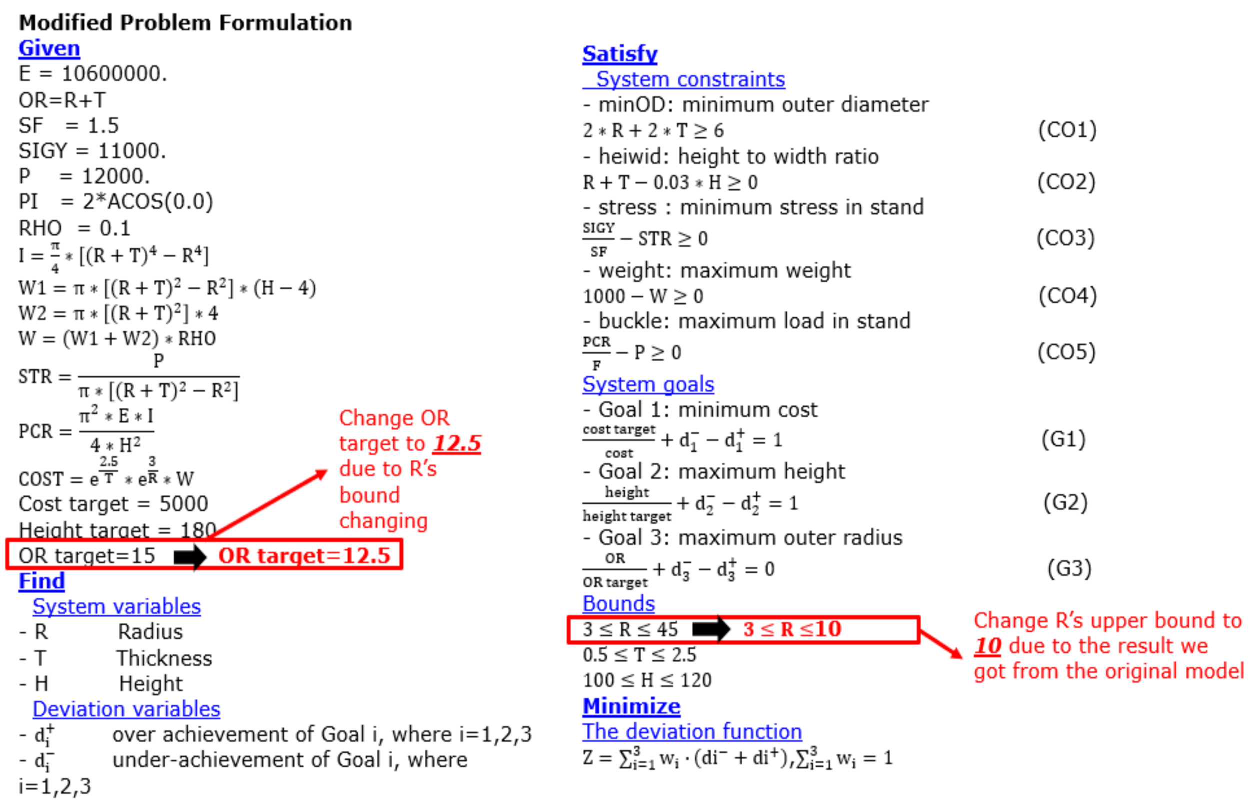
Results of modified problem formulation

	W1	W2	W3	G1	G2	G3	NORG1	NORG2	NORG3
1	1	0	0	5508	100	4.38	0	0	0.1687
2	0	1	0	53530	118.75	3.59	1	0.9375	0
3	0	0	1	12464	109.375	8.26	0.1448	0.4688	1
4	0.6	0.2	0.2	5885	117.5	5.67	0.0079	0.8750	0.4440
5	0.2	0.6	0.2	5885	117.5	5.67	0.0079	0.8750	0.4440
6	0.2	0.2	0.6	7894	110	7.74	0.0497	0.5	0.8886
7	0.5	0.35	0.15	5885	117.5	5.67	0.0079	0.8750	0.4440
8	0.15	0.5	0.35	6351	115	6.56	0.0176	0.7500	0.6360
9	0.35	0.15	0.5	6351	115	6.56	0.0176	0.7500	0.6360
10	0.7	0	0.3	5578	115	5.99	0.0015	0.7500	0.5125
11	0.3	0.7	0	5734	120	5.38	0.0047	1	0.3830
12	0	0.3	0.7	7894	110	7.74	0.0497	0.5	0.8886
13	0.34	0.33	0.33	5885	117.5	5.67	0.0079	0.8750	0.4440

Figure II. 25: Comparison of Design Scenarios; Original Problem Formulation vs. Modified Problem Specifications in Elephant Stand Problem

Based on Figure II.25, In the original problem formulation (top table), specifically concerning Goal 2, out of the 13 scenarios, six have values below 0.6 (indicated by the red entries in the yellow column on the top). To expand the solution space for this goal, a modification is made to the problem by focusing on adjusting the height to be closer to the target value of 120.

To achieve this, the upper bound of the radius is reduced, aiming for a "tall and thin" pedestal rather than a "short and flat" one. It is observed that even though the range of the radius was initially set as [3, 45], the results predominantly fall within the range of [3, 10]. Consequently, the range of the radius is changed to [3, 10]. The modified problem formulation has been presented in Figure II.26.



Modified Problem Formulation

Given

E = 10600000.
OR=R+T
SF = 1.5
SIGY = 11000.
P = 12000.
PI = 2*ACOS(0.0)
RHO = 0.1

$I = \frac{\pi}{4} * [(R+T)^4 - R^4]$

$W1 = \pi * [(R+T)^2 - R^2] * (H-4)$

$W2 = \pi * [(R+T)^2] * 4$

$W = (W1 + W2) * RHO$

$STR = \frac{P}{\pi * [(R+T)^2 - R^2]}$

$PCR = \frac{\pi^2 * E * I}{4 * H^2}$

$COST = e^{\frac{2.5}{T}} * e^{\frac{3}{R}} * W$

Cost target = 5000
Height target = 180
OR target=15 ➡ **OR target=12.5**

Change OR target to ***12.5*** due to R's bound changing

Find

System variables
- R Radius
- T Thickness
- H Height

Deviation variables
- d_i^+ over achievement of Goal i, where i=1,2,3
- d_i^- under-achievement of Goal i, where i=1,2,3

Satisfy

System constraints
- minOD: minimum outer diameter

$2 * R + 2 * T \geq 6$ (CO1)

- heiwid: height to width ratio

$R + T - 0.03 * H \geq 0$ (CO2)

- stress : minimum stress in stand

$\frac{SIGY}{SF} - STR \geq 0$ (CO3)

- weight: maximum weight

$1000 - W \geq 0$ (CO4)

- buckle: maximum load in stand

$\frac{PCR}{F} - P \geq 0$ (CO5)

System goals
- Goal 1: minimum cost

$\frac{cost\ target}{cost} + d_1^- - d_1^+ = 1$ (G1)

- Goal 2: maximum height

$\frac{height}{height\ target} + d_2^- - d_2^+ = 1$ (G2)

- Goal 3: maximum outer radius

$\frac{OR}{OR\ target} + d_3^- - d_3^+ = 0$ (G3)

Bounds

$3 \leq R \leq 45$ ➡ $\mathbf{3 \leq R \leq 10}$

$0.5 \leq T \leq 2.5$

$100 \leq H \leq 120$

Change R's upper bound to ***10*** due to the result we got from the original model

Minimize

The deviation function

$Z = \sum_{i=1}^{3} w_i \cdot (di^- + di^+), \sum_{i=1}^{3} w_i = 1$

Figure II. 26: Modified Elephant Stand Problem Formulation

By implementing this modification, more results with larger heights are obtained. The number of scenarios where Goal 2 has a normalized value below 0.6 is reduced from six to four (represented by the red entries in the yellow column on the second table in Figure II.25). A similar outcome is observed for Goal 3.

The results of implementing the modified problem specifications are depicted in Figures II.27 (for goal 1), II.28 (for goal 2), and II.29 (for goal 3). Visual representations of the refined feasible spaces for each respective goal have been provided in these figures. The refined feasible space for goal 1, highlighting the adjustments made to the problem specifications, is illustrated in Figure II.27. Similarly, in Figure II.28, the modified feasible space for Goal 2 is showcased, reflecting the impact of the problem specification modifications. The revised feasible space for goal 3, which incorporates the changes made to the problem formulation, is represented in Figure II.29.

The final satisficing result has been presented in Figure II.30 presents, combining the refined feasible spaces for all three goals. The achieved satisficing space, where all three goals are simultaneously satisfied or exceeded, is demonstrated in this figure.

By examining these figures, designers can gain valuable insights into the effects of modifying the problem specifications on the feasible solution space. A clearer understanding of the refined design space and the resulting satisficing solution that meets the goals and requirements of the design problem is enabled by the visual representations. These figures, namely II.27, II.28, II.29, and II.30 are critical visual aids for decision-making, as designers could get a comprehensive representation of the refined feasible spaces and the overall satisficing solution. A deeper understanding of the design problem is facilitated, and the exploration of improved design solutions within the modified solution space is supported through these figures.

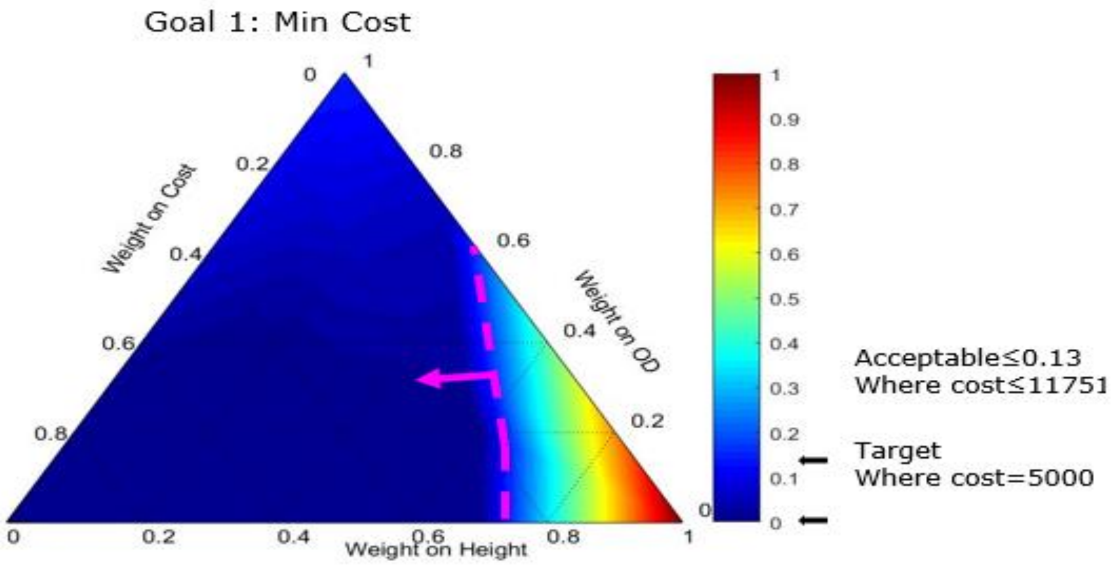


Figure II. 27: Refined Feasible Space for Goal 1 in Elephant Stand Problem; Adjusted Problem Specifications

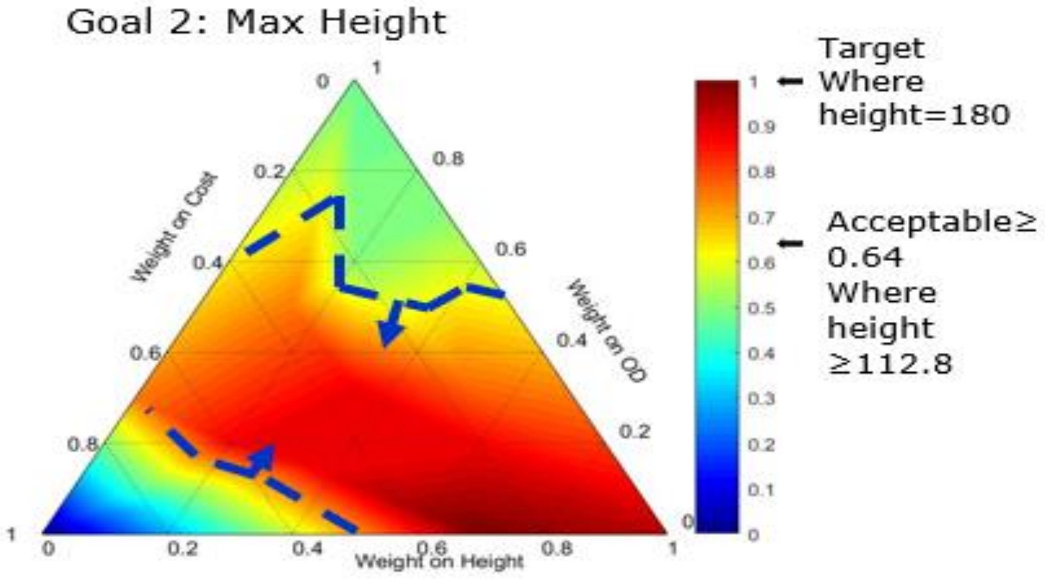


Figure II. 28: Refined Feasible Space for Goal 2 in Elephant Stand Problem; Adjusted Problem Specifications

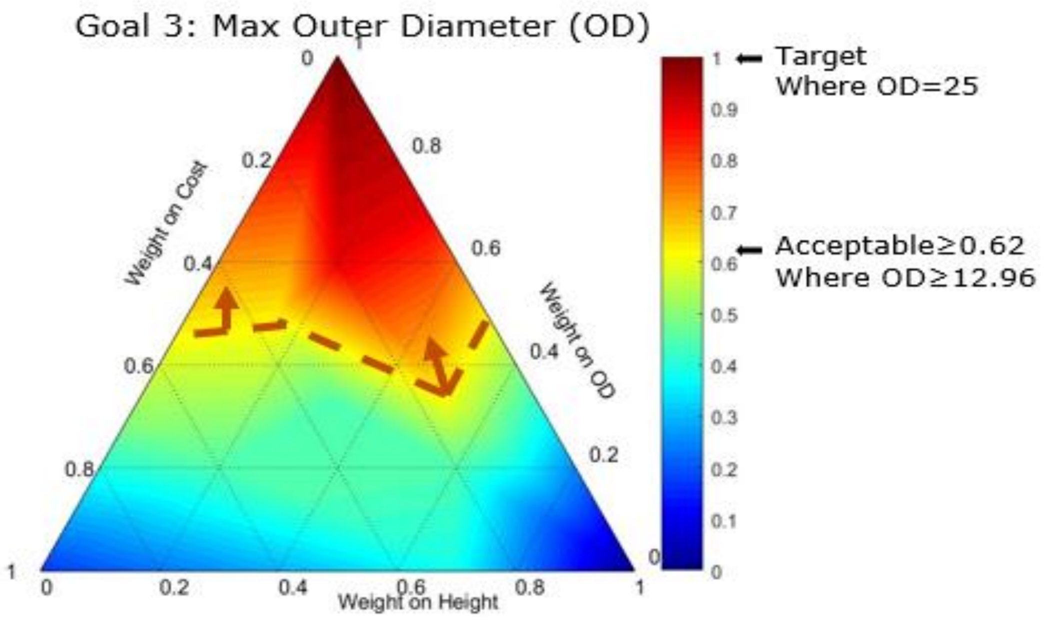


Figure II. 29: Refined Feasible Space for Goal 3 in Elephant Stand Problem; Adjusted Problem Specifications

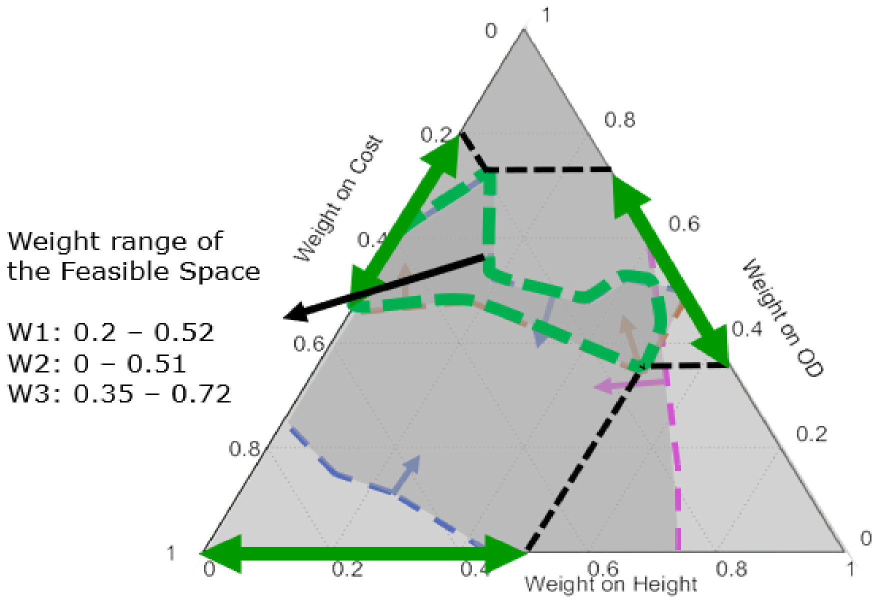


Figure II. 30: Final Satisficing Result for All Three Goals in Elephant Stand Problem with Adjusted Problem Specifications

The importance of even a small adjustment in the acceptable value that can lead to a significant expansion of the solution space is highlighted in this example. By fine-tuning specific problem specifications, designers can satisfy the design outcomes and achieve a larger satisficing space. Through careful analysis and iterative modifications, designers can identify opportunities to redefine problem constraints, variables, or functional relationships to better align with their design objectives. An exploration of alternative design possibilities is enabled by this process, and the chances of finding a satisficing space that accommodates the desired outcomes within the revised problem specifications are enhanced.

Following a thorough and detailed description of the first example, the Elephant Stand problem, which was presented in Archimedean form, our attention now turns towards the second problem, the One-Stage Reduction Gearbox, presented in preemptive form. This particular example is a part of a paper that was published in 2022. In the forthcoming section, we will delve into the intricacies of the One-Stage Reduction Gearbox problem.

II.4.2 One-Stage Reduction Gearbox in Preemptive Form

The paper titled "Designing concurrently and hierarchically coupled engineered systems," authored by Gehendra Sharma, Janet K. Allen, and Farrokh Mistree, was published in 2022. (Sharma, G., Allen, J.K. and Mistree, F., 2022, ”Designing Concurrently and Hierarchically Coupled Engineered Systems, Engineering Optimization”, Pages 1556-1576. DOI: 10.1080/0305215X.2022.2098953) The field of engineering optimization is contributed to by this paper, which investigates the design process for concurrently and hierarchically coupled engineered systems. It presents a comprehensive framework and methodology for addressing complex design problems involving interconnected and interdependent components. Valuable insights into the challenges and considerations involved in designing such systems are provided by the paper, with an emphasis on the importance of concurrent and hierarchical design approaches. By citing this article in the verification section of our DSIDES Wrapper, we acknowledge the foundational work and research conducted in this area and leverage the example presented in the paper to support and validate our own work. To link to this article: https://doi.org/10.1080/0305215X.2022.2098953

II.4.2.1 Problem statement

The problem statement is about the design of a one-stage reduction gearbox with specific requirements and objectives. The goal is to recommend gear design decisions (gear material and dimensions) and shaft design decisions (shear shaft material strength and dimensions) that result in a high-quality design with low weight, low height, and high torque transmission capability.

Note: From the problem perspective, the decision problem involves two sib problems (the gear decision problem and the shaft decision problem). The gear decision problem in itself involves two decisions (geometry decision and material decision). Hence, the formulation has three decision problems (three levels).

The problem includes the following aspects:

- Required gear ratio of 4.
- Minimum input torque of 80 Nm at 3500 rpm.
- Endurance of at least 10^7 fatigue cycles.
- Gear cutting using a rack cutter with a pressure angle of 20°.
- Required reliability of at least 99%.
- Design goals of low weight and low height while achieving maximum torque transmission capability.

The problem involves the coupled design of gears and shafts, where gear dimension design, material selection, and shaft dimension design are interconnected. The gear design decisions (material and dimensions) form a compromise decision, while the shaft design decision forms a separate compromise decision. The gear material selection decision is horizontally coupled with the compromise decisions, and the compromise decisions are vertically coupled to the shaft design decision.

The mathematical formulations for gear and shaft design are based on the work of Budynas and Nisbett (2019). The problem formulation involves defining the interactions between decisions, considering correction factors for material properties, and exploring the design space through multiple decision scenarios.

The limitations, boundaries, variables, and constraints in this problem include:

- Gear material alternatives and their properties.
- Gear dimensions, such as number of teeth, pitch diameter, and face width.
- Shaft dimensions, such as diameter and length.
- Material strength and fatigue properties.
- Bending stress and contact stress numbers.
- Speed, fatigue cycles, and reliability requirements.
- Weight assignment for decision scenarios.
- Constraints on achieving the desired gear ratio, torque transmission capability, weight, height, and reliability.

II.4.2.2 cDSP Formulation

Word Formulation:

Given

System parameters

Torque (T) ≥ 80 Nm

Gear reduction ratio (G) = 4

Pressure angle (α) = 200

Density (δ) = 7800 kg/m3

Speed (N) = 3500 rpm

Overload factor, Ko = 1

Dynamic factor, Kv = 1

Size factor, Ks = 1

Load distribution factor, KH = 1

Rim thickness factor, KB = 1

Geometry factor for bending strength, YJ = 1

Elastic coefficient, ZE = 1

Surface condition factor, ZR = 1.25

Geometry factor for pitting resistance, ZI = 1

AGMA factor of safety for bending, SF = 1

AGMA factor of safety for contact, SH = 1

Stress cycle factor for bending stress, YN = 1

Temperature factor, Y = 1

Reliability factor, YZ = 1

Stress cycle life factor for contact, ZN = 1

Hardness ratio factor for pitting, ZW = 1

Required gear ratio: The desired gear ratio for the reduction gearbox is 4.

Input torque: The input torque, denoted as T, should be at least 80 Nm at 3500 rpm.

Fatigue cycles: The gears must endure at least 10^7 fatigue cycles.

Pressure angle: The gears are cut using a rack cutter with a pressure angle (α) of 20°.

Reliability requirement: The gearbox should have a reliability of at least 99%.

Find

system variables

Gear design variables:

m: Gear module (a parameter that determines the size of the gear teeth)

Z: Number of teeth (the number of teeth on the gear)

b: Face width (the width of the gear tooth)

T: Input torque (the torque applied to the gearbox)

Shaft design variables:

D_i: Input shaft diameter (the diameter of the hole in the center of the gear)

D_o: Output shaft diameter (the overall diameter of the gear)

Sy: Shear strength for shaft material (a material property representing the maximum stress a material can withstand before undergoing permanent deformation in shear)

X_1, X_2, X_3, X_4, X_5 : bending stress number and contact stress number for the five Gear materials.

Deviation Variables:

e_1^-: under-achievement for gear selection goal

e_1^+: over achievement for gear selection goal

d_1^-: under-achievement for gear mass goal

d_1^+: over achievement for gear mass goal

d_2^-: under-achievement for gear size goal

d_2^+: over achievement for gear size goal

d_3^-: under-achievement for gear torque goal

d_3^+: over achievement for gear torque goal

d_4^-: under-achievement for shaft mass goal

d_4^+: over achievement for shaft mass goal

Satisfy

system constraints

Selection constraint for gear material alternatives: The sum of the gear material selection variables (x1, x2, x3, x4, x5) must equal 1.

Maximum allowable bending stress constraint: A constraint ensuring the bending stress in the gear is within the allowable limit.

Maximum allowable contact stress constraint: A constraint ensuring the contact stress in the gear is within the allowable limit.

Maximum allowable shear stress constraint for shafts: Constraints on the shear stress in the shafts to ensure they are within the allowable limit.

Constraints on deviation variables: Constraints ensuring the deviation variables (di^-, d_i^+) for various compromise goals are non-negative and their product is zero.

system goals

G1: Coupled selection goal : Maximize the merit function (MF) considering the gear material selection, subject to a deviation constraint.

Coupled compromise gear goals:

G2: Minimize the mass of the gear, subject to a deviation constraint.

G3: Minimize the size of the gear, subject to a deviation constraint.

G4: Maximize the torque of the gear, subject to a deviation constraint.

G5: Coupled compromise shaft goal : Minimize the mass of the shafts, subject to a deviation constraint.

Bounds

The upper and lower limit of system variables

Minimize

The deviation function.

Mathematic Formulation

Given

Material attributes

St1 = 184.2
St2 = 266.9
St3 = 301.5
St4 = 342.8
St5 = 380.0
Sc1 = 600.0
Sc2 = 944.0
Sc3 = 1088.0
Sc4 = 1034.0
Sc5 = 1241.0

Merit function Calculations

I1 = 0.0
I2 = 0.0

```
     I3  = 0.5
     I4  = 0.5
     P1   = 0.161
     P2  = 0.177
     P3  = 0.212
     P4   = 0.242
     P5  = 0.218
     a12 = 0.068
     a13 = 0.270
     a14 = 0.235
     a22 = 0.170
     a23 = 0.225
     a24 = 0.235
     a32 = 0.218
     a33 = 0.180
     a34 = 0.235
     a42 = 0.238
     a43 = 0.216
     a44 = 0.176
     a52 = 0.306
     a53 = 0.108
     a54 = 0.118
     C1=P1*(b*m**2*z**2)
     C2=P2*(b*m**2*z**2)
     C3=P3*(b*m**2*z**2)
     C4=P4*(b*m**2*z**2)
     C5=P5*(b*m**2*z**2)
     a11 = 0.4 - (C1/(C1+C2+C3+C4+C5))
     a21 = 0.4- (C2/(C1+C2+C3+C4+C5))
     a31 = 0.4- (C3/(C1+C2+C3+C4+C5))
     a41 = 0.4- (C4/(C1+C2+C3+C4+C5))
     a51 = 0.4- (C5/(C1+C2+C3+C4+C5))
     MF1= I1*a11+I2*a12+I3*a13+I4*a14
     MF2= I1*a21+I2*a22+I3*a23+I4*a24
     MF3= I1*a31+I2*a32+I3*a33+I4*a34
     MF4= I1*a41+I2*a42+I3*a43+I4*a44
     MF5= I1*a51+I2*a52+I3*a53+I4*a54
```

Select material properties.

```
     St =            X1*St1+X2*St2+X3*St3+X4*St4+X5*St5
     Sc =            X1*Sc1+X2*Sc2+X3*Sc3+X4*Sc4+X5*Sc5
     TorC=((Sc*m*z)**2*b)/(29810*191**2)
```

Find:

System variables

Gear material, x1, x2, ..., x5

Gear design variables :

Module (m)

Number of teeth (z)

Face width (b)

T: Input torque (the torque applied to the gearbox)

Shaft design variables:

Shear strength for shaft material (Sy)

Input shaft diameter (Di)

Output shaft diameter (Do)

Deviation variables

e_1^-, e_1^+ , d_1^+ , d_1^- , d_2^+ , d_2^- , d_3^+ , d_3^- , d_4^- , d_4^+

Satisfy

System constraints

Selection constraint for gear material alternatives

$$\sum_{i=1}^{5} X_i = 1$$

Maximum allowable bending stress constraint

$1 - 10.76\, YZ\, (T / St \cdot m^2 \cdot z^2 \cdot b) \geq 0$ $\rightarrow$ *St* is the maximum allowable bending stress

Maximum allowable contact stress constraint

$1 - (186.42\, YZ / S_c) \sqrt{(3.88\, (T / m \cdot z)\, (1/ b \cdot m \cdot z))} \geq 0$ $\rightarrow$ S_c is the maximum allowable contact

stress

Maximum allowable shear stress constraint for shafts

$1 – (25.46T / D_i^3 * Sy) \geq 0$ → D_i is the input shaft diameter

$1 – (101.86 T / D_o^3 *Sy) \geq 0$ → D_o is the output shaft diameter

Compromise system constraints

Constraints on deviation variables

$d_i^+ \geq 0$, $d_i^- \geq 0$ and $d_i^+ \cdot d_i^- = 0$ for i = 1, 2 and 3

System goals

Coupled selection goal

G1: Maximize merit function (*MF*) : Alt 1

$MF_i\,(m,b,z)\,x_i + e_1^- - e_1^+ = 1$

Coupled compromise gear goals

G2: Minimize mass of gear → mgear 2

$$\frac{Gear\ mass\ target}{Gear\ mass} + \mathrm{d}_1^- - \mathrm{d}_1^+ = 1$$

G3: Minimize size of gear → sgear 3

$$\frac{Size\ target}{Size} + \mathrm{d}_2^- - \mathrm{d}_2^+ = 1$$

G4: Maximize torque of gear → Torque 4

$$\frac{T}{torque\ target} + \mathrm{d}_3^- - \mathrm{d}_3^+ = 1$$

Coupled compromise shaft goal.

G5: Minimize mass of shafts → mshaft 5

$$\frac{Shaft\ mass\ target}{Shaft\ mass} + \mathrm{d}_4^- - \mathrm{d}_4^+ = 1$$

where

MF$MF_i(m, b, z) = \sum_{j=1}^{4} I_j R_{ij}(m, b, z)$

Bounds:

B1: $24 \leq b \leq 72$ (mm) B4: $200 \leq Sy \leq 400$ B7: $0 \leq x1 \leq 1$ B10: $0 \leq x4 \leq 1$

B2: $3 \leq m \leq 6$ (mm) B5: $20 \leq Di \leq 40$ B8: $0 \leq x2 \leq 1$ B11: $0 \leq x5 \leq 1$

B3: $18 \leq z \leq 30$ B6: $30 \leq Do \leq 50$ B9: $0 \leq x3 \leq 1$

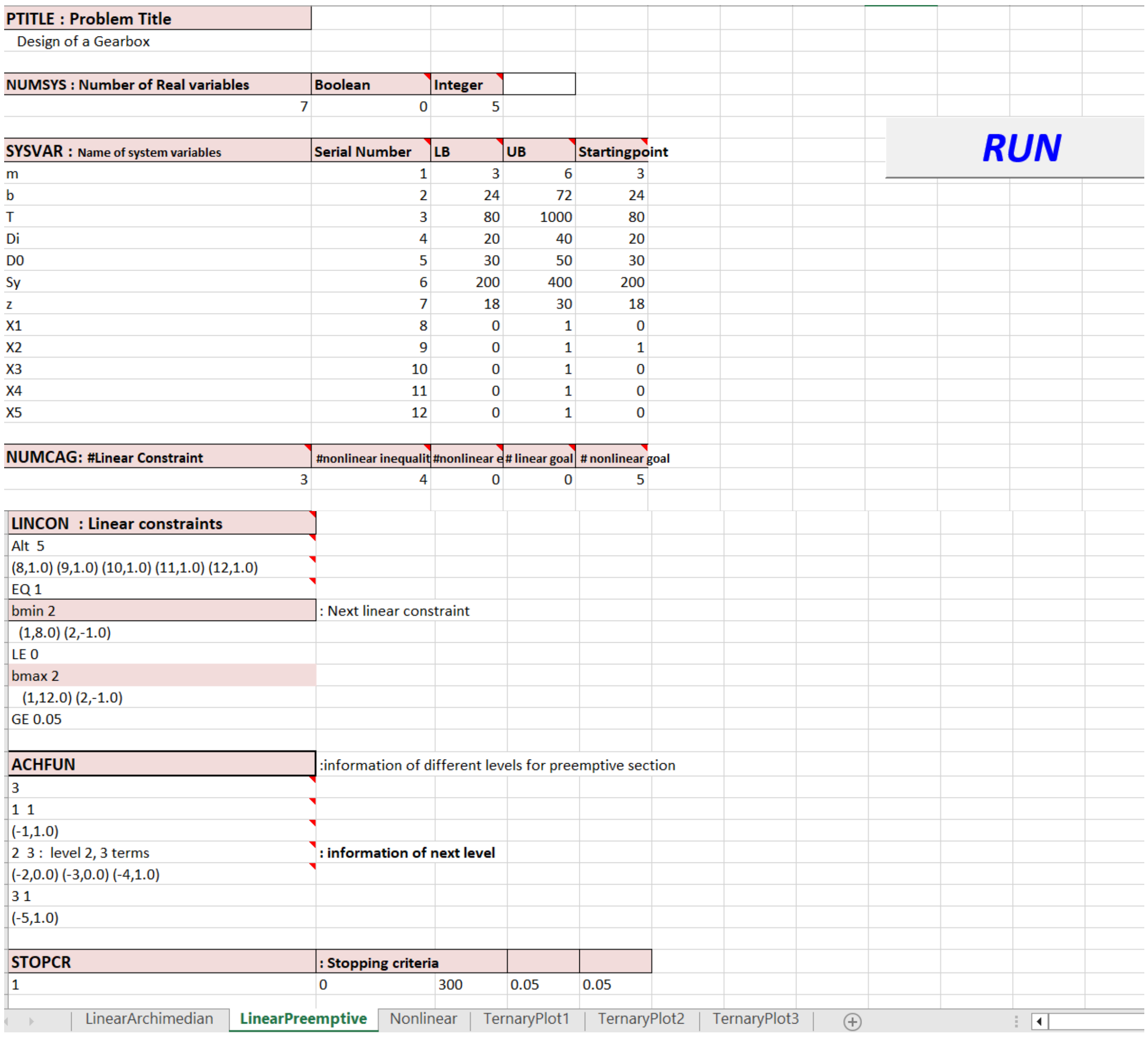
Minimize

The deviation function (pre-emptive form)

$Z = [\, e_1^- \sum_{i=1}^{3} w_i \cdot (d_i^- + d_i^+), d_4^-],\ \sum_{i=1}^{3} w_i = 1$ → w_i are weights of the various compromise goals.

II.4.2.3 Implementation in DSIDES Wrapper

At this stage, we have transformed the problem into a compromise decision support problem (cDSP) form. Our next step is to implement the cDSP using the DSIDES Wrapper. The first sheet we need to complete is the LinearSheet. Since it is a preemptive form, the user needs to prepare the input for the LinearPreemptive sheet, which is depicted in Figure II.31 of our documentation.



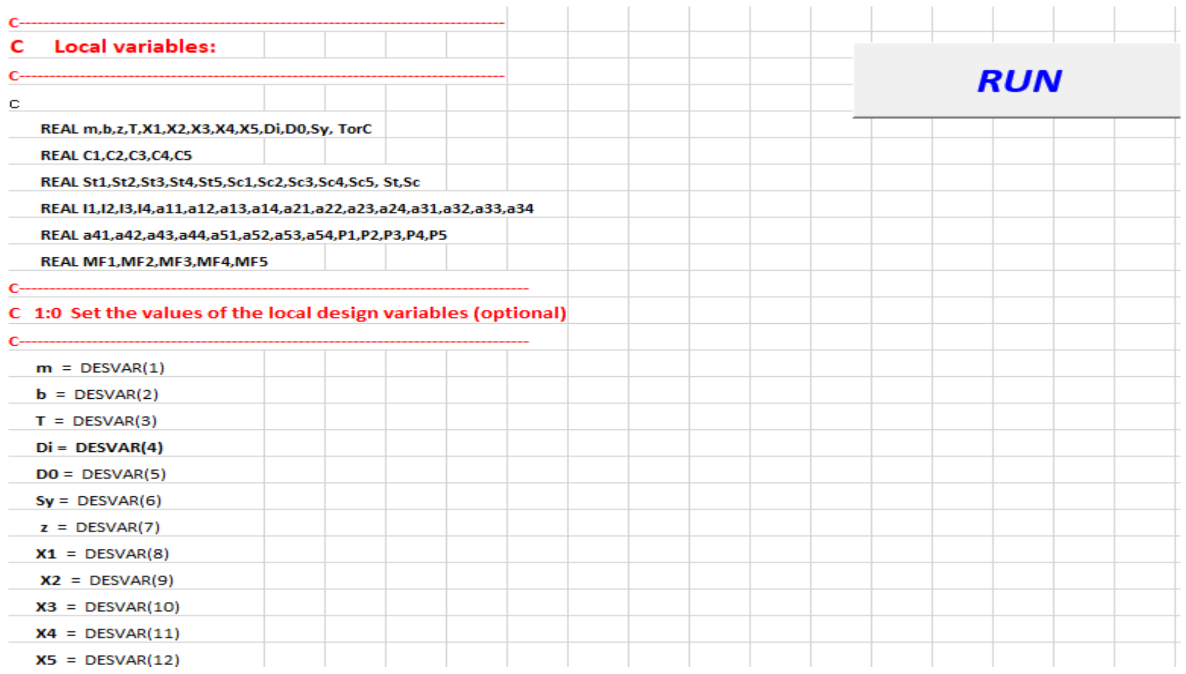
PTITLE : Problem Title				
Design of a Gearbox				
NUMSYS : Number of Real variables	Boolean	Integer		
7	0	5		
SYSVAR : Name of system variables	Serial Number	LB	UB	Startingpoint
m	1	3	6	3
b	2	24	72	24
T	3	80	1000	80
Di	4	20	40	20
D0	5	30	50	30
Sy	6	200	400	200
z	7	18	30	18
X1	8	0	1	0
X2	9	0	1	1
X3	10	0	1	0
X4	11	0	1	0
X5	12	0	1	0
NUMCAG: #Linear Constraint	#nonlinear inequalit	#nonlinear e	# linear goal	# nonlinear goal
3	4	0	0	5
LINCON : Linear constraints				
Alt 5				
(8,1.0) (9,1.0) (10,1.0) (11,1.0) (12,1.0)				
EQ 1				
bmin 2	: Next linear constraint			
(1,8.0) (2,-1.0)				
LE 0				
bmax 2				
(1,12.0) (2,-1.0)				
GE 0.05				
ACHFUN	:information of different levels for preemptive section			
3				
1 1				
(-1,1.0)				
2 3 : level 2, 3 terms	: information of next level			
(-2,0.0) (-3,0.0) (-4,1.0)				
3 1				
(-5,1.0)				
STOPCR	: Stopping criteria			
1	0	300	0.05	0.05

Figure II. 31: Overview of LinearPreemptive Sheet in Design of Gearbox Problem

Once the LinearSheet has been completed and run, a DAT file in the form of a notepad file will be generated in the output folder.

Now, the next step is about completing the NonlinearSheet. The NonlinearSheet, as represented in Figure II.30 of our documentation, is where the incorporation of nonlinear components into the cDSP construct takes place.

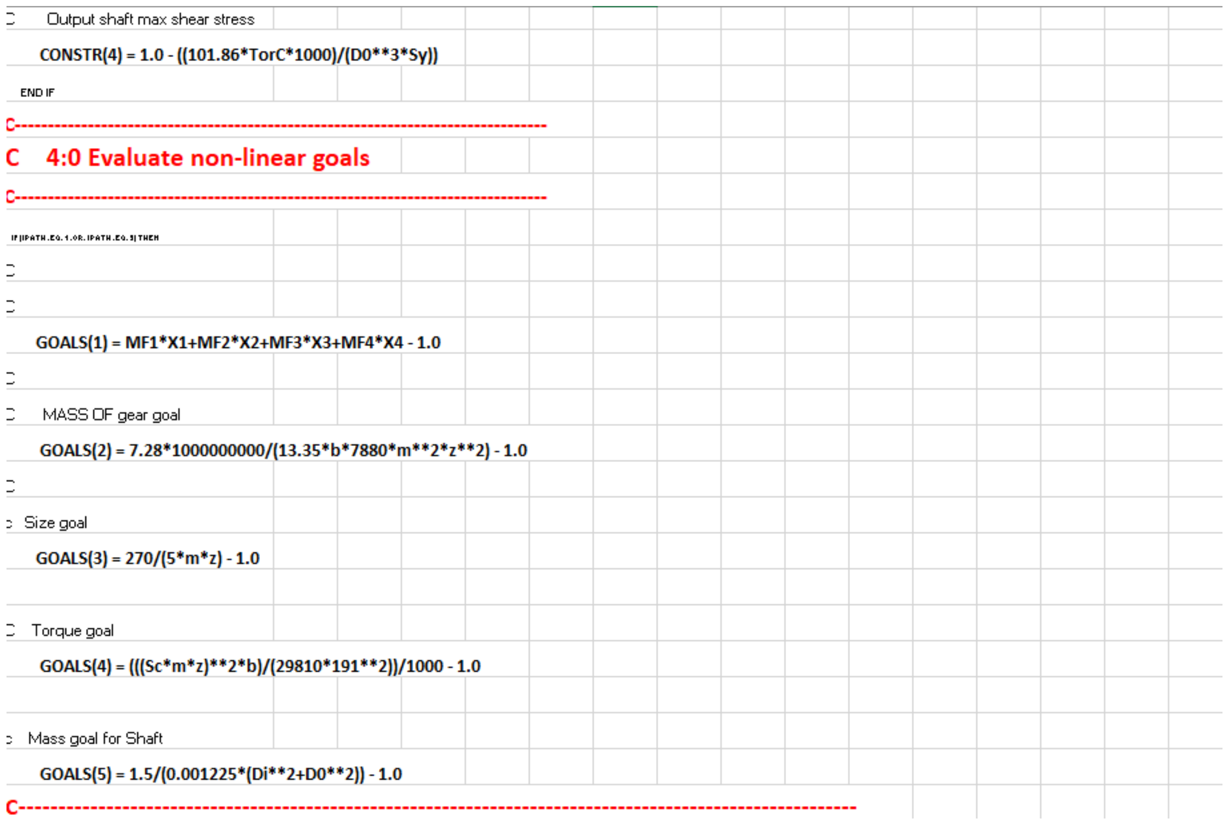
To use the NonlinearSheet, the functions or equations that describe the behavior of the nonlinear components within the cDSP will be specified using the graphical interface of the DSIDES software. The inclusion of nonlinear aspects in the system can be simulated and analyzed using the NonlinearSheet. To complete the NonlinearSheet, reference should be made to Figure II.32 in the documentation, in which an overview of the sheet's structure and organization has been provided. By inputting the appropriate equations and parameters into the NonlinearSheet, the nonlinear dynamics of the cDSP can be effectively modeled and studied.



```
C-------------------------------------------------------------------------
C    Local variables:
C-------------------------------------------------------------------------
C
    REAL m,b,z,T,X1,X2,X3,X4,X5,Di,D0,Sy, TorC
    REAL C1,C2,C3,C4,C5
    REAL St1,St2,St3,St4,St5,Sc1,Sc2,Sc3,Sc4,Sc5, St,Sc
    REAL I1,I2,I3,I4,a11,a12,a13,a14,a21,a22,a23,a24,a31,a32,a33,a34
    REAL a41,a42,a43,a44,a51,a52,a53,a54,P1,P2,P3,P4,P5
    REAL MF1,MF2,MF3,MF4,MF5
C---------------------------------------------------------------------------
C  1:0  Set the values of the local design variables (optional)
C---------------------------------------------------------------------------
   m  =  DESVAR(1)
   b  =  DESVAR(2)
   T  =  DESVAR(3)
   Di =  DESVAR(4)
   D0 =  DESVAR(5)
   Sy =  DESVAR(6)
    z  =  DESVAR(7)
   X1  =  DESVAR(8)
    X2  =  DESVAR(9)
   X3  =  DESVAR(10)
   X4  =  DESVAR(11)
   X5  =  DESVAR(12)
```

```
C--------------------------------------------------------------------------------
C  2.0  Perform analysis relevant to non-linear constraints and goals
C--------------------------------------------------------------------------------
c   Material attributes
    St1 = 184.2
    St2 = 266.9
    St3 = 301.5
    St4 = 342.8
    St5 = 380.0
    Sc1 = 600.0
    Sc2 = 944.0
    Sc3 = 1088.0
    Sc4 = 1034.0
    Sc5 = 1241.0

c   Merit function Calculations
    I1   = 0.0
    I2  = 0.0
    I3  = 0.5
    I4  = 0.5
    P1   = 0.161
    P2  = 0.177
    P3  = 0.212
    P4   = 0.242
    P5  = 0.218
    a12 = 0.068
    a13 = 0.270
    a14 = 0.235
    a22 = 0.170
    a23 = 0.225
    a24 = 0.235
    a32 = 0.218
    a33 = 0.180
    a34 = 0.235
    a42 = 0.238
    a43 = 0.216
    a44 = 0.176
    a52 = 0.306
    a53 = 0.108
    a54 = 0.118

    C1=P1*(b*m**2*z**2)
    C2=P2*(b*m**2*z**2)
    C3=P3*(b*m**2*z**2)
    C4=P4*(b*m**2*z**2)
    C5=P5*(b*m**2*z**2)

    a11 = 0.4 - (C1/(C1+C2+C3+C4+C5))
    a21 = 0.4- (C2/(C1+C2+C3+C4+C5))
    a31 = 0.4- (C3/(C1+C2+C3+C4+C5))
    a41 = 0.4- (C4/(C1+C2+C3+C4+C5))
    a51 = 0.4- (C5/(C1+C2+C3+C4+C5))
```

```
   MF1= I1*a11+I2*a12+I3*a13+I4*a14
   MF2= I1*a21+I2*a22+I3*a23+I4*a24
   MF3= I1*a31+I2*a32+I3*a33+I4*a34
   MF4= I1*a41+I2*a42+I3*a43+I4*a44
   MF5= I1*a51+I2*a52+I3*a53+I4*a54

c  Select material properties

   St =                      X1*St1+X2*St2+X3*St3+X4*St4+X5*St5
   Sc =                      X1*Sc:

   TorC=((Sc*m*z)**2*b)/(29810*191**2)

C-----------------------------------------------------------------
C  3:0  Evaluate non-linear constraints
C-----------------------------------------------------------------
 IF (IPATH.EQ.1.OR.IPATH.EQ.2) THEN

c    BENDING stress constraint.
     CONSTR(1) = 1.0 - ((10760*TorC) / (St*b*m**2*z))

C    Contact stress constraint.
      CONSTR(2) = 1.0 - ((191/Sc)*((29810*TorC)/(b*m**2*z**2))**0.5)

C    Input shaft max shear stress
      CONSTR(3) = 1.0 - ((25.46*TorC*1000)/(Di**3*Sy))
```



```
C     Output shaft max shear stress
      CONSTR(4) = 1.0 - ((101.86*TorC*1000)/(D0**3*Sy))
      END IF
C-----------------------------------------------------------------
C     4:0 Evaluate non-linear goals
C-----------------------------------------------------------------
 IF(IPATH.EQ.1.OR.IPATH.EQ.3) THEN
C
C
      GOALS(1) = MF1*X1+MF2*X2+MF3*X3+MF4*X4 - 1.0
C
C     MASS OF gear goal
      GOALS(2) = 7.28*1000000000/(13.35*b*7880*m**2*z**2) - 1.0
C
C   Size goal
      GOALS(3) = 270/(5*m*z) - 1.0

C   Torque goal
      GOALS(4) = (((Sc*m*z)**2*b)/(29810*191**2))/1000 - 1.0

C   Mass goal for Shaft
      GOALS(5) = 1.5/(0.001225*(Di**2+D0**2)) - 1.0
C---------------------------------------------------------------------------------------
```

Figure II. 32: Overview of NonLinear Sheet in Design of Gearbox Problem

Note:

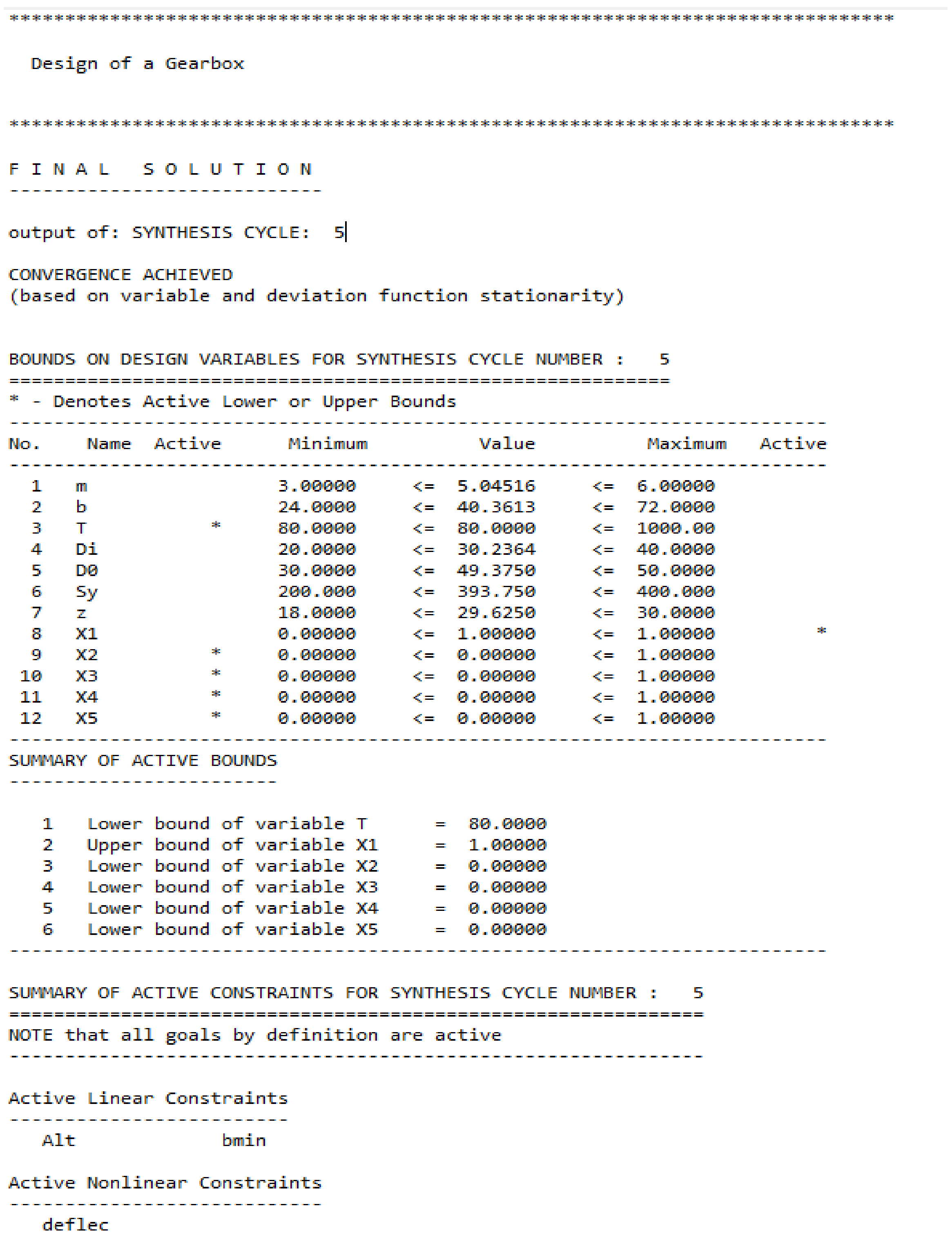
There is one important consideration to keep in mind. When importing the data into the NonlinearSheet form, allocating a space of eight characters for each data entry is important. Considering that the Wrapper will link these sheets as input to the primary DSIDES software, written in Fortran, it is important to note that the initial 8 characters will be disregarded by the program. If the user neglects these 8 empty spaces, it may lead to an error in the Wrapper's operation, resulting in the failure to generate the final results.

Final Result:

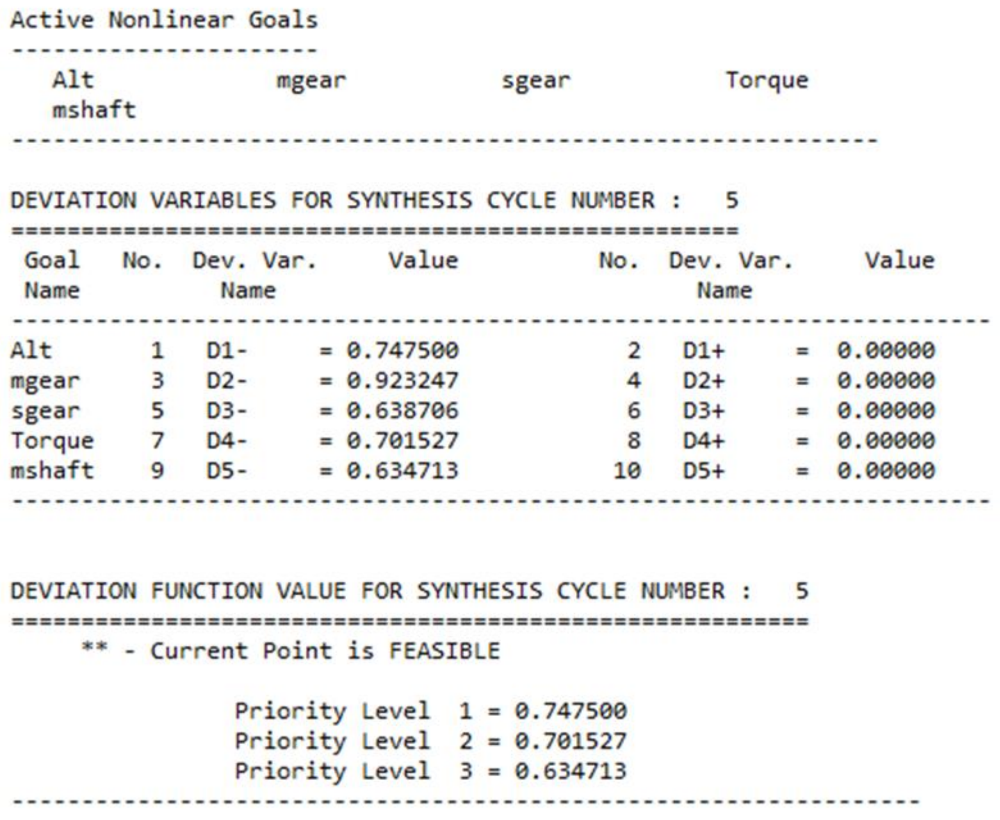
After the execution of the LinearSheet and NonlinearSheet, an output file will be generated in the Output_Files folder. The output file containing the simulation results will appear as a pop-up on the screen, providing immediate access to the results. Furthermore, a copy of the output file will be saved in the output_file folder of the project, ensuring that the results are easily accessible and can be reviewed at a later time if required.

For the purpose of presenting the final results visually, an image capturing the contents of the output file has been attached in Figure II:33 . A Snapshot of the simulation outcomes for a quick overview of the obtained results has been represented in this figure.

By having the output file readily available and with the attached image for reference, the obtained results of the cDSP model's simulation can be conveniently analyzed and interpreted.



```
********************************************************************************

  Design of a Gearbox


********************************************************************************

F I N A L   S O L U T I O N
----------------------------

output of: SYNTHESIS CYCLE:  5

CONVERGENCE ACHIEVED
(based on variable and deviation function stationarity)


BOUNDS ON DESIGN VARIABLES FOR SYNTHESIS CYCLE NUMBER :   5
============================================================
* - Denotes Active Lower or Upper Bounds
--------------------------------------------------------------------------
No.    Name  Active      Minimum          Value          Maximum   Active
--------------------------------------------------------------------------
  1   m                 3.00000     <=  5.04516     <=  6.00000
  2   b                 24.0000     <=  40.3613     <=  72.0000
  3   T           *     80.0000     <=  80.0000     <=  1000.00
  4   Di                20.0000     <=  30.2364     <=  40.0000
  5   D0                30.0000     <=  49.3750     <=  50.0000
  6   Sy                200.000     <=  393.750     <=  400.000
  7   z                 18.0000     <=  29.6250     <=  30.0000
  8   X1                0.00000     <=  1.00000     <=  1.00000        *
  9   X2          *     0.00000     <=  0.00000     <=  1.00000
 10   X3          *     0.00000     <=  0.00000     <=  1.00000
 11   X4          *     0.00000     <=  0.00000     <=  1.00000
 12   X5          *     0.00000     <=  0.00000     <=  1.00000
--------------------------------------------------------------------------
SUMMARY OF ACTIVE BOUNDS
------------------------

   1   Lower bound of variable T      =  80.0000
   2   Upper bound of variable X1     =  1.00000
   3   Lower bound of variable X2     =  0.00000
   4   Lower bound of variable X3     =  0.00000
   5   Lower bound of variable X4     =  0.00000
   6   Lower bound of variable X5     =  0.00000
--------------------------------------------------------------------------

SUMMARY OF ACTIVE CONSTRAINTS FOR SYNTHESIS CYCLE NUMBER :   5
===============================================================
NOTE that all goals by definition are active
---------------------------------------------------------------

Active Linear Constraints
-------------------------
   Alt             bmin

Active Nonlinear Constraints
----------------------------
   deflec
```



```
Active Nonlinear Goals
----------------------
   Alt            mgear          sgear          Torque
   mshaft
--------------------------------------------------------------

DEVIATION VARIABLES FOR SYNTHESIS CYCLE NUMBER :   5
====================================================
 Goal   No.  Dev. Var.    Value          No.  Dev. Var.    Value
 Name          Name                             Name
--------------------------------------------------------------------
Alt       1   D1-     = 0.747500          2   D1+     =  0.00000
mgear     3   D2-     = 0.923247          4   D2+     =  0.00000
sgear     5   D3-     = 0.638706          6   D3+     =  0.00000
Torque    7   D4-     = 0.701527          8   D4+     =  0.00000
mshaft    9   D5-     = 0.634713         10   D5+     =  0.00000
--------------------------------------------------------------------


DEVIATION FUNCTION VALUE FOR SYNTHESIS CYCLE NUMBER :   5
=========================================================
     ** - Current Point is FEASIBLE

                Priority Level  1 = 0.747500
                Priority Level  2 = 0.701527
                Priority Level  3 = 0.634713
----------------------------------------------------------------
```

Figure II. 33: Overview of Final Result in Design of a Gearbox

To accurately generate the ternary plot and identify the satisficing space, users should refer to the guidelines outlined in Section II.2.4 Ternary Plot and Section II.4.1.4 Find Satisficing Space Based on the Ternary Plot. The visual representations of the ternary plots and satisficing space have been provided in the paper, which users can utilize as a reference to validate their results and ensure they are following the correct steps for identifying the satisficing space accurately.

Following the description of the first and second examples, presented in Archimedean and Preemptive form, our attention now turns towards the third problem, the hot rod rolling problem involving sequential decisions, presented in preemptive form. This particular example is a part of a paper published in 2022. In the forthcoming section, we will delve into the details of the Hot rod rolling problem.

II.4.3 Hot rod rolling problem in Archimedean Form

The paper titled "An Information-Decision Framework to Support Cooperative Decision Making in the Top-Down Design of Cyber-Physical-Manufacturing Systems" was presented at the ASME 2022 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference (IDETC/CIE2022) held in St. Louis, Missouri from August 14-17, 2022. An innovative framework designed to facilitate cooperative decision-making in the top-down design process of cyber-physical manufacturing systems is introduced in the paper (IDETC2022-90836). A structured approach for integrating information and decision-making processes is provided by the framework to enhance collaboration and satisfying system design. By leveraging this information-decision framework, researchers and practitioners can effectively address the challenges associated with the design of complex manufacturing systems in the era of cyber-physical integration. This example has been used for verification of our wrapper.

II.4.3.1 Problem statement

The hot rolling process is an essential manufacturing technique used to shape metal sheets or rods by passing them through a sequence of heated rollers. The objective of the hot rolling process is to achieve specific product dimensions while considering various constraints and objectives to ensure high-quality and efficient production.

The problem involves determining the satisfy configuration of process parameters, including roller speeds, roller temperatures, material properties, and other relevant factors, to achieve the desired product specifications. These specifications typically include dimensions such as thickness, width, and length, as well as properties like surface finish and mechanical strength. In addition to the desired product dimensions, the hot rolling process must take into account a range of constraints and considerations. Material properties, such as the type of metal being rolled, its initial temperature, and its mechanical properties, affect the behavior of the metal during the rolling process and impose limitations on the achievable product dimensions.

A crucial role in hot rolling is played by temperature conditions, as they are influential in the material's plasticity and thermal expansion. Maintaining optimal temperature profiles throughout the rolling process is vital to controlling material flow, preventing defects, and ensuring uniformity of the final product. Furthermore, the production throughput requirements, such as the desired production rate or the number of products to be manufactured within a specific timeframe, must be considered to efficiently meet production targets. The hot rolling problem is involved addressing challenges related to process stability, energy consumption, and equipment limitations. For instance, the selection of appropriate roller speeds and temperatures should take into account the mechanical limits of the rolling equipment and the need to avoid excessive strain on the material.

The problem at hand is involved optimizing the hot rod rolling process for C-Mn steels to produce steel rods with specific mechanical properties. The goal is to achieve target values of Yield Strength (YS) of 330 MPa, Tensile Strength (TS) of 750 MPa, and Hardness (HV) of 170The process is comprised of sequential stages, including reheating, rolling, and cooling. Decisions made at each stage impact subsequent stages due to the interconnected nature of the process. A compromise decision support problem approach is employed, considering microstructural characteristics such as Ferrite grain size, Ferrite fraction, and Pearlite interlamellar spacing. The challenge is managing conflicts between the cooling process and the desired mechanical properties, seeking a satisficing solution that balances objectives and constraints. The goal is to develop a cooperative decision-making method that satisfy the process while considering the interconnected decisions and achieving the specified mechanical property targets.

II.4.3.2 cDSP Formulation

Word Formulation:

Given

Yield Strength (YS):

YS1 is calculated based on certain input variables.

YS2 is determined by a combination of input variables.

YS3 is obtained through a combination of input variables.

The overall Yield Strength (YS) is the sum of YS1, YS2, and YS3.

Tensile Strength (TS):

TS1 is computed using specific input variables.

TS2 is determined by a combination of input variables.

TS3 is calculated based on certain input variables.

The overall Tensile Strength (TS) is the sum of TS1, TS2, and TS3.

Hardness (HV):

HV1 is obtained through a combination of input variables.

HV2 is determined based on a specific input variable.

The overall Hardness (HV) is the sum of HV1 and HV2.

- Ferrite fraction – Xf : The proportion or percentage of ferrite present in the microstructure after the cooling process.

- Ferrite grain size - dα (μm): The size or average diameter of individual ferrite grains in the microstructure after cooling.
- Pearlite interlamellar spacing - S0 (μm): The distance between adjacent layers or plates of pearlite in the microstructure after cooling.
- Silicon concentration - [Si] (%): The amount or concentration of silicon in the material after cooling.
- Nitrogen concentration - [N] (%) : The amount or concentration of nitrogen in the material after cooling.
- Manganese concentration - [Mn] (%): The amount or concentration of manganese in the material after cooling.

Find

Design Variables

Ferrite Fraction	:	Ferfrc
Ferrite Grain Size (microns)	:	FGrnSz
Pearlite Inter-lamellar Spacing (microns)	:	Peaspc
Si Concentration (%)	:	Si
N Concentration (%)	:	N
Mn Concentration ()	:	Mn

Deviation variables

- d_i^+	over achievement of Goal i, where i=1,2,3
- d_i^-	under-achievement of Goal i, where i=1,2,3

Satisfy

System constraints

Yield Strength (YS) constraint: The Yield Strength value should be within the range of 220 to 330 MPa.

Tensile Strength (TS) constraint: The Tensile Strength value should fall between 450 and 750 MPa.

Hardness (HV) constraint: The Hardness value should be within the range of 131 to 170.

System goals

Goal 1: Maximize YS [MPa]	:	Yield Strength (YS)
Goal 2: Maximize TS [MPa]	:	Tensile Strength (TS)
Goal 3: Maximize HV	:	Hardness (HV)

Bounds

The upper and lower limit of design variables.

Minimize

The deviation function.

Mathematic Formulation:

Given

Yield Strength (YS) target = 330 MPa,

Tensile strength (TS) target = 750 MPa

Hardness (HV), target = 170

YS1=(478.*N**0.5)+(1200.*0.024)

YS2=FERFRC*(77.7+(59.9*MN)+(9.1*(FGRNSZ*0.001)**(-0.5)))

```
YS3=(1.-FERFRC)*(145.5+(3.5*PEASPC**(-0.5)))

YS=YS1+YS2+YS3

TS1=FERFRC*(20.+(2440.*N**0.5)+(18.5*(0.001*FGRNSZ)**(- 0.5)))

TS2=(750.*(1.-FERFRC))+(92.5*SI)

TS3=(3.*((1.-FERFRC)**0.5)*(PEASPC**(-0.5)))

TS=TS1+TS2+TS3

HV1=FERFRC*(361.-0.357*700.+(50.*SI))

HV2=175.*(1.-FERFRC)

HV=HV1+HV2
```

Find

Design Variables

X1: Ferrite fraction – Xf

X2: Ferrite Grain size - dα (µm)

X3: Pearlite interlamellar spacing - S0 (µm)

X4: Silicon conc. - [Si] (%)

X5: Nitrogen conc. - [N] (%)

X6: Manganese conc. after cooling - [Mn] (%)

Deviation variables

- d_i^+ over achievement of Goal i, where i=1,2,3

- d_i^- under-achievement of Goal i, where i=1,2,3

Satisfy

System constraints

CONSTR(1)= (YS/220.)-1 $\rightarrow$ Minimum Yield Strength

CONSTR(2)=1.-(YS/330.) $\rightarrow$ Maximum Yield Strength

CONSTR(3)=(TS/450.)-1 $\rightarrow$ Minimum Tensile Strength

CONSTR(4)=1.-(TS/750.) $\rightarrow$ Maximum Tensile Strength

CONSTR(5)=(HV/131.)-1 $\rightarrow$ Minimum Hardness

CONSTR(6)=1.-(HV/170.) $\rightarrow$ Maximum Hardness

System goals

$\{YS(X_j)/YS_{target}\} + d_1^- - d_1^+ = 1$

$\{TS(X_j)/TS_{target}\} + d_2^- - d_2^+ = 1$

$\{HV(X_j)/HV_{target}\} + d_3^- - d_3^+ = 1$

Bounds

Variable bounds

$0.1 \leq X1 \leq 0.9$

$8 \leq X2 \leq 25$ (µm)

$0.15 \leq X3 \leq 0.25$ (µm)

$0.18 \leq X4 \leq 0.3$ (%)

$0.007 \leq X5 \leq 0.009$ (%)

$0.7 \leq X6 \leq 1.5$ (%)

Deviation variable bounds

$d_i^+, d_i^- >= 0$ and $d_i^+ * d_i^- = 0$ $i = 1, 2, 3$

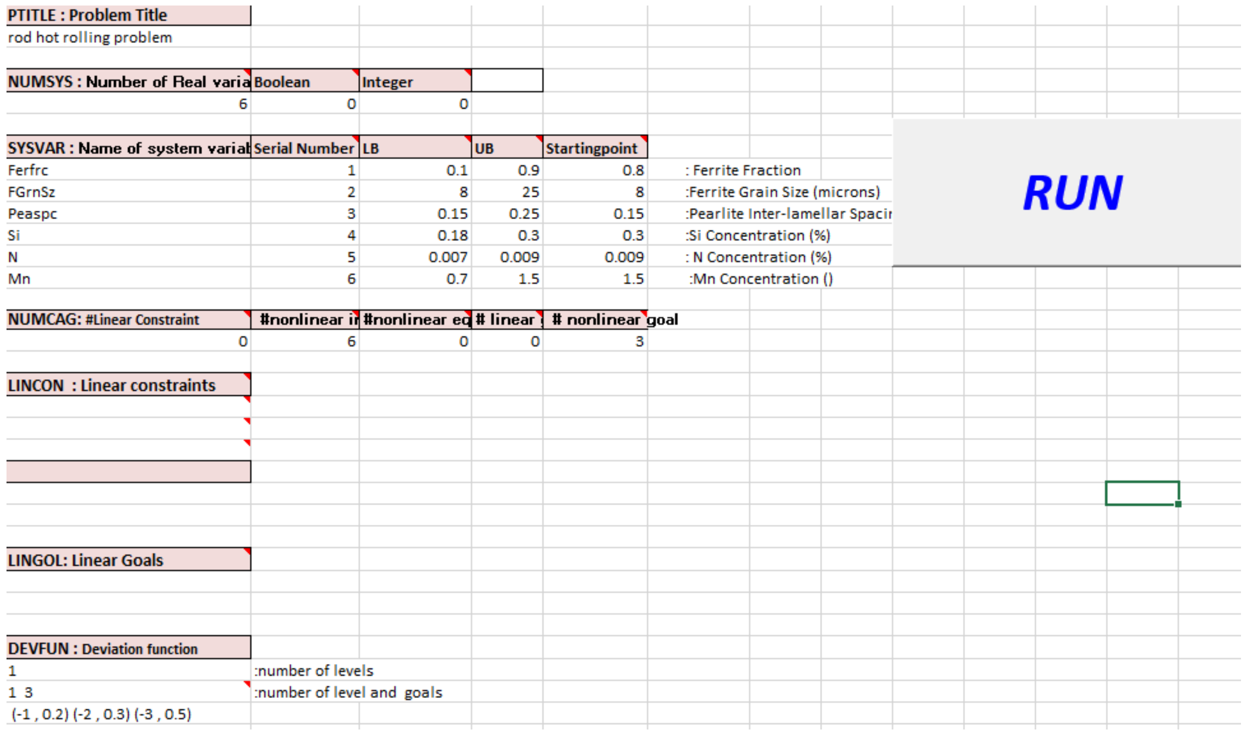
Minimize

The deviation function given below needs to be minimized.

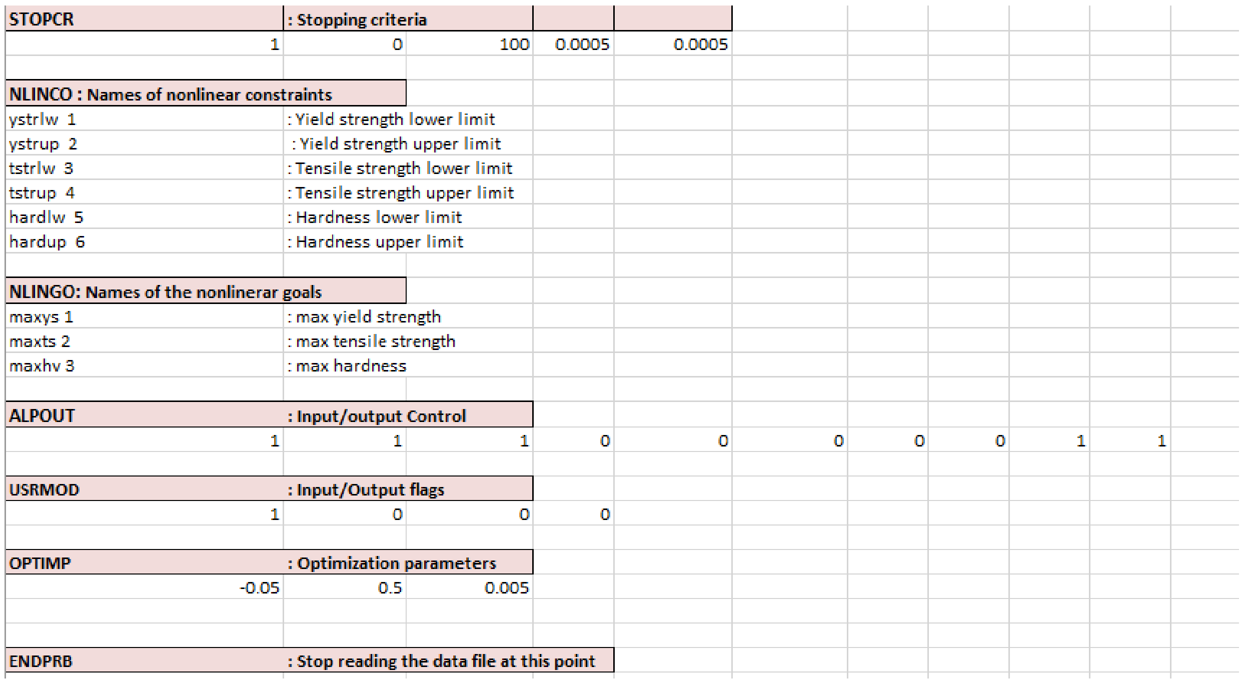
Min $Z_1 = \Sigma W_i (d_i^+ + d_i^-)$, where $\Sigma W_i = 1$ and i = 1, 2, 3

II.4.3.3 Implementation in DSIDES Wrapper

At this stage, we have transformed the problem into a compromise decision support problem (cDSP) construct. Our next step is to implement the cDSP using the DSIDES Wrapper. The first sheet we need to complete is the Linear Sheet, which is depicted in Figure II.34 of our documentation.



PTITLE : Problem Title					
rod hot rolling problem					
NUMSYS : Number of Real varia	Boolean	Integer			
6	0	0			
SYSVAR : Name of system variat	Serial Number	LB	UB	Startingpoint	
Ferfrc	1	0.1	0.9	0.8	: Ferrite Fraction
FGrnSz	2	8	25	8	:Ferrite Grain Size (microns)
Peaspc	3	0.15	0.25	0.15	:Pearlite Inter-lamellar Spacir
Si	4	0.18	0.3	0.3	:Si Concentration (%)
N	5	0.007	0.009	0.009	: N Concentration (%)
Mn	6	0.7	1.5	1.5	:Mn Concentration ()
NUMCAG: #Linear Constraint	#nonlinear ir	#nonlinear eq	# linear	# nonlinear goal	
0	6	0	0	3	
LINCON : Linear constraints					
LINGOL: Linear Goals					
DEVFUN : Deviation function					
1	:number of levels				
1 3	:number of level and goals				
(-1 , 0.2) (-2 , 0.3) (-3 , 0.5)					



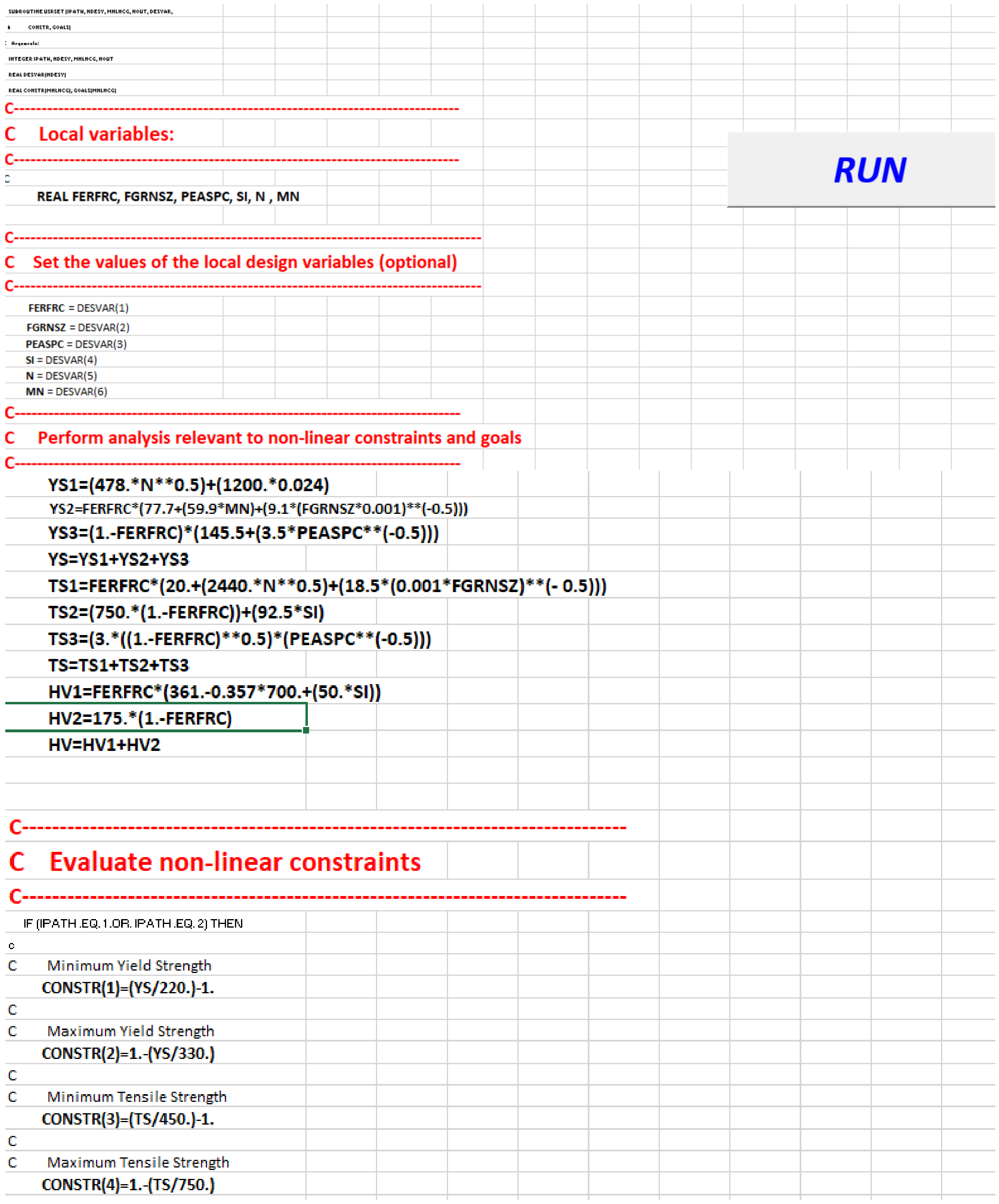
STOPCR	**: Stopping criteria**								
1	0	100	0.0005	0.0005					
NLINCO : Names of nonlinear constraints									
ystrlw 1	: Yield strength lower limit								
ystrup 2	: Yield strength upper limit								
tstrlw 3	: Tensile strength lower limit								
tstrup 4	: Tensile strength upper limit								
hardlw 5	: Hardness lower limit								
hardup 6	: Hardness upper limit								
NLINGO: Names of the nonlinerar goals									
maxys 1	: max yield strength								
maxts 2	: max tensile strength								
maxhv 3	: max hardness								
ALPOUT	**: Input/output Control**								
1	1	1	0	0	0	0	0	1	1
USRMOD	**: Input/Output flags**								
1	0	0	0						
OPTIMP	**: Optimization parameters**								
-0.05	0.5	0.005							
ENDPRB	**: Stop reading the data file at this point**								

Figure II. 34: Overview of Linear sheet for hot rod rolling problem.

Once the LinearSheet has been completed and run, a DAT file in the form of a notepad file will be generated in the output_files folder. The required input information from the linear simulation for DSIDES software is contained in this DAT file.

Now, the next step involves completing the NonlinearSheet. The NonlinearSheet, as represented in Figure II.35 of our documentation, is where the incorporation of nonlinear components into the cDSP model takes place.

To proceed with the NonlinearSheet, the DSIDES software's graphical wrapper will be utilized to define the nonlinear equations and parameters relevant to the system. The inclusion of nonlinear aspects in the system to be simulated and analyzed is enabled by the NonlinearSheet. By completing this sheet, a more comprehensive understanding of the system's behavior can be obtained, taking into account the nonlinearities present. To complete the NonlinearSheet, reference should be made to Figure II.35 in the documentation, which is an overview of the sheet's structure and organization. By inputting the appropriate equations and parameters into the NonlinearSheet, the nonlinear dynamics of the cDSP can be effectively modeled and studied.



```
SUBROUTINE USRSET (IPATH, NDESV, NNLNCG, NOUT, DESVAR,
&         CONSTR, GOALS)
C  Arguments:
INTEGER IPATH, NDESV, NNLNCG, NOUT
REAL DESVAR(NDESV)
REAL CONSTR(NNLNCG), GOALS(NNLNCG)
C-----------------------------------------------------------------------------------
C    Local variables:
C-----------------------------------------------------------------------------------
C
     REAL FERFRC, FGRNSZ, PEASPC, SI, N , MN

C-----------------------------------------------------------------------------------
C   Set the values of the local design variables (optional)
C-----------------------------------------------------------------------------------
    FERFRC = DESVAR(1)
    FGRNSZ = DESVAR(2)
    PEASPC = DESVAR(3)
    SI = DESVAR(4)
    N = DESVAR(5)
    MN = DESVAR(6)
C-----------------------------------------------------------------------------------
C    Perform analysis relevant to non-linear constraints and goals
C-----------------------------------------------------------------------------------
      YS1=(478.*N**0.5)+(1200.*0.024)
      YS2=FERFRC*(77.7+(59.9*MN)+(9.1*(FGRNSZ*0.001)**(-0.5)))
      YS3=(1.-FERFRC)*(145.5+(3.5*PEASPC**(-0.5)))
      YS=YS1+YS2+YS3
      TS1=FERFRC*(20.+(2440.*N**0.5)+(18.5*(0.001*FGRNSZ)**(- 0.5)))
      TS2=(750.*(1.-FERFRC))+(92.5*SI)
      TS3=(3.*((1.-FERFRC)**0.5)*(PEASPC**(-0.5)))
      TS=TS1+TS2+TS3
      HV1=FERFRC*(361.-0.357*700.+(50.*SI))
      HV2=175.*(1.-FERFRC)
      HV=HV1+HV2


C-----------------------------------------------------------------------------------
C   Evaluate non-linear constraints
C-----------------------------------------------------------------------------------
   IF (IPATH .EQ. 1 .OR. IPATH .EQ. 2) THEN
c
C     Minimum Yield Strength
     CONSTR(1)=(YS/220.)-1.
C
C     Maximum Yield Strength
     CONSTR(2)=1.-(YS/330.)
C
C     Minimum Tensile Strength
     CONSTR(3)=(TS/450.)-1.
C
C     Maximum Tensile Strength
     CONSTR(4)=1.-(TS/750.)
```

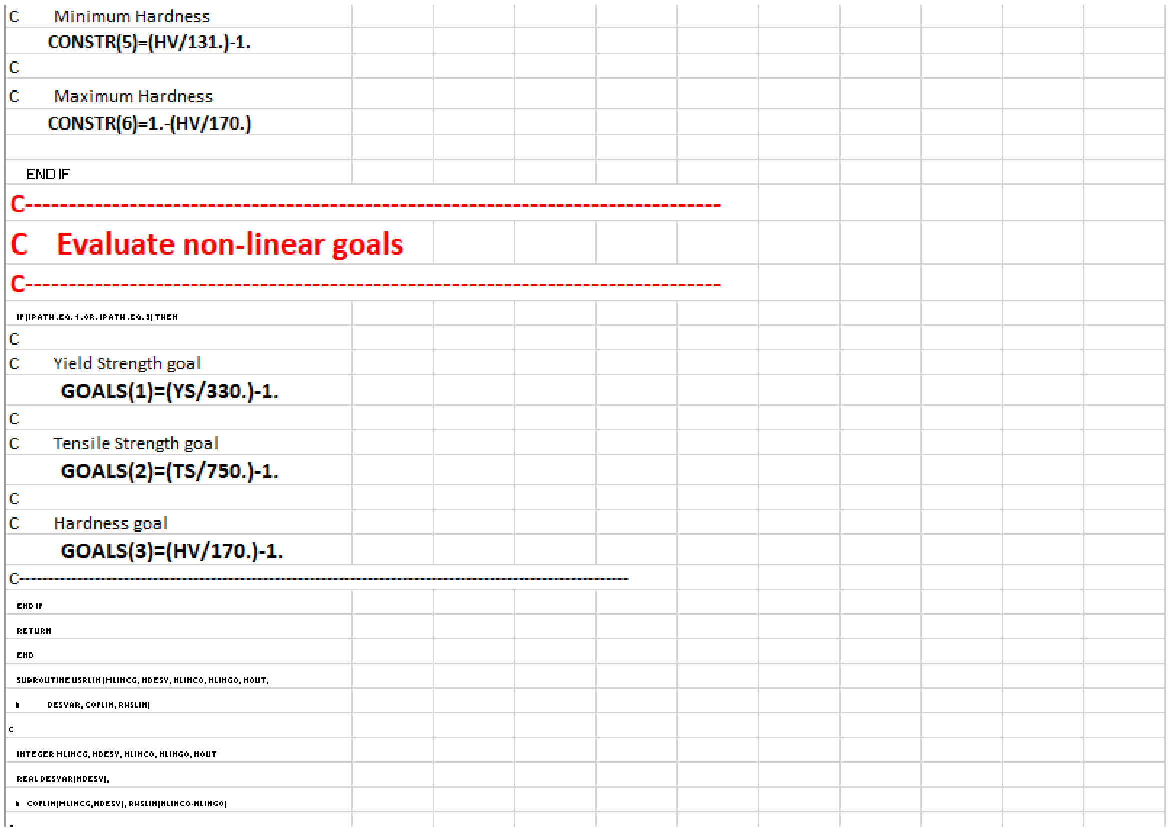
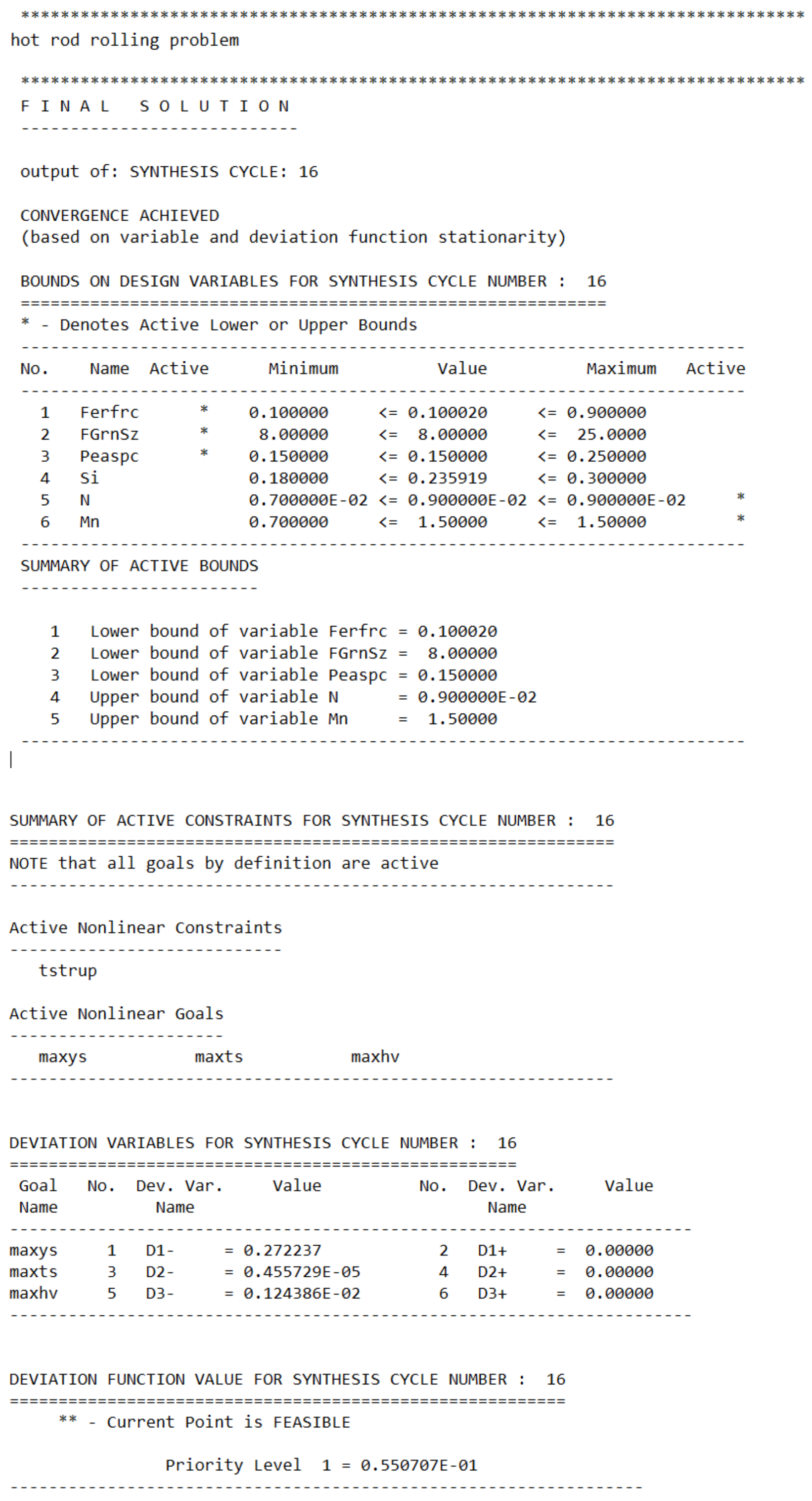

```
C      Minimum Hardness
      CONSTR(5)=(HV/131.)-1.
C
C      Maximum Hardness
      CONSTR(6)=1.-(HV/170.)

   END IF
C---------------------------------------------------------------------------------
C   Evaluate non-linear goals
C---------------------------------------------------------------------------------
  IF (IPATH.EQ.1.OR.IPATH.EQ.3) THEN
C
C      Yield Strength goal
        GOALS(1)=(YS/330.)-1.
C
C      Tensile Strength goal
        GOALS(2)=(TS/750.)-1.
C
C      Hardness goal
        GOALS(3)=(HV/170.)-1.
C-----------------------------------------------------------------------------------------------------
  END IF
  RETURN
  END
  SUBROUTINE USRLIN (NLINCG, NDESV, NLINCO, NLINGO, NOUT,
  &        DESVAR, COFLIN, RHSLIN)
C
  INTEGER NLINCG, NDESV, NLINCO, NLINGO, NOUT
  REAL DESVAR(NDESV),
  &  COFLIN(NLINCG,NDESV), RHSLIN(NLINCO-NLINGO)
```

Figure II. 35: Overview of Nonlinear sheet for hot rod rolling problem.

Final Result:

After the execution of the LinearSheet and NonlinearSheet, an output file will be generated in output_files. The output file containing the simulation results will appear as a pop-up on the screen, providing immediate access to the results. Furthermore, a copy of the output file will be saved in the output_file folder of the project, ensuring that the results are easily accessible and can be reviewed at a later time if required.

For the purpose of presenting the final results visually, an image capturing the contents of the output file has been attached in Figure II:36 . A Snapshot of the simulation outcomes for a quick overview of the obtained results has been represented in this figure. By having the output file readily available and with the attached image for reference, the obtained results of the cDSP model's simulation can be conveniently analyzed and interpreted.



```
 ********************************************************************************
hot rod rolling problem

 ********************************************************************************
 F I N A L   S O L U T I O N
 ----------------------------

 output of: SYNTHESIS CYCLE: 16

 CONVERGENCE ACHIEVED
 (based on variable and deviation function stationarity)

 BOUNDS ON DESIGN VARIABLES FOR SYNTHESIS CYCLE NUMBER :  16
 ============================================================
 * - Denotes Active Lower or Upper Bounds
 --------------------------------------------------------------------------
 No.    Name  Active      Minimum          Value          Maximum   Active
 --------------------------------------------------------------------------
   1   Ferfrc      *     0.100000     <= 0.100020     <= 0.900000
   2   FGrnSz      *      8.00000     <=  8.00000     <=  25.0000
   3   Peaspc      *     0.150000     <= 0.150000     <= 0.250000
   4   Si                0.180000     <= 0.235919     <= 0.300000
   5   N                 0.700000E-02 <= 0.900000E-02 <= 0.900000E-02     *
   6   Mn                0.700000     <=  1.50000     <=  1.50000         *
 --------------------------------------------------------------------------
 SUMMARY OF ACTIVE BOUNDS
 ------------------------

    1   Lower bound of variable Ferfrc = 0.100020
    2   Lower bound of variable FGrnSz =  8.00000
    3   Lower bound of variable Peaspc = 0.150000
    4   Upper bound of variable N      = 0.900000E-02
    5   Upper bound of variable Mn     =  1.50000
 --------------------------------------------------------------------------
|


SUMMARY OF ACTIVE CONSTRAINTS FOR SYNTHESIS CYCLE NUMBER :  16
==============================================================
NOTE that all goals by definition are active
--------------------------------------------------------------

Active Nonlinear Constraints
----------------------------
   tstrup

Active Nonlinear Goals
----------------------
   maxys           maxts           maxhv
--------------------------------------------------------------


DEVIATION VARIABLES FOR SYNTHESIS CYCLE NUMBER :  16
====================================================
 Goal    No.  Dev. Var.     Value          No.  Dev. Var.     Value
 Name          Name                              Name
----------------------------------------------------------------------
maxys     1   D1-      = 0.272237          2   D1+      =  0.00000
maxts     3   D2-      = 0.455729E-05      4   D2+      =  0.00000
maxhv     5   D3-      = 0.124386E-02      6   D3+      =  0.00000
----------------------------------------------------------------------


DEVIATION FUNCTION VALUE FOR SYNTHESIS CYCLE NUMBER :  16
=========================================================
     ** - Current Point is FEASIBLE

                Priority Level  1 = 0.550707E-01
-----------------------------------------------------------------
```

Figure 2:36 . Snapshot of the simulation outcomes in hot rod rolling problem

To accurately generate the ternary plot and identify the satisficing space, users should refer to the guidelines outlined in Section II.2.4 Ternary Plot and Section II.4.1.4 Find Satisficing Space Based on the Ternary Plot. The visual representations of the ternary plots and satisficing space have been provided in the paper, which users can utilize as a reference to validate their results and ensure they are following the correct steps for identifying the satisficing space accurately.

II.5 Summary and Way Forward

In Part 2, we have focused on designing a user-friendly wrapper for the DSIDES software. We have provided a comprehensive description of the wrapper, ensuring that all the necessary details are covered. The user manual created in this part is a valuable resource for users to understand and navigate the wrapper effectively.

Furthermore, to ensure the correctness and usability of our wrapper, three different examples in both preemptive and Archimedean forms have been presented. These examples are intended as verification and validation documents, allowing the functionality and user-friendliness of the wrapper to be assessed. Additionally, practical scenarios are offered to users, enabling them to apply their learnings from Part 1 and further enhance their understanding.

Moving forward, our thesis's next part(Part 3) is about developing a programmer's manual for DSIDES. This manual is helpful for programmers and technical users who require in-depth knowledge about the software. We provide program manuals, flowcharts, coding guidelines, and input/output subroutines that will improve user understanding and simplify comprehension.

One of the key objectives in part 3 is to consolidate all programs and subroutines in one place, creating a comprehensive documentation resource. Multiple purposes are served by this consolidation, including the facilitation of ease of access and reference, the promotion of better organization, and the enhancement of the overall clarity of the programming aspects of DSIDES.

By providing a programmer's manual, we aim to enhance the accessibility and usability of DSIDES, enabling programmers and technical users to utilize the software more efficiently and effectively. This manual will be a valuable asset for both the development team and users, ensuring a smooth and seamless experience with DSIDES.

Part Three: Program Manuals and Improvement of DSIDES

preamble about Part 3:

The main objective of creating Part Three is to provide in-depth material for users seeking to enhance or modify the code. The modification could include improving or changing only a single part or purpose of the program or considering changes in the whole program to another programming language. Since this software includes 90 subroutines, each with its specific role, understanding them becomes essential. Hence, I have compiled all these subroutines in a table, providing their names and specific purpose. With this setup, it is easier for users who want to modify or change specific parts to find the exact subroutine. Additionally, getting to know the different purposes of the subroutines is helpful to understand how the whole program works. Also, because there are five main input files in the program where these subroutines are used, users can understand these input subroutines better. I have also added flowcharts for two of the more complicated input subroutines that might be harder to grasp. Furthermore, I have included some observations based on my understanding after certain input subroutines, which could be beneficial for other users.

Overview of Part 3:

In the third part of this thesis, the programmer manuals, flowcharts, coding, and information about input subroutines in DSIDES are provided. The main focus of this section is on providing information about the input files and their respective purposes (Output files have been covered in Part two Section 2.3). The objective is to simplify their comprehension of the users by presenting their flowcharts. Additionally, other subroutines that are called through the main files will be cover in this part. This information is necessary for users wanting to improve the software and understand how the programs work. In this section, readers will have access to all the necessary information to improve the software and their work in the future. The focal objective of this section is to consolidate information about the programs and subroutines of DSIDES in a single location for documentation purposes. In the following section, the purposes of all subroutines in the program is outlined to assist users who wish to enhance the software. This will highlight the key aspects addressed by the program and identify specific subroutines that can be improved to cater to changes in particular domains. By providing this information, users will gain a deeper understanding of the program's functionalities and potential areas for refinement, enabling them to make informed decisions when customizing the software to suit their specific needs and requirements.

III.1 Overview of DSIDES Program's Nested Subroutines and their Versatility

The DSIDES program has a complete set of smaller routines, five of them are considered as main input files. Within each of these core subroutines /functions , a hierarchical structure of additional subroutines exists, each serving specific purposes as outlined in Table III.1. These nested subroutines/functions have been designed to fulfill various specialized functions, contributing to the overall effectiveness and versatility of the DSIDES program.

A compilation of 90 subroutines and their respective main purposes has been presented in Table III.1. The aim of this compilation is to provide users with a cohesive overview of each subroutine, thereby facilitating a better understanding of their functionalities.

Table III.1: List of Subroutines and Functions in the DSIDES Program

Name of Subroutine / FUNCTION	**Main Purpose**
1) ALPLIM	Set the limits of the arrays.

2) ALPDAT	This routine reads the ALP data file for compromise DSPs and sets the necessary defaults.
3) ALPMOD	Perform the synthesis cycles.
4) COPYTX(ALPUTL)	This routine copies the contents of input vector array into the output vector array.
5) ALPCTL	A main program for DSIDES.
6) ADREMO	Adaptive reduced move using a Golden Section line search. If no improvement is found in the initial design, a fixed reduced move is applied to the design point.
7) COEFF2	Determine coefficients of linearized constraint using second-order terms.
8) CONACC	Constraint accumulation of nonlinear inequality constraints and "NEW" constraints as generated via adaptation of the nonlinear constraints.
9) CONCHK	Determine and record which nonlinear constraints are to be suppressed. Also, determines the right-hand side value of linearized constraints.
10) CONCOR	Subroutine CONCOR performs the adaption of the nonlinear constraints.
11) CONLIN	This subroutine performs the linearization of constraints and goals. (The deviation variables are not considered here.)
12) CONVER	Check for convergence. This is done in several steps.
13) DERIV	Calculate gradients of nonlinear constraints and goals.
14) DEVCAL	This subroutine calculates the values of the deviation variables.
15) DFCALC	Calculate the deviation function for a new set of system variables.
16) DISCRT	Solves the discrete problem.
17) EXPCTL	Read ALP postprocessor files (control and information) and generate a tab-delimited file, ALPEXP.DAT, suitable for export to a spreadsheet.
18) FORAGEMV	Solves the discrete problem.
19) GCALC	This subroutine calculates the values of the nonlinear constraints and goals for the current set of design variables. The nonlinear constraints are evaluated in one block, and the goals are evaluated in another.
20) HJMAIN	This routine performs pattern search using the Hook and Jeeves algorithm.
21) INITAB	Initialize constant portions of Simplex Tableau.
22) INITSL	Find an initial feasible solution using the Hook-Jeeves pattern search algorithm.
23) MLINOP	Solve the L.P. problem using MULTIPLEX, recording the basis to be used in the next synthesis cycle. The algorithm has integer (branch and bound variation) capability to handle 0/1 "selection" variables.

24) TRFORM	Performs the required transformation in the tableau (AMAT) to account for (negative) RHS and (nonzero) lower bounds on the system variables.
25) MSETUP	Initial basis setup. MSETUP is called with LITK = 1; that is the initializations are performed for the first priority level.
26) ZERONE	The purpose of this routine is to modify the earlier solution such that the selection variables are forced to go to zero/one values. This is in case they deviate from the zero/one solution in the earlier phase (this may happen due to the influence of constraints).
27) MPLEX	PRIMAL MULTIPLEX: Algorithm by J.P. Ignizio
28) CHKBXB	This function checks if a reinversion of the basis is required. It returns the absolute value of the largest error.
29) CALCXB	This function calculates the basic vector XB according to XB= [BINV][B] - [BINV][Ns][Vs]. XB = MULTIPLEX vector corresponding to the vector formed by multiplying the old RHS by the current inverse. XB is also referred to as Beta. BINV = MULTIPLEX inverse matrix of the present basis. B = MULTIPLEX "b" vector of RHS values.
30) ECOMP	Calculate the deviation function priority levels NP1 to NP2 for variables NV1 to NV2. NP1 = MULTIPLEX number of starting priorities. NP2 = MULTIPLEX number of ending priority level. NV1 = MULTIPLEX number of starting variables. NV2 = MULTIPLEX number of ending variable.
31) PRICEV	Subroutine PRICEV calculates the pricing vector PIVEC according to the formula [PI] = [CB][BINV] (For details see "Introduction to Linear Goal Programming" by J.P. Ignizio, SAGE Publications, 1985, page 40.)
32) RDCOST	1 - The subroutine RDCOST calculates the vector DJ, which is known as the shadow price or reduced cost. 2- The subroutine RDCOST determines the entering nonbasic variable based on the highest absolute value of Dj.
33) ENTCOL	Update the column [aq] for entering the nonbasic variable IQ. The resulting column vector is ALPHA. The formula is [Alphaq] = [Binv][aq] IQ = MULTIPLEX variable denoting the ALPHA = MULTIPLEX vector corresponding to the updated entering column for IQ. BINV = MULTIPLEX inverse matrix of the present basis.
34) RTEST	Ratio test, which determines the leaving variable.

35) UPDBAS	Updates the basis when there is a leaving variable IP and entering variable IQ.
36) NEWRHS	The subroutine updates the XB vector for the case where the entering variable does not enter the basis but is set to one of its bounds.
37) PFRINV	Compute new basis inverse according to the product form of the inverse method.
38) SEIGHT	Step 8 of the LGP algorithm.
39) REINVR	Subroutine for reinversion. The current basis matrix is identified, and then the call to the inversion subroutine is made. Variables are used in DOUBLE PRECISION in the latter.
40) INVERS	I do not have information about its purpose.
41) LUDCMP	I do not have information about its purpose.
42) LUBKSB	I do not have information about its purpose.
43) PARCMP	This routine determines the relative importance of attributes using the reciprocal pairwise comparison matrix method. (Reference: Saaty, T. L., "A Scaling Method for Priorities in Hierarchical Structures", J. of Mathematical psychology, Vol. 15, 1977, pp. 234-281.C)
44) EIGEN	Calculate the maximum eigenvalue and the corresponding eigenvector of a matrix using iteration. (Reference: Shoup, T. E., "Applied Numerical Methods for the Personal Computer", Prentice-Hall, 1984.)
45) MATMUL	Matrix multiplication B = A*X B = real vector of length of the number of variables A = number of variables matrix X = real vector of length number of variables
46) PCIMPR	This routine is called when the consistency ratio is poor. It provides some additional information regarding each of the decisions made by the user.
47) PRMULT	Print the linear solution constraint and goal multipliers (Lagrangians).
48) PRPPCF	Create a postprocessor control data file.
49) PRPPIF	Create a postprocessor information data file.
50) PRDESV	Print design variables.
51) PRDEVV	Print deviation variables.
52) PRDFUN	Print values of current deviation function.
53) PRBNDS	Print bound activity information for a linear solution.
54) PRLINC	Print information on linear constraints and goals.
55) PRNLNC	Print nonlinear constraint/goal information.
56) PRLINA	Print linear constraint activity for a linear solution.
57) PRNLNA	Print nonlinear constraint activity for a linear solution.
58) PRADPC	Print adapted constraint information.

59) PRFACT	Print final solution linear and nonlinear constraint activity.
60) SELCAL	Perform the selection DSP calculations - Normalization of ratings - Calculation of merit function
61) SELCTL	Perform the selection DSP.
62) SELCYC	This routine determines the relative importance of attributes using the reciprocal pairwise comparison matrix method.
63) SELDAT	This routine reads the data file for selection DSPs and sets the necessary defaults.
64) SELHLO	This routine determines the relative importance of attributes using the reciprocal pairwise comparison matrix method.
65) SELMER	Provide an interface to the Selection Routines.
66) SELPRT	Selection DSP printing
67) SELSEN	Perform the sensitivity analysis for selection Decision Support problems using interval arithmetic.
68) USRSET	Evaluate nonlinear constraints and goals.
69) TIMER	This routine contains the timer routines for: Sun/O.S. systems.
70) UPDTAB	Update multiobjective goal formulation tableau arrays.
71) IQSIGN	It is a FUNCTION. This routine returns an integer flag for the inequality sign
72) MKNAME	It is a FUNCTION. This routine returns a name by appending an integer to a character string.
73) SPLITC	Splits a complex number into an integer and a real value.
74) BLKCHK	Splits a complex number into an integer and a real value.
75) IARFIL	Fill the IADCON and IACTVR arrays.
76) GETNAM	This routine allows for names to be read from data file when the text line has leading blanks.
77) LEXSML	Perform lexicographic comparison of two vectors.
78) LVCEQU	It is a FUNCTION. Compare the two vectors for convergence.
79) VECSUM	It is a FUNCTION. To return the absolute norm of a vector.
80) VECMAX	It is a FUNCTION. To return the maximum absolute value from a vector.
81) EXPTAB	Export Table printing.
82) USRANA	for user provided analysis routine.
83) USROUT	user provided output routine.
84) XPLORE	Find initial feasible solutions.
85) PTSRCH	Return a vector of length N in an N-dimensional cube.
86) ZSRCH	To generate K points in an N dimensional space.
87) PTRAND	Return a random vector of length N in an N-dimensional cube.
88) SRAN	Return a pseudo-random number between 0.0 and 1.0
89) ADNUPT	This routine calculates the design point given by REMO Only the real variables are perturbed.

90) USRMON	Monitor process .

Learning about all subroutines and their purpose could help the user to have a better understanding of how a program is working. In the next section five main input files are described in more detail with their program and show how other subroutines are used in these five main input files.

Obtaining knowledge about all the subroutines and their purposes is crucial for users to gain a comprehensive understanding of the DSIDES program's functionality in the subsequent section of this document, in-depth descriptions of the five main input files are provided, offering valuable insights into their respective programs and demonstrating the seamless integration of various subroutines. By exploring these detailed explanations, users can enhance their comprehension of the program's inner workings and the strategic utilization of subroutines within the context of these five main input files.

III.2 Main Input Files

The Fortran program includes five main input files, each with its principle. Before delving into the specifics of each subroutine, we will introduce the main objective of each one:

1) **ALPLIM:** Its purpose is to Consider the limit for the arrays.
2) **ALPDAT:** The ALP data file for compromise DSPs is read by this routine, and the necessary defaults are set.
3) **ALPMOD:** Its purpose is to perform the synthesis cycles. (it's about stopping criteria)
4) **COPYTX (ALPUTL):** Its purpose is to copy the contents of the Y (input vector) array into the X (output vector) array. This subroutine is about the synthesis cycle. So, it keeps the information of the last cycle and uses it to start the next cycle.

5) ALPCTL: A main program for DSIDES that other subroutines will call here.

Following the introduction of the five main input files, the subsequent subsections will delve into the details of each mentioned input subroutine, providing further clarification and understanding.

III.2.1 ALPLIM Subroutine

One of the main input subroutines is ALPLIM. In the ALPLIM subroutine, the limit for arrays has been determined. It has a crucial role in other input subroutines. If a user does not specify initial values for variables, the program will rely on ALPLIM to set default initial values. In simpler terms, ALPLIM is like a helper function that sets default values for variables when needed. In this subroutine, the important variables, their definition, and their initial values are identified. I have included the subroutine here, which simply assigns initial values to variables. I did not add a flowchart since the process is straightforward..

```
C                  <<<<<<<<<*>>>>>>>>>
C                     ALPLIM.CMM
C                  <<<<<<<<<*>>>>>>>>>
C    Last update: 28MAR92
C**********************************************************************
C    Set the limits of the arrays.
C**********************************************************************
C    Compromise DSP limits:
     INTEGER MDESV, MDSCV, MLINCG, MNLNCG, MAXACC
     INTEGER MAXCAG, MDEVV, MVARMX, MGOLMX
     INTEGER MLEVEL, MNSYCY, MNANCY, MTITER
C    Selection DSP limits:
     INTEGER MATTRB, MSELPR, MPAIRS
C
C    Module XPLORE limit:
C
```

```
      INTEGER MPTBST
C     Threshold values:
C
      REAL SMALL, THDIFF, TINY, VLINCO
C
C New name            Description
C --------            -----------
C MDESV     Max. number of design variables
C
C MDSCV     Max. number of discrete variables values
C
C MLINCG    Max. number of linear constraints + goals
C
C MNLNCG    Max. number of nonlinear constraints + goals
C
C MAXACC    Max. number of accumulated constraints
C
C MAXCAG    Max. number of constraints + goals ( MAXCAG = MLINCG + MNLNCG)
C
C MDEVV     Max. number of deviation variables ( MDEVV = 2*MAXCAG)
C
C MVARMX    Max. number of variables for Multiplex ( MVARMX = MDESV + 2*(2*MAXCAG +
MAXACC) )
C
C MGOLMX    Max. number of goals for multiplex ( MGOLMX = MAXCAG + MAXACC + maxnew=
                                            2*MAXCAG + MAXACC)
C

C               (NOTE: bounds not included)
C
C MLEVEL    Max. number of priority levels including
C          constraint level
C MNSYCY    MAX. number of synthesis cycles
C
C MNANCY    MAX. number of analysis cycles
C
C MTITER    MAX. number of total iterations (MTITER = MNSYCY * MNANCY)
C          Currently set to 100
C
      PARAMETER   (MDESV  = 40)
      PARAMETER   (MDSCV  = 50)
      PARAMETER   (MLINCG = 50)
      PARAMETER   (MNLNCG = 50)
      PARAMETER   (MAXACC = 20)
      PARAMETER   (MAXCAG = 100)
      PARAMETER   (MDEVV  = 200)
CCCCC      ( Based on the formulation MVARMX should be 480 but, in the program, sets as a
640, since they set the range larger than needed. )
      PARAMETER   (MVARMX = 640)
      PARAMETER   (MGOLMX = 220)
C
C     Limits increased from preceding values by Bert Bras, 10 August 1993. ( Even the limits
increased in 1993 but still the previous amount in used in the subroutine)
CCCCC      PARAMETER   (MDESV  = 200)
CCCCC      PARAMETER   (MLINCG = 50)
CCCCC      PARAMETER   (MNLNCG = 450)
CCCCC      PARAMETER   (MAXACC = 150)
CCCCC      PARAMETER   (MAXCAG = 500)
CCCCC      PARAMETER   (MDEVV  = 1000)
CCCCC      PARAMETER   (MVARMX = 2500)
CCCCC      PARAMETER   (MGOLMX = 1150)

      PARAMETER   (MLEVEL = 9)
      PARAMETER   (MNSYCY = 50)
      PARAMETER   (MNANCY = 25)
      PARAMETER   (MTITER = 100)
C
C MATTRB    MAX. number of attributes
C
C MSELPR    MAX. number of selection problems
C
C MPAIRS    MAX. number of pairs for pairwise comparison
C           MPAIRS = MATTRB * (MATTRB-1)/2

      PARAMETER   (MATTRB = 20)
      PARAMETER   (MSELPR = 5)
      PARAMETER   (MPAIRS = 190)
C
C MPTBST    MAX. number of points (best) saved within
C           module XPLORE
C
CCCCC      PARAMETER (MPTBST = 200)
C
      PARAMETER (MPTBST = 1000)
C
C SMALL     Threshold for activity of constraints and  variable bounds
C
C THDIFF    Threshold for general differences
```

```
C
C TINY      Threshold to avoid underflow
C
C VLINCO    Magnitude of small violation as a function of the RHS  in linear constraints to be
ignored          Default magnitude of small violations ignored in non- linear constraints
C
     PARAMETER (SMALL  = 1.0E-4)
     PARAMETER (THDIFF = 1.0E-5)
     PARAMETER (TINY   = 1.0E-10)
     PARAMETER (VLINCO = 1.0E-3)
```

In the subsequent section, an additional key input subroutine known as ALPDAT is introduced.

III.2.2 ALPDAT Subroutine

The main purpose of this subroutine is to read the ALP data file for compromise DSPs and set the necessary defaults. This subroutine is about all blocks that we have in LinearArchimedian and LinearPreemptive sheet, as they are represented in Table II.1 and Table II.2 in Part Two, there are 8 mandatory blocks and 27 optional blocks which are described in detail in Part Two, Sections II.2.1 and II.2.2.

As it is represented in the two mentioned tables, we have the following blocks in the linear sheets :

1) PTITLE: Problem title
2) NUMSYS: Number of System Variables
3) SYSVAR : Description of System Variables - name, type, bounds, and guess value
4) NUMCAG: Number of Constraints and Goals
5) LINCON : Linear Constraints - names and data (if specified in NUMCAG)
6) LINGOL : Linear Goals- names and data (if specified in NUMCAG)
7) DEVFUN: Deviation Function - number of levels and weights of deviation variables
8) STOPCR : Stopping Criteria (run and principal print flags, NITER, EPSZ, EPSX)
9) NLINCO : Names of Nonlinear Constraints (default names: NLCO##)

10) NL I NGO:	Names of Nonlinear Goals (default names: NLGO##)
11) ALP OUT:	Flags for Output Level, Post Processor, and Time Statistics
12) USRMOD:	Flags for User Modules (USRINP, USROUT, USRMON, USRLIN)
13) OPTIMP:	Optimization Parameters (VIOLIM, REMO, STEP)
14) INITFS:	Automatic Generation of Initial Feasible Solution
15) USRDAT:	User Data Block for Access From USRINP
16) ADPCTL :	Nonlinear Inequality Constraint Adaption Flag (LADAP)
17) USERAN:	Information for USRANA (maximum cycles - NANCY, NSYCY)
18) FIXVAR:	Fixing of Variables
19) SUPCON:	Suppression of Nonlinear Constraints
20) PVALFX :	Particular Values for Stationarity of System Variables
21) PVEPSZ :	Particular Values for Stationarity of Deviation Function Levels
22) PVSTEP :	Particular Values for STEP
23) PVCVIL:	Particular Values for VIOLIM
24) PVREMO:	Particular Values for REMO
25) ADREMO:	Adaptive Reduced Move Parameters
26) XPLORE :	Explore the design space for best initial points.
27) ENDPRB:	End of Problem Definition

Subroutine ALPDAT will go through all the blocks, and if the user defines the value for each block, it will assign it to the required variables; otherwise, will use the default value that has been assigned with the ALPLIM subroutine that has been described in section III.1.1. So ALPLIM is included in this subroutine as well. All the variables have been defined at the beginning of the subroutine in different types like single integers or real numbers, constants, matrices, or arrays. The ALPLIM subroutine has been called to assign the default value if the user did not define the value to some variables and wants to use the default value. Then the program will go through block 1 to block 27 with an if statement; if the if statement is true, the program will execute the specific block of code.

The ALPDAT subroutine is added Below. Several other subroutines are called in ALPDAT, such as IQSIGN, MKNAME, SPLITC, BLKCHK, and IARFIL, which have been introduced in Section III.1.

```
C*********************************************************************
**
C
C Subroutine ALPDAT
C
C Purpose:  This routine reads the ALP data file for
C compromises DSPs and sets the necessary defaults.
C
C---------------------------------------------------------------------
--
C Arguments         Name     Type   Description
C ---------         ----     ----   -----------
C Input:            NUINP    int    unit number of the input data file
C                   NUOUT    int    unit number of the output data file
C                   NUSER    int    unit number of the scratch user data
file
C
C Output:           NDESV    int    number of design variables
C                   NDEVAR   int    number of deviation variables
C                   NRELV    int    number of real (continuous)
variables
C ************KL NDISV    int    number of discrete variables (inc.
integer)
C              DSTEP    int/real step for discrete variables (1
default)
C                   NVALUS   int   number of discrete values (override
step)
C                              MAX = 50
C              NUMNGH   int number of variables to have tabu Nghbrhd
C              INDEX    int index of initial discrete vars (for tabu)
C              NEIGH    int counter to set up tabu with DSTEP = 1.0
C              TABUN    real   tabu neighborhood (MDESV, 50)
C                   NVINT    int    number of integer variables
C                   NVSEL    int    number of selection (boolean)
C                                   variables
C                   NMPRI    int    number of goal priority levels
C                   NLINCO   int    number of linear constraints
C                   NLINGO   int    number of linear goals
C                   NNLEQU   int    number of nonlinear equalities
C                                   constraints
```

```
C                  NNLINQ   int    number of nonlinear inequalities
C                                  constraints
C                  NNLCON   int    number of nonlinear constraints
C                  NNLGOA   int    number of nonlinear goals
C                  NNLTOT   int    total number of nonlinear
constraints
C                                  and goals
C                                  = NNLINQ+NNLEQU+NNLGOA
C                  NANCY    int    maximum number of analysis cycles
C                                  performed
C                  NHJMAX   int    maximum calls to USRSET from INITSL
C                  NSYCY    int    vector indicating the maximum number
C                                  of synthesis cycles performed within
C                                  each analysis cycle
C                  NADREM   int    ADREMO - maximum number of calls to
C                                  be made to GCALC.
C                  NPTGEN   int    number of points to be generated.
C                  NPTBST   int    number of best points to be printed.
C                  IGSEED   int    seed for the pseudo-random number
generator
C                  IGENFX   int    vector of variables to be kept
fixed.
C                                  during point generation
C                  IACTVR   int    vector of flags for active variables
C                  IADCON   int    vector of flags for admissible
C                                  nonlinear constraints
C                                  = 0  if suppressed by the user
C                                  = J  if admissible
C                                  = -n if suppressed by the program
C                  IDDEVR   chr6   vector of deviation variable names
C                  IDLICO   chr6   vector of linear constraint names
C                  IDLIGO   chr6   vector of linear goal names
C                  IDNLCO   chr6   vector of nonlinear constraint names
C                  IDNLGO   chr6   vector of nonlinear goal names
C                  IDDESV   chr6   vector of design variable names
C                  PTITLE   chr80  problem title
C                  COFLIN   real   matrix of coefficients of linear
C                                  constraints and goals
C                  LISIGN   int    vector indicating sign for linear
C                                  inequality constraints
C                  RHSLIN   real   vector of RHS values for linear
C                                  constraints and goals
C                  PESTEP   real   vector of perturbation steps for
C                                  nonlinear constraints and goals
C                  REDMOV   real   reduced move step sizes
C                  VBOUNS   real   matrix of lower and upper bounds for
```

```
C                                design variables
C                 DESVAR   real   vector of design variables
C                 DFNCOF   real   matrix of weights for deviation
C                                function
C                 DELREM   real   acceptable convergence criterion for
C                                reduced move
C                 HJCONT   real   pattern search - contraction factor
C                 HJDELT   real   pattern search - minimum allowable
C                                step
C                 HJEPSY   real   pattern search - maximum allowable
C                                violation
C                 HJEXPA   real   pattern search - expansion factor
C                 HJSTEP   real   pattern search - step size
C                 FRACX    real   vector: convergence criteria for X,
C                                the design variables
C                 FRACZ    real   vector: convergence criteria for Z,
C                                the deviation function
C                 VILCN    real   vector of acceptable nonlinear
C                                constraint violations
C                 LDRYRN   lgcl   effect dry run only
C                 LPRFIN   lgcl   print final output only
C                 LPROUT   lgcl   vector of flags for output control
C                 LPPROC   lgcl   create ALP postprocessor files
C                 LADAP    lgcl   use constraint adaptation
C                 LADREM   lgcl   use adaptive reduced move
C                 LINIT    lgcl   generate initial feasible solution
C                 LTIME    lgcl   generate time statistics
C                 LMON     lgcl   call user supplied subroutine USRMON
C                 LUINP    lgcl   call user supplied subroutine USRINP
C                 LUOUT    lgcl   call user supplied subroutine USROUT
C                 LVCOF    lgcl   call user supplied subroutine USRLIN
C                 LXPLOR   lgcl   explore design space
C                 LPRGEN   lgcl   print the best points generated in
C                                exploring design space
C                 FATAL    lgcl   failure status in ALPDAT
C
C Input/output:   none
C----------------------------------------------------------------------
--
C Common Blocks:  none
C
C Include Files:  alplim.cmm
C
C Calls to:       IQSIGN, MKNAME, SPLITC, BLKCHK, IARFIL
C----------------------------------------------------------------------
--
```

```
C Development History
C
C Author:  Ravi P. Reddy
C Date:    February 14, 1991
C
C Modifications:
C          Bert Bras, July 1993.
C            The same default and user-specified values for FRACX are
C            used for ALL variables (real and Boolean).
C            Prior to this modification, FRACX was set to 1.0 for
nonreal
C            variables. This caused the (Boolean) variable convergence
C            check in CONVER() to be incorrectly satisfied in all
cases.
C
C**********************************************************************
**
      SUBROUTINE ALPDAT(NUOUT, NUINP, NUSER,
     &     NRELV, NDISV, NVINT, TABUN, NVALUS, INDEX, NVSEL,
     &     NDESV, NDEVAR,
     &     NLINCO, NLINGO, NMPRI,
     &     NNLCON, NNLEQU, NNLGOA, NNLINQ, NNLTOT,
     &     NANCY, NSYCY,
     &     NHJMAX, NADREM, IACTVR, IADCON, LISIGN,
     &     NPTGEN, NPTBST, IGSEED, IGENFX,
     &     IDDESV, IDDEVR, IDLICO, IDLIGO, IDNLCO, IDNLGO,
     &     PTITLE, COFLIN, RHSLIN, DFNCOF,
     &     PESTEP, REDMOV, VBOUNS, DESVAR,
     &     HJEXPA, HJCONT, HJSTEP, HJEPSY, HJDELT, DELREM,
     &     FRACZ, FRACX, VILCN,
     &     FATAL, LDRYRN, LPRFIN,
     &     LPROUT, LPPROC, LTIME, LADREM, LADAP, LINIT,
     &     LMON, LUINP, LUOUT, LVCOF, LXPLOR, LPRGEN, LVDISC)
C
      INCLUDE 'alplim.cmm'
C
C----------------------------------------
C     Arguments:
C----------------------------------------
C     Logical Unit numbers for I/O
C
      INTEGER NUOUT, NUINP, NUSER
C
C
      INTEGER NRELV, NDISV, NVALUS(MDESV), DSTEP, INDEX(MDESV),
NUMNGH,
     &        NVINT, NVSEL, NDESV, NDEVAR,
     &     NLINCO, NLINGO, NMPRI,
     &     NNLCON, NNLEQU, NNLGOA, NNLINQ, NNLTOT,
     &     NANCY, NSYCY(MNANCY),
     &     NHJMAX, NADREM, IACTVR(MDESV), IADCON(MNLNCG),
     &     LISIGN(MLINCG),
     &     NPTGEN, NPTBST, IGENFX(MDESV), IGSEED
C
      CHARACTER*6 IDDESV(MDESV), IDDEVR(MDEVV),
     &     IDLICO(MLINCG), IDLIGO(MLINCG),
     &     IDNLCO(MNLNCG), IDNLGO(MNLNCG)
C
      CHARACTER*80 PTITLE(2)
C
      REAL COFLIN(MLINCG,MDESV), RHSLIN(MLINCG),
     &     DFNCOF(MLEVEL,MDEVV),
     &     PESTEP(MDESV), REDMOV(MDESV),
     &     VBOUNS(2,MDESV), DESVAR(MDESV),
     &     HJEXPA, HJCONT, HJSTEP, HJEPSY, HJDELT, DELREM,
     &     FRACZ(MLEVEL), FRACX(MDESV), VILCN(MNLNCG),
     &     TABUN(MDESV,MDSCV)
C
      LOGICAL FATAL,
     &     LDRYRN, LPRFIN,
     &     LPROUT(8), LPPROC, LTIME,
     &     LADREM, LADAP, LINIT,
     &     LMON, LUINP, LUOUT, LVCOF,
     &     LPRGEN, LXPLOR, LVDISC
C
C---------------------------------------
C     Local variables:
C---------------------------------------
C
      CHARACTER*6 MKNAME
      EXTERNAL MKNAME
      INTEGER IQSIGN
C
      CHARACTER*80 DUM, DUMOUT
      CHARACTER*6 BLKNAM,DUMNAM
      CHARACTER*1 VTYPE,ONE
      CHARACTER*2 CCSIGN
C
      LOGICAL BKIN(30), LNONAM
C
C     Modification: Following line was COMPLEX CIR(50)
C
```

```
      COMPLEX CIR(MDESV)
C
      INTEGER JDUM(MDESV), J1, J2, IPR(9), KONT
      INTEGER NUM, IVAR, IOUSER, IOUT
      INTEGER IMON, JVARC, ML, IADAP, II
      INTEGER NPSTEP, NREMOV, NFRAC
      INTEGER KDUM, NDUM, NTERMS, NLEVEL
      INTEGER KMIN, KMAX, KGES, KSGN, I, J, K, L, KST
C
      REAL REMOVS, PSTEP, VIOLIM, NEIGH
      REAL VAL, EPSZ, EPSX
      REAL XMIN, XMAX, XGES
C
C**********************************************************************
**
C     Initialize Logical Flags
C**********************************************************************
**
C
      LUINP  = .FALSE.
      LUOUT  = .FALSE.
      LVCOF  = .FALSE.
      LTIME  = .FALSE.
      LINIT  = .FALSE.
      LADREM = .FALSE.
      LADAP  = .FALSE.
      LMON   = .FALSE.
      LXPLOR = .FALSE.
      LVDISC = .FALSE.
      LPRGEN = .FALSE.
      FATAL  = .FALSE.
C
      LPROUT(1) = .TRUE.
      LPROUT(2) = .TRUE.
      LPROUT(3) = .TRUE.
      LPROUT(4) = .FALSE.
      LPROUT(5) = .FALSE.
      LPROUT(6) = .FALSE.
      LPROUT(7) = .FALSE.
      LPROUT(8) = .FALSE.
C
C     Set other defaults
C
      NANCY = 0
      REMOVS = 0.5
      PSTEP = 0.005
      VIOLIM = -VLINCO
C-------------------------------------------------------
C     No blocks have been read.
C
      D.O. 2 J=1,30
        BKIN(J)=.FALSE.
  2   CONTINUE
C
C     Beginning of main GOTO loop
C     Read next block name
C
 1111 READ(NUINP,FMT='(A)',END=9000,ERR=8888)DUM
      CALL GETNAM(DUM, BLKNAM, KST, LNONAM)
C
      IF(LNONAM)THEN
        GO TO 1111
      ENDIF
C
      WRITE(NUOUT,3)BLKNAM
    3 FORMAT(/,X,'BLOCK ',A6,X,54('-'),/)
C
CC BLOCK1 ------- PTITLE --------------------------------------------
C     Read Problem Title & User Information
C
      IF(BLKNAM.EQ.'PTITLE')THEN
        CALL BLKCHK(NUOUT,BKIN(1))
C
        DO 4 J = 1,2
          READ(NUINP,FMT='(A)',ERR=8888) PTITLE(J)
          WRITE (NUOUT, '(A)') PTITLE(J)
    4   CONTINUE
C
        GO TO 1111
      ENDIF
C
CC BLOCK2 ------- NUMSYS --------------------------------------------
C ***KL
C     Read number of design variables - Real, and dicrete
C
C     NRELV = number of real variables
C     NDISV = number of discrete variables
C     NVINT = number of integer variables
C     NVSEL = number of selection (boolean) variables
C     NDESV = number of standard variables
C           = NRELV+NDISV
C -------------------------------------------------------------------
```

```
C                      Programmer's Note
C                      -----------------
C     The current version of the program does not have the capability
C     of handling INTEGER variables. If and when this aspect of the
C     program is developed, the variable NVINT should be made active
C     in the READ statement and the echo FORMAT.
C     Remember that the integer variables must follow the real
C     variables and precede the selection variables.

C ------------------------------------------------------------------
C
      IF(BLKNAM.EQ.'NUMSYS')THEN
        CALL BLKCHK(NUOUT,BKIN(2))
C
        NVINT = 0
        READ(NUINP,FMT=*,ERR=8888)NRELV,NDISV,NVSEL
C
     IF (NDISV.GT.0) THEN
           LVDISC = .TRUE.
     ENDIF

        NDESV = NRELV+NDISV+NVSEL
        WRITE(NUOUT,20)NRELV,NDISV,NVSEL,NDESV
   20   FORMAT(3X,' Number of real variables                 = ', I5/,
     &         3X,' Number of discrete variables             = ', I5/,
     &         3X,' Number of selection (Boolean) variables = ', I5/,
     &         3X,' Total number of design variables         = ', I5)
C
        IF(NDESV.GT.MDESV)THEN
          WRITE(NUOUT,21)MDESV,NDESV
          GO TO 9999
        ENDIF
   21   FORMAT('  E ** Problem Size',/
     &  3X,' Maximum number of design variables = ',I5,/
     &  3X,'                          Specified = ',I5)
C
        GO TO 1111
      ENDIF
C
CC BLOCK3 ------- SYSVAR ---------------------------------------------
C     Read design variable information
C
      IF(BLKNAM.EQ.'SYSVAR')THEN
        CALL BLKCHK(NUOUT,BKIN(3))
C
C       Check if NUMSYS has been read.
```

```
         IF(.NOT.BKIN(2))THEN
           WRITE(NUOUT,700)
           GO TO 9999
         ENDIF
C
         WRITE(NUOUT,30)
   30    FORMAT(2X,'  Number   Name   Type    Minimum        Maximum
',
     &          3X,'Guess Value',/,
     &          2X,'  ------  ------  ----  ------------   ------------
',
     &          3X,'------------')
C
         DO 35 K=1,NDESV
           READ(NUINP,'(A)',ERR=8888)DUM
           CALL GETNAM(DUM, DUMNAM, KST, LNONAM)
           IF (LNONAM) DUMNAM = MKNAME('X',K)
C
           READ(DUM(KST:80),FMT=*,ERR=8888)J,XMIN,XMAX,XGES
C
C          Variable type assigned using serial number
C
C **K.L. changed this to NDISV and 'D'
           IF(J.LE.NRELV)THEN
             VTYPE='R'
           ELSEIF(J.GT.(NRELV+NDISV))THEN
             VTYPE='B'
           ELSE
             VTYPE='D'
           ENDIF
C
C          Check if values are within bounds
C
           IF((XGES.LT.XMIN).OR.(XGES.GT.XMAX))THEN
             XGES=0.5*(XMIN+XMAX)
             WRITE(NUOUT,32)J
   32        FORMAT(/,' I ** Guess value out of bounds, ',
     &              'reset to (XMIN+XMAX)/2 for variable number ',I3)
           ENDIF
C
           IDDESV(J)    = DUMNAM
           VBOUNS(1,J)  = XMIN
           VBOUNS(2,J)  = XMAX
           DESVAR(J)    = XGES
C
C     THIS SETS up the tabu neighborhood from the bounds.
```

```
          DSTEP = 1

          IF (J.GT.NRELV.AND.J.LT.(NRELV+NDISV+1)) THEN
             NVALUS(J) = VBOUNS(2,J) - VBOUNS(1,J) + 1

C     This sets the index number as the guess index.
             INDEX(J)=(DESVAR(J)-VBOUNS(1,J))/DSTEP + 1

C     SETS up the temporary counter for set of discrete values.
C     STARTS at the lower bound.

             NEIGH = VBOUNS(1,J)

             DO 37 L=1,NVALUS(J)
                TABUN(J,L) = NEIGH
                NEIGH = NEIGH + DSTEP
 37          CONTINUE
          ENDIF
C
C         Use different formats for REAL and other variables.
C
          IF(VTYPE.EQ.'R')THEN
            WRITE(NUOUT,33)J,IDDESV(J),VTYPE,VBOUNS(1,J),
     &                     VBOUNS(2,J),DESVAR(J)
   33       FORMAT(4X,I3,5X,A6,4X,A1,2X,G12.5,3X,G12.5,3X,G12.5)
          ELSE
            IF(VTYPE.EQ.'D')THEN
               WRITE(NUOUT,38)J,IDDESV(J),VTYPE,VBOUNS(1,J),
     &                     VBOUNS(2,J),DESVAR(J)
   38          FORMAT(4X,I3,5X,A6,4X,A1,2X,G12.5,3X,G12.5,3X,G12.5)
            ELSE
               KMIN=XMIN+0.5
               KMAX=XMAX+0.5
               KGES=XGES+0.5
               WRITE(NUOUT,34)J,IDDESV(J),VTYPE,KMIN,KMAX,KGES
   34          FORMAT(4X,I3,5X,A6,4X,A1,2X,I10,5X,I10,5X,I10)
            ENDIF
          ENDIF
C
   35   CONTINUE
C
        GO TO 1111
      ENDIF

C
```

```
CC BLOCK4 ------- DISCRETE -------------------------------------------
--
C     Read discrete design variable information
C
      IF(BLKNAM.EQ.'PVDISC')THEN
        CALL BLKCHK(NUOUT,BKIN(28))
C
        LVDISC = .TRUE.
        WRITE(NUOUT,285)
C
        READ(NUINP,FMT=*,ERR=8888)NUMNGH

        DO 284 J=1,NUMNGH

           READ(NUINP,FMT=*,ERR=8888)NUM,NVALUS(NUM),XGES
           IF (NUM .LE. NRELV) THEN
             WRITE(NUOUT,282)NUM
             GO TO 9999
           ENDIF
           IF (NUM .GT. NRELV+NDISV) THEN
             WRITE(NUOUT,282)NUM
             GO TO 9999
           ENDIF
           IF (XGES.LT.1 .OR. XGES.GT.NVALUS(NUM)) THEN
             GO TO 9999
           ENDIF
           IF (NUM .GT. 0) THEN
             READ(NUINP, FMT=*, ERR=8888)
(TABUN(NUM,I),I=1,NVALUS(NUM))
C  Guess value is read is as initial DESVAR
             INDEX(NUM) = XGES
             DESVAR(NUM) = TABUN(NUM,INDEX(NUM))
CORIGINAL     DESVAR(NUM) = TABUN(NUM,XGES)
             WRITE(NUOUT,281)NUM,IDDESV(NUM)
             WRITE(NUOUT,*)(TABUN(NUM,I),I=1,NVALUS(NUM))
             WRITE(NUOUT,286)DESVAR(NUM)
           ENDIF
 284     CONTINUE

 281  FORMAT (4X,'Variable No. ', I2, ' (', A, ')'
     &        ' has the following discrete values possible.')
 282  FORMAT(' E ** Variable number ',I3,' is not discrete')
 285  FORMAT(' I ** User inputted discrete data will be used',//)
 286  FORMAT(5X,'and the initial value is:',F6.4)

C         WRITE(NUOUT,283)
```

```
C 283   FORMAT(2X,'  Number   Name   Type    values   ',
C     &        2X,'  ------  ------  ----  ------------   ------------
',
C     &        3X,'------------')

          GO TO 1111

      ENDIF

C
CC BLOCK4 ------- NUMCAG -----------------------------------------
C     Read number of constraints and goals
C     NEW VARIABLES LINCON,LINGOL
C     NLINCO    Number of linear constraints
C     NNLINQ    Number of nonlinear inequality constraints
C     NNLEQU    Number of nonlinear equality constraints
C
C     NLINGO    Number of linear goals
C     NNLGOA    Number of nonlinear goals
C
      IF(BLKNAM.EQ.'NUMCAG')THEN
        CALL BLKCHK(NUOUT,BKIN(4))
C
        READ(NUINP,FMT=*,ERR=8888)NLINCO,NNLINQ,NNLEQU,NLINGO,NNLGOA
        WRITE(NUOUT,40) NLINCO,NNLINQ,NNLEQU,NLINGO,NNLGOA
   40   FORMAT(3X,
     &       ' Number of linear constraints               = ', I5/,
     &    3X,' Number of nonlinear inequality constraints = ', I5/,
     &    3X,' Number of nonlinear equality constraints   = ', I5/,
     &    3X,' Number of linear goals                     = ', I5/,
     &    3X,' Number of nonlinear goals                  = ', I5)
C
        NDEVAR  = (NLINGO+NNLGOA)*2
        NNLTOT  = NNLINQ+NNLEQU+NNLGOA
        NNLCON  = NNLINQ+NNLEQU
C
C       Check against maximum limits
C
        IF(NDEVAR.GT.MDEVV)THEN
          WRITE(NUOUT,44)MDEVV,NDEVAR
          GO TO 9999
        ENDIF
   44   FORMAT('  E ** Problem Size',/
     &        3X,' Maximum number of deviation variables = ',I5,/
     &        3X,'                               Specified = ',I5)
C
```

```
        IF(NNLTOT.GT.MNLNCG)THEN
          WRITE(NUOUT,45)MNLNCG,NNLTOT
          GO TO 9999
        ENDIF
   45   FORMAT('  E ** Problem Size',/
     &  3X,' Maximum number of nonlinear constraints + goals = ',I5,/
     &  3X,'                                      Specified = ',I5)
C
        IF((NLINCO+NLINGO).GT.MLINCG)THEN
          WRITE(NUOUT,46)MLINCG,NLINCO+NLINGO
          GO TO 9999
        ENDIF
   46   FORMAT('  E ** Problem Size',/
     &  3X,' Maximum number of linear constraints + goals = ',I5,/
     &  3X,'                                   Specified = ',I5)
C
C       Set Default names for nonlinear constraints
C
        DO 47 J=1,NNLCON
          IDNLCO(J)=MKNAME('NLCO',J)
   47   CONTINUE
C
C       Set Default names for nonlinear goals
C
        DO 48 J=1,NNLGOA
          IDNLGO(J)=MKNAME('NLGO',J)
   48   CONTINUE
C
C       Create names of Deviation Variables
C
        DO 41 J=1,NDEVAR
          KSGN=MOD(J,2)
          IF(KSGN.EQ.1)THEN
            ONE='-'
          ELSE
            ONE='+'
          ENDIF
          KDUM = (J+1)/2
          DUMNAM = MKNAME('D',KDUM)
          IF(KDUM.LE.9)THEN
            IDDEVR(J)=DUMNAM(1:2)//ONE//'   '
          ELSE
            IDDEVR(J)=DUMNAM(1:3)//ONE//'  '
          ENDIF
   41   CONTINUE
        WRITE(NUOUT,42)
   42   FORMAT(/,3X,' Names of deviation variables:',/)
        WRITE(NUOUT,43)(J,IDDEVR(J),J=1,NDEVAR)
   43   FORMAT(4(4X,I3,2X,A6))
C
        GO TO 1111
      ENDIF
C
CC BLOCK5 ------- LINCON --------------------------------------------
C     Reading in data for linear constraints
C
      IF(BLKNAM.EQ.'LINCON')THEN
        CALL BLKCHK(NUOUT,BKIN(5))
C
C       Check if NUMCAG has been read
C
        IF(.NOT.BKIN(4))THEN
          WRITE(NUOUT,710)
          GO TO 9999
        ENDIF
C
        WRITE(NUOUT,50)
 50     FORMAT(3X,' Linear constraints data:')
C
        DO 59 J=1,NLINCO
          READ(NUINP,'(A)',ERR=8888)DUM
          CALL GETNAM(DUM, DUMNAM, KST, LNONAM)
          IF( LNONAM ) DUMNAM = MKNAME('LICO',J)
          IDLICO(J)=DUMNAM
C
C         Initialize all cofs. to zero
C
          DO 51 K=1,NDESV
            COFLIN(J,K)=0.0
   51     CONTINUE
C
C         Read constraint information
C
          READ(DUM(KST:80),FMT=*,ERR=8888)NTERMS
          WRITE(NUOUT,52)J,IDLICO(J),NTERMS
   52     FORMAT(/,4X,I3,2X,A6,': Number of terms = ',I3)
C
          READ(NUINP,FMT=*,ERR=8888)(CIR(K),K=1,NTERMS)
C
          DUMOUT = ' '
          KONT = 0
C
```

```
          DO 54 K=1,NTERMS
            KONT = KONT+1
            CALL SPLITC(CIR(K),NDUM,VAL)
C
            COFLIN(J,NDUM) = VAL
C
            IF(VAL.GE.0.0)THEN
              ONE='+'
            IF(K.EQ.1)ONE=' '
            ELSE
              VAL = -VAL
              ONE='-'
            ENDIF
C
            J1 = (KONT-1)*23 + 11
            J2 = KONT    *23 + 11
            WRITE(DUMOUT(J1:J2),53)ONE,VAL, IDDESV(NDUM)
   53       FORMAT(X,A,' (',G11.5,'*',A6,')')
C
            IF((KONT.EQ.2) .OR. (K.EQ.NTERMS))THEN
              WRITE(NUOUT,FMT='(A)') DUMOUT
              DUMOUT = ' '
              KONT = 0
            ENDIF
C
   54     CONTINUE
C
          READ(NUINP,FMT='(A)',ERR=8888)DUM
          CALL GETNAM(DUM,DUMNAM,KST,LNONAM)
          READ(DUMNAM(1:2),FMT='(A)',ERR=8888)CCSIGN
          KSGN=IQSIGN(CCSIGN)
C
          IF(KSGN.LT.1 .OR. KSGN.GT.3)THEN
            WRITE(NUOUT,55)
   55       FORMAT(' E ** Unidentified sign for linear constraint')
            GO TO 9999
          ENDIF
C
          LISIGN(J)=KSGN
          READ(DUM(KST:80),FMT=*,ERR=8888)VAL
          RHSLIN(J)=VAL
C
          WRITE(NUOUT,56)CCSIGN,VAL
   56     FORMAT(55X,A,X,G12.5)
C
```

```
   59   CONTINUE
C
        GO TO 1111
      ENDIF
C
CC BLOCK6 ------- LINGOL ---------------------------------------------
C     Reading in data for linear goals
C
      IF(BLKNAM.EQ.'LINGOL')THEN
        CALL BLKCHK(NUOUT,BKIN(6))
C
C       Check if NUMCAG has been read.
C
        IF(.NOT.BKIN(4))THEN
          WRITE(NUOUT,710)
          GO TO 9999
        ENDIF
C
        WRITE(NUOUT,61)
   61   FORMAT(3X,' Linear goals data')
C
        DO 69 J=1,NLINGO
          READ(NUINP,'(A)',ERR=8888)DUM
          CALL GETNAM(DUM, DUMNAM, KST, LNONAM)
          READ(DUM(KST:80),FMT=*,ERR=8888)K,NTERMS
          IF( LNONAM )DUMNAM=MKNAME('LIGO',K)
          IDLIGO(K)=DUMNAM
          WRITE(NUOUT,62)K,IDLIGO(K),NTERMS
   62     FORMAT(/,2X,I3,2X,A6,' : Number of terms = ',I3)
C
C         Initialize all cofs. to zero
C
          DUM = ' '
C
          DO 65 K=1,NDESV
            COFLIN(J+NLINCO,K)=0.0
   65     CONTINUE
C
          READ(NUINP,FMT=*,ERR=8888)(CIR(K),K=1,NTERMS)
C
          DUMOUT = ' '
          KONT = 0
C
          DO 68 K=1,NTERMS
            KONT = KONT+1
            CALL SPLITC(CIR(K),NDUM,VAL)
```

```
C
            COFLIN(J+NLINCO,NDUM)=VAL
C
            IF(VAL.GE.0.0)THEN
              ONE='+'
              IF(K.EQ.1)ONE=' '
            ELSE
              VAL = -VAL
              ONE='-'
            ENDIF
C
            J1 = (KONT-1)*23+11
            J2 = KONT    *23+11
C
            WRITE(DUMOUT(J1:J2),53)ONE,VAL, IDDESV(NDUM)
C
            IF( (KONT.EQ.2) .OR. (K.EQ.NTERMS) ) THEN
              WRITE(NUOUT,FMT='(A)')DUMOUT
              DUMOUT = ' '
              KONT = 0
            ENDIF
C
   68     CONTINUE
C
          READ(NUINP,FMT=*,ERR=8888)VAL
          RHSLIN(J+NLINCO)=VAL
          WRITE(NUOUT,67)IDDEVR(2*J-1),IDDEVR(2*J),VAL
   67     FORMAT(10X,' + ( ',A,') - ( ',A,')',22X,'= ',G12.5)
C
   69   CONTINUE
C
        GO TO 1111
      ENDIF
C
CC BLOCK7 ------- DEVFUN ---------------------------------------------
C     Reading goal priorities and weights for deviation function
C
      IF((BLKNAM.EQ.'DEVFUN') .OR. (BLKNAM.EQ.'ACHFUN'))THEN
        CALL BLKCHK(NUOUT,BKIN(7))
C
C       Check if NUMCAG has been read.
C
        IF(.NOT.BKIN(4))THEN
          WRITE(NUOUT,710)
          GO TO 9999
        ENDIF
```

```
C
         READ(NUINP,FMT=*,ERR=8888)NMPRI
C
         IF(NMPRI.GT.MLEVEL-1)THEN
           WRITE(NUOUT,79)MLEVEL-1,NMPRI
           GO TO 9999
         ENDIF
C
   79    FORMAT('  E ** Problem Size',/
     &          3X,' Maximum number of goal priority levels = ',I5,/
     &          3X,'                              Specified = ',I5)
C
         WRITE(NUOUT,70)NMPRI
   70    FORMAT(3X,' Deviation Function data',//
     &          3X,' Number of priority levels = ',I3,//
     &          3X,' Level Dev.Var.    Weight',/
     &          3X,' ----- --------    ------')
C
         DO 74 J=1,NMPRI
           READ(NUINP,FMT=*,ERR=8888)NLEVEL, NTERMS
C
C Set default weights to zero.
C
CC           DO 71 K=1,NDEVAR
CC             DFNCOF(NLEVEL,K)=0.0
CC   71      CONTINUE
C
           READ(NUINP,FMT=*,ERR=8888)(CIR(K),K=1,NTERMS)
           DO 73 K=1,NTERMS
             CALL SPLITC(CIR(K),NUM,VAL)
             IF(NUM.LT.0)THEN
               KSGN=0
               NUM = -NUM
             ELSE
               KSGN=1
             ENDIF
             IVAR=2*NUM-1+KSGN
C
             DFNCOF(NLEVEL,IVAR)=VAL
C
             WRITE(NUOUT,72)NLEVEL,IDDEVR(IVAR),VAL
   72        FORMAT(4X,I3,5X,A6,2X,G12.5)
   73      CONTINUE
   74    CONTINUE
C
         GO TO 1111
```

```
      ENDIF
C
CC BLOCK8 ------- STOPCR -------------------------------------------
C     Read Stopping criteria
C
      IF(BLKNAM.EQ.'STOPCR')THEN
        CALL BLKCHK(NUOUT,BKIN(8))
C
        READ(NUINP,FMT=*,ERR=8888)J1 , J2, NUM, EPSZ, EPSX
C
C       Flag for dry run
C
        IF(J1 .EQ. 1)THEN
          LDRYRN = .FALSE.
        ELSE
          LDRYRN = .TRUE.
          WRITE(NUOUT, 87)
        ENDIF
C
C       Flag for printing final output only.
C
        LPRFIN = .FALSE.
        IF(J2 .EQ. 1)THEN
          LPRFIN = .TRUE.
          WRITE(NUOUT, 88)
        ENDIF
C
        IF(NUM.LE.0)THEN
          WRITE(NUOUT,84)
          GO TO 9999
        ENDIF
C
        NSYCY(1) = NUM
C
        WRITE(NUOUT, 82) NUM, NPTGEN, EPSX
C
        WRITE(NUOUT,86)
C
        DO 80 K = 1,NDESV
CCCC          IF (K.LE.NRELV) THEN
            FRACX(K) = EPSX * (VBOUNS(2,K) - VBOUNS(1,K))
CCCC          ELSE
CCCC            FRACX(K) = 1.0
CCCC          ENDIF
          WRITE(NUOUT,83)K,IDDESV(K),FRACX(K)
   80   CONTINUE
```

```
C
        DO 81 K = 1,NMPRI + 1
          FRACZ(K) = EPSZ
   81   CONTINUE
C
   82   FORMAT (3X,' Permitted No. of synthesis cycles    = ', I7/
     &        3X,' Obj. func. value stationarity (EPSZ) = ', G12.4/
     &        3X,' Design variable stationarity  (EPSX) = ', G12.4//)
   83   FORMAT (8X, I3, 8X, A6, 2X, G12.5)
   84   FORMAT (' W ** Synthesis cycles must be > 0')
   86   FORMAT (3X,' Values for stationarity of design',
     &           ' variables (FRACX): ',//
     &          3X,' FRACX(variable) = EPSX * variable range',//
     &          3X,' Variable No.    Name    FRACX  Value'/
     &          3X,' ------------   ------  ------------')
   87   FORMAT (' I ** Dry run, no computations will be performed',/)
   88   FORMAT (' I ** Only the final results will be printed',/)
C
        GO TO 1111
      ENDIF
C
CC BLOCK9 ------- NLINCO ---------------------------------------------
C       Reading in names of nonlinear constraints
C
        IF(BLKNAM.EQ.'NLINCO')THEN
        CALL BLKCHK(NUOUT,BKIN(9))
C
C       Check if NUMCAG has been read.
C
           IF(.NOT.BKIN(4))THEN
             WRITE(NUOUT,710)
             GO TO 9999
           ENDIF
C
        WRITE(NUOUT,91)
  91    FORMAT(4X,'Names of nonlinear constraints:',/)
C
        DO 99 J=1,NNLINQ+NNLEQU
          READ(NUINP,FMT='(A)',ERR=8888)DUM
          CALL GETNAM(DUM, DUMNAM, KST, LNONAM)
          READ(DUM(KST:80),*)K
          IF( LNONAM ) DUMNAM=MKNAME('NLCO',K)
          IDNLCO(K)=DUMNAM
C
          WRITE(NUOUT,92)J, IDNLCO(K)
  92       FORMAT(6X,'Nonlinear Constraint Number ',I3,' is ',A6)
```

```
 99      CONTINUE
C
        GO TO 1111
        ENDIF
C
CC BLOCK10------- NLINGO ---------------------------------------------
C       Reading in names of nonlinear goals
C
        IF(BLKNAM.EQ.'NLINGO') THEN
        CALL BLKCHK(NUOUT,BKIN(10))
C
C       Check if NUMCAG has been read.
C
           IF(.NOT.BKIN(4)) THEN
             WRITE(NUOUT,710)
             GO TO 9999
           ENDIF
C
        WRITE(NUOUT,101)
 101     FORMAT(4X,'Names of nonlinear goals:',/)
C
        DO 109 J = 1, NNLGOA
          READ(NUINP,'(A)',ERR=8888) DUM
          CALL GETNAM(DUM, DUMNAM, KST, LNONAM)
          READ(DUM(KST:80),FMT=*,ERR=8888) K
C
          IF( (K-NLINGO) .LE. 0) THEN
            WRITE(NUOUT, 106) NLINGO
            GO TO 9999
          ENDIF
C
          IF( LNONAM ) DUMNAM = MKNAME('NLGO',K)
          IDNLGO(K - NLINGO) = DUMNAM
          WRITE(NUOUT,102) K, IDNLGO(K - NLINGO)
 102      FORMAT(6X,'Nonlinear Goal Number ',I3,' is ',A6)
 109     CONTINUE
C
 106    FORMAT(' E ** Nonlinear goal numbers must be > NLINGO = ',I5)
C
        GO TO 1111
        ENDIF
C
CC BLOCK11------- INITFS ---------------------------------------------
C    IINIT = 0: Initial feasible solution not generated
C          = 1: Generate initial feasible solution
C
```

```
        IF(BLKNAM.EQ.'INITFS')THEN
        CALL BLKCHK(NUOUT,BKIN(11))
C
        LINIT =.TRUE.
C
        WRITE (NUOUT, 110)
  110 FORMAT (' I ** Auto. initial solution generation, based on
user',
     &       ' provided guess.')
C
        READ(NUINP,FMT=*,ERR=8888)NHJMAX,HJEXPA, HJCONT,
     &                            HJEPSY,HJSTEP,HJDELT
        WRITE(NUOUT,111)NHJMAX,HJEXPA,HJCONT,HJEPSY,HJSTEP,HJDELT
  111    FORMAT (3X/,'    Pattern search parameters:',//,
     &        3X,'   Maximum number of calls to USRSET = ', I4/,
     &        3X,'   Expansion factor                  = ', G12.4/,
     &        3X,'   Contraction factor                = ', G12.4/,
     &        3X,'   EPSY                              = ', G12.4/,
     &        3X,'   Minimum step                      = ', G12.4/,
     &        3X,'   DELTA                             = ', G12.4/)
C
        GO TO 1111
        ENDIF
C
CC BLOCK12------- ALPOUT ---------------------------------------------
C
C      Flags to control various blocks of output
C         Flag = 1 print output data block
C              = 0 do not print
C
C      LPROUT(1) : design variables
C      LPROUT(2) : deviation variables
C      LPROUT(3) : deviation function
C      LPROUT(4) : bound information
C      LPROUT(5) : linear constraint information
C      LPROUT(6) : nonlinear constraint information
C      LPROUT(7) : linear constraint activity for linear solution
C      LPROUT(8) : nonlinear constraint activity for linear solution.
C
C      LPPROC    : postprocessor information
C      LTIME     : time statistics
C
        IF(BLKNAM.EQ.'ALPOUT')THEN
        CALL BLKCHK(NUOUT,BKIN(12))
        WRITE(NUOUT,120)
C
```

```
C       Reset all flags
C
        DO 1277 K=1,8
 1277   LPROUT(K) = .FALSE.
C
        LPPROC    = .FALSE.
        LTIME     = .FALSE.
C
        READ(NUINP,FMT=*,ERR=8888)(IPR(K),K=1,8),J1, J2
C
         IF (IPR(1).EQ.1) LPROUT(1) =.TRUE.
         IF (IPR(2).EQ.1) LPROUT(2) =.TRUE.
         IF (IPR(3).EQ.1) LPROUT(3) =.TRUE.
         IF (IPR(4).EQ.1) LPROUT(4) =.TRUE.
         IF (IPR(5).EQ.1) LPROUT(5) =.TRUE.
         IF (IPR(6).EQ.1) LPROUT(6) =.TRUE.
         IF (IPR(7).EQ.1) LPROUT(7) =.TRUE.
         IF (IPR(8).EQ.1) LPROUT(8) =.TRUE.
C
         IF (LPROUT(1)) WRITE (NUOUT, 121)
         IF (LPROUT(2)) WRITE (NUOUT, 122)
         IF (LPROUT(3)) WRITE (NUOUT, 123)
         IF (LPROUT(4)) WRITE (NUOUT, 124)
         IF (LPROUT(5)) WRITE (NUOUT, 125)
         IF (LPROUT(6)) WRITE (NUOUT, 126)
         IF (LPROUT(7)) WRITE (NUOUT, 127)
         IF (LPROUT(8)) WRITE (NUOUT, 128)
C
         IF (J1 .EQ.1) LPPROC = .TRUE.
         IF (J2 .EQ.1) LTIME  = .TRUE.
C
         IF (LPPROC) WRITE (NUOUT, 720)
         IF (LTIME) WRITE  (NUOUT, 730)
C
  120 FORMAT (4X,'The following output information will be
printed:',/)
  121 FORMAT (6X,'1. System variables')
  122 FORMAT (6X,'2. Deviation variables')
  123 FORMAT (6X,'3. Deviation function')
  124 FORMAT (6X,'4. Bound information')
  125 FORMAT (6X,'5. Linear constraint information')
  126 FORMAT (6X,'6. Nonlinear constraint information')
  127 FORMAT (6X,'7. Linear constraint activity ',
     &           'for linear solution')
  128 FORMAT (6X,'8. Nonlinear constraint activity ',
     &          'for linear soln.')
  720 FORMAT (/,6X,'Post processor information will be printed')
  730 FORMAT (/,6X,'Time statistics will be printed')
C
        GO TO 1111
        ENDIF
C
CC BLOCK13------- USRMOD ---------------------------------------------
C   Read flags related to user supplied subroutines.
C      IOUSER = 0: User has not provided input routine/s
C             = 1: User has provided input routine/s (LUINP)
C      IOUT   = 0: User has not provided output routine/s
C             = 1: User has provided output routine/s (LUOUT)
C      IMON   = 0: Subroutine USRMON is not called
C             = 1: Subroutine USRMON is called
C      JVARC  = 0: Subroutine USRLIN is not called
C             = 1: Subroutine USRLIN is called
C
        IF(BLKNAM.EQ.'USRMOD')THEN
        CALL BLKCHK(NUOUT,BKIN(13))
C
        READ(NUINP,FMT=*,ERR=8888)IOUSER,IOUT,IMON,JVARC
        IF (IOUSER.EQ.1) LUINP=.TRUE.
        IF (IOUT.EQ.1)   LUOUT=.TRUE.
        IF (IMON.EQ.1)   LMON =.TRUE.
        IF (JVARC.EQ.1)  LVCOF=.TRUE.
C
        IF (LUINP) WRITE (NUOUT, 130)
        IF (LUOUT) WRITE (NUOUT, 131)
        IF (LVCOF) WRITE (NUOUT, 132)
        IF (LMON ) WRITE (NUOUT, 133)
C
  130 FORMAT(' I ** User provided input routine USRINP used.')
  131 FORMAT(' I ** User provided output routine USROUT used.')
  132 FORMAT(' I ** User provided update routine USRLIN used.')
  133 FORMAT(' I ** User provided monitoring routine USRMON used.')
C
        GO TO 1111
        ENDIF
C
CC BLOCK14------- USRDAT ---------------------------------------------
C   User provided information to be read in Subroutine USRINP
C   Required only if USRINP is called.
C
        IF(BLKNAM.EQ.'USRDAT')THEN
        CALL BLKCHK(NUOUT,BKIN(14))
C
```

```
C ML = Number of lines of user data to be read by USRINP.
C
        READ(NUINP,FMT=*,ERR=8888)ML
        WRITE(NUOUT,145)ML,NUSER
145     FORMAT(' I ** ',I3,' lines of data written to unit number
',I3)
C
        IF (ML .GT. 0) THEN
          DO 140 I = 1, ML
            READ (NUINP, FMT='(A)', ERR=8888) DUM
            WRITE (NUOUT, FMT='(A)') DUM
            WRITE (NUSER, FMT='(A)') DUM
140       CONTINUE
        ENDIF
C
        GO TO 1111
        ENDIF
C
CC BLOCK15------- OPTIMP ---------------------------------------------
C Optimization control parameters.
C
      IF(BLKNAM.EQ.'OPTIMP')THEN
        READ(NUINP,FMT=*,ERR=8888)VIOLIM, REMOVS, PSTEP
        CALL BLKCHK(NUOUT,BKIN(15))
C
C *     Error traps for VIOLIM and PSTEP values, reset to default if
C *     wrong values given.
C
        IF ( VIOLIM .GT. -VLINCO) THEN
          WRITE(NUOUT,153) -VLINCO, -VLINCO
          VIOLIM = -VLINCO
        ENDIF
C
        IF (PSTEP .LE. 0.0) THEN
          WRITE (NUOUT, 154)
          PSTEP = 0.005
        ENDIF
C
        IF ((REMOVS .LE. 0.) .OR. (REMOVS .GT. 1.0)) THEN
          REMOVS = 0.5
          WRITE (NUOUT, 155)
        ENDIF
C
        WRITE (NUOUT, 156) VIOLIM, REMOVS, PSTEP
C
        DO 150 I=1,NNLCON
```

```
  150     VILCN(I)=VIOLIM
C
        DO 151 I=1,NRELV
  151     REDMOV(I) = REMOVS
C
        DO 152 I = 1, NDESV
  152     PESTEP(I) = PSTEP * (VBOUNS(2,I) - VBOUNS(1,I))
C
  153 FORMAT (' W ** VIOLIM must not be greater ',
     &        'than ',F8.4,', value reset to ',F8.4/)
  154 FORMAT (' W ** Fractional step size cannot be less than ',
     &        'or equal to zero, default set to .005 '/)
  155 FORMAT (' W ** Reduced move limit is out of range, ',
     &        ' default value used.')
  156 FORMAT (4X,'Relaxation for nonlin. const. (VIOLIM) = ', F8.4/
     &        4X,'Reduced move coefficient      (REMO)   = ', F8.4/
     &        4X,'Linearization step size       (STEP)   = ', F8.4//
     &        4X,'PESTEP(variable) = STEP * variable range')
C
        GO TO 1111
      ENDIF
C
CC BLOCK16------- ADPCTL --------------------------------------------
C Read adaptive control parameters.
C These are primarily for the advanced user.
C
        IF(BLKNAM.EQ.'ADPCTL')THEN
        CALL BLKCHK(NUOUT,BKIN(16))
C
        LADAP  = .FALSE.
C
        READ(NUINP,FMT=*,ERR=8888)IADAP
C
          IF (IADAP .EQ. 1) THEN
            LADAP = .TRUE.
            WRITE(NUOUT,163)
          ENDIF
  163   FORMAT(' I ** Constraint and goal adaptation will be used.')
C
        GO TO 1111
        ENDIF
C
CC BLOCK17------- USERAN --------------------------------------------
C     Read adaptive control parameters.
C     These are primarily for the advanced user.
C
```

```
      IF(BLKNAM.EQ.'USERAN')THEN
        CALL BLKCHK(NUOUT,BKIN(17))
C
        READ(NUINP,FMT=*,ERR=8888)NANCY
        WRITE(NUOUT,170)NANCY
  170   FORMAT(' I ** Number of synthesis cycles = ',I3,/)
C
        READ(NUINP,FMT=*,ERR=8888)(NSYCY(J),J=1,NANCY)
        WRITE(NUOUT,171)(J,NSYCY(J),J=1,NANCY)
  171   FORMAT(4X,'Number of analysis cycles for synthesis cycle',I3,
     &          ' = ',I3)
C
        GO TO 1111
      ENDIF
C
CC BLOCK18------- FIXVAR ---------------------------------------------
C   Read and print fixing of variables information.
C
        IF(BLKNAM.EQ.'FIXVAR')THEN
        CALL BLKCHK(NUOUT,BKIN(18))
C
        READ(NUINP,FMT=*,ERR=8888)NUM
          IF (NUM .GT. 0) THEN
               READ (NUINP, FMT=*, ERR=8888) (JDUM(I), I=1,NUM)
            DO 180 J = 1, NUM
              II = JDUM(J)
              IF(II.GT.NDESV)THEN
                WRITE(NUOUT,182)II
                GO TO 9999
              ENDIF
              WRITE (NUOUT, 181) II, IDDESV(II)
  180       CONTINUE
          ENDIF
C
          CALL IARFIL ( NUM, JDUM, NDESV, IACTVR )
C
  181     FORMAT (4X,'Variable No. ', I2, ' (', A, ')'
     &         ' has been fixed at initial guess.')
  182     FORMAT(' E ** Variable number ',I3,' is unknown')
C
        GO TO 1111
        ENDIF
C
CC BLOCK19------- SUPCON ---------------------------------------------
C   Read and print constraint suppression information.
C
```

```
        IF(BLKNAM.EQ.'SUPCON')THEN
        CALL BLKCHK(NUOUT,BKIN(19))
C
        WRITE(NUOUT,191)
C
        READ(NUINP,FMT=*,ERR=8888)NUM
        IF (NUM .GT. 0) THEN
          READ (NUINP, FMT=*, ERR=8888) (JDUM(K), K=1,NUM)
          DO 190 J = 1, NUM
            WRITE (NUOUT, 192) JDUM(J)
  190     CONTINUE
        ENDIF
  191 FORMAT (4X,'Nonlinear Constraint Suppression Information: ',/)
  192 FORMAT (4X,'Suppress nonlinear constraint number: ', I4)
C
        CALL IARFIL(NUM,JDUM,NNLCON,IADCON)
C
        GO TO 1111
        ENDIF
C
CC BLOCK20------- PVALFX ------------------------------------------
C  If FRACX for individual variables to be specified read
C  and print these values
C
        IF(BLKNAM.EQ.'PVALFX')THEN
        CALL BLKCHK(NUOUT,BKIN(20))
C
        READ(NUINP,FMT=*,ERR=8888)NFRAC
        IF (NFRAC .GT. 0) THEN
C
          READ (NUINP, FMT=*, ERR=8888) (CIR(K), K=1,NFRAC)
          WRITE (NUOUT, 201)
C
          DO 200 K = 1, NFRAC
            CALL SPLITC(CIR(K),II,VAL)
CCCC            IF (II.LE.NRELV) THEN
              FRACX(II) = VAL * (VBOUNS(2,II) - VBOUNS(1,II))
CCCC            ELSE
CCCC              FRACX(II) = 1.0
CCCC            ENDIF
  200       WRITE (NUOUT, 202) II, IDDESV(II), FRACX(II), VAL
        ENDIF
C
  201 FORMAT (4X,'Particular values for stationarity of design',
     &        ' variables (FRACX): ',//
     &       4X,'FRACX(variable) = Fraction * variable range'//
     &       4X,'Variable No.    Name      Fraction     FRACX  Value'/
     &       4X,'------------   ------   ------------   ------------')
  202 FORMAT (4X,'   ', I3, 9X, A6, 3X, G12.4, 3X, G12.4)
C
        GO TO 1111
        ENDIF
C
CC BLOCK21------- PVALFZ ---------------------------------------------
C  If FRACZ for individual variables to be specified read
C  and print these values
C
        IF(BLKNAM.EQ.'PVALFZ')THEN
        CALL BLKCHK(NUOUT,BKIN(21))
C
        READ(NUINP,FMT=*,ERR=8888)NFRAC
        IF (NFRAC .GT. 0) THEN
C
          READ (NUINP, FMT=*, ERR=8888) (CIR(K), K=1,NFRAC)
          WRITE (NUOUT, 211)
C
            DO 210 K = 1, NFRAC
            CALL SPLITC(CIR(K),II,VAL)
            WRITE (NUOUT, 212) II, VAL
  210       FRACZ(II+1) = VAL
        ENDIF
C
  211 FORMAT (4X,'Particular values for stationarity of deviation',
     &        ' function (FRACZ): '//
     &        4X,'Priority Level   FRACZ  Value'/
     &        4X,'--------------   ------------')
  212 FORMAT (4X,'   ',I3, 11X, G12.4)
C
        GO TO 1111
        ENDIF
C
CC BLOCK22------- PVSTEP ---------------------------------------------
C  Perturbation step specification for nonlinear constraints and
goals.
C
        IF(BLKNAM.EQ.'PVSTEP')THEN
          CALL BLKCHK(NUOUT,BKIN(22))
C
          WRITE(NUOUT,224)PSTEP
C
          DO 225 I = 1, NDESV
 225      PESTEP(I) = PSTEP * (VBOUNS(2,I) - VBOUNS(1,I))
```

```
C
          READ (NUINP, FMT=*, ERR=8888) NPSTEP
C
          IF (NPSTEP .GT. 0) THEN
            READ (NUINP, FMT=*, ERR=8888) (CIR(K), K=1,NPSTEP)
            WRITE (NUOUT, 221)
C
            DO 220 J = 1, NPSTEP
               CALL SPLITC(CIR(J),NUM,VAL)
               IF (VAL .LE. 0.0) THEN
                  WRITE (NUOUT, 222)
                  VAL = 0.005
               ENDIF
               PESTEP(NUM) = VAL * (VBOUNS(2,NUM) - VBOUNS(1,NUM))
               WRITE (NUOUT, 223) NUM, IDDESV(NUM), PESTEP(NUM), VAL
  220       CONTINUE
C
          ENDIF
C
  221 FORMAT (4X,'Perturbation step size for linearizing',
     &          ' nonlinear constraints & goals:')
  222 FORMAT (' W ** Fractional step size cannot be less than ',
     &         'or equal to zero default set to .005 '/)
  223 FORMAT ('      Variable No.', I3, ', Name: ', A,
     &         ' linearization step size = ', F7.4,
     &         ' (fraction = ',F7.4,')')
  224 FORMAT (3X,' Default value of STEP = ',G12.5,//
     &         3X,' PESTEP(variable) = STEP * variable range')
C
        GO TO 1111
        ENDIF
C
CC BLOCK23------- PVCVIL ---------------------------------------------
C  If FRAC3 for individual nonlinear constraints to be
C  specified read and print these values
C
        IF(BLKNAM.EQ.'PVCVIL')THEN
        CALL BLKCHK(NUOUT,BKIN(23))
C
        WRITE(NUOUT,233) VIOLIM
 233    FORMAT(4X,'Default value of VIOLIM = ',G12.5)
        DO 234 I=1, NNLCON
 234       VILCN(I) = VIOLIM
C
        READ(NUINP,FMT=*,ERR=8888)NFRAC
          IF (NFRAC .GT. 0) THEN
```

```
            READ (NUINP, FMT=*, ERR=8888) (CIR(K), K=1,NFRAC)
            WRITE (NUOUT, 231)
C
            DO 230 K = 1, NFRAC
               CALL SPLITC(CIR(K),NUM,VAL)
               WRITE (NUOUT, 232) NUM, IDNLCO(NUM), VAL
  230          VILCN(NUM) = VAL
          ENDIF
C
  231 FORMAT (' '/4X,'Particular values for nonlinear constraint',
     &        ' satisfaction (VILCN):'//
     &        4X,'Nonlinear         NAME      VILCN'/
     &        4X,'Constraint No.'/
     &        4X,'--------------   ------   --------')
  232 FORMAT (8X, I3, 10X, A, 3X, F8.2)
C
        GO TO 1111
        ENDIF
C
CC BLOCK24------- PVREMO ---------------------------------------------
C  If different move limit required for variables.
C
        IF(BLKNAM.EQ.'PVREMO')THEN
        CALL BLKCHK(NUOUT,BKIN(24))
C
        WRITE(NUOUT,243)REMOVS
 243    FORMAT(3X,' Default value of reduced move = ',G12.5)
C
        DO 244 I=1,NRELV
 244    REDMOV(I) = REMOVS
C
        READ(NUINP,FMT=*,ERR=8888) NREMOV
C
         IF (NREMOV .GT. 0) THEN
            READ (NUINP, FMT=*, ERR=8888) (CIR(J), J=1,NREMOV)
            WRITE (NUOUT, 241)
            DO 240 J = 1, NREMOV
               CALL SPLITC(CIR(J),NUM,VAL)
               WRITE (NUOUT, 242) NUM, IDDESV(NUM), VAL
               REDMOV(NUM) = VAL
  240       CONTINUE
         ENDIF
C
  242 FORMAT ('      Variable No.', I3, ', Name: ', A, ', Reduced ',
     &        'move limit is = ', F7.4)
  241 FORMAT (' Reduced move limits for variables: ')
```

```
C
        GO TO 1111
        ENDIF
C
CC BLOCK25------- ADREMO ---------------------------------------------
C       Parameters for adaptive reduced move.
C
        IF(BLKNAM.EQ.'ADREMO')THEN
        CALL BLKCHK(NUOUT,BKIN(25))
C
        LADREM = .TRUE.
C
        WRITE(NUOUT,250)
 250    FORMAT(' I ** Adaptive reduced move limit will be used',//)
C
        READ(NUINP,FMT=*,ERR=8888)NADREM, DELREM
        WRITE(NUOUT,252)NADREM, DELREM
 252    FORMAT(' I ** Maximum number of calls to USRSET = ',I3,/,
     &         '      Minimum reduced move             = ',G10.2)
C
        GO TO 1111
        ENDIF
C
CC BLOCK27------- ENDPRB ---------------------------------------------
C       Special block to indicate termination.
C
        IF(BLKNAM.EQ.'ENDPRB')THEN
          WRITE(NUOUT,260)
260       FORMAT(' I ** Stop reading data file',
     &    ' - Rest of data file will be ignored',//)
          GO TO 9000
        ENDIF
CC BLOCK26------- XPLORE ---------------------------------------------
C   Read and print fixing of variables information.
C
      IF (BLKNAM.EQ.'XPLORE') THEN
        CALL BLKCHK(NUOUT,BKIN(26))
C
        LXPLOR = .TRUE.
C
        DO 275 J = 1, NDESV
          IGENFX(J) = 0
  275   CONTINUE
C
        READ(NUINP,FMT=*,ERR=8888)NPTGEN,NPTBST,J1,IGSEED
C
```

```
      IF (J1.EQ.1) THEN
        LPRGEN = .TRUE.
        WRITE (NUOUT, 278)
      ELSE
        WRITE (NUOUT, 279)
      ENDIF
C
      WRITE (NUOUT, 2711) NPTGEN, NPTBST
C
      IF ( IGSEED .GT. 2530 ) THEN
        WRITE (NUOUT, 274) IGSEED
        IGSEED = 2530
      ELSE
        IF ( IGSEED .LT. 1 ) THEN
          WRITE (NUOUT, 276) IGSEED
          IGSEED = 1
        ELSE
          WRITE (NUOUT, 277) IGSEED
        ENDIF
      ENDIF
C
      READ(NUINP,FMT=*,ERR=8888)NUM
C
      IF(NUM.GT.NDESV)THEN
        WRITE(NUOUT,273)NUM,NDESV
        GO TO 9999
      ENDIF
C
      IF (NUM .GT. 0) THEN
        READ (NUINP, FMT=*, ERR=8888) (JDUM(I), I=1,NUM)
        DO 270 J = 1, NUM
          II = JDUM(J)
C
          IF(II.GT.NDESV)THEN
            WRITE(NUOUT,272)II
            GO TO 9999
          ENDIF
C
          IGENFX(II) = 1
          WRITE (NUOUT, 271) II, IDDESV(II)
 270    CONTINUE
      ENDIF
C
 271  FORMAT (4X,'Variable No. ', I2, ' (', A, ')'
     &            ' will be fixed at initial guess.')
 272  FORMAT(' E ** Variable number ',I3,' is unknown')
```

```
  273   FORMAT(' E ** Number of fixed variables  = ',I4,/,
     &         '       Number of design variables = ',I4)
  274   FORMAT(' I ** User supplied integer seed  = ',I6,/,
     &         '       Seed reset to maximum value =   2530')
  276   FORMAT(' I ** User supplied integer seed  = ',I6,/,
     &         '       Seed reset to minimum value =      1')
  277   FORMAT(4X,'User supplied integer seed             = ',I6,/)
  278   FORMAT(4X,'Data printed to standard output file',/)
  279   FORMAT(4X,'Data NOT printed to standard output file')
 2711   FORMAT(4X,'Total number of points to be generated = ',I6,/
     &         4X,'Number of best points recorded         = ',I6)
C
        GO TO 1111
      ENDIF
C
C------------------------------------
CCC* Fall through option for bad block name
C
      WRITE(NUOUT,987)BLKNAM
987   FORMAT(' E ** Fatal error: Invalid blockname >> ',A)
      GO TO 9999
C------------------------------------
 9000 CONTINUE
C
C     Successful end of data file input indicated by
C     end-of-file or the ENDPRB block
C
C     Consistency checks:
C
C     Check if all mandatory blocks have been read.
C
      IF(.NOT.BKIN(1))THEN
        WRITE(NUOUT,510)'PTITLE'
        GO TO 9999
      ENDIF
C
      IF(.NOT.BKIN(2))THEN
        WRITE(NUOUT,510)'NUMSYS'
        GO TO 9999
      ENDIF
C
      IF(.NOT.BKIN(3))THEN
        WRITE(NUOUT,510)'SYSVAR'
        GO TO 9999
      ENDIF
C
```

```
      IF(.NOT.BKIN(4))THEN
        WRITE(NUOUT,510)'NUMCAG'
        GO TO 9999
      ENDIF
C
      IF((NLINCO.NE.0).AND.(.NOT.BKIN(5)))THEN
        WRITE(NUOUT,510)'LINCON'
        GO TO 9999
      ENDIF
C
      IF((NLINGO.NE.0).AND.(.NOT.BKIN(6)))THEN
        WRITE(NUOUT,510)'LINGOL'
        GO TO 9999
      ENDIF
C
      IF(.NOT.BKIN(7))THEN
        WRITE(NUOUT,510)'ACHFUN'
        GO TO 9999
      ENDIF
C
      IF(.NOT.BKIN(8))THEN
        WRITE(NUOUT,510)'STOPCR'
        GO TO 9999
      ENDIF
C
  510 FORMAT(' E ** Mandatory block ',A,' not specified')
C
C     Default print settings
C
      IF(.NOT.BKIN(12))THEN
        WRITE(NUOUT,120)
        WRITE(NUOUT,121)
        WRITE(NUOUT,122)
        WRITE(NUOUT,123)
      ENDIF
C
C     Default names for nonlinear constraints if NLINCO not given
C
      IF((NNLCON.GT.0).AND.(.NOT.BKIN(9)))THEN
        WRITE(NUOUT,520)
  520   FORMAT(/,3X,' Default names of nonlinear constraints')
        DO 540 J=1,NNLCON
          IDNLCO(J)=MKNAME('NLCO',J)
          WRITE(NUOUT,530)J,IDNLCO(J)
  530     FORMAT(3X,I3,2X,A)
  540   CONTINUE
```

```
      ENDIF
C
C     Default names for nonlinear goals if NLINGO not given
C
      IF((NNLGOA.GT.0).AND.(.NOT.BKIN(10)))THEN
        WRITE(NUOUT,550)
  550   FORMAT(/,3X,' Default names of nonlinear goals')
        DO 570 J=1,NNLGOA
          IDNLGO(J)=MKNAME('NLGO',J)
          WRITE(NUOUT,560)J,IDNLGO(J)
  560     FORMAT(3X,I3,2X,A)
  570   CONTINUE
      ENDIF
C
C     Other default values to be set
C
C     Default VILCN values
C
      IF((.NOT.BKIN(15)).AND.(.NOT.BKIN(23)))THEN
        WRITE(NUOUT,*)
        WRITE(NUOUT,233) VIOLIM
        DO 580 I=1,NNLCON
  580     VILCN(I) = VIOLIM
      ENDIF
C
C     Default REDMOV values
C
      IF((.NOT.BKIN(15)).AND.(.NOT.BKIN(24)))THEN
        WRITE(NUOUT,*)
        WRITE(NUOUT,243)REMOVS
        DO 590 I=1,NRELV
  590     REDMOV(I) = REMOVS
      ENDIF
C
C
C     Default PESTEP values
C
      IF((.NOT.BKIN(15)).AND.(.NOT.BKIN(22)))THEN
        WRITE(NUOUT,*)
        WRITE(NUOUT,224)PSTEP
        DO 600 I = 1, NDESV
  600     PESTEP(I) = PSTEP * (VBOUNS(2,I) - VBOUNS(1,I))
      ENDIF
C
C
C     Default IACTVR values or
```

```
C     modify corresponding linear terms to appear as constants
C     and transfer values to the RHS
C
      IF ( .NOT.BKIN(18) )THEN
        DO 610 J = 1, NDESV
          IACTVR(J) = J
  610   CONTINUE
      ELSE
        IF ( NLINCO+NLINGO .GT. 0 ) THEN
           DO 630 J = 1, NDESV
             IF ( IACTVR(J) .LE. 0 ) THEN
               DO 620 I = 1, NLINCO+NLINGO
                 IF ( COFLIN(I,J) .NE. 0.0 ) THEN
                   RHSLIN(I) = RHSLIN(I) - COFLIN(I,J)*DESVAR(J)
                   COFLIN(I,J) = 0.0
                   IF ( I .LE. NLINCO ) THEN
                     WRITE (NUOUT, 625) IDLICO(I), IDDESV(J)
                   ELSE
                     WRITE (NUOUT, 625) IDLIGO(I-NLINCO), IDDESV(J)
                   ENDIF
                 ENDIF
  620          CONTINUE
             ENDIF
  630      CONTINUE
        ENDIF
      ENDIF
C
C
C     Default IADCON values
C
      IF (.NOT.BKIN(19))THEN
        DO 650 J = 1, NNLCON
           IADCON(J) = J
  650   CONTINUE
      ENDIF
C
C
C     Sucessful completion of data input
      FATAL = .FALSE.
      WRITE(NUOUT,*)
      WRITE(NUOUT,*)
C
      RETURN
C
C
C-------------------------------------
```

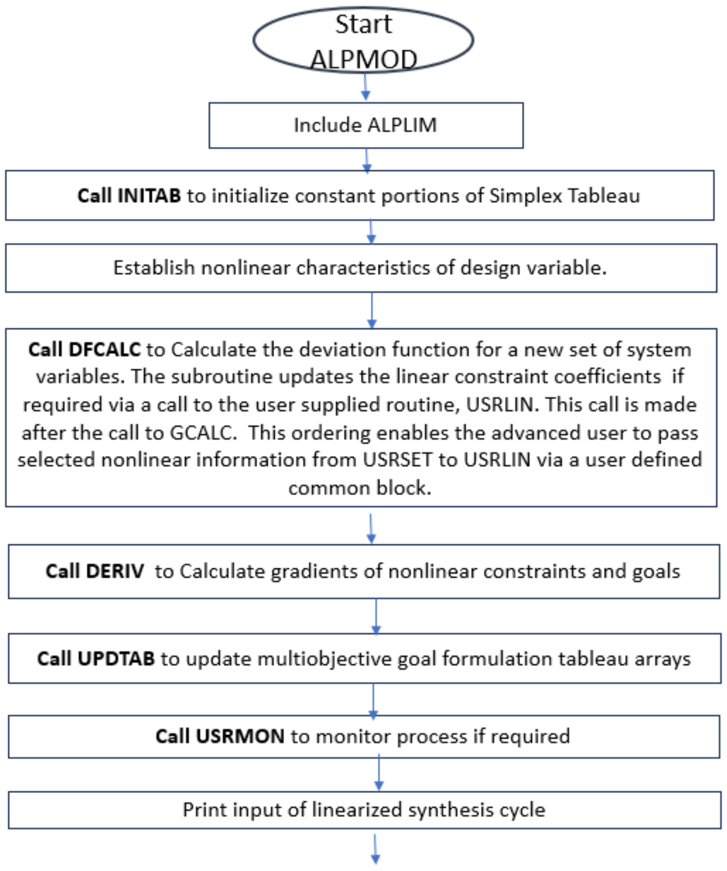
```
C
C     FORTRAN read error encountered
C
 8888 CONTINUE
C
      WRITE(NUOUT,988)BLKNAM
      GO TO 9999
C ------------------------------------
C
C     Unsuccesful data input for any reason
C
 9999 CONTINUE
C
      WRITE(NUOUT, 777)
      FATAL = .TRUE.
      WRITE(NUOUT,*)
      WRITE(NUOUT,*)
C
C
      RETURN
C
C*********************************************************************
**
C
C     Formats:
C
  625 FORMAT(' I ** ', A6,' - COFLIN and RHSLIN modified due to 
fixed',
     &       ' variable, ',A6)
  700 FORMAT(' E ** NUMSYS block must precede SYSVAR')
  710 FORMAT(' E ** NUMCAG must precede this block')
  777 FORMAT(/,' E **  Fatal error reading data file. END OF JOB.')
  988 FORMAT(' E ** Error reading data in block ',A)
C
C*********************************************************************
**
C     End of Subroutine ALPDAT
C*********************************************************************
**
C
      END
```

Up to this point, two input subroutines have been encompassed in our study: ALPLIM, where all other main input subroutines are invoked, and ALPDAT, where information from all linear sheet blocks is meticulously processed. The third input subroutine, ALPMOD, will be introduced in the next section.

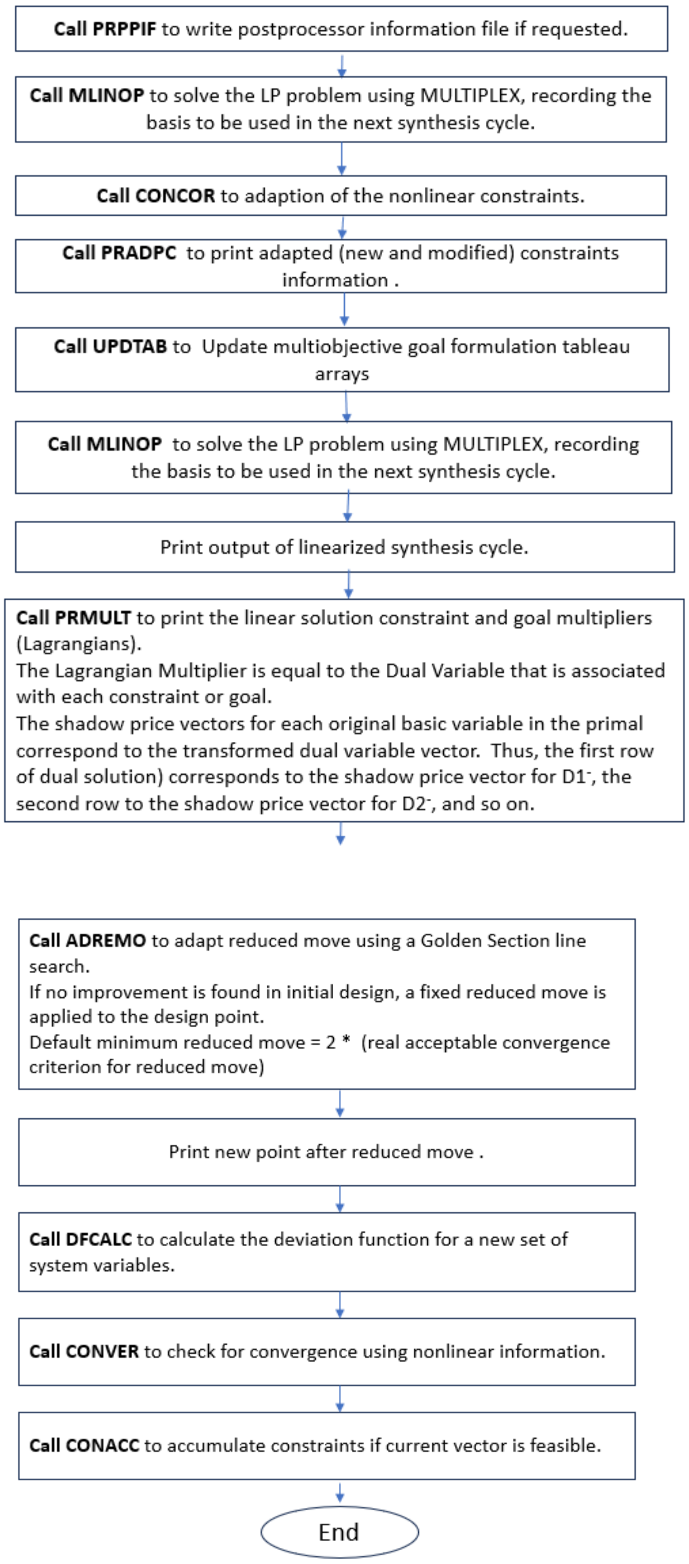
III.2.3 ALPMOD Subroutine and its Flowchart

The purpose of this subroutine is to perform the synthesis cycles. (It is about stopping criteria).

To better view this subroutine, I have considered two flowcharts. The first one is more general to give a big picture to the user, and the second flowchart is in more detail. The general flowchart of ALPMOD is represented in Figure III.1, but the detailed flowchart and the subroutine have been added in the appendix.



Start ALPMOD
Include ALPLIM
Call INITAB to initialize constant portions of Simplex Tableau
Establish nonlinear characteristics of design variable.
Call DFCALC to Calculate the deviation function for a new set of system variables. The subroutine updates the linear constraint coefficients if required via a call to the user supplied routine, USRLIN. This call is made after the call to GCALC. This ordering enables the advanced user to pass selected nonlinear information from USRSET to USRLIN via a user defined common block.
Call DERIV to Calculate gradients of nonlinear constraints and goals
Call UPDTAB to update multiobjective goal formulation tableau arrays
Call USRMON to monitor process if required
Print input of linearized synthesis cycle


Call PRPPIF to write postprocessor information file if requested.

Call MLINOP to solve the LP problem using MULTIPLEX, recording the basis to be used in the next synthesis cycle.

Call CONCOR to adaption of the nonlinear constraints.

Call PRADPC to print adapted (new and modified) constraints information .

Call UPDTAB to Update multiobjective goal formulation tableau arrays

Call MLINOP to solve the LP problem using MULTIPLEX, recording the basis to be used in the next synthesis cycle.

Print output of linearized synthesis cycle.

Call PRMULT to print the linear solution constraint and goal multipliers (Lagrangians).
The Lagrangian Multiplier is equal to the Dual Variable that is associated with each constraint or goal.
The shadow price vectors for each original basic variable in the primal correspond to the transformed dual variable vector. Thus, the first row of dual solution) corresponds to the shadow price vector for $D1^-$, the second row to the shadow price vector for $D2^-$, and so on.

Call ADREMO to adapt reduced move using a Golden Section line search.
If no improvement is found in initial design, a fixed reduced move is applied to the design point.
Default minimum reduced move = 2 * (real acceptable convergence criterion for reduced move)

Print new point after reduced move .

Call DFCALC to calculate the deviation function for a new set of system variables.

Call CONVER to check for convergence using nonlinear information.

Call CONACC to accumulate constraints if current vector is feasible.

Figure III. 1: General Flowchart of ALPMOD Subroutine

Note about some variables in ALPMOD:

1) **REDMOV** real reduced move step sizes

   Variable REDMOV is defined as a real variable(reduced move step sizes), But it is considered as a vector with length MDESV (maximum number of design variables) in the subroutine.

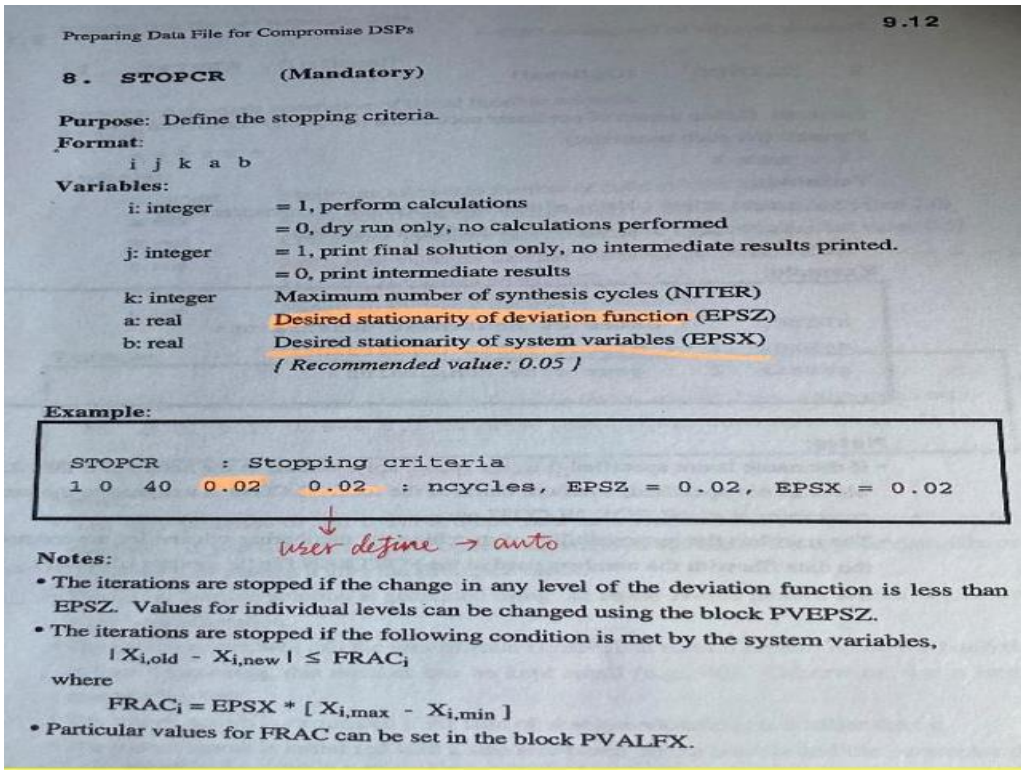
2) **FRACX** real vector: convergence criteria for X, the design variables

3) **FRACZ** real vector: convergence criteria for Z, the deviation function

   The FRACX vector has n dimensions, similar to the dimensionality of the vector of variables X. If, in any one or more dimensions, the difference between X_i ($1 <= i <= n$) is less than 0.005 (or 0.5%), then we stop the program. Therefore, FRACX probably means "fraction of X," so it is a percentage of the absolute value of X_i.

   However, in DSIDES Manual Chapter 9, Section 9.12 (8. STOPCR) (Figure 3.2), the definition is EPSZ (desired stationarity of deviation function) and EPSX (desired stationarity of system variables). EPSZ is not a vector, as the deviation function always yields a single value. Thus, if the difference in Z between two consecutive iterations is no more than 0.02, the program will terminate. However, EPSX, in my understanding, is a vector; if we set it as 0.02, it means when any X_i (one dimension of vector X) in two iterations in a row is not more different than 0.02, then we stop the program.

   The difference between FRACX and EPSX is that FRACX is the real fractional difference (%), but EPSX is a threshold. For example, see the Notes in Figure 3.2. the program will stop if the real fractional difference is less than or equal to the threshold.


Preparing Data File for Compromise DSPs 9.12

8. STOPCR (Mandatory)

Purpose: Define the stopping criteria.

Format:

i j k a b

Variables:

i: integer	= 1, perform calculations
	= 0, dry run only, no calculations performed
j: integer	= 1, print final solution only, no intermediate results printed.
	= 0, print intermediate results
k: integer	Maximum number of synthesis cycles (NITER)
a: real	Desired stationarity of deviation function (EPSZ)
b: real	Desired stationarity of system variables (EPSX)
	{ Recommended value: 0.05 }

Example:

```
STOPCR    : Stopping criteria
1 0  40  0.02   0.02   : ncycles, EPSZ = 0.02, EPSX = 0.02
```

user define → auto

Notes:

- The iterations are stopped if the change in any level of the deviation function is less than EPSZ. Values for individual levels can be changed using the block PVEPSZ.
- The iterations are stopped if the following condition is met by the system variables,

$$| X_{i,old} - X_{i,new} | \leq FRAC_i$$

where

$$FRAC_i = EPSX * [X_{i,max} - X_{i,min}]$$

- Particular values for FRAC can be set in the block PVALFX.

Figure III. 2: Block STOPCR in the DAT file

Following the discussion and representation of the ALPMOD flowchart, the subsequent section is about ALPUTL, which is denoted as ALPUTL in the files but referred to as COPYTX in all other subroutines.

III.2.4 COPYTX (ALPUTL) Subroutine

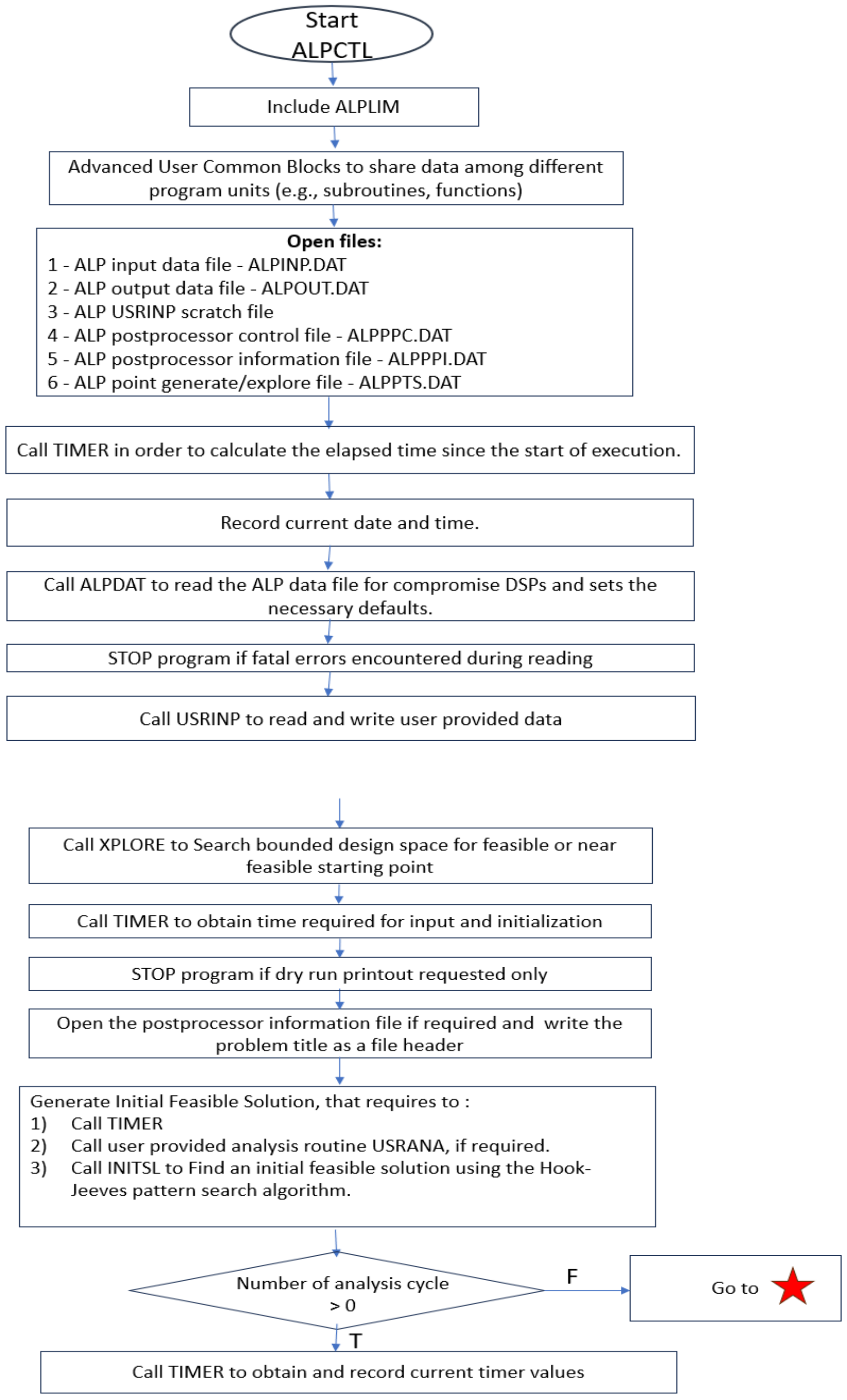
The main purpose of this subroutine is to copy the contents of the Y (input vector) array into the X (output vector) array. This subroutine is about the synthesis cycle. So it keeps the information of the last cycle and uses it to start the next cycle. For instance, if the user wants to do 20 cycles and now, we are in the 10th cycle, it will save the information on the 10th cycle in an array and pass it to the program to start the 11th cycle.

```
C Subroutine COPYTX ( ALPUTL)
C
```

```
C Purpose: This routine copies the contents of the Y array into the X
array.
C
C-------------------------------------------------------------------
--
C Arguments       Name     Type   Description
C ---------       ----     ----   -----------
C Input:          N        int    length of vectors
C                 Y        real   input vector
C
C Output:         X        real   output vector
C
C Input/Output:   none
C
C*******************************************************************
**
C-
      SUBROUTINE COPYTX ( N, X, Y)
C
      INTEGER N, K
      REAL X(N), Y(N)
C
      DO 100 K = 1, N
         X(K) = Y(K)
  100 CONTINUE
C
      RETURN
```

III.2.5 ALPCTL Subroutine and Its Flowchart

This subroutine is the main program for DSIDES and is a central module that other subroutines will invoke. By comprehending this subroutine, users can gain a clearer insight into the overall functionality and operation of the program. The flowchart and outline of the sequence of execution for the ALPCTL subroutine has been displayed in Figure III.3, and the subroutine is in the appendix since it is a long subroutine.


Start ALPCTL
Include ALPLIM
Advanced User Common Blocks to share data among different program units (e.g., subroutines, functions)
Open files:
1 - ALP input data file - ALPINP.DAT
2 - ALP output data file - ALPOUT.DAT
3 - ALP USRINP scratch file
4 - ALP postprocessor control file - ALPPPC.DAT
5 - ALP postprocessor information file - ALPPPI.DAT
6 - ALP point generate/explore file - ALPPTS.DAT
Call TIMER in order to calculate the elapsed time since the start of execution.
Record current date and time.
Call ALPDAT to read the ALP data file for compromise DSPs and sets the necessary defaults.
STOP program if fatal errors encountered during reading
Call USRINP to read and write user provided data
Call XPLORE to Search bounded design space for feasible or near feasible starting point
Call TIMER to obtain time required for input and initialization
STOP program if dry run printout requested only
Open the postprocessor information file if required and write the problem title as a file header
Generate Initial Feasible Solution, that requires to :
1) Call TIMER
2) Call user provided analysis routine USRANA, if required.
3) Call INITSL to Find an initial feasible solution using the Hook-Jeeves pattern search algorithm.
Number of analysis cycle > 0
F
Go to
T
Call TIMER to obtain and record current timer values

Establish nonlinear characteristics of design variables, that requires to:

1) Call USRANA
2) Call DFCALC to Calculate the deviation function for a new set of system variables.
3) Call COPYTX (six times) to copy the values into corresponding current best design vectors

Perform Synthesis cycles that requires to:

1) Call ALPMOD to perform the synthesis cycles.
2) Returned DESVAR (vector of design variables) corresponds with best nonlinear synthesis cycle solution.
3) Record best results in arrays
4) Record previous analysis cycle results in.
5) Identify best analysis and corresponding synthesis number

Calculate characteristics of new DESVAR (vector of design variables)

1) Call USRANA for user provided analysis routine.
2) Call DFCALC to Calculate the deviation function for a new set of system variables.

1) Test for convergence using nonlinear information

- Compare DESVST(Local variable) and DESVAR (vector of design variables)
- CALL CONVER to Check for convergence.
- Write DESVAR data into DESVST if DESVAR better than DESVST

2) Build Z vectors for DESVST, DESVX1 and DESVAR (See ALPMOD, build Z vectors for X*, X1 and X2)

Call COPYTX to Copy current synthesis results into previous result arrays.

Print analysis cycle output if requested

1) Design variables (if LPROUT(1) is true)
2) Deviation variables (if LPROUT(2) is true)
3) Deviation function (if LPROUT(3) is true)

Print message 'End analysis/synthesis cycle number' and obtain timer results for current analysis cycle

Print final analysis/synthesis cycle output – FINAL SOLUTION header information

Print best analysis/synthesis cycle output - BEST SOLUTION header information

SYNTHESIS CYCLES ONLY

1) CALL TIMER to Obtain and record current timer values.
2) CALL FORAGEMV to solve the discrete problem.
3) Call ALPMOD toto perform the synthesis cycles.
4) Returned DESVAR corresponds with best nonlinear solution
5) Check for discrete number of synthesis cycles complete
6) Obtain timer results for current analysis cycle.
7) Print final/best synthesis cycle output header information

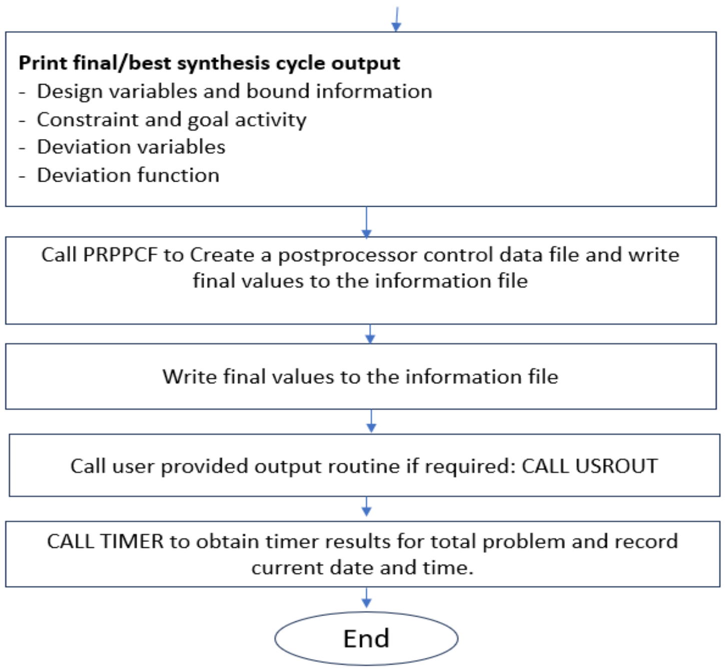

Print final/best synthesis cycle output
- Design variables and bound information
- Constraint and goal activity
- Deviation variables
- Deviation function
Call PRPPCF to Create a postprocessor control data file and write final values to the information file
Write final values to the information file
Call user provided output routine if required: CALL USROUT
CALL TIMER to obtain timer results for total problem and record current date and time.
End

Figure III. 3: Flowchart of ALPCTL Subroutine

Summary OF Part 3:

In this part, we have covered the main input subroutines and also reviewed all the subroutines and their purpose. This part could be helpful for anyone who has a plan to improve and consider changes in the program in the future. All the subroutines are located on the server of the System Realization Laboratory in OU. (Following address in the server of SRL lab : C:\Sara_DSIDES)

Summary of Thesis

The summary section of my thesis comprises five distinct components: the preamble, an introduction and an overview of the three main parts, a thesis layout flowchart, and closing remarks.

1: Preamble:

When dealing with complex systems, we need to consider that these systems have behaviors that are hard to predict or control, and uncertainties are always present since computational models are abstracts of reality. It is recognized that in many situations, it may not be possible to simultaneously optimize all objectives due to inherent conflicts, resource limitations, or uncertainty. Also, as George E.P. Box said: "All models are wrong, but some are useful." The consequences of these observations are significant. We need to accept that our models might not capture everything and that uncertainties are a part of the picture. Hence, we must accept and deal with uncertainty instead of ignoring it and find solutions that are relatively insensitive to the uncertainties.

When choosing a method to work with, we need to consider the quality and the amount of our data. To make this all work, we need a method to find solutions that achieve a reasonable compromise or balance among the objectives and identify a set of solutions that are relatively insensitive to uncertainties. Also, be able to facilitate the exploration of solution space to support human decision-making. This ties into the problems we face in supporting decisions for complex systems. These problems involve choosing between options and making compromises.

The compromise Decision Support Problem (cDSP) construct, and the Adaptive Linear Programming algorithm has been developed as a result, which was first introduced by Mistree and co-authors (1993). It is a domain-independent, multiobjective decision model based on mathematical and goal programming. They effectively deal with multiobjective problems involving bounds, linear and nonlinear constraints, goals, and consisting of Boolean and continuous variables. The requirements for this construct are:

1) Identify a set of solutions that are relatively insensitive to uncertainties.

2) Facilitate the exploration of solution space to support human decision-making.

Why do We Use Satisfying Strategy?

We choose a satisfying strategy because it is more flexible and realistic and we are able to manage uncertainty to give robust design. The fact that our math isn't perfect is acknowledged by the satisficing strategy, but it is still capable of assisting us in crafting robust designs. In contrast, the optimizing strategy is based on the assumption that math is flawless and necessitates the fulfillment of specific conditions, which may not always align with circumstances.

In the context of the difference between optimization and satisficing in terms of assumptions regarding the KKT conditions, we present Summary Figure 1.

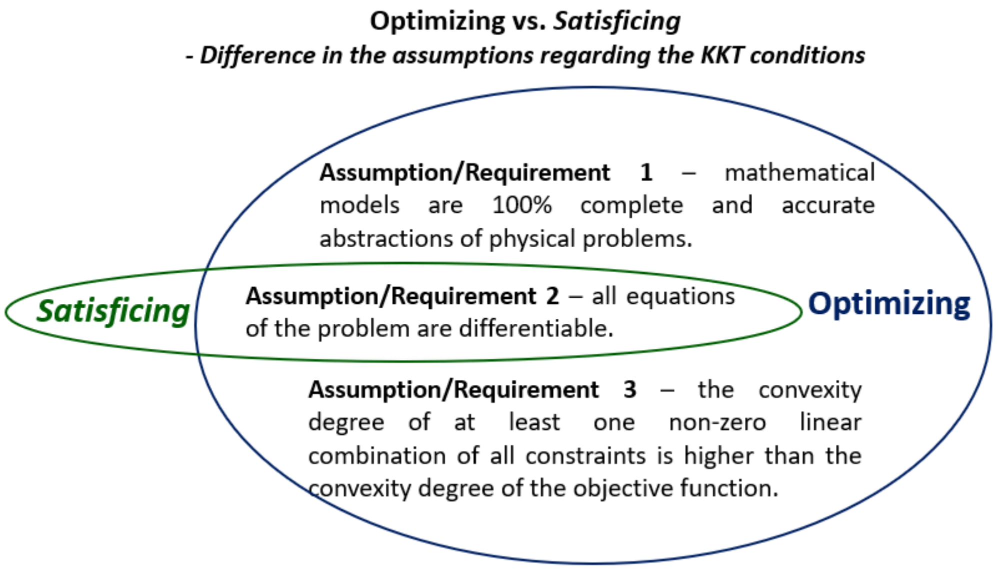
Gradient based optimization means we must reach both necessary and sufficient KKT conditions, which contain 3 different assumptions for optimizing strategy:

1) The mathematical models are 100% complete and accurate abstraction of physical problems. (Only using in optimization). It is like thinking our math is always right, even though we know it is not always true. With optimizing, when we find the best solution, we believe it'll work perfectly in any problem, any time, any situation, because our math is perfect.

2) All equations of the problems are differentiable. (Common for satisfying strategy and optimizing strategy). This condition is integral to mathematical programming, as it necessitates the utilization of first or second-order derivatives to facilitate solution derivation.

In essence, the need for differentiability across all equations in the problem is underscored by this assumption.

3) the convexity degree of at least one nonzero linear combination of all constraints is higher than the convexity degree of objective function. (Only using in optimization). This specific assumption becomes relevant when an optimizing approach is chosen, indicating that a higher level of convexity than the primary objective function is exhibited by certain constraints.


Optimizing vs. *Satisficing*
- Difference in the assumptions regarding the KKT conditions
Assumption/Requirement 1 – mathematical models are 100% complete and accurate abstractions of physical problems.
Satisficing
Assumption/Requirement 2 – all equations of the problem are differentiable.
Optimizing
Assumption/Requirement 3 – the convexity degree of at least one non-zero linear combination of all constraints is higher than the convexity degree of the objective function.

Summary Figure 1: Comparison of Optimization and Satisficing Assumptions Regarding KKT Conditions

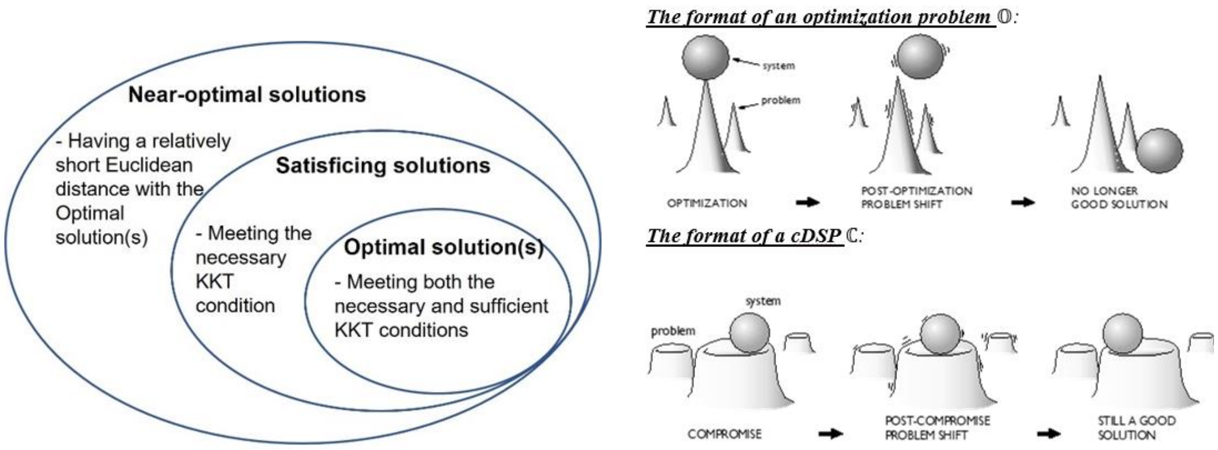
Relation Between The Optimal, Satisficing, And Near-Optimal Solutions:

As it is represented in Summary Figure 2 , in the quest for an optimal solution, the use of KKT conditions is crucial, demanding adherence to both necessary and sufficient KKT conditions. In this, the satisfaction of the first-order conditions and the Lagrange function is entailed, with these being key components. Additionally, there are sufficient KKT conditions, represented in the second-order Lagrange function, which must be met.

In contrast, satisfying solutions remain solely to the necessary KKT conditions, typically represented by the first Lagrange equation. Another concept, frequently encountered in evolutionary and generic algorithms, is the 'near-optimal' solution. These solutions, although falling short of optimality, exhibit minimal distance from the optimal solution. The exact definition of 'near' optimal varies depending on the context.

Interestingly, the encompassing of optimal solutions by satisfying solutions can occur, as the sufficient KKT conditions may be unintentionally met by some satisfying solutions. Typically, these solutions are often outperformed by most near-optimal ones due to their proximity to the optimal solution. Consequently, the concept of 'good enough' is defined.

In contrast, the objective of optimization is to reach the peak of a precise model. However, it is important to know that sometimes the model itself can be flawed.Now, in the context of the satisfying strategy, as implemented in cDSP, we shift our focus to a flat region. It is not a pinpoint spot but more like an area. Our aim is to consistently include this 'ball' within that flat area. This approach ensures that we always have a solution that works and is practical.


Near-optimal solutions
- Having a relatively short Euclidean distance with the Optimal solution(s)
Satisficing solutions
- Meeting the necessary KKT condition
Optimal solution(s)
- Meeting both the necessary and sufficient KKT conditions
The format of an optimization problem 𝕆:
system
problem
OPTIMIZATION
POST-OPTIMIZATION PROBLEM SHIFT
NO LONGER GOOD SOLUTION
The format of a cDSP ℂ:
system
problem
COMPROMISE
POST-COMPROMISE PROBLEM SHIFT
STILL A GOOD SOLUTION

Summary Figure 2: Relation between the optimal, satisficing, and near-optimal solutions

A summary of the unique features of satisficing in terms of its application to managing engineering design problems at each step has been presented in Summary Table 1.

Summary Table 1: Summary of Satisficing in Managing Engineering Design Problems at Each Step

Stage	**Feature**	**Advantage**

Formulation	Using Goals and Minimizing Deviation Variables Instead of Objectives	At a solution point, only the necessary KKT conditions are met, whereas the sufficient KKT conditions do not have to be met. Therefore, designers have a higher chance of finding a solution and a lower chance of losing a solution due to parameterizable and unparameterizable uncertainties.
Approximation	Using second-order sequential linearization	Designers can have a balance between linearization accuracy and computational complexity.
	Using accumulated linearization	Designers can manage nonconvex problems in a way, and deal with highly convex, nonlinear problems relatively more accurately.
Exploration	Combining interior-point searching and vertex searching	Designers can avoid being stuck into local optimum to some extent and identify satisficing solutions relatively insensitive to starting points changing.
Evaluation	Allowing some violations of soft requirements, such as the bounds of deviation variables	Designers can manage rigid requirements and soft requirements in different ways to ensure feasibility. As a result, goals and constraints with different scale can be managed

Mistree and co-authors designed a computer program to implement cDSP construct. It has been written in FORTRAN to identify robust satisficing solutions to design problems when the models are abstractions of reality. It is called DSIDES (Decision Support in the Design of Engineering Systems).

DSIDES is a software tool developed to help engineers and designers make better decisions in the design of complex engineering systems and provides decision support for the design of complex engineering systems.

In this thesis, our primary objective is to enhance the accessibility and user-friendliness of DSIDES by designing a user-friendly wrapper. Three key areas of focus are included in this thesis:

4) **Exploration of cDSP Construct:** In this part, the examination of the cDSP (Compromise Decision Support Problem) construct, including its structural components and the formulation of problem statements within the cDSP framework, has been discussed.
5) **Comprehensive Analysis of the DSIDES Wrapper**: A detailed exploration of the DSIDES wrapper and a step-by-step walkthrough of the wrapper's functionalities are covered.
6) **DSIDES Software Program Manuals:** Program manuals for DSIDES software are provided. These manuals are helpful resources for individuals seeking to enhance, expand, or modify the software.

2: Introduction and Overview of Three Parts

As mentioned earlier, a complex system has emergent properties that cannot be predicted or controlled, and uncertainties are always present since computational models are abstracts of reality; they cannot eliminate uncertainties that are a significant factor in any design problem. Therefore, we need to accept the incompleteness of the models, manage the uncertainty embodied therein, and identify a set of solutions that are relatively insensitive to uncertainties. Also, it could facilitate the exploration of solution space to support human decision-making. The compromise Decision Support Problem(cDSP) and the Adaptive Linear Programming algorithm have been developed, which was first introduced by Mistree and co-authors (1993).

They also designed a computer program to implement cDSP construct. It has been developed in FORTRAN to identify robust satisficing solutions to design problems when the models are abstractions of reality. It is called DSIDES (Decision Support in the Design of Engineering Systems).

Our focus in this thesis is on designing a user-friendly interface for DSIDES, providing information on the software, the necessary information to work with it effectively, and program manuals to improve the software for future use. Based on that, there are three different parts to this thesis:

4) **Part One: DSIDES Software and cDSP Construct: An Introduction.**
5) **Part Two: Designing the User-Friendly Wrapper for DSIDES.**
6) **Part Three: Program Manuals and Improvement of DSIDES**

Overview of Part 1: DSIDES Software and cDSP Construct: An Introduction.

In the first part of this thesis, enhanced information about the compromise Decision Support Problem (cDSP) and formulation of a problem in cDSP, including Archimedean and preemptive forms in detail with some examples, a short description about DSIDES software and platform of DSIDES are provided. Upon reading the first part, the reader will have learned the necessary information to formulate a problem in cDSP, allowing them to start using the software effectively.

Overview of Part 2: Designing the User-Friendly Wrapper for DSIDES

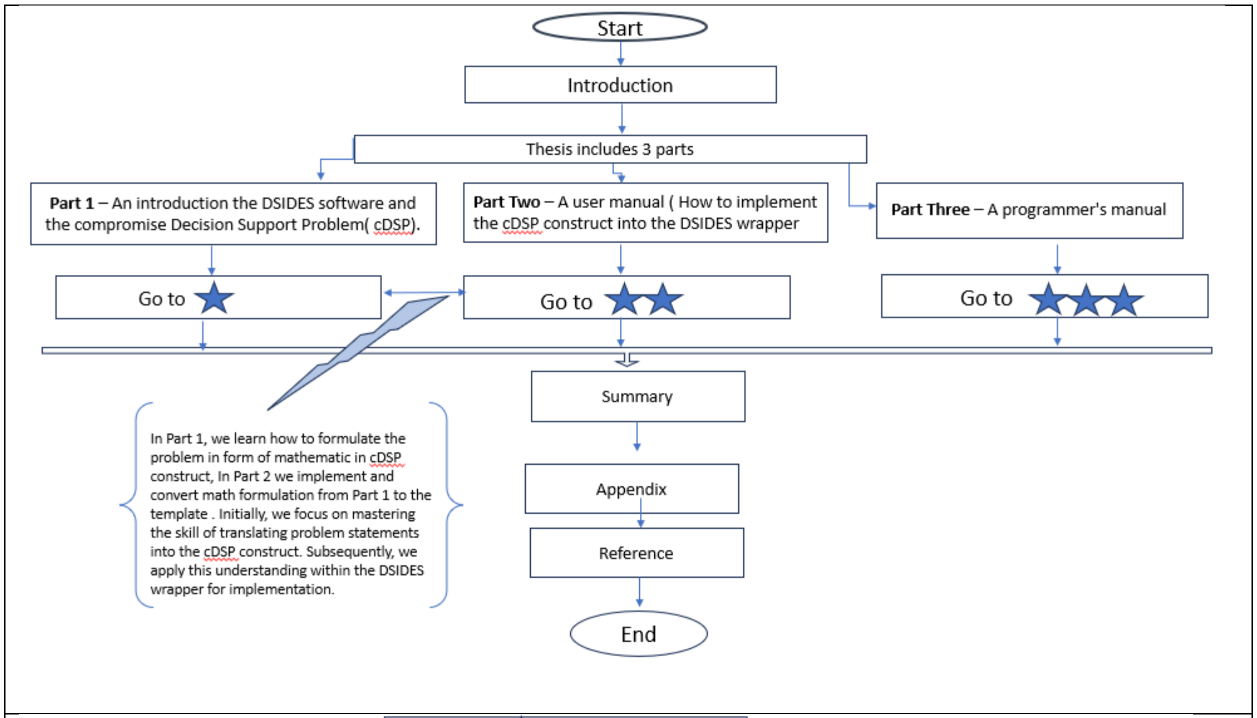
In the second part of this thesis, the main focus is on the user-friendly interface that has been designed for DSIDES. In this part, detailed information about how to work with the wrapper effectively, including implementing the formulated cDSP construct into the DSIDES wrapper to find solutions that are relatively insensitive to uncertainty, is provided. This section also includes an interpretation of the output results and verification and validation for the software. To this end, three problems are presented that demonstrate how to start from scratch, formulate the problem in cDSP, and finally implement the problem in the wrapper. This Section is a useful document for users to practice and learn how to use the DSIDES wrapper effectively.

Overview of Part 3: Program Manuals and Improvement of DSIDES

In the third part of this thesis, the program manuals, flowcharts, coding, and information about input subroutines in DSIDES are provided. The main focus of this part is on providing information about the input files and their respective purposes. The objective is to simplify their comprehension of the users by presenting their flowcharts. Additionally, other subroutines that are called through the main files will be covered in this part. This information is necessary for users who want to improve the software and understand how the programs work. In this part, readers will have access to the necessary information to improve the software and their work in the future. (The focal objective of this Section is to consolidate all the functions and subroutines of DSIDES in a single location for documentation purposes.)

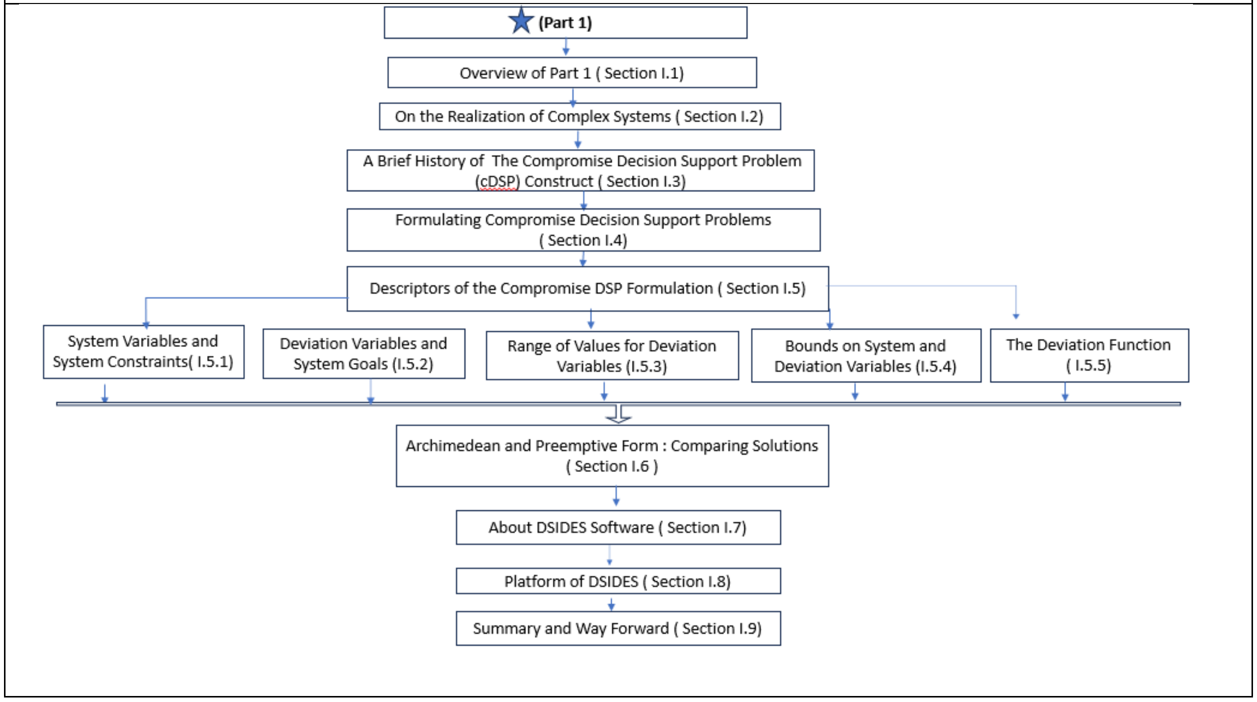
3: Thesis Layout Flowchart

An understanding of the organization and flow of the thesis is offered by an illustrative representation in Figure A, in which its sequential structure is outlined. A concise overview of the chapters, sections, and their interconnections is provided within this illustrative representation to enhance clarity and facilitate navigation.

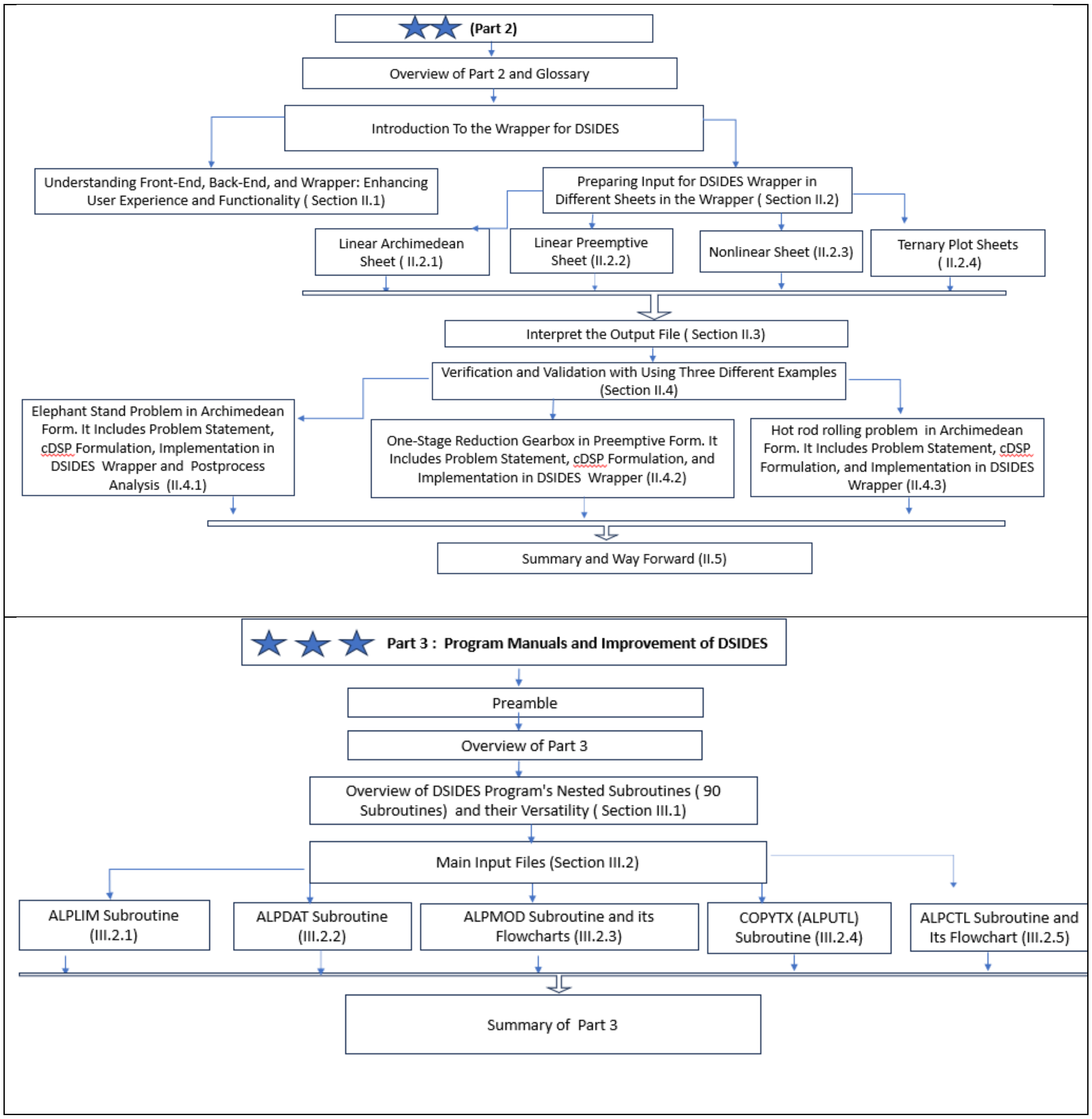


Start
Introduction
Thesis includes 3 parts
Part 1 – An introduction the DSIDES software and the compromise Decision Support Problem(cDSP).
Part Two – A user manual (How to implement the cDSP construct into the DSIDES wrapper
Part Three – A programmer's manual
Go to ★
Go to ★★
Go to ★★★
Summary
Appendix
Reference
End

In Part 1, we learn how to formulate the problem in form of mathematic in cDSP construct, In Part 2 we implement and convert math formulation from Part 1 to the template . Initially, we focus on mastering the skill of translating problem statements into the cDSP construct. Subsequently, we apply this understanding within the DSIDES wrapper for implementation.



★ (Part 1)
Overview of Part 1 (Section I.1)
On the Realization of Complex Systems (Section I.2)
A Brief History of The Compromise Decision Support Problem (cDSP) Construct (Section I.3)
Formulating Compromise Decision Support Problems (Section I.4)
Descriptors of the Compromise DSP Formulation (Section I.5)
System Variables and System Constraints(I.5.1)
Deviation Variables and System Goals (I.5.2)
Range of Values for Deviation Variables (I.5.3)
Bounds on System and Deviation Variables (I.5.4)
The Deviation Function (I.5.5)
Archimedean and Preemptive Form : Comparing Solutions (Section I.6)
About DSIDES Software (Section I.7)
Platform of DSIDES (Section I.8)
Summary and Way Forward (Section I.9)


(Part 2)
Overview of Part 2 and Glossary
Introduction To the Wrapper for DSIDES
Understanding Front-End, Back-End, and Wrapper: Enhancing User Experience and Functionality (Section II.1)
Preparing Input for DSIDES Wrapper in Different Sheets in the Wrapper (Section II.2)
Linear Archimedean Sheet (II.2.1)
Linear Preemptive Sheet (II.2.2)
Nonlinear Sheet (II.2.3)
Ternary Plot Sheets (II.2.4)
Interpret the Output File (Section II.3)
Verification and Validation with Using Three Different Examples (Section II.4)
Elephant Stand Problem in Archimedean Form. It Includes Problem Statement, cDSP Formulation, Implementation in DSIDES Wrapper and Postprocess Analysis (II.4.1)
One-Stage Reduction Gearbox in Preemptive Form. It Includes Problem Statement, cDSP Formulation, and Implementation in DSIDES Wrapper (II.4.2)
Hot rod rolling problem in Archimedean Form. It Includes Problem Statement, cDSP Formulation, and Implementation in DSIDES Wrapper (II.4.3)
Summary and Way Forward (II.5)
Part 3 : Program Manuals and Improvement of DSIDES
Preamble
Overview of Part 3
Overview of DSIDES Program's Nested Subroutines (90 Subroutines) and their Versatility (Section III.1)
Main Input Files (Section III.2)
ALPLIM Subroutine (III.2.1)
ALPDAT Subroutine (III.2.2)
ALPMOD Subroutine and its Flowcharts (III.2.3)
COPYTX (ALPUTL) Subroutine (III.2.4)
ALPCTL Subroutine and Its Flowchart (III.2.5)
Summary of Part 3

Summary Figure 3: Flowchart of Thesis Layout

After providing an overview of the thesis layout through the flowchart presentation, the detailed table of contents is presented in the subsequent section to enumerate all sections and subsections.

5: Closing Remarks

Our goal in this thesis is the development of a user-friendly Wrapper for DSIDES software, with a focus on addressing the challenges posed by uncertainty and complexity in design problems. Within the field of engineering design, a significant contribution is sought through the integration of MATLAB and Excel to facilitate the exploration of the solution space, supporting human decision-making.

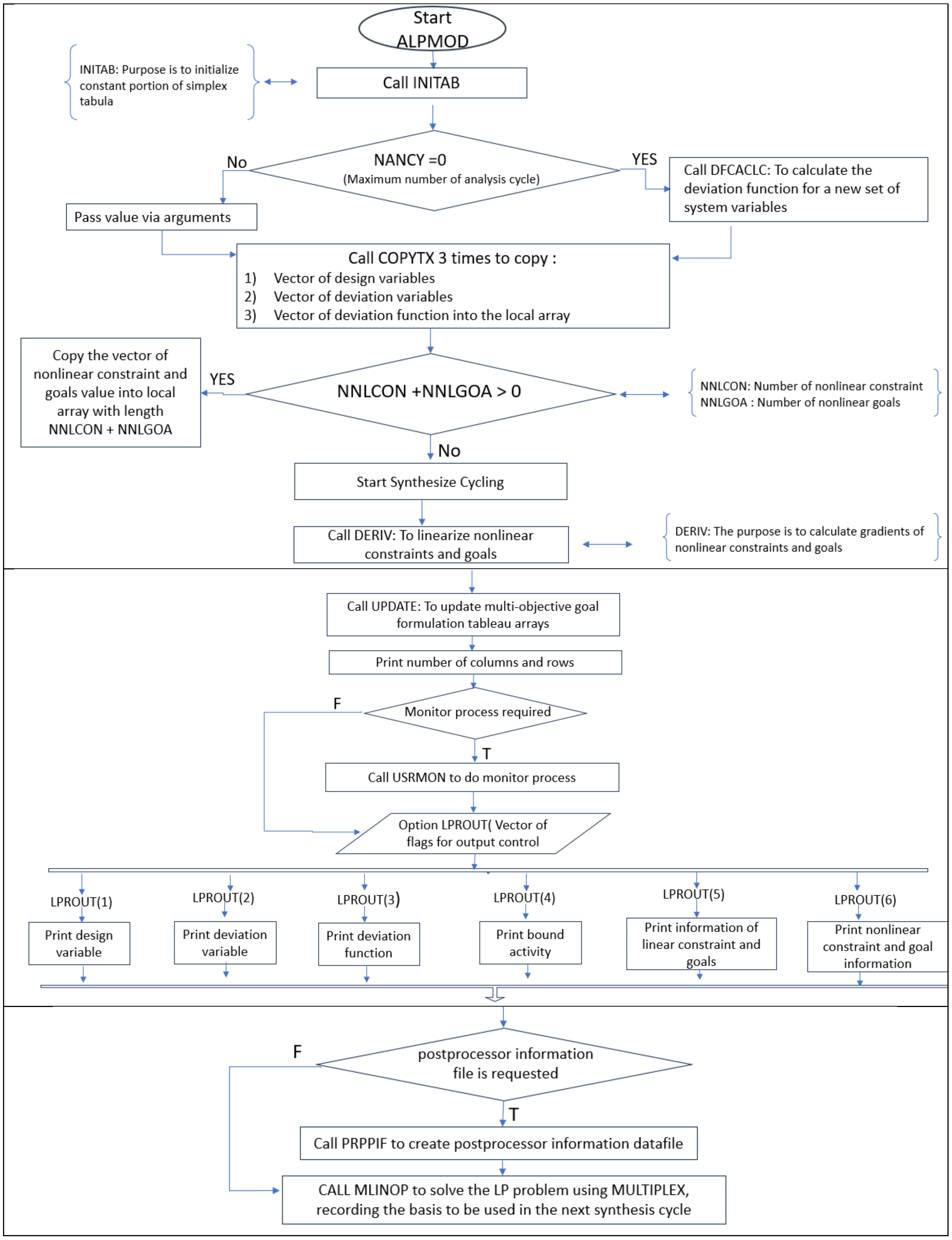
Multiobjective problems, which encompass bounds, linear and nonlinear constraints, goals, and a combination of Boolean and continuous variables, are effectively managed by the software. Substantial impact is expected, encompassing improvements in efficiency, increased productivity, and an enhanced user experience in engineering design. Furthermore, validation of the software's performance and functionality, utilizing both empirical and theoretical methods, ensures reliability and efficiency. In summary, this thesis's contribution lies in the provision of a practical and efficient solution to tackle the challenges associated with uncertainty and complexity in engineering design.

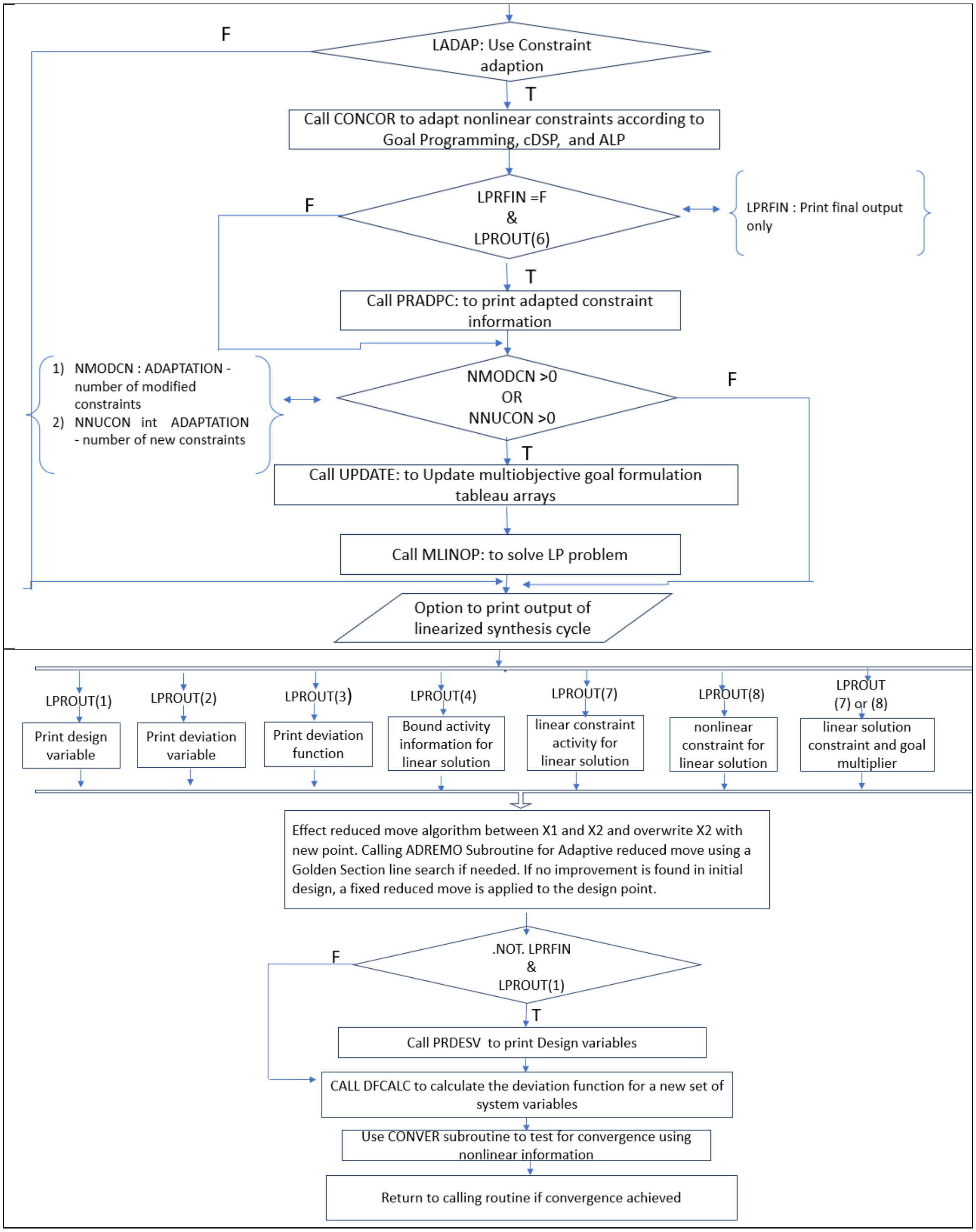
Appendix

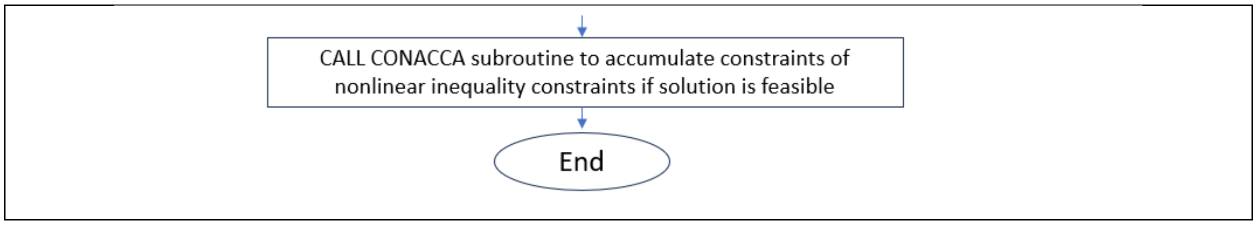
ALPMOD Subroutine:

For this subroutine, one detailed flowchart and the program are added here.

Detailed Flowchart of ALPMOD:


Start ALPMOD
INITAB: Purpose is to initialize constant portion of simplex tabula
Call INITAB
No
NANCY =0 (Maximum number of analysis cycle)
YES
Call DFCACLC: To calculate the deviation function for a new set of system variables
Pass value via arguments
Call COPYTX 3 times to copy : 1) Vector of design variables 2) Vector of deviation variables 3) Vector of deviation function into the local array
Copy the vector of nonlinear constraint and goals value into local array with length NNLCON + NNLGOA
YES
NNLCON +NNLGOA > 0
NNLCON: Number of nonlinear constraint
NNLGOA : Number of nonlinear goals
No
Start Synthesize Cycling
Call DERIV: To linearize nonlinear constraints and goals
DERIV: The purpose is to calculate gradients of nonlinear constraints and goals
Call UPDATE: To update multi-objective goal formulation tableau arrays
Print number of columns and rows
F
Monitor process required
T
Call USRMON to do monitor process
Option LPROUT(Vector of flags for output control
LPROUT(1)
LPROUT(2)
LPROUT(3)
LPROUT(4)
LPROUT(5)
LPROUT(6)
Print design variable
Print deviation variable
Print deviation function
Print bound activity
Print information of linear constraint and goals
Print nonlinear constraint and goal information
F
postprocessor information file is requested
T
Call PRPPIF to create postprocessor information datafile
CALL MLINOP to solve the LP problem using MULTIPLEX, recording the basis to be used in the next synthesis cycle


LADAP: Use Constraint adaption
F
T
Call CONCOR to adapt nonlinear constraints according to Goal Programming, cDSP, and ALP
LPRFIN =F & LPROUT(6)
F
T
LPRFIN : Print final output only
Call PRADPC: to print adapted constraint information
1) NMODCN : ADAPTATION - number of modified constraints
2) NNUCON int ADAPTATION - number of new constraints
NMODCN >0 OR NNUCON >0
F
T
Call UPDATE: to Update multiobjective goal formulation tableau arrays
Call MLINOP: to solve LP problem
Option to print output of linearized synthesis cycle
LPROUT(1)
Print design variable
LPROUT(2)
Print deviation variable
LPROUT(3)
Print deviation function
LPROUT(4)
Bound activity information for linear solution
LPROUT(7)
linear constraint activity for linear solution
LPROUT(8)
nonlinear constraint for linear solution
LPROUT (7) or (8)
linear solution constraint and goal multiplier
Effect reduced move algorithm between X1 and X2 and overwrite X2 with new point. Calling ADREMO Subroutine for Adaptive reduced move using a Golden Section line search if needed. If no improvement is found in initial design, a fixed reduced move is applied to the design point.
.NOT. LPRFIN & LPROUT(1)
F
T
Call PRDESV to print Design variables
CALL DFCALC to calculate the deviation function for a new set of system variables
Use CONVER subroutine to test for convergence using nonlinear information
Return to calling routine if convergence achieved


CALL CONACCA subroutine to accumulate constraints of nonlinear inequality constraints if solution is feasible
End

Summary Figure 4: Detailed Flowchart of ALPMOD

ALPMOD Subroutine:

```
+
C**********************************************************************
**
C
C Subroutine ALPMOD
C
C Purpose:  Perform the synthesis cycles
C
C----------------------------------------------------------------------
--
C Arguments       Name     Type   Description
C ---------       ----     ----   -----------
C Input:          NANCY    int    maximum number of analysis cycles
C                                 performed
C                 INUMAN   int    analysis cycle number
C                 NTITER   int    total number of iterations performed
C                 NUOUT    int    unit number of output data file
C                 NUPPI    int    unit number of postprocessor
C                                 information data file
C                 NRELV    int    number of real (continuous)
variables
C                 NVINT    int    number of integer variables
C                 NVSEL    int    number of selection (Boolean)
C                                 variables
C                 NDESV    int    number of design variables
C                 NDEVAR   int    number of deviation variables
C                 NLINCO   int    number of linear constraints
C                 NLINGO   int    number of linear goals
C                 NNLINQ   int    number of nonlinear inequalities
C                                 constraints
C                 NNLEQU   int    number of nonlinear equalities
C                                 constraints
C                 NNLCON   int    number of nonlinear constraints
C                 NNLGOA   int    number of nonlinear goals
```

```
C                 NNLTOT   int    total number of nonlinear
constraints
C                                 and goals
C                 NMPRI    int    number of goal priority levels
C                 NSYCY    int    vector indicating the maximum number
C                                 of synthesis cycles performed within
C                                 each analysis cycle
C                 IACTVR   int    vector of flags for active variables
C                 IADCON   int    vector of flags for admissible
C                                 nonlinear constraints
C                                 = 0  if suppressed by user
C                                 = J  if admissible
C                                 = -n if suppressed by program
C                 LISIGN   int    vector indicating sign for linear
C                                 inequality constraints
C                 JSYCY    int    synthesis cycle number corresponding
C                                 to X* (within a given analysis
cycle)
C                 NADREM   int    ADREMO - maximum number of calls to
C                                 be made to GCALC
C                 CONDEV   real   total constraint violation
C                 DELREM   real   acceptable convergence criterion for
C                                 reduced move
C                 COFLIN   real   matrix of coefficients of linear
C                                 constraints and goals
C                 DESVAR   real   vector of design variables
C                 DFNCOF   real   matrix of weights for deviation
C                                 function
C                 DEVFUN   real   vector of deviation function values
C                 DEVVAR   real   vector of deviation variables
C                 FRACX    real   vector: convergence criteria for X,
C                                 the design variables
C                 FRACZ    real   vector: convergence criteria for Z,
C                                 the deviation function
C                 GVAL     real   vector of nonlinear constraint and
C                                 goal values
C                 PESTEP   real   vector of perturbation steps for
C                                 nonlinear constraints and goals
C                 REDMOV   real   reduced move step sizes
C                 RHSLIN   real   vector of RHS values for linear
C                                 constraints and goals
C                 VBOUNS   real   matrix of lower and upper bounds for
C                                 design variables
C                 VILCN    real   vector of acceptable nonlinear
C                                 constraint violations
C                 LADAP    lgcl   use constraint adaptation
```

```
C                  LADREM   lgcl   use adaptive reduced move
C                  LMON     lgcl   call user supplied subroutine USRMON
C                  LVCOF    lgcl   call user supplied subroutine USRLIN
C                  LPRFIN   lgcl   print final output only
C                  LPROUT   lgcl   vector of flags for output control
C                  LPPROC   lgcl   create ALP postprocessor files
C                  LCONDF   lgcl   TRUE if deviation function converged
C                  LCONSV   lgcl   TRUE if design variables converged
C                  LIMPRV   lgcl   TRUE if point 2 is better than point
1
C                  IDDESV   chr6   vector of design variable names
C                  IDDEVR   chr6   vector of deviation variable names
C                  IDLICO   chr6   vector of linear constraint names
C                  IDLIGO   chr6   vector of linear goal names
C                  IDNLCO   chr6   vector of nonlinear constraint names
C                  IDNLGO   chr6   vector of nonlinear goal names
C
C Output:
C
C Input/Output:
C
C----------------------------------------------------------------------
--
C Common Blocks:  none
C
C Include Files:  alplim.cmm
C
C Calls to:
C----------------------------------------------------------------------
--
C Development History
C
C Author:  Warren Smith
C Date:    November 15, 1991
C
C Modifications:  Bert Bras, July 13, 1993.
C                 Added NRFIX for INITAB() and MLINOP().
C                 Added MODIF() for PRNLNC().
C
C**********************************************************************
**
C-NVINT---KL** Watch changing all NVINT to NDISV
C                  NVINT    int    number of integer variables
C *************KL NDISV    int    number of discrete variables (inc.
integer)
```

```
C
      SUBROUTINE ALPMOD (NANCY, INUMAN, NTITER, NUOUT, NUPPI,
     &                   NRELV, NDISV, NVSEL, NDESV, NDEVAR,
     &                   NLINCO, NLINGO,
     &                   NNLINQ, NNLEQU, NNLCON, NNLGOA, NNLTOT,
     &                   NMPRI, NSYCY, IACTVR, IADCON, LISIGN,
     &                   JSYCY, NADREM,
     &                   CONDEV, DELREM,
     &                   COFLIN, DESVAR, DFNCOF,
     &                   DEVFUN, DEVVAR, FRACX, FRACZ, GVAL,
     &                   PESTEP, REDMOV, RHSLIN, VBOUNS, VILCN,
     &                   LADAP, LADREM, LMON, LVCOF, LPRFIN, LPROUT,
     &                   LPPROC, LCONDF, LCONSV, LIMPRV,
     &                   IDDESV, IDDEVR, IDLICO, IDLIGO, IDNLCO,
IDNLGO)
C
      INCLUDE 'alplim.cmm'
C
C---------------------------------------
C     Arguments:
C---------------------------------------
C
      INTEGER NANCY, INUMAN, NTITER, NUOUT, NUPPI,
     &     NRELV, NDISV, NVINT, NVSEL, NDESV, NDEVAR,
     &     NLINCO, NLINGO, NNLINQ, NNLEQU, NNLCON, NNLGOA, NNLTOT,
     &     NMPRI, NSYCY, IACTVR(MDESV), IADCON(MNLNCG),
LISIGN(MLINCG),
     &     JSYCY, NADREM
C
      REAL CONDEV, DELREM,
     &     COFLIN(MLINCG,MDESV), DESVAR(MDESV), DFNCOF(MLEVEL,MDEVV),
     &     DEVFUN(MLEVEL), DEVVAR(MDEVV),
     &     FRACX(MDESV), FRACZ(MLEVEL),
     &     GVAL(MNLNCG), PESTEP(MDESV), REDMOV(MDESV),
     &     RHSLIN(MLINCG), VBOUNS(2,MDESV), VILCN(MNLNCG)
C
      LOGICAL LADAP, LADREM, LMON, LVCOF,
     &     LPRFIN, LPROUT(8), LPPROC,
     &     LCONDF, LCONSV, LIMPRV
C
      CHARACTER*6 IDDESV(MDESV), IDDEVR(MDEVV), IDLICO(MLINCG),
     &     IDLIGO(MLINCG), IDNLCO(MNLNCG), IDNLGO(MNLNCG)
C
C---------------------------------------
C     Local variables:
```

```
C--------------------------------------
C
      INTEGER INUMSY, IPATH, J, JJ, KON1, NFIX, NRFIX, NCOL, NROW,
     &        NACCUM, NDEVUS, NDVUSR, NMODCN, NNLCUS, NNLGUS,
     &        NNUCON, NOUT, NPRUSR, NZ, MODIF(MNLNCG),
     &        MODCON(MNLNCG), MODGDX(MNLNCG,MDESV), MODSEQ(MNLNCG),
     &        NEWCON(MNLNCG), NEWDG(MNLNCG,MDESV), NEWSEQ(MNLNCG),
     &        NACCIT(MAXACC), NACCON(MAXACC)
C
      REAL CONVX1, CONVX2, DELTA, DUMCON,
     &     AMATX(MGOLMX,MVARMX), BLO(MVARMX), BUP(MVARMX),
     &     BRHS(MGOLMX), CTWO(MLEVEL,MVARMX), CONVEX(MNLNCG),
     &     DESVX1(MDESV), DESVX2(MDESV),
     &     DEVVX1(MDEVV), DEVVX2(MDEVV),
     &     DFUNX1(MLEVEL), DFUNX2(MLEVEL),
     &     DG(MNLNCG,MDESV), DGTNGT(MNLNCG,MDESV),
     &     D2GDX2(MNLNCG,MDESV),
     &     DGACC(MAXACC,MDESV), DGNEW(MNLNCG,MDESV),
     &     DJMAT(MLEVEL,MVARMX),
     &     DUMDEV(MDEVV), DUMDFN(MLEVEL),
     &     DUMGVL(MNLNCG), DUMVAR(MDESV),
     &     GVALX1(MNLNCG), GVALX2(MNLNCG),
     &     RHS(MNLNCG), RHSACC(MAXACC), RHSNEW(MNLNCG),
     &     XVAL(MVARMX), ZVAL(MLEVEL),
     &     ZSTAR(MLEVEL), Z1(MLEVEL), Z2(MLEVEL)
C
      LOGICAL LPRCOV, LCOVIL
C
C**********************************************************************
**
C
C     Initializations
C     ---------------
C
C     Initialize accumulation constraint counter, NACCUM
      NACCUM = 0
C
C     Initialize the synthesis cycle number corresponding to X*
      JSYCY = 0
      NVINT = 0
C
C
C     Initialize constant portions of Simplex Tableau
C     (CTWO, AMATX, BUP, BLO)
C     -  CALL INITAB
```

```
C Subroutine INITAB
C
C Purpose:  Initialize constant portions of Simplex Tableau
C           -  BUP, BLO, AMATX, BRHS and CTWO

C
       CALL INITAB (NDESV, NRELV, NDISV, NDEVAR,
      &             NLINCO, NLINGO, NMPRI, NNLGOA,
      &             IACTVR, LISIGN,
      &              COFLIN, DFNCOF, RHSLIN, VBOUNS, LVCOF,
      &             NFIX, NRFIX, AMATX, BLO, BUP, BRHS, CTWO)
C
C
C     Establish nonlinear characteristics of DESVAR.
C     (DESVAR and its associated arrays (CONDEV, DEVFUN, DEVVAR,
C     GVAL) correspond to the current best design (X*))
C     -  CALL DFCALC
C        (if NANCY = 0; otherwise values are passed in via argument)
C
       IF ( NANCY .EQ. 0 ) THEN
          IPATH = 1
          LPRCOV = .TRUE.
          LCOVIL = .FALSE.



C Subroutine DFCALC
C
C Purpose:  Calculate the deviation function for a new set of
C   system variables.
C
C    IPATH = 1:  Evaluate both constraints and goals
C                Return CONDEV and DEVFUN(CONDEV  real   total
constraint violation
                                          DEVFUN   real   vector of
deviation function values

C    IPATH = 2:  Evaluate constraints only
C                Return CONDEV
C    IPATH = 3:  Evaluate goals only
C                Return DEVFUN
C
C    (The subroutine updates the linear constraint coefficients
C    if required via a call to the user supplied routine, USRLIN.
C    This call is made after the call to GCALC.  This ordering
C    enables the advanced user to pass selected nonlinear information
```

```
C    from USRSET to USRLIN via a user defined common block.)

C
           CALL DFCALC (NUOUT, IPATH, NDESV, NMPRI,
    &                   NLINCO, NLINGO, NNLINQ, NNLCON, NNLGOA,
    &                   IADCON, LISIGN, LPRCOV, LCOVIL, LVCOF,
    &                   COFLIN, RHSLIN, DESVAR, DFNCOF, VILCN,
    &                   CONDEV, DEVFUN, DEVVAR, GVAL)
        ENDIF
C
C       Copy values into corresponding X1 arrays
C       -  i.e., CONVX1, DESVX1, DEVVX1, DFUNX1 AND GVALX1
C
        CONVX1 = CONDEV
        CALL COPYTX (NDESV, DESVX1, DESVAR)
        CALL COPYTX (NDEVAR, DEVVX1, DEVVAR)
        CALL COPYTX (NMPRI, DFUNX1, DEVFUN)
        IF ( NNLCON+NNLGOA .GT. 0 ) THEN
           CALL COPYTX (NNLCON+NNLGOA, GVALX1, GVAL)
        ENDIF
C
C
C       Start Synthesis Cycling
C       -----------------------
        DO 1000 INUMSY = 1, NSYCY
C
           NTITER = NTITER + 1
           NNUCON = 0
C
C          Print header 'cycle number(s)'
C
           IF ( .NOT. LPRFIN ) THEN
              IF ( NANCY .EQ. 0 ) THEN
                 WRITE (NUOUT, 1010) INUMSY
              ELSE
                 WRITE (NUOUT, 1020) INUMAN, INUMSY
              ENDIF
           ENDIF
C
C
C          Linearize nonlinear constraints and goals
C          -  CALL DERIV (Purpose: Calculate gradients of nonlinear
constraints and goals)

C
           CALL DERIV (NUOUT, NDESV, NNLCON, NNLGOA, NVSEL, IACTVR,
     &              DESVX1, GVALX1, PESTEP, VBOUNS,
     &              IADCON, KON1, MODIF,
     &              CONVEX, DG, DGTNGT, D2GDX2, RHS)
C
C
C        Update multiobjective goal formulation tableau arrays
C        (AMATX, CTWO, RHS)
C        -  CALL UPDTAB

C Subroutine UPDTAB
C
C Purpose:  Update multiobjective goal formulation tableau arrays:
C
C           Arrangement of rows in A and RHS matrices:
C              Linear goals                              NLINGO
C              Nonlinear goals                           NNLGOA
C              Linear constraints                        NLINCO
C              Nonlinear inequality constraints          NNLINQ
C              Nonlinear equality constraints            NNLEQU
C              Nonlinear accumulated constraints         NACCUM
C              Nonlinear new (adapted) constraints       NNUCON
C
C              NOTE: Bounds are not considered explicitly.
C
C           Arrangement of rows in CTWO matrix:
C              Constraint deviation coefficients            1
C              Goal deviation coefficients           2 to (NMPRI+1)

C
         CALL UPDTAB (NACCUM, NDESV, NLINCO, NLINGO, NNLEQU,
     &                NNLGOA, NNLINQ, NNUCON,
     &                IACTVR, IADCON, NEWCON,
     &                COFLIN, DG, DGACC, DGNEW,
     &                RHS, RHSACC, RHSLIN, RHSNEW,
     &                LVCOF,
     &                NCOL, NROW,
     &                AMATX, BRHS, CTWO)
C     print *,"after UPDTAB, NCOL = ",NCOL
C     print *,"NROW = ",NROW
C
C
C        Monitor process if required
C        -  CALL USRMON
C
         IF ( LMON ) THEN
            NDVUSR = NDESV
            NDEVUS = NDEVAR
            NPRUSR = NMPRI
            NNLCUS = NNLCON
            NNLGUS = NNLGOA
            NOUT = NUOUT
            DUMCON = CONVX1
            CALL COPYTX (NDESV, DUMVAR, DESVX1)
            CALL COPYTX (NDEVAR, DUMDEV, DEVVX1)
            CALL COPYTX (NMPRI, DUMDFN, DFUNX1)
            IF ( NNLCON+NNLGOA .GT. 0 ) THEN
               CALL COPYTX (NNLCON+NNLGOA, DUMGVL, GVALX1)
            ENDIF
C
            CALL USRMON (NDVUSR, NDEVUS, NPRUSR, NNLCUS, NNLGUS, NOUT,
     &                       DUMVAR, DUMDEV, DUMCON, DUMDFN, DUMGVL)
         ENDIF
C
C
C        Print input of linearized synthesis cycle
C        -  Design variables                   (if LPROUT(1) is true)
C        -  Deviation variables                (if LPROUT(2) is true)
C        -  Deviation function                 (if LPROUT(3) is true)
C        -  Bound information                  (if LPROUT(4) is true)
C        -  Linear constraint coefficients     (if LPROUT(5) is true)
C        -  Nonlinear constraint coefficients  (if LPROUT(6) is true)
C
         IF ( .NOT. LPRFIN ) THEN
            WRITE (NUOUT, 1030)
            IF ( CONVX1 .EQ. 0.0 ) THEN
               WRITE (NUOUT, 1040)
            ELSE
               WRITE (NUOUT, 1050)
            ENDIF
            IF ( LPROUT(1) ) THEN
               CALL PRDESV (NUOUT,INUMSY, IDDESV, DESVX1, NDESV)
            ENDIF
            IF ( LPROUT(2) ) THEN
               CALL PRDEVV (NUOUT, INUMSY, NLINGO, NNLGOA,
     &                        DEVVX1, IDLIGO, IDNLGO, IDDEVR)
            ENDIF
            IF ( LPROUT(3) ) THEN
               CALL PRDFUN (NUOUT, INUMSY, NMPRI, NDEVAR,
     &                        CONVX1, DFUNX1)
            ENDIF
            IF ( LPROUT(4) .AND. INUMSY .EQ. 1 ) THEN
               CALL PRBNDS (NUOUT, INUMSY, NDESV,
     &                     VBOUNS, DESVX1, IDDESV)
           ENDIF
           IF ( LPROUT(5) ) THEN
              CALL PRLINC (NUOUT, INUMSY, NDESV, NLINCO, NLINGO,
     &                     COFLIN, LISIGN, RHSLIN, IDDESV,
     &                     IDLICO, IDLIGO, IDDEVR)
           ENDIF
           IF ( LPROUT(6) ) THEN
              CALL PRNLNC (NUOUT, INUMSY, IADCON, MODIF, NDESV,
NNLTOT,
     &                     NACCUM, NNLINQ, NNLEQU, NLINGO, NNLGOA,
     &                     NACCON, GVALX1, DG, DGACC,
     &                     DESVX1, RHS, RHSACC,
     &                     IDDESV, IDNLCO, IDNLGO, IDDEVR)
           ENDIF
        ENDIF
C
C
C       Write postprocessor information file if requested
C       -  Call PRPPIF
C
        IF ( LPPROC ) THEN
           CALL PRPPIF (NSYCY, NTITER, NDESV, NDEVAR,
     &                  NNLINQ, NNLEQU, NNLGOA, NMPRI,
     &                  CONVX1, DESVX1, DFUNX1, DEVVX1,
     &                  GVALX1, RHS, DG, DGTNGT, D2GDX2)
        ENDIF
C
C
C       Solve LP problem, exit with X2
C       -  CALL MLINOP(Solve the LP problem using MULTIPLEX,
recording the basis to be used in the next synthesis cycle.)
C
        CALL MLINOP (NUOUT, NCOL, NDESV, NFIX, NRFIX,
     &               NMPRI, NROW, NVSEL,
     &               AMATX, BLO, BUP, BRHS, CTWO,
     &               DJMAT, XVAL, ZVAL)
C
C       Write XVAL into DESVX2
C
        JJ = 0
        DO 10 J = 1, NDESV
           IF ( IACTVR(J) .GT. 0 ) THEN
              JJ = JJ + 1
              DESVX2(J) = XVAL(JJ)
           ELSE
```

```
                DESVX2(J) = DESVAR(J)
             ENDIF
   10     CONTINUE

C
C
C         Adapt constraints if requested
C         -  CALL CONCOR (Subroutine CONCOR performs the adaption of
the nonlinear constraints according to: “Goal Programming, the
Compromise Decision Support Problem and Adaptive Linear Programming"

C
          IF ( LADAP ) THEN
C
             CALL CONCOR (NDESV, NNLINQ, NRELV, IACTVR, IADCON,
     &                    CONVEX, DESVX1, DESVX2, DG, DGTNGT, D2GDX2,
     &                    RHS,
     &                    NMODCN, NNUCON,
     &                    MODCON, MODGDX, MODSEQ,
     &                    NEWCON, NEWDG, NEWSEQ,
     &                    DGNEW, RHSNEW)
C
C
C            Print adapted (new and modified) constraints if requested
C            -  supplimentary input
C
             IF ( .NOT. LPRFIN .AND. LPROUT(6) ) THEN
                CALL PRADPC (NUOUT, INUMSY, NDESV, NMODCN, NNUCON,
     &                       MODCON, MODGDX, NEWCON, NEWDG,
     &                       GVALX1, DG, DGNEW, RHSNEW, RHS, DESVX1,
     &                       IDDESV, IDNLCO)
             ENDIF
C
C
             IF ( (NMODCN .GT. 0) .OR. (NNUCON .GT. 0) ) THEN
C
C
C               Update multiobjective goal formulation tableau arrays
C               (AMATX, CTWO, RHS)
C               -  CALL UPDTAB
C
                CALL UPDTAB (NACCUM, NDESV, NLINCO, NLINGO, NNLEQU,
     &                       NNLGOA, NNLINQ, NNUCON,
     &                       IACTVR, IADCON, NEWCON,
     &                       COFLIN, DG, DGACC, DGNEW,
     &                       RHS, RHSACC, RHSLIN, RHSNEW,
     &                        LVCOF,
     &                        NCOL, NROW,
     &                        AMATX, BRHS, CTWO)
C
C
C              Solve LP problem ,exit with new X2
C              -  CALL MLINOP ( Purpose:  Solve the LP problem using
MULTIPLEX, recording          the basis to be used in the next
synthesis cycle.  Algorithm has integer (branch and bound variation)
capability to handle 0/1 "selection" variables.

C
               CALL MLINOP (NUOUT, NCOL, NDESV, NFIX, NRFIX,
     &                        NMPRI, NROW, NVSEL,
     &                        AMATX, BLO, BUP, BRHS, CTWO,
     &                        DJMAT, XVAL, ZVAL)
C
C
C              Write XVAL into DESVX2
C
               JJ = 0
               DO 20 J = 1, NDESV
                  IF ( IACTVR(J) .GT. 0 ) THEN
                     JJ = JJ + 1
                     DESVX2(J) = XVAL(JJ)
                  ELSE
                     DESVX2(J) = DESVAR(J)
                  ENDIF
   20          CONTINUE

C
            ENDIF
C
         ENDIF
C
C
C        Write DEVVX2, CONVX2 and DFUNX2 corresponding to the
C        linear solution
C
         DO 30 J = 1, NDEVAR
            JJ = JJ + 1
            DEVVX2(J) = XVAL(JJ)
   30    CONTINUE
         CONVX2 = ZVAL(1)
         DO 40 J = 2, NMPRI + 1
            DFUNX2(J-1) = ZVAL(J)
   40    CONTINUE
C
C        Print output of linearized synthesis cycle
C        - Design variables                       (if LPROUT(1) is
true)
C        - Deviation variables                    (if LPROUT(2) is
true)
C        - Deviation function                     (if LPROUT(3) is
true)
C        - Bound activity (linear)                (if LPROUT(4) is
true)
C        - Linear constraint activity (linear)    (if LPROUT(7) is
true)
C        - Nonlinear constraint activity (linear) (if LPROUT(8) is
true)
C
C        - Shadow Prices/Dual Variables/Multipliers
C          (if LPROUT(7) is true .OR. LPROUT(8) is true)
C
         IF ( .NOT. LPRFIN ) THEN
            WRITE (NUOUT, 1060)
            IF ( LPROUT(1) ) THEN
               WRITE (NUOUT, 1061)
               CALL PRDESV (NUOUT,INUMSY, IDDESV, DESVX2, NDESV)
            ENDIF
            IF ( LPROUT(2) ) THEN
               WRITE (NUOUT, 1061)
               CALL PRDEVV (NUOUT, INUMSY, NLINGO, NNLGOA,
     &                      DEVVX2, IDLIGO, IDNLGO, IDDEVR)
            ENDIF
            IF ( LPROUT(3) ) THEN
               WRITE (NUOUT, 1061)
               CALL PRDFUN (NUOUT, INUMSY, NMPRI, NDEVAR,
     &                      CONVX2, DFUNX2)
            ENDIF
            IF ( LPROUT(4) ) THEN
               WRITE (NUOUT, 1061)
               CALL PRBNDS (NUOUT, INUMSY, NDESV,
     &                      VBOUNS, DESVX2, IDDESV)
            ENDIF
            IF ( LPROUT(7) ) THEN
               WRITE (NUOUT, 1061)
               CALL PRLINA (NUOUT, INUMSY, NDESV, NLINCO,
     &                      NLINGO, LISIGN, DESVX2, COFLIN,
     &                      DEVVX2, RHSLIN, IDLICO, IDLIGO)
            ENDIF
            IF ( LPROUT(8) ) THEN
               WRITE (NUOUT, 1061)
               CALL PRNLNA (NUOUT, INUMSY, NDESV, NNLINQ, NNLEQU,
     &                      NNLTOT, NACCUM, NLINGO, MODCON, MODGDX,
     &                      NEWCON, NEWDG, NACCON, IADCON,
     &                      DG, DGNEW, DGACC, RHS, DESVX2, DEVVX2,
     &                      RHSNEW, RHSACC, IDNLCO, IDNLGO)
            ENDIF
C            IF ( LSHPRC ) THEN
C Subroutine PRMULT (Purpose:  Print the linear solution constraint
and goal multipliers
C           (Lagrangians)
C         The Lagrangian Multiplier is equal to the Dual Variable
C           that is associated with each constraint or goal.
C
C           The shadow price vectors for each original basic variable
C           (i.e., the Dn- variables) in the primal correspond to the
C           transformed dual variable vector.  Thus, the first row of
C           Y* (dual solution) corresponds to the shadow price vector
C           for D1-, the second row to the shadow price vector for
C           D2-, and so on.


               IF ( LPROUT(7) .OR. LPROUT(8) ) THEN
                  WRITE (NUOUT, 1061)
                  CALL PRMULT(NDESV, NFIX, NMPRI, NUOUT, INUMSY,
     &                        NACCUM, NLINCO, NLINGO, NNLEQU,
     &                        NNLGOA, NNLINQ, NNUCON,
     &                        IADCON, NACCON, NEWCON, DJMAT,
     &                        IDLICO, IDLIGO, IDNLCO, IDNLGO)
               ENDIF
C            ENDIF
         ENDIF
C
C
C        Effect reduced move algorithm between X1 and X2
C        and overwrite X2 with new point
C Subroutine ADREMO
C Purpose: Adaptive reduced move using a Golden Section line search.
C          If no improvement is found in initial design, a fixed
C          reduced move is applied to the design point.
C
C          Default minimum reduced move = 2 * DELREM
          (DELREM = real acceptable convergence criterion for reduced
move)
```

```
C
      IF ( LADREM ) THEN
C
         CALL ADREMO (NUOUT, NDESV, NRELV, NMPRI,
     &                NNLINQ, NLINCO, NLINGO, NNLCON, NNLGOA,
     &                IADCON, LISIGN, LVCOF, COFLIN, RHSLIN,
     &                DFNCOF, VILCN, NADREM, DELREM, CONVX1,
     &                DESVX1, DESVX2, DFUNX1, DFUNX2)
C
      ELSE
C
         IF ( NRELV+NDISV .GT. 0 ) THEN
            DO 50 J = 1, NRELV+NDISV+NVINT
               IF ( IACTVR(J) .GT. 0 ) THEN
                  DELTA = ( DESVX2(J) - DESVX1(J) ) * REDMOV(J)
                  IF ( J .GT. (NRELV+NDISV) ) THEN
                     DELTA = FLOAT(INT(DELTA + 0.5))
                  ENDIF
                  DESVX2(J) = DESVX1(J) + DELTA
               ENDIF
   50       CONTINUE
         ENDIF
      ENDIF
C
C
C     Print new point after reduced move if requested
C     -  Design variables (if LPROUT(1) is true)
C
      IF ( .NOT. LPRFIN .AND. LPROUT(1) ) THEN
         WRITE (NUOUT, 1070)
         CALL PRDESV (NUOUT,INUMSY, IDDESV, DESVX2, NDESV)
      ENDIF
C
C
C     Establish nonlinear characteristics of X2
C     GVALX2 and DFUNX2
C     -  CALL DFCALC
C
      IPATH = 1
      LPRCOV = .TRUE.
      LCOVIL = .FALSE.
C
      CALL DFCALC (NUOUT, IPATH, NDESV, NMPRI,
     &             NLINCO, NLINGO, NNLINQ, NNLCON, NNLGOA,
     &             IADCON, LISIGN, LPRCOV, LCOVIL, LVCOF,
     &                  COFLIN, RHSLIN, DESVX2, DFNCOF, VILCN,
     &                  CONVX2, DFUNX2, DEVVX2, GVALX2)
C
C
C Subroutine CONVER
C
C Purpose:  Check for convergence.  This is done in several steps.
C
C   Given X1 the current starting vector (with dev. fun. = Z1)
C         X2 the new vector (with dev. fun. = Z2)
C       & X* the best vector (with dev. fun. = Z* found at iteration
J*)
C
C       1. Check for convergence of deviation function.
C       2. Check for convergence of design variables.
C       3. If X2 is better than X1, move X1 to X2.
C       4. If X1 is better than X*, move X* to X1 and J* to J2.

C        Test for convergence using nonlinear information
C        -  compare X1 and X2
C        -  CALL CONVER
C           -  Write X2 data into X1
C           -  Write X1 data into X* if X1 better than X*
C              [-  DESVAR == X*
C               -  DEVFUN == DEVFUN of (X*)
C               -  GVAL == GVAL of (X*)]
C
C        Build Z vectors for X*, X1 and X2
C
         ZSTAR(1) = CONDEV
         Z1(1) = CONVX1
         Z2(1) = CONVX2
         DO 60 J = 2, NMPRI + 1
            ZSTAR(J) = DEVFUN(J-1)
            Z1(J) = DFUNX1(J-1)
            Z2(J) = DFUNX2(J-1)
   60    CONTINUE
         NZ = NMPRI + 1
C
         CALL CONVER (NDESV, NZ, INUMSY,
     &                  JSYCY, DESVAR, DESVX1, DESVX2,
     &                  ZSTAR, Z1, Z2,
     &                  FRACX, FRACZ,
     &                  LCONSV, LCONDF, LIMPRV)
C
         CONVX1 = Z2(1)
         CALL COPYTX (NDESV, DESVX1, DESVX2)
         CALL COPYTX (NDEVAR, DEVVX1, DEVVX2)
         DO 70 J = 2, NMPRI + 1
            DFUNX1(J-1) = Z2(J)
   70    CONTINUE
         IF ( NNLCON+NNLGOA .GT. 0 ) THEN
            CALL COPYTX (NNLCON+NNLGOA, GVALX1, GVALX2)
         ENDIF
C
         IF ( JSYCY .EQ. INUMSY ) THEN
            CONDEV = ZSTAR(1)
            CALL COPYTX (NDEVAR, DEVVAR, DEVVX1)
            DO 80 J = 2, NMPRI + 1
               DEVFUN(J-1) = ZSTAR(J)
   80       CONTINUE
            IF ( NNLCON+NNLGOA .GT. 0 ) THEN
               CALL COPYTX (NNLCON+NNLGOA, GVAL, GVALX1)
            ENDIF
         ENDIF
C
C
C        Return to calling routine if convergence achieved
C
         IF ( LCONSV .AND. LCONDF ) THEN
C
            GOTO 1001
C
         ELSE
C
C           Accumulate constraints if X1 is feasible
C           - CALL CONACC
C
            IF ( CONVX1 .EQ. 0.0 ) THEN
C
               CALL CONACC (INUMSY, NACCUM, NDESV, NNLINQ, NNUCON,
     &                          IADCON, NACCIT, NACCON, NEWCON,
     &                          CONVEX, DG, DGNEW, RHS, RHSNEW,
     &                          LADAP,
     &                          DGACC, RHSACC)
C
            ENDIF
         ENDIF
C
 1000 CONTINUE
C
 1001 CONTINUE
```

```
C
C
C     Return to calling routine
C
C
      RETURN
C
C**********************************************************************
**
C     FORMATS
C
 1010 FORMAT ('1','            S Y N T H E S I S   C Y C L E'
     &            '   N U M B E R: ',I3/,
     &        ' ','            ------------------------------'
     &            '-------------------')
 1020 FORMAT ('1','             A N A L Y S I S   C Y C L E'
     &            '   N U M B E R: ',I3/,
     &        ' ','             ----------------------------'
     &            '-------------------'//,
     &        ' ','            S Y N T H E S I S   C Y C L E'
     &            '   N U M B E R: ',I3/,
     &        ' ','            ------------------------------'
     &            '-------------------')
 1030 FORMAT (/' ',' ***   I N P U T   ***'/
     &        ' ',' ---------------------')
 1040 FORMAT (/' ',' Current point is FEASIBLE'/
     &        ' ',' .........................'//)
 1050 FORMAT (/' ',' Current point is INFEASIBLE'/
     &        ' ',' ...........................'//)
 1060 FORMAT (//' ',' ***   O U T P U T   F R O M   L I N E A R   '
     &             'S O L V E R   ***'/
     &         ' ',' --------------------------------------------'
     &             '-----------------'///)
 1061 FORMAT (' << Reminder: Following output is from linear solver
>>')
 1070 FORMAT (//' ',' ***   O U T P U T   A F T E R   R E D U C E D
'
     &             'M O V E   ***'/
     &         ' ',' ------------------------------------------------'
     &             '-------------'///)
C
C**********************************************************************
**
C     End of Subroutine ALPMOD
C**********************************************************************
**
```

```
C
      END
```

ALPCTL Subroutine

```
C
C Program ALPCTL
C
C Purpose: A main program for DSIDES: SLIPML Version 4.80 /
C                                    ALP Release 1.0
C
C-----------------------------------------------------------------------
--
C Common Blocks:  ADVUS1/ IDV, IPERTB
C
C                 IDV      int    index of design variable being
C                                 perturbed
C                 IPERTB   int    flag indicating current perturbation
C                                 = 1  if 1st (for 1st order approx.)
C                                 = 2  if 2nd (for 2nd order approx.)
C
C            INTFLAG  int    Number of calls to foraging search for
C                       discrete variables: Kemper Lewis
C
C
C                 ADINTE/ NRELV, NDISV, NVSEL, NDESV, NDVUSR, NDEVAR,
C                         NLINCO, NLINGO, NMPRI,
C                         NNLINQ, NNLEQU, NNLCON, NNLGOA, NNLTOT
C
C
C                 ADREAL/ VBOUNS
C
C
C                 ADLOGI/ LMON, LUINP, LUOUT, LVCOF
C
C
C                 ADCHAR/ PTITLE,
C                         IDDESV, IDDEVR, IDLICO, IDLIGO, IDNLCO,
IDNLGO
C
C
C Include Files:  alplim.cmm
```

```
C
C Calls to:       TIMER, ALPDAT, USRINP, XPLORE, USRANA, COPYTX,
INITSL,
C                 DFCALC, ALPMOD, CONVER, PRDESV, PRDEVV, PRDFUN,
DISCRT,
C                 PRBNDS, PRPPCF, PRFACT, and USROUT
C----------------------------------------------------------------------
--
C Development History
C
C Author:  Warren Smith
C Date:    November 15, 1991
C
C Modifications:
C   June 16, 1993 (Bert Bras):
C       - Correction of user analysis cycle convergence check
C
C**********************************************************************
**
C-
      PROGRAM ALPCTL
C
      INCLUDE 'alplim.cmm'
C
C---------------------------------------
C     Advanced User Common Blocks:
C---------------------------------------
C
      COMMON/ADVUS1/ IDV, IPERTB

      INTEGER IDV, IPERTB

      INTEGER INTFLAG
C
C
C  *****KL added NDISV in all commons
      COMMON/ADINTE/ NRELV, NDISV, NVSEL, NDESV, NDVUSR, NDEVAR,
     &              NLINCO, NLINGO, NMPRI,
     &              NNLINQ, NNLEQU, NNLCON, NNLGOA, NNLTOT
      INTEGER NRELV, NDISV, NVINT, INDEX(MDESV),
     &          NVSEL, NDESV, NDVUSR, NDEVAR, NDSCC(MDESV)
      INTEGER NLINCO, NLINGO, NMPRI
      INTEGER NNLINQ, NNLEQU, NNLCON, NNLGOA, NNLTOT
C
C
      COMMON/ADREAL/ VBOUNS(2,MDESV)
      REAL VBOUNS
C
C
      COMMON/ADLOGI/ LMON, LUINP, LUOUT, LVCOF
      LOGICAL LMON, LUINP, LUOUT, LVCOF
C
C
      COMMON/ADCHAR/ PTITLE,
     &               IDDESV, IDDEVR, IDLICO, IDLIGO, IDNLCO, IDNLGO
      CHARACTER*80 PTITLE(2)
      CHARACTER*6 IDDESV(MDESV), IDDEVR(MDEVV), IDLICO(MLINCG),
     &     IDLIGO(MLINCG), IDNLCO(MNLNCG), IDNLGO(MNLNCG)
C
C---------------------------------------
C     Other Local variables:
C---------------------------------------
C
      INTEGER NUINP, NUOUT, NUPPC, NUPPI, NUSER, NOUT, NUSR
      INTEGER NANCY, NSYCY(MNANCY), NTITER
      INTEGER NHJMAX, NADREM
      INTEGER IACTVR(MDESV), IADCON(MNLNCG), LISIGN(MLINCG)
      INTEGER IGSEED, NPTBST, NPTGEN, NUPTS, IGENFX(MDESV)
      INTEGER I, INUMAN, IPATH, J, JANCY, JSYCY, JSYCST, NLEVEL, K
C
      REAL DESVAR(MDESV), DUMVAR(MDESV), CONDEV, DEVFUN(MLEVEL),
     &     DEVVAR(MDEVV), GVAL(MNLNCG), Z2(MLEVEL), TABUN(MDESV,MDSCV)
      REAL DESVX1(MDESV), CONVX1, DFUNX1(MLEVEL),
     &     DEVVX1(MDEVV), Z1(MLEVEL)
      REAL DESVST(MDESV), CONVST, DFUNST(MLEVEL), DEVVST(MDEVV),
     &     GVALST(MNLNCG), ZSTAR(MLEVEL)
      REAL COFLIN(MLINCG,MDESV), DFNCOF(MLEVEL,MDEVV),
     &     FRACX(MDESV), FRACZ(MLEVEL), PESTEP(MDESV), REDMOV(MDESV),
     &     RHSLIN(MLINCG), VILCN(MNLNCG), GOALS(MNLNCG),
CONSTR(MNLNCG)
      REAL HJEXPA, HJCONT, HJSTEP, HJEPSY, HJDELT
      REAL DELREM
C
      LOGICAL LFATAL, LDRYRN, LPRFIN, LPROUT(8), LPPROC, LADREM,
LADAP,
     &        LINIT, LPRGEN, LTIME, LXPLOR
      LOGICAL LCONDF, LCDF, LCONSV, LCSV, LIMPRV, LXFEAS
      LOGICAL LPRCOV, LCOVIL, LVDISC
C
      CHARACTER CURTIM*24, TIMCOM*75
      REAL INITIM(3), EXETIM(3)
C
```

```
C***********************************************************************
**
C                       *******   WARNING   *******
C
C         ******  INPUT AND OUTPUT UNIT NUMBER ASSIGNMENTS  ******
C
      NUINP = 11
      NUOUT = 13
      NUSER = 99
      NUPPC = 98
      NUPPI = 97
      NUPTS = 96
C
C     Open files:
C     1 - ALP input data file - ALPINP.DAT                (Unit #
NUINP)
C     2 - ALP output data file - ALPOUT.DAT               (Unit #
NUOUT)
C     3 - ALP USRINP scratch file                         (Unit #
NUSER)
C     4 - ALP postprocessor control file - ALPPPC.DAT     (Unit #
NUPPC)
C     5 - ALP postprocessor information file - ALPPPI.DAT (Unit #
NUPPI)
C     6 - ALP point generate/explore file - ALPPTS.DAT    (Unit #
NUPTS)
C
      OPEN (UNIT=NUOUT, FILE='ALPOUT.DAT', ACCESS='SEQUENTIAL',
     &      FORM='FORMATTED', STATUS='UNKNOWN')
C
CSYS********************************************************************
**
CSYS                    *******   WARNING   *******
CSYS
CSYS               ******* SYSTEM DEPENDENT ROUTINES *******
CSYS
CSYS  The routine TIMER is system dependent.
CSYS
CSYS  NUOUT  : unit number of output data file
CSYS  KODE   : indicates the timer option to be used.
CSYS  CURTIM : contains the current date plus time and is CHARACTER*24
CSYS  TIMCOM : contains an input string to be printed with the timer
CSYS           results
CSYS  EXETIM : returns the time in (CPU)seconds.
CSYS           The contents of EXETIM may differ when the system
CSYS           is installed on different operating systems.
CSYS          Check the comments in the timer routine for an
CSYS          explanation of the contents of EXETIM.
CSYS  INITIM : contains a reference time used to calculate
CSYS          the elapsed time since the start of execution.
CSYS
CSYS  The initial time (starting time) is stored in INITIM.
CSYS  This initial time may be required by subroutine TIMER in
CSYS  order to calculate the elapsed time since the start of
CSYS  execution. For instance, this is required
CSYS  for the VAX/VMS timer routine, but not for the Sun/OS timer
CSYS  routine
CSYS
CSYS
      CURTIM = '          '
      TIMCOM = '          '
CSYS
      CALL TIMER ( NUOUT, 2, TIMCOM, CURTIM, INITIM, INITIM )
CSYS
CSYS*******************************************************************
**
C
C     Record current date and time.
C
      CALL TIMER ( NUOUT, 0, TIMCOM, CURTIM, INITIM, EXETIM )
C
C     Write output program title block
C
      WRITE (NUOUT, 1010) CURTIM
      WRITE (NUOUT, 1020)
      WRITE (NUOUT, 1030)
      WRITE (NUOUT, 1031) MDESV, MLINCG, MNLNCG, MAXACC,
     &                    MLEVEL, MNSYCY, MNANCY, MTITER,
     &                    MATTRB, MSELPR, MPTBST
      WRITE (NUOUT, 1020)
C
C
C     Initializations
C     ---------------
C
      IDV = 0
      IPERTB = 0
C
C     Read in control information and initialize values
C     -  Call ALPDAT
C
      OPEN (UNIT=NUINP, FILE='ALPINP.DAT', ACCESS='SEQUENTIAL',
     &      FORM='FORMATTED', STATUS='OLD')
C
      OPEN (UNIT=NUSER, STATUS='SCRATCH',
     &      FORM='FORMATTED', ACCESS='SEQUENTIAL')
C
      CALL ALPDAT (NUOUT, NUINP, NUSER,
     &        NRELV, NDISV, NVINT, TABUN, NDSCC, INDEX, NVSEL,
     $           NDESV, NDEVAR,
     &        NLINCO, NLINGO, NMPRI,
     &        NNLCON, NNLEQU, NNLGOA, NNLINQ, NNLTOT,
     &        NANCY, NSYCY,
     &        NHJMAX, NADREM, IACTVR, IADCON, LISIGN,
     &        NPTGEN, NPTBST, IGSEED, IGENFX,
     &        IDDESV, IDDEVR, IDLICO, IDLIGO, IDNLCO, IDNLGO,
     &        PTITLE, COFLIN, RHSLIN, DFNCOF,
     &        PESTEP, REDMOV, VBOUNS, DESVAR,
     &        HJEXPA, HJCONT, HJSTEP, HJEPSY, HJDELT, DELREM,
     &        FRACZ, FRACX, VILCN,
     &        LFATAL, LDRYRN, LPRFIN,
     &        LPROUT, LPPROC, LTIME, LADREM, LADAP, LINIT,
     &        LMON, LUINP, LUOUT, LVCOF, LXPLOR, LPRGEN, LVDISC)
C
      CLOSE (NUINP)
C
C     STOP program if fatal errors encountered during reading
C
      IF ( LFATAL ) THEN
         CLOSE (NUSER)
         GOTO 9999
      ENDIF
C
C
C     Read and write user provided data
C     -  Call USRINP
C
      IF ( LUINP ) THEN
         REWIND (NUSER)
         NDVUSR = NDESV
         NUSR = NUSER
         NOUT = NUOUT
         CALL COPYTX (NDESV, DUMVAR, DESVAR)
         CALL USRINP (NDVUSR, NUSR, NOUT, DUMVAR)
      ENDIF
C
      CLOSE (NUSER)
C
```

```
C
C     Search bounded design space for feasibile or near feasible
C     starting point
C     -  Call XPLORE
C
      IF ( LXPLOR ) THEN
         OPEN (UNIT=NUPTS, FILE='ALPPTS.DAT', ACCESS='SEQUENTIAL',
     &         FORM='FORMATTED', STATUS='UNKNOWN')
C
         CALL XPLORE (IGSEED, NDESV, NLINCO, NLINGO, NMPRI, NNLINQ,
     &                NNLCON, NNLGOA, NPTBST, NPTGEN, NUOUT, NUPTS,
     &                IADCON, IGENFX, LISIGN,
     &                COFLIN, DFNCOF, RHSLIN, VBOUNS, VILCN,
     &                LPRGEN, LVCOF, IDDESV, PTITLE,
     &                DESVAR)
C
         CLOSE (NUPTS)
      ENDIF
C
C
C     End of input and initialization
C     Obtain timer results if required
C
      IF ( LTIME ) THEN
         TIMCOM = 'Time required for input and initialization:'
         CALL TIMER ( NUOUT, -1, TIMCOM, CURTIM, INITIM, EXETIM )
      ENDIF
C
C
C     STOP program if dry run printout requested only
C
      IF ( LDRYRN ) THEN
         IF ( LUOUT ) THEN
            NDVUSR = NDESV
            NOUT = NUOUT
            CALL COPYTX (NDESV, DUMVAR, DESVAR)
            LCDF = .FALSE.
            LCSV = .FALSE.
            LXFEAS = .FALSE.
            CALL USROUT (NDVUSR, NOUT, DUMVAR, LCDF, LCSV, LXFEAS)
         ENDIF
         WRITE (NUOUT, 1040)
         GOTO 9999
      ENDIF
C
C
```

```
C      Open the postprocessor information file if required and
C      write the problem title as a file header
C
       IF ( LPPROC ) THEN
          OPEN (UNIT=NUPPI, FILE='ALPPPI.DAT', ACCESS='SEQUENTIAL',
     &          FORM='FORMATTED', STATUS='UNKNOWN')
          WRITE (NUPPI, *) PTITLE(1)
          WRITE (NUPPI, *) PTITLE(2)
       ENDIF
C
C
C      Generate Initial Feasible Solution
C      ----------------------------------
C      -  Call USRANA (for user supplied/default starting point only)
C      -  Call INITSL
C
       IF ( LINIT ) THEN
          IF ( LTIME ) THEN
             CALL TIMER ( NUOUT, 2, TIMCOM, CURTIM, INITIM, EXETIM )
          ENDIF
C
C         Call user provided analysis routine USRANA, if required
C
          IF ( NANCY .GT. 0 ) THEN
             NDVUSR = NDESV
             NOUT = NUOUT
             CALL COPYTX (NDESV, DUMVAR, DESVAR)
             CALL USRANA (NDVUSR, NOUT, DUMVAR)
          ENDIF
C
          CALL INITSL (NUOUT, LVCOF, NLINCO, NNLINQ, NNLCON,
     &                 NRELV, NDESV, IACTVR, IADCON,
     &                 NHJMAX, HJEXPA, HJCONT, HJSTEP, HJEPSY, HJDELT,
     &                 DESVAR, VBOUNS, COFLIN, LISIGN, RHSLIN, VILCN,
     &                 CONDEV)
C
          WRITE (NUOUT, 1050) CONDEV
          IF ( NANCY .GT. 0 ) THEN
             WRITE (NUOUT, 1060)
          ENDIF
          WRITE (NUOUT, 1020)
C
          IF ( LTIME ) THEN
             TIMCOM = 'Time required for generation of initial
solution:'
             CALL TIMER ( NUOUT, -3, TIMCOM, CURTIM, EXETIM, EXETIM )
```

```
            ENDIF
         ENDIF
C
C
C
C        Perform appropriate cycling
C        ---------------------------
C
         NTITER = 0
C
C       NANCY = NUMBER OF ANALYSIS CYCLES!!

         IF ( NANCY .GT. 0 ) THEN
C
C           ANALYSIS CYCLES
C           '''''''''''''''
C
C           Obtain and record current timer values
C
            CALL TIMER ( NUOUT, 2, TIMCOM, CURTIM, INITIM, EXETIM )
C
C           Establish nonlinear characteristics of DESVAR.
C           (DESVST and its associated arrays (CONVST, DFUNST, DEVVST,
C           GVAL) correspond to the current best design (X*))
C           -  Call USRANA
C           -  Call DFCALC
C
            NDVUSR = NDESV
            NOUT = NUOUT
            CALL USRANA (NDVUSR, NOUT, DESVAR)
C
            IPATH = 1
            LPRCOV = .TRUE.
            LCOVIL = .FALSE.
C
            CALL DFCALC (NUOUT, IPATH, NDESV, NMPRI,
     &                   NLINCO, NLINGO, NNLINQ, NNLCON, NNLGOA,
     &                   IADCON, LISIGN, LPRCOV, LCOVIL, LVCOF,
     &                   COFLIN, RHSLIN, DESVAR, DFNCOF, VILCN,
     &                   CONDEV, DEVFUN, DEVVAR, GVAL)
C
C           Copy values into corresponding X* arrays
C           -  i.e., CONVST, DESVST, DEVVST, DFUNST AND GVALST
C
            CONVST = CONDEV
            CONVX1 = CONDEV
            CALL COPYTX (NDESV, DESVST, DESVAR)
            CALL COPYTX (NDEVAR, DEVVST, DEVVAR)
            CALL COPYTX (NMPRI, DFUNST, DEVFUN)
            CALL COPYTX (NDESV, DESVX1, DESVAR)
            CALL COPYTX (NDEVAR, DEVVX1, DEVVAR)
            CALL COPYTX (NMPRI, DFUNX1, DEVFUN)
            IF ( NNLCON+NNLGOA .GT. 0 ) THEN
               CALL COPYTX (NNLCON+NNLGOA, GVALST, GVAL)
            ENDIF
C
C
            JANCY = 0
C
C
C        Start Analysis Cycling
C        ----------------------
            DO 1000 INUMAN = 1, NANCY
C
C
C           Perform Synthesis cycles
C           -  Call ALPMOD
C           -  Returned DESVAR corresponds with best nonlinear
C              synthesis cycle solution
C           -  Record best results in arrays ....ST()
C           -  Record previous analysis cycle results in ....X1()
C           -  JANCY, JSYCST identify best analysis and corresponding
C              synthesis number
C
               CALL ALPMOD (NANCY, INUMAN, NTITER, NUOUT, NUPPI,
     &                   NRELV, NDISV, NVSEL, NDESV, NDEVAR,
     &                   NLINCO, NLINGO,
     &                   NNLINQ, NNLEQU, NNLCON, NNLGOA, NNLTOT,
     &                   NMPRI, NSYCY, IACTVR, IADCON, LISIGN,
     &                   JSYCY, NADREM,
     &                   CONDEV, DELREM,
     &                   COFLIN, DESVAR, DFNCOF,
     &                   DEVFUN, DEVVAR, FRACX, FRACZ, GVAL,
     &                   PESTEP, REDMOV, RHSLIN, VBOUNS, VILCN,
     &                   LADAP, LADREM, LMON, LVCOF, LPRFIN, LPROUT,
     &                   LPPROC, LCONDF, LCONSV, LIMPRV,
     &                   IDDESV, IDDEVR, IDLICO, IDLIGO, IDNLCO,
IDNLGO)
C
C
C           Calculate characteristics of new DESVAR
C           -  Call USRANA
```

```
C           -  Call DFCALC
C
            NDVUSR = NDESV
            NOUT = NUOUT
            CALL USRANA (NDVUSR, NOUT, DESVAR)
C
            IPATH = 1
            LPRCOV = .TRUE.
            LCOVIL = .FALSE.
C
            CALL DFCALC (NUOUT, IPATH, NDESV, NMPRI,
     &                   NLINCO, NLINGO, NNLINQ, NNLCON, NNLGOA,
     &                   IADCON, LISIGN, LPRCOV, LCOVIL, LVCOF,
     &                   COFLIN, RHSLIN, DESVAR, DFNCOF, VILCN,
     &                   CONDEV, DEVFUN, DEVVAR, GVAL)
C
C
C           Test for convergence using nonlinear information
C           -  compare DESVST and DESVAR
C           -  CALL CONVER
C              -  Write DESVAR data into DESVST
C                 if DESVAR better than DESVST
C
C           Build Z vectors for DESVST, DESVX1 and DESVAR
C           (See ALPMOD, build Z vectors for X*, X1 and X2)
C
            NLEVEL = NMPRI + 1
            ZSTAR(1) = CONVST
            Z1(1) = CONVX1
            Z2(1) = CONDEV
            DO 61 J = 2, NMPRI + 1
               ZSTAR(J) = DFUNST(J-1)
               Z1(J) = DFUNX1(J-1)
               Z2(J) = DEVFUN(J-1)
   61       CONTINUE
C
            CALL CONVER (NDESV, NLEVEL, INUMAN,
     &                   JANCY, DESVST, DESVX1, DESVAR,
     &                   ZSTAR, Z1, Z2,
     &                   FRACX, FRACZ,
     &                   LCONSV, LCONDF, LIMPRV)
C
            IF ( JANCY .EQ. INUMAN ) THEN
               JSYCST = JSYCY
               CONVST = ZSTAR(1)
               CALL COPYTX (NDEVAR, DEVVST, DEVVAR)
```

```
                DO 81 J = 2, NMPRI + 1
                   DFUNST(J-1) = ZSTAR(J)
   81           CONTINUE
                IF ( NNLCON+NNLGOA .GT. 0 ) THEN
                   CALL COPYTX (NNLCON+NNLGOA, GVALST, GVAL)
                ENDIF
             ENDIF
C
C            Copy current synthesis results into previous result
arrays.
C
             CONVX1 = CONDEV
             CALL COPYTX (NMPRI, DFUNX1, DEVFUN)
             CALL COPYTX (NDESV, DESVX1, DESVAR)
             CALL COPYTX (NDEVAR, DEVVX1, DEVVAR)
C
C            Print analysis cycle output if requested
C            -  Design variables     (if LPROUT(1) is true)
C            -  Deviation variables  (if LPROUT(2) is true)
C            -  Deviation function   (if LPROUT(3) is true)
C
             IF ( .NOT. LPRFIN ) THEN
                WRITE (NUOUT, 1070) INUMAN
                IF ( LPROUT(1) ) THEN
                   CALL PRDESV (NUOUT, JSYCY, IDDESV, DESVAR, NDESV)
                ENDIF
                IF ( LPROUT(2) ) THEN
                   CALL PRDEVV (NUOUT, JSYCY, NLINGO, NNLGOA,
     &                          DEVVAR, IDLIGO, IDNLGO, IDDEVR)
                ENDIF
                IF ( LPROUT(3) ) THEN
                   CALL PRDFUN (NUOUT, JSYCY, NMPRI, NDEVAR,
     &                          CONDEV, DEVFUN)
                ENDIF
             ENDIF
C
C
C            Print message 'End analysis/synthesis cycle number' and
C            obtain timer results for current analysis cycle
C
             WRITE (NUOUT, 1080) INUMAN
             IF ( LTIME ) THEN
                TIMCOM = 'Time required to complete analysis cycle:'
                CALL TIMER ( NUOUT, -3, TIMCOM, CURTIM, EXETIM, EXETIM
)
             ENDIF
```

```
C
            IF ( LCONSV .AND. LCONDF ) THEN
C
C              Print final analysis/synthesis cycle output - FINAL
C              SOLUTION header information
C
               WRITE (NUOUT, 1090) (PTITLE(J), J = 1, 2)
               WRITE (NUOUT, 1100) JANCY, JSYCST
               IF ( CONVST .GT. 0.0 ) THEN
                  WRITE (NUOUT, 1111)
                  WRITE (NUOUT, 1113)
                  WRITE (NUOUT, 1114)
                  WRITE (NUOUT, 1113)
                  WRITE (NUOUT, 1115)
               ENDIF
               GOTO 1001
            ENDIF
C
 1000    CONTINUE
C
C
C        Print best analysis/synthesis cycle output - BEST SOLUTION
C        header information
C
         WRITE (NUOUT, 1090) (PTITLE(J), J = 1, 2)
         WRITE (NUOUT, 1110) JANCY, JSYCST
         WRITE (NUOUT, 1111)
         WRITE (NUOUT, 1112)
         IF ( CONVST .GT. 0.0 ) THEN
            WRITE (NUOUT, 1113)
         ENDIF
         WRITE (NUOUT, 1114)
         IF ( CONVST .GT. 0.0 ) THEN
            WRITE (NUOUT, 1113)
         ENDIF
         WRITE (NUOUT, 1112)
         WRITE (NUOUT, 1115)
C
C
      ELSE
C
C        SYNTHESIS CYCLES ONLY
C        '''''''''''''''''''''
C
C        Obtain and record current timer values.
C
```

```
         CALL TIMER ( NUOUT, 2, TIMCOM, CURTIM, INITIM, EXETIM )
C
C******Discrete Part of solution -> Call TABU/FORAGING Algorithm
C
         IPATH = 1

         IF( LVDISC ) THEN
            INTFLAG = 1
         ELSE
            INTFLAG = 50
         ENDIF

 83      IF (INTFLAG.LT.14) THEN

            CALL FORAGEMV(INTFLAG,DESVAR,NDESV,NRELV,NDISV,TABUN,
NDSCC,
     &                        IPATH,NNLTOT,NOUT,NNLCON, NNLGOA,
     &                        DFNCOF, NMPRI, INDEX, VBOUNS, IACTVR)

            print *,"DESVAR(pre-alp) = ",(DESVAR(I),I=1,NDESV)
C              SET INACTIVE vars to discrete ones
            DO 82 K=NRELV+1,NRELV+NDISV
               IACTVR(K) = 0
 82         CONTINUE
         ENDIF
C
C
C
C        Perform Synthesis cycles
C        -  Call ALPMOD
C        -  Returned DESVAR corresponds with best nonlinear solution
C
         INUMAN = 0
C
         CALL ALPMOD (NANCY, INUMAN, NTITER, NUOUT, NUPPI,
     &             NRELV, NDISV, NVSEL, NDESV, NDEVAR,
     &             NLINCO, NLINGO,
     &             NNLINQ, NNLEQU, NNLCON, NNLGOA, NNLTOT,
     &             NMPRI, NSYCY, IACTVR, IADCON, LISIGN,
     &             JSYCY, NADREM,
     &             CONDEV, DELREM,
     &             COFLIN, DESVAR, DFNCOF,
     &             DEVFUN, DEVVAR, FRACX, FRACZ, GVAL,
     &             PESTEP, REDMOV, RHSLIN, VBOUNS, VILCN,
     &             LADAP, LADREM, LMON, LVCOF, LPRFIN, LPROUT,
     &             LPPROC, LCONDF, LCONSV, LIMPRV,
     &                IDDESV, IDDEVR, IDLICO, IDLIGO, IDNLCO, IDNLGO)
C
C     This is to check for discrete number of synthesis cycles complete
C
      WRITE (NUOUT,*) "************************************"
      WRITE (NUOUT,*) "This is the end of synthesis cycles.”
      WRITE (NUOUT,*) "************************************"

      print *,"DESVAR(post-alp) = ",(DESVAR(I),I=1,NDESV)

      IF (INTFLAG.LT.1) THEN
         INTFLAG = INTFLAG + 1
         goto 83
      ENDIF
C
C        Obtain timer results for current analysis cycle.
C
      CALL USRSET (IPATH, NDESV, MNLNCG, NOUT, DESVAR,
     &             CONSTR, GOALS)

         IF ( LTIME ) THEN
            TIMCOM = 'Time required to complete synthesis cycles:'
            CALL TIMER ( NUOUT, -3, TIMCOM, CURTIM, EXETIM, EXETIM )
         ENDIF
C
C
C        Print final/best synthesis cycle output header information
C
         WRITE (NUOUT, 1090) (PTITLE(J), J=1,2)
         IF ( LCONSV .AND. LCONDF ) THEN
            WRITE (NUOUT, 1120) JSYCY
            IF ( CONDEV .GT. 0.0 ) THEN
               WRITE (NUOUT, 1111)
               WRITE (NUOUT, 1113)
               WRITE (NUOUT, 1114)
               WRITE (NUOUT, 1113)
               WRITE (NUOUT, 1115)
            ENDIF
         ELSE
            WRITE (NUOUT, 1130) JSYCY
            WRITE (NUOUT, 1111)
            WRITE (NUOUT, 1112)
            IF ( CONDEV .GT. 0.0 ) THEN
               WRITE (NUOUT, 1113)
            ENDIF
```

```
              WRITE (NUOUT, 1114)
              IF ( CONDEV .GT. 0.0 ) THEN
                 WRITE (NUOUT, 1113)
              ENDIF
              WRITE (NUOUT, 1112)
              WRITE (NUOUT, 1115)
           ENDIF
C
C
        ENDIF
C
 1001 CONTINUE
C
C
      IF ( NANCY .GT. 0 ) THEN
C
C        Copy X* values back into original arrays
C
         JSYCY = JSYCST
         CONDEV = CONVST
         CALL COPYTX (NDESV, DESVAR, DESVST)
         CALL COPYTX (NDEVAR, DEVVAR, DEVVST)
         CALL COPYTX (NMPRI, DEVFUN, DFUNST)
         IF ( NNLCON+NNLGOA .GT. 0 ) THEN
            CALL COPYTX (NNLCON+NNLGOA, GVAL, GVALST)
         ENDIF
C
      ENDIF

C
C     Print final/best synthesis cycle output
C     -  Design variables and bound information
C     -  Constraint and goal activity
C     -  Deviation variables
C     -  Deviation function
C
      CALL PRBNDS (NUOUT, JSYCY, NDESV,
     &             VBOUNS, DESVAR, IDDESV)
      CALL PRFACT (NUOUT, JSYCY, NDESV, NLINCO, NLINGO,
     &             NNLCON, NNLGOA, IADCON,
     &             COFLIN, DESVAR, GVAL, RHSLIN,
     &             LVCOF, IDLICO, IDLIGO, IDNLCO, IDNLGO)
      CALL PRDEVV (NUOUT, JSYCY, NLINGO, NNLGOA,
     &             DEVVAR, IDLIGO, IDNLGO, IDDEVR)
      CALL PRDFUN (NUOUT, JSYCY, NMPRI, NDEVAR, CONDEV, DEVFUN)
C
```

```
      IF ( LCONSV .AND. LCONDF ) THEN
         IF ( CONDEV .GT. 0.0 ) THEN
            WRITE (NUOUT, 1111)
            WRITE (NUOUT, 1113)
            WRITE (NUOUT, 1114)
            WRITE (NUOUT, 1113)
            WRITE (NUOUT, 1115)
         ENDIF
      ELSE
         WRITE (NUOUT, 1111)
         WRITE (NUOUT, 1112)
         IF ( CONDEV .GT. 0.0 ) THEN
            WRITE (NUOUT, 1113)
         ENDIF
         WRITE (NUOUT, 1114)
         IF ( CONDEV .GT. 0.0 ) THEN
            WRITE (NUOUT, 1113)
         ENDIF
         WRITE (NUOUT, 1112)
         WRITE (NUOUT, 1115)
      ENDIF
C
C
C     Write postprocessor control file if requested and write
C     final values to the information file
C     -  Call PRPPCF
C
      IF ( LPPROC ) THEN
         OPEN (UNIT=NUPPC, FILE='ALPPPC.DAT', ACCESS='SEQUENTIAL',
     &         FORM='FORMATTED', STATUS='UNKNOWN')
C
         CALL PRPPCF (NUPPC, PTITLE, NDESV, NDEVAR,
     &                NMPRI, NNLINQ, NNLEQU, NNLGOA, NTITER,
     &                VBOUNS, IDDESV, IDDEVR, IDNLCO, IDNLGO)
C
         CLOSE (NUPPC)
C
C        Write final values to the information file
C
         WRITE (NUPPI, *)
         WRITE (NUPPI, *) NTITER+1, ' = The Final Solution'
         WRITE (NUPPI, *)
         DO 30 I = 1, NDESV
            WRITE (NUPPI, *) DESVAR(I), ' = desvar(', I, ')'
   30    CONTINUE
         DO 40 I = 1, NDEVAR
```

```
               WRITE (NUPPI, *) DEVVAR(I), ' = devvar(', I, ')'
   40       CONTINUE
            WRITE (NUPPI, *)
            WRITE (NUPPI, *) CONDEV, ' = condev'
            DO 50 I = 1, NMPRI
               WRITE (NUPPI, *) DEVFUN(I), ' = devfun(', I, ')'
   50       CONTINUE
            DO 60 I = 1, NNLINQ+NNLEQU+NNLGOA
               WRITE (NUPPI, *) GVAL(I), ' = gval(', I ,')'
   60       CONTINUE
C
            CLOSE (NUPPI)
C
         ENDIF
C
C
C        Perform sensitivity analysis.
C
C         IF ( LCONSV .AND. LCONDF .AND. CONDEV .EQ. 0.0 ) THEN
C            CALL POSSOL
C         ENDIF
C
C
C        Call user provided output routine if required
C
         IF ( LUOUT ) THEN
            IF ( CONDEV .EQ. 0.0 ) THEN
               LXFEAS = .TRUE.
            ELSE
               LXFEAS = .FALSE.
            ENDIF
            NDVUSR = NDESV
            NOUT = NUOUT
            CALL COPYTX (NDESV, DUMVAR, DESVAR)
            LCDF = LCONDF
            LCSV = LCONSV
            CALL USROUT (NDVUSR, NOUT, DUMVAR, LCDF, LCSV, LXFEAS)
            IF ( LCONSV .AND. LCONDF ) THEN
               IF ( CONDEV .GT. 0.0 ) THEN
                  WRITE (NUOUT, 1111)
                  WRITE (NUOUT, 1113)
                  WRITE (NUOUT, 1114)
                  WRITE (NUOUT, 1113)
                  WRITE (NUOUT, 1115)
               ENDIF
            ELSE
               WRITE (NUOUT, 1111)
               WRITE (NUOUT, 1112)
               IF ( CONDEV .GT. 0.0 ) THEN
                  WRITE (NUOUT, 1113)
               ENDIF
               WRITE (NUOUT, 1114)
               IF ( CONDEV .GT. 0.0 ) THEN
                  WRITE (NUOUT, 1113)
               ENDIF
               WRITE (NUOUT, 1112)
               WRITE (NUOUT, 1115)
            ENDIF
         ENDIF
   C
   C
   C     Obtain timer results for total problem.
   C
    9999 CONTINUE
         IF (LTIME) THEN
            TIMCOM = 'Total time required for problem:'
            CALL TIMER ( NUOUT, -1, TIMCOM, CURTIM, INITIM, EXETIM )
         ENDIF
   C
   C
   C     Record current date and time.
   C
         CALL TIMER ( NUOUT, 0, TIMCOM, CURTIM, INITIM, EXETIM )
   C
         WRITE (NUOUT, 1020)
         WRITE (NUOUT, 1140) CURTIM
         WRITE (NUOUT, 1150)
   C
         CLOSE (NUOUT)
   C
   C
   C*********************************************************************
   **
   C     FORMATS
   C
    1010 FORMAT('1','S L I P M L  -  JOB BEGUN ON : ', A)
    1020 FORMAT(/,' ', 79('*'))
    1030 FORMAT(/,
        &' SLIPML     -  Version 4.80, ALP Release 1.0, February
   1992'//,
        &'               Systems Realization Laboratory'/,
        &'               Design Methods Group'/,
        &'                 The George W. Woodruff School',
        &                  ' of Mechanical Engineering'/,
        &'                 Georgia Institute of Technology',/,
        &'                 Atlanta, Georgia 30332-0405'/
        &'                 United States of America'////
        &' Authorship  -   Farrokh Mistree and friends ...'//
        &'                 Janet Allen        Eduardo Bascaran'/
        &'                 Bert Bras          Owen Hughes'/
        &'                 Azim Jivan         Harsh Karandikar'/,
        &'                 Saiyid Kamal       Tim Lyon'/
        &'                 Ravi Reddy         Warren Smith'/,
        &'                 Srinivas Vadde',///)
    1031 FORMAT(' Problem Size Limits:'//
        &'                 System variables                   = ',I4/
        &'                 Linear constraints and goals       = ',I4/
        &'                 Nonlinear constraints and goals    = ',I4/
        &'                 Accumulated constraints            = ',I4/
        &'                 Goal priority levels               = ', I4/
        &'                 Synthesis cycles                   = ', I4/
        &'                 Analysis cycles                    = ', I4/
        &'                 Iterations (synthesis + ananlysis) = ', I4/
        &'                 Selection attributes               = ', I4/
        &'                 Selection problems (coupled)       = ', I4/
        &'                 Points saved in XPLORE             = ', I4)
    1040 FORMAT (//,' *** End of dry run data check'/
        &           '      --------------------------')
    1050 FORMAT (//,'              ATTEMPT TO GENERATE AN INITIAL
   FEASIBLE'
        &           ' SOLUTION'/
        &           '              --------------------------------------
   -'
        &           '---------'//
        &           ' CONSTRAINT VIOLATION of Generated Point: ',G12.5,/
        &           ' (considered feasible if violation is zero)'//
        &           ' The generated point is used as the input point for
   ',
        &           ' Synthesis Cycle Number 1')
    1060 FORMAT (//,' NOTE: Since USRANA is not called continually
   during'/
        &        '       the pattern search procedure, true feasibility'/
        &        '       of the generated point cannot be guaranteed.'/
        &        '       The USRANA information derived is only exact
   for'/
        &        '       the users initial guess.  As the generated'/
        &        '       point moves further away from the initail'/
        &        '       guess, the more imprecise the USRANA
   assumptions'/
        &        '       may become and the greater the risk that true'/
        &        '       feasibility is not achieved.')
    1070 FORMAT (//' ','   O U T P U T   O F   A N A L Y S I S   C Y C L
   E'
        &              '   N U M B E R: ',I3//,
        &          ' ','   ----------------------------------------------
   -'
        &              '-------------------')
    1080 FORMAT (//' *** End analysis/synthesis cycle number :',I3/
        &          '     ----------------------------------------')
    1090 FORMAT ('1'/,' ', 79('*'),//,A80,/,A80,//,' ', 79('*'))
    1100 FORMAT (/,' ','F I N A L   S O L U T I O N'/
        &        ' ','---------------------------'//
        &        ' ','from:      ANALYSIS CYCLE: ',I3,/
        &        ' ','output of: SYNTHESIS CYCLE:',I3,//
        &        ' ','CONVERGENCE ACHIEVED ',/
        & ' ','(based on variable and deviation function
   stationarity)'///)
    1110 FORMAT (/,' ','B E S T   S O L U T I O N   without convergence'/
        &        ' ','-------------------------'//
        &        ' ','from:      ANALYSIS CYCLE: ',I3,/
        &        ' ','output of: SYNTHESIS CYCLE:',I3,//
        &        ' ','MAXIMUM NUMBER OF ANALYSIS CYCLES PERFORMED'/
        &        ' ','CONVERGENCE NOT ACHIEVED ',/
        & ' ','(based on variable and deviation function
   stationarity)'///)
    1111 FORMAT (16X,'***********************************************')
    1112 FORMAT (16X,/
        &        16X,'C O N V E R G E N C E   N O T   A C H I E V E D',/
        &        16X,'C O N V E R G E N C E   N O T   A C H I E V E D',/
        &        16X)
    1113 FORMAT (16X,/
        &        16X,'F E A S I B I L I T Y   N O T   A C H I E V E D',/
        &        16X,'F E A S I B I L I T Y   N O T   A C H I E V E D',/
        &        16X)
    1114 FORMAT (16X,/
        &        16X,'W   W   AAA   RRRR   N   N  IIIII  N   N   GGG',/
        &        16X,'W   W  A   A  R   R  NN  N    I    NN  N  G   G',/
        &        16X,'W   W  A   A  R   R  NN  N    I    NN  N  G',/
        &        16X,'W   W  AAAAA  RRRR   N N N    I    N N N  G',/
        &        16X,'W W W  A   A  R  R   N  NN    I    N  NN  G  GG',/
        &        16X,'W W W  A   A  R   R  N  NN    I    N  NN  G   G',/
        &        16X,' W W   A   A  R   R  N   N  IIIII  N   N   GGG',/
        &        16X)
    1115 FORMAT
   (16X,'************************************************',///)
    1120 FORMAT (/,' ','F I N A L   S O L U T I O N'/
        &         ' ','-----------------------------'//
        &         ' ','output of: SYNTHESIS CYCLE:',I3,//
        &         ' ','CONVERGENCE ACHIEVED ',/
        & ' ','(based on variable and deviation function
   stationarity)'///)
    1130 FORMAT (/,' ','B E S T   S O L U T I O N   without convergence'/
        &         ' ','-------------------------'//
        &         ' ','output of: SYNTHESIS CYCLE:',I3,//
        &         ' ','MAXIMUM NUMBER OF ITERATIONS PERFORMED'/
        &         ' ','CONVERGENCE NOT ACHIEVED ',/
        & ' ','(based on variable and deviation function
   stationarity)'///)
    1140 FORMAT (//' S L I P M L  -  JOB COMPLETED ON : ', A)
    1150 FORMAT (//' ********        End of Computation        ********')
   C
   C*********************************************************************
   **
   C      End of Program ALPCTL
   C*********************************************************************
   **
   C
         END
```

References:

1. Muster, D. and Mistree, F., "The Decision Support Problem Technique in Engineering Design," The International Journal of Applied Engineering Education, vol. 4, no. 1, pp. 23-33, 1988.

2. Jon A. Shupe, Decision-Based Design: Taxonomy and Implementation, PhD Dissertation, University of Houston, May 1988.

3. Mistree, F., Smith, W. F. and Bras, B. A., 1993, "A Decision-Based Approach to Concurrent Engineering" in Handbook of Concurrent Engineering, pages 127-158, (H. R. Parsaei and W. Sullivan, Eds.), New York: Chapman & Hall.

4. Mistree, F., Smith, W. F., Bras, B., Allen, J. K. and Muster, D., " Decision-Based Design: A Contemporary Paradigm for Ship Design," Transactions, Society of Naval Architects and Marine Engineers, vol. 98, pp. 565-597, 1990.

5. Ignizio, J. P. (1985) Multiobjective Mathematical Programming via the MULTIPLEX Model and Algorithm, European Journal of Operational Research, 22, 338-346.

6. Mistree, F., Hughes, O. F. and Bras, B. A., 1993, "The Compromise Decision Support Problem and the Adaptive Linear Programming Algorithm" in Structural Optimization: Status and Promise, pages 247-286, (M. P. Kamat, Ed.), Washington, D.C.: AIAA.

7. Warren Smith, "Modeling and Exploration of Ship Systems in the Early Stages of Decision-Based Design," PhD Dissertation, University of Houston, July 1992, .

8. Guo, L, "Model Evolution for the Realization of Complex Systems," PhD Dissertation, Industrial and Systems Engineering, University of Oklahoma, 2021.

9. Mistree, F., Allen, J. K. and Chirehdast, M. (1989). Decision support in the concurrent engineering design of gas turbine engines. Engineering optimization, 14(3), 209-236.

10. This paper introduces the concept of DSIDES and its application in the concurrent engineering design of gas turbine engines.

11. Mistree, F., & Chen, W. (1990). Satisficing trade-offs in engineering design. ASME Journal of Mechanical Design, 112(2), 223-229.

12. This paper discusses the satisficing approach in engineering design and the role of DSIDES in identifying robust satisficing solutions.

13. Yang, R. J., & Wang, X. (2005). A method for robust design based on decision support in the design of engineering systems. Journal of Mechanical Science and Technology, 19(5), 1200-1206.

14. This paper presents a method for robust design based on DSIDES, highlighting its role in supporting decision-making in engineering systems design.

15. Zhao, L., & Chang, X. (2011). Design optimization for manufacturing system based on DSIDES. Advanced Materials Research, 308-310, 672-676.

16. This paper discusses the application of DSIDES in the design optimization of manufacturing systems.

17. Official DSIDES Website: http://www.mae.ufl.edu/mdo/DSIDES.htm

18. The official website provides information about DSIDES, including its features, applications, and related publications.

19. Mistree, F., & Wiecek, M. M. (1995). Concurrent design of products and processes: A strategy for the next generation in manufacturing. In Proceedings of the 1995 ASME International Mechanical Engineering Congress and Exposition (pp. 429-436).

20. Mistree, F., Hughes, O. F., & Bras, B. A. (1993). The Decision Support Problem in Concurrent Engineering. Advances in Concurrent Engineering, 1, 113-130.

21. This paper by Mistree and his colleagues discusses the Decision Support Problem (DSP) in concurrent engineering and introduces ALP as an approach to addressing the challenges associated with decision-making in complex engineering systems.

22. Mistree, F., & Allen, J. K. (1994). Decision support in the design of large-scale complex engineering systems. Systems Engineering, 1(3), 143-161.

23. Mistree and Allen's paper presents a methodology for decision support in the design of large-scale complex engineering systems using ALP. It explores the concept of adaptability and the integration of multidisciplinary design objectives.

24. Mistree, F., Allen, J. K., & Sitework, D. (1998). Integrated design and manufacturing in mechanical engineering: Decision-based approaches. Journal of Manufacturing Systems, 17(5), 371-392.

25. This paper discusses the integration of design and manufacturing processes in mechanical engineering and presents ALP as a decision-based approach to support concurrent design and manufacturing decision-making.

26. Allen, J. K., & Mistree, F. (1996). Validating product and process design decisions using adaptive linear programming. Engineering with Computers, 12(4), 238-250.

27. Allen and Mistree's paper focuses on the validation of product and process design decisions using ALP. It discusses the implementation of ALP and its application in addressing uncertainties and variations in design and manufacturing processes.

28. Allen, J. K., & Mistree, F. (1997). Adaptive linear programming applied to structural design. Structural Optimization, 13(1), 5-14.

29. The application of ALP to structural design problems is explored in this paper. The adaptive nature of ALP and its ability to handle uncertainties and variations in structural design decision-making are discussed.

30. Thole, S.P., Ramu, P., “Design space exploration and optimization using self-organizing maps”, Struct Multidisc Optima 62, 1071–1088 (2020).