OPTIMAL SYNTHESIS OF FORCE-GENERATING PLANAR FOUR-LINK MECHANISMS

by

Michael Thomas Scardina

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

MECHANICAL ENGINEERING

APPROVED:

Charles F. Reinholtz, Chairman

Paul H. Tidwell

R. G. Kirk

October 25, 1996
Blacksburg, Virginia

Key words: Synthesis, Mechanism, Optimization, Linkage, Force-Generation

OPTIMAL SYNTHESIS OF FORCE-GENERATING PLANAR FOUR-LINK MECHANISMS

by

Michael Thomas Scardina

Charles F. Reinholtz (Chairman)

Mechanical Engineering

ABSTRACT

This thesis presents a technique for the optimal synthesis of planar four-bar linkages for specified force or torque generation.

Unlike most previous research in mechanism force synthesis, this thesis targets linkage applications for which the mechanical advantage is prescribed but the position function is not. The advantage of this approach is that emphasis is placed on the force-generating properties of the linkage and not the output positions of the mechanism.

Closed-form synthesis of force-generating linkages based on Burmester theory has recently been developed in detail. Unfortunately, closed-form methods can only be used to solve a limited class of problems and frequently require substantial intuition on the part of the designer in adjusting the input parameter specifications. The approach presented here uses optimization theory to search a solution space for the mechanism that most nearly meets the designer's specification. Use of optimization allows a greater number of constraints to be applied to the synthesis such that more practical solutions can be obtained. The proposed design technique seeks to minimize an objective function that depends primarily on the force generation properties of the linkage and secondarily on the other applied constraints.

As a demonstration of the theoretical method, the optimal linkage synthesis technique is applied to a specific problem, namely, the design of a linkage for a weight-loaded exercise machine. Example solutions are generated and evaluated against the design constraints and mechanical advantage requirements. The design methodology presented has been implemented into a software package which is currently being used in industry for the design of similar linkages.

Acknowledgments

There are so many people I wish to thank for their support in completion of this research. First and foremost, I want to thank Dr. Charles Reinholtz, chairman of my graduate committee and friend. I can't imagine any faculty member devoting as much of their time and attention to his students as Dr. Reinholtz. Without his constant encouragement, guidance, and unselfish support throughout my studies, I would have never made it to the end.

I also want to acknowledge the other members of my graduate committee, Dr. Gordon Kirk and Dr. Paul Tidwell. Their helpful advice and support of my efforts is greatly appreciated.

Thanks are extended to Nautilus and all of the great people I had the opportunity to work with for their generous financial support of this research effort and encouraging feedback along the way. Special thanks go to Greg Webb and Michael A. Lo Presti.

Special thanks go to my friends and fellow students who offered technical and moral support along the way. Especially, Randy and Anita Soper, Dave Schmiel, and Andrew Baxter for their friendship and for keeping things fun all the time.

I want to thank God, my family, and especially my parents for their unending love and support of all my endeavors.

Finally, I want to thank my lovely fiancée, Soo, for all of the support and encouragement she showed me in this effort.

Contents

List of Figures

List of Tables

Nomenclature

A_r	Area under the resistance curve
B	Nondimensional bearing force
e	Exponential constant
F	Force
g_u	Inequality constraint function
h_v	Equality constraint function
i	$\sqrt{-1}$; Index
l_{in}	Input arm length
l_n	Length of link n
l_w	Weight arm length
L_n	Length of link n, nondimensional
n	Link #, Index
P	Scalar penalty value
q	Current optimization design variable
R	Resistance curve, nondimensional
s	Displacement
S	Strength curve, nondimensional
t	Time; Parametric variable
T	Torque

v	Linear velocity
v^w_y	Vertical velocity component of the weight
w_i	Weighting factor *i*
W	Load weight
Δx_i	Optimization step size for design variable
x_n	Optimization design variable *n*
$\mathbf{X}^0$	Initial optimization design vector
$\mathbf{X}$	Optimization design vector
β	User input angle
θ	Independent input motion variable
θ_n	Angle of link *n*
θ_{in}	Input arm offset
θ_w	Weight arm offset
μ	Mechanical advantage
ξ	Four-bar closure, +1 or -1
τ	Linkage transmission angle
Φ	Linkage output angle
χ	Linkage ground link orientation
ω	Angular velocity
ω_n	Angular velocity of link *n*

1. Introduction

1.1 Background and Motivation

The focus of this research is the design of planar four-bar mechanisms to satisfy specific input force or torque requirements. The research was performed under contract with Nautilus, Inc., a Virginia-based fitness equipment company, and the Virginia Center for Innovative Technology (CIT). For Nautilus, the intent of the work was to produce software tools and general algorithms to aid in the design of linkages suitable for use in their newest fitness equipment. The methods and techniques for linkage synthesis developed in this thesis have numerous application in many types of process machinery such as presses, mixers, and engaging mechanisms as well as in control actuation and, of course, exercise equipment. For example, the feasibility of using force-generating linkages in place of variable mechanical advantage cams in compound archery bows is currently under investigation.

The work presented in this thesis is of particular interest to Nautilus. As the world leader in the design and production of fitness equipment, they demand optimal performance from every piece of equipment they manufacture. In addition to the stringent specifications on strength and durability, they insist on providing the user with an efficient workout. They do this by designing the equipment to provide resistance that approximately matches the user's ability to exert force. One concept that will be frequently appearing in this thesis is the *nondimensional strength curve*. This refers the

ability of a user to exert varying forces over the entire range of motion of a given exercise. A person's ability to exert force depends on muscle strength and geometry of the muscle and skeletal structure involved. Although strength tends to vary widely in magnitude, the general shape of the strength curve tends to be similar across broad populations. This is due in part to the similarity in structural geometry (Lieber, 1992, Thompson, 1973, Schneck, 1990, and Harrison, 1970). A typical strength curve for a rowing exercise is depicted in Fig. 2-2. If the strength curve is normalized with respect to a reference force (such as the maximum over the range), then the resulting curve will be approximately the same for the most people.

In order to achieve the optimum workout, the exercise machine should match as closely as possible the user's ability to exert force at each point over the range of motion (also referred to as the *stroke* of an exercise). If a constant resistance like a free weight (i.e., a barbell) is used, the amount of weight is limited by the minimum force the user can exert over the stroke. Because of this, the amount of *work* that the user can do over the stroke is

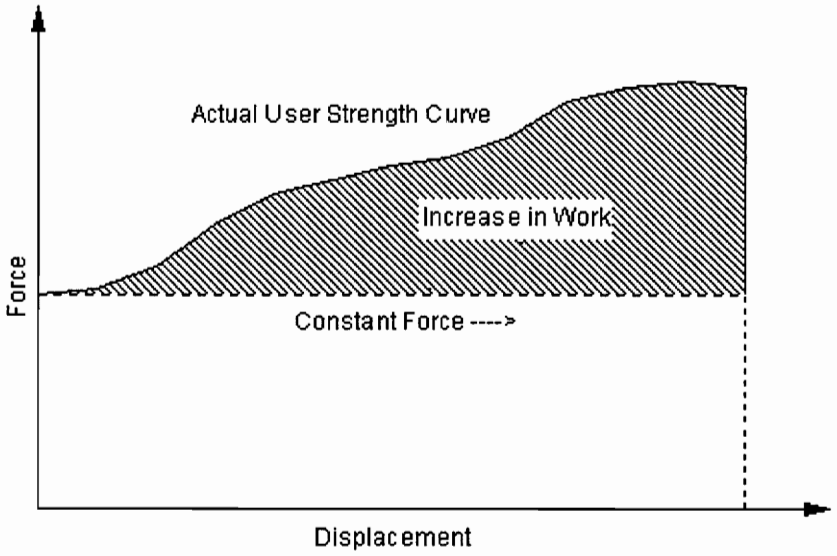


Figure 1-1, How Strength Matching Increases Work

reduced. In this case, the work that is done is equal to the area under the curve relating force and displacement. The amount of the reduction of the work is shown graphically in Fig. 1-1. Similar to the nondimensional strength curve of the user is the nondimensional curve of the machine referred to as the *resistance curve*. The resistance curve represents the resisting force of the machine over the range of motion. Hence, the primary goal of this thesis work is to develop a technique for designing linkage-based exercise machines which have resistance curves that closely match the nondimensional strength curves of a typical user for a given exercise.

1.2 Tools and Concepts Behind Optimum Synthesis

This section presents basic topics in kinematics and optimization that are needed in the remainder of this thesis. Kinematics is the study of mechanisms from the point of view of motion requirements. Optimization is any process which seeks the best solution to a given problem. Specifically, optimization refers to the numerical minimization of a certain function which represents the quality of solutions to a problem.

1.2.1 Position Analysis of Planar Four-Bar Linkages

Position analysis of the four-bar linkage is an essential element of the design technique detailed in this thesis. The analysis of mechanisms is generally nonlinear. The analysis of the four-bar linkage for position, velocity and acceleration is well documented. The method used in this thesis is adapted from Mabie and Reinholtz (1987) and is detailed below.

A powerful and frequently used method for solving closed-loop kinematic mechanisms (or chains) is the *loop-closure* method. The idea behind the loop-closure method is to sum the vectors which describe one or more closed loops around the mechanism. In the following analysis, vectors are represented in the complex plane with x and y coordinates corresponding to the real and imaginary axes, respectively. For example,

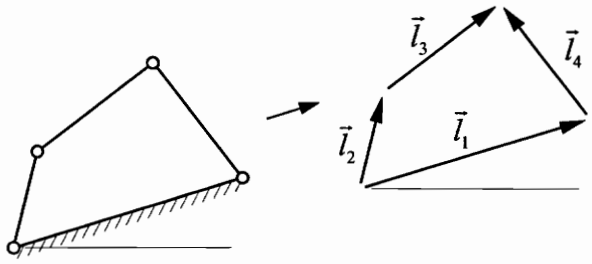


Figure 1-2, Loop-Closure of Four-bar

$$\vec{l}_n = l_n e^{i\theta_n} \tag{1.1}$$

is a vector of magnitude l_n at an orientation θ_n with respect to the positive x axis. Applying loop-closure to the standard four-bar linkage shown in Fig. 1-2 yields the equation

$$\vec{l}_1 + \vec{l}_4 = \vec{l}_2 + \vec{l}_3 \tag{1.2}$$

Using complex vector notation with the real axis along l_1,

$$l_1 + l_4 e^{i\theta_4} = l_2 e^{i\theta_2} + l_3 e^{i\theta_3} \tag{1.3}$$

Now, the Euler identity

$$e^{i\theta} = \cos(\theta) + i\sin(\theta) \tag{1.4}$$

is substituted and we arrive at

$$l_4 \sin(\theta_4) = l_2 \sin(\theta_2) + l_3 \sin(\theta_3) \tag{1.5a}$$

from the imaginary part and

$$l_1 + l_4 \cos(\theta_4) = l_2 \cos(\theta_2) + l_3 \cos(\theta_3) \tag{1.5b}$$

from the real part. In the standard four-bar position analysis problem, the lengths of the four links and the input angle, θ_2, are known. Inspection of the last two equations reveals that there are two equations in two unknowns, θ_3 and θ_4. Solution of these two equations is difficult because the unknowns are arguments of trigonometric functions. The closed-form solution method involves isolating the term containing one of the unknown angles from each of Equations 1.5a and 1.5b onto one side of the equation. Next, those two equations are squared and added to each other to eliminate the isolated angle. After algebraic manipulation, it can be shown that the resulting equation is of the general form

$$E \sin(\theta) + F \cos(\theta) + G = 0, \tag{1.6}$$

where θ is the angle that was not isolated in the previous steps (either θ_3 or θ_4). For the standard four-bar linkage case, the values of E, F, and G are

$$E_3 = 2 l_2 l_3 \sin(\theta_2) \tag{1.7a}$$

$$F_3 = 2 l_2 l_3 \cos(\theta_2) - 2 l_1 l_3 \tag{1.7b}$$

$$G_3 = l_2^{\,2} + l_3^{\,2} + l_1^{\,2} - l_4^{\,2} - 2 l_1 l_2 \cos(\theta_2) \tag{1.7c}$$

$$E_4 = -2 l_2 l_4 \sin(\theta_2) \tag{1.7d}$$

$$F_4 = 2 l_1 l_4 - 2 l_4 l_2 \cos(\theta_2) \tag{1.7e}$$

$$G_4 = l_2^{\,2} + l_4^{\,2} + l_1^{\,2} - l_3^{\,2} - 2 l_1 l_2 \cos(\theta_2) \tag{1.7f}$$

where the subscripts 3 and 4 refer to the angle which was not isolated. For example, if θ_4

were isolated in Equations 1.5a and 1.5b, then Equations 1.7a - 1.7c are used to solve for θ_3.

Solution of Equation 1.6 could be solved using a numerical approach, but a closed-form solution is more desirable for application to the current research. The closed-form solution is achieved by making a trigonometric substitution called the tangent half-angle identity which is given by

$$\tan\left(\frac{\theta}{2}\right) = t, \tag{1.8}$$

$$\cos(\theta) = \frac{1-t^2}{1+t^2} \tag{1.9}$$

$$\sin(\theta) = \frac{2t}{1+t^2}. \tag{1.10}$$

Substitution of Equations 1.9 and 1.10 into Equation 1.6 results in

$$E\left(\frac{2t}{1+t^2}\right) + F\left(\frac{1-t^2}{1+t^2}\right) + G = 0, \tag{1.11}$$

which simplifies to

$$(G-F)t^2 + 2Et + (G+F) = 0. \tag{1.12}$$

This equation can be solved using the quadratic formula which results in the following expression for θ:

$$\theta = 2 \cdot \tan^{-1}\left(\frac{-E \pm \sqrt{E^2 + F^2 - G^2}}{G-F}\right). \tag{1.13}$$

This equation gives interesting information about the four-bar mechanism. Notice the

term under the radical sign in Equation 1.13, which is referred to as the discriminant of the polynomial. There are three possible situations which can arise here. For the case when the discriminant is negative, the linkage cannot assemble for the current input angle θ_2; the vector loop does not close for the given link lengths. Another case is the discriminant being equal to zero, which indicates that the linkage is at a dead point or singular point (Mabie and Reinholtz, 1987). When the discriminant is greater than zero, there are two possible solutions. Equation 1.13 indicates that there would be four possible solutions: two from the inverse tangent function and two from the quadratic solution for t in the equation. Note, however, that only two solutions are physically possible and that the equations are generating a pair of repeated solutions. The two physically valid solutions are called the branches or *closures* of the linkage (Mabie and Reinholtz, 1987). With this is mind, the solution to the position analysis problem is given by

$$\theta_3 = 2 \cdot \tan^{-1}\left(\frac{-E_3 + \xi\sqrt{E_3^{\,2} + F_3^{\,2} - G_3^{\,2}}}{G_3 - F_3}\right), \tag{1.14a}$$

$$\theta_4 = 2 \cdot \tan^{-1}\left(\frac{-E_4 - \xi\sqrt{E_4^{\,2} + F_4^{\,2} - G_4^{\,2}}}{G_4 - F_4}\right), \tag{1.14b}$$

where the default inverse tangent function (first or fourth quadrant) is assumed. The value of ξ is either +1 or -1, which determines the closure of the linkage.

Once the position analysis of the linkage has been completed, velocity and acceleration analysis can be performed by taking the first and second time derivatives of

Equation 1.3. This analysis is omitted here but is available in a number of excellent references (Mabie and Reinholtz, 1987, Hall, 1961, Sandor and Erdman, 1984).

1.2.2 Principle of Virtual Work

A powerful method applicable to the analysis of linkages is called *virtual work*, or D'Alembert's principle. The virtual work principle is derived from Newton's Third Law and says that if a body is in equilibrium under the action of external forces then the total work done by these forces is zero for a small or *virtual displacement* of the body. A virtual displacement is imaginary and hence it is time independent, however, it must be consistent with all kinematic constraints. The concept of virtual work is expressed mathematically as

$$\sum \vec{F}_n \cdot \delta\vec{s}_n + \sum \vec{T}_n \cdot \delta\vec{\theta}_n = 0, \tag{1.15}$$

where F and T are the external forces and torques summed over *n*, the index of the bodies or locations on which they act (Mabie and Reinholtz, 1987, pp. 421-422). Because we are performing a static analysis, the virtual displacements in Equation 1.15 are identical to the true differential displacements of the system. Dividing Equation 1.15 by *dt* yields

$$\sum \vec{F}_n \cdot \vec{v}_n + \sum \vec{T}_n \cdot \vec{\omega}_n = 0, \tag{1.16}$$

where the velocities are virtual also because of the static analysis. If it is also assumed that the mechanism being considered is a planar linkage with only one planar input torque and one planar output torque applied to it, a most useful result is found for this case.

$$T_{in}\omega_{in} - T_{out}\omega_{out} = 0 \tag{1.17}$$

From this equation, the *mechanical advantage*, μ, is defined as the torque or velocity ratio

$$\mu = \frac{T_{out}}{T_{in}} = \frac{\omega_{in}}{\omega_{out}}. \tag{1.18}$$

1.2.3 Important Assumptions in the Analysis

For the mechanism synthesis problem being considered, two basic assumptions have been made which simplify the analytical portion of the problem. First, it is assumed that the dynamic effects of the system are small compared to the static forces. In actuality, the magnitude of any dynamic effects are dependent upon how the machine is used. Nautilus machines are intended to be used in a slow and controlled manner. If the machines are used properly, the dynamic effects should indeed be small in comparison to the static effects.

The second assumption made is that the effect of the link masses on the input force is small compared to the effect of the applied weight load. This assumption is particularly appropriate for the stronger users who tend to load a large amount of weight onto the machine. Certain exercises that target stronger muscle groups such as the legs will almost always require the user to load a significant weight onto the machine. Other exercises, however, target weaker muscles like the wrists and neck, so less weight will be applied. It is in the lower weight cases when the link masses and inertia may become significant. Although link masses will be excluded for the present design work, it is possible to consider the mass of the links as part of the optimization problem. This is not

true for closed-form synthesis techniques in which the inclusion of link masses and inertias would make the problem solution inefficient or impossible. The numerical optimization-based techniques that will be used in solving the synthesis problem in this thesis eliminate the need to solve complex analytical equations. This also affords the designer more flexibility when seeking solutions because exact solutions to the governing equations are not required.

1.2.4 Basic Principles of Optimization

In general, optimization can be defined as any process which seeks to find the best solution to a problem. In its most primitive form, optimization can be nothing more than the repeated analysis of randomly determined mechanisms to find the best design. The designer ultimately decides that a satisfactory solution has been obtained with respect to the desired result and the previous design iterations.

For the purpose of this research, optimization is considered to be the rational process of finding the best possible solution to an objective subject to some number of *constraints*. This definition raises questions concerning what optimization methods are available and exactly how is the "best possible" solution to the problem defined.

There is a large and expanding base of knowledge concerning optimization methods. All numerical optimization methods rely on mathematical algorithms which are programmed into a digital computer. Fortunately, the availability and affordability of very powerful personal computers makes implementation of these methods accessible. Most mechanism optimization problems are nonlinear and thus fall into the category of

nonlinear mathematical programming.

Defining the "best possible" solution to a problem can be difficult. Consider, for example, a part that is to be designed for maximum strength and minimum weight. These two criteria tend to be competing characteristics and a compromise between the two objectives must ultimately be made. This situation will almost always arise when specifying multiple objective and constraints for an acceptable solution to a problem. It is then necessary to establish the relative importance or *weight* of each objective. Assigning these weights will likely involve some amount of subjective opinion. Even so, the mathematical solution obtained from optimal techniques will not always correspond to the most practical physical solution. Most mathematical models of physical systems contain some number of approximations and simplifying assumptions.

In general, there are five principal elements to an optimization problem (Reinholtz, 1983). These are listed briefly here and explained in the paragraphs that follow.

1. The choice of a conceptual design for the problem.
2. Developing a model of the physical system which can be analyzed to identify the design variables and the governing equations for the system.
3. Setting up a scalar function (often referred to as the *objective function*) of the design variables found above, which measure the quality of the design.
4. Specification of all necessary design constraints.
5. Determining the set of values of the design variables which produce the value

of the objective function from (3) and are consistent with all of the applied constraints from (4).

The first four elements in previous list constitute the *problem definition* while the fifth element requires the application of the optimization theory to solve the problem mathematically. The first element, choosing the conceptual design for the problem, refers to identifying what type or types of mechanisms such as cams, linkages, gears, etc. are suitable candidates for solving the problem being considered. In addition, it is necessary to specify the particular configuration of the chosen mechanism in order to proceed. For the work presented in this thesis, a four-bar linkage with a weight rigidly attached to the output link was chosen. This conceptual design is covered in detail in Chapter 2 - Problem Definition and Constraints.

In step (2) above, a physical model of the conceptual design is developed and then analyzed. In this case, basic mechanism theory is applied to the four-bar linkage to perform the position, velocity, and acceleration analyses of the four-bar as well as an analysis for the force-generating characteristics of the linkage. The analysis assumes that the links are rigid and have negligible weight when compared to the applied loads on the mechanism.

The last two elements of the problem definition are to establish an objective function and to define the problem constraints. The objective function is a scalar function that is used to quantitatively express the quality of the mechanism being designed. Evaluation of the objective function for a particular design yields a numerical result that

can be compared with values obtained for other designs to indicate the relative quality of successive designs. In many cases, including the problem addressed in this thesis, the objective function is defined as the sum-squared error of some relevant quantity at a discrete number of positions. For example, the objective function might be the sum-squared error of the deviation between the actual and the desired coupler point paths of a mechanism in a path generation problem. When multiple criteria are specified, a weighted combination of the different objectives can be built in to the objective function to attempt to satisfy all the objectives of the design. Determining the objective function and specifying constraints for a problem requires the designer to have a full understanding of the problem being addressed. The choice of a poor objective function will likely result in finding poor or unsatisfactory physical solutions to the problem.

1.3 Literature Review

In many applications of linkage design, input forces are of importance. For example, a linkage might be designed for use in a press mechanism, a nonlinear force actuator, or an exercise machine. Historically, there have been two major alternatives for dealing with the issue of forces at the linkage input. The first addresses input forces only after the linkage has been synthesized based on other design criteria, such as output position. Post-synthesis dynamic analysis of the linkage (typically including link masses and the effects of dynamic motion) allows the designer to evaluate the quality of the design based on force issues. Synthesized linkages which do not meet the force requirements are addressed either through multiple iterations of the position synthesis, so

called dynamic synthesis or dynamic time-response synthesis, or through equilibration. Dynamic synthesis of linkages is reviewed in a survey article by Starr (1974). Many solutions to the mini-max problem (minimization of the maximum input force) are of this type, for example Ogot and Gilmore (1991).

The second synthesis approach, and the focus of this thesis, addresses input force as an objective in the synthesis of the linkage. We have called this type of problem force-generating linkage synthesis. Force-generating linkage synthesis generally assumes massless links and negligible dynamic effects. The forces of interest are assumed to be generated by resistance elements such as springs and load weights.

In force-generating linkage synthesis, forces are accounted for directly in the synthesis procedure. Historically, these problems have been solved by either closed-form or numerical techniques.

Two types of closed-form precision point synthesis techniques have been employed. These are categorized as mixed position-force or force-only synthesis problems. When synthesizing for a mixture of force and position constraints, it is important to remember that synthesizing for a prescribed input/output force relationship is equivalent to specifying velocity ratios, since the mechanical advantage is the inverse of the velocity ratio. Therefore, these mixed constraint synthesis problems are considered to be synthesis for Infinitesimally Separated Positions (ISP)., or order synthesis (Sandor and Erdman, 1984). ISP synthesis for force was used by Gustavson (1968), for example, in the design of four-bar linkages for applications in postage meters and automobile trunk

hinges.

A second method for closed-form synthesis has been used in the solution of the force-only problem. This technique employs integration of the force constraint to form an equivalent position constraint based on the principle of conservation of energy. This is referred to as the *Integrated Force-Constraint* (IFC) approach. This technique is demonstrated by Tao (1964) and has recently been fully developed (Soper, et al., 1995; Soper, 1995).

Both of the closed-form techniques discussed above have disadvantages which have motivated the work presented in this thesis. In force-only applications, ISP synthesis consumes precision points at twice the rate of the IFC approach, because precision points are being used unnecessarily to control output position. The IFC approach uses precision points more efficiently in the force-only application; however, the approach is an indirect one. Because the precision points are in position (integrated force) the solution curve must be differentiated to evaluate its fit to the force-constraint curve. Differentiation increases error. Therefore, errors between the synthesized position curve and the objective curve will be magnified. This tends to make acceptable solution linkages difficult to find. It also implies that the linkage may be more susceptible to errors introduced by manufacturing tolerances for the same reason.

Closed-form methods do have two distinct advantages over optimal methods, however. The first advantage of a closed-form precision point approach is that it can clearly define the solution-space of the problem. In the case of applying optimal

techniques, the solution-space corresponding to a given set of design specifications may be poorly defined. This will typically result in an increased number of iterations during the optimization search. The second advantage is that closed-form techniques do not require an initial guess at the solution as a starting point for the solution search. This is often referred to as a *seed* for the optimization algorithm. As a general rule, good initial guesses will ultimately produce better, more practical solutions (Venkataraman, et al., 1992).

The inefficiencies of the closed-form techniques in dealing with the force-only constrained problem have motivated the examination of numerical techniques for synthesizing force-generating linkages.

Bokelberg and Gilmore (1990) employed numerical techniques to solve a force-only linkage synthesis problem for application in rehabilitation exercise machines. They did not, however, synthesize for an optimal (regressive) solution. Instead, they employed numerical techniques to solve the precision point synthesis problem. In their case, a set of implicit, nonlinear equations in both position and velocity were solved at the limiting number of precision points. Using this technique, a finite number of solutions can be found, but only one at a time, depending upon the initial guess at the solution. In addition to the limited set of solutions obtained, each solution will be subject to the same problems as the solutions generated by the closed-form techniques described above. Their technique is useful for applications where the development of closed-form equations would be too time consuming to justify.

Other authors do document the solution of force-generating linkage synthesis problems using optimal techniques, but none addresses the force-only problem. Midha, Turic, and Bosnik (1984) document several methods, some of them numerical, for linkage synthesis when the input forces were only of concern over a limited range of the output motion. Bagci and Rieser (1984) applied optimization to the synthesis of function generators satisfying derivatives of the generated displacements (i.e. velocity or force) at a finite number of design positions.

Compared to closed-form methods, optimal methods can handle a larger number of design parameters as well as any number of applied constraints on the design. Although closed-form methods yield exact solutions at a finite number of points, for the practical applications targeted here, best-fit solutions at an arbitrary number of points affords the designer greater flexibility. The optimal synthesis method developed in this thesis addresses the problem of force constraints without associated output positions.

2. Problem Definition and Constraints

2.1 Kinematic Model and Design Variables

The central thrust of the work presented in this thesis is to develop methodologies for designing linkages which produce specified static resistance curves. The particular application of such linkages is in the design of static weight-loaded exercise fitness equipment. The primary objective is to build linkage-based machines which produce resistance curves that closely match the strength of the human user for a given type of exercise.

In Section 1.2.4 of the previous chapter, the principles and five basic steps of applying optimization theory to mechanism design were discussed. For clarification, the first two steps involved were to choose a conceptual design for the problem and then to develop and analyze a model of the physical system to be used. The analysis step should identify the design variables and develop governing equations for the system.

The conceptual model chosen for the solution is a four-bar linkage. Deciding exactly what this linkage should look like in terms of a physical model is left to the designer. It should be noted, however, that there is certainly a tradeoff between the simplicity of the model and the accuracy and practicality of the results. In this case, linkages that produce the desired resistance curves are sought. There are several types of linkage models worth considering including various configurations of four-, five-, and six-bar linkages. While increasing the number of design parameters will typically afford

the designer more flexibility in finding quality solutions, it also increases the overall complexity of the resulting design. In addition, the analysis of the model may become difficult and the ease of producing the solution will increase. For these reasons, a basic four-link mechanism with a static weight fixed to the grounded output link was chosen as the conceptual model for the solution. The author's experience has shown that satisfactory resistance curves can be obtained using this configuration of a four-bar linkage. The approach developed in this thesis can be applied to more complicated linkage designs and other types of mechanisms.

2.1.1 Weighted Grounded Link Model

A conceptual model of the four-bar linkage to be synthesized is shown in Fig. 2-1. At the heart of the model is the four-bar linkage constructed by links l_1, l_2, l_3, and l_4. On the left side of the linkage, the input arm l_{in} is rigidly attached to link l_1 and offset by a

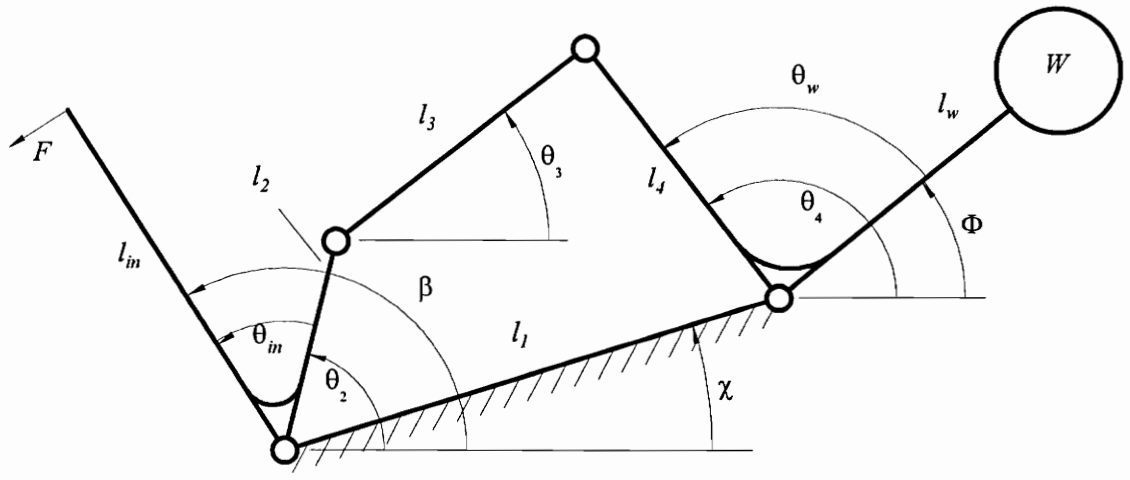


Figure 2-1, The Four Bar Linkage, Weighted Grounded Link Model

fixed offset angle θ_{in}. Similarly, on the right side of linkage, a static load W is attached to the weight arm l_w which is fixed to link l_4 at a fixed offset angle θ_w. The four-bar linkage transforms the load on the weight arm l_w to a nonlinear input arm l_{in} force as the linkage is moved through its range of motion. The goal of synthesis is to match the nondimensional resistance curve produced by the linkage, R, to the nondimensional strength curve of the user, S, over the range of motion of the exercise, β. Note in the diagram that β is always measured from horizontal.

The strength curve for an exercise is determined experimentally by measuring the amount of static force the user can exert at each angle β over the range of motion of the exercise. Although the input force F in Fig. 2-1 is usually applied directly to the input link l_{in} by the user, it may also be supplied by a secondary input. A practical example of this is the transmission of applied force from another four-bar linkage used for body guidance which is connected to the input link in Fig. 2-1. Either way, what is ultimately required for synthesis is the functional relationship between the user strength S and the input arm angle β. For the remainder of the thesis, it will be assumed that the measured force F is the perpendicular component of the force the user applies to the input link. This assumption is consistent with the way in which the data is typically collected in testing. However, for any case in which the force is applied in some other known direction, for example horizontally, the strength data could simply be transformed using trigonometric relationships.

When designing a force-generating mechanism, it is convenient to normalize the

strength curve with respect to the maximum force encountered over the range of motion of the exercise. The goal of synthesis is to match the nondimensional resistance curve of the linkage to the nondimensional strength curve of the user. The magnitude of the resistance is set by the amount of weight, W, applied to the output link. The resistance curve R is defined as the inverse mechanical advantage of the mechanism. This is also the ratio of the perpendicular input force F to the weight load W. Ideally

$$R = 1/\mu = F/W = S. \tag{2.1}$$

As discussed in Chapter 1, the mechanical advantage μ is defined as the torque or velocity ratio

$$\mu = \frac{T_{out}}{T_{in}} = \frac{\omega_{in}}{\omega_{out}}. \tag{2.2}$$

For the remainder of this thesis, it will be assumed that discussions of strength and resistance curves will refer to the nondimensionalized forms.

2.1.2 Strength Data Discussion

The user strength curve, S, for a given exercise is experimentally determined from a test apparatus. In order to properly set up this test device, the range of motion, β, and the length, l_{in}, of the input arm, must be known. The range of motion is usually dictated by the proper physiologic motion for a given exercise. For purposes of this thesis, it is assumed that this information is known at the outset of the synthesis procedure. The strength data is gathered at a finite number of points over the range of motion of the exercise. Static force exerted by the user at a given input arm position is measured using

either a load cell or a strain gauge calibrated to measure force. Collection of static rather than dynamic force measurements tends to produce a good estimate of the force the user can actually exert during motion of the exercise. This is true because these types of exercise machines are designed for slow, controlled movements rather than high-speed repetitions. Recall that a simplifying assumption has already been made to ignore the dynamic effects of the motion. The slow-speed of movement of the linkage is the reason this assumption is reasonable.

The strength data tests produce a series of discrete data points which, once normalized, can be plotted as the nondimensional strength, S, versus the input arm angle β as shown in Fig. 2-2. In cases when the raw strength data does not produce a smooth series of points, it may be beneficial to smooth out the data using a least-squares or other data-fitting method as shown in Fig. 2-3.

One reason that data collected from a strength test rig may not be produce a smooth curve is that it is sometimes difficult for the tested person to exert their maximum force at each test point. This is sometimes caused by decreasing muscle strength or fatigue during testing or an awkward arrangement of the test apparatus which makes it difficult or uncomfortable for the user to exert force. It is recommended that the strength tests be performed several times by multiple subjects because repeated strength testing allows for an average strength to be measured. In general, human strength curves should produce a smooth curve. The presence of large changes in strength over a small range of motion or test points which clearly lie outside the majority of the other test points tend to

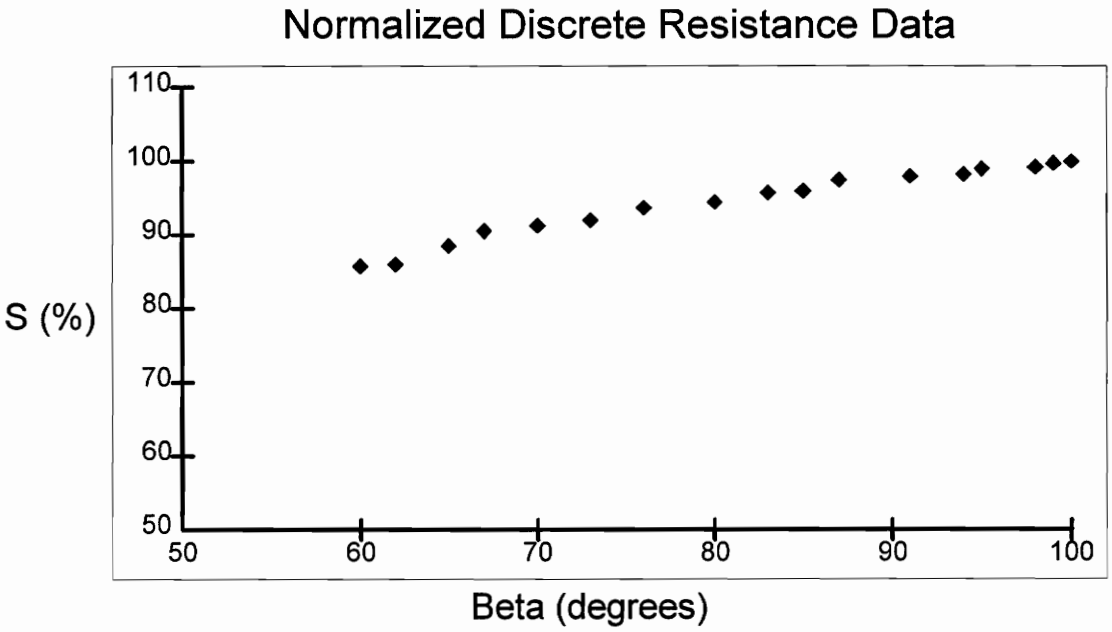


Figure 2-2, Typical User Strength Curve Data For a Compound Row Exercise

indicate a problem with the test.

2.1.3 Optimization Design Variables

It has previously been assumed that the designer will know the range of motion, β, of the linkage input arm as well as its length, l_{in}. The remaining linkage parameters labeled in Fig. 2-1 constitute the design variables of the optimization. These include the input and weight are offset angles θ_{in} and θ_{w}, respectively, the lengths of the base four-bar linkage l_1 through l_4, the weight arm length l_w, and the orientation of the ground link relative to the horizontal, χ. This gives a total of eight design variables which will be referred to as the *design vector* **X** with scalar components x_1 through x_8. These scalar components or optimization design variables are labeled in Fig. 2-4 along with the input

arm angle β.

2.2 Development of the Objective Function

For the four-bar synthesis problem to be solved using optimization, a basis for defining the optimum linkage must be specified. This basis is called the *objective*

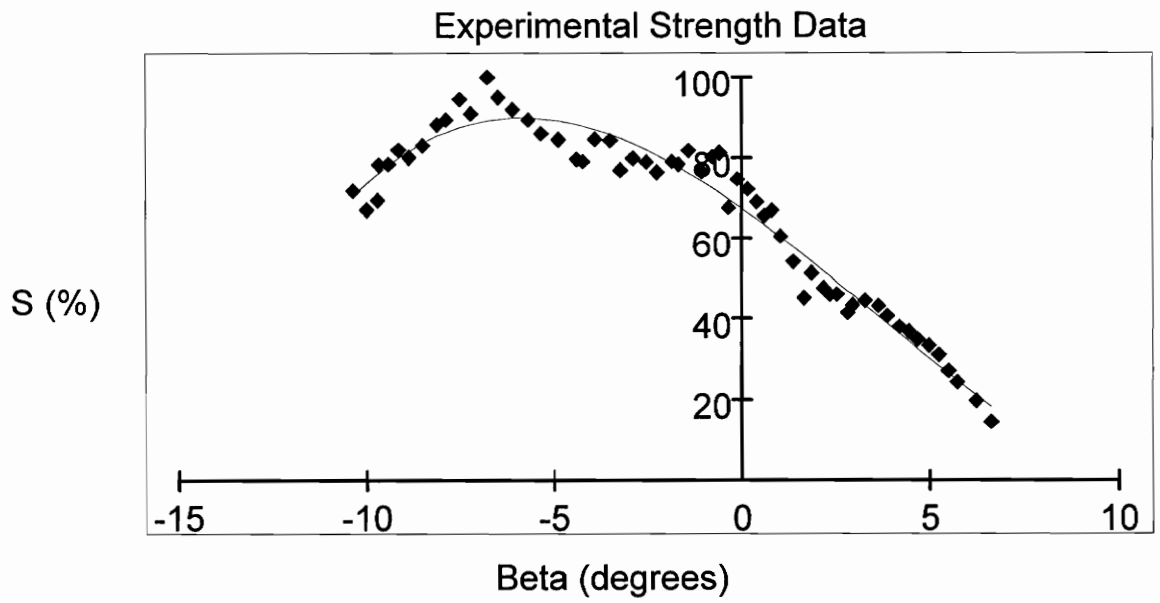


Figure 2-3, Rough Data With Cubic Polynomial Fit Overlay - Seated Calf Raise Exercise

function. It is a scalar function of the design variables whose numerical value is an indication of the "quality" of the system being considered. For the four-bar linkage synthesis problem addressed herein, it is defined as the summed-squared error between the force curve and the strength curve at some number of points spanning the input range. This is often referred to as the *structural error* of the mechanism. The motivation for

squaring the error is that it penalizes negative and positive error equally and it also penalizes larger errors at an increasing rate. Again, the design vector **X** for this problem consists of eight components, $x_1 \rightarrow x_8$ which are labeled in Fig. 2-4. Components x_1, x_2, x_3, and x_4 are the basic four-bar linkage dimensions. Component x_5 is the length of the

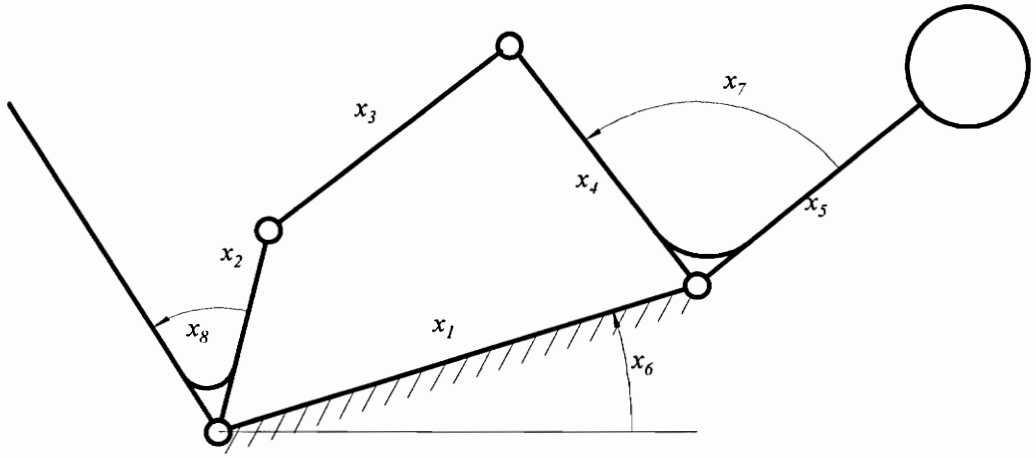


Figure 2-4, Optimization Design Variables

weight arm and angle x_6 is the base rotation of the linkage from horizontal. Angles x_7 and x_8 are the offset angles of the weight arm and the input arm, respectively. Recall that the length of the input arm l_{in} is a specified input in the formulation of the problem as is the desired user strength curve we are seeking to match. Because the four-bar linkage has two possible structural assemblies, the *closure* of the four-bar must also be chosen at the outset.

The optimization problem is stated mathematically as follows: Find

$$\mathbf{X} = \{x_1, x_2, \ldots, x_n\}^T, \tag{2.3}$$

which minimizes the objective function $f(\mathbf{X},\theta)$ subject to the inequality constraints

$$g_u(\mathbf{X},\theta) \geq 0, \qquad u = 1,2,...,r \tag{2.4}$$

and the equality constraints

$$h_v(\mathbf{X},\theta) = 0, \qquad v = 1,2,...,s \tag{2.5}$$

In this case, **X** is the design vector whose components are the design variables and θ is an independent set of input motion variables which have a known range of values. The input motion variables θ will usually represent quantities such as input link position or time. Note that because the objective function $f(\mathbf{X},\theta)$ and the constraint equations may be functions of the vector θ, it may be necessary to evaluate the objective function and the constraints for every value of the independent input parameter(s) θ. This is the case for the linkage synthesis problem presented in this thesis. The objective function and constraints must be evaluated for every input data point (β value) on the objective user strength curve.

The optimal linkage synthesis problem addressed in this thesis constitutes an eight-dimensional (eight design variables) nonlinear optimization problem. The objective function selected for this problem can be expressed as

$$\text{OF} = f(\mathbf{X};\beta,l_{in}) = \sum_{i=1}^{n} w_i[S(\beta)_i - S(\beta)_i^{desired}]^2, \tag{2.6}$$

where β is the input arm angle whose range of motion is known from the objective user strength curve and n is the number of points on this specified strength curve. The objective function is a measure of the deviation between the desired resistance, $S(\beta)^{\text{desired}}$, and the actual resistance produced by the linkage, $S(\beta)$, at each input arm angle β. The

variable w_i in the objective function is a weighting factor which could be assigned for each point on the curve to indicate the relative importance of minimizing deviation at that particular point. For example, it may be desirable to more closely match the strength curves at points at the extremes of the data.. By assigning weighting factors at the ends to be greater than one, the value of the objective function will be larger when the deviation is large near the endpoints. Because the objective function is being minimized, this will have the effect of guiding the optimization routine towards the desired result.

2.3 Analysis of the Linkage for the Resistance Curve

To evaluate the objective function for the optimization method in the previous section, it is necessary to analyze linkages to determine strength characteristics for a given set of design variables. For each input arm angle β, the resistance provided by linkage $S(\beta)$ must be computed. The analysis of the linkage model for this purpose is presented as follows. Applying the virtual work principle from Section 1.2.2 to the mechanism, assuming massless links and negligible dynamic effects, gives

$$F \cdot l_{in} \cdot \omega_2 - W \cdot v^w{}_y = 0, \tag{2.7}$$

where $v^w{}_y$ is the vertical component of the velocity of the weight W given by

$$v^w{}_y = l_w \cdot \omega_4 \cdot \cos(\Phi). \tag{2.8}$$

Substituting Equation 2.8 into Equation 2.7 yields

$$F \cdot l_{in} \cdot \omega_2 - W \cdot l_w \cdot \omega_4 \cdot \cos(\Phi) = 0. \tag{2.9}$$

Solving Equation 2.9 for the ratio F/W yields the desired resistance function

$$S(\beta) = \frac{l_w}{l_{in}} \cdot \frac{\omega_4}{\omega_2} \cdot \cos(\Phi(\beta)). \tag{2.10}$$

From a loop-closure velocity analysis it can be shown that

$$\omega_4 / \omega_2 = \frac{l_2}{l_4}\left(\frac{\sin(\theta_3)\cos(\theta_2) - \cos(\theta_3)\sin(\theta_2)}{\cos(\theta_4)\sin(\theta_3) - \cos(\theta_3)\sin(\theta_4)}\right). \tag{2.11}$$

Substituting this angular velocity ratio and the definition of $\Phi = \theta_4 - \theta_w$ into Equation 2.10 yields the final result for the resistance function

$$S(\beta) = \frac{l_w}{l_{in}} \cdot \frac{l_2}{l_4}\left(\frac{\sin(\theta_3)\cos(\theta_2) - \cos(\theta_3)\sin(\theta_2)}{\cos(\theta_4)\sin(\theta_3) - \cos(\theta_3)\sin(\theta_4)}\right) \cdot \cos(\theta_4 - \theta_w). \tag{2.12}$$

With Equation 2.12 the linkage resistance can be computed for any value of β. Notice that Equation 2.12 is a function of the design variables and the independent input parameter θ_2 which is related to the input arm angle β by the relationship $\theta_2 = \beta - \theta_{in}$. The four-bar position analysis presented in Section 1.2.1 is used to obtain the values of θ_3 and θ_4 as a function of θ_2 when using Equation 2.12.

2.4 Analysis of the Linkage for Coupler and Bearing Stresses

As will be discussed in the next section, two important considerations pertinent to the design of force-transmitting linkages are the stresses induced in the coupler link and at the grounded bearings. The coupler link can be subjected to either tensile or compressive stresses which should be checked to ensure that neither buckling nor tensile fracture occurs. Although stresses cannot actually be determined without knowledge of

the cross-sectional area of the relevant links or the applied weight load on the linkage, the nondimensional forces which produce these stresses can be calculated. The force in the coupler link is nondimensionalized with respect to the applied load W and will be denoted by F_3. Summing moments about the grounded pivot of the input link l_2 gives

$$R \cdot l_{in} - F_3 l_2 \sin(\theta_2 - \theta_3) = 0 \tag{2.13}$$

which is rearranged to yield

$$F_3 = \frac{l_{in}}{l_2} \cdot \frac{1}{\sin(\theta_2 - \theta_3)} R . \tag{2.14}$$

A positive sign for F_3 indicates tension while a negative sign indicates compression. Note that the F_3 depends upon the size of linkage because l_2 appears in the denominator of Equation 2.14. Scaling the linkage up in size will decrease the coupler force; scaling the linkage down increases the coupler force.

The nondimensionalized bearing forces at the input and output ground pivots, B_{in} and B_{out} are found by summing the forces on the two grounded links, l_2 and l_4. For the input ground

$$Rie^{i\beta} + F_3 e^{i\theta_3} + \vec{B}_{in} = 0 \tag{2.15}$$

which can be shown to yield

$$B_{in} = \sqrt{(R\sin(\beta) - F_3\cos(\theta_3))^2 + (R\cos(\beta) + F_3\sin(\theta_3))^2} . \tag{2.16}$$

Similarly for the output ground pivot

$$-i - F_3 e^{i\theta_3} + \vec{B}_{out} = 0 \tag{2.17}$$

which yields

$$B_{out} = \sqrt{(F_3 \cos(\theta_3))^2 + (F_3 \sin(\theta_3) + 1)^2} \ . \qquad (2.18)$$

F_3 in Equations 2.15 - 2.18 refers to the force in the coupler link computed from Equation 2.14.

2.5 Design Constraints and Penalty Functions

The primary objective of the linkage synthesis technique being developed is to match the resistance curve generated by the linkage to the user strength curve for a given exercise. As mentioned before, however, there are a number of secondary considerations and design constraints that also need to be addressed. Often, a possible solution linkage that is an excellent fit in terms of its strength characteristics is rejected due to these other considerations. Although some of these other constraints are also qualitative in nature, many of them are not. The various constraints which are applicable to the problem are discussed in the next section.

2.5.1 Applicable Design Constraints

The constraints that apply to the design of Nautilus fitness equipment are related to factors such as performance, aesthetics, and practical considerations. Performance constraints relate to the quantitative performance of the machine based on a complete analysis of the system. These include such issues as quantitative limits on the stresses developed in certain parts of the machine. Aesthetic constraints have more to do with qualitative opinion and visual appearance to the user. These types of considerations tend to be more psychological in the sense that the user wants to use something that appears to

be simple and effective. Practical constraints address issues such as ease and cost of manufacture, retrofitting, safety, number of parts, assembly, and shipping.

2.5.1.1 Performance Constraints

1. A common constraint applied to the design of any four-bar linkage to be used for force-transmission is to keep the transmission angle γ of the linkage as close to 90° as possible. The transmission angle is defined as the acute angle between the coupler link (l_3) and the output link (l_4) shown in Fig. 2-5. Maintaining the transmission angle as close to 90° as possible ensures the most effective transfer of force in the coupler link to torque about the grounded output arm pivot. Acceptable values of γ are typically no less than 30°. The inequality function then for this constraint is

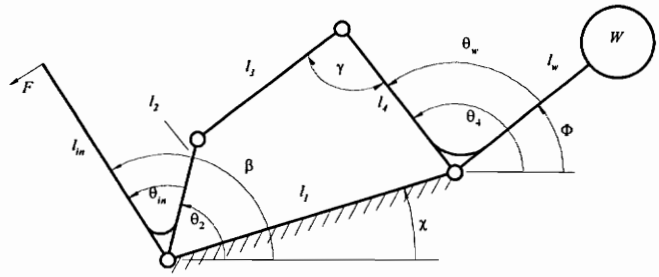


Figure 2-5, Transmission Angle γ of the Four-Bar

$$\gamma \geq 30^{\circ}. \tag{2.19}$$

Even though γ is not one of our design variables, it is a function of the design variables and the independent input parameter β. The relationship can be found from the position analysis of the four-bar to be

$$\gamma = \cos^{-1}\left(\frac{l_2^2 - l_3^2 - l_4^2 + l_1^2 - 2 \cdot l_2 \cdot \cos(\beta - \theta_{in})}{2 \cdot l_3 \cdot l_4}\right). \tag{2.20}$$

2. The designer must be aware of the limits of material strength when designing the linkage. The maximum axial stress developed in the coupler link (l_3) as well as the bearing stresses at the ground pivot locations need to be kept at acceptable limits to avoid excessive fatigue and ultimate failure. Analysis of the linkage for these static stresses was presented in Section 2.4.

2.5.1.2 Practical Constraints

1. Nautilus designers use a number of common frames for the various machines they manufacture. The benefits of this are that frame design work is minimized, cost is reduced by using common parts, and no new frame inventory is required. Hence, whenever possible, a new machine should make use of one of these existing frames. The new linkage design should remain completely within the envelope of the frame over the range of motion to avoid collisions with the user or bystanders. Also, the grounded pivots of the linkage should be positioned close to convenient locations on the frame where they can be easily anchored.
2. It is desirable that the new linkage designs be suitable for retrofit into existing cam-based equipment. In order to achieve this, the linkage must fit within the existing frame and must be able to use the existing ground pivot anchoring locations. This allows for simple change-out replacement of the cam system with the new linkage assembly.
3. Reasonable limits on the lengths of the links in the mechanism should be applied. Very short links are difficult to manufacture and present potential interference

problems with other parts which can produce hazardous pinch points. On the other hand, longer links require more material to manufacture which increases cost and adds mass. Because the mass of the links has been assumed to be small compared to the applied load weight, it is important to keep the mass of the links small to achieve the desired performance. Longer links are also more likely to violate the frame constraints described above.

4. The vertical position of the applied load weights should be kept as low to the floor as possible. The applied loads will account for a significant portion of the total weight of the machine, especially for stronger users lifting more weight. For stability, it is important to maintain a low center of gravity of the machine over the range of motion of the exercise. If the weight is raised too high above the floor, the stability of the machine is compromised and the risk of the machine toppling over is increased. Large vertical motions of the weight also mean that the weight has a greater distance to fall if it is dropped from the top of the stroke. This may present a danger to both the user and to those standing near by. This constraint can be integrated into the optimization by enforcing inequality limits on the starting and ending output angles Φ_{min} and Φ_{max}, respectively. A range of acceptable output angles of the weight arm might be defined as

$$-20^\circ \leq \Phi_{\min} \leq 20^\circ, \tag{2.21}$$

$$10^\circ \leq \Phi_{\max} \leq 50^\circ. \tag{2.22}$$

5. Horizontal motion of the weight does not effect the input force, because no work is done. However, the dynamic effects, which have been assumed to be negligible in the analysis of the linkage, may become significant for large horizontal motion of the weight. This also negatively affects the machine's stability and reduces accuracy of the computed resistance curve.

A number of constraints and considerations come into play when designing a new machine, so it may be difficult to achieve all the design constraints simultaneously. However, what is being sought is an acceptable combination of all of these factors. As noted earlier, some of the constraints can be integrated into the numerical optimization routine to guide the search procedure to acceptable solutions. Unfortunately, many non-quantitative design considerations must be considered by the designer in selecting the best solution linkage. In cases when the choice of design solutions proves inadequate, the designer can relax constraints and re-run the optimization to see if acceptable solutions can be found.

2.5.2 Penalty Function Assessments

The previous discussion has covered some of the typical constraints and considerations which are relevant to the design of force-generating linkages. Inequality constraints were explained in the context of controlling transmission angles, link lengths, weight arm output angles, and so on. However, the actual integration of these inequalities was not discussed in detail. These quantitative constraints are integrated into the objective function using *penalty functions*. Penalty functions are scalar functions

whose value is computed and added onto the value of the objective function. A penalty function's value is usually assessed as follows:

1. The penalty function has a value of zero for designs which do not violate the constraint the penalty is attempting to enforce.
2. The penalty function is assigned a value which is either:
   i. Fixed at a sufficiently large enough value such that a design which violates the constraint will never be considered optimal.
   ii. Assigned a large base value plus an additional value based on how severely the design violates the constraint. For example, consider a linkage whose coupler link length is limited to 10 units. If a design has a coupler length of 14 units, a penalty might be assessed as 50 penalty points plus 20 penalty points for every unit of length the design violates the constraint. This would yield a penalty of

$$P = 50 + (20 \cdot (14 - 10)) = 130. \tag{2.23}$$

   The advantage of this method is that it establishes a quantitative degree of constraint violation which will help to guide the optimization routine in the direction of an acceptable solution. Solutions that are way out of range will be penalized a larger amount.

Any number of penalty functions may be integrated into the design optimization. It should be noted, though, that as the number of applied constraints increases, the time required to converge to a solution increases and the chance of finding an acceptable solution decreases. Again, the designer can always relax the number and limits of the

constraints on the problem to increase the size of the solution set.

The definition of the synthesis problem is now complete. Subsequent chapters will address the application of optimal techniques to the solution example problems. Discussion of a Microsoft-Windows-based linkage design package developed based on the optimization theory is included in Chapter 3.

3. Optimization-Based Synthesis

The linkage synthesis problem addressed in this research is cast as a nonlinear optimization problem. Nonlinearity enters the problem because the analysis equations of the linkage and hence the objective function are nonlinear in some of the design variables. The advantages of employing optimal techniques versus closed-form analytical methods were discussed in Chapter 1. A scalar objective function based on the resistance-generating properties of the linkage as well as other applied constraints was developed in the previous chapter. The ultimate goal of the optimization technique is to minimize the value of this function.

3.1 Optimization Techniques

3.1.1 Unconstrained Nonlinear Optimization

The nonlinear optimization techniques employed in the solution of the problem were developed for solving unconstrained problems. However, the methods have been extended to handle constraints by using penalty function assessments, which consider the applied constraints directly when computing values of the objective function.

The two major groups of nonlinear minimization techniques are *direct search* methods and *descent* methods. Descent methods are often referred to as *first order* or *gradient* methods because they require either the analytical or numerical partial derivatives of the objection function with respect to the design variables while direct

search methods do not. Direct search methods are classified as *zeroth order*. A summary of the common methods within each group is presented in Table 3.1 (Reinholtz, 1983).

A lengthy discussion of each of the methods listed in the table constitutes a text in itself and as such is excluded. As one might expect, the random search and grid search techniques tend to be inefficient. These methods are not without merit, however, as they often find good use in minimizing the more complex discontinuous, nondifferentiable, or

Table 3.1, Unconstrained Nonlinear Optimization Techniques

Direct Search Methods	Descent (Gradient) Methods
Random search	Steepest descent method
Grid search	Conjugate gradient method (Fletcher -Reeves)
Univariate search	Newton's method
Pattern search (Hooke and Jeeves method, Powell's method)	Variable metric method (Davidson-Fletcher-Powell)
Rosenbrock's method	
Simplex method	

abruptly varying objective functions. These methods may also be useful in obtaining a good starting point or *seed* for another more efficient method. Use of more than one method to solve a given problem is a good way to improve the overall solution efficiency.

Eason and Fenton (1974) present a useful comparison of the many various numerical optimization techniques employed in the solution of mechanical design problems. Their primary basis for comparison was that an ideal optimization code should solve virtually any problem conveniently and with minimal cost. Although no one single

code which they tested met this requirement perfectly there were several which were worth considering and one in particular which stood out among the group. The method which they found most satisfactory was the pattern-search method of Hooke and Jeeves. Their technique has been shown to be effective in finding solutions to mechanism optimization problems and as such has been chosen as the optimization technique employed in solving the problem considered in this thesis. The Hooke and Jeeves method is noted for of its robustness, simplicity, and its ability to handle a wide variety of problems when compared with classical optimization methods.

3.2 Optimization Implementation

The following sub-sections are devoted to discussing the optimization technique employed in the solution of the synthesis problem, choosing initial design vectors as seeds to the optimization, and specifying convergence limits for the optimization.

3.2.1 Hooke and Jeeves Nonlinear Optimization Technique

The Hooke and Jeeves optimization technique is a straightforward, direct search numerical technique. The expression "direct search" is used to describe the sequential comparison of objective function values without calculation of the function derivative. Objective function values for each trial solution are compared with the "best" solution obtained up to that point. A strategy for determining what the next trial solution should be can be developed based on the earlier results. Perhaps the greatest merit of the direct search methods are that they emphasize the use of simple, repeatable strategies rather

than complex tactics. Since the direct search methods tend to use repeatable identical arithmetic operations with simple logic, they are also well suited for computer implementation. Because it is a direct search method, the Hooke and Jeeves technique does not require the computation of the objective function gradient.

The Hooke and Jeeves method uses a sequential stepping technique which consists of alternating exploratory and pattern moves. The exploratory move is made to determine the local behavior of the objective function in the solution space, while the pattern move makes use of information from the exploratory move to attempt to make a larger step in the direction determined by the exploratory moves (Hooke and Jeeves, 1961).

Recall from Section 2.1.3 that the linkage synthesis problem being solved constitutes an eight-dimensional design space. That is, there are eight scalar design parameters x_1 through x_8 in the design vector $\mathbf{X}$ for which values are being sought. Application of the Hooke and Jeeves procedure is as follows:

1. An initial starting point (design vector) in design space $\mathbf{X}^0$ is chosen and the objective function is evaluated at this point. A discussion of how to choose an initial starting point $\mathbf{X}^0$ is included in the next section. The initial and minimum step sizes Δx_i and Δx_{imin}, respectively, for each element of the design vector x_1 through x_8 must be specified before any move is made in the design space. Once all of the inputs to the optimization have been established, an exploratory move is made in the first design parameter by incrementing by the pre-specified step Δx_1.

The objective function is evaluated at the new point and compared to the previous value of the objective function. If it has improved, the point is retained as the new base point, otherwise a negative step $-\Delta x_1$ is taken and the objective function then reevaluated. The base point is again retained only if the objective function at this point has improved. In the case of neither step being successful, no move is made in solution space. After each exploration, the value of the step Δx is modified. A successful exploratory move results in an increase in step size and a failed exploration results in a decrease. This exploratory procedure is performed for each of the design parameters x_1 through x_8 to get the local behavior of the objective function.

2. The next step is to repeat all of the successful exploratory moves at once from the most recently retained point. This is referred to as a pattern move and, like the exploratory move, the new point in solution space is retained if the value of the objective function has improved compared to the most recently recorded value. This pattern move can also be repeated until the objective function is no longer improving because often times, a given pattern move will be successful several times. However, when the pattern move fails, the exploration begins again with the first design parameter at the most recently retained point.
3. This process of making exploratory and pattern moves is repeated until all of the step values Δx_i are below their pre-specified limits Δx_{min}. Once iteration has ended, a local minimum has been reached. A subsequent section discusses the

evaluation of the converged local minimum. There is no assurance that a convergence constitutes an acceptable solution. This is because the optimization procedure locates local minima and theoretically cannot identify a global minimum. Repeatedly performing the optimization with a different initial design vector each time will produce a set of converged candidate solutions that can be evaluated relative to each other based on the value of their objective functions and other practical criteria.

A flowchart of the optimization solution algorithm is shown in Fig. 3-1. It should be noted that no numerical optimization method is capable of discerning local from global minima. As an example, given ten different starting points $\mathbf{X}^0$, the Hooke and Jeeves method will usually converge to ten different local minima. This is not necessarily a problem when one considers the nature of the types of problems being addressed. In most cases, defining what the "optimum" solution to a problem really is would be difficult because there can be numerous details and considerations surrounding a given problem which may be difficult or impossible to include in the mathematical formulation of the problem. For the purposes of this research, in particular, practical solutions are being sought rather than the absolute "best" solution. It is better that the designer have a collection of candidate solutions (from converged local minima) that can be evaluated individually according to more qualitative criteria that were not considered in the mathematical optimization of the problem.

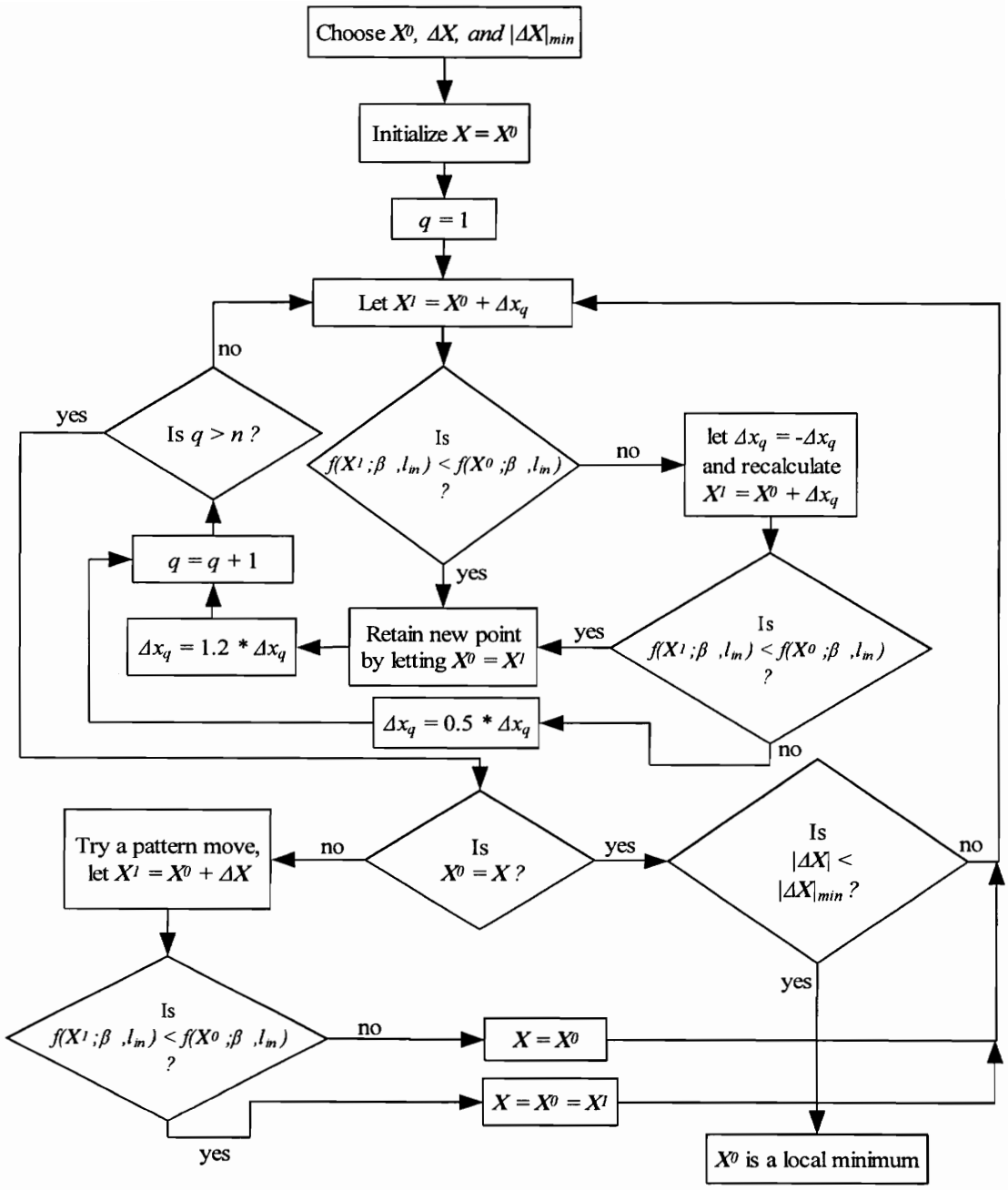


Figure 3-1, Hooke and Jeeves Optimization Flowchart

3.2.2 Choosing an Initial Design Vector

In Step 1 of the Hooke and Jeeves optimization method, the need for a starting point or initial seed for the mathematical procedure is established. This is shown as the topmost block in the optimization flowchart where $\mathbf{X}^0$ is chosen. Choice of this initial starting point for the optimization is important because a good initial guess at the solution will usually result in a faster convergence. Because the Hooke and Jeeves technique is a direct search method which follows a pre-programmed search routine, control over the search for solutions is relinquished after the optimization program is started. As a result, having some idea what acceptable solutions might look like before starting the procedure can increase the chance of realizing a quick convergence to an acceptable solution.

While it is not required that the seed to the optimization be close to an acceptable solution in the design space, a good seed will tend to produce better solutions in less time. Unfortunately, there is no one specific method of choosing an initial seed for the optimization that will guarantee a good solution will be found. On the other hand, there a number of clever techniques which are sometimes useful in determining where to start the optimization. Perhaps the best way to determine an initial design vector $\mathbf{X}^0$ is to use slightly perturbed results from prior optimization searches or the results from a closed-form synthesis process, such as the one presented by Soper (1995). The optimization procedure can be highly sensitive to small changes in the initial design variables and may converge to better solutions with minimal changes to the initial seed.

3.3 Software Implementation

A Microsoft Windows-based linkage design package has been developed based on the Hooke and Jeeves optimization theory. The purpose of the software is to automate and expedite both the synthesis and the analysis of force-generating four-bar linkages using the Windows interface. The program is highly interactive and graphical in nature, which allows the designer to quickly identify candidate solutions to a given synthesis problem and to perform on-screen analyses of each solution.

3.3.1 Program Description

The linkage optimization software was designed for two modes of operation. The first of these is the synthesis mode, in which the designer specifies the problem objectives, constraints, and other parameters required for the synthesis procedure. The second mode is the analysis mode, which is used to more closely examine and evaluate candidate solutions which were identified during the synthesis mode of operation.

3.3.1.1 Synthesis Mode Input and Output

The synthesis mode of the optimization software is used to collect information from the designer regarding the linkage to be synthesized. The objective strength curve which the linkage is designed to match and the length of the linkage input arm, l_{in}, must be specified before the optimization procedure can begin. The start and end angles of the input arm β_{start} and β_{end}, respectively, are assumed to be the two extreme data points on

the objective force profile.

There are also several optional input parameters with default values that can be modified if desired. A list of these optional program inputs and their default values follows:

1. Number of random initial seed design vectors to generate and optimize (default: 10,000).
2. Maximum acceptable error for a converged solution - used to determine if a convergence is saved to a file of candidate solutions for later analysis (default: 0.1).
3. Initial optimization step sizes Δx_i for design variables (default: 0.01).

4. Minimum optimization step sizes Δx_{imin} for design variables - used to determine convergence to a local minimum (default: 0.00001).
5. Constraints on acceptable linkage output angles Φ_{start} and Φ_{end} (default: not constrained and no range specified).
6. Choice of which of the eight design variables will be optimized - allows one or more of the design variables to be remain a constant value during the optimization (default: all variables to be optimized).

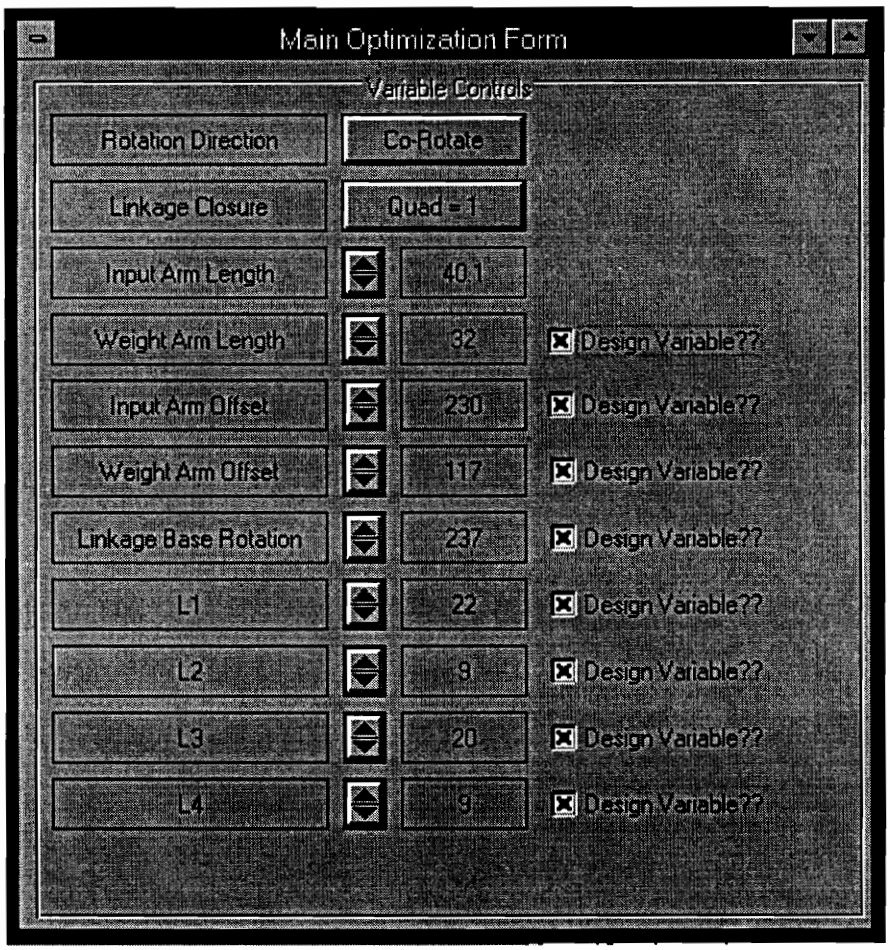


Figure 3-2, Main Optimization Input Screen

7. Choice of the closure of the linkage - the value of ξ discussed in Chapter 1 is either +1 or -1, which determines how the linkage assembles physically (default: ξ=+1).

8. Choice of a range of allowable values for random selection of design variables can be made - this is discussed in further detail in the next section on program flow (default: 0 to 100 length units for length variables and -360° to +360° for angular variables).

If no change is made to an optional parameter, the default value is used once the optimization algorithm is begun. Figure 3-2 is a picture of the main optimization screen. This screen is used to specify the initial design vector $\mathbf{X}^0$, the closure of the linkage, and which of the eight design variables are to be optimized.

3.3.1.2 Synthesis Mode Program Flow

A flowchart describing the optimization program flow during the synthesis mode is shown in Fig. 3-3. A brief description of the purpose and functioning of each block follows:

1. The required program inputs are entered into the program.
2. The optional program inputs are modified as necessary or default values are used.
3. The optimization program begins.
4. The initial guess $\mathbf{X}^0$ is analyzed. This consists of computing and plotting the resistance curve of the guess linkage, computing the objective function and assessing any constraints and penalties, and drawing the initial linkage in the graphics window.
5. The Hooke and Jeeves optimization routine is run until it converges to a local minimum. The converged solution linkage is then drawn in the graphics window and the resistance curve for the linkage is drawn as an overlay on the objective strength curve for comparison. Relevant information about the convergence is displayed in

the text output window.

6. If the objective function value is less than or equal to the maximum acceptable error (this is an optional program input with a default value), the converged solution is written to an output file for later use in the analysis mode of the software. Otherwise, the convergence information is discarded.
7. If the maximum number of optimization runs (also an optional input with default) has been reached, the output file is closed and the optimization ceases. Otherwise, optimization continues by taking a "random leap" in the design space.
8. The "random leap" refers to using a random number generator to automatically select values for the design variables so that the Hooke and Jeeves optimization can be re-executed with a new initial guess. The previous section presented a discussion of the optional inputs to the program. One of these inputs is the specification of a range of allowable values that a particular design variable can have. For example, the weight arm length l_w could be constrained to have an initial random value in the range of 3 to 20 length units. A similar range can be specified for each of the eight design variables. If no allowable range was specified, reasonable default values have been built into the program. Specifically, angular variables are constrained to lie between $-360°$ and $+360°$, and length variables are constrained to lie between 2 and 50 length units.
9. The new initial starting point for the optimization generated by the random leap is then analyzed and displayed as in Step 4 above.

10. The Hooke and Jeeves optimization is then re-executed with the new initial seed. Steps 5 through 9 are repeated until the number of optimization runs reaches the pre-specified maximum number of executions.

The output file created during synthesis mode contains a database of all of the converged local minima whose objective functions values were less than or equal to the maximum acceptable error specified at the outset of the program (an optional input with a default value). This database is used as the input file in the analysis mode of the software which is detailed in the following sections.

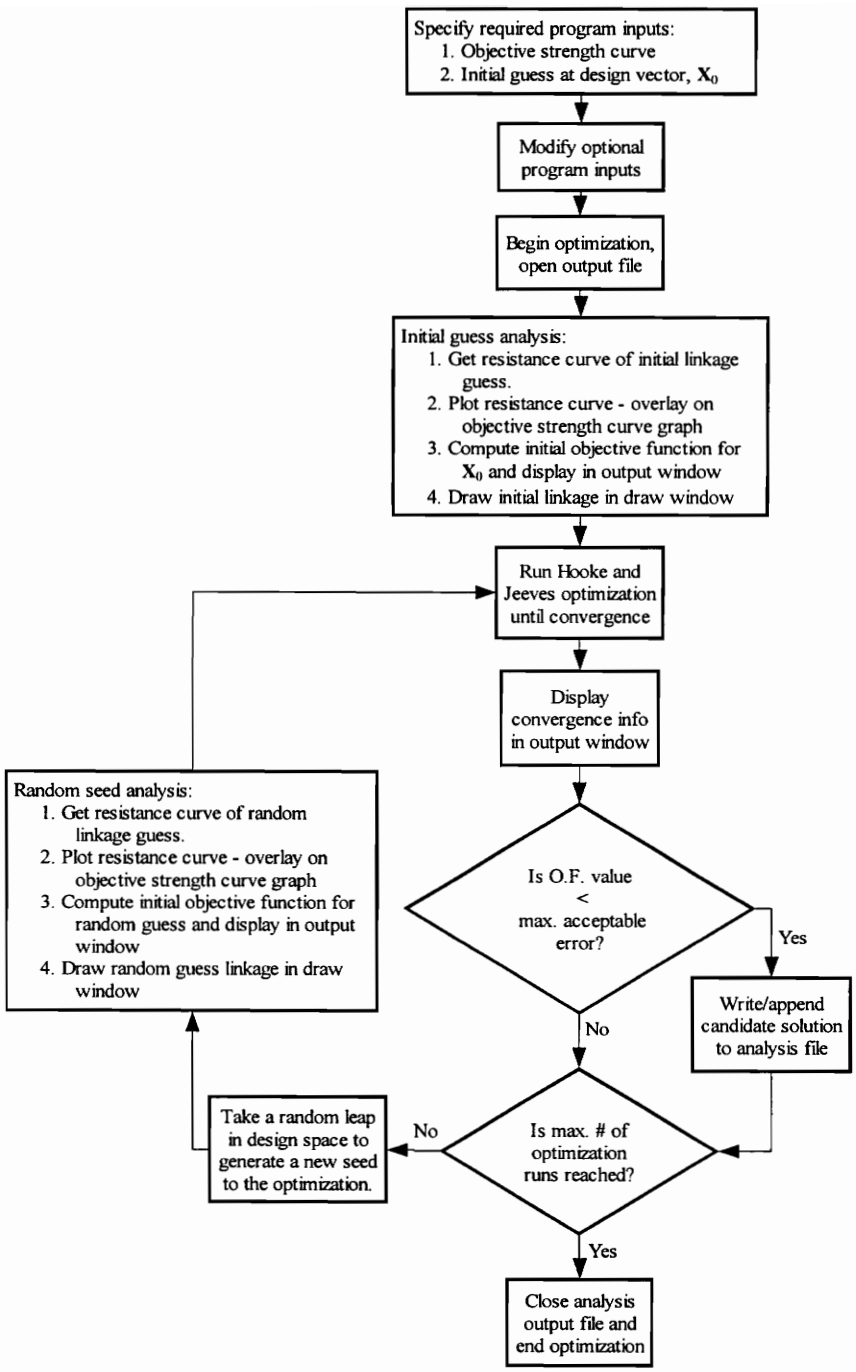


Figure 3-3, Software Synthesis Mode - Flow Diagram

3.3.1.3 Analysis Mode Input and Output

Each of the converged candidate solutions in the output file from the synthesis can be inspected individually in analysis mode. The purpose of analysis mode is to enable the designer to review a set of candidate solutions graphically and to make decisions concerning which linkage is best for its intended purpose. Although a number of mathematical constraints and or criteria may have been implemented into the synthesis portion of the program, there may be other practical and performance criteria that may not have been included. The analysis code allows the designer to see an on-screen animation of the linkage over its range of motion. Additionally, the abilities to zoom and pan in the graphics window and to rotate and scale the *base four-bar* linkage interactively are built into the analysis code. The base four-bar refers to the assembly of links l_1 through l_4. The resistance curve generated by the linkage is independent of the scale and rotation of the base four-bar linkage. This capability is particularly useful to the designer because it may help to identify different ways of installing the linkage into a machine.

Note that, while scaling the base linkage up or down does not alter the resistance curve generated by the linkage, it does affect the bearing forces and the force in the coupler link l_3. Scaling the linkage up in size will decrease these forces while scaling the linkage down will increase these forces. Figure 3-4 shows two linkages which produce equivalent strength curves. Note that the base four-bar of second linkage is rotated 90° CCW from the first linkage and is also scaled by a factor of 2.

Another element of analysis mode is the ability to adjust any of the design variables of a given candidate solution and to see the resulting resistance curve re-computed and displayed immediately. The new objective function value is reported for

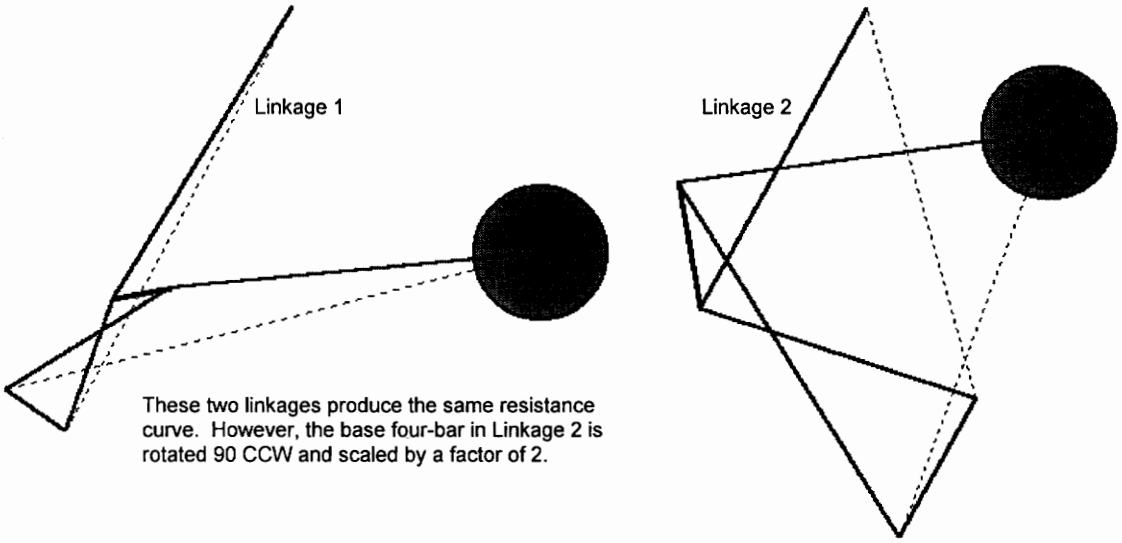


Figure 3-4, Linkages With Equivalent Resistance Curves

comparison. For example, the designer may want to see what effect altering the length of the ground link l_1 will have on the resistance curve.

4. Example Problem

4.1 Design of a Compound Row Machine

The design of a linkage for a weight-loaded exercise machine is presented as a practical application of the synthesis technique. The solution linkage is for use in a new compound rowing machine which targets major muscle groups in the upper back. The strength data for this example is taken directly from Soper, et al. (1995) and is shown in Table 4.1. A plot of this data is shown in Fig. 4-1. The purpose of the synthesis is to produce a linkage which generates a resistance curve that closely matches this user strength profile. The objective function defined for the problem attempts to minimize the structural error between the user strength curve and the machine resistance curve.

Table 4.1, Strength Data for Compound Row Exercise

Input Angle, β (degrees)	Strength Data, S (%)
60	85.75
62	86
65	88.5
67	90.5
70	91.25
73	92.0
76	93.75
80	94.5
83	95.75
85	96.0
87	97.5
91	98.0
94	98.25
95	99.0
98	99.25
99	99.75
100	100.0

4.1.1 Input Information and Constraints

The minimum required inputs to the software are the objective strength curve and the length of the input arm, l_{in}. From Table 4.1, the range of β for this strength profile is from 60° to 100°. Based on other information relevant to the problem, the input arm

length is chosen to be 40.1 inches. The default number of optimization convergences is changed from 10000 to 50 in order to reduce the computation time. All other optional inputs are left at the default values (See Section 3.3.1.1).

The constraints applied to the problem are used to control the range of the output angle, Φ. Specifically, the output start and end angles Φ_{start} and Φ_{end}, respectively, are constrained such that

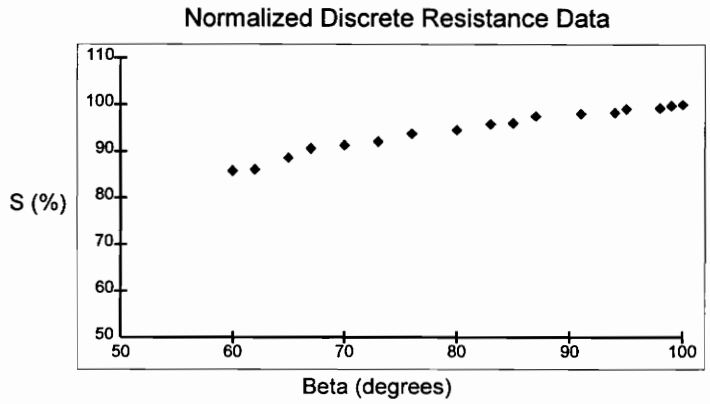


Figure 4-1, Strength Curve for a Compound Row Exercise

$$-10^{\circ} \leq \Phi_{start} \leq 30^{\circ}, \tag{4.1}$$

$$20^{\circ} \leq \Phi_{end} \leq 70^{\circ}. \tag{4.2}$$

The penalty for violation of each of these constraints is 1000 units per degree outside of the range. Penalizing designs that are further out of range more than designs that are close to within range has the effect of guiding the optimization away from unacceptable areas in the solution space.

Choice of an initial design vector is required as a starting point for the optimization. For this problem, previous design work based on closed-form techniques produced results from which the starting point for the optimization is taken. The initial

starting design vector $\mathbf{X}^0$ is shown in Table 4.1 with the corresponding value of the objective function. For this example, all eight parameters are treated as design parameters and will be optimized. Figure 4-2 shows a plot of the resistance curve generated by the initial solution guess detailed in Table 4.1. The objective strength curve is overlaid for comparison.

The program inputs are complete and the optimization routine is begun. The optimization routine is set to converge 50 times before stopping. Between each

Table 4.2, Initial Guess Design Vector

Design Parameter	Description	Initial Value
x_1	link 1 length, l_1	22 in.
x_2	link 2 length, l_2	9 in.
x_3	link 3 length, l_3	20 in.
x_4	link 4 length, l_4	9 in.
x_5	weight arm length, l_w	32 in.
x_6	linkage base rotation, χ	237°
x_7	weight arm offset angle, θ_w	117°
x_8	input arm offset angle, θ_{in}	230°
Objective Function Value $f(\mathbf{X}^0)$		0.2406 units

convergence, the program selects random values for each of the design variables being optimized to use in the next execution. Reasonable limits have been placed on the range of these random values for each variable (See Section 3.3.1.1). Only those optimization runs that converge to objective function values less than the maximum allowable are saved to an analysis file. Upon completion of the optimization, 4 candidate solutions

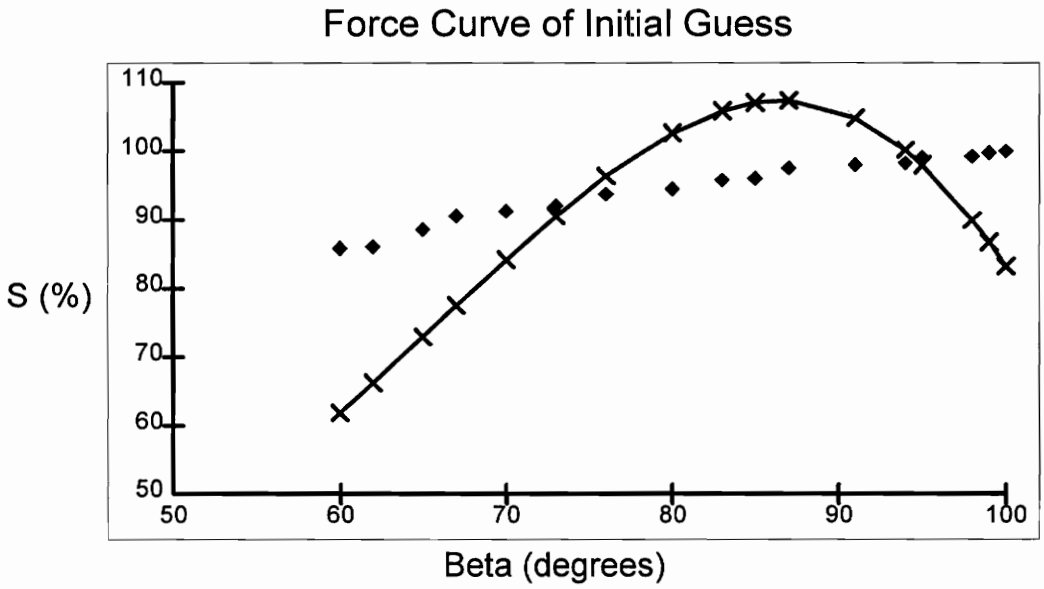


Figure 4-2, Resistance Curve of Initial Guess Linkage

have been written to the output file for further review and analysis. In other words, 4 out of the 50 optimization runs met the specified design optimization criteria. Of these four, two are duplicate solutions, which leaves only three unique candidates. Evaluation of these three candidates in the analysis mode of the software yields the chosen solution. The design vector of this solution is shown in Table 4.2 and the corresponding resistance curve generated by the solution is shown in Fig. 4-3 with an overlay of the objective strength curve. Figure 4-4 is a plot of the force in the coupler link over the range of motion and Fig. 4-5 is a drawing of the solution linkage.

Table 4.3, Solution Design Vector

Design Parameter	Description	Initial Value
x_1	link 1 length, l_1	6.95 in.
x_2	link 2 length, l_2	16.24 in.
x_3	link 3 length, l_3	8.61 in.
x_4	link 4 length, l_4	22.86 in.
x_5	weight arm length, l_w	45 in.
x_6	linkage base rotation, χ	10.47°
x_7	weight arm offset angle, θ_w	-154.26°
x_8	input arm offset angle, θ_{in}	167.42°
Objective Function Value $f(\mathbf{X}^{final})$		7.645E-3 units

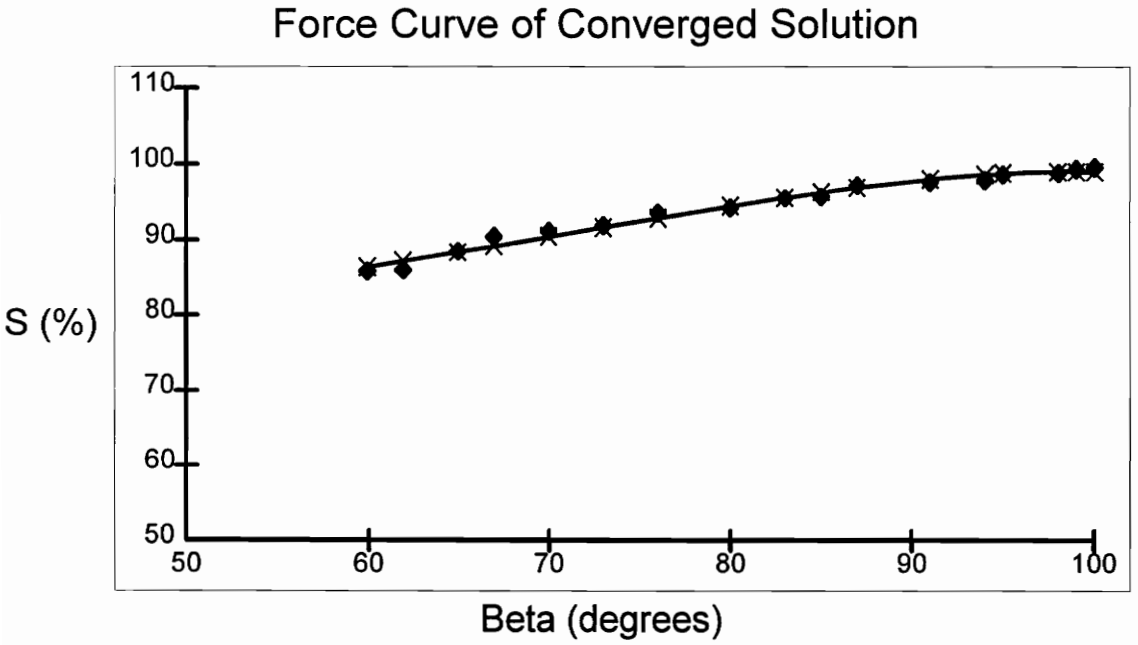


Figure 4-3, Resistance Curve of Solution Linkage

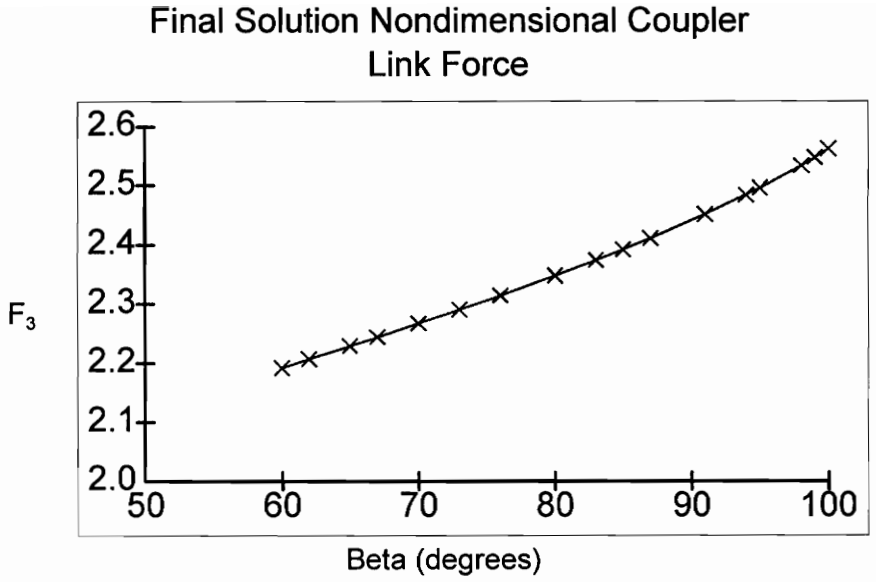


Figure 4-4, Solution Linkage Coupler Link Force

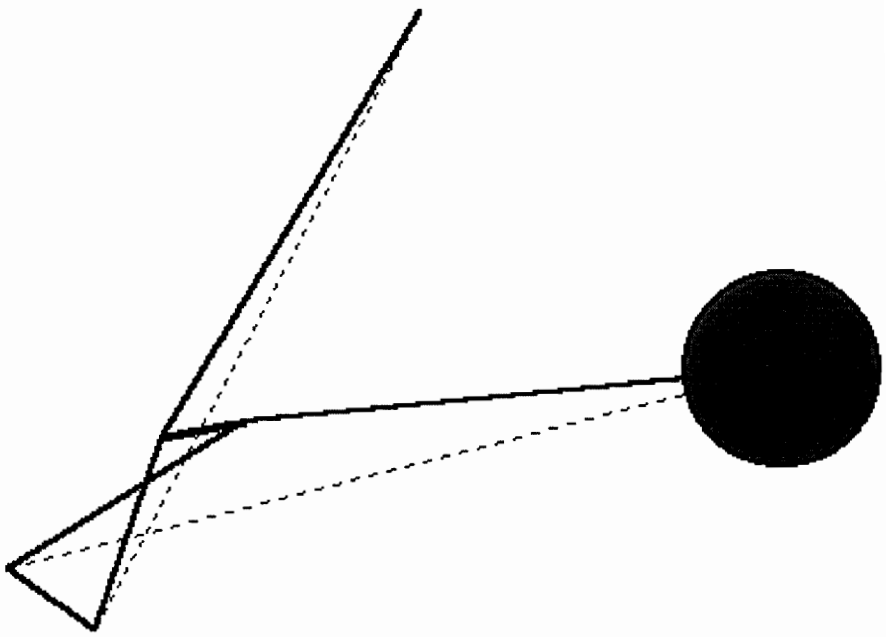


Figure 4-5, Chosen Solution Linkage

5. Conclusions and Recommendations

5.1 Summary

This thesis has presented a completely developed technique for the optimal synthesis of planar four-bar linkages which satisfy specific force or torque generating requirements. The work presented in this thesis differs from most previous work in mechanism force synthesis in that it targets applications which require prescribed input forces without corresponding output positions. In addition, the use of optimization theory and techniques rather than closed-form methods emphasizes the desire to solve the synthesis problem with maximum flexibility. Closed-form techniques can only be used to solve a limited class of problems and often require substantial intuition on the part of the designer in order to obtain practical solutions to a problem. The optimization technique presented allows a increased number of constraints to be applied to the synthesis problem. An objective function based primarily on the force-generation properties (or resistance curve) of the linkage and secondarily on these constraints is minimized using a direct-search constrained nonlinear optimization algorithm.

The design methodology developed in this thesis has been implemented into a comprehensive linkage synthesis and analysis computer program. This program enables the designer to specify the requirements and constraints for a particular synthesis problem and uses optimization to identify candidate solutions. An analysis processor is included

in the program for on-screen visualization and sensitivity analysis of the linkages.

5.2 Ideas for Further Research

During the course of this research, some areas of further research related to the current work were identified:

1. The optimization techniques presented could be modified to include other weight loading cases. For example, the load weight could be moved from the output link (weight arm) to the coupler link. One case currently under consideration involves attaching a cable to the end of the weight arm which then wraps a pulley and attaches at the other end to a vertically guided linear weight stack. In this case, the weight is not attached to linkage and is constrained to move only in the vertical direction.
2. Better methods for collecting static strength data would be beneficial. Several problems with determining physiologic strength curves for various exercises were noted in the data collection process. In addition, a method for collecting dynamic measurements of strength would also be helpful in attaining accurate strength data. Dynamic measurement of strength is a more accurate means of strength data collection because it captures the way in which real exercise is performed.
3. The linkage synthesis problem addressed in this thesis assumed massless links and negligible dynamic effects. Extending the current work to include a complete dynamic synthesis and analysis, while a substantial undertaking, is possible using the optimal techniques. Such an implementation will yield more accurate results for

situations where the links masses are large and dynamic effects are significant.

4. A synthesis technique for spatial linkages is a good candidate for further exploration. Either static or dynamic models could be solved using the optimal techniques presented in this research.

Appendix A, Optimization Software - ForceSynth

This appendix contains the source code for the Microsoft-Windows based ForceSynth Optimization Software discussed in Section 3.3. The program was developed in Microsoft Visual Basic Professional Edition® Version 3.0 with an assortment of third-party custom controls from various vendors included for specific features such as live graphing and tab controls. A function index and code module file reference have been included to simplify locating portions of the code.

ForceSynth Software

Nautilus Int'l, Inc.

Produced by Michael T. Scardina

Code Listing From
12/28/96

Table Of Contents

MDI.FRM - imgCopyButton_Click

```
Private Sub imgCopyButton_Click()
   imgCopyButton.Refresh
   EditCopyProc
End Sub
```

```
Private Sub imgCopyButton_MouseDown(button As Integer, Shift As Integer, X As
 -->Single, Y As Single)
   imgCopyButton.Picture = imgCopyButtonDn.Picture
End Sub
```

```
Private Sub imgCopyButton_MouseMove(button As Integer, Shift As Integer, X As
 -->Single, Y As Single)
   ' If the button is pressed, display the up bitmap if the
   ' mouse is dragged outside the button's area, otherwise
   ' display the up bitmap
   Select Case button
   Case 1
      If X <= 0 Or X > imgCopyButton.Width Or Y < 0 Or Y > imgCopyButton.Height
       -->Then
         imgCopyButton.Picture = imgCopyButtonUp.Picture
      Else
         imgCopyButton.Picture = imgCopyButtonDn.Picture
      End If
   End Select
End Sub
```

```
Private Sub imgCopyButton_MouseUp(button As Integer, Shift As Integer, X As
 -->Single, Y As Single)
   imgCopyButton.Picture = imgCopyButtonUp.Picture
End Sub
```

MDI.FRM - imgCopyButton_Click

```
Private Sub imgCutButton_Click()
   imgCutButton.Refresh
   EditCutProc
End Sub
```

```
Private Sub imgCutButton_MouseDown(button As Integer, Shift As Integer, X As
 -->Single, Y As Single)
   imgCutButton.Picture = imgCutButtonDn.Picture
End Sub
```

```
Private Sub imgCutButton_MouseMove(button As Integer, Shift As Integer, X As
 -->Single, Y As Single)
   ' If the button is pressed, display the up bitmap if the
   ' mouse is dragged outside the button's area, otherwise
   ' display the up bitmap
   Select Case button
   Case 1
      If X <= 0 Or X > imgCutButton.Width Or Y < 0 Or Y > imgCutButton.Height
       -->Then
         imgCutButton.Picture = imgCutButtonUp.Picture
      Else
         imgCutButton.Picture = imgCutButtonDn.Picture
      End If
   End Select
End Sub
```

```
Private Sub imgCutButton_MouseUp(button As Integer, Shift As Integer, X As
 -->Single, Y As Single)
   imgCutButton.Picture = imgCutButtonUp.Picture
End Sub
```

MDI.FRM - imgFileNewButton_Click

```
Private Sub imgFileNewButton_Click()
   imgFileNewButton.Refresh
   FileNew
End Sub
```

```
Private Sub imgFileNewButton_MouseDown(button As Integer, Shift As Integer, X
 -->As Single, Y As Single)
   imgFileNewButton.Picture = imgFileNewButtonDn.Picture
End Sub
```

```
Private Sub imgFileNewButton_MouseMove(button As Integer, Shift As Integer, X
 -->As Single, Y As Single)
   ' If the button is pressed, display the up bitmap if the
   ' mouse is dragged outside the button's area, otherwise
   ' display the up bitmap
   Select Case button
   Case 1
      If X <= 0 Or X > imgFileNewButton.Width Or Y < 0 Or Y > imgFileNewButton.
       -->Height Then
         imgFileNewButton.Picture = imgFileNewButtonUp.Picture
      Else
         imgFileNewButton.Picture = imgFileNewButtonDn.Picture
      End If
   End Select
End Sub
```

```
Private Sub imgFileNewButton_MouseUp(button As Integer, Shift As Integer, X As
 -->Single, Y As Single)
   imgFileNewButton.Picture = imgFileNewButtonUp.Picture
End Sub
```

MDI.FRM - imgFileNewButton_Click

```
Private Sub imgFileOpenButton_Click()
   imgFileOpenButton.Refresh
   FOpenProc
End Sub
```

```
Private Sub imgFileOpenButton_MouseDown(button As Integer, Shift As Integer, X
 -->As Single, Y As Single)
   imgFileOpenButton.Picture = imgFileOpenButtonDn.Picture
End Sub
```

```
Private Sub imgFileOpenButton_MouseMove(button As Integer, Shift As Integer, X
 -->As Single, Y As Single)
   ' If the button is pressed, display the up bitmap if the
   ' mouse is dragged outside the button's area, otherwise
   ' display the up bitmap
   Select Case button
   Case 1
      If X <= 0 Or X > imgFileOpenButton.Width Or Y < 0 Or Y > imgFileOpenButton.
       -->Height Then
         imgFileOpenButton.Picture = imgFileOpenButtonUp.Picture
      Else
         imgFileOpenButton.Picture = imgFileOpenButtonDn.Picture
      End If
   End Select
End Sub
```

```
Private Sub imgFileOpenButton_MouseUp(button As Integer, Shift As Integer, X As
 --> Single, Y As Single)
   imgFileOpenButton.Picture = imgFileOpenButtonUp.Picture
End Sub
```

MDI.FRM - imgPasteButton_Click

```
Private Sub imgPasteButton_Click()
   imgPasteButton.Refresh
   EditPasteProc
End Sub
```

```
Private Sub imgPasteButton_MouseDown(button As Integer, Shift As Integer, X As
 --> Single, Y As Single)
   imgPasteButton.Picture = imgPasteButtonDn.Picture
End Sub
```

```
Private Sub imgPasteButton_MouseMove(button As Integer, Shift As Integer, X As
 -->Single, Y As Single)
   ' If the button is pressed, display the up bitmap if the
   ' mouse is dragged outside the button's area, otherwise
   ' display the up bitmap
   Select Case button
   Case 1
      If X <= 0 Or X > imgPasteButton.Width Or Y < 0 Or Y > imgPasteButton.Height
       -->Then
         imgPasteButton.Picture = imgPasteButtonUp.Picture
      Else
         imgPasteButton.Picture = imgPasteButtonDn.Picture
      End If
   End Select
End Sub
```

```
Private Sub imgPasteButton_MouseUp(button As Integer, Shift As Integer, X As
 -->Single, Y As Single)
   imgPasteButton.Picture = imgPasteButtonUp.Picture
End Sub
```


MDI.FRM - imgPasteButton_Click

```
Private Sub MDIForm_Load()
   ' Application starts here (Load event of Startup form)

   'Maximize MDI Form
   frmMDI.WindowState = 2
   frmMDI.Show

   frmMainOptimzation.Show
   'Always set working directory to directory containing the application.
   ChDir App.Path

   'Initialize document form arrays, and show first document.
   'I changed this to not show an initial document, must use File: New
   ReDim Document(0)
   ReDim FState(0)

   'this not needed now
   'document(1).Tag = 1
   'FState(1).Dirty = False
   'document(1).Show
   'document(1).Caption = "untitled"

   'Read MDINOTE.INI and set recent file menu items appropriately
   GetRecentFiles

   'defines the global string for a carriage return
   gstrNewLine = Chr$(13) + Chr$(10)

End Sub
```

MDI.FRM - MDIForm_QueryUnload

```
Private Sub MDIForm_QueryUnload(Cancel As Integer, UnloadMode As Integer)

    Unload frmSynthCurves
    Unload frmDataGrid
    Unload frmDiagram
    Unload frmLinkageDraw
    Unload frmAnalysisLinkageDraw
    Unload frmAnalysisTextOutput
    Unload frmAnalysisCurves
    Unload frmFind
    Unload frmMainOptimzation
    Unload frmMDI
    Unload frmTextViewEdit
    Unload frmSynthTextOutput
    End

End Sub
```

```
Private Sub MDIForm_Unload(Cancel As Integer)
    ' If the Unload was not canceled (in the QueryUnload events for the Notepad
     -->forms)
    ' there will be no document windows left, so go ahead and end the application.

    If Not AnyPadsLeft() Then
        End
    End If

End Sub
```

MDI.FRM - MDIForm_QueryUnload

```
Private Sub mnuAnalysisSwitch_Click()

    frmMDI.Caption = "Nautilus Force Optimization [ANALYSIS MODE]"
    If frmMainOptimzation.WindowState = 0 Then frmMainOptimzation.WindowState
     -->= 1
    If frmLinkageDraw.WindowState = 0 Then frmLinkageDraw.WindowState = 1
    If frmSynthTextOutput.WindowState = 0 Then frmSynthTextOutput.
     -->WindowState = 1
    If frmDataGrid.WindowState = 0 Then frmDataGrid.WindowState = 1
    If frmSynthCurves.WindowState = 0 Then frmSynthCurves.WindowState = 1

    frmAnalysisLinkageDraw.Show
    frmAnalysisCurves.Show
    frmAnalysisTextOutput.Show
    frmErrorTable.Show
    frmAnalysisLinkageDraw.SetFocus

End Sub
```

```
Private Sub mnuFileExit_Click()

    Unload frmSynthCurves
    Unload frmDataGrid
    Unload frmDiagram
    Unload frmLinkageDraw
    Unload frmAnalysisLinkageDraw
    Unload frmAnalysisTextOutput
    Unload frmAnalysisCurves
    Unload frmFind
    Unload frmMainOptimzation
    Unload frmMDI
    Unload frmTextViewEdit
    Unload frmSynthTextOutput
    End
End Sub
```

```
Private Sub mnuFileLoadData_Click()

   frmMDI.mnuOptimizationStart.Enabled = True
   LOAD_STRENGTH_DATA
   frmMainOptimzation.SetFocus

End Sub
```

```
Private Sub mnuFileNewText_Click()
   FileNew
End Sub
```

```
Private Sub mnuFileOpenText_Click()
   ChDir App.Path
   FOpenProc
End Sub
```

```
Private Sub mnuLaunchClipboard_Click()

   Dim intReturnCode As Integer

   intReturnCode = Shell("c:\windows\clipbrd.exe", 1)

End Sub
```

```
Private Sub mnuOptimizationConstraints_Click()

   'constraint_form.Show MODAL
   frmConfig.Show MODAL

End Sub
```

MDI.FRM - mnuFileExit_Click

```
Private Sub mnuOptimizationDiagram_Click()

   frmDiagram.Show

End Sub
```

```
Private Sub mnuOptimizationNumLeaps_Click()

On Error GoTo errortrap_leaps

Dim this_message As String
Dim this_title As String
Dim this_default As String

this_message = "Enter a new value for the number of random leaps to make:"
this_title = "Input Integer Value..."
this_default = Str(gintNumOptRuns)

popup_leaps:
temp = InputBox(this_message, this_title, this_default)

If temp = "" Then Exit Sub

temp = CInt(temp)

gintNumOptRuns = temp

'Debug.Print temp

Exit Sub

errortrap_leaps:

status = MsgBox("Lets try that again with a number this time (or just hit cancel!)", 1
```

```
 --> + 48 + 0 + 0, "Input Error!")

If status = 2 Then Exit Sub
If status = 1 Then GoTo popup_leaps

End Sub

---

Private Sub mnuOptimizationSetError_Click()

On Error GoTo errortrap_tolerance

Dim this_message As String
Dim this_title As String
Dim this_default As String

this_message = "Enter a new value for the maximum allowable error for a linkage:"
this_title = "Input Decimel Value..."
this_default = Str(gdblErrorLimit)

popup_tolerance:
temp = InputBox(this_message, this_title, this_default)

If temp = "" Then Exit Sub

temp = CDbl(temp)

gdblErrorLimit = temp

'Debug.Print temp

Exit Sub

errortrap_tolerance:

status = MsgBox("Lets try that again with a number this time (or just hit cancel!)", 1
 --> + 48 + 0 + 0, "Input Error!")
```

MDI.FRM - mnuOptimizationNumLeaps_Click

```
If status = 2 Then Exit Sub
If status = 1 Then GoTo popup_tolerance

End Sub

---

Private Sub mnuOptimizationStart_Click()

  mnuOptimizationStart.Enabled = False
  mnuOptimizationStop.Enabled = True

  'disable and grey-out the spinbuttons while optimizing
  Call DisableControl(frmMainOptimzation.spnLenInputArm, LIGHTGRAY)
  Call DisableControl(frmMainOptimzation.spnLenWeightArm, LIGHTGRAY)
  Call DisableControl(frmMainOptimzation.spnInputArmOffset, LIGHTGRAY)
  Call DisableControl(frmMainOptimzation.spnWeightArmOffset, LIGHTGRAY)
  Call DisableControl(frmMainOptimzation.spnBaseRotation, LIGHTGRAY)
  Call DisableControl(frmMainOptimzation.spnLengthLink1, LIGHTGRAY)
  Call DisableControl(frmMainOptimzation.spnLengthLink2, LIGHTGRAY)
  Call DisableControl(frmMainOptimzation.spnLengthLink3, LIGHTGRAY)
  Call DisableControl(frmMainOptimzation.spnLengthLink4, LIGHTGRAY)

  'disable and grey-out the checkboxes while optimizing
  Call DisableControl(frmMainOptimzation.chkLenWeightArm, Null)
  Call DisableControl(frmMainOptimzation.chkInputArmOffset, Null)
  Call DisableControl(frmMainOptimzation.chkWeightArmOffset, Null)
  Call DisableControl(frmMainOptimzation.chkBaseRotation, Null)
  Call DisableControl(frmMainOptimzation.chkLengthLink1, Null)
  Call DisableControl(frmMainOptimzation.chkLengthLink2, Null)
  Call DisableControl(frmMainOptimzation.chkLengthLink3, Null)
  Call DisableControl(frmMainOptimzation.chkLengthLink4, Null)

  'disable and grey-out the toggle-buttons while optimizing
  'note that these buttons have a disabled-color property
  'already set up so a manual color change is not needed
  Call DisableControl(frmMainOptimzation.cmdMechClosure, Null)
```

MDI.FRM - mnuOptimizationStart_Click

```
Call DisableControl(frmMainOptimzation.cmdMechType, Null)

'disable and grey-out the display boxes while optimizing
Call DisableControl(frmMainOptimzation.pnlLenInputArm, MEDIUMGRAY)
Call DisableControl(frmMainOptimzation.pnlLenWeightArm, MEDIUMGRAY)
Call DisableControl(frmMainOptimzation.pnlInputArmOffset, MEDIUMGRAY)
Call DisableControl(frmMainOptimzation.pnlWeightArmOffset, MEDIUMGRAY)
Call DisableControl(frmMainOptimzation.pnlBaseRotation, MEDIUMGRAY)
Call DisableControl(frmMainOptimzation.pnlLengthLink1, MEDIUMGRAY)
Call DisableControl(frmMainOptimzation.pnlLengthLink2, MEDIUMGRAY)
Call DisableControl(frmMainOptimzation.pnlLengthLink3, MEDIUMGRAY)
Call DisableControl(frmMainOptimzation.pnlLengthLink4, MEDIUMGRAY)

'================================================================
 -->=======

  'open the cumulative solutions file
  Open "C:\solution.txt" For Output As #2
  Print #2, "Optimization started at "; Time
  Print #2,
  'close file now, will reopen for append
  Close #2

  If (gconUseDebug = 1) Then Open "C:\debuger.txt" For Output As #1
  If (gconUseDebug = 1) Then Print #1,
  If (gconUseDebug = 1) Then Print #1, "Optimization started at "; Time
  If (gconUseDebug = 1) Then Print #1,

  'get the values currently in the label boxes
  gdblBaseRot = CDbl(frmMainOptimzation.pnlBaseRotation.Caption) *
   -->gconDegToRad
  gdblInputOff = CDbl(frmMainOptimzation.pnlInputArmOffset.Caption) *
   -->gconDegToRad
  gdblWhtOff = CDbl(frmMainOptimzation.pnlWeightArmOffset.Caption) *
   -->gconDegToRad
  gdblLen1 = CDbl(frmMainOptimzation.pnlLengthLink1.Caption)
  gdblLen2 = CDbl(frmMainOptimzation.pnlLengthLink2.Caption)
```

MDI.FRM - mnuOptimizationStart_Click

```
gdblLen3 = CDbl(frmMainOptimzation.pnlLengthLink3.Caption)
gdblLen4 = CDbl(frmMainOptimzation.pnlLengthLink4.Caption)
gdblWhtLen = CDbl(frmMainOptimzation.pnlLenWeightArm.Caption)

'initialize the different variables, including starting values for the design
 -->variables
gdblInitDesVar(1) = gdblBaseRot
gdblInitDesVar(2) = gdblInputOff
gdblInitDesVar(3) = gdblWhtOff
gdblInitDesVar(4) = gdblLen1
gdblInitDesVar(5) = gdblLen2
gdblInitDesVar(6) = gdblLen3
gdblInitDesVar(7) = gdblLen4
gdblInitDesVar(8) = gdblWhtLen

'these are needed so that we can assess penalties and get the range of
'motion of the initial dataset
gdblLastDesVar(1) = gdblBaseRot
gdblLastDesVar(2) = gdblInputOff
gdblLastDesVar(3) = gdblWhtOff
gdblLastDesVar(4) = gdblLen1
gdblLastDesVar(5) = gdblLen2
gdblLastDesVar(6) = gdblLen3
gdblLastDesVar(7) = gdblLen4
gdblLastDesVar(8) = gdblWhtLen

'============================================================

'set the current drawable variables for plotting
gdblCurrDrawVar(1) = gdblBaseRot
gdblCurrDrawVar(2) = gdblInputOff
gdblCurrDrawVar(3) = gdblWhtOff
gdblCurrDrawVar(4) = gdblLen1
gdblCurrDrawVar(5) = gdblLen2
gdblCurrDrawVar(6) = gdblLen3
gdblCurrDrawVar(7) = gdblLen4
gdblCurrDrawVar(8) = gdblWhtLen
```

MDI.FRM - mnuOptimizationStart_Click

```
'enable zooming in the draw window now
frmLinkageDraw.mnuZoomIn.Enabled = True
frmLinkageDraw.mnuZoomOut.Enabled = True

'draw the initial linkage in the draw window as a starting reference
Call DRAW_SYNTH_LINKAGE(gdblCurrDrawVar(4), gdblCurrDrawVar(5),
 -->gdblCurrDrawVar(6), gdblCurrDrawVar(7), gdblCurrDrawVar(1),
 -->gintLinkType, gintClosure, gdblInputLen, gdblCurrDrawVar(8),
 -->gdblCurrDrawVar(2), gdblCurrDrawVar(3))
'reset the plot data
frmSynthCurves.gphComboPlot.DataReset = 1
frmSynthCurves.gphComboPlot.DataReset = 8
frmSynthCurves.gphComboPlot.ThisPoint = 1
frmSynthCurves.gphComboPlot.NumPoints = 2
frmSynthCurves.gphComboPlot.OverlayGraph = 1

'this loop plots the loaded discrete strength data and the strength curve of the
 -->initial values
For index = 1 To gintNumDataPoints Step 1
   gdblTheta2 = gdblCurveBeta(index, 1) * gconDegToRad - gdblInitDesVar(2)
   Call COMPUTE_STRENGTH(gdblStrength, gdblCouplerForce, gintLinkType,
    -->gintClosure, gdblTheta2, gdblInitDesVar(1), gdblInitDesVar(2),
    -->gdblInitDesVar(3), gdblInitDesVar(4), gdblInitDesVar(5), gdblInitDesVar(6
    -->), gdblInitDesVar(7), gdblInitDesVar(8), gd
   frmSynthCurves.gphComboPlot.XPosData = gdblCurveBeta(index, 1)
   frmSynthCurves.gphComboPlot.GraphData = gdblCurveStrength(index, 1)
   frmSynthCurves.gphComboPlot.OverlayGraphData = gdblStrength * 100
   frmSynthCurves.gphComboPlot.ThisPoint = frmSynthCurves.gphComboPlot.
    -->ThisPoint + 1
   frmSynthCurves.gphComboPlot.NumPoints = frmSynthCurves.gphComboPlot.
    -->ThisPoint + 1

   squared_error = Abs(gdblStrength - gdblCurveStrength(index, 1) / 100) ^ 2
   sum_squared_error = sum_squared_error + squared_error
   Debug.Print squared_error, sum_squared_error
Next index
```

MDI.FRM - mnuOptimizationStart_Click

```
ASSESS_PENALTIES
sum_squared_error = sum_squared_error + gdblPenalty
Debug.Print "The error for the initial dataset specified (including applicable
 -->penalties) is "; sum_squared_error
MsgBox "The error for the initial dataset specified (including applicable penalties
 -->) is " + sum_squared_error
If (gconUseDebug = 1) Then Print #1, "Test_error for set = ";
 -->sum_squared_error

'complete the plot and show it
frmSynthCurves.gphComboPlot.GraphTitle = "Initial Seed Force Profile"
frmSynthCurves.gphComboPlot.NumPoints = frmSynthCurves.gphComboPlot.
 -->ThisPoint - 1
frmSynthCurves.gphComboPlot.DrawMode = 2

'============================================================
 -->============

'this loop is for the number of random jumps to make in searching
'for solutions.
For j = 1 To gintNumOptRuns Step 1

   'flag which enables window events
   DoEvents

   'initialize the step sizes and exploratory success/failure variables
   'also reset the on/off flags of design variables to all on
   For i = 1 To gintNumDesVar Step 1
      gintDesVarConstant(i) = 1
      gintExplrSuccess(i) = 0
      gdblMinStepSize(i) = 0.001
      gdblStepSize(i) = 0.01
   Next i

   'get the flags for the design variables from the check boxes,
   'to see which will be constants, and which will be optimized
   gintDesVarConstant(1) = Abs(frmMainOptimzation.chkBaseRotation.Value)
```

MDI.FRM - mnuOptimizationStart_Click

```
gintDesVarConstant(2) = Abs(frmMainOptimzation.chkInputArmOffset.Value)
gintDesVarConstant(3) = Abs(frmMainOptimzation.chkWeightArmOffset.
 -->Value)
gintDesVarConstant(4) = Abs(frmMainOptimzation.chkLengthLink1.Value)
gintDesVarConstant(5) = Abs(frmMainOptimzation.chkLengthLink2.Value)
gintDesVarConstant(6) = Abs(frmMainOptimzation.chkLengthLink3.Value)
gintDesVarConstant(7) = Abs(frmMainOptimzation.chkLengthLink4.Value)
gintDesVarConstant(8) = Abs(frmMainOptimzation.chkLenWeightArm.Value)

'call to transfer control to the optimization procedure
'don't need to pass arguments because the variables are already global and
 -->can be seen everywhere
EXECUTE_OPTIMIZATION

'now get the output range of motion for the converged solution.
'this may not be necessary since the assess penalties routine
'already computed the angles beforehand.
Call GET_OUTPUT_RANGE(gdblPhiStart, gdblPhiEnd, gdblPhiRange)

'plot the strength curve solution minimum obtained, or otherwise
frmSynthCurves.gphComboPlot.DataReset = 1
frmSynthCurves.gphComboPlot.DataReset = 8
frmSynthCurves.gphComboPlot.ThisPoint = 1
frmSynthCurves.gphComboPlot.NumPoints = 2
frmSynthCurves.gphComboPlot.OverlayGraph = 1

For index = 1 To gintNumDataPoints Step 1
   gdblTheta2 = gdblCurveBeta(index, 1) * gconDegToRad - gdblLastDesVar(
    -->2)
   Call COMPUTE_STRENGTH(gdblStrength, gdblCouplerForce, gintLinkType,
    -->gintClosure, gdblTheta2, gdblLastDesVar(1), gdblLastDesVar(2),
    -->gdblLastDesVar(3), gdblLastDesVar(4), gdblLastDesVar(5),
    -->gdblLastDesVar(6), gdblLastDesVar(7), gdblLastDesVar(8)
   frmSynthCurves.gphComboPlot.XPosData = gdblCurveBeta(index, 1)
   frmSynthCurves.gphComboPlot.GraphData = gdblCurveStrength(index, 1)
   frmSynthCurves.gphComboPlot.OverlayGraphData = gdblStrength * 100
   frmSynthCurves.gphComboPlot.ThisPoint = frmSynthCurves.gphComboPlot.
```

MDI.FRM - mnuOptimizationStart_Click

```
    -->ThisPoint + 1
   frmSynthCurves.gphComboPlot.NumPoints = frmSynthCurves.
    -->gphComboPlot.ThisPoint + 1
Next index

frmSynthCurves.gphComboPlot.GraphTitle = "Solution #" + j + " Strength
 -->Curve"
frmSynthCurves.gphComboPlot.NumPoints = frmSynthCurves.gphComboPlot.
 -->ThisPoint - 1
frmSynthCurves.gphComboPlot.DrawMode = 2

'set the current drawable variables for plotting in the draw window
gdblCurrDrawVar(1) = gdblLastDesVar(1)
gdblCurrDrawVar(2) = gdblLastDesVar(2)
gdblCurrDrawVar(3) = gdblLastDesVar(3)
gdblCurrDrawVar(4) = gdblLastDesVar(4)
gdblCurrDrawVar(5) = gdblLastDesVar(5)
gdblCurrDrawVar(6) = gdblLastDesVar(6)
gdblCurrDrawVar(7) = gdblLastDesVar(7)
gdblCurrDrawVar(8) = gdblLastDesVar(8)

'draw the solution linkage in the draw window
Call DRAW_SYNTH_LINKAGE(gdblCurrDrawVar(4), gdblCurrDrawVar(5),
 -->gdblCurrDrawVar(6), gdblCurrDrawVar(7), gdblCurrDrawVar(1),
 -->gintLinkType, gintClosure, gdblInputLen, gdblCurrDrawVar(8),
 -->gdblCurrDrawVar(2), gdblCurrDrawVar(3))

'print solution parameters to a cumulative solutions text file if error is
 -->acceptable
If (gdblLastSumSqrError <= gdblErrorLimit) Then
   Open "C:\solution.txt" For Append As #2
   Print #2, j
   If gintLinkType = 1 Then
      Print #2, gdblCurveBeta(1, 1)
      Print #2, gdblCurveBeta(gintNumDataPoints, 1)
   ElseIf gintLinkType = -1 Then
      Print #2, gdblCurveBeta(gintNumDataPoints, 1)
```

MDI.FRM - mnuOptimizationStart_Click

```
        Print #2, gdblCurveBeta(1, 1)
    End If
    Print #2, gintLinkType
    Print #2, gintClosure
    Print #2, gdblInputLen
    Print #2, gdblLastDesVar(8)
    Print #2, gdblLastDesVar(2) / gconDegToRad
    Print #2, gdblLastDesVar(3) / gconDegToRad
    Print #2, gdblLastDesVar(1) / gconDegToRad
    Print #2, gdblLastDesVar(4)
    Print #2, gdblLastDesVar(5)
    Print #2, gdblLastDesVar(6)
    Print #2, gdblLastDesVar(7)
    Print #2, gdblPhiStart
    Print #2, gdblPhiEnd
    Print #2, gdblPhiRange
    Print #2, gdblLastSumSqrError
    Print #2,
    Close #2
End If

'take a random jump in space now
'get some new seed values to rerum the hooke and jeeves
'only randomize variables flagged as actual design variables
'Note that the angular variables can be either pos. or neg.
If gintDesVarConstant(1) = 1 Then gdblBaseRot = (Int((gdblRndBaseRotHi -
 -->gdblRndBaseRotLow + 1) * Rnd + gdblRndBaseRotLow)) *
 -->gconDegToRad
If gintDesVarConstant(2) = 1 Then gdblInputOff = (Int((gdblRndInputOffHi -
 -->gdblRndInputOffLow + 1) * Rnd + gdblRndInputOffLow)) * gconDegToRad
If gintDesVarConstant(3) = 1 Then gdblWhtOff = (Int((gdblRndWhtOffHi -
 -->gdblRndWhtOffLow + 1) * Rnd + gdblRndWhtOffLow)) * gconDegToRad
If gintDesVarConstant(4) = 1 Then gdblLen1 = Int((gdblRndLen1Hi -
 -->gdblRndLen1Low + 1) * Rnd + gdblRndLen1Low)
If gintDesVarConstant(5) = 1 Then gdblLen2 = Int((gdblRndLen2Hi -
 -->gdblRndLen2Low + 1) * Rnd + gdblRndLen2Low)
If gintDesVarConstant(6) = 1 Then gdblLen3 = Int((gdblRndLen3Hi -
```

MDI.FRM - mnuOptimizationStart_Click

```
 -->gdblRndLen3Low + 1) * Rnd + gdblRndLen3Low)
If gintDesVarConstant(7) = 1 Then gdblLen4 = Int((gdblRndLen4Hi -
 -->gdblRndLen4Low + 1) * Rnd + gdblRndLen4Low)
If gintDesVarConstant(8) = 1 Then gdblWhtLen = Int((gdblRndWhtLenHi -
 -->gdblRndWhtLenLow + 1) * Rnd + gdblRndWhtLenLow)

'put the text in the output text box now
'open the temp file
Open "C:\TEMP.TXT" For Output As #50
    'print all info to temp file
    Print #50, "Values for solution number: "; j
    If gintLinkType = 1 Then
        Print #50, "Input Arm Start Angle = "; Format(gdblCurveBeta(1, 1), "###0.
         -->0##")
        Print #50, "Input Arm End Angle = "; Format(gdblCurveBeta(
         -->gintNumDataPoints, 1), "###0.0##")
        Print #50, "Linkage Type = Co-rotating"
    ElseIf gintLinkType = -1 Then
        Print #50, "Input Arm Start Angle = "; Format(gdblCurveBeta(
         -->gintNumDataPoints, 1), "###0.0##")
        Print #50, "Input Arm End Angle = "; Format(gdblCurveBeta(1, 1), "###0.0
         -->##")
        Print #50, "Linkage Type = Counter-rotating"
    End If
    Print #50, "Linkage Closure = "; Str(gintClosure)
    Print #50, "Input Arm Length = "; Format(gdblInputLen, "###0.0##")
    Print #50, "Weight Arm Length = "; Format(gdblLastDesVar(8), "###0.0##")
    Print #50, "Input Arm Offset = "; Format(gdblLastDesVar(2) /
     -->gconDegToRad, "###0.0##")
    Print #50, "Weight Arm Offset = "; Format(gdblLastDesVar(3) /
     -->gconDegToRad, "###0.0##")
    Print #50, "Linkage Base Rotation = "; Format(gdblLastDesVar(1) /
     -->gconDegToRad, "###0.0##")
    Print #50, "Link 1 length = "; Format(gdblLastDesVar(4), "###0.0##")
    Print #50, "Link 2 length = "; Format(gdblLastDesVar(5), "###0.0##")
    Print #50, "Link 3 length = "; Format(gdblLastDesVar(6), "###0.0##")
    Print #50, "Link 4 length = "; Format(gdblLastDesVar(7), "###0.0##")
```

MDI.FRM - mnuOptimizationStart_Click

```
    Print #50, "Weight Arm Start Angle = "; Format(gdblPhiStart, "###0.0##")
    Print #50, "Weight Arm End Angle = "; Format(gdblPhiEnd, "###0.0##")
    Print #50, "Output Range of Motion = "; Format(gdblPhiRange, "###0.0##")
    Print #50, "The total error is = "; gdblLastSumSqrError
    Print #50,
    Print #50, "Random seed for rotation is "; Format(gdblBaseRot /
     -->gconDegToRad, "###0.0##")
    Print #50, "Random seed for input arm offset is "; Format(gdblInputOff /
     -->gconDegToRad, "###0.0##")
    Print #50, "Random seed for weight arm offset is "; Format(gdblWhtOff /
     -->gconDegToRad, "###0.0##")
    Print #50, "Random seed for gdblLen1 is "; Format(gdblLen1, "###0.0##")
    Print #50, "Random seed for gdblLen2 is "; Format(gdblLen2, "###0.0##")
    Print #50, "Random seed for gdblLen3 is "; Format(gdblLen3, "###0.0##")
    Print #50, "Random seed for gdblLen4 is "; Format(gdblLen4, "###0.0##")
    Print #50, "Random seed for weight arm length is "; Format(gdblWhtLen,
     -->"###0.0##")
'close the temp file
Close #50

'reopen the temp file for input
Open "C:\TEMP.TXT" For Input As #50
'suck contents of file into text box
frmSynthTextOutput.txtSynthTextOutput.Text = Input$(LOF(50), 50)
'close the file
Close #50
'delete the temp file from disk
Kill "C:\TEMP.TXT"

'reset the variables for the next pass through
gdblInitDesVar(1) = gdblBaseRot
gdblInitDesVar(2) = gdblInputOff
gdblInitDesVar(3) = gdblWhtOff
gdblInitDesVar(4) = gdblLen1
gdblInitDesVar(5) = gdblLen2
gdblInitDesVar(6) = gdblLen3
gdblInitDesVar(7) = gdblLen4
```

MDI.FRM - mnuOptimizationStart_Click

```
    gdblInitDesVar(8) = gdblWhtLen

    Debug.Print "random gdblBaseRot is ", gdblBaseRot / gconDegToRad
    Debug.Print "random input arm offset is ", gdblInputOff / gconDegToRad
    Debug.Print "random weight arm offset is ", gdblWhtOff / gconDegToRad
    Debug.Print "random gdblLen1 is ", gdblLen1
    Debug.Print "random gdblLen2 is ", gdblLen2
    Debug.Print "random gdblLen3 is ", gdblLen3
    Debug.Print "random gdblLen4 is ", gdblLen4
    Debug.Print "random weight arm length is ", gdblWhtLen

    If (gintAnimStop = True) Then Exit For

Next j   'next pass through converged iteration

'mark the end time
If (gconUseDebug = 1) Then Print #1, "Optimization ended at "; Time
'close the debug text file
If (gconUseDebug = 1) Then Close #1

'mark the end time
Open "C:\solution.txt" For Append As #2
Print #2, "Optimization ended at "; Time
'close the solutions text file
Close #2

'reset the menus states
mnuOptimizationStart.Enabled = True
mnuOptimizationStop.Enabled = False

'Enable the spinbuttons now that the optimization is finished
Call EnableControl(frmMainOptimzation.spnLenInputArm)
Call EnableControl(frmMainOptimzation.spnLenWeightArm)
Call EnableControl(frmMainOptimzation.spnInputArmOffset)
Call EnableControl(frmMainOptimzation.spnWeightArmOffset)
Call EnableControl(frmMainOptimzation.spnBaseRotation)
Call EnableControl(frmMainOptimzation.spnLengthLink1)
```

MDI.FRM - mnuOptimizationStart_Click

```
    Call EnableControl(frmMainOptimzation.spnLengthLink2)
    Call EnableControl(frmMainOptimzation.spnLengthLink3)
    Call EnableControl(frmMainOptimzation.spnLengthLink4)

    'Enable the checkboxes
    Call EnableControl(frmMainOptimzation.chkLenWeightArm)
    Call EnableControl(frmMainOptimzation.chkInputArmOffset)
    Call EnableControl(frmMainOptimzation.chkWeightArmOffset)
    Call EnableControl(frmMainOptimzation.chkBaseRotation)
    Call EnableControl(frmMainOptimzation.chkLengthLink1)
    Call EnableControl(frmMainOptimzation.chkLengthLink2)
    Call EnableControl(frmMainOptimzation.chkLengthLink3)
    Call EnableControl(frmMainOptimzation.chkLengthLink4)

    'disable and grey-out the toggle-buttons while optimizing
    'note that these buttons have a disabled-color property
    'already set up so a manual color change is not needed
    Call EnableControl(frmMainOptimzation.cmdMechClosure)
    Call EnableControl(frmMainOptimzation.cmdMechType)

    'disable and grey-out the display boxes while optimizing
    Call EnableControl(frmMainOptimzation.pnlLenInputArm)
    Call EnableControl(frmMainOptimzation.pnlLenWeightArm)
    Call EnableControl(frmMainOptimzation.pnlInputArmOffset)
    Call EnableControl(frmMainOptimzation.pnlWeightArmOffset)
    Call EnableControl(frmMainOptimzation.pnlBaseRotation)
    Call EnableControl(frmMainOptimzation.pnlLengthLink1)
    Call EnableControl(frmMainOptimzation.pnlLengthLink2)
    Call EnableControl(frmMainOptimzation.pnlLengthLink3)
    Call EnableControl(frmMainOptimzation.pnlLengthLink4)

    'reset the halt flag
    gintAnimStop = False

End Sub
```

MDI.FRM - mnuOptimizationStart_Click

```
Private Sub mnuOptimizationStop_Click()

    MsgBox "Halt Optimization Procedure?  You will have to wait until the next
     -->iteration converges after clicking OK ..."
    gintAnimStop = True
    mnuOptimizationStop.Enabled = False

End Sub
```

```
Private Sub mnuRecentFile_Click(index As Integer)
  OpenFile (mnuRecentFile(index).Caption)
  ' Update recent files list.
  GetRecentFiles
End Sub
```

```
Private Sub mnuWindowArrange_Click()

    frmMDI.Arrange ARRANGE_ICONS

End Sub
```

```
Private Sub mnuWindowCascade_Click()

    frmMDI.Arrange CASCADE

End Sub
```

MDI.FRM - mnuWindowTile_Click

```
Private Sub mnuWindowTile_Click()

   frmMDI.Arrange TILE_HORIZONTAL

End Sub
```

MDI.FRM - mnuWindowTile_Click

```
Sub ASSESS_PENALTIES()

'This routine is called to apply any activated constraints on the problem
'and to assess any applicable penalties as necessary.  The penalty counter is
'set to zero at the beginning of the routine, and then the range of motion of
'the mechanism is calculated using a call to GET_OUTPUT_RANGE.
'Because the angles computed by GET_OUTPUT_RANGE may not be in the
'correct quadrants, All angles are clamped to be either between 0 and 180
'degrees or 0 and -180 degrees within that routine.  This makes calculation
'of the total output range of motion more reasonalble and generally correct.


   'reset all penalties to zero
   gdblPenalty = 0
   'get the output range of motion for the current linkage
   Call GET_OUTPUT_RANGE(gdblPhiStart, gdblPhiEnd, gdblPhiRange)
   Debug.Print "Output Angles Are: "; gdblPhiStart, gdblPhiEnd, gdblPhiRange

   'If the starting output angle penalty function is activated, do this check and
    -->assess
   ' the appropriate penalty
   If (gintPhiStartConstrainOn = 1) Then
      Debug.Print "Starting Output Angle is Being Checked. A penalty may be
       -->assessed"
      If (gconUseDebug = 1) Then Print #1, "Starting Output Angle is Being Checked
       -->. A penalty may be assessed"

      If (gdblPhiStart < gdblPhiStartLow) Then
         'If the starting output angle is less than the minimum alloweed...
         Debug.Print "Starting Output Angle Less than minimum allowed:  "; Abs(
          -->gdblPhiStartLow - gdblPhiStart)
         'The penalty is based on how far out of the allowable range the value is.
         gdblPenalty = gdblPenalty + (Abs(gdblPhiStartLow - gdblPhiStart) *
          -->gdblPhiStartPenalty)
      ElseIf (gdblPhiStart > gdblPhiStartHi) Then
         'If the starting output angle is more than the maximum alloweed...
```

```
            Debug.Print "Starting Output Angle More than maximum allowed:  "; Abs(
             -->gdblPhiStart - gdblPhiStartHi)
            'The penalty is based on how far out of the allowable range the value is.
            gdblPenalty = gdblPenalty + (Abs(gdblPhiStart - gdblPhiStartHi) *
             -->gdblPhiStartPenalty)
        End If
    End If

    If (gintPhiEndConstrainOn = 1) Then
        Debug.Print "Ending Output Angle is Being Checked. A penalty may be
         -->assessed"
        If (gconUseDebug = 1) Then Print #1, "Ending Output Angle is Being Checked
         -->. A penalty may be assessed"

        If (gdblPhiEnd < gdblPhiEndLow) Then
            'If the ending output angle is less than the minimum alloweed...
            Debug.Print "Ending Output Angle Less than minimum allowed:  "; Abs(
             -->gdblPhiEndLow - gdblPhiEnd)
            gdblPenalty = gdblPenalty + (Abs(gdblPhiEndLow - gdblPhiEnd) *
             -->gdblPhiEndPenalty)
            'The penalty is based on how far out of the allowable range the value is.
        ElseIf (gdblPhiEnd > gdblPhiEndHi) Then
            'If the ending output angle is more than the maximum alloweed...
            Debug.Print "Ending Output Angle More than maximum allowed:  "; Abs(
             -->gdblPhiEnd - gdblPhiEndHi)
            'The penalty is based on how far out of the allowable range the value is.
            gdblPenalty = gdblPenalty + (Abs(gdblPhiEnd - gdblPhiEndHi) *
             -->gdblPhiEndPenalty)
        End If
    End If

End Sub
```



```
Sub COMPUTE_STRENGTH(output_strength As Double, coupler_link_force As
 -->Double, I_type As Integer, closure As Integer, current_input As Double,
 -->linkage_rotation As Double, input_arm_offset As Double, weight_arm_offset
 -->As Double, length1 As Double, length2 A

'This routine computes the nondimensional resistance R and the coupler
'force of current linkage at a single input arm angle beta.  This routine
'is called at points over the input range of motion to determine the
'entire resistance curve.

    Dim temp1  As Double, temp2 As Double
    Dim rot_x As Double, rot_y As Double
    Dim E3 As Double, F3 As Double, G3 As Double
    Dim E4 As Double, F4 As Double, G4 As Double
    Dim N3 As Double, N4 As Double
    Dim m_sub_g As Double, n_sub_g As Double, d_sub_g As Double

    'Debug.Print output_strength
    'Debug.Print coupler_link_force
    'Debug.Print I_type
    'Debug.Print closure
    'Debug.Print current_input
    'Debug.Print linkage_rotation
    'Debug.Print input_arm_offset
    'Debug.Print weight_arm_offset
    'Debug.Print length1
    'Debug.Print length2
    'Debug.Print length3
    'Debug.Print length4
    'Debug.Print length_w
    'Debug.Print length_in

    return_error = "No Errors"
    'these are the rotation effects
    rot_x = length1 * Cos(linkage_rotation)
    rot_y = length1 * Sin(linkage_rotation)
```

OPTIMIZE.BAS - COMPUTE_STRENGTH

```
'these are used for the position analysis to get Theta3, Theta4, etc.
E3 = 2 * length2 * length3 * Sin(current_input) - 2 * length3 * rot_y
F3 = 2 * length2 * length3 * Cos(current_input) - 2 * length3 * rot_x
G3 = length2 ^ 2 + length3 ^ 2 + rot_x ^ 2 + rot_y ^ 2 - length4 ^ 2 - 2 * length2 *
 -->rot_x * Cos(current_input) - 2 * length2 * rot_y * Sin(current_input)
temp1 = E3 ^ 2 + F3 ^ 2 - G3 ^ 2

E4 = -2 * length2 * length4 * Sin(current_input) + 2 * length4 * rot_y
F4 = -2 * length2 * length4 * Cos(current_input) + 2 * length4 * rot_x
G4 = length2 ^ 2 + length4 ^ 2 + rot_x ^ 2 + rot_y ^ 2 - length3 ^ 2 - 2 * length2 *
 -->rot_x * Cos(current_input) - 2 * length2 * rot_y * Sin(current_input)
temp2 = E4 ^ 2 + F4 ^ 2 - G4 ^ 2

'Debug.Print temp1, temp2

'if both are positive then the linkage assembles, otherwise if either one or both
 -->fails then assembly is no good
If (temp1 >= 0 And temp2 >= 0) Then
   N3 = -E3 + closure * Sqr(temp1)
   N4 = -E4 - closure * Sqr(temp2)
   theta3 = 2 * atan(N3, (G3 - F3))
   theta4 = 2 * atan(N4, (G4 - F4))
   angle_phi = theta4 - weight_arm_offset
   'If (tphi / gconDegToRad > 360) Then tphi = tphi - 360 * gconDegToRad
   coupler_link_force = length_w * Cos(theta4 - weight_arm_offset) / (length4 *
    -->Sin(180 * gconDegToRad + theta3 - theta4))
   'output_strength = -(length2 / length_in) * Sin(theta3 - current_input) *
    -->coupler_link_force * l_type

   n_sub_g = Sin(theta3) * Cos(current_input) - Cos(theta3) * Sin(current_input)
   d_sub_g = Cos(theta4) * Sin(theta3) - Cos(theta3) * Sin(theta4)
   m_sub_g = (length_w * length2 * Cos(angle_phi)) / (length_in * length4)

   output_strength = l_type * (m_sub_g * n_sub_g) / d_sub_g   'new method
    -->from Randy's Thesis 9/27/95
   'Debug.Print current_input / gconDegToRad, theta3 / gconDegToRad, theta4 /
```

OPTIMIZE.BAS - COMPUTE_STRENGTH

```
    -->gconDegToRad, output_strength
ElseIf (temp1 < 0 And temp2 >= 0) Then
   return_error = "Negative Radicals"
   theta3 = 0#
   theta4 = 0#
   angle_phi = 0#
   coupler_link_force = 0#
   output_strength = Abs(temp1) * 100
   'Debug.Print current_input / gconDegToRad, theta3 / gconDegToRad, theta4 /
    -->gconDegToRad, output_strength
ElseIf (temp1 >= 0 And temp2 < 0) Then
   return_error = "Negative Radicals"
   theta3 = 0#
   theta4 = 0#
   angle_phi = 0#
   coupler_link_force = 0#
   output_strength = Abs(temp2) * 100
   'Debug.Print current_input / gconDegToRad, theta3 / gconDegToRad, theta4 /
    -->gconDegToRad, output_strength
ElseIf (temp1 < 0 And temp2 < 0) Then
   return_error = "Negative Radicals"
   theta3 = 0#
   theta4 = 0#
   angle_phi = 0#
   coupler_link_force = 0#
   output_strength = Abs(temp1 * temp2) * 100
   'Debug.Print current_input / gconDegToRad, theta3 / gconDegToRad, theta4 /
    -->gconDegToRad, output_strength
End If

End Sub
```

OPTIMIZE.BAS - DisableControl

```
Sub DisableControl(WhichControl As Control, NewColor)

'This routine disables and colors the control that is passed in.

   WhichControl.Enabled = False

   If Not IsNull(NewColor) Then
      WhichControl.ForeColor = NewColor
   End If

End Sub
```

```
Sub DRAW_SYNTH_LINKAGE(length1 As Double, length2 As Double, length3 As
 -->Double, length4 As Double, linkage_rotation As Double, I_type As Integer,
 -->closure As Integer, length_in As Double, length_w As Double,
 -->input_arm_offset As Double, weight_arm_offset As

'This plots the linkage in the graphics window during synthesis mode.

   'dump solution parameters to debug window
   'Debug.Print "Rotation Direction = "; I_type
   'Debug.Print "Closure = "; closure
   'Debug.Print "Input Arm Length = "; length_in
   'Debug.Print "Weight Arm Length = "; length_w
   'Debug.Print "Input Arm Offset = "; input_arm_offset / gconDegToRad
   'Debug.Print "Weight Arm Offset = "; weight_arm_offset / gconDegToRad
   'Debug.Print "Linkage Base Rotation = "; linkage_rotation / gconDegToRad
   'Debug.Print "Link 1 length = "; length1
   'Debug.Print "Link 2 length = "; length2
   'Debug.Print "Link 3 length = "; length3
   'Debug.Print "Link 4 length = "; length4

   If I_type = 1 Then
      gdblTheta2 = gdblCurveBeta(1, 1) * gconDegToRad - input_arm_offset
   Else
```

OPTIMIZE.BAS - DisableControl

```
      gdblTheta2 = gdblCurveBeta(gintNumDataPoints, 1) * gconDegToRad -
       -->input_arm_offset
   End If

   Debug.Print "Start Angle = "; gdblCurveBeta(1, 1)
   Debug.Print "Current input angle = "; gdblTheta2 / gconDegToRad

   Call COMPUTE_STRENGTH(gdblStrength, gdblCouplerForce, I_type, closure,
    -->gdblTheta2, linkage_rotation, input_arm_offset, weight_arm_offset, length1,
    -->length2, length3, length4, length_w, length_in, gdblTheta3, gdblTheta4,
    -->gdblPhi)

   linkx1 = 0
   linky1 = 0

   linkx4 = length1 * Cos(linkage_rotation)
   linky4 = length1 * Sin(linkage_rotation)

   linkx2 = length2 * Cos(gdblTheta2)
   linky2 = length2 * Sin(gdblTheta2)

   linkx3 = linkx4 + length4 * Cos(gdblTheta4)
   linky3 = linky4 + length4 * Sin(gdblTheta4)

   If I_type = 1 Then
      linkxin = length_in * Cos(gdblCurveBeta(1, 1) * gconDegToRad)
      linkyin = length_in * Sin(gdblCurveBeta(1, 1) * gconDegToRad)
   Else
      linkxin = length_in * Cos(gdblCurveBeta(gintNumDataPoints, 1) *
       -->gconDegToRad)
      linkyin = length_in * Sin(gdblCurveBeta(gintNumDataPoints, 1) *
       -->gconDegToRad)
   End If

   linkxout = linkx4 + length_w * Cos(gdblPhi)
   linkyout = linky4 + length_w * Sin(gdblPhi)
```

```
    frmLinkageDraw.Cls

    frmLinkageDraw.DrawWidth = 2
    frmLinkageDraw.DrawStyle = 0

    frmLinkageDraw.FillStyle = SOLID
    frmLinkageDraw.ForeColor = BLACK
    frmLinkageDraw.FillColor = MEDIUMGRAY
    'frmLinkageDraw.Circle (linkxout, linkyout), length4 / 5#
    frmLinkageDraw.Circle (linkxout, linkyout), 8#

    frmLinkageDraw.ForeColor = gconInputArmColor
    frmLinkageDraw.Line (linkx1, linky1)-(linkx2, linky2)
    frmLinkageDraw.Line (linkx1, linky1)-(linkxin, linkyin)

    frmLinkageDraw.ForeColor = gconWhtArmColor
    frmLinkageDraw.Line (linkx3, linky3)-(linkx4, linky4)
    frmLinkageDraw.Line (linkx4, linky4)-(linkxout, linkyout)

    frmLinkageDraw.ForeColor = gconCouplerColor
    frmLinkageDraw.Line (linkx2, linky2)-(linkx3, linky3)

    frmLinkageDraw.ForeColor = BLACK
    frmLinkageDraw.DrawWidth = 3
    frmLinkageDraw.Line (linkx4, linky4)-(linkx1, linky1)

    frmLinkageDraw.DrawWidth = 1
    frmLinkageDraw.DrawStyle = 2
    frmLinkageDraw.ForeColor = DARKPURPLE
    frmLinkageDraw.Line (linkx2, linky2)-(linkxin, linkyin)
    frmLinkageDraw.Line (linkx3, linky3)-(linkxout, linkyout)

    frmLinkageDraw.Refresh

End Sub
```

OPTIMIZE.BAS - DRAW_SYNTH_LINKAGE

```
Sub EnableControl(WhichControl As Control)

'This routine enables and colors the control that is passed in.

   WhichControl.Enabled = True
   WhichControl.ForeColor = BLACK

End Sub
```

```
Sub EXECUTE_OPTIMIZATION()
'This is the main optimization procedure.  Hooke and Jeeves method is coded here.
'The particulars are left out for brevity but the standard H & J flowchart is
'adhered to for the particular problem.

Dim intCount As Integer

   'reset all of the errors to zero for each call to this sub
   gdblSqrError = 0#
   gdblSumSqrError = 0#
   gdblLastSqrError = 0#
   gdblLastSumSqrError = 0#
   gdblPenalty = 0#

   gdblLastDesVar(1) = gdblBaseRot
   gdblLastDesVar(2) = gdblInputOff
   gdblLastDesVar(3) = gdblWhtOff
   gdblLastDesVar(4) = gdblLen1
   gdblLastDesVar(5) = gdblLen2
   gdblLastDesVar(6) = gdblLen3
   gdblLastDesVar(7) = gdblLen4
   gdblLastDesVar(8) = gdblWhtLen

   gdblDesVar(1) = gdblBaseRot
   gdblDesVar(2) = gdblInputOff
   gdblDesVar(3) = gdblWhtOff
```

OPTIMIZE.BAS - EXECUTE_OPTIMIZATION

```
gdblDesVar(4) = gdblLen1
gdblDesVar(5) = gdblLen2
gdblDesVar(6) = gdblLen3
gdblDesVar(7) = gdblLen4
gdblDesVar(8) = gdblWhtLen

gdblDesVarAfterExplr(1) = gdblBaseRot
gdblDesVarAfterExplr(2) = gdblInputOff
gdblDesVarAfterExplr(3) = gdblWhtOff
gdblDesVarAfterExplr(4) = gdblLen1
gdblDesVarAfterExplr(5) = gdblLen2
gdblDesVarAfterExplr(6) = gdblLen3
gdblDesVarAfterExplr(7) = gdblLen4
gdblDesVarAfterExplr(8) = gdblWhtLen

'========== begin hooke and jeeves code here
-->==========

'calculate the objective function (SSE) of the initial design vector
For intCount = 1 To gintNumDataPoints Step 1
    gdblTheta2 = gdblCurveBeta(intCount, 1) * gconDegToRad - gdblInputOff
    Call COMPUTE_STRENGTH(gdblStrength, gdblCouplerForce, gintLinkType,
     -->gintClosure, gdblTheta2, gdblInitDesVar(1), gdblInitDesVar(2),
     -->gdblInitDesVar(3), gdblInitDesVar(4), gdblInitDesVar(5), gdblInitDesVar(6
     -->), gdblInitDesVar(7), gdblInitDesVar(8), gd
    gdblLastSqrError = Abs(gdblStrength - gdblCurveStrength(intCount, 1) / 100)
     -->^ 2
    gdblLastSumSqrError = gdblLastSumSqrError + gdblLastSqrError
    'Debug.Print gdblLastSqrError, gdblLastSumSqrError; " This is the base error"
Next intCount

ASSESS_PENALTIES

gdblLastSumSqrError = gdblLastSumSqrError + gdblPenalty

Debug.Print gdblLastSumSqrError; " This is the base starting point error for the
 -->initial design vector with penalties"
```

OPTIMIZE.BAS - EXECUTE_OPTIMIZATION

```
If (gconUseDebug = 1) Then Print #1, gdblLastSumSqrError; " This is the base
 -->starting point error for the initial design vector with penalties"

Debug.Print
If (gconUseDebug = 1) Then Print #1,

'label for a goto
reset_q:

  'set the current design variable, gintQ = 1 ... gintNumDesVar
  gintQ = 1

'label for a goto
step_des_var:
  DoEvents

  'if the current design variable is not marked as a constant(a true design
   -->variable)
  If gintDesVarConstant(gintQ) = 1 Then

    'increment the current design variable by its current delta
    gdblDesVar(gintQ) = gdblLastDesVar(gintQ) + gdblStepSize(gintQ)

    For intCount = 1 To gintNumDesVar Step 1
      Debug.Print "Current vars after incrementing positive, and the previous
       -->values = "; gdblDesVar(intCount); gdblLastDesVar(intCount)
      If (gconUseDebug = 1) Then Print #1, "Current vars after incrementing
       -->positive, and the previous values = "; gdblDesVar(intCount);
       -->gdblLastDesVar(intCount)
    Next intCount

    Debug.Print
    If (gconUseDebug = 1) Then Print #1,

  Else
    'increment to the next design variable
    gintQ = gintQ + 1
```

OPTIMIZE.BAS - EXECUTE_OPTIMIZATION

```
Debug.Print "Constant des. var encountered, just incremented design variable
 --> gintQ to "; gintQ

If (gconUseDebug = 1) Then Print #1, "Constant design variable encountered,
 -->just incremented design variable gintQ to "; gintQ
'if we've finished with exploratory moves for each design
'variable, lets begin a pattern move, else we continue exploring above

If (gintQ <= gintNumDesVar) Then
   GoTo step_des_var
Else
   GoTo begin_pattern_move
End If

End If


'reset the errors to be computed in each pass to zero
gdblSqrError = 0
gdblSumSqrError = 0

'calculate the objective function at this new point
For intCount = 1 To gintNumDataPoints Step 1
   gdblTheta2 = gdblCurveBeta(intCount, 1) * gconDegToRad - gdblInputOff
   Call COMPUTE_STRENGTH(gdblStrength, gdblCouplerForce, gintLinkType,
    -->gintClosure, gdblTheta2, gdblDesVar(1), gdblDesVar(2), gdblDesVar(3),
    -->gdblDesVar(4), gdblDesVar(5), gdblDesVar(6), gdblDesVar(7),
    -->gdblDesVar(8), gdblInputLen, gdblTheta3, gdblThet
   gdblSqrError = Abs(gdblStrength - gdblCurveStrength(intCount, 1) / 100) ^ 2
   gdblSumSqrError = gdblSumSqrError + gdblSqrError
   'Debug.Print gdblSqrError, gdblSumSqrError; "This is error after positive step
    -->in gintQ="; gintQ
Next intCount

ASSESS_PENALTIES
```

OPTIMIZE.BAS - EXECUTE_OPTIMIZATION

```
gdblSumSqrError = gdblSumSqrError + gdblPenalty

Debug.Print gdblSumSqrError; "This is error after a positive step in gintQ = ";
 -->gintQ

If (gconUseDebug = 1) Then Print #1, gdblSumSqrError; "This is error after a
 -->positive step in gintQ = "; gintQ

'check to see if objective function value is improved at this new point
If (gdblSumSqrError < gdblLastSumSqrError) Then
   'retain this point and increase step size
   Debug.Print "positive step: computed error, and the comparison error";
    -->gdblSumSqrError; gdblLastSumSqrError
   If (gconUseDebug = 1) Then Print #1, "positive step: computed error, and the
    -->comparison error"; gdblSumSqrError; gdblLastSumSqrError
   Debug.Print "Error after a positive step was less in gintQ = "; gintQ
   If (gconUseDebug = 1) Then Print #1, "Error after a positive step was less in
    -->gintQ = "; gintQ
   Debug.Print
   If (gconUseDebug = 1) Then Print #1,

   'flag this exploratory move as successful for the soon to come pattern move
   gintExplrSuccess(gintQ) = 1

   Debug.Print "Positive exploratory move successful in gintQ = "; gintQ
   If (gconUseDebug = 1) Then Print #1, "Positive exploratory move successful
    -->in gintQ = "; gintQ

   gdblLastDesVar(gintQ) = gdblDesVar(gintQ)
   gdblStepSize(gintQ) = 1.2 * gdblStepSize(gintQ)
   'retain the error at this new retained point for the next comparison
   gdblLastSumSqrError = gdblSumSqrError

Else

   Debug.Print "positive step: computed error, and the comparison error";
    -->gdblSumSqrError; gdblLastSumSqrError
```

OPTIMIZE.BAS - EXECUTE_OPTIMIZATION

```
If (gconUseDebug = 1) Then Print #1, "positive step: computed error, and the
 -->comparison error"; gdblSumSqrError; gdblLastSumSqrError
Debug.Print "Error after a positive step was more, not ok in gintQ = "; gintQ
If (gconUseDebug = 1) Then Print #1, "Error after a positive step was more,
 -->not ok in gintQ = "; gintQ
Debug.Print
If (gconUseDebug = 1) Then Print #1,

gdblSqrError = 0
gdblSumSqrError = 0

'reverse direction and go the other way with the design variable
gdblStepSize(gintQ) = -gdblStepSize(gintQ)
gdblDesVar(gintQ) = gdblLastDesVar(gintQ) + gdblStepSize(gintQ)

For intCount = 1 To gintNumDesVar Step 1
   Debug.Print "Current vars after incrementing negative, and the previous
    -->values = "; gdblDesVar(intCount); gdblLastDesVar(intCount)
   If (gconUseDebug = 1) Then Print #1, "Current vars after incrementing
    -->negative, and the previous values = "; gdblDesVar(intCount);
    -->gdblLastDesVar(intCount)
Next intCount

Debug.Print
If (gconUseDebug = 1) Then Print #1,

'recalculate the objective function at this point in the opposite direction
For intCount = 1 To gintNumDataPoints Step 1
   gdblTheta2 = gdblCurveBeta(intCount, 1) * gconDegToRad - gdblInputOff
   Call COMPUTE_STRENGTH(gdblStrength, gdblCouplerForce, gintLinkType,
    -->gintClosure, gdblTheta2, gdblDesVar(1), gdblDesVar(2), gdblDesVar(3
    -->), gdblDesVar(4), gdblDesVar(5), gdblDesVar(6), gdblDesVar(7),
    -->gdblDesVar(8), gdblInputLen, gdblTheta3, gdbl
   gdblSqrError = Abs(gdblStrength - gdblCurveStrength(intCount, 1) / 100)
    -->^ 2
   gdblSumSqrError = gdblSumSqrError + gdblSqrError
   'Debug.Print gdblSqrError, gdblSumSqrError; "This is error after negative
```

OPTIMIZE.BAS - EXECUTE_OPTIMIZATION

```
    -->step in gintQ="; gintQ
Next intCount

ASSESS_PENALTIES

gdblSumSqrError = gdblSumSqrError + gdblPenalty

Debug.Print gdblSumSqrError; "This is error after negative step in gintQ = ";
 -->gintQ
If (gconUseDebug = 1) Then Print #1, gdblSumSqrError; "This is error after
 -->negative step in gintQ = "; gintQ

'For intCount = 1 To gintNumDesVar Step 1
'   Debug.Print "current vars after increment, old values = "; gdblDesVar(
 -->intCount); gdblLastDesVar(intCount)
'Next intCount
'Debug.Print
If (gconUseDebug = 1) Then Print #1,

'check the objective function at the reversed point and see if it's any better
If (gdblSumSqrError < gdblLastSumSqrError) Then
   'retain this point and increase step size for this design variable
   Debug.Print "negative step: computed error, comparison error";
    -->gdblSumSqrError; gdblLastSumSqrError
   If (gconUseDebug = 1) Then Print #1, "negative step: computed error,
    -->comparison error"; gdblSumSqrError; gdblLastSumSqrError
   Debug.Print "Error after a negative step was less, ok in gintQ = "; gintQ
   If (gconUseDebug = 1) Then Print #1, "Error after a negative step was less
    -->, ok in gintQ = "; gintQ
   Debug.Print
   If (gconUseDebug = 1) Then Print #1,

   'flag this exploratory move as successful for the soon to come pattern
    -->move
   gintExplrSuccess(gintQ) = 1

   Debug.Print "Negative exploratory move successful in gintQ = "; gintQ
```

OPTIMIZE.BAS - EXECUTE_OPTIMIZATION

```
        If (gconUseDebug = 1) Then Print #1, "Negative exploratory move
         -->successful in gintQ = "; gintQ

        gdblLastDesVar(gintQ) = gdblDesVar(gintQ)
        gdblStepSize(gintQ) = 1.2 * gdblStepSize(gintQ)
        'retain the error at this new retained point for the next comparison
        gdblLastSumSqrError = gdblSumSqrError

    Else

        'don't retain point and decrease step size for this design variable
        Debug.Print "computed error, comparison error"; gdblSumSqrError;
         -->gdblLastSumSqrError
        If (gconUseDebug = 1) Then Print #1, "computed error, comparison error";
         -->gdblSumSqrError; gdblLastSumSqrError
        Debug.Print "Error after negative step was also more, not ok in gintQ = ";
         -->gintQ
        If (gconUseDebug = 1) Then Print #1, "Error after negative step was also
         -->more, not ok in gintQ = "; gintQ
        Debug.Print
        If (gconUseDebug = 1) Then Print #1,

        'flag this exploratory move as unsuccessful for the coming pattern move,
         -->this variable will not be altered in a pattern move
        gintExplrSuccess(gintQ) = 0

        Debug.Print "No exploratory move successful in gintQ = "; gintQ
        If (gconUseDebug = 1) Then Print #1, "No exploratory move successful in
         -->gintQ = "; gintQ

        gdblStepSize(gintQ) = 0.5 * gdblStepSize(gintQ)
    End If
End If

For intCount = 1 To gintNumDesVar Step 1
    Debug.Print "Final vars after the conditionals, also their previous old values (
     -->gdblDesVar(), gdblLastDesVar() "; gdblDesVar(intCount);
```

OPTIMIZE.BAS - EXECUTE_OPTIMIZATION

```
     -->gdblLastDesVar(intCount)
    If (gconUseDebug = 1) Then Print #1, "Final vars after the conditionals, also
     -->their previous old values (gdblDesVar(), gdblLastDesVar() "; gdblDesVar(
     -->intCount); gdblLastDesVar(intCount)
Next intCount

Debug.Print
If (gconUseDebug = 1) Then Print #1,

'increment to the next design variable
gintQ = gintQ + 1

Debug.Print "Just incremented design variable gintQ to "; gintQ
If (gconUseDebug = 1) Then Print #1, "Just incremented design variable gintQ to
 -->"; gintQ

'if we've finished with exploratory moves for each design
'variable, lets begin a pattern move, else we continue exploring above
If (gintQ <= gintNumDesVar) Then GoTo step_des_var

'label for a goto statement above
begin_pattern_move:

For intCount = 1 To gintNumDesVar
    Debug.Print gdblLastDesVar(intCount), gdblDesVarAfterExplr(intCount)
    If (gconUseDebug = 1) Then Print #1, gdblLastDesVar(intCount),
     -->gdblDesVarAfterExplr(intCount)
Next intCount

'before pattern move, see if we should bail out because the variables aren't
 -->changing anymore
If gdblLastDesVar(1) = gdblDesVarAfterExplr(1) And gdblLastDesVar(2) =
 -->gdblDesVarAfterExplr(2) And gdblLastDesVar(3) = gdblDesVarAfterExplr(3
 -->) And gdblLastDesVar(4) = gdblDesVarAfterExplr(4) And gdblLastDesVar(5
 -->) = gdblDesVarAfterExplr(5) And gdblLast
    Debug.Print "No change in the variables here, so we'll check and see if we've
     --> reached the limits."
```

OPTIMIZE.BAS - EXECUTE_OPTIMIZATION

```
If (gconUseDebug = 1) Then Print #1, "No change in the variables here, so
 -->we'll check and see if we've reached the limits."

'check to see if all of the step minimums were exceeded
If Abs(gdblStepSize(1) * gintDesVarConstant(1)) < gdblMinStepSize(1) And
 -->Abs(gdblStepSize(2) * gintDesVarConstant(2)) < gdblMinStepSize(2) And
 -->Abs(gdblStepSize(3) * gintDesVarConstant(3)) < gdblMinStepSize(3) And
 -->Abs(gdblStepSize(4) * gintDesVarCo
   Debug.Print "All minimum steps exceeded, jumping to end"
   If (gconUseDebug = 1) Then Print #1, "All minimum steps exceeded,
    -->jumping to end"
   Debug.Print
   Debug.Print
   Debug.Print
   Debug.Print
   If (gconUseDebug = 1) Then Print #1,
   If (gconUseDebug = 1) Then Print #1,
   If (gconUseDebug = 1) Then Print #1,
   If (gconUseDebug = 1) Then Print #1,

   GoTo exceeded_min_step

Else

   'this is assigning the current values of the design variables after the
    -->exploratory
   'moves to an array for comparing after each complete exploration before a
    -->pattern move is attempted
   Debug.Print "The variables did not change with that pass, but the minimum
    -->steps were not exceeded, so just go back up top"
   If (gconUseDebug = 1) Then Print #1, "The variables did not change with
    -->that pass, but the minimum steps were not exceeded, so just go back
    -->up top"

   For intCount = 1 To gintNumDesVar
      gdblDesVarAfterExplr(intCount) = gdblLastDesVar(intCount)
   Next intCount
```

OPTIMIZE.BAS - EXECUTE_OPTIMIZATION

```
      GoTo reset_q

   End If

Else

   'this is assigning the current values of the design variables after the
    -->exploratory
   'moves to an array for comparing after each complete exploration before a
    -->pattern move is attempted
   For intCount = 1 To gintNumDesVar
      gdblDesVarAfterExplr(intCount) = gdblLastDesVar(intCount)
   Next intCount

End If

Debug.Print "gintQ reached limits, Begin a pattern move now"
If (gconUseDebug = 1) Then Print #1, "gintQ reached limits, Begin a pattern
 -->move now"

'Now try a pattern move by repeating the previously successful exploratory
 -->moves
For intCount = 1 To gintNumDesVar Step 1
   'increment each design variable that had a successful exploratory move
   gdblDesVar(intCount) = gdblLastDesVar(intCount) + gdblStepSize(intCount) *
    -->gintExplrSuccess(intCount)

   Debug.Print " Variables after a pattern move increment: var, last_var, step,
    -->success = "; gdblDesVar(intCount), gdblLastDesVar(intCount),
    -->gdblStepSize(intCount), gintExplrSuccess(intCount)
   If (gconUseDebug = 1) Then Print #1, " Variables after a pattern move
    -->increment: var, last_var, step, success = "; gdblDesVar(intCount),
    -->gdblLastDesVar(intCount), gdblStepSize(intCount), gintExplrSuccess(
    -->intCount)

   'reset all of the exploratory success values to 0 for next pass through
```

OPTIMIZE.BAS - EXECUTE_OPTIMIZATION

```
    gintExplrSuccess(intCount) = 0
Next intCount

'reset the strength errors to zero again for the next calculation
gdblSqrError = 0
gdblSumSqrError = 0

'recalculate the objective function after this pattern move
For intCount = 1 To gintNumDataPoints Step 1
    gdblTheta2 = gdblCurveBeta(intCount, 1) * gconDegToRad - gdblInputOff
    Call COMPUTE_STRENGTH(gdblStrength, gdblCouplerForce, gintLinkType,
     -->gintClosure, gdblTheta2, gdblDesVar(1), gdblDesVar(2), gdblDesVar(3),
     -->gdblDesVar(4), gdblDesVar(5), gdblDesVar(6), gdblDesVar(7),
     -->gdblDesVar(8), gdblInputLen, gdblTheta3, gdblThet
    gdblSqrError = Abs(gdblStrength - gdblCurveStrength(intCount, 1) / 100) ^ 2
    gdblSumSqrError = gdblSumSqrError + gdblSqrError
    'Debug.Print gdblSqrError, gdblSumSqrError; "This is error after pattern move"
Next intCount

ASSESS_PENALTIES

gdblSumSqrError = gdblSumSqrError + gdblPenalty

Debug.Print "This is error after the pattern move"; gdblSumSqrError
If (gconUseDebug = 1) Then Print #1, "This is error after the pattern move";
 -->gdblSumSqrError
Debug.Print
If (gconUseDebug = 1) Then Print #1,

'check the objective function after the pattern move and see if it's any better
 -->than the last starting point (the saved error)
If (gdblSumSqrError < gdblLastSumSqrError) Then
    Debug.Print "pattern move: computed error, comparison error";
     -->gdblSumSqrError; gdblLastSumSqrError
    If (gconUseDebug = 1) Then Print #1, "pattern move: computed error,
     -->comparison error"; gdblSumSqrError; gdblLastSumSqrError
    Debug.Print "Error after a pattern move was less, ok"
```

OPTIMIZE.BAS - EXECUTE_OPTIMIZATION

```
    If (gconUseDebug = 1) Then Print #1, "Error after a pattern move was less,
     -->ok"
    Debug.Print
    If (gconUseDebug = 1) Then Print #1,

    'retain this point for each of the design variables
    For intCount = 1 To gintNumDesVar Step 1
        gdblLastDesVar(intCount) = gdblDesVar(intCount)
    Next intCount

    'also retain the error at this new retained point for the next comparison in
     -->exploratory moves
    gdblLastSumSqrError = gdblSumSqrError

Else

    'don't retain point, reset the incremented values of the pattern move to their
     -->previous values, then reexplore
    For intCount = 1 To gintNumDesVar Step 1
        gdblDesVar(intCount) = gdblLastDesVar(intCount)
    Next intCount

    Debug.Print "pattern move: computed error, comparison error";
     -->gdblSumSqrError; gdblLastSumSqrError
    If (gconUseDebug = 1) Then Print #1, "pattern move: computed error,
     -->comparison error"; gdblSumSqrError; gdblLastSumSqrError
    Debug.Print "Error after a pattern move was more, not ok"
    Debug.Print
    If (gconUseDebug = 1) Then Print #1,
End If

For intCount = 1 To gintNumDesVar Step 1
    Debug.Print "The vars after the pattern move: var, last_var = "; gdblDesVar(
     -->intCount), gdblLastDesVar(intCount)
    If (gconUseDebug = 1) Then Print #1, "The vars after the pattern move: var,
     -->last_var = "; gdblDesVar(intCount), gdblLastDesVar(intCount)
Next intCount
```

```
'Before returning to the exploratory section at the top, reset all of the step
 -->values to positive ones, for consistency
For intCount = 1 To gintNumDesVar Step 1
   If gdblStepSize(intCount) < 0 Then gdblStepSize(intCount) = -gdblStepSize(
    -->intCount)
Next intCount

Debug.Print "Going back to top"
If (gconUseDebug = 1) Then Print #1, "Going back to top"
Debug.Print
Debug.Print
Debug.Print
Debug.Print
Debug.Print
If (gconUseDebug = 1) Then Print #1,
If (gconUseDebug = 1) Then Print #1,
If (gconUseDebug = 1) Then Print #1,
If (gconUseDebug = 1) Then Print #1,
If (gconUseDebug = 1) Then Print #1,

   GoTo reset_q

'goto label used for minimum step size reached
exceeded_min_step:

End Sub
```

```
Function FILE_EXISTS(filename_or_path As String) As Integer

'This function determines if the specified file already exists
'and returns the answer.

On Error GoTo path_error

   FILE_EXISTS% = (Dir$(filename_or_path) <> "")
```



```
   Exit Function

path_error:

   FILE_EXISTS% = 0
   Exit Function

End Function
```

```
Function FIND_VECTOR_MAX(a() As Double) As Double

'This function returns the largest number in an array.

Dim intStart As Integer, intFinish As Integer, intCount As Integer

intStart = LBound(a, 1) + 1
intFinish = UBound(a, 1)

Max = -1000
For intCount = intStart To intFinish Step 1
   If a(intCount, 1) > Max Then Max = a(intCount, 1)
Next intCount

FIND_VECTOR_MAX = Max

End Function
```

```
Function FIND_VECTOR_MIN(a() As Double) As Double

'This function returns the smallest number in an array.

Dim intStart As Integer, intFinish As Integer, intCount As Integer

intStart = LBound(a, 1) + 1
intFinish = UBound(a, 1)
```

```
Min = 1000
For intCount = intStart To intFinish Step 1
   If a(intCount, 1) < Min Then Min = a(intCount, 1)
Next intCount

FIND_VECTOR_MIN = Min

End Function
```

```
Sub GET_OUTPUT_RANGE(gdblPhiStart As Double, gdblPhiEnd As Double,
 -->gdblPhiRange As Double)

'This routine gets the starting and ending weight arm angles for a linkage.
'The strength routine is called to get the output angle phi and then these angles are
'converted to degrees.  The values are also clamped to be either between
'0 and 180 degrees and 0 and -180 degrees.  The values determined here are
'used in assessing penalties for constraints in the calling routine.

   'get the starting output angle
   gdblTheta2 = gdblCurveBeta(1, 1) * gconDegToRad - gdblLastDesVar(2)
   Call COMPUTE_STRENGTH(gdblStrength, gdblCouplerForce, gintLinkType,
    -->gintClosure, gdblTheta2, gdblLastDesVar(1), gdblLastDesVar(2),
    -->gdblLastDesVar(3), gdblLastDesVar(4), gdblLastDesVar(5),
    -->gdblLastDesVar(6), gdblLastDesVar(7), gdblLastDesVar(8), gdblIn
   gdblPhiStart = gdblPhi / gconDegToRad

   Do Until gdblPhiStart >= -180#
      gdblPhiStart = gdblPhiStart + 360#
   Loop

   Do Until gdblPhiStart <= 180#
      gdblPhiStart = gdblPhiStart - 360#
   Loop

   'get the ending output angle
```



```
   gdblTheta2 = gdblCurveBeta(gintNumDataPoints, 1) * gconDegToRad -
    -->gdblLastDesVar(2)
   Call COMPUTE_STRENGTH(gdblStrength, gdblCouplerForce, gintLinkType,
    -->gintClosure, gdblTheta2, gdblLastDesVar(1), gdblLastDesVar(2),
    -->gdblLastDesVar(3), gdblLastDesVar(4), gdblLastDesVar(5),
    -->gdblLastDesVar(6), gdblLastDesVar(7), gdblLastDesVar(8), gdblIn
   gdblPhiEnd = gdblPhi / gconDegToRad

   Do Until gdblPhiEnd >= -180#
      gdblPhiEnd = gdblPhiEnd + 360#
   Loop

   Do Until gdblPhiEnd <= 180#
      gdblPhiEnd = gdblPhiEnd - 360#
   Loop

   'get the output range in absolute value
   gdblPhiRange = Abs(gdblPhiEnd - gdblPhiStart)

End Sub
```

```
Sub LOAD_STRENGTH_DATA()

'This routine loads the objective strength curve for the optimization
'and then makes calls to procedures to plot it after loading it into
'arrays.

Dim dblMinX As Double
Dim dblMaxX As Double

start_of_procedure:

   On Error GoTo getdatacancelerror

   gintNumDataPoints = 0
   gintCurrDataPoint = 0
```

OPTIMIZE.BAS - LOAD_STRENGTH_DATA

```
ChDir App.Path
frmMainOptimzation.dlgLoadStrengthData.Filename = ""
frmMainOptimzation.dlgLoadStrengthData.Action = 1

gstrStrengthDataFile = frmMainOptimzation.dlgLoadStrengthData.Filename

'================================================================
-->=========================

If (FILE_EXISTS(gstrStrengthDataFile)) = 0 Then
   error_on_open_message = "The strength data file " + gstrStrengthDataFile +
    -->" was not found at the expected location.  Check the path, filename,
    -->spelling, etc..  The Acquire Strength Data dialog box will be displayed
    -->after closing this message."
   MsgBox error_on_open_message, 0 + 16, "Problem Opening File"
   GoTo start_of_procedure
End If

Open gstrStrengthDataFile For Input As #10
Do While Not EOF(10)
   Input #10, dummy1, dummy2
   gintNumDataPoints = gintNumDataPoints + 1
Loop
Close #10

frmDataGrid.grdStrengthData.Rows = gintNumDataPoints + 1

ReDim gdblCurveBeta(gintNumDataPoints, 1) As Double
ReDim gdblCurveStrength(gintNumDataPoints, 1) As Double

Open gstrStrengthDataFile For Input As #10
Do
   gintCurrDataPoint = gintCurrDataPoint + 1
   Input #10, gdblCurveBeta(gintCurrDataPoint, 1), gdblCurveStrength(
    -->gintCurrDataPoint, 1)
   frmDataGrid.grdStrengthData.Row = gintCurrDataPoint
```

OPTIMIZE.BAS - LOAD_STRENGTH_DATA

```
   frmDataGrid.grdStrengthData.Col = 0
   frmDataGrid.grdStrengthData.Text = gdblCurveBeta(gintCurrDataPoint, 1)
   frmDataGrid.grdStrengthData.Col = 1
   frmDataGrid.grdStrengthData.Text = gdblCurveStrength(gintCurrDataPoint, 1)
   'Debug.Print gdblCurveBeta(gintCurrDataPoint, 1), gdblCurveStrength(
    -->gintCurrDataPoint, 1)
Loop Until EOF(10)
Close #10

frmDataGrid.grdStrengthData.Visible = True

frmSynthCurves.Caption = "Discrete Data From " + gstrStrengthDataFile
frmSynthCurves.gphComboPlot.OverlayGraph = 0
frmSynthCurves.gphComboPlot.DataReset = 1
frmSynthCurves.gphComboPlot.DataReset = 8
frmSynthCurves.gphComboPlot.ThisPoint = 1
frmSynthCurves.gphComboPlot.NumPoints = 2

For i = 1 To gintNumDataPoints Step 1
   frmSynthCurves.gphComboPlot.XPosData = gdblCurveBeta(i, 1)
   frmSynthCurves.gphComboPlot.GraphData = gdblCurveStrength(i, 1)
   frmSynthCurves.gphComboPlot.ThisPoint = frmSynthCurves.gphComboPlot.
    -->ThisPoint + 1
   frmSynthCurves.gphComboPlot.NumPoints = frmSynthCurves.gphComboPlot.
    -->ThisPoint + 1
Next i
frmSynthCurves.gphComboPlot.NumPoints = frmSynthCurves.gphComboPlot.
 -->ThisPoint - 1
frmSynthCurves.gphComboPlot.DrawMode = 2

frmDataGrid.Show
frmSynthCurves.Show

dblMinX = FIND_VECTOR_MIN(gdblCurveBeta())
dblMaxX = FIND_VECTOR_MAX(gdblCurveBeta())
gdblCurveBetaRange = dblMaxX - dblMinX
'Debug.Print dblMinX / gconDegToRad, dblMaxX / gconDegToRad,
```

```
    -->gdblCurveBetaRange / gconDegToRad

    Exit Sub

getdatacancelerror:

    MsgBox "An Error has occured in acquiring the strength curve data.  Check the
     -->file perhaps."
    Close #10
    frmMDI.mnuOptimizationStart.Enabled = False
    Exit Sub

End Sub
```

OPTIMIZE.BAS - LOAD_STRENGTH_DATA

```
Function AnyPadsLeft () As Integer
    Dim i As Integer

    ' Cycle throught the document array.
    ' Return True if there is at least one
    ' open document remaining.
    For i = 1 To UBound(Document)
        If Not FState(i).Deleted Then
            AnyPadsLeft = True
            Exit Function
        End If
    Next

End Function
```

```
Sub CenterForm (frmParent As Form, frmChild As Form)
' This procedure centers a child form over a parent form.
' Calling this routine loads the dialog. Use the Show method
' to display the dialog after calling this routine ( ie MyFrm.Show 1)

Dim l, t
    ' get left offset
    l = frmParent.Left + ((frmParent.Width - frmChild.Width) / 2)
    If (l + frmChild.Width > screen.Width) Then
        l = screen.Width = frmChild.Width
    End If

    ' get top offset
    t = frmParent.Top + ((frmParent.Height - frmChild.Height) / 2)
    If (t + frmChild.Height > screen.Height) Then
        t = screen.Height - frmChild.Height
    End If

    ' center the child formfv
    frmChild.Move l, t
```

```
End Sub

---

Sub EditCopyProc ()
   ' Copy selected text to Clipboard.
   ClipBoard.SetText frmMDI.ActiveForm.ActiveControl.SelText
End Sub

---

Sub EditCutProc ()
   ' Copy selected text to Clipboard.
   ClipBoard.SetText frmMDI.ActiveForm.ActiveControl.SelText
   ' Delete selected text.
   frmMDI.ActiveForm.ActiveControl.SelText = ""
End Sub

---

Sub EditPasteProc ()
   ' Place text from Clipboard into active control.
   frmMDI.ActiveForm.ActiveControl.SelText = ClipBoard.GetText()
End Sub

---

Sub FileNew ()
   Dim fIndex As Integer

   fIndex = FindFreeIndex()
   Document(fIndex).Tag = fIndex
   Document(fIndex).Caption = "Untitled:" & fIndex
   Document(fIndex).Show

   ' Make sure toolbar edit buttons are visible
   frmMDI!imgCutButton.Visible = True
   frmMDI!imgCopyButton.Visible = True
   frmMDI!imgPasteButton.Visible = True
```

MDINOTE.BAS - CenterForm

```
End Sub

---

Function FindFreeIndex () As Integer
   Dim i As Integer
   Dim ArrayCount As Integer

   ArrayCount = UBound(Document)

   ' Cycle throught the document array. If one of the
   ' documents has been deleted, then return that
   ' index.
   For i = 1 To ArrayCount
      If FState(i).Deleted Then
         FindFreeIndex = i
         FState(i).Deleted = False
         Exit Function
      End If
   Next

   ' If none of the elements in the document array have
   ' been deleted, then increment the document and the
   ' state arrays by one and return the index to the
   ' new element.

   ReDim Preserve Document(ArrayCount + 1)
   ReDim Preserve FState(ArrayCount + 1)
   FindFreeIndex = UBound(Document)
End Function

---

Sub FindIt ()
   Dim start, pos, findstring, sourcestring, msg, response, Offset

   If (gCurPos = frmMDI.ActiveForm.ActiveControl.SelStart) Then
      Offset = 1
   Else
```

MDINOTE.BAS - FindIt

```
        Offset = 0
    End If

    If gFirstTime Then Offset = 0

    start = frmMDI.ActiveForm.ActiveControl.SelStart + Offset

    If gFindCase Then
        findstring = gFindString
        sourcestring = frmMDI.ActiveForm.ActiveControl.Text
    Else
        findstring = UCase(gFindString)
        sourcestring = UCase(frmMDI.ActiveForm.ActiveControl.Text)
    End If

    If gFindDirection = 1 Then
        pos = InStr(start + 1, sourcestring, findstring)
    Else
        For pos = start - 1 To 0 Step -1
            If pos = 0 Then Exit For
            If Mid(sourcestring, pos, Len(findstring)) = findstring Then Exit For
        Next
    End If

    ' If string is found
    If pos Then
        frmMDI.ActiveForm.ActiveControl.SelStart = pos - 1
        frmMDI.ActiveForm.ActiveControl.SelLength = Len(findstring)
    Else
        msg = "Cannot find " & Chr(34) & gFindString & Chr(34)
        response = MsgBox(msg, 0, App.Title)
    End If

    gCurPos = frmMDI.ActiveForm.ActiveControl.SelStart
    gFirstTime = False

End Sub
```

MDINOTE.BAS - FindIt

```
Sub GetRecentFiles ()
  Dim retval, key, i, j
  Dim IniString As String

  ' This variable must be large enough to hold the return string
  ' from the GetPrivateProfileString API.
  IniString = String(255, 0)

  ' Get recent file strings from MDINOTE.INI
  For i = 1 To 4
    key = "RecentFile" & i
    retval = GetPrivateProfileString("Recent Files", key, "Not Used", IniString, Len(
     -->IniString), "MDINOTE.INI")
    If retval And Left(IniString, 8) <> "Not Used" Then
      ' Update the MDI form's menu.
      frmMDI.mnuRecentFile(0).Visible = True
      frmMDI.mnuRecentFile(i).Caption = IniString
      frmMDI.mnuRecentFile(i).Visible = True

      ' Iterate through all the notepads and update each menu.
      For j = 1 To UBound(Document)
        If Not FState(j).Deleted Then
          Document(j).mnuRecentFile(0).Visible = True
          Document(j).mnuRecentFile(i).Caption = IniString
          Document(j).mnuRecentFile(i).Visible = True
        End If
      Next j
    End If
  Next i

End Sub
```

MDINOTE.BAS - OptionsToolbarProc

```
Sub OptionsToolbarProc (CurrentForm As Form)
    CurrentForm.mnuOToolbar.Checked = Not CurrentForm.mnuOToolbar.Checked
    If TypeOf CurrentForm Is MDIForm Then
    Else
        'frmMDI.mnuOToolbar.Checked = CurrentForm.mnuOToolbar.Checked
    End If
    If CurrentForm.mnuOToolbar.Checked Then
        frmMDI.picToolbar.Visible = True
    Else
        frmMDI.picToolbar.Visible = False
    End If
End Sub
```

```
Sub WriteRecentFiles (OpenFileName)
  Dim i, j, key, retval
  Dim IniString As String
  IniString = String(255, 0)

  ' Copy RecentFile1 to RecentFile2, etc.
  For i = 3 To 1 Step -1
    key = "RecentFile" & i
    retval = GetPrivateProfileString("Recent Files", key, "Not Used", IniString, Len(
     -->IniString), "MDINOTE.INI")
    If retval And Left(IniString, 8) <> "Not Used" Then
      key = "RecentFile" & (i + 1)
      retval = WritePrivateProfileString("Recent Files", key, IniString, "MDINOTE.INI")
    End If
  Next i

  ' Write openfile to first Recent File.
    retval = WritePrivateProfileString("Recent Files", "RecentFile1", OpenFileName, "
     -->MDINOTE.INI")

End Sub
```

MDINOTE.BAS - OptionsToolbarProc

```
Private Sub cmdMechClosure_Click()

    If gintClosure = 1 Then
        gintClosure = -1
    ElseIf gintClosure = -1 Then
        gintClosure = 1
    End If
    Debug.Print gintClosure

    If gintClosure = 1 Then
        cmdMechClosure.Caption = "Quad = 1"
        cmdMechClosure.ForceUp = True
    Else
        cmdMechClosure.Caption = "Quad = -1"
        cmdMechClosure.ForceDown = True
    End If

End Sub
```

```
Private Sub cmdMechType_Click()

    If gintLinkType = 1 Then
        gintLinkType = -1
    ElseIf gintLinkType = -1 Then
        gintLinkType = 1
    End If
    Debug.Print gintLinkType

    If gintLinkType = 1 Then
        cmdMechType.Caption = "Co-Rotate"
        cmdMechType.ForceUp = True
    Else
        cmdMechType.Caption = "Counter-Rotate"
        cmdMechType.ForceDown = True
    End If
```

```
End Sub

---

Private Sub Form_Load()

   SystemMenu% = GetSystemMenu(frmMainOptimzation.hWnd, 0)
   Res% = RemoveMenu(SystemMenu%, 6, MF_BYPOSITION)
   Res% = RemoveMenu(SystemMenu%, 6, MF_BYPOSITION) 'also remove the
    -->separator line

   frmMainOptimzation.Top = 0
   frmMainOptimzation.Left = 0
   frmMainOptimzation.Height = 6780
   frmMainOptimzation.Width = 6360

   frmSynthTextOutput.Show
   frmLinkageDraw.Show
   frmDataGrid.Show
   frmSynthCurves.Show
   'frmDiagram.Show
   INITIALIZE_FORM

End Sub

---

Private Sub INITIALIZE_FORM()

'This routine is called when the program is first started up and
'not again after that.  It simply supplies some default values for
'variables that can be changed by the user.  It also sets up some
'defaults for the optimization routine.  Several linkage profiles
'are included.

'gdblLen1 = 27.63
'gdblLen2 = 12.89
'gdblLen3 = 14.42
```

MAIN.FRM - cmdMechType_Click

```
'gdblLen4 = 5.08
'gdblInputLen = 14.25
'gdblWhtLen = 36#
'gdblBaseRot = 131.98
'gdblInputOff = 225.78
'gdblWhtOff = 346.71
'gintClosure = 1
'gintLinkType = 1

'set the total number of possible design variables to use
gintNumDesVar = 8

'sets the number of solutions to look for
gintNumOptRuns = 1000

'set the halt flag to false
gintAnimStop = False

'set some constraint defaults
gdblErrorLimit = 0.3
gintPhiStartConstrainOn = -2
gdblPhiStartLow = -30#
gdblPhiStartHi = 20#
gdblPhiStartPenalty = 500#
gintPhiEndConstrainOn = -2
gdblPhiEndLow = 0#
gdblPhiEndHi = 50#
gdblPhiEndPenalty = 500#

'set some random leap controls
gdblRndBaseRotHi = 360#
gdblRndBaseRotLow = 0
gdblRndInputOffHi = 360
gdblRndInputOffLow = 0
gdblRndWhtOffHi = 360
gdblRndWhtOffLow = 0
gdblRndLen1Low = 3
```

```
gdblRndLen1Hi = 20
gdblRndLen2Low = 3
gdblRndLen2Hi = 20
gdblRndLen3Low = 3
gdblRndLen3Hi = 20
gdblRndLen4Low = 3
gdblRndLen4Hi = 20
gdblRndWhtLenLow = 10
gdblRndWhtLenHi = 40

'Initializes the random-number generator
'It uses the clock timer as the seed
Randomize

'this set is good calf
gdblLen1 = 5#
gdblLen2 = 3.55
gdblLen3 = 0.67
gdblLen4 = 8.59
gdblInputLen = 14.25
gdblWhtLen = 36#
gdblBaseRot = 0#
gdblInputOff = 143.9
gdblWhtOff = -155.03
gintClosure = -1
gintLinkType = 1

'this set is good compound row
'gdblLen1 = 5#
'gdblLen2 = 11.92
'gdblLen3 = 7.04
'gdblLen4 = 16.96
'gdblInputLen = 40.1
'gdblWhtLen = 45#
'gdblBaseRot = 0#
'gdblInputOff = 177.89
'gdblWhtOff = -164.73
```

MAIN.FRM - INITIALIZE_FORM

```
'gintClosure = 1
'gintLinkType = 1

'B_start = 60
'B_finish = 100

'convert all angles from degrees to radians
gdblBaseRot = gdblBaseRot * gconDegToRad
gdblInputOff = gdblInputOff * gconDegToRad
gdblWhtOff = gdblWhtOff * gconDegToRad

'redimension all of the arrays to the proper size
ReDim gdblInitDesVar(gintNumDesVar)
ReDim gdblDesVar(gintNumDesVar)
ReDim gintExplrSuccess(gintNumDesVar)
ReDim gintDesVarConstant(gintNumDesVar)
ReDim gdblLastDesVar(gintNumDesVar)
ReDim gdblDesVarAfterExplr(gintNumDesVar)
ReDim gdblStepSize(gintNumDesVar)
ReDim gdblMinStepSize(gintNumDesVar)
ReDim gdblCurrDrawVar(gintNumDesVar)

'this call places the value of each variable in its form display box
UPDATE_ALL_DISPLAYS

End Sub

---

Private Sub pnlBaseRotation_DblClick()

   On Error GoTo errortrap5

   Dim this_message As String
   Dim this_title As String
   Dim this_default As String

   this_message = "Enter a new value for the linkage base rotation:"
```

MAIN.FRM - pnlBaseRotation_DblClick

```
    this_title = "Input Decimel Value..."
    this_default = frmMainOptimzation.pnlBaseRotation.Caption

popup5:
    temp = InputBox(this_message, this_title, this_default)

    If temp = "" Then Exit Sub

    temp = CDbl(temp)

    gdblBaseRot = temp * gconDegToRad

    pnlBaseRotation.Caption = gdblBaseRot / gconDegToRad

    'Debug.Print temp

    Exit Sub

errortrap5:

    status = MsgBox("Lets try that again with a number this time (or just hit cancel
     -->!)", 1 + 48 + 0 + 0, "Input Error!")

    If status = 2 Then Exit Sub
    If status = 1 Then GoTo popup5

End Sub
```

```
Private Sub pnlInputArmOffset_DblClick()

    On Error GoTo errortrap3

    Dim this_message As String
    Dim this_title As String
    Dim this_default As String
```

MAIN.FRM - pnlBaseRotation_DblClick

```
    this_message = "Enter a new value for the input arm offset:"
    this_title = "Input Decimel Value..."
    this_default = frmMainOptimzation.pnlInputArmOffset.Caption

popup3:
    temp = InputBox(this_message, this_title, this_default)

    If temp = "" Then Exit Sub

    temp = CDbl(temp)

    gdblInputOff = temp * gconDegToRad

    pnlInputArmOffset.Caption = gdblInputOff / gconDegToRad

    'Debug.Print temp

    Exit Sub

errortrap3:

    status = MsgBox("Lets try that again with a number this time (or just hit cancel
     -->!)", 1 + 48 + 0 + 0, "Input Error!")

    If status = 2 Then Exit Sub
    If status = 1 Then GoTo popup3


End Sub
```

```
Private Sub pnlLengthLink1_DblClick()

    On Error GoTo errortrap6

    Dim this_message As String
    Dim this_title As String
```

```
    Dim this_default As String

    this_message = "Enter a new value for link 1 length:"
    this_title = "Input Decimel Value..."
    this_default = frmMainOptimzation.pnlLengthLink1.Caption

popup6:
    temp = InputBox(this_message, this_title, this_default)

    If temp = "" Then Exit Sub

    temp = CDbl(temp)

    gdblLen1 = temp

    If (gdblLen1 < 0.2) Then gdblLen1 = 0.2
    pnlLengthLink1.Caption = gdblLen1

    'Debug.Print temp

    Exit Sub

errortrap6:

    status = MsgBox("Lets try that again with a number this time (or just hit cancel
     -->!)", 1 + 48 + 0 + 0, "Input Error!")

    If status = 2 Then Exit Sub
    If status = 1 Then GoTo popup6

End Sub
```



```
Private Sub pnlLengthLink2_DblClick()

    On Error GoTo errortrap7

    Dim this_message As String
    Dim this_title As String
    Dim this_default As String

    this_message = "Enter a new value for link 2 length:"
    this_title = "Input Decimel Value..."
    this_default = frmMainOptimzation.pnlLengthLink2.Caption

popup7:
    temp = InputBox(this_message, this_title, this_default)

    If temp = "" Then Exit Sub

    temp = CDbl(temp)

    gdblLen2 = temp

    If (gdblLen2 < 0.2) Then gdblLen2 = 0.2
    pnlLengthLink2.Caption = gdblLen2

    'Debug.Print temp

    Exit Sub

errortrap7:

    status = MsgBox("Lets try that again with a number this time (or just hit cancel
     -->!)", 1 + 48 + 0 + 0, "Input Error!")

    If status = 2 Then Exit Sub
    If status = 1 Then GoTo popup7
```

```
End Sub

---

Private Sub pnlLengthLink3_DblClick()

   On Error GoTo errortrap8

   Dim this_message As String
   Dim this_title As String
   Dim this_default As String

   this_message = "Enter a new value for link 3 length:"
   this_title = "Input Decimel Value..."
   this_default = frmMainOptimzation.pnlLengthLink3.Caption

popup8:
   temp = InputBox(this_message, this_title, this_default)

   If temp = "" Then Exit Sub

   temp = CDbl(temp)

   gdblLen3 = temp

   If (gdblLen3 < 0.2) Then gdblLen3 = 0.2
   pnlLengthLink3.Caption = gdblLen3

   'Debug.Print temp

   Exit Sub

errortrap8:

   status = MsgBox("Lets try that again with a number this time (or just hit cancel
    -->!)", 1 + 48 + 0 + 0, "Input Error!")

   If status = 2 Then Exit Sub
```



```
   If status = 1 Then GoTo popup8

End Sub

---

Private Sub pnlLengthLink4_DblClick()

   On Error GoTo errortrap9

   Dim this_message As String
   Dim this_title As String
   Dim this_default As String

   this_message = "Enter a new value for link 4 length:"
   this_title = "Input Decimel Value..."
   this_default = frmMainOptimzation.pnlLengthLink4.Caption

popup9:
   temp = InputBox(this_message, this_title, this_default)

   If temp = "" Then Exit Sub

   temp = CDbl(temp)

   gdblLen4 = temp

   If (gdblLen4 < 0.2) Then gdblLen4 = 0.2
   pnlLengthLink4.Caption = gdblLen4

   'Debug.Print temp

   Exit Sub

errortrap9:

   status = MsgBox("Lets try that again with a number this time (or just hit cancel
    -->!)", 1 + 48 + 0 + 0, "Input Error!")
```

MAIN.FRM - pnlLengthLink4_DblClick

```
    If status = 2 Then Exit Sub
    If status = 1 Then GoTo popup9

End Sub
```

```
Private Sub pnlLenInputArm_DblClick()

    On Error GoTo errortrap1

    Dim this_message As String
    Dim this_title As String
    Dim this_default As String

    this_message = "Enter a new value for the input arm length:"
    this_title = "Input Decimel Value..."
    this_default = frmMainOptimzation.pnlLenInputArm.Caption

popup1:
    temp = InputBox(this_message, this_title, this_default)

    If temp = "" Then Exit Sub

    temp = CDbl(temp)

    gdblInputLen = temp

    If (gdblInputLen < 0.2) Then gdblInputLen = 0.2

    pnlLenInputArm.Caption = gdblInputLen

    'Debug.Print temp

    Exit Sub

errortrap1:
```

MAIN.FRM - pnlLengthLink4_DblClick

```
    status = MsgBox("Lets try that again with a number this time (or just hit cancel
     -->!)", 1 + 48 + 0 + 0, "Input Error!")

    If status = 2 Then Exit Sub
    If status = 1 Then GoTo popup1

End Sub
```

```
Private Sub pnlLenWeightArm_DblClick()

    On Error GoTo errortrap2

    Dim this_message As String
    Dim this_title As String
    Dim this_default As String

    this_message = "Enter a new value for the weight arm length:"
    this_title = "Input Decimel Value..."
    this_default = frmMainOptimzation.pnlLenWeightArm.Caption

popup2:
    temp = InputBox(this_message, this_title, this_default)

    If temp = "" Then Exit Sub

    temp = CDbl(temp)

    gdblWhtLen = temp

    If (gdblWhtLen < 0.2) Then gdblWhtLen = 0.2

    pnlLenWeightArm.Caption = gdblWhtLen

    'Debug.Print temp
```

```
    Exit Sub

errortrap2:

    status = MsgBox("Lets try that again with a number this time (or just hit cancel
     -->!)", 1 + 48 + 0 + 0, "Input Error!")

    If status = 2 Then Exit Sub
    If status = 1 Then GoTo popup2

End Sub
```

```
Private Sub pnlWeightArmOffset_DblClick()

    On Error GoTo errortrap4

    Dim this_message As String
    Dim this_title As String
    Dim this_default As String

    this_message = "Enter a new value for the weight arm offset:"
    this_title = "Input Decimel Value..."
    this_default = frmMainOptimzation.pnlWeightArmOffset.Caption

popup4:
    temp = InputBox(this_message, this_title, this_default)

    If temp = "" Then Exit Sub

    temp = CDbl(temp)

    gdblWhtOff = temp * gconDegToRad

    pnlWeightArmOffset.Caption = gdblWhtOff / gconDegToRad

    'Debug.Print temp
```

MAIN.FRM - pnlLenWeightArm_DblClick

```
    Exit Sub

errortrap4:

    status = MsgBox("Lets try that again with a number this time (or just hit cancel
     -->!)", 1 + 48 + 0 + 0, "Input Error!")

    If status = 2 Then Exit Sub
    If status = 1 Then GoTo popup4

End Sub
```

```
Private Sub spnBaseRotation_SpinDown()
    DoEvents
    gdblBaseRot = gdblBaseRot - 0.5 * gconDegToRad
    pnlBaseRotation.Caption = gdblBaseRot / gconDegToRad
End Sub
```

```
Private Sub spnBaseRotation_SpinUp()
    DoEvents
    gdblBaseRot = gdblBaseRot + 0.5 * gconDegToRad
    pnlBaseRotation.Caption = gdblBaseRot / gconDegToRad
End Sub
```

```
Private Sub spnInputArmOffset_SpinDown()
    DoEvents
    gdblInputOff = gdblInputOff - 0.5 * gconDegToRad
    pnlInputArmOffset.Caption = gdblInputOff / gconDegToRad
End Sub
```

MAIN.FRM - spnInputArmOffset_SpinUp

```
Private Sub spnInputArmOffset_SpinUp()
   DoEvents
   gdblInputOff = gdblInputOff + 0.5 * gconDegToRad
   pnlInputArmOffset.Caption = gdblInputOff / gconDegToRad
End Sub
```

```
Private Sub spnLengthLink1_SpinDown()
   DoEvents
   gdblLen1 = gdblLen1 - 0.25
   pnlLengthLink1.Caption = gdblLen1
End Sub
```

```
Private Sub spnLengthLink1_SpinUp()
   DoEvents
   gdblLen1 = gdblLen1 + 0.25
   pnlLengthLink1.Caption = gdblLen1
End Sub
```

```
Private Sub spnLengthLink2_SpinDown()
   DoEvents
   gdblLen2 = gdblLen2 - 0.25
   pnlLengthLink2.Caption = gdblLen2
End Sub
```

```
Private Sub spnLengthLink2_SpinUp()
   DoEvents
   gdblLen2 = gdblLen2 + 0.25
   pnlLengthLink2.Caption = gdblLen2
End Sub
```

MAIN.FRM - spnInputArmOffset_SpinUp

```
Private Sub spnLengthLink3_SpinDown()
   DoEvents
   gdblLen3 = gdblLen3 - 0.25
   pnlLengthLink3.Caption = gdblLen3
End Sub
```

```
Private Sub spnLengthLink3_SpinUp()
   DoEvents
   gdblLen3 = gdblLen3 + 0.25
   pnlLengthLink3.Caption = gdblLen3
End Sub
```

```
Private Sub spnLengthLink4_SpinDown()
   DoEvents
   gdblLen4 = gdblLen4 - 0.25
   pnlLengthLink4.Caption = gdblLen4
End Sub
```

```
Private Sub spnLengthLink4_SpinUp()
   DoEvents
   gdblLen4 = gdblLen4 + 0.25
   pnlLengthLink4.Caption = gdblLen4
End Sub
```

```
Private Sub spnLenInputArm_SpinDown()
   DoEvents
   gdblInputLen = gdblInputLen - 0.5
   pnlLenInputArm.Caption = gdblInputLen
End Sub
```

MAIN.FRM - spnLenInputArm_SpinUp

```
Private Sub spnLenInputArm_SpinUp()
   DoEvents
   gdblInputLen = gdblInputLen + 0.5
   pnlLenInputArm.Caption = gdblInputLen
End Sub
```

```
Private Sub spnLenWeightArm_SpinDown()
   DoEvents
   gdblWhtLen = gdblWhtLen - 0.5
   pnlLenWeightArm.Caption = gdblWhtLen
End Sub
```

```
Private Sub spnLenWeightArm_SpinUp()
   DoEvents
   gdblWhtLen = gdblWhtLen + 0.5
   pnlLenWeightArm.Caption = gdblWhtLen
End Sub
```

```
Private Sub spnWeightArmOffset_SpinDown()
   DoEvents
   gdblWhtOff = gdblWhtOff - 0.5 * gconDegToRad
   pnlWeightArmOffset.Caption = gdblWhtOff / gconDegToRad
End Sub
```

```
Private Sub spnWeightArmOffset_SpinUp()
   DoEvents
   gdblWhtOff = gdblWhtOff + 0.5 * gconDegToRad
   pnlWeightArmOffset.Caption = gdblWhtOff / gconDegToRad
End Sub
```

MAIN.FRM - spnLenInputArm_SpinUp

```
Private Sub UPDATE_ALL_DISPLAYS()

'This routine updates the numeric displays on the main
'optimization form whenever changes are detected.

   pnlLenInputArm.Caption = gdblInputLen
   pnlLenWeightArm.Caption = gdblWhtLen
   pnlInputArmOffset.Caption = gdblInputOff / gconDegToRad
   pnlWeightArmOffset.Caption = gdblWhtOff / gconDegToRad
   pnlBaseRotation.Caption = gdblBaseRot / gconDegToRad
   pnlLengthLink1.Caption = gdblLen1
   pnlLengthLink2.Caption = gdblLen2
   pnlLengthLink3.Caption = gdblLen3
   pnlLengthLink4.Caption = gdblLen4

   If gintClosure = 1 Then
      cmdMechClosure.Caption = "Quad = 1"
      cmdMechClosure.ForceUp = True
   Else
      cmdMechClosure.Caption = "Quad = -1"
      cmdMechClosure.ForceDown = True
   End If

   If gintLinkType = 1 Then
      cmdMechType.Caption = "Co-Rotate"
      cmdMechType.ForceUp = True
   Else
      cmdMechType.Caption = "Counter-Rotate"
      cmdMechType.ForceDown = True
   End If
End Sub
```

MAIN.FRM - UPDATE_ALL_DISPLAYS

```
Private Sub Form_Load()

   SystemMenu% = GetSystemMenu(frmDataGrid.hWnd, 0)
   Res% = RemoveMenu(SystemMenu%, 6, MF_BYPOSITION)
   Res% = RemoveMenu(SystemMenu%, 6, MF_BYPOSITION) 'also remove the
    -->separator line

   frmDataGrid.grdStrengthData.ColWidth(0) = 1000
   frmDataGrid.grdStrengthData.ColWidth(1) = 1000

   frmDataGrid.grdStrengthData.Row = 0
   frmDataGrid.grdStrengthData.Col = 0
   frmDataGrid.grdStrengthData.Text = "Theta (deg)"
   frmDataGrid.grdStrengthData.Row = 0
   frmDataGrid.grdStrengthData.Col = 1
   frmDataGrid.grdStrengthData.Text = "Strength (%)"

   frmDataGrid.Width = 2115
   frmDataGrid.Height = 5265
   frmDataGrid.Left = frmLinkageDraw.Left + frmLinkageDraw.Width
   frmDataGrid.Top = 0

End Sub
```

```
Private Sub grdStrengthData_DblClick()

'Debug.Print grdStrengthData.Row, grdStrengthData.Col, grdStrengthData.Text

On Error GoTo errortrap_gridclick

Dim this_message As String
Dim this_title As String
Dim this_default As String

this_message = "Enter a new value for this position:"
```

MAIN.FRM - UPDATE_ALL_DISPLAYS

```
this_title = "Input Decimel Value..."
this_default = Str$(grdStrengthData.Text)

popup_gridclick:
temp = InputBox(this_message, this_title, this_default)

If temp = "" Then Exit Sub

temp = CDbl(temp)

'Debug.Print temp

If grdStrengthData.Col = 0 Then
   gdblCurveBeta(grdStrengthData.Row, 1) = temp
End If

If grdStrengthData.Col = 1 Then
   gdblCurveStrength(grdStrengthData.Row, 1) = temp
End If

'For i = 1 To gintNumDataPoints
   'Debug.Print gdblCurveBeta(i, 1), gdblCurveStrength(i, 1)
'Next i

grdStrengthData.Text = temp

'this next line moved to update_button_click 4/47/95
'If (have_strength_data = 1) Then perform_curve_fit_with_data

Exit Sub

errortrap_gridclick:

status = MsgBox("Lets try that again with a number this time (or just hit cancel!)", 1
 --> + 48 + 0 + 0, "Input Error!")

If status = 2 Then Exit Sub
```

DATA_DIS.FRM - grdStrengthData_DblClick

```
If status = 1 Then GoTo popup_gridclick


End Sub
```

```
Private Sub mnuDataSaveToDisk_Click()

Dim output_filename As String
Dim output_beta As Single, output_strength As Single

On Error GoTo save_strength_cancel_error

   dlgSaveStrengthData.Action = 1

   output_filename = dlgSaveStrengthData.Filename

   Open output_filename For Output As #20

   For i = 1 To (frmDataGrid.grdStrengthData.Rows - 1) Step 1
      frmDataGrid.grdStrengthData.Row = i
      frmDataGrid.grdStrengthData.Col = 0
      output_beta = CSng(frmDataGrid.grdStrengthData.Text)
      frmDataGrid.grdStrengthData.Col = 1
      output_strength = CSng(frmDataGrid.grdStrengthData.Text)
      Write #20, output_beta, output_strength
   Next i

   Close #20
   Exit Sub

save_strength_cancel_error:

Exit Sub

End Sub
```

DATA_DIS.FRM - grdStrengthData_DblClick

```
Private Sub mnuUpdate_Click()

   min_x = FIND_VECTOR_MIN(gdblCurveBeta())
   max_x = FIND_VECTOR_MAX(gdblCurveBeta())
   gdblCurveBetaRange = max_x - min_x

   frmSynthCurves.Caption = "Discrete Data From " + gstrStrengthDataFile
   frmSynthCurves.gphComboPlot.DataReset = 1
   frmSynthCurves.gphComboPlot.DataReset = 8
   frmSynthCurves.gphComboPlot.ThisPoint = 1
   frmSynthCurves.gphComboPlot.NumPoints = 2

   For i = 1 To gintNumDataPoints Step 1
      frmSynthCurves.gphComboPlot.XPosData = gdblCurveBeta(i, 1)
      frmSynthCurves.gphComboPlot.GraphData = gdblCurveStrength(i, 1)
      'frmSynthCurves.gphComboPlot.OverlayGraphData = Value * 100
      frmSynthCurves.gphComboPlot.ThisPoint = frmSynthCurves.gphComboPlot.
       -->ThisPoint + 1
      frmSynthCurves.gphComboPlot.NumPoints = frmSynthCurves.gphComboPlot.
       -->ThisPoint + 1
   Next i
   frmSynthCurves.gphComboPlot.NumPoints = frmSynthCurves.gphComboPlot.
    -->ThisPoint - 1
   frmSynthCurves.gphComboPlot.DrawMode = 2

End Sub
```

COMBOPLO.FRM - Form_Load

```
Private Sub Form_Load()

    SystemMenu% = GetSystemMenu(frmSynthCurves.hWnd, 0)
    Res% = RemoveMenu(SystemMenu%, 6, MF_BYPOSITION)
    Res% = RemoveMenu(SystemMenu%, 6, MF_BYPOSITION) 'also remove the
     -->separator line

    frmSynthCurves.Top = frmLinkageDraw.Top + frmLinkageDraw.Height
    frmSynthCurves.Left = frmSynthTextOutput.Left + frmSynthTextOutput.Width
    frmSynthCurves.Width = frmMDI.Width / 2.5
    frmSynthCurves.Height = frmMDI.Height / 3

End Sub
```

```
Private Sub Form_Resize()

    gphComboPlot.Width = frmSynthCurves.ScaleWidth
    gphComboPlot.Height = frmSynthCurves.ScaleHeight

End Sub
```

```
Private Sub gphComboPlot_HotHit(hitset As Integer, hitpoint As Integer)

    'This procedure is called whenever a mouse click is
    'received on one of the points on the graph.  Note that
    'the hothit property must be enabled for the graph.

    Dim sngHitX As Single, sngHitY As Single, strMessage As String


    'get the information for the hit point.
    gphComboPlot.ThisSet = hitset
    gphComboPlot.ThisPoint = hitpoint
    sngHitX = gphComboPlot.XPosData
```

COMBOPLO.FRM - Form_Load

```
    sngHitY = gphComboPlot.GraphData

    'create a message for the hit saying what the value is.
    strMessage = "You hit (" + Str(sngHitX) + ", " + Str(sngHitY) + ")."

    'popup a messagebox.
    MsgBox strMessage, 0 + 48, "Graph HotHit"

End Sub
```

```
Private Sub mnuCloseWindow_Click()

    frmSynthCurves.Hide

End Sub
```

```
Private Sub mnuCopyPlot_Click()

    frmSynthCurves.gphComboPlot.DrawMode = 4

End Sub
```

```
Private Sub mnuPrintPlot_Click()

    frmSynthCurves.gphComboPlot.DrawMode = 5
    frmSynthCurves.gphComboPlot.DrawMode = 1
    frmSynthCurves.gphComboPlot.DrawMode = 2

End Sub
```

COMBOPLO.FRM - mnuPrintPlot_Click

```
Private Sub Form_Load()

   SystemMenu% = GetSystemMenu(frmLinkageDraw.hWnd, 0)
   Res% = RemoveMenu(SystemMenu%, 6, MF_BYPOSITION)
   Res% = RemoveMenu(SystemMenu%, 6, MF_BYPOSITION) 'also remove the
    -->separator line

   frmLinkageDraw.Width = frmMDI.Width / 3
   frmLinkageDraw.Height = frmMDI.Width / 3
   frmLinkageDraw.Top = 0
   frmLinkageDraw.Left = frmMainOptimzation.Left + frmMainOptimzation.Width

   msngFormZoom = 1#
   frmLinkageDraw.Scale (msngFormZoom * (-ScaleWidth / 2), msngFormZoom * (-
    -->ScaleHeight / 2))-(msngFormZoom * (-ScaleWidth / 2 + ScaleWidth),
    -->msngFormZoom * (-ScaleHeight / 2 + ScaleHeight))

End Sub
```

```
Private Sub mnuZoomIn_Click()

   msngFormZoom = 0.75
   frmLinkageDraw.Scale (msngFormZoom * (-ScaleWidth / 2), msngFormZoom * (-
    -->ScaleHeight / 2))-(msngFormZoom * (-ScaleWidth / 2 + ScaleWidth),
    -->msngFormZoom * (-ScaleHeight / 2 + ScaleHeight))

   'redraw the linkage in the draw window aaccording to the current set of
    -->drawable variables
   Call DRAW_SYNTH_LINKAGE(gdblCurrDrawVar(4), gdblCurrDrawVar(5),
    -->gdblCurrDrawVar(6), gdblCurrDrawVar(7), gdblCurrDrawVar(1),
    -->gintLinkType, gintClosure, gdblInputLen, gdblCurrDrawVar(8),
    -->gdblCurrDrawVar(2), gdblCurrDrawVar(3))

End Sub
```

COMBOPLO.FRM - mnuPrintPlot_Click

```
Private Sub mnuZoomOut_Click()

   msngFormZoom = 1.33
   frmLinkageDraw.Scale (msngFormZoom * (-ScaleWidth / 2), msngFormZoom * (-
    -->ScaleHeight / 2))-(msngFormZoom * (-ScaleWidth / 2 + ScaleWidth),
    -->msngFormZoom * (-ScaleHeight / 2 + ScaleHeight))

   'redraw the linkage in the draw window aaccording to the current set of
    -->drawable variables
   Call DRAW_SYNTH_LINKAGE(gdblCurrDrawVar(4), gdblCurrDrawVar(5),
    -->gdblCurrDrawVar(6), gdblCurrDrawVar(7), gdblCurrDrawVar(1),
    -->gintLinkType, gintClosure, gdblInputLen, gdblCurrDrawVar(8),
    -->gdblCurrDrawVar(2), gdblCurrDrawVar(3))

End Sub
```

TEXTOUTP.FRM - Form_Load

```
Private Sub Form_Load()

    SystemMenu% = GetSystemMenu(frmSynthTextOutput.hWnd, 0)
    Res% = RemoveMenu(SystemMenu%, 6, MF_BYPOSITION)
    Res% = RemoveMenu(SystemMenu%, 6, MF_BYPOSITION) 'also remove the
     -->separator line

    frmSynthTextOutput.Width = frmMDI.Width / 4
    frmSynthTextOutput.Height = frmMDI.Height / 2
    frmSynthTextOutput.Top = frmLinkageDraw.Top + frmLinkageDraw.Height
    frmSynthTextOutput.Left = frmLinkageDraw.Left

End Sub
```

```
Private Sub Form_Resize()

    If WindowState <> 1 And ScaleHeight <> 0 Then
        txtSynthTextOutput.Visible = False
        txtSynthTextOutput.Height = ScaleHeight
        txtSynthTextOutput.Width = ScaleWidth
        txtSynthTextOutput.Visible = True
    End If

End Sub
```

```
Private Sub mnuCopySelectedText_Click()

    Clipboard.SetText frmSynthTextOutput.txtSynthTextOutput.SelText

End Sub
```

TEXTOUTP.FRM - Form_Load

```
Private Sub Form_Load()

    'SystemMenu% = GetSystemMenu(frmDiagram.hWnd, 0)
    'Res% = RemoveMenu(SystemMenu%, 6, MF_BYPOSITION)
    'Res% = RemoveMenu(SystemMenu%, 6, MF_BYPOSITION)'also remove the
     -->separator line

    frmDiagram.Width = 8985
    frmDiagram.Height = 4493

    picDiagram.Width = frmDiagram.Width
    picDiagram.Height = frmDiagram.Height

End Sub
```

```
Private Sub Form_QueryUnload(Cancel As Integer, UnloadMode As Integer)

    Unload frmDiagram

End Sub
```

```
Private Sub Form_Resize()

    tempwidth = frmDiagram.Width
    tempheight = frmDiagram.Height

    form_aspect = tempwidth / tempheight
    Debug.Print "aspect = "; form_aspect

    If form_aspect < 2# Then
        tempwidth = tempheight * 2#
    Else
        tempheight = tempwidth / 2#
    End If
```

DETAILS.FRM - Form_Resize

```
    frmDiagram.Width = tempwidth
    frmDiagram.Height = tempheight

    picDiagram.Width = frmDiagram.Width
    picDiagram.Height = frmDiagram.Height

End Sub
```

```
Private Sub mnuPrintDiagram_Click()

    frmDiagram.PrintForm

End Sub
```

DETAILS.FRM - Form_Resize

```
Private Sub ANALYSIS_LINKAGE_SOLVER(w_offset As Single, l_closure As
 -->Integer, l1 As Single, l2 As Single, l3 As Single, l4 As Single, rot As Single, t2
 -->As Single, t3 As Single, t4 As Single, tphi As Single, return_error As String)

'This procedure is used in Analysis Mode to solve the four-bar linkage
'so that it can be plotted or aniumated.  It returns the values of theta3,
'theta4, and the output angle phi.

    Call POSITION_ANALYSIS(l_closure, l1, l2, l3, l4, rot, t2, t3, t4, return_error)

    If return_error = "No Errors" Then
        tphi = t4 - w_offset
    Else
        tphi = 0
    End If

End Sub
```

```
Private Sub ANIMATION_ROUTINE()

'This routine is called by clicking the animate button in
'analysis mode.  It animates the linkage over its range of
'motion and then returns to the starting angle.

    Dim sngStepSize As Single, sngKCount As Single
    Dim sngCurrentBeta As Single, sngCurrentAngle As Single

    'disable and grey out the spin buttons during animation
    Call DisableControl(spnAnalysisZoom, LIGHTGRAY)
    Call DisableControl(spnAnalysisRotate, LIGHTGRAY)
    Call DisableControl(spnAnalysisScale, LIGHTGRAY)
    Call DisableControl(spnAnalysisSolution, LIGHTGRAY)
    pnlSolutionNumber.Enabled = False

    'disable the menus and toolbar during animation
```

EXT_LINK.FRM - ANIMATION_ROUTINE

```
mnuActionAnimate.Enabled = False
barAnalysisDraw.ButtonEnabled(0) = False
barAnalysisDraw.ButtonEnabled(1) = False
barAnalysisDraw.ButtonEnabled(2) = False

sngStepSize = msngBetaRange / (mintNumAnimSteps * gconDegToRad)

For sngCurrentAngle = 0 To (msngBetaRange / gconDegToRad + sngStepSize)
 -->Step sngStepSize

  DoEvents

  'If (mintLinkType = 1) Then sngCurrentBeta = msngBetaStart + (mintLinkType
  -->* sngCurrentAngle * gconDegToRad)
  'If (mintLinkType = -1) Then sngCurrentBeta = msngBetaStart - (mintLinkType
  -->* sngCurrentAngle * gconDegToRad)
  sngCurrentBeta = msngBetaStart(mintCurrentSolNum) + sngCurrentAngle *
  -->gconDegToRad

  Debug.Print sngCurrentBeta / gconDegToRad

  msngTheta2 = sngCurrentBeta - msngInputOff(mintCurrentSolNum)

  Call ANALYSIS_LINKAGE_SOLVER(msngWhtOff(mintCurrentSolNum),
  -->mintClosure(mintCurrentSolNum), msngLen1(mintCurrentSolNum),
  -->msngLen2(mintCurrentSolNum), msngLen3(mintCurrentSolNum),
  -->msngLen4(mintCurrentSolNum), msngBaseRot(mintCurrentSolNum),
  -->msngThe

  'If (mstrErrorFlag = "Linkage Doesn't Assemble") Then GoTo
  -->bad_linkage_specified

  'Debug.Print "t2,t3,t4,phi1"; msngTheta2 / gconDegToRad, msngTheta3 /
  -->gconDegToRad, msngTheta4 / gconDegToRad, msngPhi / gconDegToRad

  msngX1 = 0
  msngY1 = 0
```

EXT_LINK.FRM - ANIMATION_ROUTINE

```
  msngX4 = msngLen1(mintCurrentSolNum) * Cos(msngBaseRot(
  -->mintCurrentSolNum))
  msngY4 = msngLen1(mintCurrentSolNum) * Sin(msngBaseRot(
  -->mintCurrentSolNum))

  msngX2 = msngLen2(mintCurrentSolNum) * Cos(msngTheta2)
  msngY2 = msngLen2(mintCurrentSolNum) * Sin(msngTheta2)

  msngX3 = msngX4 + msngLen4(mintCurrentSolNum) * Cos(msngTheta4)
  msngY3 = msngY4 + msngLen4(mintCurrentSolNum) * Sin(msngTheta4)

  msngXIn = msngLenInputArm(mintCurrentSolNum) * Cos(sngCurrentBeta)
  msngYIn = msngLenInputArm(mintCurrentSolNum) * Sin(sngCurrentBeta)

  msngXOut = msngX4 + msngLenWhtArm(mintCurrentSolNum) * Cos(msngPhi)
  msngYOut = msngY4 + msngLenWhtArm(mintCurrentSolNum) * Sin(msngPhi)

  If mintSweepAnim = 0 Then frmAnalysisLinkageDraw.picAnalysisDraw.Cls
  DRAW_LINKAGE

Next sngCurrentAngle

'reverse direction now

For sngCurrentAngle = 0 To (msngBetaRange / gconDegToRad + sngStepSize)
 -->Step sngStepSize

  DoEvents

  'If (mintLinkType = 1) Then sngCurrentBeta = msngBetaEnd - (mintLinkType *
  -->sngCurrentAngle * gconDegToRad)
  'If (mintLinkType = -1) Then sngCurrentBeta = msngBetaEnd + (mintLinkType
  -->* sngCurrentAngle * gconDegToRad)
  sngCurrentBeta = msngBetaEnd(mintCurrentSolNum) - sngCurrentAngle *
  -->gconDegToRad
```

EXT_LINK.FRM - ANIMATION_ROUTINE

```
Debug.Print sngCurrentBeta / gconDegToRad

msngTheta2 = sngCurrentBeta - msngInputOff(mintCurrentSolNum)

Call ANALYSIS_LINKAGE_SOLVER(msngWhtOff(mintCurrentSolNum),
 -->mintClosure(mintCurrentSolNum), msngLen1(mintCurrentSolNum),
 -->msngLen2(mintCurrentSolNum), msngLen3(mintCurrentSolNum),
 -->msngLen4(mintCurrentSolNum), msngBaseRot(mintCurrentSolNum),
 -->msngThe

'If (mstrErrorFlag = "Linkage Doesn't Assemble") Then GoTo
 -->bad_linkage_specified

'Debug.Print "t2,t3,t4,phi1"; msngTheta2 / gconDegToRad, msngTheta3 /
 -->gconDegToRad, msngTheta4 / gconDegToRad, msngPhi / gconDegToRad

msngX1 = 0
msngY1 = 0

msngX4 = msngLen1(mintCurrentSolNum) * Cos(msngBaseRot(
 -->mintCurrentSolNum))
msngY4 = msngLen1(mintCurrentSolNum) * Sin(msngBaseRot(
 -->mintCurrentSolNum))

msngX2 = msngLen2(mintCurrentSolNum) * Cos(msngTheta2)
msngY2 = msngLen2(mintCurrentSolNum) * Sin(msngTheta2)

msngX3 = msngX4 + msngLen4(mintCurrentSolNum) * Cos(msngTheta4)
msngY3 = msngY4 + msngLen4(mintCurrentSolNum) * Sin(msngTheta4)

msngXIn = msngLenInputArm(mintCurrentSolNum) * Cos(sngCurrentBeta)
msngYIn = msngLenInputArm(mintCurrentSolNum) * Sin(sngCurrentBeta)

msngXOut = msngX4 + msngLenWhtArm(mintCurrentSolNum) * Cos(msngPhi)
msngYOut = msngY4 + msngLenWhtArm(mintCurrentSolNum) * Sin(msngPhi)

If mintSweepAnim = 0 Then frmAnalysisLinkageDraw.picAnalysisDraw.Cls
```

EXT_LINK.FRM - ANIMATION_ROUTINE

```
    DRAW_LINKAGE

Next sngCurrentAngle

're-enable the spin buttons after animation
Call EnableControl(spnAnalysisZoom)
Call EnableControl(spnAnalysisRotate)
Call EnableControl(spnAnalysisScale)
Call EnableControl(spnAnalysisSolution)
pnlSolutionNumber.Enabled = True

're-enable the menus and toolbar buttons
mnuActionAnimate.Enabled = True
barAnalysisDraw.ButtonEnabled(0) = True
barAnalysisDraw.ButtonEnabled(1) = True
barAnalysisDraw.ButtonEnabled(2) = True

End Sub

---

Private Sub ASSEMBLE_STRING()

'This procedure assembles one long textg string that is used
'to display all relevant information about the current linkage
'in analysis mode.  The text string is displayed in the text
'box on frmAnalysisTextOutput.

frmAnalysisTextOutput.txtAnalysisOutput.Text = ""
frmAnalysisTextOutput.txtAnalysisOutput.Text = frmAnalysisTextOutput.
 -->txtAnalysisOutput.Text + "Input Arm Start Angle = " + Format(msngBetaStart(
 -->mintCurrentSolNum) / gconDegToRad, "####.###") + gstrNewLine
frmAnalysisTextOutput.txtAnalysisOutput.Text = frmAnalysisTextOutput.
 -->txtAnalysisOutput.Text + "Input Arm End Angle = " + Format(msngBetaEnd(
 -->mintCurrentSolNum) / gconDegToRad, "####.###") + gstrNewLine
If mintLinkType(mintCurrentSolNum) = 1 Then
   frmAnalysisTextOutput.txtAnalysisOutput.Text = frmAnalysisTextOutput.
    -->txtAnalysisOutput.Text + "Linkage Type = Co-rotating" + gstrNewLine
```

```
ElseIf mintLinkType(mintCurrentSolNum) = -1 Then
    frmAnalysisTextOutput.txtAnalysisOutput.Text = frmAnalysisTextOutput.
     -->txtAnalysisOutput.Text + "Linkage Type = Counter-rotating" +
     -->gstrNewLine
End If
frmAnalysisTextOutput.txtAnalysisOutput.Text = frmAnalysisTextOutput.
 -->txtAnalysisOutput.Text + "Linkage Closure = " + Str(mintClosure(
 -->mintCurrentSolNum)) + gstrNewLine + gstrNewLine
frmAnalysisTextOutput.txtAnalysisOutput.Text = frmAnalysisTextOutput.
 -->txtAnalysisOutput.Text + "Input Arm Length = " + Format(msngLenInputArm(
 -->mintCurrentSolNum), "####.###") + gstrNewLine
frmAnalysisTextOutput.txtAnalysisOutput.Text = frmAnalysisTextOutput.
 -->txtAnalysisOutput.Text + "Weight Arm Length = " + Format(msngLenWhtArm
 -->(mintCurrentSolNum), "####.###") + gstrNewLine
frmAnalysisTextOutput.txtAnalysisOutput.Text = frmAnalysisTextOutput.
 -->txtAnalysisOutput.Text + "Input Arm Offset = " + Format(msngInputOff(
 -->mintCurrentSolNum) / gconDegToRad, "####.###") + gstrNewLine
frmAnalysisTextOutput.txtAnalysisOutput.Text = frmAnalysisTextOutput.
 -->txtAnalysisOutput.Text + "Weight Arm Offset = " + Format(msngWhtOff(
 -->mintCurrentSolNum) / gconDegToRad, "####.###") + gstrNewLine
frmAnalysisTextOutput.txtAnalysisOutput.Text = frmAnalysisTextOutput.
 -->txtAnalysisOutput.Text + "Linkage Base Rotation = " + Format(msngBaseRot(
 -->mintCurrentSolNum) / gconDegToRad, "####.###") + gstrNewLine +
 -->gstrNewLine
frmAnalysisTextOutput.txtAnalysisOutput.Text = frmAnalysisTextOutput.
 -->txtAnalysisOutput.Text + "Ground Link Length (L1) = " + Format(msngLen1(
 -->mintCurrentSolNum), "####.###") + gstrNewLine
frmAnalysisTextOutput.txtAnalysisOutput.Text = frmAnalysisTextOutput.
 -->txtAnalysisOutput.Text + "Input Link Length (L2) = " + Format(msngLen2(
 -->mintCurrentSolNum), "####.###") + gstrNewLine
frmAnalysisTextOutput.txtAnalysisOutput.Text = frmAnalysisTextOutput.
 -->txtAnalysisOutput.Text + "Coupler Link Length (L3) = " + Format(msngLen3(
 -->mintCurrentSolNum), "####.###") + gstrNewLine
frmAnalysisTextOutput.txtAnalysisOutput.Text = frmAnalysisTextOutput.
 -->txtAnalysisOutput.Text + "Output Link Length (L4) = " + Format(msngLen4(
 -->mintCurrentSolNum), "####.###") + gstrNewLine + gstrNewLine
frmAnalysisTextOutput.txtAnalysisOutput.Text = frmAnalysisTextOutput.
```

EXT_LINK.FRM - ASSEMBLE_STRING

```
 -->txtAnalysisOutput.Text + "Weight Arm Start Angle = " + Format(
 -->msngOutputStartAngle(mintCurrentSolNum), "####.###") + gstrNewLine
frmAnalysisTextOutput.txtAnalysisOutput.Text = frmAnalysisTextOutput.
 -->txtAnalysisOutput.Text + "Weight Arm End Angle = " + Format(
 -->msngOutputEndAngle(mintCurrentSolNum), "####.###") + gstrNewLine
frmAnalysisTextOutput.txtAnalysisOutput.Text = frmAnalysisTextOutput.
 -->txtAnalysisOutput.Text + "Output Range of Motion = " + Format(
 -->msngOutputRange(mintCurrentSolNum), "####.###") + gstrNewLine
frmAnalysisTextOutput.txtAnalysisOutput.Text = frmAnalysisTextOutput.
 -->txtAnalysisOutput.Text + "Solution Error = " + Str(msngSolutionError(
 -->mintCurrentSolNum))

End Sub

---

Private Sub barAnalysisDraw_Click(button As Integer, Group As Integer, State As
 -->Integer)

    Select Case button

    Case 0
        OPEN_SOLUTIONS_FILE
    Case 1
        PRINT_LINKAGE
    Case 2
        ANIMATION_ROUTINE
    End Select

End Sub

---

Private Sub COMPUTE_RANGE_OF_MOTION()

'This routine computes the range of motion of the output, phi.
'This information is displayed along with other pertinent linkage
'info in the output text box.
```

EXT_LINK.FRM - COMPUTE_RANGE_OF_MOTION

```
'sets the input angle to the first point
msngTheta2 = msngBetaStart(mintCurrentSolNum) - msngInputOff(
 -->mintCurrentSolNum)
Call ANALYSIS_LINKAGE_SOLVER(msngWhtOff(mintCurrentSolNum),
 -->mintClosure(mintCurrentSolNum), msngLen1(mintCurrentSolNum), msngLen2
 -->(mintCurrentSolNum), msngLen3(mintCurrentSolNum), msngLen4(
 -->mintCurrentSolNum), msngBaseRot(mintCurrentSolNum), msngTheta2,
msngOutputStartAngle(mintCurrentSolNum) = msngPhi / gconDegToRad

Do Until msngOutputStartAngle(mintCurrentSolNum) > -180#
   msngOutputStartAngle(mintCurrentSolNum) = msngOutputStartAngle(
    -->mintCurrentSolNum) + 360#
Loop

Do Until msngOutputStartAngle(mintCurrentSolNum) < 180#
   msngOutputStartAngle(mintCurrentSolNum) = msngOutputStartAngle(
    -->mintCurrentSolNum) - 360#
Loop

'sets the input angle to the last point
msngTheta2 = msngBetaEnd(mintCurrentSolNum) - msngInputOff(
 -->mintCurrentSolNum)
Call ANALYSIS_LINKAGE_SOLVER(msngWhtOff(mintCurrentSolNum),
 -->mintClosure(mintCurrentSolNum), msngLen1(mintCurrentSolNum), msngLen2
 -->(mintCurrentSolNum), msngLen3(mintCurrentSolNum), msngLen4(
 -->mintCurrentSolNum), msngBaseRot(mintCurrentSolNum), msngTheta2,
msngOutputEndAngle(mintCurrentSolNum) = msngPhi / gconDegToRad

Do Until msngOutputEndAngle(mintCurrentSolNum) > -180#
   msngOutputEndAngle(mintCurrentSolNum) = msngOutputEndAngle(
    -->mintCurrentSolNum) + 360#
Loop

Do Until msngOutputEndAngle(mintCurrentSolNum) < 180#
   msngOutputEndAngle(mintCurrentSolNum) = msngOutputEndAngle(
    -->mintCurrentSolNum) - 360#
Loop
```

EXT_LINK.FRM - COMPUTE_RANGE_OF_MOTION

```
  'get the range
  msngOutputRange(mintCurrentSolNum) = Abs(msngOutputEndAngle(
   -->mintCurrentSolNum) - msngOutputStartAngle(mintCurrentSolNum))

End Sub
```

```
Private Sub DRAW_LINKAGE()

'This procedure uses previously computed linkage info to draw the current
'linkage in the draw window at the current input angle beta.

  frmAnalysisLinkageDraw.picAnalysisDraw.DrawWidth = 2
  frmAnalysisLinkageDraw.picAnalysisDraw.DrawStyle = 0

  frmAnalysisLinkageDraw.picAnalysisDraw.FillStyle = SOLID
  frmAnalysisLinkageDraw.picAnalysisDraw.ForeColor = BLACK
  frmAnalysisLinkageDraw.picAnalysisDraw.FillColor = MEDIUMGRAY
  'frmAnalysisLinkageDraw.picAnalysisDraw.Circle (msngXOut, msngYOut),
   -->msngLenWhtArm(mintCurrentSolNum) / 5#
  frmAnalysisLinkageDraw.picAnalysisDraw.Circle (msngXOut, msngYOut), 8#

  frmAnalysisLinkageDraw.picAnalysisDraw.ForeColor = gconInputArmColor
  frmAnalysisLinkageDraw.picAnalysisDraw.Line (msngX1, msngY1)-(msngX2,
   -->msngY2)
  frmAnalysisLinkageDraw.picAnalysisDraw.Line (msngX1, msngY1)-(msngXIn,
   -->msngYIn)

  frmAnalysisLinkageDraw.picAnalysisDraw.ForeColor = gconWhtArmColor
  frmAnalysisLinkageDraw.picAnalysisDraw.Line (msngX3, msngY3)-(msngX4,
   -->msngY4)
  frmAnalysisLinkageDraw.picAnalysisDraw.Line (msngX4, msngY4)-(msngXOut
   -->, msngYOut)

  frmAnalysisLinkageDraw.picAnalysisDraw.ForeColor = gconCouplerColor
  frmAnalysisLinkageDraw.picAnalysisDraw.Line (msngX2, msngY2)-(msngX3,
```

```
 -->msngY3)

 frmAnalysisLinkageDraw.picAnalysisDraw.ForeColor = BLACK
 frmAnalysisLinkageDraw.picAnalysisDraw.DrawWidth = 3
 frmAnalysisLinkageDraw.picAnalysisDraw.Line (msngX4, msngY4)-(msngX1,
  -->msngY1)

 frmAnalysisLinkageDraw.picAnalysisDraw.DrawWidth = 1
 frmAnalysisLinkageDraw.picAnalysisDraw.DrawStyle = 2
 frmAnalysisLinkageDraw.picAnalysisDraw.ForeColor = DARKPURPLE
 frmAnalysisLinkageDraw.picAnalysisDraw.Line (msngX2, msngY2)-(msngXIn,
  -->msngYIn)
 frmAnalysisLinkageDraw.picAnalysisDraw.Line (msngX3, msngY3)-(msngXOut
  -->, msngYOut)

 frmAnalysisLinkageDraw.picAnalysisDraw.Refresh

End Sub

-------------------------------------------------------------------------

Private Sub Form_Load()

 SystemMenu% = GetSystemMenu(frmAnalysisLinkageDraw.hWnd, 0)
 Res% = RemoveMenu(SystemMenu%, 6, MF_BYPOSITION)
 Res% = RemoveMenu(SystemMenu%, 6, MF_BYPOSITION) 'also remove the
  -->separator line

 frmAnalysisLinkageDraw.Caption = "Linkage Analysis [DRAW]"
 frmAnalysisLinkageDraw.Top = 0
 frmAnalysisLinkageDraw.Left = 0
 frmAnalysisLinkageDraw.Width = 6600
 frmAnalysisLinkageDraw.Height = 7665

 msngFormPanLR = 0
 msngFormPanUD = 0
 msngFormZoom = 40
```

EXT_LINK.FRM - DRAW_LINKAGE

```
 picAnalysisDraw.Height = frmAnalysisLinkageDraw.ScaleHeight -
  -->barAnalysisDraw.Height
 picAnalysisDraw.Scale (-msngFormZoom - msngFormPanLR, msngFormZoom -
  -->msngFormPanUD)-(msngFormZoom - msngFormPanLR, -msngFormZoom -
  -->msngFormPanUD)

 'set up some required defaults for the analysis
 mintCurrentSolNum = 0
 mintNumPlotPoints = 25
 mintNumAnimSteps = 15
 mintAlwaysUpdate = 1
 mintSweepAnim = 0

 'flags that an overlay graph is not used on startup
 mintUseOverlay = 0
 'turn off the strength curve plot overlay option in case it is on
 frmAnalysisCurves.gphAnalysisStrengthPlot.OverlayGraph = 0

 'disable and gray out some buttons on startup
 Call DisableControl(spnAnalysisZoom, LIGHTGRAY)
 Call DisableControl(spnAnalysisRotate, LIGHTGRAY)
 Call DisableControl(spnAnalysisScale, LIGHTGRAY)
 Call DisableControl(spnAnalysisSolution, LIGHTGRAY)
 pnlSolutionNumber.Enabled = False

'note, a default linkage is no longer used because the array size cannot be
 -->redimensioned


'========== some default input parameters of a physical external linkage, the
 -->military press ================

  'msngBetaStart(mintCurrentSolNum) = 186.6 * gconDegToRad
  'msngBetaEnd(mintCurrentSolNum) = 139.51 * gconDegToRad
  'mintLinkType(mintCurrentSolNum) = -1
  'mintClosure(mintCurrentSolNum) = 1
  'msngLenInputArm(mintCurrentSolNum) = 29.126
```

```
    'msngLenWhtArm(mintCurrentSolNum) = 19.986          'this is the military press
     -->machine
    'msngInputOff(mintCurrentSolNum) = 320.93 * gconDegToRad
    'msngWhtOff(mintCurrentSolNum) = 11.85 * gconDegToRad
    'msngBaseRot(mintCurrentSolNum) = -124.02 * gconDegToRad
    'msngLen1(mintCurrentSolNum) = 34.105
    'msngLen2(mintCurrentSolNum) = 11.765
    'msngLen3(mintCurrentSolNum) = 21.307
    'msngLen4(mintCurrentSolNum) = 10.346

    'ASSEMBLE_STRING
    'SOLVE_AND_PLOT_LINKAGE

End Sub
```

```
Private Sub Form_Resize()

'on resize event, we need to stretch the drawing_area with the window
 -->boundaries,
'but since the picture box is already attached horizontally, we need only control
'the height.

    picAnalysisDraw.Height = frmAnalysisLinkageDraw.ScaleHeight -
     -->barAnalysisDraw.Height
    'picAnalysisDraw.Scale (-msngFormZoom - msngFormPanLR, msngFormZoom
     -->- msngFormPanUD)-(msngFormZoom - msngFormPanLR, -msngFormZoom -
     -->msngFormPanUD)

End Sub
```

EXT_LINK.FRM - Form_Load

```
Private Sub mnuActionAnimate_Click()

    ANIMATION_ROUTINE

End Sub
```

```
Private Sub mnuActionOverlayCurve_Click()

    OVERLAY_STRENGTH_CURVE
    frmAnalysisLinkageDraw.SetFocus
    SOLVE_AND_PLOT_LINKAGE

End Sub
```

```
Private Sub mnuActionRecomputeCurve_Click()

    SOLVE_AND_PLOT_LINKAGE
    frmAnalysisLinkageDraw.SetFocus

End Sub
```

```
Private Sub mnuActionRemoveOverlay_Click()

    'flag that an overlay will no longer be used
    mintUseOverlay = 0
    'turn off the strength curve plot overlay option
    frmAnalysisCurves.gphAnalysisStrengthPlot.OverlayGraph = 0
    frmAnalysisLinkageDraw.mnuActionOverlayCurve.Enabled = True
    frmAnalysisLinkageDraw.mnuActionRemoveOverlay.Enabled = False
    frmAnalysisLinkageDraw.SetFocus
    SOLVE_AND_PLOT_LINKAGE

End Sub
```

```
Private Sub mnuCopyDrawing_Click()

   frmAnalysisLinkageDraw.MousePointer = 11
   Clipboard.SetData frmAnalysisLinkageDraw.Image, 2
   frmAnalysisLinkageDraw.MousePointer = 0

End Sub
```

```
Private Sub mnuFileCloseAnalysis_Click()

   If frmMainOptimzation.WindowState = 1 Then frmMainOptimzation.WindowState
    -->= 0
   If frmLinkageDraw.WindowState = 1 Then frmLinkageDraw.WindowState = 0
   If frmSynthTextOutput.WindowState = 1 Then frmSynthTextOutput.
    -->WindowState = 0
   If frmDataGrid.WindowState = 1 Then frmDataGrid.WindowState = 0
   If frmSynthCurves.WindowState = 1 Then frmSynthCurves.WindowState = 0
   frmMainOptimzation.SetFocus
   frmMDI.Caption = "Nautilus Force Optimization [OPTIMIZE MODE]"

   Unload frmAnalysisLinkageDraw
   Unload frmAnalysisCurves
   Unload frmAnalysisTextOutput
   Unload frmErrorTable

End Sub
```

```
Private Sub mnuFileOpenSolutions_Click()

   OPEN_SOLUTIONS_FILE

End Sub
```

EXT_LINK.FRM - mnuActionRemoveOverlay_Click

```
Private Sub mnuFilePrintDrawing_Click()

   PRINT_LINKAGE

End Sub
```

```
Private Sub mnuOptionsSteps_Click()

On Error GoTo mnuOptionsSteps_errortrap

Dim this_message As String
Dim this_title As String
Dim this_default As String

this_message = "Enter the number of steps to use in animating solution linkages (
 -->one-direction):"
this_title = "Input Integer Value..."
this_default = Str(mintNumAnimSteps)

popup_mnuOptionsSteps:
temp = InputBox(this_message, this_title, this_default)

If temp = "" Then Exit Sub

temp = CInt(temp)

mintNumAnimSteps = temp

'Debug.Print temp

Exit Sub

mnuOptionsSteps_errortrap:

status = MsgBox("Lets try that again with a number this time (or just hit cancel!)", 1
```

```
 --> + 48 + 0 + 0, "Input Error!")

If status = 2 Then Exit Sub
If status = 1 Then GoTo popup_mnuOptionsSteps

End Sub

---

Private Sub mnuOptionsSweep_Click()

   mnuOptionsSweep.CHECKED = Not (mnuOptionsSweep.CHECKED)

   If (mnuOptionsSweep.CHECKED = True) Then
      mintSweepAnim = 1
   Else
      mintSweepAnim = 0
   End If

End Sub

---

Private Sub mnuParamBaseRotation_Click()

   On Error GoTo mnuParamBaseRotation_errortrap

   Dim this_message As String
   Dim this_title As String
   Dim this_default As String

   this_message = "Enter a new value for the linkage rotation (CCW +):"
   this_title = "Linkage Rotation..."
   this_default = Str$(CSng(msngBaseRot(mintCurrentSolNum) / gconDegToRad))

popup_mnuParamBaseRotation:
   temp = InputBox(this_message, this_title, this_default)

   If temp = "" Then Exit Sub
```

EXT_LINK.FRM - mnuOptionsSteps_Click

```
   msngBaseRot(mintCurrentSolNum) = temp * gconDegToRad

   'SOLVE_AND_PLOT_LINKAGE

   'Debug.Print temp

   ASSEMBLE_STRING

   Exit Sub

mnuParamBaseRotation_errortrap:

   status = MsgBox("Lets try that again with a number this time (or just hit cancel
    -->!)", 1 + 48 + 0 + 0, "Input Error!")

   If status = 2 Then Exit Sub
   If status = 1 Then GoTo popup_mnuParamBaseRotation

End Sub

---

Private Sub mnuParamBetaEnd_Click()

   On Error GoTo mnuParamBetaEnd_errortrap

   Dim this_message As String
   Dim this_title As String
   Dim this_default As String

   this_message = "Enter the ending value for the input rotation beta:"
   this_title = "Input Rotation End Angle..."
   this_default = Str$(CSng(msngBetaEnd(mintCurrentSolNum) / gconDegToRad))

popup_mnuParamBetaEnd:
   temp = InputBox(this_message, this_title, this_default)
```

```
    If temp = "" Then Exit Sub

    msngBetaEnd(mintCurrentSolNum) = temp * gconDegToRad

    'SOLVE_AND_PLOT_LINKAGE

    'Debug.Print temp

    ASSEMBLE_STRING

    Exit Sub

mnuParamBetaEnd_errortrap:

    status = MsgBox("Lets try that again with a number this time (or just hit cancel
     -->!)", 1 + 48 + 0 + 0, "Input Error!")

    If status = 2 Then Exit Sub
    If status = 1 Then GoTo popup_mnuParamBetaEnd


End Sub
```

```
Private Sub mnuParamBetaStart_Click()

    On Error GoTo mnuParamBetaStart_errortrap

    Dim this_message As String
    Dim this_title As String
    Dim this_default As String

    this_message = "Enter the starting value for the input rotation beta:"
    this_title = "Input Rotation Start Angle..."
    this_default = Str$(CSng(msngBetaStart(mintCurrentSolNum) / gconDegToRad))

popup_mnuParamBetaStart:
```



```
    temp = InputBox(this_message, this_title, this_default)

    If temp = "" Then Exit Sub

    msngBetaStart(mintCurrentSolNum) = temp * gconDegToRad

    'SOLVE_AND_PLOT_LINKAGE

    'Debug.Print temp

    ASSEMBLE_STRING

    Exit Sub

mnuParamBetaStart_errortrap:

    status = MsgBox("Lets try that again with a number this time (or just hit cancel
     -->!)", 1 + 48 + 0 + 0, "Input Error!")

    If status = 2 Then Exit Sub
    If status = 1 Then GoTo popup_mnuParamBetaStart

End Sub
```

```
Private Sub mnuParamInputLength_Click()

    On Error GoTo mnuParamInputLength_errortrap

    Dim this_message As String
    Dim this_title As String
    Dim this_default As String

    this_message = "Enter a new value for the input arm length (l_in):"
    this_title = "Input Arm Length..."
    this_default = Str$(msngLenInputArm(mintCurrentSolNum))
```

```
popup_mnuParamInputLength:
   temp = InputBox(this_message, this_title, this_default)

   If temp = "" Then Exit Sub

   msngLenInputArm(mintCurrentSolNum) = temp
   If (msngLenInputArm(mintCurrentSolNum) <= 0) Then msngLenInputArm(
    -->mintCurrentSolNum) = 0.1

   'SOLVE_AND_PLOT_LINKAGE

   'Debug.Print temp

   ASSEMBLE_STRING

   Exit Sub

mnuParamInputLength_errortrap:

   status = MsgBox("Lets try that again with a number this time (or just hit cancel
    -->!)", 1 + 48 + 0 + 0, "Input Error!")

   If status = 2 Then Exit Sub
   If status = 1 Then GoTo popup_mnuParamInputLength

End Sub
```

```
Private Sub mnuParamInputOffset_Click()

   On Error GoTo mnuParamInputOffset_errortrap

   Dim this_message As String
   Dim this_title As String
   Dim this_default As String

   this_message = "Enter a new value for the input arm offset:"
```



```
   this_title = "Input Arm Offset..."
   this_default = Str$(CSng(msngInputOff(mintCurrentSolNum) / gconDegToRad))

popup_mnuParamInputOffset:
   temp = InputBox(this_message, this_title, this_default)

   If temp = "" Then Exit Sub

   msngInputOff(mintCurrentSolNum) = temp * gconDegToRad

   'SOLVE_AND_PLOT_LINKAGE

   'Debug.Print temp

   ASSEMBLE_STRING

   Exit Sub

mnuParamInputOffset_errortrap:

   status = MsgBox("Lets try that again with a number this time (or just hit cancel
    -->!)", 1 + 48 + 0 + 0, "Input Error!")

   If status = 2 Then Exit Sub
   If status = 1 Then GoTo popup_mnuParamInputOffset

End Sub
```

```
Private Sub mnuParamLength1_Click()

   On Error GoTo mnuParamLength1_errortrap

   Dim this_message As String
   Dim this_title As String
   Dim this_default As String
```

EXT_LINK.FRM - mnuParamLength1_Click

```
    this_message = "Enter a new value for the length of link 1 (L1):"
    this_title = "Link 1 Length..."
    this_default = Str$(msngLen1(mintCurrentSolNum))

popup_mnuParamLength1:
    temp = InputBox(this_message, this_title, this_default)

    If temp = "" Then Exit Sub

    msngLen1(mintCurrentSolNum) = temp
    If (msngLen1(mintCurrentSolNum) <= 0) Then msngLen1(mintCurrentSolNum) =
     -->0.1

    'SOLVE_AND_PLOT_LINKAGE

    'Debug.Print temp

    ASSEMBLE_STRING

    Exit Sub

mnuParamLength1_errortrap:

    status = MsgBox("Lets try that again with a number this time (or just hit cancel
     -->!)", 1 + 48 + 0 + 0, "Input Error!")

    If status = 2 Then Exit Sub
    If status = 1 Then GoTo popup_mnuParamLength1

End Sub
```

EXT_LINK.FRM - mnuParamLength1_Click

```
Private Sub mnuParamLength2_Click()

    On Error GoTo mnuParamLength2_errortrap

    Dim this_message As String
    Dim this_title As String
    Dim this_default As String

    this_message = "Enter a new value for the length of link 2 (L2):"
    this_title = "Link 2 Length..."
    this_default = Str$(msngLen2(mintCurrentSolNum))

popup_mnuParamLength2:
    temp = InputBox(this_message, this_title, this_default)

    If temp = "" Then Exit Sub

    msngLen2(mintCurrentSolNum) = temp
    If (msngLen2(mintCurrentSolNum) <= 0) Then msngLen2(mintCurrentSolNum) =
     -->0.1

    'SOLVE_AND_PLOT_LINKAGE

    'Debug.Print temp

    ASSEMBLE_STRING

    Exit Sub

mnuParamLength2_errortrap:

    status = MsgBox("Lets try that again with a number this time (or just hit cancel
     -->!)", 1 + 48 + 0 + 0, "Input Error!")

    If status = 2 Then Exit Sub
    If status = 1 Then GoTo popup_mnuParamLength2
```

```
End Sub

Private Sub mnuParamLength3_Click()

    On Error GoTo mnuParamLength3_errortrap

    Dim this_message As String
    Dim this_title As String
    Dim this_default As String

    this_message = "Enter a new value for the length of link 3 (L3):"
    this_title = "Link 3 Length..."
    this_default = Str$(msngLen3(mintCurrentSolNum))

popup_mnuParamLength3:
    temp = InputBox(this_message, this_title, this_default)

    If temp = "" Then Exit Sub

    msngLen3(mintCurrentSolNum) = temp
    If (msngLen3(mintCurrentSolNum) <= 0) Then msngLen3(mintCurrentSolNum) =
     -->0.1

    'SOLVE_AND_PLOT_LINKAGE

    'Debug.Print temp

    ASSEMBLE_STRING

    Exit Sub

mnuParamLength3_errortrap:

    status = MsgBox("Lets try that again with a number this time (or just hit cancel
     -->!)", 1 + 48 + 0 + 0, "Input Error!")
```

EXT_LINK.FRM - mnuParamLength2_Click

```
    If status = 2 Then Exit Sub
    If status = 1 Then GoTo popup_mnuParamLength3


End Sub

Private Sub mnuParamLength4_Click()

    On Error GoTo mnuParamLength4_errortrap

    Dim this_message As String
    Dim this_title As String
    Dim this_default As String

    this_message = "Enter a new value for the length of link 4 (L4):"
    this_title = "Link 4 Length..."
    this_default = Str$(msngLen4(mintCurrentSolNum))

popup_mnuParamLength4:
    temp = InputBox(this_message, this_title, this_default)

    If temp = "" Then Exit Sub

    msngLen4(mintCurrentSolNum) = temp
    If (msngLen4(mintCurrentSolNum) <= 0) Then msngLen4(mintCurrentSolNum) =
     -->0.1

    'SOLVE_AND_PLOT_LINKAGE

    'Debug.Print temp

    ASSEMBLE_STRING

    Exit Sub
```

```
mnuParamLength4_errortrap:

   status = MsgBox("Lets try that again with a number this time (or just hit cancel
    -->!)", 1 + 48 + 0 + 0, "Input Error!")

   If status = 2 Then Exit Sub
   If status = 1 Then GoTo popup_mnuParamLength4

End Sub
```

```
Private Sub mnuParamLinkageClosure_Click()

   On Error GoTo mnuParamLinkageClosure_errortrap

   Dim this_message As String
   Dim this_title As String
   Dim this_default As String

   this_message = "Enter the linkage closure (1 or -1 only):"
   this_title = "Linkage Closure..."
   this_default = Str$(mintClosure(mintCurrentSolNum))

popup_mnuParamLinkageClosure:
   temp = InputBox(this_message, this_title, this_default)

   If temp = "" Then Exit Sub

   'Debug.Print temp

   mintClosure(mintCurrentSolNum) = CInt(temp)
   If (mintClosure(mintCurrentSolNum) = 1) Or (mintClosure(mintCurrentSolNum) = -
    -->1) Then
      'SOLVE_AND_PLOT_LINKAGE
   Else
      GoTo mnuParamLinkageClosure_errortrap
   End If
```

EXT_LINK.FRM - mnuParamLength4_Click

```
   ASSEMBLE_STRING

   Exit Sub

mnuParamLinkageClosure_errortrap:

   status = MsgBox("Lets try that again with a number this time (or just hit cancel
    -->!)", 1 + 48 + 0 + 0, "Input Error!")

   If status = 2 Then Exit Sub
   If status = 1 Then GoTo popup_mnuParamLinkageClosure

End Sub
```

```
Private Sub mnuParamLinkageType_Click()

   On Error GoTo mnuParamLinkageType_errortrap

   Dim this_message As String
   Dim this_title As String
   Dim this_default As String

   this_message = "Enter a linkage type (1 for co-rotating, -1 for counter):"
   this_title = "Linkage Type..."
   this_default = Str$(mintLinkType(mintCurrentSolNum))

popup_mnuParamLinkageType:
   temp = InputBox(this_message, this_title, this_default)

   If temp = "" Then Exit Sub

   mintLinkType(mintCurrentSolNum) = CInt(temp)
   If (mintLinkType(mintCurrentSolNum) = 1) Or (mintLinkType(mintCurrentSolNum)
    -->= -1) Then
      'SOLVE_AND_PLOT_LINKAGE
```

```
    Else
        GoTo mnuParamLinkageType_errortrap
    End If

    'Debug.Print temp

    ASSEMBLE_STRING

    Exit Sub

mnuParamLinkageType_errortrap:

    status = MsgBox("Lets try that again with a number this time (or just hit cancel
     -->!)", 1 + 48 + 0 + 0, "Input Error!")

    If status = 2 Then Exit Sub
    If status = 1 Then GoTo popup_mnuParamLinkageType

End Sub
```

```
Private Sub mnuParamWeightLength_Click()

    On Error GoTo mnuParamWeightLength_errortrap

    Dim this_message As String
    Dim this_title As String
    Dim this_default As String

    this_message = "Enter a new value for the weight arm length (l_w):"
    this_title = "Weight Arm Length..."
    this_default = Str$(msngLenWhtArm(mintCurrentSolNum))

popup_mnuParamWeightLength:
    temp = InputBox(this_message, this_title, this_default)

    If temp = "" Then Exit Sub
```

EXT_LINK.FRM - mnuParamLinkageType_Click

```
    msngLenWhtArm(mintCurrentSolNum) = temp
    If (msngLenWhtArm(mintCurrentSolNum) <= 0) Then msngLenWhtArm(
     -->mintCurrentSolNum) = 0.1

    'SOLVE_AND_PLOT_LINKAGE

    'Debug.Print temp

    ASSEMBLE_STRING

    Exit Sub

mnuParamWeightLength_errortrap:

    status = MsgBox("Lets try that again with a number this time (or just hit cancel
     -->!)", 1 + 48 + 0 + 0, "Input Error!")

    If status = 2 Then Exit Sub
    If status = 1 Then GoTo popup_mnuParamWeightLength

End Sub
```

```
Private Sub mnuParamWeightOffset_Click()

    On Error GoTo mnuParamWeightOffset_errortrap

    Dim this_message As String
    Dim this_title As String
    Dim this_default As String

    this_message = "Enter a new value for the weight arm offset:"
    this_title = "Weight Arm Offset..."
    this_default = Str$(CSng(msngWhtOff(mintCurrentSolNum) / gconDegToRad))

popup_mnuParamWeightOffset:
```

EXT_LINK.FRM - mnuParamWeightOffset_Click

```
    temp = InputBox(this_message, this_title, this_default)

    If temp = "" Then Exit Sub

    msngWhtOff(mintCurrentSolNum) = temp * gconDegToRad

    'SOLVE_AND_PLOT_LINKAGE

    'Debug.Print temp

    ASSEMBLE_STRING

    Exit Sub

mnuParamWeightOffset_errortrap:

    status = MsgBox("Lets try that again with a number this time (or just hit cancel
     -->!)", 1 + 48 + 0 + 0, "Input Error!")

    If status = 2 Then Exit Sub
    If status = 1 Then GoTo popup_mnuParamWeightOffset

End Sub
```

```
Private Sub OPEN_SOLUTIONS_FILE()

'This procedure opens a file that contains candidate solution
'profiles created during synthesis mode.  All of the solutions
'are loaded into large arrays so that the user can view them quickly.

ReDim sngTemporary(1500, 17) As Single
Dim strJunk As String, intIndex As Integer, strSolFileName As String
Dim intCount As Integer

On Error GoTo CancelAnalysisFileOpen
```

EXT_LINK.FRM - mnuParamWeightOffset_Click

```
Screen.MousePointer = HOURGLASS
frmMDI.MousePointer = HOURGLASS

intIndex = 0

    'triggers the file open dialog
    frmAnalysisLinkageDraw.solutions_dialog.Action = 1
    strSolFileName = frmAnalysisLinkageDraw.solutions_dialog.Filename

    Open strSolFileName For Input As #100
        'get the first 2 label lines as junk
        Input #100, strJunk
        Input #100, strJunk
        Do
            Debug.Print "solution #", intIndex
            Input #100, strJunk  'ignore the solution number printed here by the
             -->optimization
            Input #100, sngTemporary(intIndex, 0)
            Input #100, sngTemporary(intIndex, 1)
            Input #100, sngTemporary(intIndex, 2)
            Input #100, sngTemporary(intIndex, 3)
            Input #100, sngTemporary(intIndex, 4)
            Input #100, sngTemporary(intIndex, 5)
            Input #100, sngTemporary(intIndex, 6)
            Input #100, sngTemporary(intIndex, 7)
            Input #100, sngTemporary(intIndex, 8)
            Input #100, sngTemporary(intIndex, 9)
            Input #100, sngTemporary(intIndex, 10)
            Input #100, sngTemporary(intIndex, 11)
            Input #100, sngTemporary(intIndex, 12)
            Input #100, sngTemporary(intIndex, 13)
            Input #100, sngTemporary(intIndex, 14)
            Input #100, sngTemporary(intIndex, 15)
            Input #100, sngTemporary(intIndex, 16)
            Input #100, strJunk   'ignore the trailing blank line here
            'For i = 0 To 16 Step 1
            '    Debug.Print sngTemporary(intIndex, i)
```

```
        'Next i
        Debug.Print
        intIndex = intIndex + 1   'increment the index by one
      Loop Until EOF(100)

EOFReached:
    Close #100  'close the input file
    'decrement by one since we overincremented by one at the end of the file
    mintNumFileSolns = intIndex

    If mintNumFileSolns < 1 Then
        MsgBox "You have selected an empty solutions file - Operation canceled.", 0
         -->+ 64 + 0
        Close #100
        Screen.MousePointer = DEFAULT
        frmMDI.MousePointer = DEFAULT
        Exit Sub
    End If

    MsgBox Str(mintNumFileSolns) + " solutions were found in the chosen file.", 0 +
     -->64 + 0
    'redimension the variable arrays to hold data
    ReDim msngLenInputArm(mintNumFileSolns) As Single
    ReDim msngLen1(mintNumFileSolns) As Single
    ReDim msngLen2(mintNumFileSolns) As Single
    ReDim msngLen3(mintNumFileSolns) As Single
    ReDim msngLen4(mintNumFileSolns) As Single
    ReDim msngLenWhtArm(mintNumFileSolns) As Single
    ReDim msngBetaStart(mintNumFileSolns) As Single
    ReDim msngBetaEnd(mintNumFileSolns) As Single
    ReDim msngInputOff(mintNumFileSolns) As Single
    ReDim msngWhtOff(mintNumFileSolns) As Single
    ReDim mintClosure(mintNumFileSolns) As Integer
    ReDim msngBaseRot(mintNumFileSolns) As Single
    ReDim mintLinkType(mintNumFileSolns) As Integer
    ReDim msngOutputStartAngle(mintNumFileSolns) As Single
    ReDim msngOutputEndAngle(mintNumFileSolns) As Single
```

EXT_LINK.FRM - OPEN_SOLUTIONS_FILE

```
    ReDim msngOutputRange(mintNumFileSolns) As Single
    ReDim msngSolutionError(mintNumFileSolns) As Single

    'now that we're through the file ok, make assignments for all the solutions in the
     -->file
    For intCount = 0 To (mintNumFileSolns - 1) Step 1
        msngBetaStart(intCount) = sngTemporary(intCount, 0) * gconDegToRad
        msngBetaEnd(intCount) = sngTemporary(intCount, 1) * gconDegToRad
        mintLinkType(intCount) = sngTemporary(intCount, 2)
        mintClosure(intCount) = sngTemporary(intCount, 3)
        msngLenInputArm(intCount) = sngTemporary(intCount, 4)
        msngLenWhtArm(intCount) = sngTemporary(intCount, 5)
        msngInputOff(intCount) = sngTemporary(intCount, 6) * gconDegToRad
        msngWhtOff(intCount) = sngTemporary(intCount, 7) * gconDegToRad
        msngBaseRot(intCount) = sngTemporary(intCount, 8) * gconDegToRad
        msngLen1(intCount) = sngTemporary(intCount, 9)
        msngLen2(intCount) = sngTemporary(intCount, 10)
        msngLen3(intCount) = sngTemporary(intCount, 11)
        msngLen4(intCount) = sngTemporary(intCount, 12)
        msngSolutionError(intCount) = sngTemporary(intCount, 16)
    Next intCount

    'fill the table of errors now
    frmErrorTable.txtErrorTable.Text = ""
    For intCount = 0 To (mintNumFileSolns - 1) Step 1
        frmErrorTable.txtErrorTable.Text = frmErrorTable.txtErrorTable.Text + "
         -->Solution # " + Str(intCount) + ":  " + Format(msngSolutionError(intCount),
         -->"######0.0######") + gstrNewLine
    Next intCount

    'here, compute the output information and put the info into the linkage arrays
    For mintCurrentSolNum = 0 To (mintNumFileSolns - 1) Step 1
        COMPUTE_RANGE_OF_MOTION
    Next mintCurrentSolNum

    'set the current solution to the first and set the caption on the display
    mintCurrentSolNum = 0
```

```
pnlSolutionNumber.Caption = Str(mintCurrentSolNum)

'enable the spin buttons on the menubar
Call EnableControl(spnAnalysisZoom)
Call EnableControl(spnAnalysisRotate)
Call EnableControl(spnAnalysisScale)
Call EnableControl(spnAnalysisSolution)

pnlSolutionNumber.Enabled = True

'enable the disabled menus
mnuFilePrintDrawing.Enabled = True
mnuCopyDrawing.Enabled = True
mnuParameters.Enabled = True
mnuOptions.Enabled = True
mnuAction.Enabled = True

'enable the toolbar buttons
barAnalysisDraw.ButtonEnabled(1) = True   'print button
barAnalysisDraw.ButtonEnabled(2) = True   'animate button

'enable the mouseclick events
picAnalysisDraw.Enabled = True

'change caption to include the filename being looked at
frmAnalysisLinkageDraw.Caption = "Linkage Analysis [DRAW] - " +
 -->strSolFileName

'put the text in the window and get the strength curve
ASSEMBLE_STRING
SOLVE_AND_PLOT_LINKAGE

Screen.MousePointer = DEFAULT
frmMDI.MousePointer = DEFAULT

Exit Sub
```



```
CancelAnalysisFileOpen:
'if EOF reached during read, we'll get that error here and continue above
If Err = 62 Then
   GoTo EOFReached
End If

'getting to here indicates a bad file was opened
'close the opened file in case of an error
Close #100
MsgBox "An error has occured or operation cancelled by user.", 0 + 48 + 0
Screen.MousePointer = DEFAULT
frmMDI.MousePointer = DEFAULT
Exit Sub

End Sub
```

```
Private Sub OVERLAY_STRENGTH_CURVE()

'This procedure is used to overlay strength curve data from a
'strength data file on the resistance curve of the solution
'currently being viewed in analysis mode.

Dim strMessage As String, varDummy1, varDummy2

start_of_procedure:

   On Error GoTo cancel_overlay

   mintNumOverPoints = 0
   mintOverPointIndex = 0

   ChDir App.Path
   frmMainOptimzation.dlgLoadStrengthData.Filename = ""
   frmMainOptimzation.dlgLoadStrengthData.Action = 1

   mstrOverlayFile = frmMainOptimzation.dlgLoadStrengthData.Filename
```

EXT_LINK.FRM - OVERLAY_STRENGTH_CURVE

```
'================================================

If (FILE_EXISTS(mstrOverlayFile)) = 0 Then
   strMessage = "The strength data file " + mstrOverlayFile + " was not found at
    -->the expected location.  Check the path, filename, spelling, etc..  The
    -->Acquire Strength Data dialog box will be displayed after closing this
    -->message."
   MsgBox strMessage, 0 + 16, "Problem Opening File"
   GoTo start_of_procedure
End If

Open mstrOverlayFile For Input As #60
Do While Not EOF(60)
   Input #60, varDummy1, varDummy2
   mintNumOverPoints = mintNumOverPoints + 1
Loop
Close #60

ReDim msngOverBetaVals(mintNumOverPoints, 1) As Single
ReDim msngOverStrengthVals(mintNumOverPoints, 1) As Single

Open mstrOverlayFile For Input As #60
Do
   mintOverPointIndex = mintOverPointIndex + 1
   Input #60, msngOverBetaVals(mintOverPointIndex, 1),
    -->msngOverStrengthVals(mintOverPointIndex, 1)
   'Debug.Print msngOverBetaVals(mintOverPointIndex, 1),
    -->msngOverStrengthVals(mintOverPointIndex, 1)
Loop Until EOF(60)
Close #60

'flag that an overlay will be used
mintUseOverlay = 1
'turn on the strength curve plot overlay option
frmAnalysisCurves.gphAnalysisStrengthPlot.OverlayGraph = 1
```

EXT_LINK.FRM - OVERLAY_STRENGTH_CURVE

```
   'make some menu switches
   frmAnalysisLinkageDraw.mnuActionOverlayCurve.Enabled = False
   frmAnalysisLinkageDraw.mnuActionRemoveOverlay.Enabled = True

   Exit Sub

cancel_overlay:

   MsgBox "An error has occured in acquiring the strength curve data.  Check the
    -->file perhaps."
   Close #60
   Exit Sub

End Sub
```

```
Private Sub picAnalysisDraw_MouseDown(button As Integer, Shift As Integer, X
 -->As Single, Y As Single)

   If button = LEFT_BUTTON Then
      msngFormPanLR = msngFormPanLR + X
      msngFormPanUD = msngFormPanUD + Y
      frmAnalysisLinkageDraw.picAnalysisDraw.Scale (-msngFormZoom -
       -->msngFormPanLR, msngFormZoom - msngFormPanUD)-(msngFormZoom -
       -->msngFormPanLR, -msngFormZoom - msngFormPanUD)
      frmAnalysisLinkageDraw.picAnalysisDraw.Cls
      DRAW_LINKAGE
   End If

End Sub
```

```
Private Sub picAnalysisDraw_MouseMove(button As Integer, Shift As Integer, X
 -->As Single, Y As Single)

    If button = LEFT_BUTTON Then
        msngFormPanLR = msngFormPanLR + X
        msngFormPanUD = msngFormPanUD + Y
        frmAnalysisLinkageDraw.picAnalysisDraw.Scale (-msngFormZoom -
         -->msngFormPanLR, msngFormZoom - msngFormPanUD)-(msngFormZoom -
         -->msngFormPanLR, -msngFormZoom - msngFormPanUD)
        frmAnalysisLinkageDraw.picAnalysisDraw.Cls
        DRAW_LINKAGE
    End If

End Sub
```

```
Private Sub pnlSolutionNumber_DblClick()

On Error GoTo pnlSolutionNumber_errortrap

Dim this_message As String
Dim this_title As String
Dim this_default As String

this_message = "Enter a solution number to examine:"
this_title = "Input Integer Value..."
this_default = pnlSolutionNumber.Caption

popup_pnlSolutionNumber:
temp = InputBox(this_message, this_title, this_default)

If temp = "" Then Exit Sub

temp = CDbl(temp)

mintCurrentSolNum = temp
```

EXT_LINK.FRM - picAnalysisDraw_MouseMove

```
If mintCurrentSolNum < 0 Then mintCurrentSolNum = 0
If mintCurrentSolNum > (mintNumFileSolns - 1) Then mintCurrentSolNum = (
 -->mintNumFileSolns - 1)
pnlSolutionNumber.Caption = Str(mintCurrentSolNum)
ASSEMBLE_STRING
SOLVE_AND_PLOT_LINKAGE

'Debug.Print temp

Exit Sub

pnlSolutionNumber_errortrap:

status = MsgBox("Lets try that again with a number this time (or just hit cancel!)", 1
 --> + 48 + 0 + 0, "Input Error!")

If status = 2 Then Exit Sub
If status = 1 Then GoTo popup_pnlSolutionNumber


End Sub
```

```
Private Sub POSITION_ANALYSIS(pos_closure As Integer, pos_l1 As Single,
 -->pos_l2 As Single, pos_l3 As Single, pos_l4 As Single, pos_rot As Single,
 -->pos_t2 As Single, pos_t3 As Single, pos_t4 As Single, pos_error As String)

'This procedure performs a position analysis of the four-bar linkage
'and returns the angles theta3 and theta4 as well as an error message
'if the linkage cannot physically assemble.

    Dim sngTemp1  As Single, sngTemp2 As Single, sngRotX As Single, sngRotY
     -->As Single
    Dim E3 As Single, F3 As Single, G3 As Single, E4 As Single, F4 As Single, G4
     -->As Single
    Dim N3 As Single, N4 As Single
```

```
pos_error = "No Errors"
'these are the rotation effects
sngRotX = pos_l1 * Cos(pos_rot)
sngRotY = pos_l1 * Sin(pos_rot)

'these are used for the position analysis to get Theta3, Theta4, etc.
E3 = 2 * pos_l2 * pos_l3 * Sin(pos_t2) - 2 * pos_l3 * sngRotY
F3 = 2 * pos_l2 * pos_l3 * Cos(pos_t2) - 2 * pos_l3 * sngRotX
G3 = pos_l2 ^ 2 + pos_l3 ^ 2 + sngRotX ^ 2 + sngRotY ^ 2 - pos_l4 ^ 2 - 2 *
 -->pos_l2 * sngRotX * Cos(pos_t2) - 2 * pos_l2 * sngRotY * Sin(pos_t2)
sngTemp1 = E3 ^ 2 + F3 ^ 2 - G3 ^ 2

E4 = -2 * pos_l2 * pos_l4 * Sin(pos_t2) + 2 * pos_l4 * sngRotY
F4 = -2 * pos_l2 * pos_l4 * Cos(pos_t2) + 2 * pos_l4 * sngRotX
G4 = pos_l2 ^ 2 + pos_l4 ^ 2 + sngRotX ^ 2 + sngRotY ^ 2 - pos_l3 ^ 2 - 2 *
 -->pos_l2 * sngRotX * Cos(pos_t2) - 2 * pos_l2 * sngRotY * Sin(pos_t2)
sngTemp2 = E4 ^ 2 + F4 ^ 2 - G4 ^ 2

'if both are positive then the linkage assembles, otherwise if either one or both
 -->fails then assembly is no good
If (sngTemp1 >= 0 And sngTemp2 >= 0) Then
   N3 = -E3 + pos_closure * Sqr(sngTemp1)
   N4 = -E4 - pos_closure * Sqr(sngTemp2)
   pos_t3 = 2 * atan(N3, (G3 - F3))
   pos_t4 = 2 * atan(N4, (G4 - F4))
   'Debug.Print "theta3,theta4"; pos_t3 / gconDegToRad, pos_t4 /
    -->gconDegToRad
Else
   pos_error = "Linkage Doesn't Assemble"
   pos_t3 = 0#
   pos_t4 = 0#
   'Debug.Print "error bad linkage"
End If

End Sub
```

EXT_LINK.FRM - POSITION_ANALYSIS

```
Private Sub PRINT_LINKAGE()

'This routine prints a drawing of the current solution linkage
'and also dumps the relevant linkage info currently displayed
'in the analysis output window.

   Printer.Scale (-msngFormZoom - msngFormPanLR, msngFormZoom -
    -->msngFormPanUD)-(msngFormZoom - msngFormPanLR, -msngFormZoom -
    -->msngFormPanUD)

   Printer.DrawWidth = 12
   Printer.DrawStyle = 0

   Printer.Line (msngX1, msngY1)-(msngX2, msngY2)
   Printer.Line (msngX3, msngY3)-(msngX4, msngY4)
   Printer.Line (msngX2, msngY2)-(msngX3, msngY3)

   Printer.FillStyle = TRANSPARENT
   'printer.FillStyle = UPWARD_DIAGONAL
   'printer.Circle (msngXOut, msngYOut), msngLenWhtArm(mintCurrentSolNum) /
    -->5#
   Printer.Circle (msngXOut, msngYOut), 8#

   Printer.DrawWidth = 25
   Printer.Line (msngX4, msngY4)-(msngX1, msngY1)
   Printer.Line (msngX1, msngY1)-(msngXIn, msngYIn)
   Printer.Line (msngX4, msngY4)-(msngXOut, msngYOut)

   Printer.DrawStyle = 2
   Printer.DrawWidth = 12
   Printer.Line (msngX3, msngY3)-(msngXOut, msngYOut)
   Printer.Line (msngX2, msngY2)-(msngXIn, msngYIn)

   Printer.NewPage

   Printer.CurrentX = Printer.ScaleLeft + Printer.ScaleWidth / 5
```

```
    Printer.CurrentY = Printer.ScaleTop + Printer.ScaleHeight / 10

    Printer.Print "Input Arm Start Angle:  "; Format(msngBetaStart(
     -->mintCurrentSolNum) / gconDegToRad, "###0.0##")
    Printer.CurrentX = Printer.ScaleLeft + Printer.ScaleWidth / 5
    Printer.Print "Input Arm End Angle:  "; Format(msngBetaEnd(mintCurrentSolNum)
     -->/ gconDegToRad, "###0.0##")
    Printer.CurrentX = Printer.ScaleLeft + Printer.ScaleWidth / 5
    If mintLinkType(mintCurrentSolNum) = 1 Then
       Printer.Print "Linkage Type:  "; "Co-rotating"
    ElseIf mintLinkType(mintCurrentSolNum) = -1 Then
       Printer.Print "Linkage Type:  "; "Counter-rotating"
    End If
    Printer.CurrentX = Printer.ScaleLeft + Printer.ScaleWidth / 5
    Printer.Print "Linkage Closure:  "; mintClosure(mintCurrentSolNum)
    Printer.CurrentX = Printer.ScaleLeft + Printer.ScaleWidth / 5
    Printer.Print "Input Arm Length:  "; Format(msngLenInputArm(mintCurrentSolNum
     -->), "###0.0##")
    Printer.CurrentX = Printer.ScaleLeft + Printer.ScaleWidth / 5
    Printer.Print "Weight Arm Length:  "; Format(msngLenWhtArm(
     -->mintCurrentSolNum), "###0.0##")
    Printer.CurrentX = Printer.ScaleLeft + Printer.ScaleWidth / 5
    Printer.Print "Input Arm Offset:  "; Format(msngInputOff(mintCurrentSolNum) /
     -->gconDegToRad, "###0.0##")
    Printer.CurrentX = Printer.ScaleLeft + Printer.ScaleWidth / 5
    Printer.Print "Weight Arm Offset:  "; Format(msngWhtOff(mintCurrentSolNum) /
     -->gconDegToRad, "###0.0##")
    Printer.CurrentX = Printer.ScaleLeft + Printer.ScaleWidth / 5
    Printer.Print "Linkage Base Rotation:  "; Format(msngBaseRot(
     -->mintCurrentSolNum) / gconDegToRad, "###0.0##")
    Printer.CurrentX = Printer.ScaleLeft + Printer.ScaleWidth / 5
    Printer.Print "Link 1 length:  "; Format(msngLen1(mintCurrentSolNum), "###0.0
     -->##")
    Printer.CurrentX = Printer.ScaleLeft + Printer.ScaleWidth / 5
    Printer.Print "Link 2 length:  "; Format(msngLen2(mintCurrentSolNum), "###0.0
     -->##")
    Printer.CurrentX = Printer.ScaleLeft + Printer.ScaleWidth / 5
```



```
    Printer.Print "Link 3 length:  "; Format(msngLen3(mintCurrentSolNum), "###0.0
     -->##")
    Printer.CurrentX = Printer.ScaleLeft + Printer.ScaleWidth / 5
    Printer.Print "Link 4 length:  "; Format(msngLen4(mintCurrentSolNum), "###0.0
     -->##")
    Printer.CurrentX = Printer.ScaleLeft + Printer.ScaleWidth / 5
    Printer.Print "Weight Arm Start Angle:  "; Format(msngOutputStartAngle(
     -->mintCurrentSolNum), "###0.0##")
    Printer.CurrentX = Printer.ScaleLeft + Printer.ScaleWidth / 5
    Printer.Print "Weight Arm End Angle:  "; Format(msngOutputEndAngle(
     -->mintCurrentSolNum), "###0.0##")
    Printer.CurrentX = Printer.ScaleLeft + Printer.ScaleWidth / 5
    Printer.Print "Output Range of Motion:  "; Format(msngOutputRange(
     -->mintCurrentSolNum), "###0.0##")
    Printer.CurrentX = Printer.ScaleLeft + Printer.ScaleWidth / 5
    Printer.Print "Solution Error:  "; Str(msngSolutionError(mintCurrentSolNum))

    Printer.EndDoc

    frmAnalysisCurves.gphAnalysisStrengthPlot.DrawMode = 5
    frmAnalysisCurves.gphAnalysisCouplerForcePlot.DrawMode = 5

    'printer.EndDoc

End Sub

---

Private Sub SOLVE_AND_PLOT_LINKAGE()

'This routine combines the position analysis of the four-bar with
'computing the resistance curve and coupler link force of the current
'linkage.  The resistance curve is plotted on the analysis graph and so
'is the coupler force curve.  The linkage is also drawn in the graphics
'window.  This routine is used exclusively when recalculation of the
'linkage or the curves is required.

    Dim dblMSubG As Double, dblNSubG As Double, dblDSubG As Double
```

```
Dim sngStepSize As Single, sngKCount As Single
Dim sngCurrentBeta As Single, sngStrength As Single, sngCouplerForce As
 -->Single

'msngInputOff = msngInputOff * gconDegToRad
'msngWhtOff = msngWhtOff * gconDegToRad
'msngBetaStart = msngBetaStart * gconDegToRad
'msngBetaEnd = msngBetaEnd * gconDegToRad
'msngBaseRot = msngBaseRot * gconDegToRad

'sets the start angle by definition
msngTheta2 = msngBetaStart(mintCurrentSolNum) - msngInputOff(
 -->mintCurrentSolNum)

'makes a call to the local position analysis while also returns the values needed
 -->in this procedure
Call ANALYSIS_LINKAGE_SOLVER(msngWhtOff(mintCurrentSolNum),
 -->mintClosure(mintCurrentSolNum), msngLen1(mintCurrentSolNum), msngLen2
 -->(mintCurrentSolNum), msngLen3(mintCurrentSolNum), msngLen4(
 -->mintCurrentSolNum), msngBaseRot(mintCurrentSolNum), msngTheta2,

'flags the bad assembly and exits the procedure
If (mstrErrorFlag = "Linkage Doesn't Assemble") Then GoTo
 -->bad_linkage_specified

'Debug.Print "t2,t3,t4,phi1"; msngTheta2 / gconDegToRad, msngTheta3 /
 -->gconDegToRad, msngTheta4 / gconDegToRad, msngPhi / gconDegToRad

'here, the points are all calculated
msngX1 = 0
msngY1 = 0

msngX4 = msngLen1(mintCurrentSolNum) * Cos(msngBaseRot(
 -->mintCurrentSolNum))
msngY4 = msngLen1(mintCurrentSolNum) * Sin(msngBaseRot(
 -->mintCurrentSolNum))
```

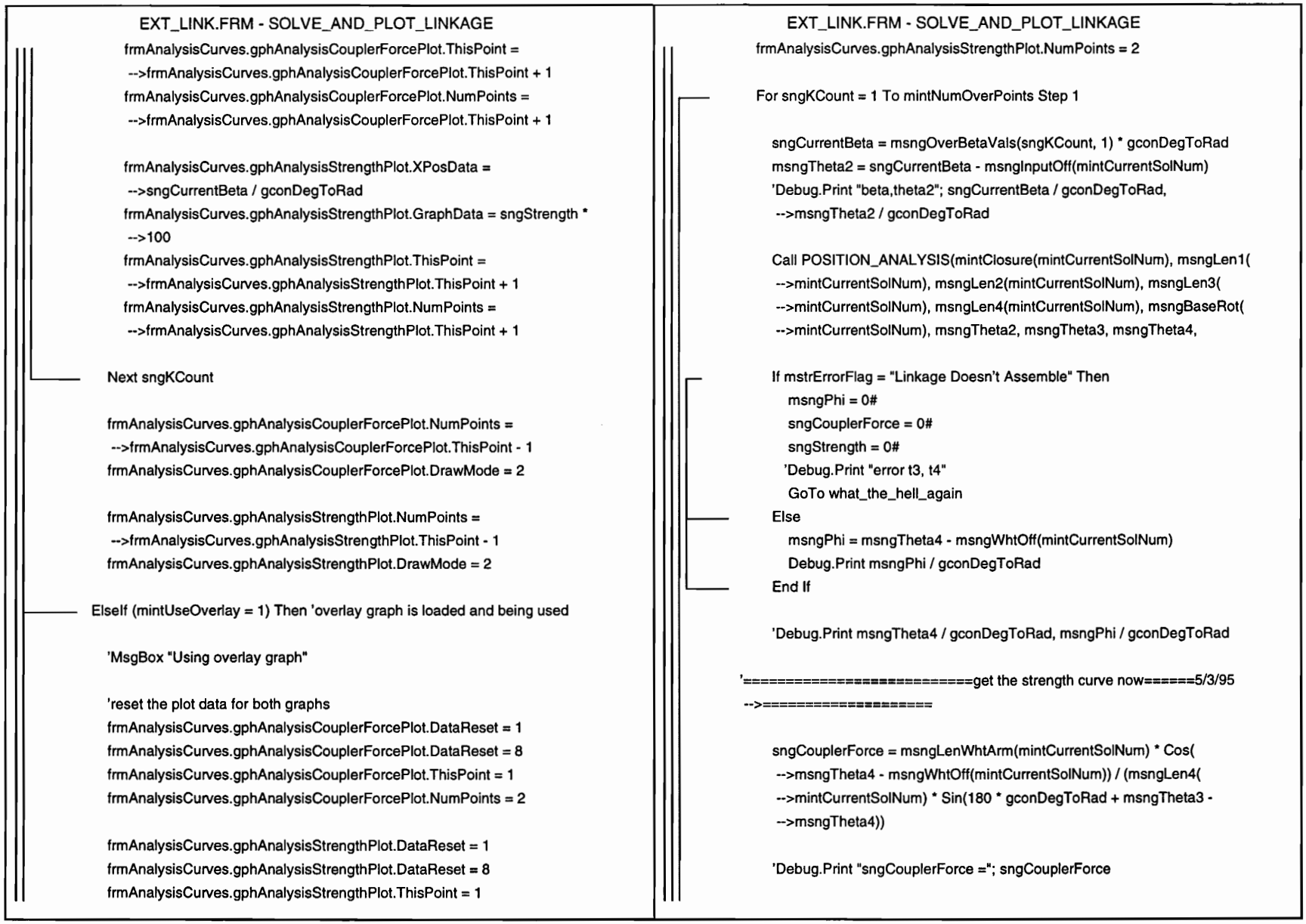
EXT_LINK.FRM - SOLVE_AND_PLOT_LINKAGE

```
msngX2 = msngLen2(mintCurrentSolNum) * Cos(msngTheta2)
msngY2 = msngLen2(mintCurrentSolNum) * Sin(msngTheta2)

msngX3 = msngX4 + msngLen4(mintCurrentSolNum) * Cos(msngTheta4)
msngY3 = msngY4 + msngLen4(mintCurrentSolNum) * Sin(msngTheta4)

'the starting end of the curve has already been taken care of
msngXIn = msngLenInputArm(mintCurrentSolNum) * Cos(msngBetaStart(
 -->mintCurrentSolNum))
msngYIn = msngLenInputArm(mintCurrentSolNum) * Sin(msngBetaStart(
 -->mintCurrentSolNum))

msngXOut = msngX4 + msngLenWhtArm(mintCurrentSolNum) * Cos(msngPhi)
msngYOut = msngY4 + msngLenWhtArm(mintCurrentSolNum) * Sin(msngPhi)

'============================================================
 -->==============
 'actually take the points and draws the linkage on the form
 frmAnalysisLinkageDraw.picAnalysisDraw.Cls
 DRAW_LINKAGE


'============================================================
 -->==============

 If (mintAlwaysUpdate = 1) Then  'update the graphs each time through

  If (mintUseOverlay = 0) Then 'if no overlay graph has been loaded to be used

   'reset the plot data for both graphs
   frmAnalysisCurves.gphAnalysisCouplerForcePlot.DataReset = 1
   frmAnalysisCurves.gphAnalysisCouplerForcePlot.DataReset = 8
   frmAnalysisCurves.gphAnalysisCouplerForcePlot.ThisPoint = 1
   frmAnalysisCurves.gphAnalysisCouplerForcePlot.NumPoints = 2

   frmAnalysisCurves.gphAnalysisStrengthPlot.DataReset = 1
   frmAnalysisCurves.gphAnalysisStrengthPlot.DataReset = 8
```

```
frmAnalysisCurves.gphAnalysisStrengthPlot.ThisPoint = 1
frmAnalysisCurves.gphAnalysisStrengthPlot.NumPoints = 2

'get the span of the data set, then divide it up "mintNumPlotPoints" times to
 -->get the increment from step to step
msngBetaRange = msngBetaEnd(mintCurrentSolNum) - msngBetaStart(
 -->mintCurrentSolNum)
sngStepSize = msngBetaRange / (mintNumPlotPoints * gconDegToRad)

'Debug.Print msngBetaRange, sngStepSize, msngBetaStart /
 -->gconDegToRad, msngBetaEnd/ gconDegToRad

For sngKCount = 0 To (msngBetaRange / gconDegToRad + sngStepSize)
 -->Step sngStepSize

   sngCurrentBeta = msngBetaStart(mintCurrentSolNum) + sngKCount *
    -->gconDegToRad
   msngTheta2 = sngCurrentBeta - msngInputOff(mintCurrentSolNum)
   'Debug.Print "beta,theta2"; sngCurrentBeta / gconDegToRad,
    -->msngTheta2 / gconDegToRad

   Call POSITION_ANALYSIS(mintClosure(mintCurrentSolNum), msngLen1(
    -->mintCurrentSolNum), msngLen2(mintCurrentSolNum), msngLen3(
    -->mintCurrentSolNum), msngLen4(mintCurrentSolNum), msngBaseRot(
    -->mintCurrentSolNum), msngTheta2, msngTheta3, msngTheta4,

   If mstrErrorFlag = "Linkage Doesn't Assemble" Then
      msngPhi = 0#
      sngCouplerForce = 0#
      sngStrength = 0#
     'Debug.Print "error t3, t4"
      GoTo what_the_hell
   Else
      msngPhi = msngTheta4 - msngWhtOff(mintCurrentSolNum)
      Debug.Print msngPhi / gconDegToRad
   End If
```



```
      'Debug.Print msngTheta4 / gconDegToRad, msngPhi / gconDegToRad

'==========================get the strength curve now======5/3/95
 -->====================

      sngCouplerForce = msngLenWhtArm(mintCurrentSolNum) * Cos(
       -->msngTheta4 - msngWhtOff(mintCurrentSolNum)) / (msngLen4(
       -->mintCurrentSolNum) * Sin(180 * gconDegToRad + msngTheta3 -
       -->msngTheta4))

      'Debug.Print "sngCouplerForce ="; sngCouplerForce

      'this method not used anymore, so its commented.  The new method is
       -->below
      'sngStrength = -(msngLen2 / msngLenInputArm) * Sin(msngTheta3 -
       -->msngTheta2) * sngCouplerForce * mintLinkType
      'frmAnalysisLinkageDraw.Print sngStrength

      dblNSubG = Sin(msngTheta3) * Cos(msngTheta2) - Cos(msngTheta3) *
       -->Sin(msngTheta2)
      dblDSubG = Cos(msngTheta4) * Sin(msngTheta3) - Cos(msngTheta3) *
       -->Sin(msngTheta4)
      dblMSubG = (msngLenWhtArm(mintCurrentSolNum) * msngLen2(
       -->mintCurrentSolNum) * Cos(msngPhi)) / (msngLenInputArm(
       -->mintCurrentSolNum) * msngLen4(mintCurrentSolNum))

      sngStrength = mintLinkType(mintCurrentSolNum) * (dblMSubG *
       -->dblNSubG) / dblDSubG   'new method from Randy's Thesis 9/27/95

      'Debug.Print sngCurrentBeta / gconDegToRad, sngStrength * 100

what_the_hell:

      frmAnalysisCurves.gphAnalysisCouplerForcePlot.XPosData =
       -->sngCurrentBeta / gconDegToRad
      frmAnalysisCurves.gphAnalysisCouplerForcePlot.GraphData =
       -->sngCouplerForce
```



EXT_LINK.FRM - SOLVE_AND_PLOT_LINKAGE

```
frmAnalysisCurves.gphAnalysisCouplerForcePlot.ThisPoint =
 -->frmAnalysisCurves.gphAnalysisCouplerForcePlot.ThisPoint + 1
frmAnalysisCurves.gphAnalysisCouplerForcePlot.NumPoints =
 -->frmAnalysisCurves.gphAnalysisCouplerForcePlot.ThisPoint + 1

frmAnalysisCurves.gphAnalysisStrengthPlot.XPosData =
 -->sngCurrentBeta / gconDegToRad
frmAnalysisCurves.gphAnalysisStrengthPlot.GraphData = sngStrength *
 -->100
frmAnalysisCurves.gphAnalysisStrengthPlot.ThisPoint =
 -->frmAnalysisCurves.gphAnalysisStrengthPlot.ThisPoint + 1
frmAnalysisCurves.gphAnalysisStrengthPlot.NumPoints =
 -->frmAnalysisCurves.gphAnalysisStrengthPlot.ThisPoint + 1

Next sngKCount

frmAnalysisCurves.gphAnalysisCouplerForcePlot.NumPoints =
 -->frmAnalysisCurves.gphAnalysisCouplerForcePlot.ThisPoint - 1
frmAnalysisCurves.gphAnalysisCouplerForcePlot.DrawMode = 2

frmAnalysisCurves.gphAnalysisStrengthPlot.NumPoints =
 -->frmAnalysisCurves.gphAnalysisStrengthPlot.ThisPoint - 1
frmAnalysisCurves.gphAnalysisStrengthPlot.DrawMode = 2

ElseIf (mintUseOverlay = 1) Then 'overlay graph is loaded and being used

'MsgBox "Using overlay graph"

'reset the plot data for both graphs
frmAnalysisCurves.gphAnalysisCouplerForcePlot.DataReset = 1
frmAnalysisCurves.gphAnalysisCouplerForcePlot.DataReset = 8
frmAnalysisCurves.gphAnalysisCouplerForcePlot.ThisPoint = 1
frmAnalysisCurves.gphAnalysisCouplerForcePlot.NumPoints = 2

frmAnalysisCurves.gphAnalysisStrengthPlot.DataReset = 1
frmAnalysisCurves.gphAnalysisStrengthPlot.DataReset = 8
frmAnalysisCurves.gphAnalysisStrengthPlot.ThisPoint = 1
```

EXT_LINK.FRM - SOLVE_AND_PLOT_LINKAGE

```
frmAnalysisCurves.gphAnalysisStrengthPlot.NumPoints = 2

For sngKCount = 1 To mintNumOverPoints Step 1

  sngCurrentBeta = msngOverBetaVals(sngKCount, 1) * gconDegToRad
  msngTheta2 = sngCurrentBeta - msngInputOff(mintCurrentSolNum)
  'Debug.Print "beta,theta2"; sngCurrentBeta / gconDegToRad,
   -->msngTheta2 / gconDegToRad

  Call POSITION_ANALYSIS(mintClosure(mintCurrentSolNum), msngLen1(
   -->mintCurrentSolNum), msngLen2(mintCurrentSolNum), msngLen3(
   -->mintCurrentSolNum), msngLen4(mintCurrentSolNum), msngBaseRot(
   -->mintCurrentSolNum), msngTheta2, msngTheta3, msngTheta4,

  If mstrErrorFlag = "Linkage Doesn't Assemble" Then
    msngPhi = 0#
    sngCouplerForce = 0#
    sngStrength = 0#
   'Debug.Print "error t3, t4"
    GoTo what_the_hell_again
  Else
    msngPhi = msngTheta4 - msngWhtOff(mintCurrentSolNum)
    Debug.Print msngPhi / gconDegToRad
  End If

  'Debug.Print msngTheta4 / gconDegToRad, msngPhi / gconDegToRad

'============================get the strength curve now======5/3/95
 -->====================

  sngCouplerForce = msngLenWhtArm(mintCurrentSolNum) * Cos(
   -->msngTheta4 - msngWhtOff(mintCurrentSolNum)) / (msngLen4(
   -->mintCurrentSolNum) * Sin(180 * gconDegToRad + msngTheta3 -
   -->msngTheta4))

  'Debug.Print "sngCouplerForce ="; sngCouplerForce
```

```
'this method not used anymore, so its commented.  The new method is
 -->below
'sngStrength = -(msngLen2 / msngLenInputArm) * Sin(msngTheta3 -
 -->msngTheta2) * sngCouplerForce * mintLinkType
'frmAnalysisLinkageDraw.Print sngStrength

dblNSubG = Sin(msngTheta3) * Cos(msngTheta2) - Cos(msngTheta3) *
 -->Sin(msngTheta2)
dblDSubG = Cos(msngTheta4) * Sin(msngTheta3) - Cos(msngTheta3) *
 -->Sin(msngTheta4)
dblMSubG = (msngLenWhtArm(mintCurrentSolNum) * msngLen2(
 -->mintCurrentSolNum) * Cos(msngPhi)) / (msngLenInputArm(
 -->mintCurrentSolNum) * msngLen4(mintCurrentSolNum))

sngStrength = mintLinkType(mintCurrentSolNum) * (dblMSubG *
 -->dblNSubG) / dblDSubG   'new method from Randy's Thesis 9/27/95

'Debug.Print sngCurrentBeta / gconDegToRad, sngStrength * 100

what_the_hell_again:

frmAnalysisCurves.gphAnalysisCouplerForcePlot.XPosData =
 -->sngCurrentBeta / gconDegToRad
frmAnalysisCurves.gphAnalysisCouplerForcePlot.GraphData =
 -->sngCouplerForce
frmAnalysisCurves.gphAnalysisCouplerForcePlot.ThisPoint =
 -->frmAnalysisCurves.gphAnalysisCouplerForcePlot.ThisPoint + 1
frmAnalysisCurves.gphAnalysisCouplerForcePlot.NumPoints =
 -->frmAnalysisCurves.gphAnalysisCouplerForcePlot.ThisPoint + 1

frmAnalysisCurves.gphAnalysisStrengthPlot.XPosData =
 -->msngOverBetaVals(sngKCount, 1)
frmAnalysisCurves.gphAnalysisStrengthPlot.GraphData = sngStrength *
 -->100
frmAnalysisCurves.gphAnalysisStrengthPlot.OverlayGraphData =
 -->msngOverStrengthVals(sngKCount, 1)
frmAnalysisCurves.gphAnalysisStrengthPlot.ThisPoint =
```

EXT_LINK.FRM - SOLVE_AND_PLOT_LINKAGE

```
 -->frmAnalysisCurves.gphAnalysisStrengthPlot.ThisPoint + 1
frmAnalysisCurves.gphAnalysisStrengthPlot.NumPoints =
 -->frmAnalysisCurves.gphAnalysisStrengthPlot.ThisPoint + 1

Next sngKCount

frmAnalysisCurves.gphAnalysisCouplerForcePlot.NumPoints =
 -->frmAnalysisCurves.gphAnalysisCouplerForcePlot.ThisPoint - 1
frmAnalysisCurves.gphAnalysisCouplerForcePlot.DrawMode = 2

frmAnalysisCurves.gphAnalysisStrengthPlot.NumPoints =
 -->frmAnalysisCurves.gphAnalysisStrengthPlot.ThisPoint - 1
frmAnalysisCurves.gphAnalysisStrengthPlot.DrawMode = 2


End If 'endif for overlay graph

End If 'endif for update every time

bad_linkage_specified:

If (mstrErrorFlag = "Linkage Doesn't Assemble") Then
   status = MsgBox("The linkage parameters specified do not constitute a
    -->physically assembled linkage.  Check each parameter entered.", 0 + 48 + 0
    -->+ 0, "Linkage Doesn't Assemble !!!")
End If

End Sub

---

Private Sub spnAnalysisRotate_SpinDown()

   msngBaseRot(mintCurrentSolNum) = msngBaseRot(mintCurrentSolNum) - 5 *
    -->gconDegToRad
   msngInputOff(mintCurrentSolNum) = msngInputOff(mintCurrentSolNum) + 5 *
    -->gconDegToRad
   msngWhtOff(mintCurrentSolNum) = msngWhtOff(mintCurrentSolNum) - 5 *
```

```
    -->gconDegToRad
    ASSEMBLE_STRING
    mintAlwaysUpdate = -1
    SOLVE_AND_PLOT_LINKAGE
    mintAlwaysUpdate = 1

End Sub
```

```
Private Sub spnAnalysisRotate_SpinUp()

    msngBaseRot(mintCurrentSolNum) = msngBaseRot(mintCurrentSolNum) + 5 *
     -->gconDegToRad
    msngInputOff(mintCurrentSolNum) = msngInputOff(mintCurrentSolNum) - 5 *
     -->gconDegToRad
    msngWhtOff(mintCurrentSolNum) = msngWhtOff(mintCurrentSolNum) + 5 *
     -->gconDegToRad
    ASSEMBLE_STRING
    mintAlwaysUpdate = -1
    SOLVE_AND_PLOT_LINKAGE
    mintAlwaysUpdate = 1

End Sub
```

```
Private Sub spnAnalysisScale_SpinDown()

    msngScaleFactor = 1 / 1.1
    msngLen1(mintCurrentSolNum) = msngLen1(mintCurrentSolNum) *
     -->msngScaleFactor
    msngLen2(mintCurrentSolNum) = msngLen2(mintCurrentSolNum) *
     -->msngScaleFactor
    msngLen3(mintCurrentSolNum) = msngLen3(mintCurrentSolNum) *
     -->msngScaleFactor
    msngLen4(mintCurrentSolNum) = msngLen4(mintCurrentSolNum) *
     -->msngScaleFactor
    ASSEMBLE_STRING
```

EXT_LINK.FRM - spnAnalysisRotate_SpinDown

```
    mintAlwaysUpdate = -1
    SOLVE_AND_PLOT_LINKAGE
    mintAlwaysUpdate = 1

End Sub
```

```
Private Sub spnAnalysisScale_SpinUp()

    msngScaleFactor = 1.1
    msngLen1(mintCurrentSolNum) = msngLen1(mintCurrentSolNum) *
     -->msngScaleFactor
    msngLen2(mintCurrentSolNum) = msngLen2(mintCurrentSolNum) *
     -->msngScaleFactor
    msngLen3(mintCurrentSolNum) = msngLen3(mintCurrentSolNum) *
     -->msngScaleFactor
    msngLen4(mintCurrentSolNum) = msngLen4(mintCurrentSolNum) *
     -->msngScaleFactor
    ASSEMBLE_STRING
    mintAlwaysUpdate = -1
    SOLVE_AND_PLOT_LINKAGE
    mintAlwaysUpdate = 1

End Sub
```

```
Private Sub spnAnalysisSolution_SpinDown()

    mintCurrentSolNum = mintCurrentSolNum - 1
    If mintCurrentSolNum < 0 Then mintCurrentSolNum = mintNumFileSolns - 1
    pnlSolutionNumber.Caption = Str(mintCurrentSolNum)
    ASSEMBLE_STRING
    SOLVE_AND_PLOT_LINKAGE

End Sub
```

EXT_LINK.FRM - spnAnalysisSolution_SpinUp

```
Private Sub spnAnalysisSolution_SpinUp()

   mintCurrentSolNum = mintCurrentSolNum + 1
   If mintCurrentSolNum > (mintNumFileSolns - 1) Then mintCurrentSolNum = 0
   pnlSolutionNumber.Caption = Str(mintCurrentSolNum)
   ASSEMBLE_STRING
   SOLVE_AND_PLOT_LINKAGE

End Sub
```

```
Private Sub spnAnalysisZoom_SpinDown()

   msngFormZoom = msngFormZoom + 3
   frmAnalysisLinkageDraw.picAnalysisDraw.Scale (-msngFormZoom -
    -->msngFormPanLR, msngFormZoom - msngFormPanUD)-(msngFormZoom -
    -->msngFormPanLR, -msngFormZoom - msngFormPanUD)
   frmAnalysisLinkageDraw.picAnalysisDraw.Cls
   DRAW_LINKAGE

End Sub
```

```
Private Sub spnAnalysisZoom_SpinUp()

   msngFormZoom = msngFormZoom - 3
   frmAnalysisLinkageDraw.picAnalysisDraw.Scale (-msngFormZoom -
    -->msngFormPanLR, msngFormZoom - msngFormPanUD)-(msngFormZoom -
    -->msngFormPanLR, -msngFormZoom - msngFormPanUD)
   frmAnalysisLinkageDraw.picAnalysisDraw.Cls
   DRAW_LINKAGE

End Sub
```

EXT_LINK.FRM - spnAnalysisSolution_SpinUp

```
Private Sub Form_Load()

   SystemMenu% = GetSystemMenu(frmAnalysisCurves.hWnd, 0)
   Res% = RemoveMenu(SystemMenu%, 6, MF_BYPOSITION)
   Res% = RemoveMenu(SystemMenu%, 6, MF_BYPOSITION) 'also remove the
    -->separator line

   frmAnalysisCurves.Top = 0
   frmAnalysisCurves.Left = frmAnalysisLinkageDraw.Width
   frmAnalysisCurves.Width = frmMDI.Width / 2.5
   frmAnalysisCurves.Height = frmMDI.Height / 3#

   frmAnalysisCurves.gphAnalysisStrengthPlot.PrintInfo(1) = Printer.hDC

   frmAnalysisCurves.gphAnalysisStrengthPlot.PrintInfo(6) = Printer.ScaleLeft
   frmAnalysisCurves.gphAnalysisStrengthPlot.PrintInfo(7) = Printer.ScaleTop
   frmAnalysisCurves.gphAnalysisStrengthPlot.PrintInfo(8) = Printer.ScaleWidth
   frmAnalysisCurves.gphAnalysisStrengthPlot.PrintInfo(9) = Printer.ScaleHeight

   frmAnalysisCurves.gphAnalysisStrengthPlot.PrintInfo(2) = Printer.CurrentX
   frmAnalysisCurves.gphAnalysisStrengthPlot.PrintInfo(3) = Printer.CurrentY
   frmAnalysisCurves.gphAnalysisStrengthPlot.PrintInfo(4) = Printer.ScaleWidth / 1
    -->.2

   frmAnalysisCurves.gphAnalysisCouplerForcePlot.PrintInfo(1) = Printer.hDC

   frmAnalysisCurves.gphAnalysisCouplerForcePlot.PrintInfo(6) = Printer.ScaleLeft
   frmAnalysisCurves.gphAnalysisCouplerForcePlot.PrintInfo(7) = Printer.ScaleTop
   frmAnalysisCurves.gphAnalysisCouplerForcePlot.PrintInfo(8) = Printer.
    -->ScaleWidth
   frmAnalysisCurves.gphAnalysisCouplerForcePlot.PrintInfo(9) = Printer.
    -->ScaleHeight

   frmAnalysisCurves.gphAnalysisCouplerForcePlot.PrintInfo(2) = Printer.CurrentX
   frmAnalysisCurves.gphAnalysisCouplerForcePlot.PrintInfo(3) = Printer.CurrentY
   frmAnalysisCurves.gphAnalysisCouplerForcePlot.PrintInfo(4) = Printer.
```

```
    -->ScaleWidth / 1.2

End Sub

-------------------------------------------------------------

Private Sub Form_Resize()

   frmAnalysisCurves.gphAnalysisStrengthPlot.Width = frmAnalysisCurves.
    -->ScaleWidth
   frmAnalysisCurves.gphAnalysisStrengthPlot.Height = frmAnalysisCurves.
    -->ScaleHeight

   frmAnalysisCurves.gphAnalysisCouplerForcePlot.Width = frmAnalysisCurves.
    -->ScaleWidth
   frmAnalysisCurves.gphAnalysisCouplerForcePlot.Height = frmAnalysisCurves.
    -->ScaleHeight

End Sub

-------------------------------------------------------------

Private Sub mnuCopyGraph_Click()

   If gphAnalysisStrengthPlot.Visible = True Then
      frmAnalysisCurves.gphAnalysisStrengthPlot.DrawMode = 4
   ElseIf gphAnalysisCouplerForcePlot.Visible = True Then
      frmAnalysisCurves.gphAnalysisCouplerForcePlot.DrawMode = 4
   End If

End Sub

-------------------------------------------------------------

Private Sub mnuPrintGraph_Click()

   If gphAnalysisStrengthPlot.Visible = True Then
      frmAnalysisCurves.gphAnalysisStrengthPlot.DrawMode = 5
   ElseIf gphAnalysisCouplerForcePlot.Visible = True Then
      frmAnalysisCurves.gphAnalysisCouplerForcePlot.DrawMode = 5
```

EXTERN.FRM - Form_Load

```
   End If

End Sub

-------------------------------------------------------------

Private Sub mnuSwitchGraph_Click()

   gphAnalysisStrengthPlot.Visible = Not (gphAnalysisStrengthPlot.Visible)
   gphAnalysisCouplerForcePlot.Visible = Not (gphAnalysisCouplerForcePlot.
    -->Visible)

End Sub
```

EXT_TEXT.FRM - Form_Load

```
Private Sub Form_Load()

   SystemMenu% = GetSystemMenu(frmAnalysisTextOutput.hWnd, 0)
   Res% = RemoveMenu(SystemMenu%, 6, MF_BYPOSITION)
   Res% = RemoveMenu(SystemMenu%, 6, MF_BYPOSITION) 'also remove the
    -->separator line

   frmAnalysisTextOutput.Width = frmMDI.Width / 4
   frmAnalysisTextOutput.Height = frmMDI.Height / 2
   frmAnalysisTextOutput.Top = frmAnalysisCurves.Top + frmAnalysisCurves.
    -->Height
   frmAnalysisTextOutput.Left = frmAnalysisCurves.Left

End Sub
```

```
Private Sub Form_Resize()

   If Me.WindowState <> 1 And Me.ScaleHeight <> 0 Then
      txtAnalysisOutput.Visible = False
      txtAnalysisOutput.Height = Me.ScaleHeight
      txtAnalysisOutput.Width = Me.ScaleWidth
      txtAnalysisOutput.Visible = True
   End If

End Sub
```

```
Private Sub mnuCopySelectedText_Click()

   Clipboard.SetText frmAnalysisTextOutput.txtAnalysisOutput.SelText

End Sub
```

EXT_TEXT.FRM - Form_Load

```
Private Sub chkMaxOutAngle_Click(Value As Integer)

   If Value = True Then gintPhiEndConstrainOn = 1
   If Value = False Then gintPhiEndConstrainOn = -2

   If gintPhiEndConstrainOn = 1 Then
      Call EnableControl(pnlMaxOutAngleLow)
      Call EnableControl(pnlMaxOutAngleHi)
      Call EnableControl(pnlMaxOutAnglePenalty)
   ElseIf gintPhiEndConstrainOn = -2 Then
      Call DisableControl(pnlMaxOutAngleLow, MEDIUMGRAY)
      Call DisableControl(pnlMaxOutAngleHi, MEDIUMGRAY)
      Call DisableControl(pnlMaxOutAnglePenalty, MEDIUMGRAY)
   End If

   Debug.Print gintPhiEndConstrainOn

End Sub
```

```
Private Sub chkMinOutAngle_Click(Value As Integer)

   If Value = True Then gintPhiStartConstrainOn = 1
   If Value = False Then gintPhiStartConstrainOn = -2

   If gintPhiStartConstrainOn = 1 Then
      Call EnableControl(pnlMinOutAngleLow)
      Call EnableControl(pnlMinOutAngleHi)
      Call EnableControl(pnlMinOutAnglePenalty)
   ElseIf gintPhiStartConstrainOn = -2 Then
      Call DisableControl(pnlMinOutAngleLow, MEDIUMGRAY)
      Call DisableControl(pnlMinOutAngleHi, MEDIUMGRAY)
      Call DisableControl(pnlMinOutAnglePenalty, MEDIUMGRAY)
   End If

   Debug.Print gintPhiStartConstrainOn
```

```
End Sub

---

Private Sub Form_Load()

    pnlMinOutAngleLow.Caption = gdblPhiStartLow
    pnlMinOutAngleHi.Caption = gdblPhiStartHi

    pnlMaxOutAngleLow.Caption = gdblPhiEndLow
    pnlMaxOutAngleHi.Caption = gdblPhiEndHi

    pnlMinOutAnglePenalty.Caption = gdblPhiStartPenalty
    pnlMaxOutAnglePenalty.Caption = gdblPhiEndPenalty

    If gintPhiStartConstrainOn = 1 Then
        chkMinOutAngle.Value = True
        Call EnableControl(pnlMinOutAngleLow)
        Call EnableControl(pnlMinOutAngleHi)
        Call EnableControl(pnlMinOutAnglePenalty)
    ElseIf gintPhiStartConstrainOn = -2 Then
        chkMinOutAngle.Value = False
        Call DisableControl(pnlMinOutAngleLow, MEDIUMGRAY)
        Call DisableControl(pnlMinOutAngleHi, MEDIUMGRAY)
        Call DisableControl(pnlMinOutAnglePenalty, MEDIUMGRAY)
    End If

    If gintPhiEndConstrainOn = 1 Then
        chkMaxOutAngle.Value = True
        Call EnableControl(pnlMaxOutAngleLow)
        Call EnableControl(pnlMaxOutAngleHi)
        Call EnableControl(pnlMaxOutAnglePenalty)
    ElseIf gintPhiEndConstrainOn = -2 Then
        chkMaxOutAngle.Value = False
        Call DisableControl(pnlMaxOutAngleLow, MEDIUMGRAY)
        Call DisableControl(pnlMaxOutAngleHi, MEDIUMGRAY)
        Call DisableControl(pnlMaxOutAnglePenalty, MEDIUMGRAY)
```

CONFIG.FRM - chkMinOutAngle_Click

```
    End If

    pnlRandRotLow.Caption = gdblRndBaseRotLow
    pnlRandRotHi.Caption = gdblRndBaseRotHi
    pnlRandInputOffLow.Caption = gdblRndInputOffLow
    pnlRandInputOffHi.Caption = gdblRndInputOffHi
    pnlRandWeightOffLow.Caption = gdblRndWhtOffLow
    pnlRandWeightOffHi.Caption = gdblRndWhtOffHi
    pnlRandL1Low.Caption = gdblRndLen1Low
    pnlRandL1Hi.Caption = gdblRndLen1Hi
    pnlRandL2Low.Caption = gdblRndLen2Low
    pnlRandL2Hi.Caption = gdblRndLen2Hi
    pnlRandL3Low.Caption = gdblRndLen3Low
    pnlRandL3Hi.Caption = gdblRndLen3Hi
    pnlRandL4Low.Caption = gdblRndLen4Low
    pnlRandL4Hi.Caption = gdblRndLen4Hi
    pnlRandWeightLenLow.Caption = gdblRndWhtLenLow
    pnlRandWeightLenHi.Caption = gdblRndWhtLenHi

End Sub

---

Private Sub pnlMaxOutAngleHi_Click()

    On Error GoTo errortrap6

    Dim this_message As String
    Dim this_title As String
    Dim this_default As String

    this_message = "Enter a new value for the upper bound on the maximum output
     -->angle:"
    this_title = "Input Decimel Value..."
    this_default = pnlMaxOutAngleHi.Caption

popup6:
    temp = InputBox(this_message, this_title, this_default)
```

```
    If temp = "" Then Exit Sub

    temp = CDbl(temp)

    gdblPhiEndHi = temp

    pnlMaxOutAngleHi.Caption = gdblPhiEndHi

    'Debug.Print temp

    Exit Sub

errortrap6:

    status = MsgBox("Lets try that again with a number this time (or just hit cancel
     -->!)", 1 + 48 + 0 + 0, "Input Error!")

    If status = 2 Then Exit Sub
    If status = 1 Then GoTo popup6

End Sub
```

```
Private Sub pnlMaxOutAngleLow_Click()

    On Error GoTo errortrap2

    Dim this_message As String
    Dim this_title As String
    Dim this_default As String

    this_message = "Enter a new value for the lower bound on the maximum output
     -->angle:"
    this_title = "Input Decimel Value..."
    this_default = pnlMaxOutAngleLow.Caption
```



```
popup2:
    temp = InputBox(this_message, this_title, this_default)

    If temp = "" Then Exit Sub

    temp = CDbl(temp)

    gdblPhiEndLow = temp

    pnlMaxOutAngleLow.Caption = gdblPhiEndLow

    'Debug.Print temp

    Exit Sub

errortrap2:

    status = MsgBox("Lets try that again with a number this time (or just hit cancel
     -->!)", 1 + 48 + 0 + 0, "Input Error!")

    If status = 2 Then Exit Sub
    If status = 1 Then GoTo popup2

End Sub
```

```
Private Sub pnlMaxOutAnglePenalty_Click()

    On Error GoTo errortrap4

    Dim this_message As String
    Dim this_title As String
    Dim this_default As String

    this_message = "Enter a new value for the penalty assessed on exceeding the
     -->maximum output angle:"
    this_title = "Input Decimel Value..."
```

CONFIG.FRM - pnlMaxOutAnglePenalty_Click

```
    this_default = pnlMaxOutAnglePenalty.Caption

popup4:
    temp = InputBox(this_message, this_title, this_default)

    If temp = "" Then Exit Sub

    temp = CDbl(temp)

    gdblPhiEndPenalty = temp

    pnlMaxOutAnglePenalty.Caption = gdblPhiEndPenalty

    'Debug.Print temp

    Exit Sub

errortrap4:

    status = MsgBox("Lets try that again with a number this time (or just hit cancel
     -->!)", 1 + 48 + 0 + 0, "Input Error!")

    If status = 2 Then Exit Sub
    If status = 1 Then GoTo popup4


End Sub
```

```
Private Sub pnlMinOutAngleHi_Click()

    On Error GoTo errortrap5

    Dim this_message As String
    Dim this_title As String
    Dim this_default As String
```

CONFIG.FRM - pnlMaxOutAnglePenalty_Click

```
    this_message = "Enter a new value for the upper bound on the minimum output
     -->angle:"
    this_title = "Input Decimel Value..."
    this_default = pnlMinOutAngleHi.Caption

popup5:
    temp = InputBox(this_message, this_title, this_default)

    If temp = "" Then Exit Sub

    temp = CDbl(temp)

    gdblPhiStartHi = temp

    pnlMinOutAngleHi.Caption = gdblPhiStartHi

    'Debug.Print temp

    Exit Sub

errortrap5:

    status = MsgBox("Lets try that again with a number this time (or just hit cancel
     -->!)", 1 + 48 + 0 + 0, "Input Error!")

    If status = 2 Then Exit Sub
    If status = 1 Then GoTo popup5

End Sub
```

```
Private Sub pnlMinOutAngleLow_Click()

    On Error GoTo errortrap1

    Dim this_message As String
    Dim this_title As String
```

CONFIG.FRM - pnlMinOutAngleLow_Click

```
    Dim this_default As String

    this_message = "Enter a new value for the lower bound on the minimum output
     -->angle:"
    this_title = "Input Decimel Value..."
    this_default = pnlMinOutAngleLow.Caption

popup1:
    temp = InputBox(this_message, this_title, this_default)

    If temp = "" Then Exit Sub

    temp = CDbl(temp)

    gdblPhiStartLow = temp

    pnlMinOutAngleLow.Caption = gdblPhiStartLow

    'Debug.Print temp

    Exit Sub

errortrap1:

    status = MsgBox("Lets try that again with a number this time (or just hit cancel
     -->!)", 1 + 48 + 0 + 0, "Input Error!")

    If status = 2 Then Exit Sub
    If status = 1 Then GoTo popup1

End Sub
```

CONFIG.FRM - pnlMinOutAngleLow_Click

```
Private Sub pnlMinOutAnglePenalty_Click()

    On Error GoTo errortrap3

    Dim this_message As String
    Dim this_title As String
    Dim this_default As String

    this_message = "Enter a new value for the penalty assessed on exceeding the
     -->minimum output angle:"
    this_title = "Input Decimel Value..."
    this_default = pnlMinOutAnglePenalty.Caption

popup3:
    temp = InputBox(this_message, this_title, this_default)

    If temp = "" Then Exit Sub

    temp = CDbl(temp)

    gdblPhiStartPenalty = temp

    pnlMinOutAnglePenalty.Caption = gdblPhiStartPenalty

    'Debug.Print temp

    Exit Sub

errortrap3:

    status = MsgBox("Lets try that again with a number this time (or just hit cancel
     -->!)", 1 + 48 + 0 + 0, "Input Error!")

    If status = 2 Then Exit Sub
    If status = 1 Then GoTo popup3
```

```
End Sub

---

Private Sub pnlRandInputOffHi_Click()

   On Error GoTo errortrap14

   Dim this_message As String
   Dim this_title As String
   Dim this_default As String

   this_message = "Enter a new value for the upper bound on the input arm offset
    -->during a random jump:"
   this_title = "Input Decimel Value..."
   this_default = pnlRandInputOffHi.Caption

popup14:
   temp = InputBox(this_message, this_title, this_default)

   If temp = "" Then Exit Sub

   temp = CDbl(temp)

   gdblRndInputOffHi = temp

   pnlRandInputOffHi.Caption = gdblRndInputOffHi

   'Debug.Print temp

   Exit Sub

errortrap14:

   status = MsgBox("Lets try that again with a number this time (or just hit cancel
    -->!)", 1 + 48 + 0 + 0, "Input Error!")

   If status = 2 Then Exit Sub
```

CONFIG.FRM - pnlMinOutAnglePenalty_Click

```
   If status = 1 Then GoTo popup14

End Sub

---

Private Sub pnlRandInputOffLow_Click()

   On Error GoTo errortrap13

   Dim this_message As String
   Dim this_title As String
   Dim this_default As String

   this_message = "Enter a new value for the lower bound on the input arm offset
    -->during a random jump:"
   this_title = "Input Decimel Value..."
   this_default = pnlRandInputOffLow.Caption

popup13:
   temp = InputBox(this_message, this_title, this_default)

   If temp = "" Then Exit Sub

   temp = CDbl(temp)

   gdblRndInputOffLow = temp

   pnlRandInputOffLow.Caption = gdblRndInputOffLow

   'Debug.Print temp

   Exit Sub

errortrap13:

   status = MsgBox("Lets try that again with a number this time (or just hit cancel
    -->!)", 1 + 48 + 0 + 0, "Input Error!")
```

```
    If status = 2 Then Exit Sub
    If status = 1 Then GoTo popup13

End Sub

-----------------------------------------------------------------

Private Sub pnlRandL1Hi_Click()

    On Error GoTo errortrap18

    Dim this_message As String
    Dim this_title As String
    Dim this_default As String

    this_message = "Enter a new value for the upper bound on the length of link 1
     -->during a random jump:"
    this_title = "Input Decimel Value..."
    this_default = pnlRandL1Hi.Caption

popup18:
    temp = InputBox(this_message, this_title, this_default)

    If temp = "" Then Exit Sub

    temp = CDbl(temp)

    gdblRndLen1Hi = temp

    pnlRandL1Hi.Caption = gdblRndLen1Hi

    'Debug.Print temp

    Exit Sub

errortrap18:
```

CONFIG.FRM - pnlRandInputOffLow_Click

```
    status = MsgBox("Lets try that again with a number this time (or just hit cancel
     -->!)", 1 + 48 + 0 + 0, "Input Error!")

    If status = 2 Then Exit Sub
    If status = 1 Then GoTo popup18

End Sub

-----------------------------------------------------------------

Private Sub pnlRandL1Low_Click()

    On Error GoTo errortrap17

    Dim this_message As String
    Dim this_title As String
    Dim this_default As String

    this_message = "Enter a new value for the lower bound on the length of link 1
     -->during a random jump:"
    this_title = "Input Decimel Value..."
    this_default = pnlRandL1Low.Caption

popup17:
    temp = InputBox(this_message, this_title, this_default)

    If temp = "" Then Exit Sub

    temp = CDbl(temp)

    gdblRndLen1Low = temp

    pnlRandL1Low.Caption = gdblRndLen1Low

    'Debug.Print temp

    Exit Sub
```

```
errortrap17:

   status = MsgBox("Lets try that again with a number this time (or just hit cancel
    -->!)", 1 + 48 + 0 + 0, "Input Error!")

   If status = 2 Then Exit Sub
   If status = 1 Then GoTo popup17

End Sub
```

```
Private Sub pnlRandL2Hi_Click()

   On Error GoTo errortrap20

   Dim this_message As String
   Dim this_title As String
   Dim this_default As String

   this_message = "Enter a new value for the upper bound on the length of link 2
    -->during a random jump:"
   this_title = "Input Decimel Value..."
   this_default = pnlRandL2Hi.Caption

popup20:
   temp = InputBox(this_message, this_title, this_default)

   If temp = "" Then Exit Sub

   temp = CDbl(temp)

   gdblRndLen2Hi = temp

   pnlRandL2Hi.Caption = gdblRndLen2Hi

   'Debug.Print temp
```



```
   Exit Sub

errortrap20:

   status = MsgBox("Lets try that again with a number this time (or just hit cancel
    -->!)", 1 + 48 + 0 + 0, "Input Error!")

   If status = 2 Then Exit Sub
   If status = 1 Then GoTo popup20

End Sub
```

```
Private Sub pnlRandL2Low_Click()

   On Error GoTo errortrap19

   Dim this_message As String
   Dim this_title As String
   Dim this_default As String

   this_message = "Enter a new value for the lower bound on the length of link 2
    -->during a random jump:"
   this_title = "Input Decimel Value..."
   this_default = pnlRandL2Low.Caption

popup19:
   temp = InputBox(this_message, this_title, this_default)

   If temp = "" Then Exit Sub

   temp = CDbl(temp)

   gdblRndLen2Low = temp

   pnlRandL2Low.Caption = gdblRndLen2Low
```

```
    'Debug.Print temp

    Exit Sub

errortrap19:

    status = MsgBox("Lets try that again with a number this time (or just hit cancel
     -->!)", 1 + 48 + 0 + 0, "Input Error!")

    If status = 2 Then Exit Sub
    If status = 1 Then GoTo popup19

End Sub
```

```
Private Sub pnlRandL3Hi_Click()

    On Error GoTo errortrap22

    Dim this_message As String
    Dim this_title As String
    Dim this_default As String

    this_message = "Enter a new value for the upper bound on the length of link 3
     -->during a random jump:"
    this_title = "Input Decimel Value..."
    this_default = pnlRandL3Hi.Caption

popup22:
    temp = InputBox(this_message, this_title, this_default)

    If temp = "" Then Exit Sub

    temp = CDbl(temp)

    gdblRndLen3Hi = temp
```

CONFIG.FRM - pnlRandL2Low_Click

```
    pnlRandL3Hi.Caption = gdblRndLen3Hi

    'Debug.Print temp

    Exit Sub

errortrap22:

    status = MsgBox("Lets try that again with a number this time (or just hit cancel
     -->!)", 1 + 48 + 0 + 0, "Input Error!")

    If status = 2 Then Exit Sub
    If status = 1 Then GoTo popup22

End Sub
```

```
Private Sub pnlRandL3Low_Click()

    On Error GoTo errortrap21

    Dim this_message As String
    Dim this_title As String
    Dim this_default As String

    this_message = "Enter a new value for the lower bound on the length of link 3
     -->during a random jump:"
    this_title = "Input Decimel Value..."
    this_default = pnlRandL3Low.Caption

popup21:
    temp = InputBox(this_message, this_title, this_default)

    If temp = "" Then Exit Sub

    temp = CDbl(temp)
```

```
    gdblRndLen3Low = temp

    pnlRandL3Low.Caption = gdblRndLen3Low

    'Debug.Print temp

    Exit Sub

errortrap21:

    status = MsgBox("Lets try that again with a number this time (or just hit cancel
     -->!)", 1 + 48 + 0 + 0, "Input Error!")

    If status = 2 Then Exit Sub
    If status = 1 Then GoTo popup21

End Sub
```

```
Private Sub pnlRandL4Hi_Click()

    On Error GoTo errortrap24

    Dim this_message As String
    Dim this_title As String
    Dim this_default As String

    this_message = "Enter a new value for the upper bound on the length of link 4
     -->during a random jump:"
    this_title = "Input Decimel Value..."
    this_default = pnlRandL4Hi.Caption

popup24:
    temp = InputBox(this_message, this_title, this_default)

    If temp = "" Then Exit Sub
```

CONFIG.FRM - pnlRandL3Low_Click

```
    temp = CDbl(temp)

    gdblRndLen4Hi = temp

    pnlRandL4Hi.Caption = gdblRndLen4Hi
    'Debug.Print temp

    Exit Sub

errortrap24:

    status = MsgBox("Lets try that again with a number this time (or just hit cancel
     -->!)", 1 + 48 + 0 + 0, "Input Error!")

    If status = 2 Then Exit Sub
    If status = 1 Then GoTo popup24

End Sub
```

```
Private Sub pnlRandL4Low_Click()

    On Error GoTo errortrap23

    Dim this_message As String
    Dim this_title As String
    Dim this_default As String

    this_message = "Enter a new value for the lower bound on the length of link 4
     -->during a random jump:"
    this_title = "Input Decimel Value..."
    this_default = pnlRandL4Low.Caption

popup23:
    temp = InputBox(this_message, this_title, this_default)

    If temp = "" Then Exit Sub
```

```
    temp = CDbl(temp)

    gdblRndLen4Low = temp

    pnlRandL4Low.Caption = gdblRndLen4Low

    'Debug.Print temp

    Exit Sub

errortrap23:

    status = MsgBox("Lets try that again with a number this time (or just hit cancel
     -->!)", 1 + 48 + 0 + 0, "Input Error!")

    If status = 2 Then Exit Sub
    If status = 1 Then GoTo popup23

End Sub
```

```
Private Sub pnlRandRotHi_Click()

    On Error GoTo errortrap12

    Dim this_message As String
    Dim this_title As String
    Dim this_default As String

    this_message = "Enter a new value for the upper bound on the linkage base
     -->rotation during a random jump:"
    this_title = "Input Decimel Value..."
    this_default = pnlRandRotHi.Caption

popup12:
    temp = InputBox(this_message, this_title, this_default)
```

CONFIG.FRM - pnlRandL4Low_Click

```
    If temp = "" Then Exit Sub

    temp = CDbl(temp)

    gdblRndBaseRotHi = temp

    pnlRandRotHi.Caption = gdblRndBaseRotHi

    'Debug.Print temp

    Exit Sub

errortrap12:

    status = MsgBox("Lets try that again with a number this time (or just hit cancel
     -->!)", 1 + 48 + 0 + 0, "Input Error!")

    If status = 2 Then Exit Sub
    If status = 1 Then GoTo popup12

End Sub
```

```
Private Sub pnlRandRotLow_Click()

    On Error GoTo errortrap11

    Dim this_message As String
    Dim this_title As String
    Dim this_default As String

    this_message = "Enter a new value for the lower bound on the linkage base
     -->rotation during a random jump:"
    this_title = "Input Decimel Value..."
    this_default = pnlRandRotLow.Caption
```

```
popup11:
   temp = InputBox(this_message, this_title, this_default)

   If temp = "" Then Exit Sub

   temp = CDbl(temp)

   gdblRndBaseRotLow = temp

   pnlRandRotLow.Caption = gdblRndBaseRotLow

   'Debug.Print temp

   Exit Sub

errortrap11:

   status = MsgBox("Lets try that again with a number this time (or just hit cancel
    -->!)", 1 + 48 + 0 + 0, "Input Error!")

   If status = 2 Then Exit Sub
   If status = 1 Then GoTo popup11

End Sub
```

```
Private Sub pnlRandWeightLenHi_Click()

   On Error GoTo errortrap26

   Dim this_message As String
   Dim this_title As String
   Dim this_default As String

   this_message = "Enter a new value for the upper bound on the length of the
    -->weight arm during a random jump:"
   this_title = "Input Decimel Value..."
```

CONFIG.FRM - pnlRandRotLow_Click

```
   this_default = pnlRandWeightLenHi.Caption

popup26:
   temp = InputBox(this_message, this_title, this_default)

   If temp = "" Then Exit Sub

   temp = CDbl(temp)

   gdblRndWhtLenHi = temp

   pnlRandWeightLenHi.Caption = gdblRndWhtLenHi

   'Debug.Print temp

   Exit Sub

errortrap26:

   status = MsgBox("Lets try that again with a number this time (or just hit cancel
    -->!)", 1 + 48 + 0 + 0, "Input Error!")

   If status = 2 Then Exit Sub
   If status = 1 Then GoTo popup26

End Sub
```

```
Private Sub pnlRandWeightLenLow_Click()

   On Error GoTo errortrap25

   Dim this_message As String
   Dim this_title As String
   Dim this_default As String

   this_message = "Enter a new value for the lower bound on the length of the
```

```
    -->weight arm during a random jump:"
    this_title = "Input Decimel Value..."
    this_default = pnlRandWeightLenLow.Caption

popup25:
    temp = InputBox(this_message, this_title, this_default)

    If temp = "" Then Exit Sub

    temp = CDbl(temp)

    gdblRndWhtLenLow = temp

    pnlRandWeightLenLow.Caption = gdblRndWhtLenLow

    'Debug.Print temp

    Exit Sub

errortrap25:

    status = MsgBox("Lets try that again with a number this time (or just hit cancel
     -->!)", 1 + 48 + 0 + 0, "Input Error!")

    If status = 2 Then Exit Sub
    If status = 1 Then GoTo popup25

End Sub
```

```
Private Sub pnlRandWeightOffHi_Click()

    On Error GoTo errortrap16

    Dim this_message As String
    Dim this_title As String
    Dim this_default As String
```



```
    this_message = "Enter a new value for the upper bound on the weight arm
     -->offset during a random jump:"
    this_title = "Input Decimel Value..."
    this_default = pnlRandWeightOffHi.Caption

popup16:
    temp = InputBox(this_message, this_title, this_default)

    If temp = "" Then Exit Sub

    temp = CDbl(temp)

    gdblRndWhtOffHi = temp

    pnlRandWeightOffHi.Caption = gdblRndWhtOffHi

    'Debug.Print temp

    Exit Sub

errortrap16:

    status = MsgBox("Lets try that again with a number this time (or just hit cancel
     -->!)", 1 + 48 + 0 + 0, "Input Error!")

    If status = 2 Then Exit Sub
    If status = 1 Then GoTo popup16

End Sub
```

CONFIG.FRM - pnlRandWeightOffLow_Click

```
Private Sub pnlRandWeightOffLow_Click()

    On Error GoTo errortrap15

    Dim this_message As String
    Dim this_title As String
    Dim this_default As String

    this_message = "Enter a new value for the lower bound on the weight arm
     -->offset during a random jump:"
    this_title = "Input Decimel Value..."
    this_default = pnlRandWeightOffLow.Caption

popup15:
    temp = InputBox(this_message, this_title, this_default)

    If temp = "" Then Exit Sub

    temp = CDbl(temp)

    gdblRndWhtOffLow = temp

    pnlRandWeightOffLow.Caption = gdblRndWhtOffLow

    'Debug.Print temp

    Exit Sub

errortrap15:

    status = MsgBox("Lets try that again with a number this time (or just hit cancel
     -->!)", 1 + 48 + 0 + 0, "Input Error!")

    If status = 2 Then Exit Sub
    If status = 1 Then GoTo popup15
```

CONFIG.FRM - pnlRandWeightOffLow_Click

```
End Sub
```

ERRORTAB.FRM - Form_Load

```
Private Sub Form_Load()

   SystemMenu% = GetSystemMenu(frmErrorTable.hWnd, 0)
   Res% = RemoveMenu(SystemMenu%, 6, MF_BYPOSITION)
   Res% = RemoveMenu(SystemMenu%, 6, MF_BYPOSITION) 'also remove the
    -->separator line

   frmErrorTable.Width = frmMDI.Width / 4
   frmErrorTable.Height = frmMDI.Height / 2
   frmErrorTable.Top = frmAnalysisTextOutput.Top
   frmErrorTable.Left = frmAnalysisTextOutput.Left + frmAnalysisTextOutput.Width

End Sub
```

```
Private Sub Form_Resize()

   If WindowState <> 1 And ScaleHeight <> 0 Then
      txtErrorTable.Visible = False
      txtErrorTable.Height = ScaleHeight
      txtErrorTable.Width = ScaleWidth
      txtErrorTable.Visible = True
   End If

End Sub
```

```
Private Sub mnuCopySelection_Click()

   Clipboard.SetText frmErrorTable.txtErrorTable.SelText

End Sub
```

ERRORTAB.FRM - Form_Load

```
Private Sub chkCase_Click()
   gFindCase = chkCase.Value
End Sub
```

```
Private Sub cmdcancel_Click()
   gFindString = Text1.Text
   gFindCase = chkCase.Value
   Unload frmFind
End Sub
```

```
Private Sub cmdFind_Click()
   gFindString = Text1.Text
   FindIt
End Sub
```

```
Private Sub Form_Load()
   cmdFind.Enabled = False
   gFindDirection = 1
End Sub
```

```
Private Sub optDirection_Click(index As Integer)
   gFindDirection = index
End Sub
```

```
Private Sub Text1_Change()
   FirstTime = True

   If Text1.Text = "" Then
      cmdFind.Enabled = False
   Else
      cmdFind.Enabled = True
```

FIND.FRM - Text1_Change

```
    End If
End Sub
```

FIND.FRM - Text1_Change

```
Private Sub Form_Load()
    Dim i As Integer

    mnuFontName(0).Caption = Screen.Fonts(0)
    For i = 1 To Screen.FontCount - 1
        Load mnuFontName(i)
        mnuFontName(0).Caption = Screen.Fonts(i)
    Next
End Sub
```

```
Private Sub Form_QueryUnload(Cancel As Integer, UnloadMode As Integer)
    Dim Msg, Filename, NL
    Dim Response As Integer

    If FState(Me.Tag).Dirty Then
        Filename = Me.Caption
        NL = Chr$(10) & Chr$(13)
        Msg = "The text in [" & Filename & "] has changed."
        Msg = Msg & NL
        Msg = Msg & "Do you want to save the changes?"
        Response = MsgBox(Msg, 51, frmMDI.Caption)
        Select Case Response
        ' User selects Yes
        Case 6
            'Get the filename to save the file
            Filename = GetFileName()
            'If the user did notspecify a file name,
            'cancel the unload; otherwise, save it.
            If Filename = "" Then
                Cancel = True
            Else
                SaveFileAs (Filename)
            End If

        ' User selects No
```

NOTEPAD.FRM - Form_QueryUnload

```
        ' Ok to unload
        Case 7
            Cancel = False
        ' User selects Cancel
        ' Cancel the unload
        Case 2
            Cancel = True
        End Select
    End If
End Sub
```

```
Private Sub Form_Resize()
    If WindowState <> 1 And ScaleHeight <> 0 Then
        Text1.Visible = False
        Text1.Height = ScaleHeight
        Text1.Width = ScaleWidth
        Text1.Visible = True
    End If
End Sub
```

```
Private Sub Form_Unload(Cancel As Integer)
    FState(Me.Tag).Deleted = True

'Hide toolbar edit buttons if no frmTextViewEdit windows
    If Not AnyPadsLeft() Then
        frmMDI!imgCutButton.Visible = False
        frmMDI!imgCopyButton.Visible = False
        frmMDI!imgPasteButton.Visible = False
    End If
End Sub
```

NOTEPAD.FRM - Form_QueryUnload

```
Private Sub mnuECopy_Click()
    EditCopyProc
End Sub
```

```
Private Sub mnuECut_Click()
    EditCutProc
End Sub
```

```
Private Sub mnuEDelete_Click()
  ' If cursor is not at the end of the frmTextViewEdit.
  If Screen.ActiveControl.SelStart <> Len(Screen.ActiveControl.Text) Then
    ' If nothing is selected, extend selection by one.
    If Screen.ActiveControl.SelLength = 0 Then
      Screen.ActiveControl.SelLength = 1
      ' If cursor is on a blank line, extend selection by two.
      If Asc(Screen.ActiveControl.SelText) = 13 Then
        Screen.ActiveControl.SelLength = 2
      End If
    End If
    ' Delete selected text.
    Screen.ActiveControl.SelText = ""
  End If
End Sub
```

```
Private Sub mnuEPaste_Click()
    EditPasteProc
End Sub
```

```
Private Sub mnuESelectAll_Click()
   frmMDI.ActiveForm.Text1.SelStart = 0
   frmMDI.ActiveForm.Text1.SelLength = Len(frmMDI.ActiveForm.Text1.Text)
End Sub
```

```
Private Sub mnuETime_Click()
   Dim TimeStr As String, DateStr As String

   Text1.SelText = Now
End Sub
```

```
Private Sub mnuFClose_Click()
   Unload Me
End Sub
```

```
Private Sub mnuFExit_Click()
   ' Unloading the MDI form invokes the QueryUnload event
   ' for each child form, then the MDI form - before unloading
   ' the MDI form. Setting the Cancel argument to True in any of the
   ' QueryUnload events aborts the unload.

   Unload frmMDI


End Sub
```

```
Private Sub mnuFNew_Click()
   FileNew
End Sub
```

NOTEPAD.FRM - mnuESelectAll_Click

```
Private Sub mnuFontName_Click(index As Integer)
   Text1.FontName = mnuFontName(index).Caption
End Sub
```

```
Private Sub mnuFOpen_Click()
   FOpenProc
End Sub
```

```
Private Sub mnuFSave_Click()
   Dim Filename As String

   If Left(Me.Caption, 8) = "Untitled" Then
      ' The file hasn't been saved yet,
      ' get the filename, then call the
      ' save procedure
      Filename = GetFileName()
   Else
      ' The caption contains the name of the open file
      Filename = Me.Caption
   End If
   ' call the save procedure, if Filename = Empty then
   ' the user selected Cancel in the Save As dialog, otherwise
   ' save the file
   If Filename <> "" Then
      SaveFileAs Filename
   End If
End Sub
```

NOTEPAD.FRM - mnuFSaveAs_Click

```
Private Sub mnuFSaveAs_Click()
   Dim SaveFileName As String

   SaveFileName = GetFileName()
   If SaveFileName <> "" Then SaveFileAs (SaveFileName)
   ' Update the recent files menu
   UpdateFileMenu (SaveFileName)
End Sub
```

```
Private Sub mnuOptions_Click()
   mnuOToolbar.CHECKED = frmMDI!picToolbar.Visible
End Sub
```

```
Private Sub mnuOToolbar_Click()
   OptionsToolbarProc Me
End Sub
```

```
Private Sub mnuRecentFile_Click(index As Integer)
  OpenFile (mnuRecentFile(index).Caption)
  ' Update recent files list for new frmTextViewEdit.
  GetRecentFiles
End Sub
```

```
Private Sub mnuSFind_Click()
   If Me!Text1.SelText <> "" Then
      frmFind!Text1.Text = Me!Text1.SelText
   Else
      frmFind!Text1.Text = findstring
   End If
   gFirstTime = True
   frmFind.Show
End Sub
```

NOTEPAD.FRM - mnuFSaveAs_Click

```
Private Sub mnuSFindNext_Click()
   If Len(gFindString) > 0 Then
      FindIt
   Else
      mnuSFind_Click
   End If
End Sub
```

```
Private Sub mnuWArrange_Click()
   frmMDI.Arrange ARRANGE_ICONS
End Sub
```

```
Private Sub mnuWCascade_Click()
   frmMDI.Arrange CASCADE
End Sub
```

```
Private Sub mnuWTile_Click()
   frmMDI.Arrange TILE_HORIZONTAL
End Sub
```

```
Private Sub Text1_Change()
   FState(Me.Tag).Dirty = True
End Sub
```

```
Private Sub Text1_GotFocus()
   If frmFind.Visible Then
      frmFind.ZOrder 0
   End If
End Sub
```

G

I

L

M

S

T

U

W

References

1. Bagci, C., and Rieser, G. M., 1984, "Optimum Synthesis of Function Generators Involving Derivative Constraints", *Mechanism and Machine Theory*, Vol. 19, No. 1, pp. 157-164.
2. Bokelberg, E. H., and Gilmore, B. J., 1990, "A Kinematic Design Methodology for Exercise/Rehabilitation Machines Using Springs and Mechanical Advantage to Provide Variable Resistance", *Proceedings of the 21st Biennial Mechanisms Conference*, Chicago, IL, Sept. 16-19, pp. 279-286.
3. Dhande, S. G., and Chakraborty, J., 1973, "Analysis and Synthesis of Mechanical Error in Linkages -- A Stochastic Approach", *Journal of Engineering for Industry*, August, pp. 672-676.
4. Eason, E. D., and Fenton, R. G., 1974, "A Comparison of Numerical Optimization Methods for Engineering Design", *Journal of Engineering for Industry*, Vol. 89, Series B, pp. 196-200.
5. Fox, R. L., and Willmert, K. D., 1967, "Optimum Design of Curve-Generating Linkages with Inequality Constraints", *Journal of Engineering for Industry*, Vol. 89, Series B, pp. 144-152.
6. Fox, R. L., and Gupta, K. C., 1973, "Optimization Technology as Applied to Mechanism Design", *Journal of Engineering for Industry*, Vol. 95, Series B, pp. 657-663.

7. Garrett, R. E., and Hall, A. S., Jr., 1968, "Optimal Synthesis of Randomly Generated Linkages", *Journal of Engineering for Industry*, Vol. 90, Series B, pp. 475-480.

8. Gustavson, R. E., 1968, "Design of Planar Torque-Transmitting Four-Bar Linkage", ASME Paper 68-Mech-40.

9. Hall, Allen S., 1961, Kinematics and Linkage Design, Waveland Press, Prospect Heights, IL, p. 47.

10. Harrison,, J. Y., 1970, "Maximizing Human Power Output by Suitable Selection of Motion Cycle and Load", *Human Factors*, Vol. 12, No. 3, pp. 315-329.

11. Hartenberg, R. S., and Denavit, J., 1964, *Kinematic Synthesis of Linkages*, McGraw-Hill, New York.

12. Hooke, R., and Jeeves, T. A., 1961, ""Direct Search" Solution of Numerical and Statistical Problems", *Journal of the Association for Computing Machinery*, Vol. 8, No. 2, pp. 212-229.

13. Huang, C., and Roth, B., 1994, "Position-Force Synthesis of Closed-Loop Linkages", *Journal of Mechanical Design*, Vol. 116, March, pp. 155-162; 1992, *Proceedings of the 22nd Biennial Mechanisms Conference*, Scottsdale, AZ, Sept. 13-16, pp. 243-251.

14. Lieber, R. L., 1992, "The Production of Movement", *Skeletal Muscle Structure and Function: Implications for Rehabilitation and Sports Medicine*, Williams and Wilkins, Baltimore, MD, pp. 111-158.

15. Mabie, H. H., and Reinholtz, C. F., 1987, *Mechanisms and Dynamics of Machinery*, 4th Edition, John Wiley and Sons, New York.

16. Mallik, A. K., and Dhande, S. G., 1987, "Analysis and Synthesis of Mechanical Error in Path-Generating Linkages Using a Stochastic Approach", *Mechanism and Machine Theory*, Vol. 22, No. 2, pp. 115-123.

17. Midha, A., Turic, D. A., and Bosnik, J. R., 1984, "Creativity in the Classroom -- A Collection of Case Studies in Linkage Synthesis", *Mechanism and Machine Theory*, Vol. 20, No. 6, pp. 25-44.

18. Nathan, R. H., 1985, "A Constant Force Generation Mechanism", *Journal of Mechanisms, Transmissions, and Automation in Design*, Vol. 107, December, pp. 508-512.

19. Ogot, M. M., and Gilmore, B. J., 1991, "An Automated Procedure for the Maximization of the Mechanical Advantage of Planar Mechanisms", *Proceedings of the 17th Annual Design Automation Conference*, Miami, FL, Sept. 22-25, pp. 311-319.

20. Potter, T. E., Willmert, K. D., and Sathyamoorthy, M., 1992, "Nonlinear Optimal Design of Dynamic Mechanical Systems", *Proceedings of the 22nd Biennial Mechanisms Conference*, Scottsdale, AZ, Sept. 13-16, pp. 634-641.

21. Reinholtz, C. F., 1983, "Optimization of Spatial Mechanisms", Ph.D. Thesis., University of Florida, Gainesville, FL.

22. Reinholtz, C. F., Arun, V., and Williams, R. L. II, 1987, "A Reassessment of the Tasks of Kinematic Synthesis", *Proceedings of the 10th Applied Mechanisms Conference*, New Orleans, LA, Dec. 6-9.

23. Rigelman, G. A., and Kramer, S. N., 1988, "A Computer-Aided Design Technique for the Synthesis of Planar Four Bar Mechanisms Satisfying Specified Kinematic and Dynamic Conditions", *Journal of Mechanisms, Transmissions, and Automation in Design*, Vol. 110, September, pp. 263-268.

24. Root, R. R., and Ragsdell, K. M., 1976, "A Survey of Optimization Methods Applied to the Design of Mechanisms", *Journal of Engineering for Industry*, Vol. 98, Series B, pp. 1036-1041.

25. Sandor, G. N., and Erdman, A. G., 1984, *Advanced Mechanism Design: Analysis and Synthesis*, Vol. 2, Prentice-Hall.

26. Schneck, D. J., 1990, "Principles of Human Posture and Locomotion", *Engineering Principles of Physiologic Function*, New York University Press, New York, pp. 371-427.

27. Shoup, T. E., and Pelan, B. J., 1971, "Design of Four-Bar Mechanisms for Optimum Transmission Angle and Optimum Structural Error", *Proceedings of 2nd OSU Applied Mechanism Conference*, Oklahoma State University, Stillwater, OK, October 7-8, Paper No. 4.

28. Soper, R. R., Scardina, M. T., Tidwell, P., Reinholtz, C., and LoPresti, M. A., 1995, "Closed-Form Synthesis of Force-Generating Planar Four-Bar Linkages", *Proceedings of the 1995 Design Technical Conferences*, Boston, MA, Sept. 17-21, pp. 845-851.

29. Soper, R. R., 1995, "Synthesis of Planar Four-Link Mechanisms for Force Generation", MS Thesis, Virginia Polytechnic Institute and State University, Blacksburg, VA, August.

30. Starr, P. J., 1974, "Dynamic Synthesis of Linkages: An Emerging Field", *Proceedings of the 1974 Design Engineering Technical Conference*, Oct. 5-9, New York, NY, pp. 1-13.

31. Tao, D. C., 1964, "Design of a Four-Bar Linkage for the Transmission of Torque", *Applied Linkage Synthesis*, Addison-Wesley Co., Reading, MA, pp. 61-63.

32. Thompson, C. W., 1973, *Manual of Structural Kinesiology*, 7th Edition, C. V. Mosby Co., St. Louis.

33. Venkataraman, S. C., Kinzel, G. L., and Waldron, K. J., 1992, "Optimal Synthesis of Four-Bar Linkages for Four-Position Rigid-Body Guidance With Selective Tolerance Specifications", *Proceedings of the 22nd Biennial Mechanisms Conference*, Sept. 13-16, Scottsdale, AZ, pp. 652-659.

Vita

Michael Scardina was born on March 31, 1972 in Fairfax, VA. He is the son of Dr. John A. Scardina, a government engineer, and Joyce Lynn Scardina, a proud homemaker and has one older brother, David. Michael grew up in nearby Fairfax Station, VA, attending Fairview Elementary and Robinson High School. After high school graduation in 1990, Michael attended college at VA Tech in Blacksburg, VA, and worked every summer at the MITRE Corporation in McLean, VA. Mike graduated with honors in Mechanical Engineering in May, 1994. He returned to VA Tech in August, 1994 to pursue a Master's degree in Mechanical engineering which he completed with honors in October, 1996.

Michael currently lives in Arlington, VA and is employed as an Engineer at MPR Associates, Inc., a consulting and professional engineering firm located in Alexandria, VA.