ABSTRACT

OPTIMAL SYNTHESIS OF A PLANAR FOUR-BAR FUNCTION GENERATOR BASED ON INDIRECT MEASURE OF STRUCTURAL ERROR

by
Robert A. Ellis III

A novel technique for optimal synthesis of a planar four-bar function generator is presented. An indirect measure of structural error is used to formulate an objective function which, along with its derivatives, is straightforward to evaluate and results in an elegant and robust algorithm. Lagrange multipliers are used to incorporate the loop closure constraint into the objective function.

In this thesis, the formulation of the objective function and the minimization algorithm are described, examples are presented, and the virtues and shortcomings of the algorithm are discussed.

OPTIMAL SYNTHESIS OF A PLANAR FOUR-BAR FUNCTION GENERATOR BASED ON INDIRECT MEASURE OF STRUCTURAL ERROR

by
Robert A. Ellis III

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Mechanical Engineering

Department of Mechanical Engineering

January 1998

APPROVAL PAGE

OPTIMAL SYNTHESIS OF A PLANAR FOUR-BAR FUNCTION GENERATOR BASED ON INDIRECT MEASURE OF STRUCTURAL ERROR

Robert A. Ellis III

[illegible]	12/10/97
Dr. Ian S. Fischer, Thesis Advisor	Date
Associate Professor of Mechanical Engineering, NJIT	
[illegible]	12/10/97
Dr. Zhiming Ji	Date
Associate Professor of Mechanical Engineering, NJIT	
[illegible]	12/10/97
Dr. Tina Chu	Date
Assistant Professor of Mechanical Engineering, NJIT	

BIOGRAPHICAL SKETCH

Author: Robert A. Ellis III

Degree: Master of Science in Mechanical Engineering

Date: January 1998

Date of Birth:

Place of Birth:

Undergraduate and Graduate Education:

- Master of Science in Mechanical Engineering,
  New Jersey Institute of Technology, Newark, NJ, 1998

- Bachelor of Science in Mechanical and Aerospace Engineering,
  Princeton University, Princeton, NJ, 1979

Major: Mechanical Engineering

This thesis is dedicated to my parents.

ACKNOWLEDGMENT

I would like to express my deep appreciation to Dr. Ian Fischer, my thesis advisor, for his encouragement, insight, and advice throughout the course of this research. I would also like to thank Dr. Zhiming Ji and Dr. Tina Chu for serving on my committee.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Continued)

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES
(Continued)

Figure **Page**

CHAPTER 1

INTRODUCTION

1.1 Summary of Research

Four-bar linkages are used in many different types of machinery. The art of synthesizing a linkage to generate a specified motion, path, or function has advanced from graphical techniques to analytical methods developed after 1950. Existing techniques rely on satisfying the required motion at a limited number of points, and approximating it elsewhere. A technique which would enable a large number of points to be considered, and which would result in an optimal linkage, would be an improvement on existing methods. The research presented here describes an optimal synthesis algorithm based on a novel formulation of the problem.

The relationship between the input and output-shaft rotation in a four-bar linkage, such as that shown in figure 1.1, is determined by the geometry of the mechanism. Given the link lengths a_1, a_2, a_3, a_4, and the starting input and output-shaft angles θ_{10} and θ_{40}, the input-shaft can be rotated through an angle ϕ to its new angle θ_1, and the resulting output-shaft rotation ψ, and position θ_4 can be computed.

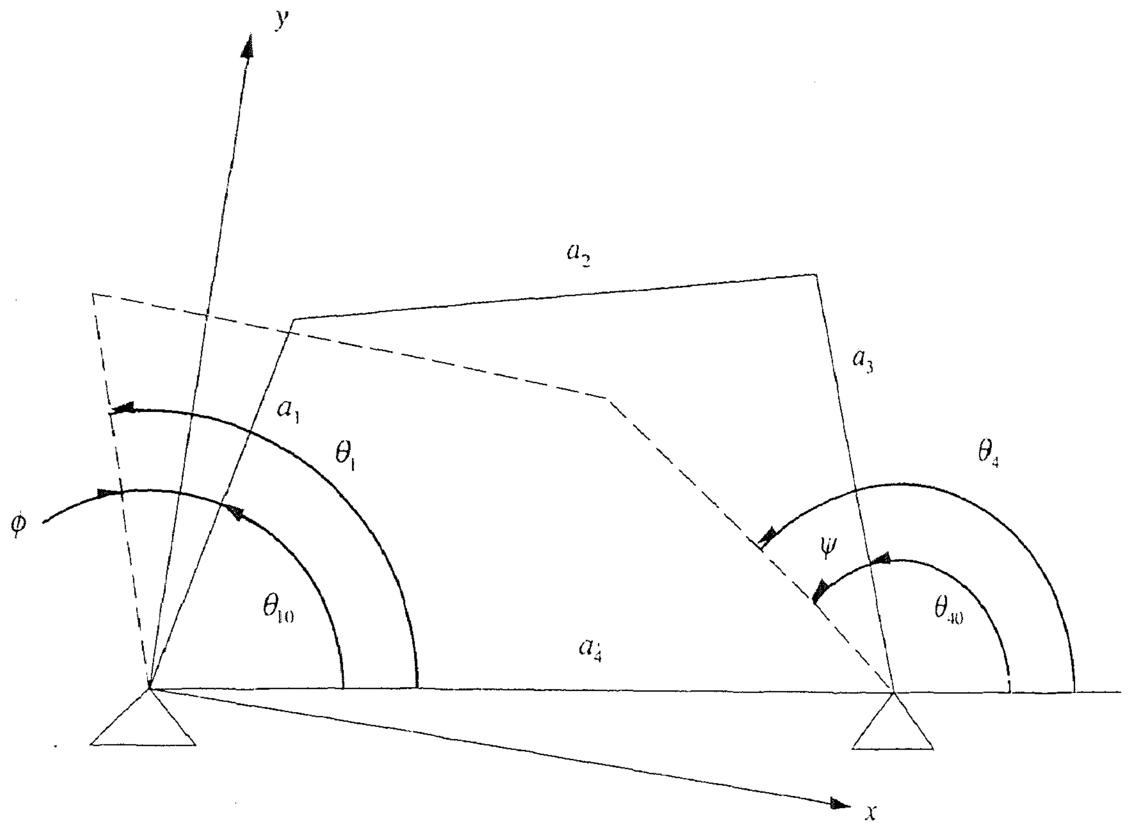

Figure 1.1 A Planar Four-Bar Linkage

Given an appropriate mechanism geometry, the relationship between the input-and-output-shaft rotations will approximate a specified function. In other words, the function $y = f(x)$ can be mapped onto $\psi = g(\phi)$. A mechanism designed for this purpose is a function generator. Calculating the output-shaft rotation, ψ, as a function of the input-shaft rotation ϕ is straightforward once the link lengths are specified. Less straightforward, but more useful, is the solution of the inverse problem: given a prescribed relation between the input-and-output-shaft rotations, determine the link lengths that will generate it.

This inverse problem is known as mechanism synthesis. Precision synthesis techniques, which produce a mechanism that matches an ideal function at five or fewer pairs of input and output-shaft angles, or precision points, have typically been used to solve this problem. Another approach is to synthesize a mechanism that approximates a function closely at a large number of points. This class of synthesis techniques is known as optimal synthesis.

In this thesis, a new technique for optimal synthesis of four-bar function generators, using a unique formulation of the objective function, is described. The objective function is defined, and a minimization algorithm is presented. A computer program, which implements this algorithm, is described. Several examples of optimal synthesis, using this algorithm, are presented.

1.2 Definitions

A planar four-bar linkage is a mechanism which consists of four links (or bars), connected to each other by revolute joints to form a closed loop. (figure 1.1) One of the links, link 4, known as the frame, or ground, is fixed. Link 1, called the input crank, typically has its motion prescribed. Link 2 is the coupler, and link 3 is the output crank. The input and output cranks are connected to the frame by revolute joints. The moving ends of the cranks are connected to the coupler by revolute joints. The orientation of the input crank relative to the frame is called the input angle, and the orientation of the output crank is called the output angle.

There are three basic tasks performed by linkages: path generation, motion generation, and function generation. In path generation, (see figure 1.2) a point on the coupler follows a specified path.

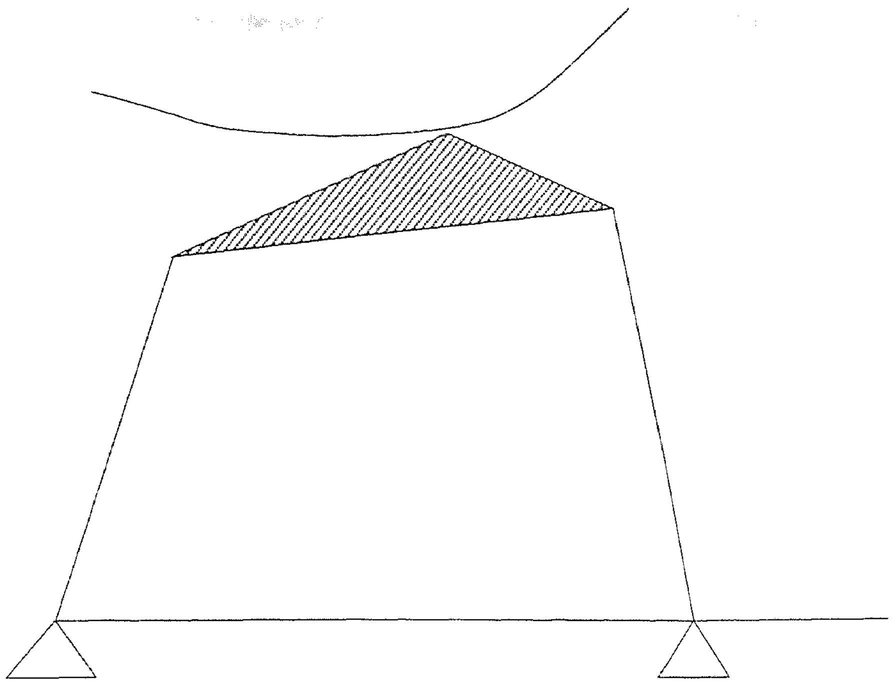

Figure 1.2 Path Generation

Motion generation also requires a point on the coupler to follow a path, but the orientation of the coupler is specified along the path (figure 1.3).

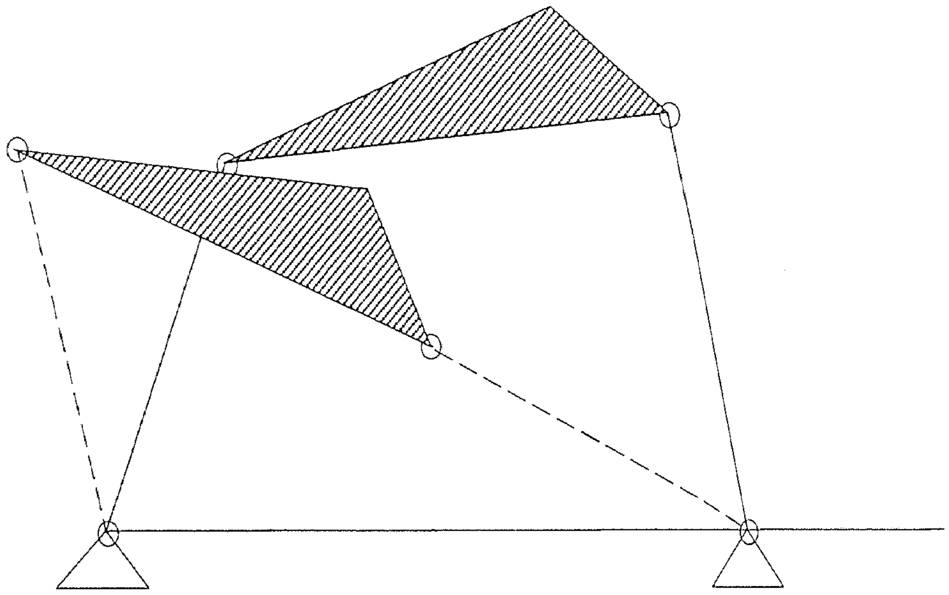

Figure 1.3 Motion Generation

In a function generator, the rotation of the output shaft is a function of the rotation of the input-shaft (figure 1.4).

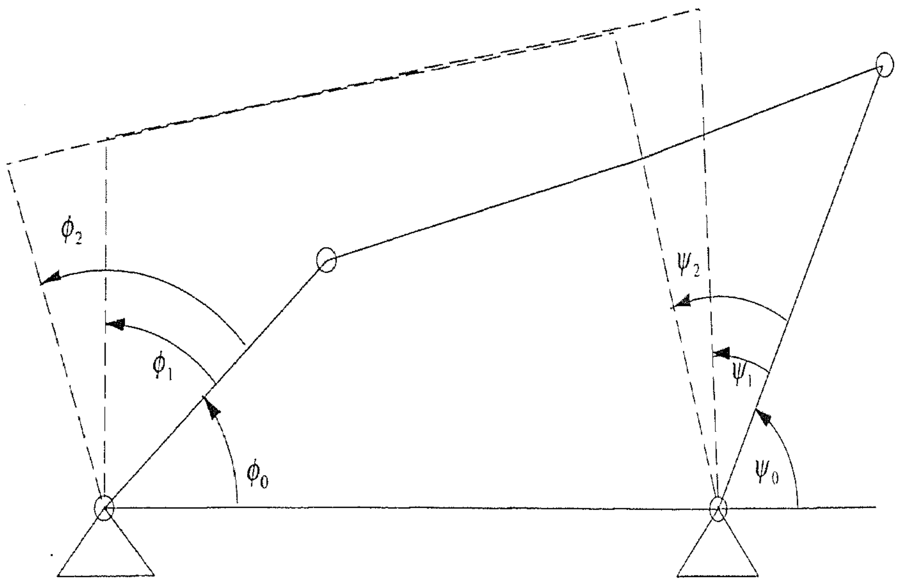

Figure 1.4 A Four-Bar Function Generator

The research reported in this thesis is concerned only with function generation.

Each of the links shown in figure 1.1 can be represented as a vector in the x-y plane. The linkage is defined by specifying the magnitude and orientation of each vector. This set of values, expressed in vector form as $[a_1,a_2,a_3,a_4,\theta_1,\theta_2,\theta_3,\theta_4]$, is known as the vector of design parameters.

There are, in fact, less than eight independent design parameters. The orientation of the frame relative to the coordinate system is not a design parameter because it does not affect the motion of the links relative to the frame. One of the coordinate system axes is usually placed on the frame link. The requirement that the four links form a closed loop results in one vector equation, and further reduces the number of independent design parameters by two, to five.

The relationship between the input angle and output angle of a mechanism such as that in figure 1.1 is determined by the design parameters. If these parameters are properly chosen, the relationship can approximate a desired function. The desired function is called the ideal function and the function actually produced by the linkage is called the generated function. The difference between the ideal function and the generated function is known as structural error. It is straightforward to calculate the progression of a linkage through its range of motion, given the design parameters. The inverse problem, synthesis, is not as simple and is defined as follows: Find a set of design parameters that will result in a desired relationship between the input and output angles of a four bar linkage.

Techniques have been developed for the synthesis of linkages that match an ideal function at five or fewer positions. These positions are known as precision points, and the techniques are referred to as precision synthesis. A linkage synthesized by this method will deviate from the desired progression at positions other than the precision points. This deviation is known as the structural error of the linkage.

Another, more recent, synthesis technique is to search for a set of design parameters that minimizes a special function of the structural error at a large number of positions, and thus approximates the ideal function. This is referred to as approximate, or optimal, synthesis. The function of the structural error, which is subject to special requirements, is known as the objective function. The positions where the ideal function is to be approximated are called the synthesis points.

Consider the function generator shown in figure 1.1. The input and output crank lengths are a_1 and a_3, respectively. The coupler length is a_2 and the frame length is a_4. The starting input and output-shaft angles are θ_{10} and θ_{40}, and the rotations of the input-and-output-shafts from the starting angles are ϕ and ψ. The desired progression of this linkage is $\psi = g(\phi)$, and a set of synthesis points $\{(\phi_1, \psi_1), \ldots, (\phi_k, \psi_k)\}$ is defined. The starting angles of the input and output-shafts are specified, as are their ranges of motion.

The goal of this research is to develop a synthesis technique that minimizes some norm of the structural error at all of the synthesis points.

1.3 Background

The techniques that already exist for the synthesis of function generators that satisfy $\psi = g(\phi)$ are limited to five or fewer values of input angle ϕ. Graphical techniques[1] have been used since the 19th century. Sandor and Erdman[2] have presented a graphical method for synthesizing a function generator, with the frame and output crank arbitrarily specified and using three pairs of input-and-output-shaft angles. Mechanisms synthesized by this technique satisfy the desired functional relationship at three points, and approximate it elsewhere. Another graphical technique reported by Sandor and Erdman, the overlay method, can be used for more than three precision points, but it may not provide an exact solution at any point.

In 1955, Freudenstein[3] presented an analytical technique for the synthesis of a function generator. The length of the frame link, and the starting input and output-shaft angles (see figure 1.5) were specified, and the remaining link lengths were unknowns. To obtain a solution, the links are treated as complex vectors, with the frame lying on the positive real axis. (figure 1.5)

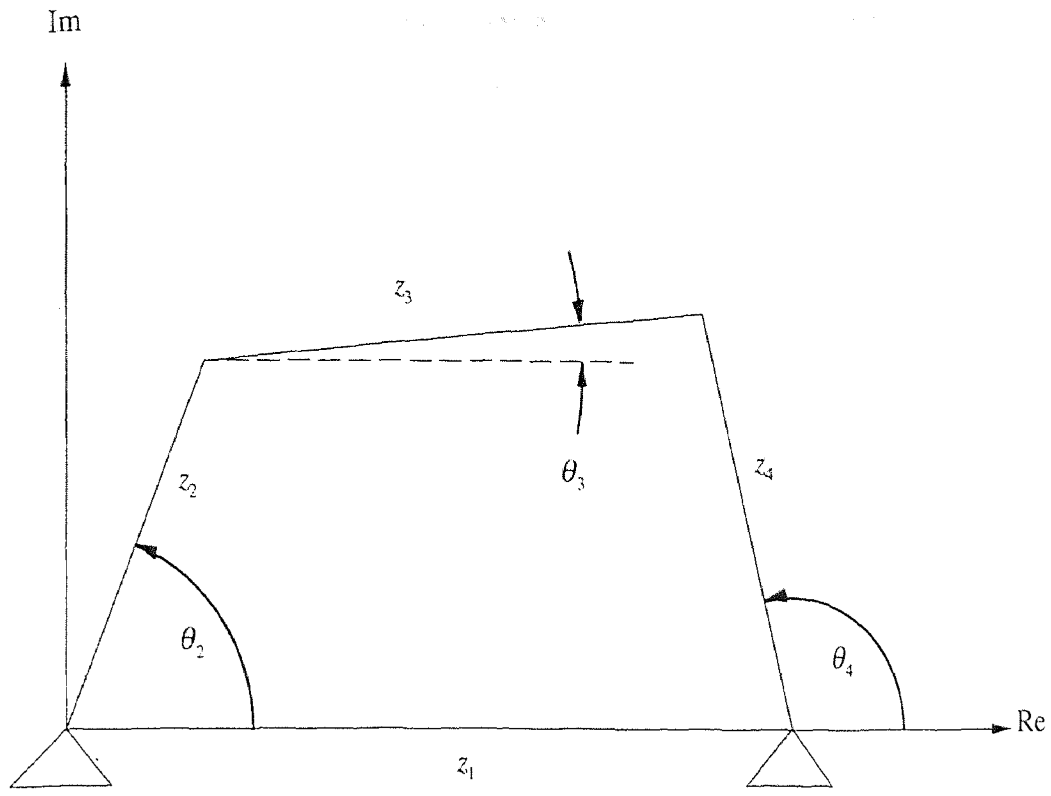

Figure 1.5 Representation of a Four-Bar Linkage in the Complex Plane

Recognizing that the links must form a closed loop, the complex vector equation

$$z_3 e^{i\theta_3} = z_1 - z_2 e^{i\theta_2} + z_4 e^{i\theta_4} \tag{1.1}$$

is obtained. Multiplying each side of the equation by its complex conjugate, and some additional algebra, results in the scalar equation

$$z_3^2 = z_1^2 + z_2^2 + z_4^2 - 2z_1 z_2 \cos\theta_2 + 2z_1 z_4 \cos\theta_4 - 2z_2 z_4 \cos(\theta_2 - \theta_4) \tag{1.2}$$

known as Freudenstein's equation. Because the mechanism is a function generator, the angles θ_2 and θ_4 are specified in pairs. Therefore, the quantities z_2, z_3, and z_4 are the only unknowns. It is apparent that for each (θ_2, θ_4) pair, a new and independent version of Freudenstein's equation will be obtained. Since there are only three unknowns, it follows that a solution is impossible if more than three (θ_2, θ_4) pairs are specified.

This limitation is a fundamental problem inherent in all precision synthesis techniques: it is not, in general, possible to synthesize a mechanism using a continuous path. One can only synthesize a mechanism that matches the path exactly at a small number of points, and approximates it elsewhere.

It is sometimes preferable to approximate a function closely at a large number of points than to match it exactly at five points or less. Many different functions can satisfy the same set of five or fewer points, as shown in figure 1.6.

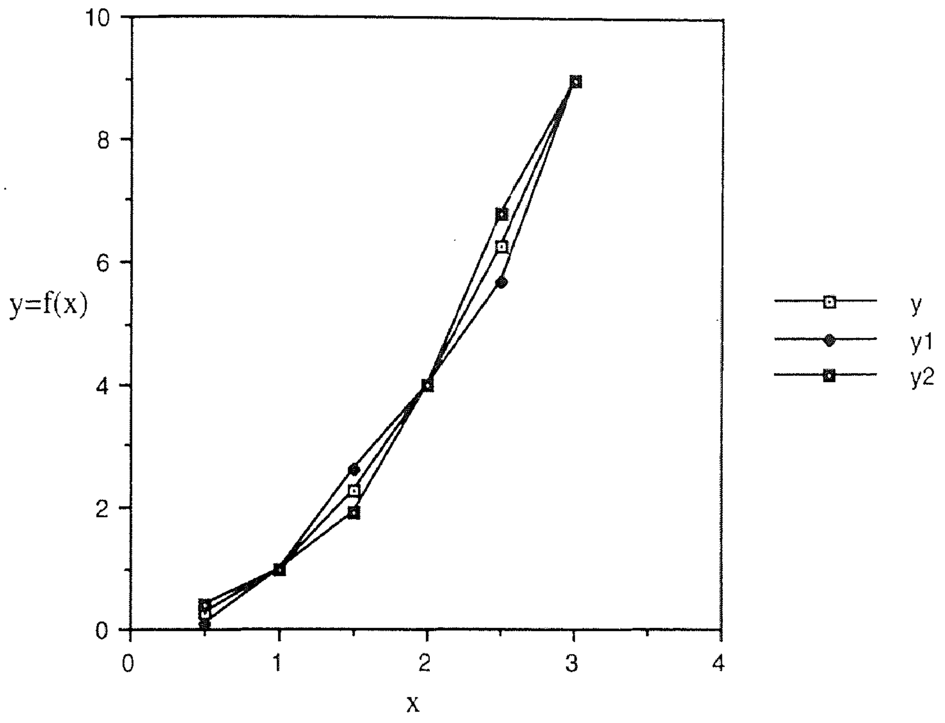

Figure 1.6 Different functions which pass through the same three precision points;

Using a large number of points for synthesis eliminates this ambiguity. The purpose of this research is to develop a synthesis technique that allows the use of an arbitrarily large number of points over the range of the function to be generated, thus addressing the biggest deficiency of precision synthesis techniques.

CHAPTER 2

PREVIOUS WORK

2.1 Precision Techniques

As stated earlier, the function generated by a linkage can, in general, only match the desired function at five or fewer points. This becomes evident upon examination of Freudenstein's equation

$$z_3^2 = z_1^2 + z_2^2 + z_4^2 - 2z_1z_2\cos\theta_2 + 2z_1z_4\cos\theta_4 - 2z_2z_4\cos(\theta_2 - \theta_4). \quad (2.1)$$

In a function generator, the input and output-shaft angles θ_2, θ_4 and the length of the frame link z_3 are specified, leaving vectors z_2, z_3, and z_4 as the three unknowns in the equation. Since the equation is applied at each precision point, the use of more than three precision points results in an over constrained set of equations. If the starting input and output angles θ_{20} and θ_{40} are used as design parameters, with

$$\theta_2 = \theta_{20} + \phi$$

and (2.2)

$$\theta_4 = \theta_{40} + \psi,$$

there are five unknowns in the equation, and up to five precision points can be used.

The first analytical synthesis techniques used five or fewer precision points, located within the prescribed function range, usually spaced according to the Chebyshev technique. Structural error was analyzed after synthesis, and the spacing of the precision points would be varied if the magnitude of the error was unacceptable. Freudenstein[4], in 1959, presented a systematic technique for varying the precision point spacing in order to minimize structural error.

While the preceding techniques can produce linkages having extremely low structural error, precision synthesis methods cannot be used with large numbers of synthesis points. And there are many instances where such capability is desirable, and perhaps necessary. Thus, a major goal of research in mechanism synthesis has been the development of approximate synthesis techniques. Here the goal is to obtain a mechanism that generates a motion, path or function that is acceptably close to a desired progression at an arbitrarily large number of points. An advantage of using a large number of points is that the deviation of the generated motion, path or function from the desired one is minimized. The loss of accuracy at the precision points is not critical, as manufacturing tolerances in a real linkage would introduce some error at those positions anyway.

2.2 Approximate Techniques

Early approximate synthesis techniques attempted to find a set of design parameters which would minimize the sum of squares of deviations at the synthesis points. Sarkisyan, Gupta and Roth[5], in 1973, presented a technique for least square approximation of path generation, for the special cases of circular or straight paths. The approximation minimized the deviation of the sum of the square of the circle radius, as shown in figure 2.1.

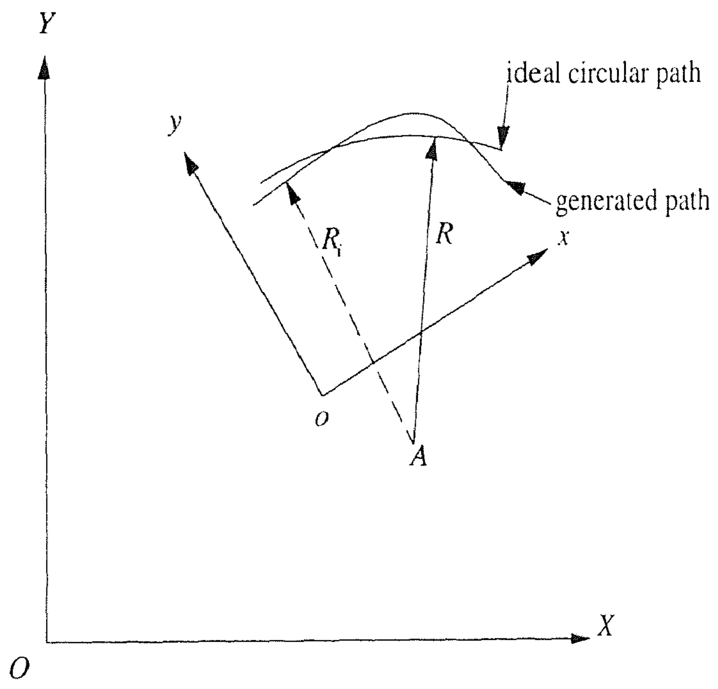

Figure 2.1 Deviation from an ideal circular path

Referring to figure 2.1, at each position i, the generated path differs from the ideal path by an amount

$$\Delta_i = R_i - R \tag{2.3}$$

The objective function is defined as

$$S = \sum_{i=1}^{k} \Delta^2_{qi}\,, \tag{2.4}$$

where

$$\Delta_{qi} = R_i^2 - R^2\,. \tag{2.5}$$

In minimizing the objective function, it was assumed that Δ_i was small enough that its square could be neglected. This assumption, that the starting point for an optimization is a linkage with a small structural error, was a significant limitation. Algorithms for determining least square circle-point curves, analogous to the circle-point curves in Burmester theory, were developed.

In 1975, Gupta and Roth[6] presented an approximate synthesis technique for circular and straight paths, which used a more general norm of the errors as an objective function. In this method, the ideal function is $f(x)$, and it is approximated by a function $F(\mathbf{A},x)$ where x is the same scalar as in $f(x)$, and $\mathbf{A}$ is a vector of parameters. The error function $G(\mathbf{A},x)$ can then be defined as

$$G(\mathbf{A},x) = f(x) - F(\mathbf{A},x)\,. \tag{2.6}$$

The L$_p$-norm of the error function $G(\mathbf{A},x)$ is then defined as

$$\left\| G(\mathbf{A},x) \right\|_p = \left[\int_{x_1}^{x_2} \left| G(\mathbf{A},x) \right|^p dx \right]^{1/p}, \quad p \geq 1. \tag{2.7}$$

In these equations, for the case of circular path generation, x would be replaced by the radius of the circular path. For certain special cases of synthesis, closed form solutions were obtained by this technique.

Sandor and others applied techniques of mathematical programming to the problem of optimal synthesis. A 1975 paper by Alizade, Novruzbekov and Sandor[7] used the penalty function approach to convert a problem of minimization with equality and inequality constraints into an unconstrained minimization problem. An objective function based on the sum of the squares of the errors was employed. Inequality constraints, such as maximum permissible transmission angles, were incorporated, as was Freudenstein's equation, an equality constraint. A 1975 paper by Alizade, Rao and Sandor[8] extended these principles to spatial, and other types of planar, mechanisms. In Selective Precision Synthesis[9], developed by Kramer and Sandor, an arbitrary accuracy neighborhood is defined around each synthesis point, and nonlinear programming techniques are used to find a mechanism whose output passes through each accuracy neighborhood. (figure 2.2)

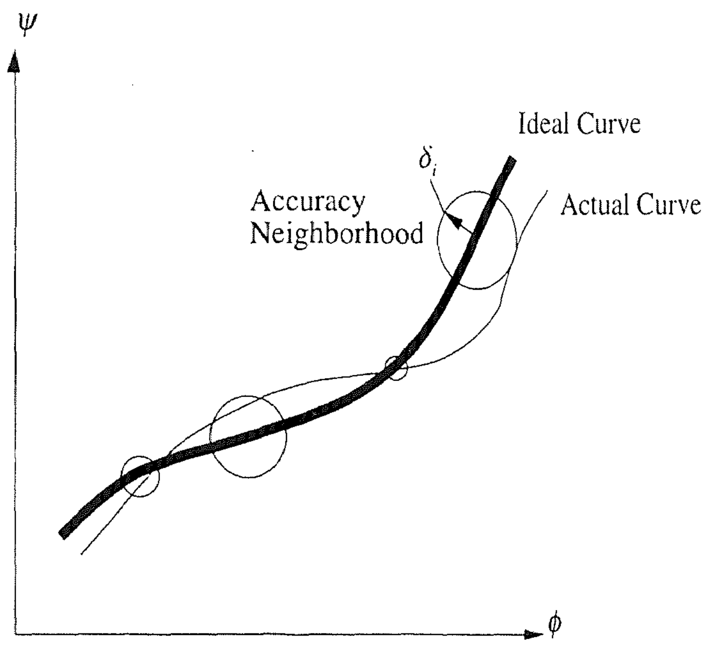

Figure 2.2 Varying Accuracy Neighborhoods

This synthesis technique allows the precision to be varied over the travel of the mechanism. Reducing precision in areas where it is not needed can enable a useful mechanism to be synthesized where other techniques would be unduly restrictive.

Pugh[10], in 1984, applied pareto optimization techniques to the optimization of four-bar function generators, with the objective function based on the structural error and the departure of the transmission angle from 90 degrees. In this technique, a systematic search of the entire solution domain was undertaken, with the objective function evaluated at ten equally spaced positions of the linkage.

Aviles et. al[11], in 1985, proposed a novel technique in which the same objective function could be applied to path generation, motion generation, and function generation. In this technique, the mechanism is forced to satisfy the ideal motion, and all of the links are permitted to deform so as to accommodate this prescribed motion. The objective function is the sum of the squares of the link deformations. Because all of the links are able to deform, there are an infinite number of deformed configurations at each synthesis point, and in is not possible to directly evaluate a unique value of structural error at a synthesis point. A quasi-Newton optimization method was used.

More recently, kinematic mapping techniques have been employed. Kinematic mapping involves mapping a two-dimensional motion, which consists of two translational components and one rotational component, into a curve in four-dimensional image space. This desired image curve is to be approximated by a curve, which is a function of the design parameters. A successful algorithm will synthesize a mechanism whose image curve is acceptably close to the desired image curve. Ravani and Roth[12] employed this technique to the synthesis of motion generators, with the generated image curve defined as the intersection of two surfaces that are functions of the design parameters. Wu and Fischer[13] used quaternion algebra to define the intersecting surfaces as constraint manifolds, which are a function of the design parameters. A difficulty with kinematic mapping is that calculation of the distance between the desired and generated image curves is tedious. Also, when evaluating the objective function as part of the iterative process, the potential for the algorithm to toggle between two valid configurations of the mechanism can result in a failure to converge.

It should be noted that, although some of these synthesis techniques were developed for motion or path generation, they are relevant to function generation as well. Function generation is simply a special case of path generation with timing. Motion generation, like path generation with timing, specifies two translational displacements and one angular displacement at each precision point. Thus, synthesis techniques for motion and path generation are relevant to function generation.

At present, no approximate synthesis method has emerged as a standard to be used by design engineers. Precision synthesis techniques are still the only ones that can be depended on to create a mechanism for a specific task.

The existing approximate synthesis techniques all attempt to minimize some norm of the structural error, measured directly. The technique here uses a different approach. An indirect measure of the structural error is used in order to obtain a better mathematical formulation of the objective function. The structural error of mechanisms synthesized by this technique will be analyzed in order to verify that the use of an indirect measure of error results in an acceptable level of accuracy.

CHAPTER 3

ANALYSIS

3.1 Formulation of the Objective Function

3.1.1 Review of Optimal Synthesis

An optimally synthesized function generator is one that gives the best possible approximation of the ideal function over the specified input range. The number of synthesis points, or points at which the deviation from the ideal function is measured, is greater than the maximum number of precision points.

The quality of the approximation of the ideal function is determined by the value of some function of the deviations from the ideal function at the synthesis points. The deviation at a synthesis point is known as the structural error. An objective function can be defined somewhat arbitrarily, but it must increase monotonically with the magnitude of the structural error.

Consider a four-bar function generator which approximates a function $y = f(x)$, $x_a \le x < x_b$. This function is mapped onto the (ϕ, ψ) plane of input and output-shaft rotations as $\psi = g(\phi)$, $\phi_a \le \phi < \phi_b$. At the synthesis points, where the input-shaft rotations are $\{\phi_1, \phi_2, \ldots, \phi_k\}$, the output-shaft rotations generated by the mechanism will be $\{\psi_1, \psi_2, \ldots, \psi_k\}$. The difference at each synthesis point between the generated output-shaft rotation and the ideal output-shaft rotation is

$$\{(\psi_1 - g(\phi_1)), (\psi_2 - g(\phi_2)), \ldots, (\psi_k - g(\phi_k))\} = \{\Delta_1, \Delta_2, \ldots, \Delta_k\} \tag{3.1}$$

At the ith synthesis point, the structural error is Δ_i. In optimization problems it is common to use the sum of the squares of the structural errors as the objective function. Thus, when there are k synthesis points, the objective function U would be

$$U = \sum_{i=1}^{k} \Delta^2_i \tag{3.2}$$

There are numerous other permissible objective functions, for example:

$$U = \sum_{i=1}^{n} \Delta_i^4 ,$$

$$U = \sum_{i=1}^{n} |\Delta_i| ,$$

$$U = \sum_{i=1}^{n} \max(|\Delta_i|) , \tag{3.3}$$

$$U = \sum_{i=1}^{n} \log(\Delta_i^2) .$$

The sum of the squares of the structural errors is used most often because some of the more common optimization routines perform best when the objective function defines a quadratic surface.

3.1.2 Measurement of the Structural Error

3.1.2.1 Direct Measurement: In the previous section, the structural error was defined as the difference between actual and ideal output-shaft rotations. Its measurement is straightforward. For a given input-shaft rotation ϕ_i, the output-shaft rotation ψ_i can be obtained directly if the configuration of the mechanism is known.

Consider the function generator in figure 3.1. Given the link lengths $a_1 = 4$, $a_2 = 8$, $a_3 = 6$, and $a_4 = 10$, and a starting input-shaft angle ϕ_0 of 60 degrees, the output-shaft angle ψ_0 can be shown to be 93.89 degrees.

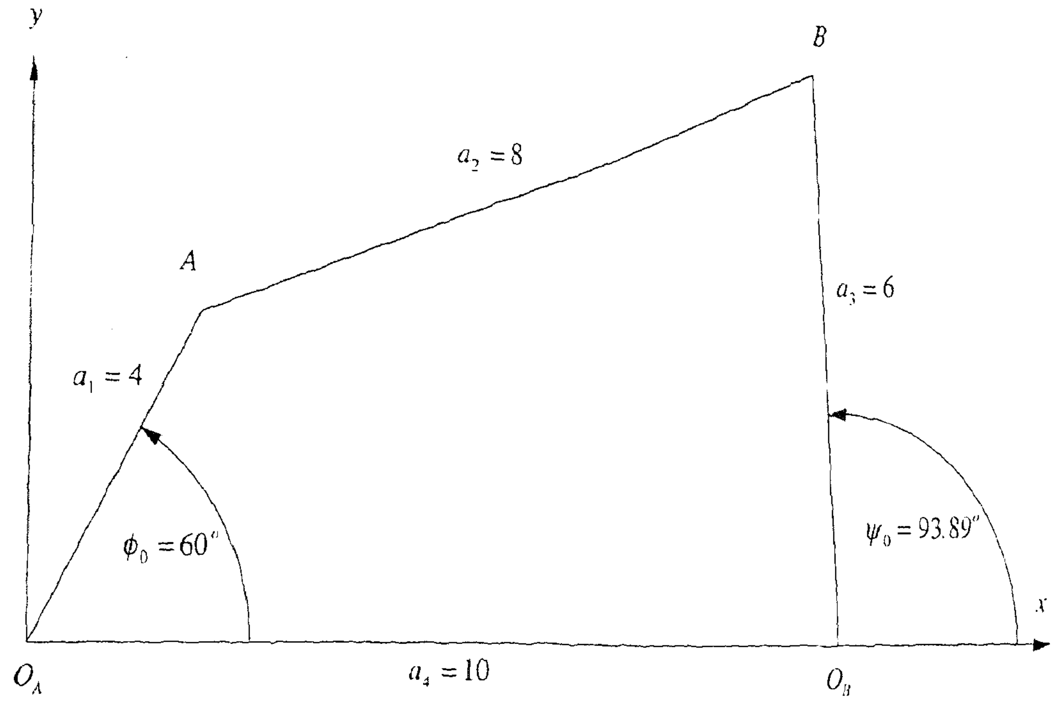

Figure 3.1 A Four-Bar Function Generator

When the input-shaft is rotated through the angle $\phi_1 = 10^\circ$, the mechanism has the configuration shown in figure 3.2, and we have $\psi_1 = 98.93^\circ - 93.89^\circ = 5.04^\circ$. If the ideal value of ψ_1 had been 15.0 degrees, the structural error at that point would be 10.0 degrees.

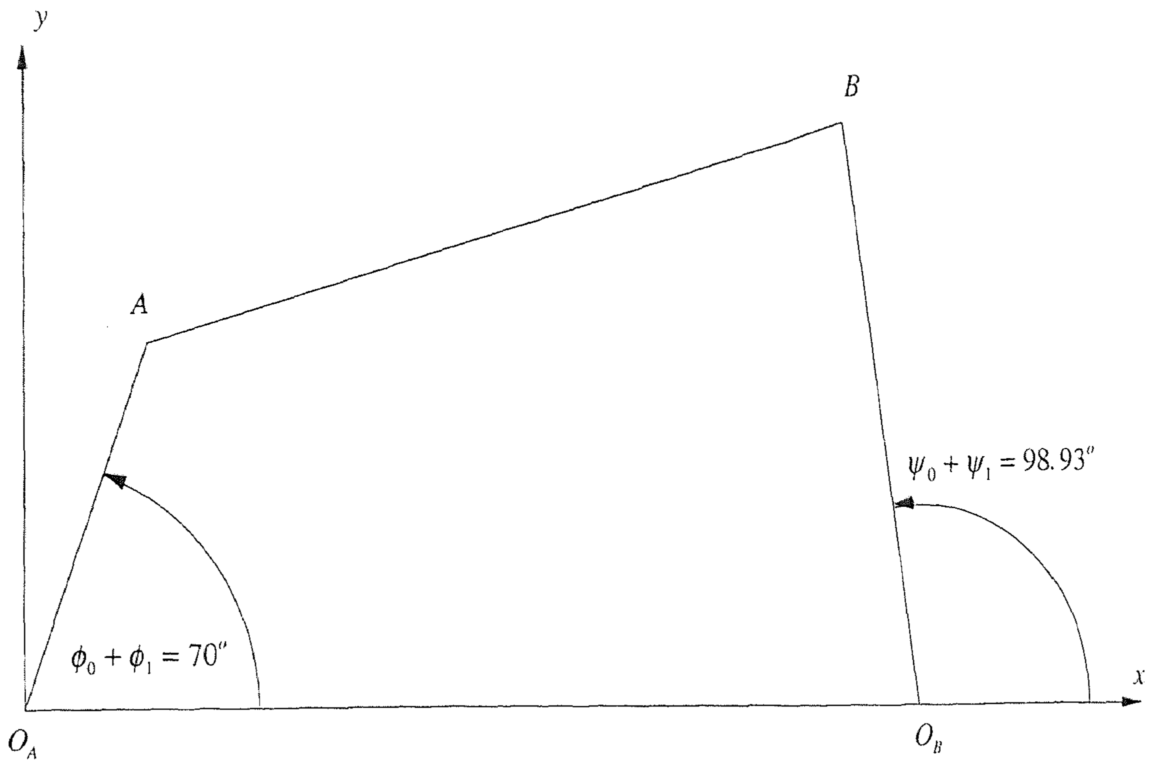

Figure 3.2 The mechanism of figure 3.1 with the input-shaft rotated an additional ten degrees

There is another valid solution for the output-shaft angle, as shown in figure 3.3. Here, the output-shaft angle is 214.13 degrees, implying a rotation $\psi_1 = 120.24^o$. A common difficulty with algorithms that compute the structural error of a function generator is a tendency to toggle between the two valid solutions. Optimal synthesis algorithms must repeatedly compute the structural error of a function generator, and this toggling is a common problem in the development of synthesis techniques.

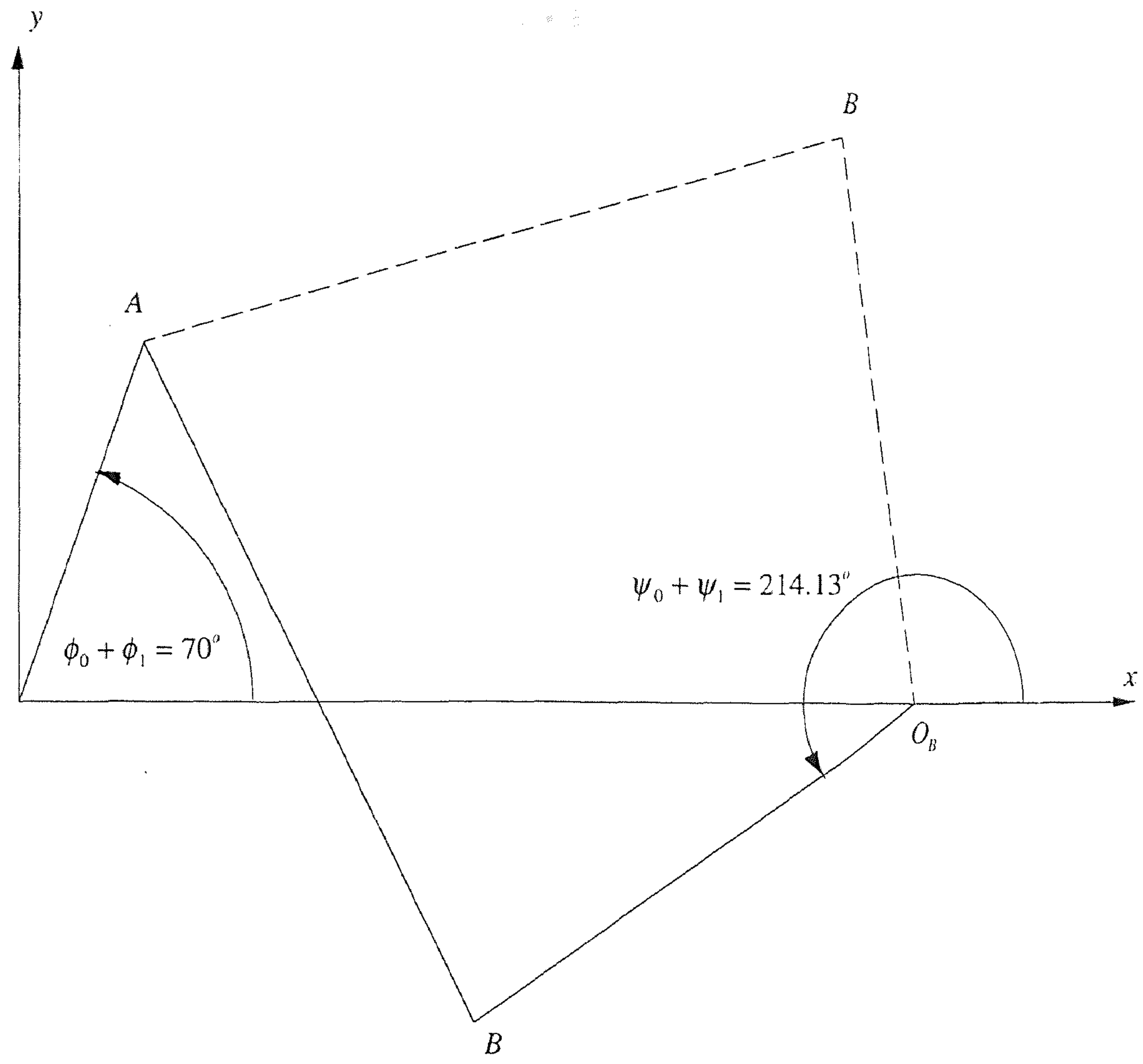

Figure 3.3 Alternate solution for the mechanism of figure 3.2

3.1.2.2 Indirect Measurement: An indirect measurement of the structural error is used in the technique presented in this thesis. Referring to figure 3.4, for a given input-shaft angle ϕ, the output-shaft angle will be the actual ψ. If the output-shaft angle is set to its ideal angle, the coupler will stretch to a new length $(a_2 + b)$. The amount of stretching, b, is then an indirect measure of the structural error.

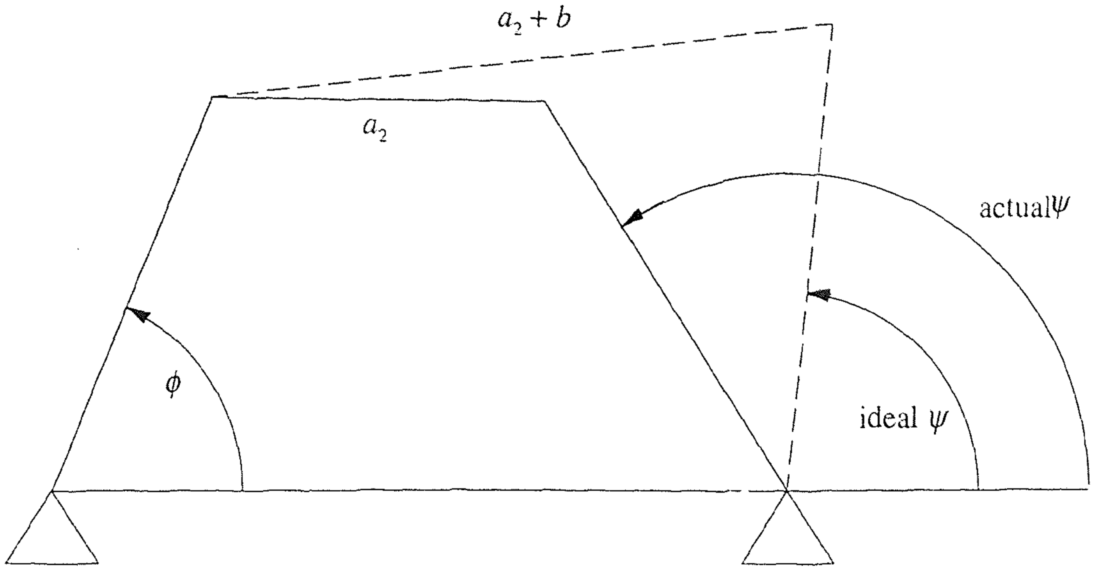

Figure 3.4 Indirect measurement of structural error

This concept can be further illustrated by referring to the mechanism in figure 3.5, with $\phi_1 = 10^o$ and the ideal $\psi_1 = 15^o$. With the input and output links set in their ideal positions, the deformed length of the coupler will be the distance between points A and B.

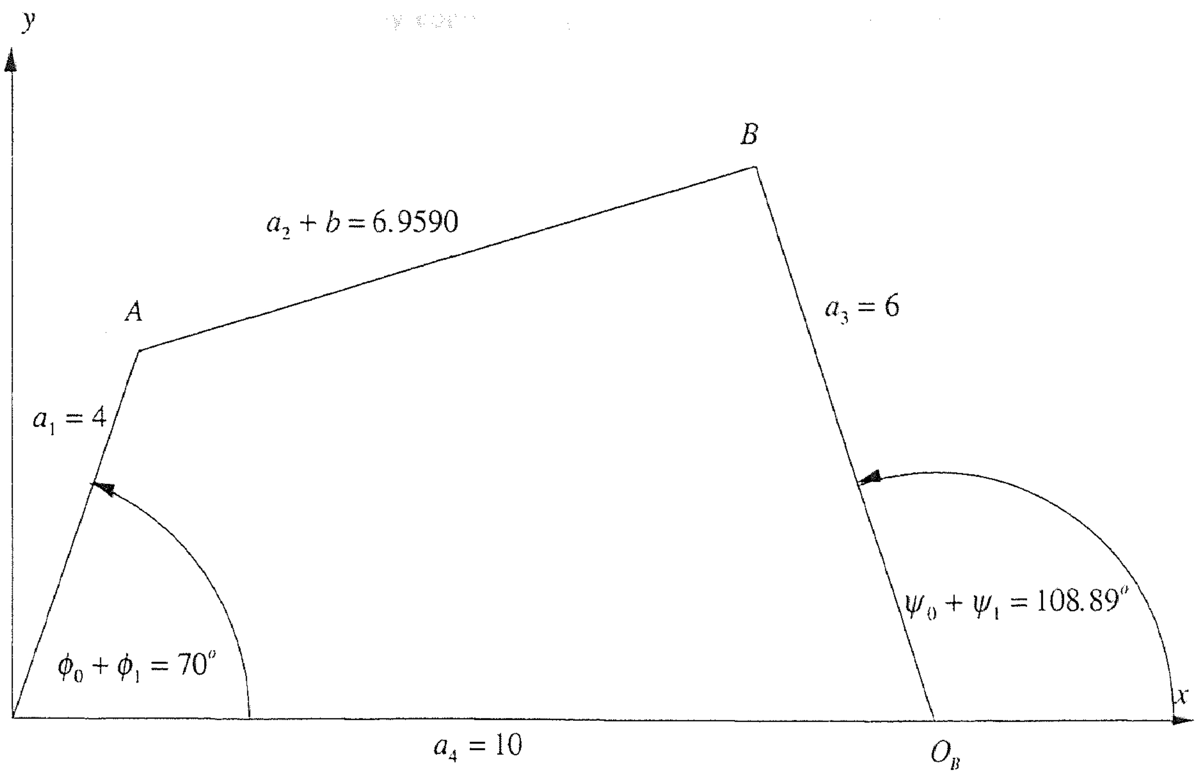

Figure 3.5 The mechanism of figure 3.2 with prescribed input-shaft and output-shaft angles. The coupler deforms to accomodate the rotations.

From the dimensions on the figure, the positions of points A and B can be easily calculated:

$$x_A = 1.368,$$

$$y_A = 3.759,$$

$$x_B = 8.058,$$

$$y_B = 5.677.$$

The distance between points A and B is then 6.959 length units. Since the original undeformed length of the coupler is 8 length units, the deformation b is -1.041 length units. This stretching of the coupler at each synthesis point is an indirect measurement of the structural error. It is easily computed, and it has a unique value at each synthesis point.

Derivation of an expression for the coupler deformation is straightforward. Referring to figure 3.6, the coordinates of point A are

$$(x_A, y_A) = (-a_1 c\theta_1, a_1 s\theta_1) \tag{3.4}$$

and the coordinates of point B are

$$(x_B, y_B) = (a_4 + a_3 c\theta_4, -a_3 s\theta_4). \tag{3.5}$$

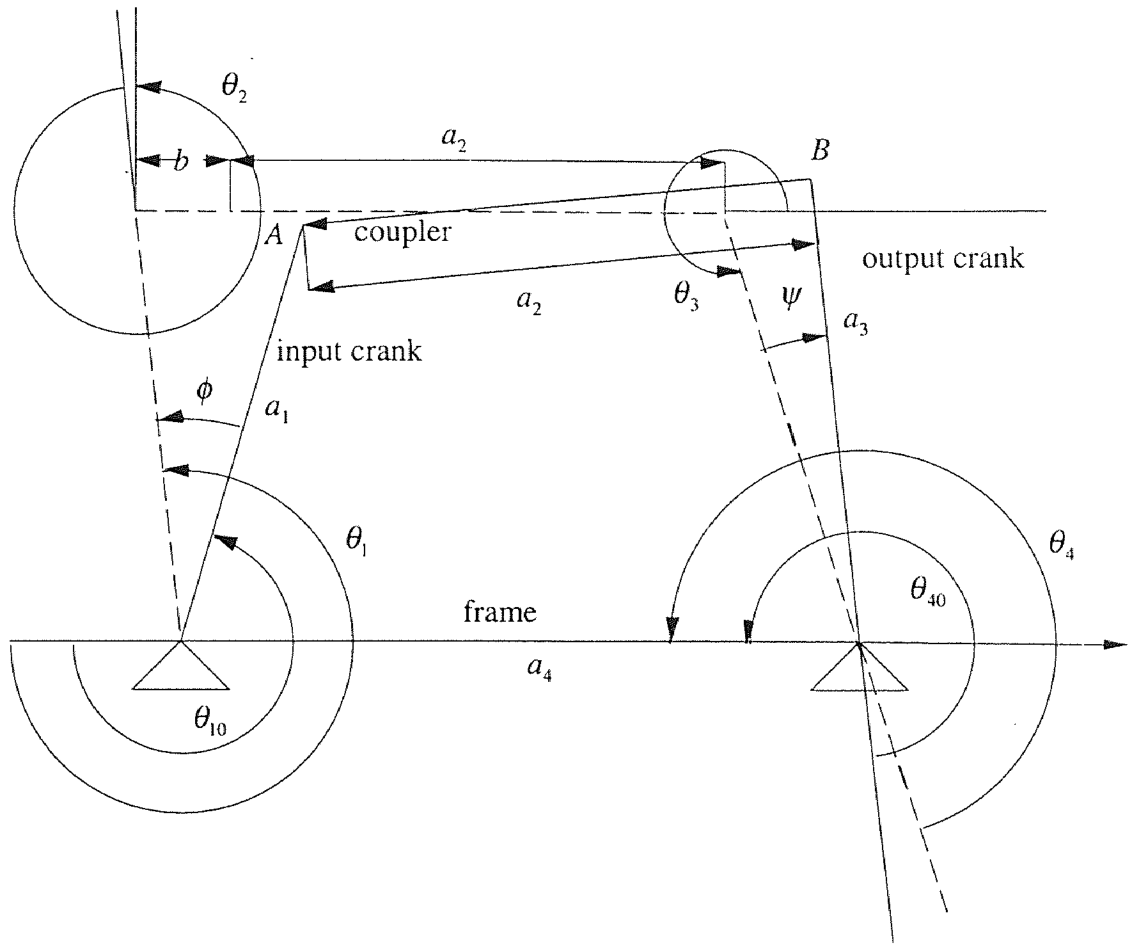

Figure 3.6 Definition of design parameters and coupler deformation

The distance between points A and B, which is also equal to the deformed coupler length $(a_2 + b)$, can be computed as follows:

$$(a_2 + b) = \sqrt{(a_4 + a_3 c\theta_4 + a_1 c\theta_1)^2 + (a_3 s\theta_4 - a_1 s\theta_1)^2}\,. \tag{3.6}$$

Therefore, the coupler deformation is

$$b = \sqrt{\left((a_4 + a_3 c\theta_4 + a_1 c\theta_1)^2 + (a_3 s\theta_4 - a_1 s\theta_1)^2\right)} - a_2\,. \tag{3.7}$$

These expressions can be simplified by writing

$$h = a_4 + a_3 c\theta_4 + a_1 c\theta_1 \tag{3.8}$$

$$v = a_3 s\theta_4 - a_1 s\theta_1 \tag{3.9}$$

$$q = \sqrt{h^2 + v^2}\,. \tag{3.10}$$

Then we have

$$b^2 = (q - a_2)^2. \tag{3.11}$$

Figure 3.7 illustrates the physical significance of the quantities h, v, and q. When the input and output shaft angles are set to their ideal values, h is the horizontal distance between points A and B, v is the vertical distance between these two points, and q is the length of a line segment drawn between the two points.

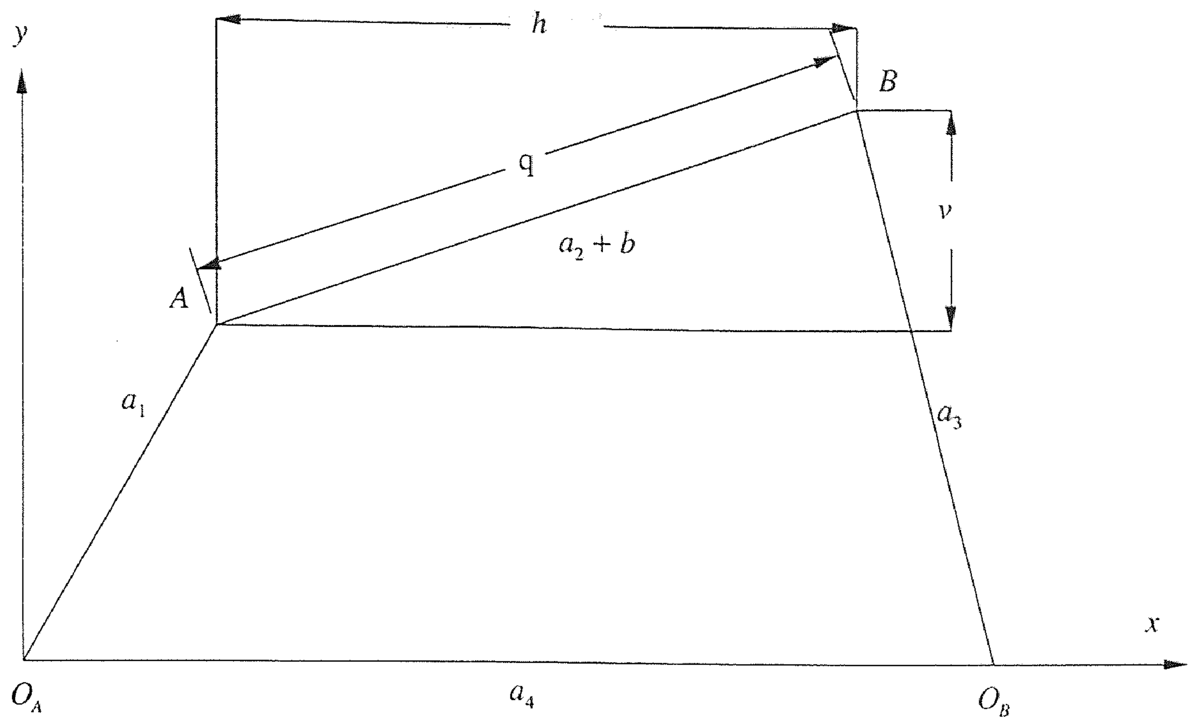

Figure 3.7 The physical significance of h, v, and q.

Note that in these equations,

$$c\theta = \cos(\theta) \text{ and } s\theta = \sin(\theta). \qquad (3.12)$$

3.1.3 Definition of the Objective Function

Having defined an indirect measurement of the structural error, we can now define the objective function. In general, for each synthesis point i, there is a pair of ideal input and output-shaft rotations ϕ_i and ψ_i, and a coupler deformation b_i. Our objective function will be the sum of the squares of the coupler deformations at all of the synthesis points. Mathematically, when there are k synthesis points,

$$U = \sum_{i=1}^{k} b_i^2. \qquad (3.13)$$

Function generators typically define the input and output-shaft angles as rotations from a starting position. This is accomplished in our formulation by defining θ_{10} and θ_{40} as the starting input and output-shaft angles, respectively, as shown in figure 3.6. Then,

$$\theta_1 = \theta_{10} + \phi,$$
$$\theta_4 = \theta_{40} + \psi \qquad (3.14)$$

where the angles ϕ and $\psi = f(\phi)$ define the relationship between input and output-shaft rotations. Each synthesis point consists of a pair of input and output-shaft rotations. It is important to note that, because this formulation of the objective function uses the coupler stretch due to prescribed input and output-shaft rotations as a measure of error, the angles, ϕ, ψ θ_1, and θ_4 represent the ideal shaft rotations as specified at the synthesis points. The difference between the actual shaft rotations and the ideal positions is not considered until the structural error of a mechanism is analyzed.

The goal of the optimal synthesis technique presented in this thesis is to find a set of design parameters that minimizes the objective function, U, as defined in equation (3.13).

3.1.4 The Loop Closure Constraint

No relationship among the design parameters has been implied in the indirect measurement of structural error or the definition of the objective function. However, it is evident that the four links, with the coupler in its undeformed state, must form a closed loop. Referring to figure 3.6, this constraint is represented by two equations:

$$-a_1 s\theta_1 + a_2 c\theta_1\theta_2 + a_3 s\theta_4 = 0$$
$$a_1 c\theta_1 + a_2 s\theta_1\theta_2 + a_3 c\theta_4 + a_4 = 0 \qquad (3.15)$$

These equations are not independent, and reduce to Freudenstein's equation

$$F = a_1^2 + a_3^2 + a_4^2 - a_2^2 + 2a_1a_3c\theta_1\theta_4 + 2a_1a_4c\theta_1 + 2a_3a_4c\theta_4 = 0, \qquad (3.16)$$

where

$$c\theta_1\theta_4 = \cos(\theta_1 + \theta_4). \tag{3.17}$$

Freudenstein's equation is typically used to compute the relationship between input and output-shaft angles of a four bar linkage. In our case, the loop closure constraint forces the solution of the optimization problem to satisfy an additional condition, namely, that the links form a closed loop. The constraint is applied at the starting position of the linkage. It is incorporated into the optimization problem through the use of Lagrange multipliers.

3.1.5 The Objective Function with the Loop Closure Constraint

3.1.5.1 The Role of the Loop Closure Constraint: If the components of the vector of design parameters were independent of each other, optimal synthesis would simply amount to minimizing the objective function U with respect to that vector. A common approach to minimizing the objective function is to solve for a vector of design parameters such that the gradient of the objective function is equal to zero. Mathematically, the vector equation to be solved is

$$\nabla U = 0, \tag{3.18}$$

where

$$\nabla U = \left[\frac{\partial U}{\partial r_1}, \frac{\partial U}{\partial r_2}, \ldots, \frac{\partial U}{\partial r_n}\right]. \tag{3.19}$$

This system of n equations can be solved for the vector of design parameters $\mathbf{r}$, using suitable techniques to assure that the solution results in a global minimum of the objective function.

In traditional optimal synthesis techniques, the loop closure constraint is used to compute the structural error and is thus automatically incorporated in the definition of the objective function. The indirect measurement of structural error presented here does not rely on the loop closure constraint, and Freudenstein's equation has not appeared in our definition of the objective function. In order for our optimal synthesis technique to function properly, the loop closure constraint must be incorporated into the definition of the objective function.

This requirement can be easily verified by referring back to the definition of structural error as the deformation of the coupler that is necessary to connect the ends of the input and output links. Clearly, without the use of the loop closure constraint, minimization of our objective function could result in a set of link lengths and starting angles that do not form a closed loop.

The presence of the loop closure constraint reduces the number of independent design parameters. Mathematically speaking, it limits the domain in design parameter space in which a solution can be found. The method of Lagrange multipliers is commonly used in the minimization of a function subject to constraints among the input variables. Before addressing the minimization of our objective function subject to the loop closure constraint, it is helpful to discuss the theory of Lagrange multipliers.

3.1.5.2 Discussion of Lagrange Multipliers: Consider a scalar function , $U(\mathbf{r})$, of an n-dimensional vector $\mathbf{r}$. Let the components of the vector $\mathbf{r}$ be related by m constraint equations

$$F_1(\mathbf{r}) = 0, F_2(\mathbf{r}) = 0, \ldots, F_m(\mathbf{r}) = 0, \quad m < n \tag{3.20}$$

Each constraint equation reduces by one the number of independent components of the vector $\mathbf{r}$. Now consider the scalar function

$$V = U + \lambda_1 F_1 + \lambda_2 F_2 + \ldots + \lambda_m F_m \tag{3.21}$$

It can be shown mathematically that, where the function U subject to the constraints $F_1, \ldots, F_m$ has an extremum, the gradient of the function V is equal to zero.[14]

Thus, in mathematical terms, an extremum exists when

$$\nabla V = \nabla U + \lambda_1 \nabla F_1 + \lambda_2 \nabla F_2 + \ldots + \lambda_m \nabla F_m = 0, \tag{3.22}$$

where

$$\nabla_{\mathrm{r}} V = \left[\frac{\partial V}{\partial r_1}, \frac{\partial V}{\partial r_2}, \ldots, \frac{\partial V}{\partial r_n} \right]. \tag{3.23}$$

The new variables $\lambda_1, \lambda_2, \ldots, \lambda_m$ are referred to as Lagrange multipliers, and are additional unknown quantities. The equation $\nabla_{\mathrm{r}} V = 0$ represents n scalar equations. The components of the vector $\mathbf{r}$, along with the Lagrange multipliers, represent $n+m$ unknowns. The constraints $F_1 = 0, \ldots, F_m = 0$ provide an additional m equations, resulting in $n+m$ equations in $n+m$ unknowns.

$$\begin{gathered}
\frac{\partial V}{\partial r_1} = \frac{\partial U}{\partial r_1} + \lambda_1 \frac{\partial F_1}{\partial r_1} + \ldots + \lambda_m \frac{\partial F_m}{\partial r_1} = 0 \\
\frac{\partial V}{\partial r_m} = \frac{\partial U}{\partial r_m} + \lambda_1 \frac{\partial F_1}{\partial r_m} + \ldots + \lambda_m \frac{\partial F_m}{\partial r_m} = 0 \\
\frac{\partial V}{\partial \lambda_1} = F_1 = 0 \\
\frac{\partial V}{\partial \lambda_m} = F_m = 0
\end{gathered} \tag{3.24}$$

If we define the partitioned vector x, consisting of the vector of design parameters r and the set of Lagrange multipliers $\lambda_1, \lambda_2, \ldots, \lambda_m$ as

$$\mathbf{x} = \left[r_1 \ldots, r_n, \lambda_1, \ldots, \lambda_m \right], \tag{3.25}$$

Then we can write $\nabla_{\mathrm{x}} V = 0$ in place of the preceding equations, where

$$\nabla_x V = \left[\frac{\partial V}{\partial r_1}, \ldots, \frac{\partial V}{\partial r_n}, \frac{\partial V}{\partial \lambda_1}, \ldots, \frac{\partial V}{\partial \lambda_m} \right] = \left[\frac{\partial V}{\partial x_1}, \ldots, \frac{\partial V}{\partial x_{n+m}} \right] \tag{3.26}$$

This equation $\nabla_{\mathrm{x}} V = 0$, considered as a vector equation $\mathbf{g}(\mathbf{x}) = 0$ with $n+m$ components, must be solved in order to minimize a scalar function of n variables subject to m constraint equations. The following simple example illustrates the use of Lagrange multipliers.

Consider the function

$$U(\mathbf{r}) = U(x,y) = x^2 + y^2, \tag{3.27}$$

$$\mathbf{r} = [x \quad y], \tag{3.28}$$

subject to the constraint

$$y = x + 2 \tag{3.29}$$

or

$$F = y - x - 2 = 0. \tag{3.30}$$

The modified objective function V then becomes

$$V = x^2 + y^2 + \lambda(y - x - 2). \tag{3.31}$$

Taking the gradient of V with respect to $\mathbf{r}$, we obtain

$$\nabla V = \begin{bmatrix} 2x - \lambda \\ 2y + \lambda \end{bmatrix} \tag{3.32}$$

The vector equation $\mathbf{g}(\mathbf{x}) = 0$ then becomes

$$\mathbf{g}(\mathbf{x}) = \nabla_{\mathbf{x}} V = \begin{bmatrix} \nabla_{\mathbf{r}} V \\ F \end{bmatrix} = \begin{bmatrix} \dfrac{\partial V}{\partial r_1} \\ \dfrac{\partial V}{\partial r_2} \\ F \end{bmatrix} = 0 \tag{3.33}$$

$$\begin{aligned} 2x - \lambda &= 0 \\ 2y + \lambda &= 0 \\ y - x - 2 &= 0 \end{aligned} \tag{3.34}$$

which can be solved to obtain $x = -1$, $y = 1$, $\lambda = -2$.

3.1.5.3 Formulation of the Modified Objective Function: The objective function has previously been defined as the sum of the squares of the structural errors at each synthesis point.

$$U = \sum_{i=1}^{k} b_i^2 \tag{3.13}$$

The constraint equation relating the components of the vector of design parameters is

$$F = a_1^2 + a_3^2 + a_4^2 - a_2^2 + 2a_1a_3c\theta_1\theta_4 + 2a_1a_4c\theta_1 + 2a_3a_4c\theta_4 = 0. \tag{3.16}$$

Therefore, the modified objective function becomes

$$V = U + \lambda F. \tag{3.35}$$

It has been stated earlier that the constraint equation is applied when the input and output-shaft angles are at their starting values. This is equivalent to defining the starting position of the mechanism as a precision point. In fact, due to our technique for measuring the structural error, each position of the mechanism for which the loop closure constraint is invoked becomes a precision point. The indirect measure of the structural error has been defined as the deformation of the coupler that results when the input and output-shaft angles are set to their ideal values. But the constraint equation, applied at a synthesis point, requires that the links form a closed loop without any coupler deformation. This is equivalent to the definition of a precision point, which is that the output-shaft angle generated by the mechanism matches its ideal value. In the development of this synthesis technique we have chosen arbitrarily to apply the constraint equation at only the starting position.

3.2 Minimization of the Objective Function

3.2.1 Formulation of the Equations

Through the use of Lagrange multipliers, we have incorporated the loop closure constraint into the objective function. It is now necessary to solve for a vector of design parameters, $\mathbf{r}$, which minimizes the objective function U subject to the constraint $F(\mathbf{r}) = 0$. This is accomplished by setting the gradient of the modified objective function equal to zero. The equations to be solved are

$$\nabla U + \lambda \nabla F = 0 \tag{3.36}$$

and

$$F = 0. \tag{3.37}$$

If we define the partitioned vector x as

$$\mathbf{x} = [\mathbf{r} \mid \lambda] \tag{3.38}$$

and note that

$$\frac{\partial V}{\partial \lambda} = F, \tag{3.39}$$

the equations can be rewritten as

$$\nabla_x V = 0 \text{ or } \mathbf{g}(\mathbf{x}) = 0, \tag{3.40}$$

where

$$\nabla_x = \left[\frac{\partial}{\partial r_1}, \ldots, \frac{\partial}{\partial r_n}, \frac{\partial}{\partial \lambda} \right]. \tag{3.41}$$

In order to expand and solve these equations, it is necessary to develop expressions for the partial derivatives of U and F with respect to the design parameters.

3.2.2 Partial Derivatives of the Unmodified Objective Function

We have previously defined the unmodified objective function U as the sum of the squares of the coupler deformations at all of the synthesis points:

$$U = \sum_{i=1}^{k} b_i^2. \tag{3.13}$$

The partial derivative of U with respect to the jth component r_j of the vector of design parameters $\mathbf{r}$ must therefore be:

$$\frac{\partial U}{\partial r_j} = \sum_{i=1}^{k} \frac{\partial (b_i^2)}{\partial r_j} = 2\sum_{i=1}^{k} b_i \frac{\partial b_i}{\partial r_j}. \tag{3.42}$$

Here, b_i is the coupler deformation at the ith synthesis point, and r_j is the jth component of the vector of design parameters $\mathbf{r}$, where

$$\mathbf{r} = [a_1, a_2, a_3, a_4, \theta_{10}, \theta_{40}] \tag{3.43}$$

Referring back to equations (3.4) through (3.11), where we defined the coupler deformation, at the ith synthesis point we introduce the factors:

$$h_i = a_4 + a_3 c\theta_{40}\psi_i + a_1 c\theta_{10}\phi_i, \tag{3.44}$$

$$v_i = a_3 s\theta_{40}\psi_i - a_1 s\theta_{10}\phi_i, \tag{3.45}$$

$$q_i = \sqrt{h_i^2 + v_i^2}, \tag{3.46}$$

and

$$p_i = 1 - a_2 / q_i. \tag{3.47}$$

In these equations,

$$\theta_{40}\psi_i = \theta_{40} + \psi_i \text{ and } \theta_{10}\phi_i = \theta_{10} + \phi_i. \tag{3.48}$$

Then,

$$b_i = q_i - a_2. \tag{3.49}$$

We can now evaluate the appropriate partial derivatives.

$$\begin{aligned}
\frac{\partial U}{\partial a_1} &= 2\sum_{i=1}^{k}(q_i - a_2)\frac{\partial(q_i - a_2)}{\partial a_1} \\
&= 2\sum_{i=1}^{k}(q_i - a_2)\frac{\partial\left[\sqrt{h_i^2 + v_i^2} - a_2\right]}{\partial a_1} \\
&= 2\sum_{i=1}^{k}(q_i - a_2)\frac{\partial\left[\sqrt{h_i^2 + v_i^2}\right]}{\partial a_1} \\
&= 2\sum_{i=1}^{k}\frac{(q_i - a_2)}{2\sqrt{h_i^2 + v_i^2}}\frac{\partial(h_i^2 + v_i^2)}{\partial a_1} \\
&= \sum_{i=1}^{k}\frac{q_i - a_2}{q_i}\left[2h_i\frac{\partial h_i}{\partial a_1} + 2v_i\frac{\partial v_i}{\partial a_1}\right] \\
&= 2\sum_{i=1}^{k}p_i\left(h_i c\theta_{10}\phi_i - v_i s\theta_{10}\phi_i\right).
\end{aligned} \tag{3.50}$$

A similar process can be used to find the remaining components of the gradient of U.

$$\frac{\partial U}{\partial a_2} = -2\sum_{i=1}^{k}(q_i - a_2) \tag{3.51}$$

$$\frac{\partial U}{\partial a_3} = 2\sum_{i=1}^{k}p_i\left(h_i c\theta_{40}\psi_i + v_i s\theta_{40}\psi_i\right) \tag{3.52}$$

$$\frac{\partial U}{\partial a_4} = 2\sum_{i=1}^{k}h_i p_i \tag{3.53}$$

$$\frac{\partial U}{\partial \theta_{10}} = -2\sum_{i=1}^{k}p_i\left(a_1 h_i s\theta_{10}\phi_i + a_1 v_i c\theta_{10}\phi_i\right) \tag{3.54}$$

$$\frac{\partial U}{\partial \theta_{40}} = 2\sum_{i=1}^{k}p_i\left(-a_3 h_i s\theta_{40}\psi_i + a_3 v_i c\theta_{40}\psi_i\right) \tag{3.55}$$

3.2.3 Partial Derivatives of the Constraint Function

Referring to our previous definition, the constraint function is

$$F = a_1^2 + a_3^2 + a_4^2 - a_2^2 + 2a_1a_3c\theta_{10}\theta_{40} + 2a_1a_4c\theta_{10} + 2a_3a_4c\theta_{40} \tag{3.16}$$

The partial derivatives of the constraint function are therefore

$$\frac{\partial F}{\partial a_1} = 2\left(a_1 + a_3c\theta_{10}\theta_{40} + a_4c\theta_{10}\right) \tag{3.56}$$

$$\frac{\partial F}{\partial a_2} = -2a_2 \tag{3.57}$$

$$\frac{\partial F}{\partial a_3} = 2\left(a_3 + a_1c\theta_{10}\theta_{40} + a_4c\theta_{40}\right) \tag{3.58}$$

$$\frac{\partial F}{\partial a_4} = 2\left(a_4 + a_1c\theta_{10} + a_3c\theta_{40}\right) \tag{3.59}$$

$$\frac{\partial F}{\partial \theta_{10}} = -2a_1\left(a_3s\theta_{10}\theta_{40} + a_4s\theta_{10}\right) \tag{3.60}$$

$$\frac{\partial F}{\partial \theta_{40}} = -2a_3\left(a_1s\theta_{10}\theta_{40} + a_4s\theta_{40}\right) \tag{3.61}$$

3.2.4 The Complete Vector Equation

The expressions from the previous two sections can be combined to obtain the vector equation which will be solved for the optimum set of design parameters. Writing the vector function $\mathbf{g}(\mathbf{x})$ as

$$\mathbf{g}(\mathbf{x}) = \left[g_1(\mathbf{x}), \ldots, g_7(\mathbf{x})\right] \tag{3.62}$$

we obtain

$$g_1 = \frac{\partial U}{\partial a_1} + \lambda \frac{\partial F}{\partial a_1}$$

$$= 2\left[\sum_{i=1}^{k} p_i\left(h_i c\theta_{10}\phi_i - v_i s\theta_{10}\phi_i\right)\right] + 2\lambda\left(a_1 + a_3 c\theta_{10}\theta_{40} + a_4 c\theta_{10}\right) = 0 \qquad (3.63)$$

$$g_2 = \frac{\partial U}{\partial a_2} + \lambda \frac{\partial F}{\partial a_2} = -2\left[\sum_{i=1}^{k}\left(q_i - a_2\right)\right] - 2\lambda a_2 = 0 \qquad (3.64)$$

$$g_3 = \frac{\partial U}{\partial a_3} + \lambda \frac{\partial F}{\partial a_3}$$

$$= 2\left[\sum_{i=1}^{k} p_i\left(h_i c\theta_{40}\psi_i + v_i s\theta_{40}\psi_i\right)\right] + 2\lambda\left(a_3 + a_1 c\theta_{10}\theta_{40} + a_4 c\theta_{40}\right) = 0 \qquad (3.65)$$

$$g_4 = \frac{\partial U}{\partial a_4} + \lambda \frac{\partial F}{\partial a_4} = 2\left[\sum_{i=1}^{k} p_i h_i\right] + 2\lambda\left(a_4 + a_1 c\theta_{10} + a_3 c\theta_{40}\right) = 0 \qquad (3.66)$$

$$g_5 = \frac{\partial U}{\partial \theta_{10}} + \lambda \frac{\partial F}{\partial \theta_{10}}$$

$$= -2\left[\sum_{i=1}^{k} a_1 p_i\left(h_i s\theta_{10}\phi_i + v_i c\theta_{10}\phi_i\right)\right] - 2\lambda a_1\left(a_3 s\theta_{10}\theta_{40} + a_4 s\theta_{10}\right) = 0 \qquad (3.67)$$

$$g_6 = \frac{\partial U}{\partial \theta_{40}} + \lambda \frac{\partial F}{\partial \theta_{40}}$$

$$= 2\left[\sum_{i=1}^{k} a_3 p_i\left(-h_i s\theta_{40}\psi_i + v_i c\theta_{40}\psi_i\right)\right] - 2\lambda a_3\left(a_1 s\theta_{10}\theta_{40} + a_4 s\theta_{40}\right) = 0 \qquad (3.68)$$

$$g_7 = F$$

$$= a_1^2 + a_3^2 + a_4^2 - a_2^2 + 2a_1 a_3 c\theta_{10}\theta_{40} + 2a_1 a_4 c\theta_{10} + 2a_3 a_4 c\theta_{40} = 0 \qquad (3.69)$$

3.3 Minimization Techniques

3.3.1 The Newton-Raphson Method for a Scalar Function of one Variable

There are two basic approaches to finding the point at which a function has a minimum. One is to work directly with the function to be minimized. The downhill Simplex method, and Powell's methods are examples of this approach. The other approach is to find a point where the gradient, or derivative in the case of a scalar function of a scalar, of the function is equal to zero. This approach converts the problem of finding a minimum to that of finding the root of an equation. We have defined an objective function which, along with its derivatives, is straightforward to evaluate. The Lagrange multiplier technique involves setting derivatives equal to zero, so it therefore seems prudent to minimize the objective function by setting its gradient equal to zero.

The objective function, the modified objective function with the constraint equation and Lagrange multipliers, and its gradient, have previously been defined. (Equations 3.13,3.35,3.63-69). In order to minimize the modified objective function V, the vector equation

$$\nabla_{\mathbf{x}} V = 0 \text{ or } \mathbf{g}(\mathbf{x}) = 0 \tag{3.40}$$

must be solved. As mentioned earlier, the nature of these equations requires a numerical solution. A common technique for finding roots of scalar or vector equations is the Newton-Raphson method. Before discussing the application of this method to vector equations, it is useful to examine its application to a scalar equation.

Consider the scalar function $y = g(x)$, shown in figure 3.8. Let x_0 be the initial guess of the solution to $g(x) = 0$. At $x = x_0$, the function y has a nonzero value $g(x_0)$.

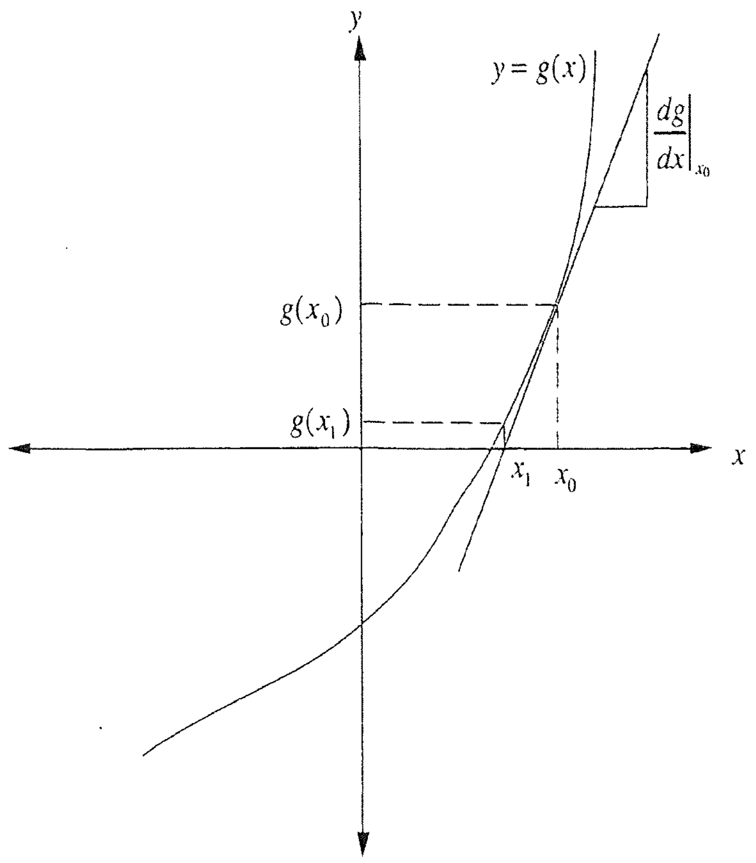

Figure 3.8 Illustration of the Newton-Raphson method

A better approximation of the root of the equation can be found by calculating the derivative of y at $x = x_0$, and defining the new approximation x_1 as

$$x_1 = x_0 - \frac{g(x_0)}{\left(\frac{dg}{dx}\right)_{x=x_0}} \tag{3.70}$$

At $x = x_1$, a new approximation is defined as

$$x_2 = x_1 - \frac{g(x_1)}{\left(\frac{dg}{dx}\right)_{x=x_1}}. \tag{3.71}$$

Eventually, the distance between successive approximations becomes small enough that the solution is said to converge, Mathematically, the convergence criterion can be written as

$$\left|\frac{x_{k+1}-x_k}{x_k}\right| \le \varepsilon, \tag{3.72}$$

where ε is an arbitrarily small quantity.

3.3.2 The Newton-Raphson Method for a Vector Function

We have defined the vector equation $\nabla_x V = 0$ to be solved in order to minimize the modified objective function. The Newton method described in the previous section can be easily extended to this vector function.

Consider the vector equation $\mathbf{g}(\mathbf{x}) = 0$. As in the scalar case, we can make an initial guess as to the solution to this equation. Let this initial guess be the vector $\mathbf{x}_0$. At this point, the function g will have a value $\mathbf{g}(\mathbf{x}_0)$. The next approximation to the root is defined as

$$\mathbf{x}_1 = \mathbf{x}_0 - \left[J^{-1}(\mathbf{x}_0)\right]\mathbf{g}(\mathbf{x}_0). \tag{3.73}$$

In this equation, J is the Jacobian matrix. For an n-component vector function $\mathbf{g}(\mathbf{x})$, the Jacobian matrix is an $n \times n$ matrix of partial derivatives whose elements are

$$j_{ij} = \frac{\partial g_i}{\partial x_j}. \tag{3.74}$$

The convergence parameter is usually some norm of the difference between successive approximations, and the convergence criterion can be written as

$$\left\|\frac{\mathbf{x}_{k+1}-\mathbf{x}_k}{\mathbf{x}_k}\right\| \le \varepsilon. \tag{3.75}$$

The vector

$$\delta\mathbf{x} = -\left[J^{-1}(\mathbf{x}_0)\right]\mathbf{g}(\mathbf{x}_0) \tag{3.76}$$

is called the Newton step, and the unit vector in that direction is called the Newton direction.

3.3.3 Limitations of the Newton Method

Consider again the case of a scalar function of a scalar. If the function approaches zero monotonically, as shown in figure 3.8, the Newton method is likely to converge rapidly to the solution. If, however, the function $y = g(x)$ traces a path with an inflection such as that shown in figure 3.9, and one of the approximations or the initial guess falls at a point where the curve has a small slope, the subsequent iteration may result in a value of x well outside of the permissible domain, and the Newton method will not converge to a solution.

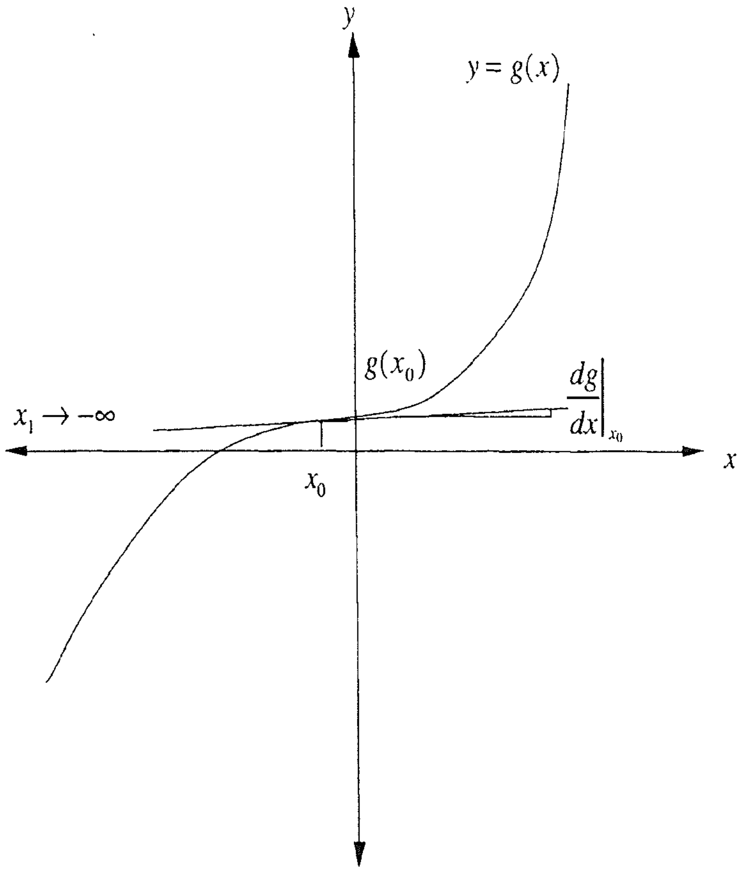

Figure 3.9 Nonconvergence of the Newton method

This limitation of the Newton method extends to the solution of the vector equation $\mathbf{g}(\mathbf{x}) = 0$. It can be seen from equation (3.73) that, when the determinant of the

Jacobian matrix is very small, the Newton step is very large. This may result in a value of $\mathbf{x}$ outside of the permissible domain, and subsequent nonconvergence.

The Newton method will reliably find the solution to the equation $\mathbf{g}(\mathbf{x}) = 0$, provided that the hypersurface defined by $\mathbf{g}(\mathbf{x})$ is suitably contoured and that the initial guess is suitably close to the solution. This limitation is not acceptable for our problem. In the next section, an improvement on the Newton method is discussed.

3.3.4 A Globally Convergent Newton Method

Press et al[16] have presented a modified Newton method which is not as dependent on the accuracy of the initial guess. In their method, the Newton step

$$\delta\mathbf{x} = -\left[J^{-1}(\mathbf{x}_0)\right]\mathbf{g}(\mathbf{x}_0) \tag{3.76}$$

is computed, but it is not automatically used. Instead, a test is performed to determine whether the function

$$f = \frac{1}{2}(\mathbf{g} \bullet \mathbf{g}) \tag{3.77}$$

has decreased in value. Note that f is always positive, and that as $\mathbf{g}$ approaches zero, f will also approach zero.

Press et al have also shown that moving from an initial $\mathbf{x}$ in the direction of the Newton step will reduce f and thus bring $\mathbf{g}$ closer to zero. Mathematically,

$$\nabla f \bullet \delta\mathbf{x} = \left(\mathbf{g}^{\mathrm{T}}[J]\right)\left(-\left[J^{-1}\right]\mathbf{g}\right) = \mathbf{g}^{\mathrm{T}}\left(-[J]\left[J^{-1}\right]\right)\mathbf{g} = -\mathbf{g}^{\mathrm{T}}\mathbf{g} = -\mathbf{g} \bullet \mathbf{g} < 0 \tag{3.78}$$

The algorithm presented by Press et. al proceeds in the following sequence: given a value of $\mathbf{x}$, the functions $\mathbf{g}$ and f are computed, along with the Jacobian matrix J and the Newton step $\delta\mathbf{x}$. Then, $\mathbf{g}$ and f are evaluated at $\mathbf{x} + \delta\mathbf{x}$. If f has actually decreased, the next iteration proceeds from this point. If f has not decreased, the algorithm backtracks along the Newton direction until it reaches a point where f is less than its previous value. The next iteration proceeds from this point.

3.3.5 Consideration of Other Minimization Techniques

Our fundamental problem is to minimize an objective function $U(\mathbf{r})$ subject to the constraint $F(\mathbf{r}) = 0$. The constraint has been incorporated into a modified objective function

$$V = U + \lambda F. \tag{3.36}$$

We have sought to minimize the function V indirectly, by finding the point at which its gradient is zero.

A number of minimization techniques work directly on the objective function, rather than explicitly solving for a zero gradient. Among them are the downhill simplex method, the direction set (Powell's) method, the Davidson-Fletcher-Powell algorithm, and simulated annealing methods. Methods such as these are not suitable for minimizing the modified objective function V.

If we examine the modified objective function $V = U + \lambda F$, it is evident that the requirement of $F = 0$ is not contained in this definition. In fact, an algorithm which attempts to minimize V can find a solution which has a large positive value of U, a large positive value of F, and a large negative value of λ, resulting in a minimum value of V which may be less than zero. This solution also represents a mechanism in which the links do not form a closed loop.

The constraint equation $F = 0$ does not appear unless we require that the gradient of V is equal to zero. Referring back to equation (3.39), the final component of the gradient of V is

$$\frac{\partial V}{\partial \lambda} = F = 0. \tag{3.39}$$

The objective function must therefore be minimized by finding a point at which its gradient is zero. This will insure that the structural error is minimized and that the links will form a closed loop. The globally convergent Newton method is a good algorithm for this purpose.

3.4 Formulation of the Minimization Algorithm

3.4.1 Implementation of the Newton Method for Solving the Vector Equation

3.4.1.1 Outline of the Solution Sequence: As mentioned earlier, our search for a vector of design parameters, $\mathbf{r}$, which minimizes the objective function U subject to the constraint $F(\mathbf{r})=0$, amounts to solving the vector equation $\mathbf{g}(\mathbf{x})=0$. To solve this equation using the Newton-Raphson method, the following steps are required:

First, $\mathbf{x}_0$, an initial guess for the vector $\mathbf{x}$, is made. Then, the vector function $\mathbf{g}$ and the Jacobian matrix J are evaluated at this value of $\mathbf{x}$. A new value of $\mathbf{x}$ is calculated using the equation

$$\mathbf{x}_1 = \mathbf{x}_0 - \left[J^{-1}(\mathbf{x}_0)\right]\mathbf{g}(\mathbf{x}_0). \tag{3.73}$$

This value of $\mathbf{x}$ is used as the initial guess for the next iteration. When successive values of $\mathbf{x}$ are sufficiently close to each other, the solution is said to converge, and the iterations are terminated. Figure 3.10 illustrates the solution sequence for the Newton-Raphson method.

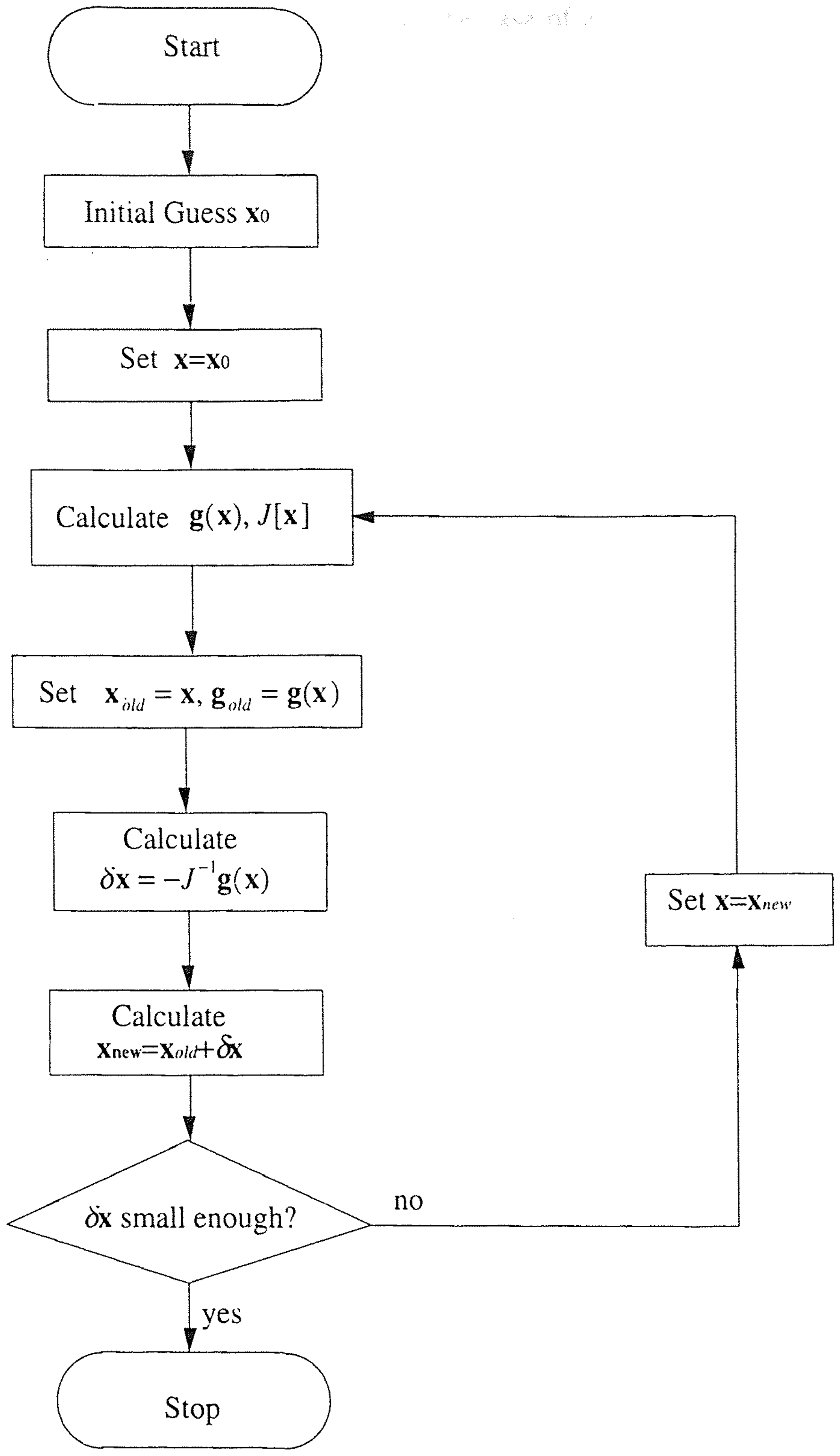
Start
Initial Guess $\mathbf{x}_0$
Set $\mathbf{x}=\mathbf{x}_0$
Calculate $\mathbf{g}(\mathbf{x}), J[\mathbf{x}]$
Set $\mathbf{x}_{old}=\mathbf{x}, \mathbf{g}_{old}=\mathbf{g}(\mathbf{x})$
Calculate $\delta\mathbf{x}=-J^{-1}\mathbf{g}(\mathbf{x})$
Calculate $\mathbf{x}_{new}=\mathbf{x}_{old}+\delta\mathbf{x}$
$\delta\mathbf{x}$ small enough?
no
Set $\mathbf{x}=\mathbf{x}_{new}$
yes
Stop

Figure 3.10 Flowchart for the Newton-Raphson method

The sequence is modified slightly for the case of a globally convergent Newton method. After the Newton step,

$$\delta \mathrm{x} = -\left[J^{-1}(\mathrm{x}_0)\right]\mathrm{g}(\mathrm{x}_0), \tag{3.76}$$

and a new value of $\mathbf{x}$, are computed, the function $\frac{1}{2}\mathbf{g}(\mathbf{x}) \bullet \mathbf{g}(\mathbf{x})$ is evaluated. If it has not decreased from its value in the previous iteration, the magnitude of the Newton step is reduced and the new value of $\mathbf{x}$ is computed. A new value of $\frac{1}{2}(\mathrm{g} \bullet \mathbf{g})$ is also computed. This step is repeated until the value of $\frac{1}{2}(\mathbf{g} \bullet \mathbf{g})$ is less than in the previous iteration. Figure 3.11 illustrates the solution sequence for the globally convergent Newton method.

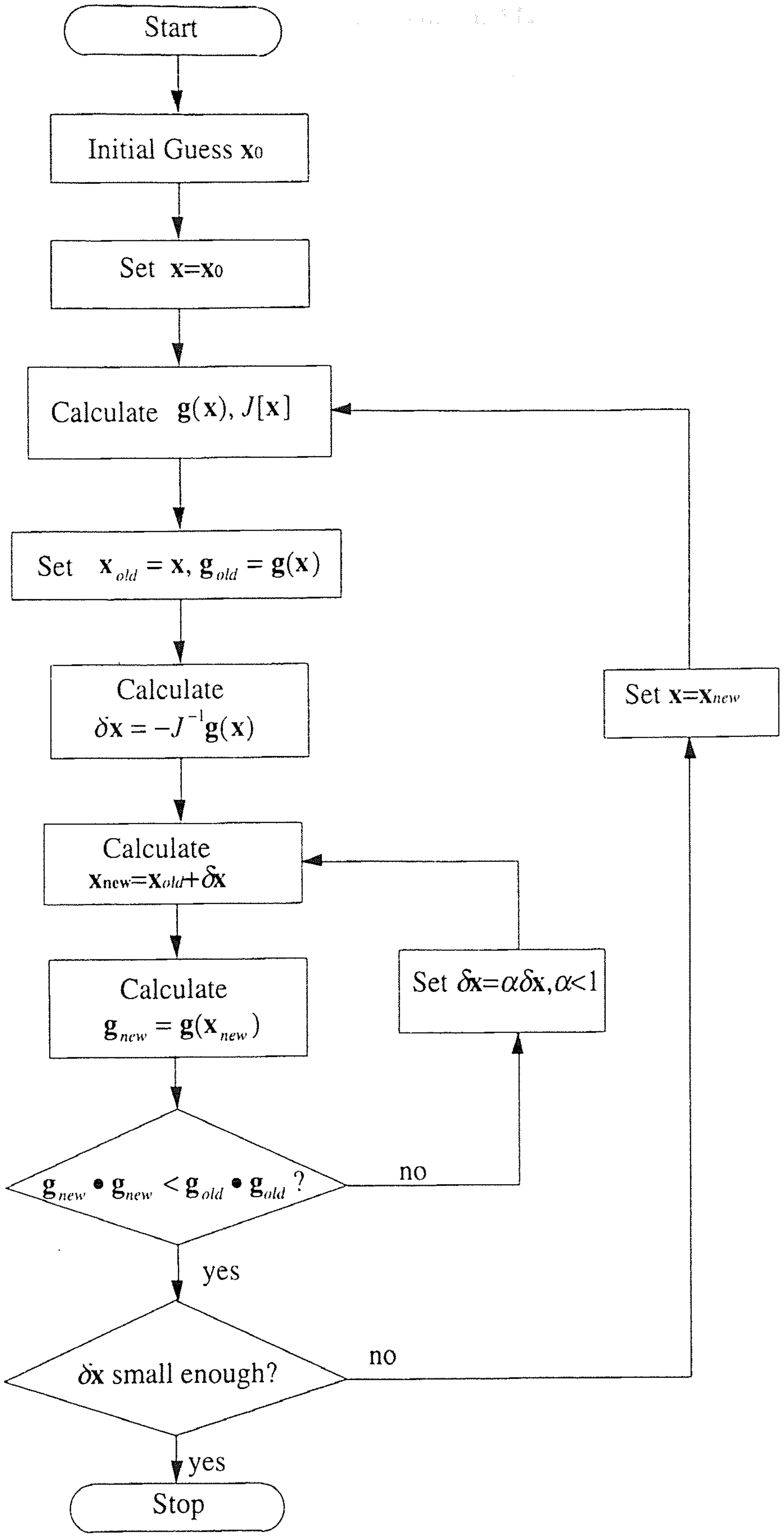
Start
Initial Guess $\mathbf{x}_0$
Set $\mathbf{x}=\mathbf{x}_0$
Calculate $\mathbf{g}(\mathbf{x}), J[\mathbf{x}]$
Set $\mathbf{x}_{old}=\mathbf{x}, \mathbf{g}_{old}=\mathbf{g}(\mathbf{x})$
Calculate $\delta\mathbf{x}=-J^{-1}\mathbf{g}(\mathbf{x})$
Calculate $\mathbf{x}_{new}=\mathbf{x}_{old}+\delta\mathbf{x}$
Calculate $\mathbf{g}_{new}=\mathbf{g}(\mathbf{x}_{new})$
$\mathbf{g}_{new}\bullet\mathbf{g}_{new}<\mathbf{g}_{old}\bullet\mathbf{g}_{old}$?
no
Set $\delta\mathbf{x}=\alpha\delta\mathbf{x}, \alpha<1$
yes
$\delta\mathbf{x}$ small enough?
no
Set $\mathbf{x}=\mathbf{x}_{new}$
yes
Stop

Figure 3.11 Flowchart for the globally-convergent Newton method

3.4.1.2 Definition and Evaluation of the Jacobian Matrix: The Jacobian matrix of a vector function $\mathbf{g}(\mathbf{x})$ is defined as follows: the ijth entry of the Jacobian matrix J is

$$j_{ij} = \frac{\partial g_i}{\partial x_j}. \qquad (3.79)$$

The entries of the Jacobian can be obtained directly by taking partial derivatives of the components of $\mathbf{g}$ given in equations (3.63) through (3.69). For example,

$$j_{13} = \frac{\partial g_1}{\partial x_3} = \frac{\partial g_1}{\partial a_3}. \qquad (3.80)$$

Here, a_3 is the length of link 3, the output link.

Evaluation of the Jacobian is simplified because the matrix is symmetric. This is easily verified by referring back to the definition of $\mathbf{g}(\mathbf{x})$. (Equation 3.40)

$$\mathbf{g} = \nabla_{\mathbf{x}} V. \qquad (3.81)$$

Therefore,

$$g_i = \frac{\partial V}{\partial x_i} \qquad (3.82)$$

and

$$j_{ij} = \frac{\partial g_i}{\partial x_j} = \frac{\partial}{\partial x_j}\frac{\partial V}{\partial x_i} = \frac{\partial^2 V}{\partial x_j \partial x_i} = \frac{\partial^2 V}{\partial x_i \partial x_j} = \frac{\partial}{\partial x_i}\frac{\partial V}{\partial x_j} = \frac{\partial g_j}{\partial x_i} = j_{ji} \qquad (3.83)$$

Another technique for evaluating the Jacobian matrix is the forward difference technique. From the definition of the partial derivative $\partial g_i / \partial x_j$ we can write

$$\frac{\partial g_i}{\partial x_j} \approx \frac{g_i\left(\mathbf{x} + \Delta\mathbf{x}^{(j)}\right) - g_i(\mathbf{x})}{\Delta x_j}, \qquad (3.84)$$

where $\Delta\mathbf{x}^{(j)}$ is a vector whose components are zero except for the jth component, which has the value Δx_j. For example,

$$\Delta \mathrm{x}^{(2)} = \left[0, \Delta x_2, 0, 0, 0, 0, 0\right]. \qquad (3.85)$$

Using this definition, the partial derivatives can be evaluated numerically by evaluating $\mathbf{g}$ at some value of $\mathbf{x}$, then subtracting it from $\mathbf{g}\left(\mathbf{x}+\Delta\mathbf{x}^{(j)}\right)$, and dividing the result by Δx_j. The forward difference technique is used in the algorithm presented in this thesis.

3.4.1.3 Modifications when One or More Design Parameters are Specified: It is often not necessary to solve for the entire vector of design parameters. One or more of the parameters may be arbitrarily specified. Also, in a function generator, the ratios of the link lengths, rather than their actual lengths, are important. For this case, one of the link lengths is generally set to a specified value.

Specifying a design parameter reduces the order of the problem to be solved. A vector $\mathbf{y}$, consisting of the unspecified, or independent, design parameters and the Lagrange multiplier, can be defined. For example, if the length of the frame link a_4 and the starting input-shaft and output-shaft angles are specified, we have

$$\mathbf{x}=\left[a_1,a_2,a_3,a_4,\theta_{10},\theta_{40},\lambda\right], \tag{3.86}$$

$$\mathbf{y}=\left[a_1,a_2,a_3,\lambda\right], \tag{3.87}$$

$$\mathbf{g}=\left[\frac{\partial V}{\partial a_1},\frac{\partial V}{\partial a_2},\frac{\partial V}{\partial a_3},\frac{\partial V}{\partial \lambda}\right]. \tag{3.88}$$

The Jacobian is a 4×4 matrix whose entries are

$$j_{ij}=\frac{\partial g_i}{\partial y_j}. \tag{3.89}$$

The goal of the research presented in this thesis is the optimal synthesis of a function generator whose frame link, starting input-shaft angle, and starting output-shaft angle are specified. From this point onwards, only the reduced problem will be addressed.

3.4.2 Techniques for Assuring Convergence to a True Minimum

When minimizing a function by searching for a point where its gradient is zero, there is a chance that the algorithm will converge to a local minimum, rather than a global one. A common and straightforward technique for avoiding this problem is to run the numerical solution starting from a number of different, and usually random, initial guesses. The technique presented here is similar. In this problem, the starting input-shaft and output-shaft angles are specified, as is the length of the frame link. This leaves the link lengths a_1, a_2, a_3 and the Lagrange multiplier λ as unknowns. The initial guess for the Lagrange multiplier, λ, is arbitrarily chosen to be unity.

Further insight into the nature of the objective function, and function generating mechanisms, leads to the development of a random procedure for selecting initial guesses of a_1, a_2, a_3, and λ.

In a function generator, the ratios of the link lengths, and not their absolute lengths, are important. Thus, we might expect the initial guesses of a_1, a_2, and a_3 to be k_1a_4, k_2a_4, and k_3a_4, respectively. And, since a useful synthesis algorithm should not require an accurate initial guess, it is reasonable to set $k_1 = k_2 = k_3 = k$. The initial values of a_1, a_2, and a_3 therefore become ka_4, ka_4, and ka_4. All that is left is to choose a value of k, and a scheme to vary the initial guesses randomly.

Referring back to the definition of the indirect measure of structural error, it can be seen that if $a_1 = a_3 = 0$, there will be no deformation of the coupler, and if the synthesis algorithm converges to this solution, it will appear to have found an optimal mechanism. If we choose a small value of k for the initial guess, it is likely that the algorithm will converge to this trivial solution. Therefore, it seems prudent to choose a value of k much greater than one for the initial guess. In the numerical solutions that will appear in later sections, k is generally chosen to be between 5 and 10.

The initial guess of the vector of unknowns, $\mathbf{y}$, is

$$[ka_4, ka_4, ka_4, 1] = [a_1, a_2, a_3, \lambda]. \tag{3.90}$$

To obtain the initial guess for the next solution, each component of $\mathbf{y}$ is multiplied by the parameter

$$\gamma = \alpha + (1-\alpha) ran(t), \tag{3.91}$$

where t is a dummy variable, $ran(t)$ is a random number between 0 and 1, and α is a parameter between 0 and 1 chosen by the user. A different value of $ran(t)$ is calculated separately for each component of $\mathbf{y}$.

CHAPTER 4

IMPLEMENTATION OF THE ALGORITHM INTO A COMPUTER PROGRAM

4.1 General Description of the Computer Program

A Fortran computer program which implements the numerical algorithm has been written. The source code of this program, "lnkoph", is given in Appendix A.

The program consists of an input module, an initialization module, an iteration module, and an output module. Before running the program, the user must define the function to be generated in the subprogram f(x). This subprogram is then compiled and linked to the main program.

When the program is run, the user is asked whether each design parameter is specified or free, and prompted for a value of the specified parameter or an initial guess of the free parameter. Next, the program asks for the maximum input-shaft and output-shaft rotations, the number of divisions (which is one less than the number of synthesis points, and the upper and lower values of the domain of the function. Finally, the user is prompted for the iteration decrement parameter (α in equation 3.91). The globally convergent Newton method is then used with ten different initial guesses, and the set of design parameters, along with the structural error of the resulting mechanism, is output for each case. The user is then able to examine the output and select the best mechanism. A detailed description of the program follows.

4.2 Description of the Input Module

4.2.1 Design Parameters

In the class of synthesis problems considered here, the length of the frame link, and the starting input-shaft and output-shaft angles, are specified. That leaves the lengths of the input link, coupler link, and output link, and the Lagrange multiplier as the free variables. Referring back to equation (3.87), the vector of free parameters is

$$\mathbf{y} = \left[a_1, a_2, a_3, \lambda\right]. \tag{4.1}$$

At input, the user is asked whether each component of the vector of design parameters $\mathbf{x}$, is specified or free. If specified, it is flagged so that the program will not include it in the vector $\mathbf{y}$, and it is then used as a constant. Link lengths are entered in arbitrary and consistent length units. Starting shaft angles are input in degrees counterclockwise from horizontal. The Lagrange multiplier is generally not specified, and a value of 1 is used as an initial guess. After the components of the vector $\mathbf{x}$ have been defined, the synthesis points are set up.

4.2.2 Definition of the Synthesis Points

To determine the input-shaft and output-shaft rotations that comprise the synthesis points, it is necessary to know the function to be generated, the domain of interest, and the range of motion of the input-shaft and the output-shaft. The domain of the function is divided into an arbitrary number, $k-1$, of equal intervals. This results in a total of k synthesis points. Mathematically, if we are trying to generate

$$y = f(x), \quad x_a \le x < x_b \tag{4.2}$$

and we divide the domain into $k-1$ intervals, we have a set of x-values

$$x_i = x_a + \frac{i-1}{k-1}(x_b - x_a)$$

$$\text{for} \tag{4.3}$$

$$1 \le i \le k.$$

The corresponding set of y-values is

$$y_i = f(x_i). \tag{4.4}$$

Given a range or input-shaft rotation r_ϕ and an output-shaft range r_ψ, we can write

$$\phi_i = \frac{x_i - x_a}{x_b - x_a} r_\phi \tag{4.5}$$

and

$$\psi_i = \frac{y_i - y_{\min}}{y_{\max} - y_{\min}} r_\psi, \tag{4.6}$$

where $y_{\max}$ is the greatest value of $f(x)$ over its domain, and $y_{\min}$ is the smallest value. If $f(x)$ increases or decreases monotonically between x_a and x_b, this equation becomes

$$\psi_i = \frac{y_i - y_a}{y_b - y_a} r_\psi. \tag{4.7}$$

The subroutine "siminp" computes the synthesis points. Using the user-supplied subprogram which defines the function $f(x)$, it prompts for the maximum input-shaft and output-shaft rotations, the number of divisions, and the domain of the function. Rotations are input in degrees counterclockwise, and the angles for the synthesis points are computed in degrees counterclockwise. The input-shaft angles are stored in the vector "phi", and the output-shaft angles are stored in the vector "psi".

4.2.3 The Iteration Decrement Parameter

In section 3.4.2, it was stated that a good initial guess for the input, coupler and output link lengths a_1, a_2, and a_3 is somewhere between five and ten times the length of the frame link a_4. Subsequent initial guesses are obtained by multiplying each link length from the previous initial guess by the parameter γ from equation (3.91):

$$\gamma = \alpha + (1-\alpha) ran(t). \tag{4.8}$$

The user-specified parameter α, known as the iteration decrement parameter, controls the rate at which the initial guesses approach zero. Recalling that the algorithm tends to converge to the trivial solution $a_1 = a_3 = 0$ when the initial guesses are too small, we can see that α should be small enough to allow a significant variation between initial guesses, but large enough to prevent the initial guess from approaching the trivial solution. Experience has shown that a reasonable value is $\alpha = 0.7$.

4.3 The Initialization Module

In the initialization module of the program, angles are converted from their input values in degrees counterclockwise from horizontal to radians in the coordinate system that was used to formulate the problem. (figure 4.1) An initial call is then made to the random number generator.

Following these steps, the program enters what is called the external iteration loop. The purpose of this loop is to run the solution algorithm with a set of different initial guesses. The loop is repeated ten times. It starts at the statement

```
do 1000 niter=1,10
```

and ends at statement number 1000.

The first portion of the external loop initializes the globally convergent Newton method. First, the vector of free parameters $\mathbf{y}$, consisting of the unspecified components of the vector $\mathbf{x}$, is extracted. With the exception of the first time the loop is executed, each component of the vector $\mathbf{y}$ is multiplied by a new random variable γ as defined in equation (3.91). Then, the globally convergent Newton algorithm is started by calling the subroutine "newt".

4.4 The Iterative Solution

4.4.1 Overview

The subroutine "newt", presented by Press et. al, is a ready-to-use subroutine which is good at obtaining solutions to a vector equation $\mathbf{g}(\mathbf{x}) = 0$ when the initial guess is not necessarily close to the answer. As it was not developed as part of this research, but rather used as a tool to solve the problem, it will be described briefly. The routine is used in this case to solve the vector equation

$$\mathbf{g}(\mathbf{y}) = \nabla_{\mathbf{y}} V = 0. \qquad (4.9)$$

It starts by checking to see if the initial guess is a solution to $g(y) = 0$, and if not, it evaluates the Jacobian matrix and the Newton step, and then adds some fraction of the Newton step to the vector $\mathbf{y}$ so that the magnitude of the function $\frac{1}{2}(\mathbf{g} \bullet \mathbf{g})$ is reduced. The iterations are repeated until the solution converges or the limit on iterations is exceeded. A user-supplied subroutine "funcv" calculates the vector $\mathbf{g}$, and the subroutine "fdjac" evaluates the Jacobian matrix using the forward difference technique. The routine "fdjac" can be replaced by a user-supplied routine which evaluates the Jacobian matrix analytically.

4.4.2 Evaluation of the Vector of Functions to be Zeroed

The vector of functions to be zeroed is the gradient of the modified objective function V with respect to the vector of free parameters $\mathbf{y}$. In other words,

$$\mathbf{g}(\mathbf{y}) = \nabla_y V = \left[\frac{\partial V}{\partial a_1}, \frac{\partial V}{\partial a_2}, \frac{\partial V}{\partial a_3}, \frac{\partial V}{\partial \lambda} \right]. \tag{4.10}$$

Equations for the full vector $\mathbf{g}(\mathbf{x}) = \nabla_x V$ have already been derived, and they have been coded into the subroutine "funcv". This routine computes $\mathbf{g}(\mathbf{x})$, represented in the program as the vector "fvec", then selects the components that pertain to the vector $\mathbf{y}$, and returns those components as the vector "gstar".

When the routine "funcv" is called, the vector $\mathbf{y}$ is an input, and the vector $\mathbf{x}$, along with the vectors of input-shaft and output-shaft rotations, are accessed through common blocks. The components of $\mathbf{y}$, a_1, a_2, a_3, and λ in this case, are substituted into the appropriate components of the vector $\mathbf{x}$. Then, the evaluation of $\mathbf{g}(\mathbf{x})$ is straightforward using the analytical expressions. After $\mathbf{g}(\mathbf{x})$ is evaluated, the components pertaining to the free parameters, g_1, g_2, g_3, and g_7 in this problem, are extracted to form the vector "gstar".

4.4.3 Evaluation of the Jacobian Matrix

The subroutine "fdjac", presented by Press et al, is used to evaluate the Jacobian matrix. The forward difference approach was chosen over the direct evaluation of the components because it did not involve any extra programming, it was not computationally intensive, and it produced adequate results.

Input variables for the routine are "n", the order of the problem; "x", the vector of parameters which corresponds to $\mathbf{y}$ in our formulation; the vector of functions "fvec", which corresponds to $\mathbf{g}$; and "np", the dimension of the Jacobian matrix. The Jacobian matrix, "df", is output. Elements of the Jacobian matrix are calculated by using the forward difference formula

$$df(i,j)=(f(i)-fvec(i))/h. \qquad (4.11)$$

Here, the vector "f" is obtained by calling the routine "funcv" that generates the vector "fvec", but using a modified vector "xnew" as the input. The vector "xnew" is obtained by adding the quantity

$$h = EPS * x(j) \qquad (4.12)$$

to x(j). If x(j) is zero, an error will result if the result of (4.12) is used to evaluate df(i,j) in equation (4.11). To prevent this, the program checks to see if x(j) is equal to zero before evaluating the expression in (4.12). If x(j) is equal to zero, then the value of "h" is set equal to "EPS".

After the jth column of the Jacobian matrix is evaluated, this process is repeated with j varying from one to np.

4.4.4 Determination of the Vector of Free Parameters

The subroutine "lnsrch" searches for a new value of $\mathbf{y}$, along the Newton direction, which reduces the magnitude of $\frac{1}{2}(\mathbf{g}\bullet\mathbf{g})$ by a sufficient amount. It uses as inputs the order of the problem, "n"; the starting point "xold"; "fold", the value of $\frac{1}{2}(\mathbf{g} \bullet \mathbf{g})$ and its gradient , "g"; a direction vector "p", which is the Newton direction; and "stpmax", a limit on the length of the Newton step. The values of "xold", "fold", "g", "p", and "stpmax" are evaluated in the subroutine "newt" before "lnsrch" is called. The vector xold is simply the vector of free parameters, $\mathbf{y}$, and "fold" is then $\frac{1}{2}(\mathbf{g} \bullet \mathbf{g})$. Its gradient is therefore $\mathbf{g}^T[J]$. The initial Newton step, "p", is $-[J^{-1}]\mathbf{g}$. In the subroutine "lnsrch", the technique of adjusting the magnitude of the Newton step, previously described in section 3.3.4, is employed. The subroutine returns a new vector ,"x", which corresponds to the vector of free parameters, $\mathbf{y}$.

4.4.5 Convergence Test

The quantity "temp" is evaluated for each component of the vector "x" in the subroutine "newt".

$$\text{temp} = (\text{abs}(x(i)\text{-xold}(i)))/\max(\text{abs}(x(i)),1.) \tag{4.13}$$

The lowest value of temp is compared to the tolerance parameter, "TOLX", and if it is less than that parameter, the solution is said to converge and the iterations are terminated.

4.5 The Output Module

After each call to the subroutine "newt", the main program inserts the components of the vector of free parameters, $\mathbf{y}$ into the appropriate location in the vector of design parameters, $\mathbf{x}$. Then, it calculates the structural error at each synthesis point for the mechanism which has been generated, and uses it to compute the root mean square structural error for the mechanism. After the output is written, the main program returns to the beginning of the external iteration loop, computes another initial guess, and issues another call to the subroutine "newt".

CHAPTER 5

TESTING OF THE ALGORITHM ON A SAMPLE PROBLEM

5.1 Description of the Problem

In order to assess the effectiveness of our optimal synthesis algorithm, it is necessary to synthesize some four-bar function generators using this technique, and evaluate the structural error of each mechanism. It would also be helpful to compare the structural errors of our mechanisms to the structural errors of function generators synthesized by precision techniques. We will therefore find some examples of function generators synthesized by precision techniques, use our optimal method to synthesize mechanisms which generate the same functions over the same range, and compare the accuracy of the mechanisms.

Freudenstein, in a 1958 article, synthesized a set of mechanisms which generated different functions[17]. The precision synthesis technique was used, with five precision points. Structural error was minimized by adjusting the location of the precision points. The unspecified design parameters were the input link length, the coupler link length, and the starting input-shaft and output-shaft angles.

We will attempt to synthesize mechanisms which generate the same set of functions, but we will address the case where the only free design parameters are the lengths of the input, coupler, and output links. The starting input-shaft and output-shaft angles are specified, as is the length of the frame link. The ranges of motion for both the input-shaft and the output-shaft are prescribed. Figure 5.1 shows the configuration of the four-bar function generators.

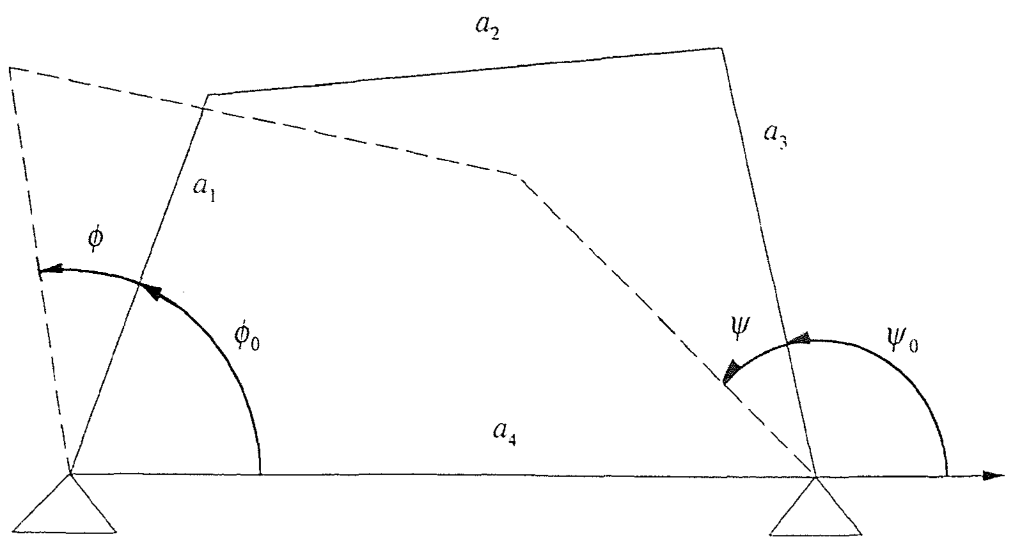

Figure 5.1 Configuration for the test problem

Table 5.1 summarizes the functions to be generated, and the specified design parameters. The negative values for the input and output ranges indicate that the shaft rotations are clockwise for all of these examples.

Table 5.1 Specified parameters for the sample problems

Function	Interval	Starting Input-shaft Angle(deg)	Starting Output-shaft Angle (deg)	Input Range (deg)	Output Range (deg)
$\log_{10} x$	$1 \le x \le 2$	-52.6	-79.1	-60.0	-60.0
$\sin x$	$0 \le x \le 90^o$	242.3	284.4	-90.0	-90.0
$\tan x$	$0 \le x \le 45^o$	90.3	55.8	-90.0	-90.0
e^x	$0 \le x \le 1$	118.4	139.6	-90.0	-90.0
$1/x$	$1 \le x \le 2$	-33.8	59.8	-90.0	-90.0
$x^{1.5}$	$0 \le x \le 1$	185.2	211.7	-90.0	-90.0
x^2	$0 \le x \le 1$	209.3	126.2	-90.0	-90.0
$x^{2.5}$	$0 \le x \le 1$	88.3	135.5	-90.0	-90.0
x^3	$0 \le x \le 1$	85.9	142.4	-90.0	-90.0

5.2 Setting up the Problem

The first step in setting up the computer program "lnkoph" is for the user to enter the function to be generated into the subprogram "function f(x)". A sample listing of this subprogram, for the case $f(x) = \log_{10} x$ is shown below:

```
function f(x)
f=alog10(x)
return
end
```

This subprogram is then compiled and linked to the main program, "lnkoph", which can then be executed.

When the program is run, the user is prompted for initial guesses of the unspecified design parameters, the values of the specified design parameters, and an initial guess of the Lagrange multiplier, λ. In all of the runs presented here, the unspecified design parameters are the link lengths a_1, a_2, and a_3, and the initial guess for each is chosen to be five times the length of the frame link, a_4. Because the length of the frame link is specified to be 100 length units for all runs, the initial guesses for the other three links will be 500 length units. The initial guess for the Lagrange multiplier is unity.

The program then sets up the synthesis points. The user is prompted for $x_{\min}$ and $x_{\max}$, the ends of the interval on which the function $f(x)$ is to be generated. Then, the program asks for the maximum values of ϕ and ψ, the input-shaft and output-shaft rotations. Finally, the user is prompted for the number of segments into which the interval $x_{\min} \le x \le x_{\max}$ will be divided. This quantity, "nstep", is one less than the number of synthesis points. In all of the cases presented here, the interval is divided into 30 segments, resulting in 31 synthesis points.

The last user input is the iteration decrement parameter, α. It is set to 0.7 for all of the runs presented here.

CHAPTER 6

SUMMARY OF RESULTS

6.1 Results from the Optimal Synthesis Program

Six of the nine runs converged to an acceptable result. Table 6.1 summarizes the results of these runs. In the three runs that did not yield an acceptable solution, the algorithm converged for several of the initial guesses, but the structural error in the resulting mechanisms was unacceptable. Some of the initial guesses resulted in converges to the trivial solution $a_1 = a_3 = 0$.

Table 6.1 Results from Optimal Synthesis Program

Function	Interval	ϕ_{max} (deg)	ψ_{max} (deg)	ϕ_0 (deg)	ψ_0 (deg)	a_1/a_4	a_2/a_4	a_3/a_4	Error(deg) rms	Error(deg) max
$\log_{10} x$	$1 \le x \le 2$	-60	-60	-52.6	-79.1	3.33	0.86	3.49	.07	.11
$\sin x$	$0 \le x \le 90^o$	-90	-90	242.3	284.4	2.54	2.78	0.86	.20	.74
e^x	$0 \le x \le 1$	-90	-90	118.4	139.6	3.49	0.91	3.35	.08	.33
x^2	$0 \le x \le 1$	-90	-90	209.3	126.2	1.86	2.67	0.51	.06	.19
$x^{2.5}$	$0 \le x \le 1$	-90	-90	88.3	135.5	1.83	0.95	1.25	.32	.78
x^3	$0 \le x \le 1$	-90	-90	85.9	142.4	1.65	0.99	1.08	.46	1.4

Table 6.2 summarizes the results generated by Freudenstein.

Table 6.2 Results from Freudenstein

Function	Interval	$\phi_{\max}$	$\psi_{\max}$	ϕ_0	ψ_0	a_1/a_4	a_2/a_4	a_3/a_4	Error(deg) max
$\log_{10} x$	$1 \le x \le 2$	-60	-60	-52.6	-79.1	3.35	0.85	3.49	.01
$\sin x$	$0 \le x \le 90^{o}$	-90	-90	242.3	284.4	1.83	2.24	0.69	.19
e^x	$0 \le x \le 1$	-90	-90	118.4	139.6	3.50	0.88	3.40	.03
x^2	$0 \le x \le 1$	-90	-90	209.3	126.2	2.52	3.33	0.56	.07
$x^{2.5}$	$0 \le x \le 1$	-90	-90	88.3	135.5	1.80	0.91	1.27	.41
x^3	$0 \le x \le 1$	-90	-90	85.9	142.4	1.61	0.93	1.07	.51

6.2 Convergence

Table 6.3 shows the number of initial guesses, the number of successful convergences, the maximum structural error, and the value of the Lagrange multiplier for a number of test cases. There was not a single convergence to the trivial solution, $a_1 = a_3 = 0$, in any of the runs.

Table 6.3 Statistics from Optimal Synthesis Runs

Function	Interval	Initial Guesses	Successful Convergence	RMS Error (deg)	λ Lagrange Multiplier
$\log_{10} x$	$1 \le x \le 2$	10	5	.07	-.0015
$\sin x$	$0 \le x \le 90^o$	6	1	.20	.0000148
e^x	$0 \le x \le 1$	6	6	.08	-.00367
x^2	$0 \le x \le 1$	6	2	.06	-.00009
$x^{2.5}$	$0 \le x \le 1$	10	8	.32	.00268
x^3	$0 \le x \le 1$	10	8	.46	.0009

CHAPTER 7

DISCUSSION OF RESULTS

7.1 Overview

In this chapter, the results of the sample problem of Chapter 5 are examined in more detail, and the performance of the optimal synthesis algorithm is discussed. The purpose of the optimal synthesis algorithm is to produce a function generator whose input-shaft and output-shaft rotations are as close as possible to their ideal values at a large number of synthesis points. A good algorithm should synthesize mechanisms having a low structural error, and should not be strongly dependent on the initial guess of the solution. In the following sections, the ability of the algorithm to converge to a solution is discussed, and the structural errors of the solutions are compared to Freudenstein's.

7.2 Convergence

The program "lnkoph" attempted to start the globally convergent Newton routine at ten different initial guesses for each test case. One measure of the performance of the globally-convergent Newton method, and the algorithm for selecting initial guesses, is the number of initial guesses that result in convergence to a valid solution.

In Table 6.3, some of the test cases have less than ten initial guesses. This occurs when the globally convergent Newton method fails to converge in an acceptable number of iterations. In this case, the execution of the program is halted, with the results up to the point of termination saved. In many of these cases, an acceptable solution has been obtained by the time execution has stopped.

Because there is a small number of initial guesses, the acceptable solutions can be selected in a straightforward manner. The actual structural error of the mechanism resulting from each solution is printed in the output file. It is only necessary to choose the mechanism with the lowest structural error.

Referring back to Table 6.1, and comparing the values of a_1 / a_4, a_2 / a_4, and a_3 / a_4 to the initial guess $a_1 / a_4 = a_2 / a_4 = a_3 / a_4 = 5$, it is evident that the initial guess is not close to any of the solutions. This indicates that our formulation of the problem is valid, and that the globally convergent Newton method is a robust algorithm.

7.3 Detailed Description of the Results from One Run

It is now useful to examine one of the cases in table 5.1 in more detail. The function $y = x^2$, $0 \le x \le 1$, is to be generated by a mechanism having 90 degrees of input-shaft and output-shaft rotation in the clockwise direction. The starting input-and-output-shaft angles are 209.3 degrees and 126.2 degrees, respectively, as shown in the table.

Thirty-one synthesis points, spaced evenly at three degree intervals for the input-shaft, were used. Thus,

$$\{\phi_1, \phi_2, \ldots, \phi_{30}, \phi_{31}\} = \{0, -3, \cdots, -87, -90\}. \qquad (7.1)$$

The function $y = x^2$ was mapped onto the output shaft rotation as follows:

$$\phi_i = 90x_i \qquad (7.2)$$

$$\psi_i = \frac{90}{f(1) - f(0)}\left[f(x_i) - f(0)\right] = 90x_i^2 \qquad (7.3)$$

or,

$$\{\psi_1, \psi_2, \ldots, \psi_{30}, \psi_{31}\} = \{0, -0.10, \ldots, -84.10, -90.00\}. \qquad (7.4)$$

Then, link length a_4 was set to 100 length units, and initial guesses of 500 length units were entered for link lengths a_1, a_2, and a_3. The program was run, and solutions were obtained for six initial guesses. Two of these solutions were identical, and had low structural error. This repeated solution appears in table 6.1. Convergence was obtained in 13 iterations in one case, and 11 in the other. Figure 7.1 shows the convergence test parameter plotted against the iteration count for both cases.

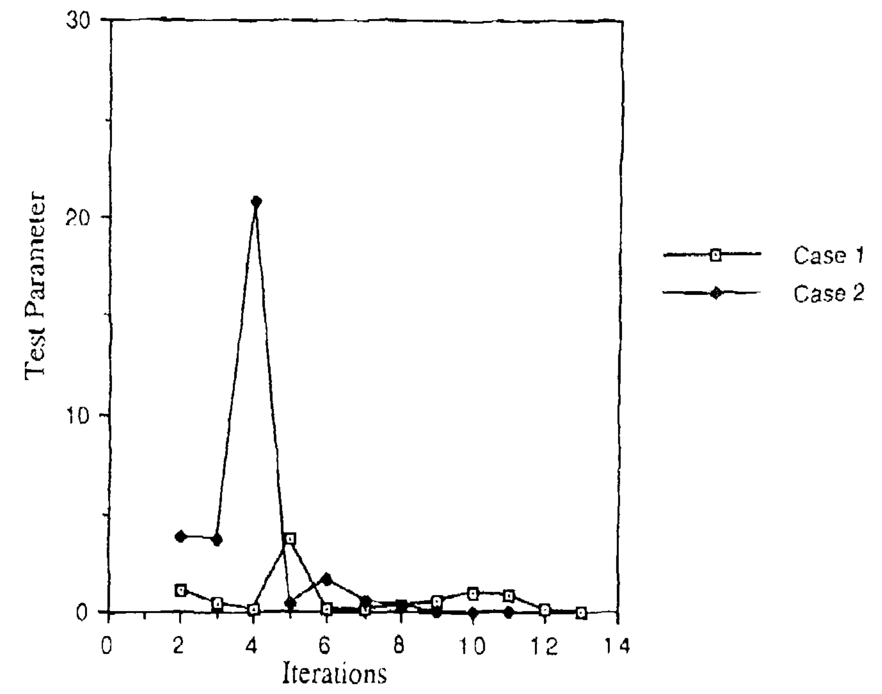

Figure 7.1 Convergence test parameter vs. number of iterations. The first iteration is not included because of the large value of the test parameter. Function generator is for

$y = x^2, \quad 0 \le x \le 1$

When the result is compared to the linkage synthesized by the precision technique with error minimization in the original reference, it is evident that a different, but usable, solution has been obtained. Our solution has

$$\{a_1/a_4 \quad a_2/a_4 \quad a_3/a_4\} = \{1.86 \quad 2.67 \quad 0.51\} \tag{7.5}$$

where Freudenstein had

$$\{a_1/a_4 \quad a_2/a_4 \quad a_3/a_4\} = \{2.52 \quad 3.33 \quad 0.56\}. \tag{7.6}$$

The maximum structural error of our mechanism is 0.19 degrees, whereas Freudenstein obtained 0.07 degrees. Given a total output travel of 90 degrees, a maximum error of 0.19 degrees, with a root mean square error of 0.06 degrees, is acceptable.

7.4 Comparison of Precision Synthesis and Optimal Synthesis Results

It is instructive to compare the mechanisms obtained by precision synthesis to those resulting from optimal synthesis. Referring to tables 6.1 and 6.2, we see that four of the six successful optimal synthesis runs - $\log_{10} x$, e^x, $x^{2.5}$, and x^3 - converged to essentially the same solution as the precision synthesis results. The other two optimal synthesis runs converged to significantly different configurations from their corresponding precision solutions.

There are several reasons for an optimally synthesized mechanism to differ from one obtained by precision synthesis. The first reason is that an optimal synthesis algorithm does not solve the same problem as a precision synthesis routine. The former, we recall, is trying to generate a function which passes as close as possible to a large number of points, while the precision solution generates an exact match at five or fewer points. Furthermore, the error minimization technique used with precision synthesis is based on the maximum error, whereas the optimal synthesis minimizes the sum of squares of the errors over all of the synthesis points.

In our case, an additional cause for discrepancy is that the objective function is based on the coupler deformation, which is an indirect measure of the structural error. The coupler deformation is not, in general, linearly proportional to the structural error over the range of motion. It is therefore quite reasonable to expect our optimal synthesis routine to converge to a result different from that of precision synthesis.

The variation of structural error with the number of synthesis points can also be examined. We will examine in detail the structural error of the mechanism which generates the function $f(x) = x^3$. Figure 7.2 shows the structural error over the range of motion, for the case of 31 synthesis points.

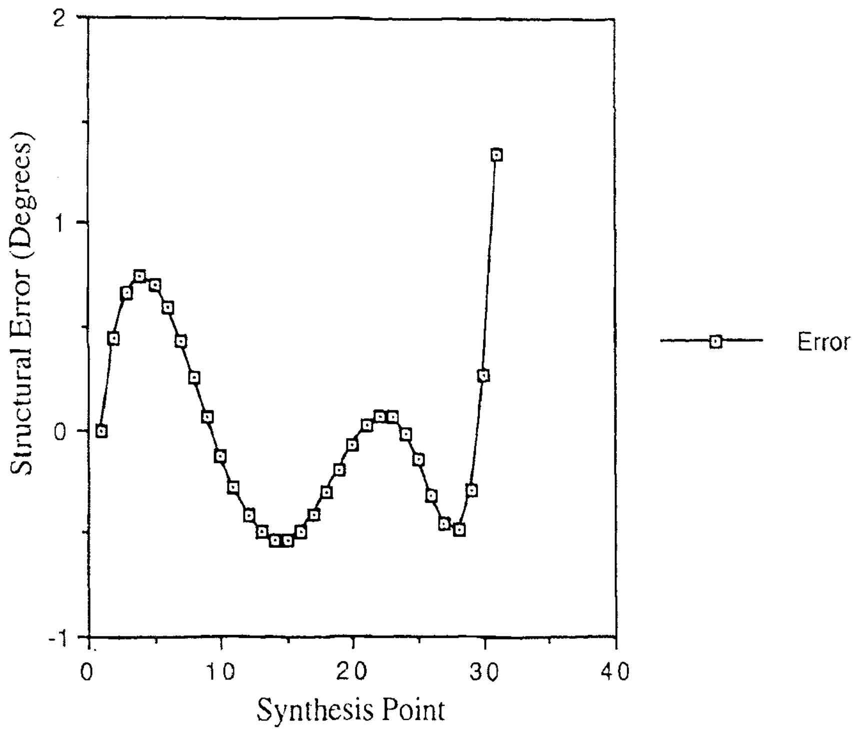

Figure 7.2 Structural error of function generator for $y = x^3, \quad 0 \le x \le 1$

The program "lnkoph" was then run with 11, 21, 41, 51, and 101 synthesis points. Figure 7.3 shows the rms structural error versus the number of synthesis points.

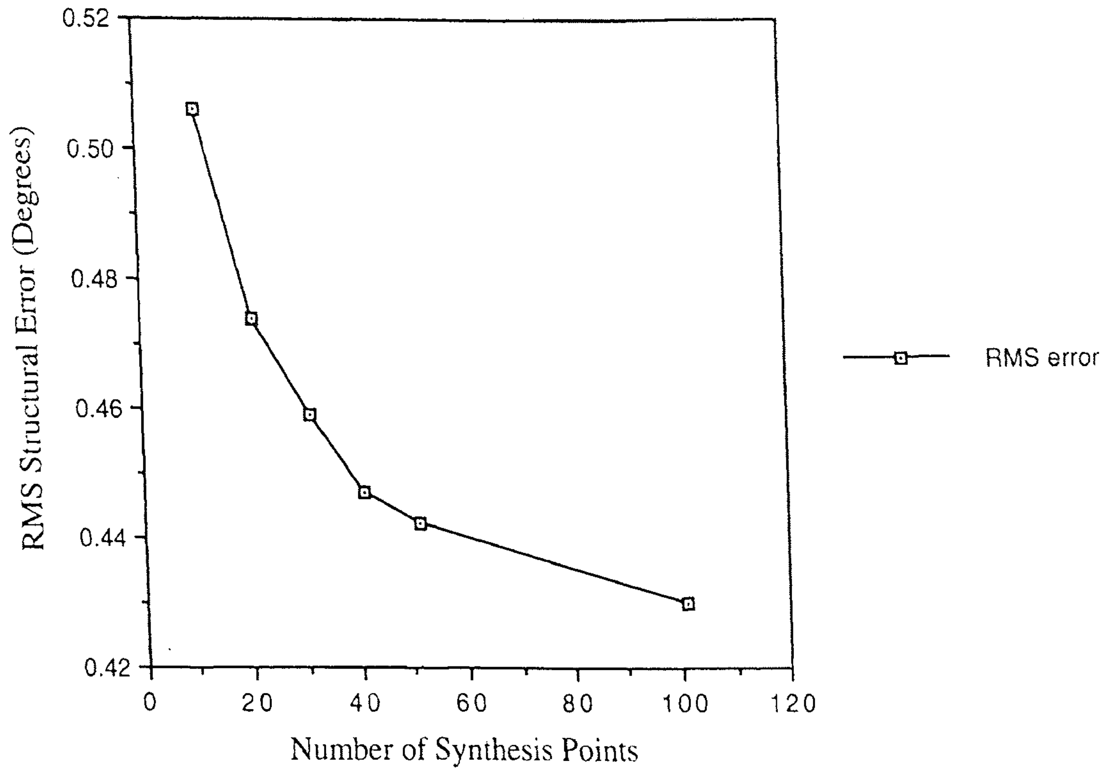

Figure 7.3 RMS structural error vs. number of synthesis points for $y = x^3, \quad 0 \le x \le 1$

When we examine figure 7.2, we see that, with the exception of the last synthesis point, the largest magnitude of the structural error is 0.75 degrees, which is close to the maximum value of .51 degrees from Freudenstein's work. Figure 7.3 shows that the benefit of additional synthesis points decreases as we add more points. The reduction in error as we go from 51 synthesis points to 101 synthesis points is the same as that obtained in moving from 41 points to 51 points.

We now examine the mechanism that generates the function $f(x) = x^2$. This is an important example because the mechanism is significantly different from that which Freudenstein obtained, yet its structural error is low. Figure 7.4 shows the structural error over the range of motion.

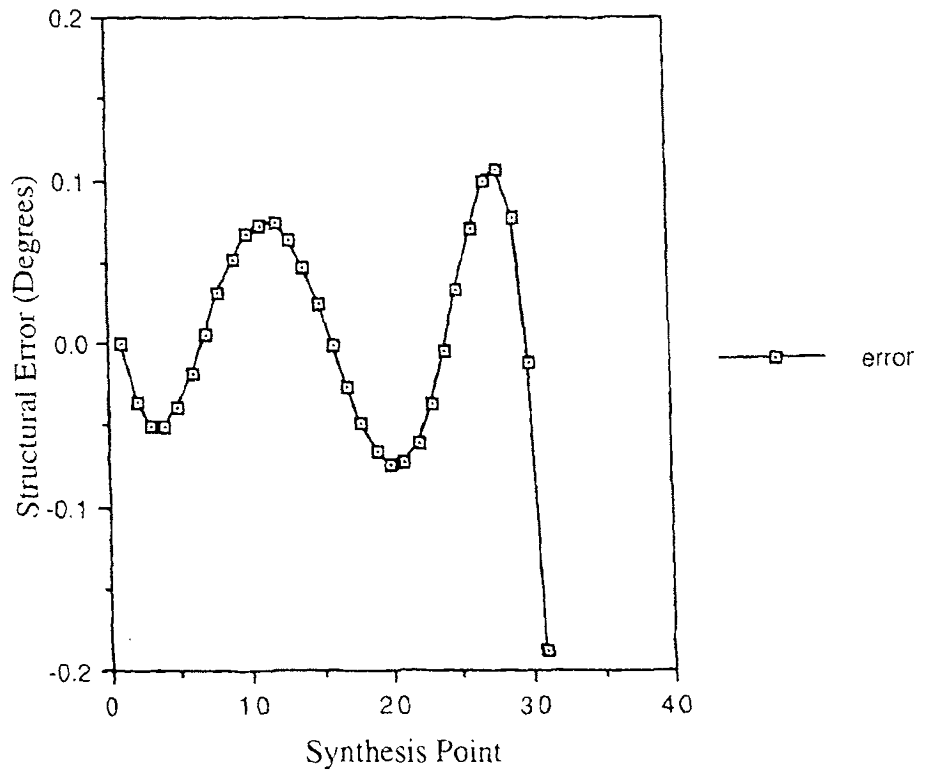

Figure 7.4 Structural error of function generator for $y = x^2$, $0 \le x \le 1$

The results presented in this section show that a mechanism synthesized by the algorithm presented in this thesis performs comparably to one obtained from precision synthesis with error minimization.

7.5 Significance of the Lagrange Multiplier

During the course of this investigation, it was found that small values of the Lagrange multiplier, in general, correspond to mechanisms that have a low structural error. In fact, for all of the runs with 31 synthesis points, values of the Lagrange multiplier greater than 0.005 were found to correspond to mechanisms with unacceptably high structural error. Further study revealed that the magnitude of the Lagrange multiplier did not vary monotonically with the rms structural error, but was more of a qualitative indicator of its magnitude.

CHAPTER 8

CONCLUSIONS

We have shown that it is possible to formulate an optimal synthesis algorithm whose objective function is based on an indirect measure of structural error. Evaluation of the objective function and its derivatives is straightforward.

The use of Lagrange multipliers is required in order to introduce the loop-closure constraint. The objective function, subject to this constraint, can be minimized by using a globally convergent Newton method to find the point where its gradient is zero.

An optimal synthesis algorithm based on this formulation has been tested and found to produce mechanisms whose structural error is comparable to that of the best mechanisms obtained from precision synthesis techniques with error minimization. The algorithm has a high rate of success, and does not require that the initial guess be close to the result.

The optimally synthesized mechanisms presented here are the result of one run of the computer program, with synthesis points at arbitrary locations. There is no laborious respacing of precision points, as is required for the error minimization techniques used with precision synthesis. Furthermore, the technique presented here can be applied to an arbitrary set of synthesis points, while the precision techniques extract their synthesis points from a defined function.

Future research connected with this work might include extending the algorithm to the case where the starting output-shaft angle or the starting input-shaft angle is not a specified design parameter, further development of the algorithm for selecting initial guesses, and further analysis of the significance of the Lagrange multiplier.

APPENDIX

LISTING OF PROGRAM "LNKOPH"

```
      program lnkoph
c-----program for optimal synthesis of linkages
c-----using new constraint function derived 2/7/96
c-----uses iters3.f and funcv3.f and fcner1.f
c-----no IMSL subroutines
c
      parameter(n=6,m=1,itmax=100)
      real jstar(n+m,n+m),gstar(n+m),jsinv(n+m,n+m)
      real g(n+m),x(n+m),phi(50),psi(50),y(7),yinit(7)
      real lambda,uobj(itmax),answer(50),error(50)
      character*1 response
c-----names associated with entries in the vector of design parameters
      character*6 despar(7),prstat(2)
c-----a vector of flags: nonzero entry indicates that the design
c-----parameter is specified
      integer desflg(7)
      data prstat/'free','fixed'/
      data despar/'a1','a2','a3','a4','theta1','theta4','lambda'/
      data desflg/7*0/
c
      common/angs/phi,psi,lmax
      common/desprm/x,desflg
c
c-----open the output file
c
```

```
      open(unit=11,file='lnkoph.dat',form='formatted',status='unknown')
c
      pi=acos(-1.)
c
c-----define the specified design parameters
c
c-----ns is the number of specified design parameters
      ns=0
      do 50 i=1,n+m
      write(*,9004)despar(i)
      read(*,9008)response
      if(response .eq. 'n')go to 45
      write(*,9005)despar(i)
      read(*,*)x(i)
c-----flag this design parameter
      desflg(i)=1
c-----increment the number of specified design parameters
      ns=ns+1
      go to 50
45    continue
c-----enter the initial guess for a non-specified design parameter
      write(*,9006)despar(i)
      read(*,*)x(i)
50    continue
c-----nf is the number of free parameters in the problem
      nf=n+m-ns
c
```

```
c-----define the synthesis points based on the function to be generated
c-----angles are ccw from horizontal. psi will be
c-----converted to minus psi
c
      call siminp(phi,psi,lmax)
      do 100 l=1,lmax
      write(11,9201)l,phi(l),l,psi(l)
      phi(l)=phi(l)*pi/180.
      psi(l)=-psi(l)*pi/180.
 100  continue
c
      write(11,9202)
c
c-----write the initial guess to the data file
c
      do 120 i=1,n+m
      write(11,9203)despar(i),x(i),prstat(desflg(i)+1)
 120  continue
c
c-----enter the iteration decrement parameter
c
      write(*,9007)
      read(*,*)dcrmnt
c
c-----write the iteration decrement parameter
c
      write(11,9202)
      write(11,9204)dcrmnt
      write(11,9202)
c
c-----make an initial call to the random number generator
c
      idum=27
      x(1)=x(1)+0.*ran0(idum)
c
c-----set up the external iteration loop for random intial
c-----guesses of the design parameters
c
      do 1000 niter=1,10
c
c-----convert the input and output shaft angles
c
      x(5)=(x(5)+180.)*pi/180.
      x(6)=(360.-x(6))*pi/180.
c
c-----do an iterative solution
c
c-----define the vector of free parameters, y, consisting of the
c-----nonspecified parameters from the vector x
c
      is=0
      do 200 i=1,n+m
      if(desflg(i) .ne.0)go to 200
      is=is+1
      y(is)=x(i)
      if(niter .eq. 1)yinit(is)=x(i)
200   continue
c
c-----after the first iteration, do a random decrement of
c-----the unspecified design parameter initial guesses
c
      if(niter .eq. 1)go to 260
c
      do 250 if=1,nf
      y(if)=yinit(if)*(dcrmnt+(1.-dcrmnt)*ran0(idum))
      yinit(if)=y(if)
250   continue
260   continue
c
c-----call the iterative solution subroutine
c
      itchk=0
      call newt(y,nf,check,itchk)
c
      if(itchk .ne. 0)write(11,9205)
      if(itchk .ne. 0)go to 880
c
c-----update the x vector using jsinv and gstar
c
      is=0
      do 500 i=1,n+m
```

```
    if(desflg(i) .ne. 0)go to 500
    is=is+1
    x(i)=y(is)
500 continue
c
c-----calculate the structural error of the mechanism obtained
c-----from this outer iteration
c
    go to 860
    fmerit=x(7)**2
    if(fmerit .gt. 1.e-8)go to 860
    call fcner1(x,n,m,phi,psi,lmax,struct,answer,error)
    errmax=0.
    do 850 kk=1,lmax
    if(error(kk)**2 .gt. errmax)errmax=error(kk)**2
850 continue
    errmax=sqrt(errmax)*180./pi
    go to 880
860 continue
    struct=1000000.
    errmax=1000000.
880 continue
c
c-----convert the input and output shaft angles to degrees
c-----counterclockwise from horizontal
c
    x(5)=(x(5)-pi)*180./pi
```

```
      x(6)=(2.*pi-x(6))*180./pi
c
c-----write the results to the screen
c
      write(*,9102)(despar(i),x(i),i=1,n+m),struct,errmax
      write(11,9102)(despar(i),x(i),i=1,n+m),struct,errmax
c
1000  continue
c
      close(11)
c
      stop
c
9003  format(' enter the maximum number of iterations')
9004  format('do you want to specify ',a6,'? y or n')
9005  format('enter ',a6)
9006  format('enter initial guess for ',a6)
9007  format('enter the iteration decrement parameter')
9008  format(a1)
c
9051  format(i2)
9052  format(2f10.0)
c
9101  format(8(1pe10.2))
9102  format(4(a6,'=',1pe10.2,1x)/2(a6,'=',1pe10.2,1x)
     1/a6,'=',1pe10.2,'  rms structural error = ',1pe10.2
     2/'maximum error = ',1pe10.2/1x)
```

```
c
9201  format(' phi',i2,' =',f10.2,' degrees     psi',i2,' =',f10.2
     1,' degrees')
9202  format(1x)
9203  format(a6,5x,1pe10.2,5x,a6)
9204  format('iteration decrement parameter = ',f6.3)
9205  format('MAXITS exceeded in newt'/1x)
c
      end
      SUBROUTINE newt(x,n,check,itchk)
      INTEGER n,nn,NP,MAXITS
      LOGICAL check
      REAL x(n),fvec,TOLF,TOLMIN,TOLX,STPMX
      PARAMETER (NP=40,MAXITS=200,TOLF=1.e-4,TOLMIN=1.e-6,TOLX=1.e-7,
     *STPMX=100.)
      COMMON /newtv/ fvec(NP),nn
      SAVE /newtv/
CU    USES fdjac,fmin,lnsrch,lubksb,ludcmp
      INTEGER i,its,j,indx(NP)
      REAL d,den,f,fold,stpmax,sum,temp,test,fjac(NP,NP),g(NP),p(NP),
     *xold(NP),fmin
      EXTERNAL fmin
      nn=n
      f=fmin(x)
      test=0.
      do 11 i=1,n
        if(abs(fvec(i)).gt.test)test=abs(fvec(i))
```

```
11   continue
    if(test.lt..01*TOLF)return
    sum=0.
    do 12 i=1,n
      sum=sum+x(i)**2
12   continue
    stpmax=STPMX*max(sqrt(sum),float(n))
    do 21 its=1,MAXITS
    write(*,1511)its,test
      call fdjac(n,x,fvec,NP,fjac)
      do 14 i=1,n
        sum=0.
        do 13 j=1,n
          sum=sum+fjac(j,i)*fvec(j)
13        continue
        g(i)=sum
14      continue
      do 15 i=1,n
        xold(i)=x(i)
15      continue
      fold=f
      do 16 i=1,n
        p(i)=-fvec(i)
16      continue
      call ludcmp(fjac,n,NP,indx,d)
      call lubksb(fjac,n,NP,indx,p)
      call lnsrch(n,xold,fold,g,p,x,f,stpmax,check,fmin)
      test=0.
      do 17 i=1,n
        if(abs(fvec(i)).gt.test)test=abs(fvec(i))
17      continue
      if(test.lt.TOLF)then
        check=.false.
        return
      endif
      if(check)then
        test=0.
        den=max(f,.5*n)
        do 18 i=1,n
          temp=abs(g(i))*max(abs(x(i)),1.)/den
          if(temp.gt.test)test=temp
18        continue
        if(test.lt.TOLMIN)then
          check=.true.
        else
          check=.false.
        endif
        return
      endif
      test=0.
      do 19 i=1,n
        temp=(abs(x(i)-xold(i)))/max(abs(x(i)),1.)
        if(temp.gt.test)test=temp
19      continue
```

```
      if(test.lt.TOLX)return
21    continue
c     pause 'MAXITS exceeded in newt'
1511  format(i5,1pe10.2)
      itchk=1000
      return
      END
C  (C) Copr. 1986-92 Numerical Recipes Software )=3&W#R2.
      SUBROUTINE ludcmp(a,n,np,indx,d)
      INTEGER n,np,indx(n),NMAX
      REAL d,a(np,np),TINY
      PARAMETER (NMAX=500,TINY=1.0e-20)
      INTEGER i,imax,j,k
      REAL aamax,dum,sum,vv(NMAX)
      d=1.
      do 12 i=1,n
        aamax=0.
        do 11 j=1,n
          if (abs(a(i,j)).gt.aamax) aamax=abs(a(i,j))
11      continue
        if (aamax.eq.0.) pause 'singular matrix in ludcmp'
        vv(i)=1./aamax
12    continue
      do 19 j=1,n
        do 14 i=1,j-1
          sum=a(i,j)
          do 13 k=1,i-1
        sum=sum-a(i,k)*a(k,j)
13      continue
      a(i,j)=sum
14    continue
    aamax=0.
    do 16 i=j,n
      sum=a(i,j)
      do 15 k=1,j-1
        sum=sum-a(i,k)*a(k,j)
15      continue
      a(i,j)=sum
      dum=vv(i)*abs(sum)
      if (dum.ge.aamax) then
        imax=i
        aamax=dum
      endif
16    continue
    if (j.ne.imax)then
      do 17 k=1,n
        dum=a(imax,k)
        a(imax,k)=a(j,k)
        a(j,k)=dum
17      continue
      d=-d
      vv(imax)=vv(j)
    endif
    indx(j)=imax
```

```
      if(a(j,j).eq.0.)a(j,j)=TINY
      if(j.ne.n)then
        dum=1./a(j,j)
        do 18 i=j+1,n
          a(i,j)=a(i,j)*dum
18        continue
      endif
19    continue
      return
      END
C  (C) Copr. 1986-92 Numerical Recipes Software )=3&W#R2.
      SUBROUTINE lubksb(a,n,np,indx,b)
      INTEGER n,np,indx(n)
      REAL a(np,np),b(n)
      INTEGER i,ii,j,ll
      REAL sum
      ii=0
      do 12 i=1,n
        ll=indx(i)
        sum=b(ll)
        b(ll)=b(i)
        if (ii.ne.0)then
          do 11 j=ii,i-1
            sum=sum-a(i,j)*b(j)
11        continue
        else if (sum.ne.0.) then
          ii=i
        endif
        b(i)=sum
12      continue
      do 14 i=n,1,-1
        sum=b(i)
        do 13 j=i+1,n
          sum=sum-a(i,j)*b(j)
13        continue
        b(i)=sum/a(i,i)
14    continue
      return
      END
C  (C) Copr. 1986-92 Numerical Recipes Software )=3&W#R2.
      SUBROUTINE lnsrch(n,xold,fold,g,p,x,f,stpmax,check,func)
      INTEGER n
      LOGICAL check
      REAL f,fold,stpmax,g(n),p(n),x(n),xold(n),func,ALF,TOLX
      PARAMETER (ALF=1.e-4,TOLX=1.e-7)
      EXTERNAL func
CU    USES func
      INTEGER i
      REAL a,alam,alam2,alamin,b,disc,f2,fold2,rhs1,rhs2,slope,sum,temp,
     *test,tmplam
      check=.false.
      sum=0.
      do 11 i=1,n
        sum=sum+p(i)*p(i)
```

```
11   continue
    sum=sqrt(sum)
    if(sum.gt.stpmax)then
      do 12 i=1,n
        p(i)=p(i)*stpmax/sum
12      continue
    endif
    slope=0.
    do 13 i=1,n
      slope=slope+g(i)*p(i)
13   continue
    test=0.
    do 14 i=1,n
      temp=abs(p(i))/max(abs(xold(i)),1.)
      if(temp.gt.test)test=temp
14   continue
    alamin=TOLX/test
    alam=1.
1    continue
      do 15 i=1,n
        x(i)=xold(i)+alam*p(i)
15      continue
      f=func(x)
      if(alam.lt.alamin)then
        do 16 i=1,n
          x(i)=xold(i)
16        continue
```

```
  check=.true.
  return
 else if(f.le.fold+ALF*alam*slope)then
  return
 else
  if(alam.eq.1.)then
    tmplam=-slope/(2.*(f-fold-slope))
  else
    rhs1=f-fold-alam*slope
    rhs2=f2-fold2-alam2*slope
    a=(rhs1/alam**2-rhs2/alam2**2)/(alam-alam2)
    b=(-alam2*rhs1/alam**2+alam*rhs2/alam2**2)/(alam-alam2)
    if(a.eq.0.)then
      tmplam=-slope/(2.*b)
    else
      disc=b*b-3.*a*slope
      tmplam=(-b+sqrt(disc))/(3.*a)
    endif
    if(tmplam.gt..5*alam)tmplam=.5*alam
  endif
 endif
 alam2=alam
 f2=f
 fold2=fold
 alam=max(tmplam,.1*alam)
goto 1
END
```

```
C  (C) Copr. 1986-92 Numerical Recipes Software )=3&W#R2.
      SUBROUTINE fdjac(n,x,fvec,np,df)
      INTEGER n,np,NMAX
      REAL df(np,np),fvec(n),x(n),EPS
      PARAMETER (NMAX=40,EPS=1.e-4)
CU    USES funcv
      INTEGER i,j
      REAL h,temp,f(NMAX)
      do 12 j=1,n
        temp=x(j)
        h=EPS*abs(temp)
        if(h.eq.0.)h=EPS
        x(j)=temp+h
        h=x(j)-temp
        call funcv(n,x,f)
        x(j)=temp
        do 11 i=1,n
          df(i,j)=(f(i)-fvec(i))/h
11      continue
12    continue
      return
      END
C  (C) Copr. 1986-92 Numerical Recipes Software )=3&W#R2.
      FUNCTION fmin(x)
      INTEGER n,NP
      REAL fmin,x(*),fvec
      PARAMETER (NP=40)
      COMMON /newtv/ fvec(NP),n
      SAVE /newtv/
CU    USES funcv
      INTEGER i
      REAL sum
      call funcv(n,x,fvec)
      sum=0.
      do 11 i=1,n
        sum=sum+fvec(i)**2
11    continue
      fmin=0.5*sum
      return
      END
C  (C) Copr. 1986-92 Numerical Recipes Software )=3&W#R2.
      FUNCTION ran0(idum)
      INTEGER idum,IA,IM,IQ,IR,MASK
      REAL ran0,AM
      PARAMETER (IA=16807,IM=2147483647,AM=1./IM,IQ=127773,IR=2836,
     *MASK=123459876)
      INTEGER k
      idum=ieor(idum,MASK)
      k=idum/IQ
      idum=IA*(idum-k*IQ)-IR*k
      if (idum.lt.0) idum=idum+IM
      ran0=AM*idum
      idum=ieor(idum,MASK)
      return
      END
C  (C) Copr. 1986-92 Numerical Recipes Software $2$11-.
      subroutine funcv(nf,y,gstar)
c
      real y(nf),gstar(nf),x(7),g(7),phi(50),psi(50),lambda
      integer desflg(7)
c
      common/angs/phi,psi,lmax
      common/desprm/x,desflg
c
c-----define y as a subset of the vector x
c
      is=0
      do 20 i=1,7
      if(desflg(i) .ne. 0)go to 20
      is=is+1
      x(i)=y(is)
20    continue
c
c-----initialize the residual vector g
c
      do 50 i=1,7
      g(i)=0.
50    continue
c
c-----define terms to be used in calculations
c
```

```
c-----link lengths
      a1=x(1)
      a2=x(2)
      a3=x(3)
      a4=x(4)
c-----initial shaft angles
      t1=x(5)
      t4=x(6)
c-----lagrange multiplier
      lambda=x(7)
c-----trigonometric functions
      ct1=cos(t1)
      st1=sin(t1)
      ct4=cos(t4)
      st4=sin(t4)
      t1t4=t1+t4
      ct1t4=cos(t1+t4)
      st1t4=sin(t1+t4)
c
c-----evaluation of residual vector
c-----sum partial derivative terms at each synthesis point
c
      do 100 l=1,lmax
c
c-----deine terms to be used in calculations
c
      t1phi=t1+phi(l)
      t4psi=t4+psi(l)
c
      ct1phi=cos(t1phi)
      st1phi=sin(t1phi)
      ct4psi=cos(t4psi)
      st4psi=sin(t4psi)
c
      h=a4+a3*ct4psi+a1*ct1phi
      v=a3*st4psi-a1*st1phi
c
      q=sqrt(h**2+v**2)
      p=1.-a2/q
c
c-----also evaluate the first n components of the residual vector g
c
      g(1)=g(1)+2.*p*(h*ct1phi-v*st1phi)
      g(2)=g(2)+2.*(a2-q)
      g(3)=g(3)+2.*p*(h*ct4psi-v*st4psi)
      g(4)=g(4)+2.*h*p
      g(5)=g(5)-2.*a1*p*(h*st1phi+v*ct1phi)
      g(6)=g(6)-2.*a3*p*(h*st4psi-v*ct4psi)
c
100   continue
c
c-----calculate the constraint components of g
c
      g(1)=g(1)+lambda*2.*(a1+a3*ct1t4+a4*ct1)
      g(2)=g(2)-lambda*2.*a2
      g(3)=g(3)+lambda*2.*(a3+a1*ct1t4+a3*ct4)
      g(4)=g(4)+lambda*2.*(a4+a1*ct1+a3*ct4)
      g(5)=g(5)-lambda*2.*a1*(a3*st1t4+a4*st1)
      g(6)=g(6)-2.*a3*(a1*st1t4+a4*st4)
      g(7)=a1**2+a3**2+a4**2-a2**2+2.*a1*a3*ct1t4+2.*a1*a4*ct1
     1+2.*a3*a4*ct4
c
c-----define the gstar vector, which consists of the entries of g that
c-----correspond to the unspecified design parameters
c
      is=0
      do 200 i=1,7
      if(desflg(i) .ne. 0)go to 200
      is=is+1
      gstar(is)=g(i)
200   continue
c
      return
      end
      subroutine fcner1(x,n,m,phi,psi,lmax,struct,answer,error)
c
      real x(n+m),phi(lmax),psi(lmax),answer(lmax),error(lmax)
c
      pi=acos(-1.)
c
c-----define design parameters in terms of x
```

```
c
      a1=x(1)
      a2=x(2)
      a3=x(3)
      a4=x(4)
      thet1=(x(5)-pi)
      thet4=2.*pi-x(6)
c
      struct=0.
c
      xold=a4+a3*cos(thet4)
      yold=a3*sin(thet4)
c
      do 100 istep=1,lmax
      theta=thet1+phi(istep)
      c1=a1*cos(theta)
      d1=a1*sin(theta)
      if(d1 .eq. 0.)go to 30
      p=(a4-c1)/d1
      q=(a3**2-a2**2-a4**2+c1**2+d1**2)/(2.*d1)
c
      x1=((a4-p*q)+sqrt((a4-p*q)**2-(1.+p**2)*(a4**2+q**2-a3**2)))
     1/(1.+p**2)
      x2=((a4-p*q)-sqrt((a4-p*q)**2-(1.+p**2)*(a4**2+q**2-a3**2)))
     1/(1.+p**2)
      y1=p*x1+q
      y2=p*x2+q
      go to 60
30    continue
      x1=(a3**2-a2**2+c1**2-a4**2)/(2.*(c1-a4))
      x2=x1
      y1=sqrt(a3**2-(x1-a4)**2)
      y2=-1.*y1
60    continue
      dist1=(x1-xold)**2+(y1-yold)**2
      dist2=(x2-xold)**2+(y2-yold)**2
      xnew=x1
      ynew=y1
      if(dist2 .lt. dist1)xnew=x2
      if(dist2 .lt. dist1)ynew=y2
c
      cth4=(xnew-a4)/a3
      sth4=ynew/a3
      th4=acos(cth4)
      if(sth4 .lt. 0.)th4=-1.*th4
      angout=th4-thet4
      if(angout .gt. 2.*pi)angout=angout-2.*pi
      if(angout .le. -2.*pi)angout=angout+2.*pi
      answer(istep)=angout
      error(istep)=angout+psi(istep)
      struct=struct+(angout+psi(istep))**2
100   continue
c
c-----convert the structural error, which is a sum of squares,
c-----to a root mean square term,and convert from radians to degrees
c
      struct=(180./pi)*sqrt(struct/(1.*lmax))
c
      return
c
      end
```

REFERENCES

1. Shigley, J. and J. Uicker, *Theory of Machines and Mechanisms*, McGraw-Hill, New York (1980): pp. 331-336

2. Sandor, G. N. and A. Erdman, *Advanced Mechanism Design*, Prentice-Hall, Englewood Cliffs, NJ (1984)

3. Freudenstein, F. "An Analytical Approach to the Design of Four-Link Mechanisms", *Trans. ASME* 76 (1954): pp. 483-492

4. Freudenstein, F., "Structural Error Analysis in Plane Kinematic Synthesis", *ASME Journal of Engineering for Industry*, vol 81 no. 1 (January 1959): pp. 15-22

5. Sarkisyan, Y. L., Gupta, K. C. and B. Roth, "Kinematic Geometry Associated with the Least-Square Approximation of a Given Motion", *ASME Journal of Engineering for Industry*, Vol. 95 (May 1973): pp. 503-510

6. Gupta, K. C. and B. Roth, "A General Approximation Theory for Mechanism Synthesis", *ASME Journal of Applied Mechanics*, vol. 42 (June 1975): pp. 451-457

7. Alizade, R.I., Novruzbekov, I. G. and G. N. Sandor, "Optimization of Four-Bar Function Generating Mechanisms Using Penalty Functions with Inequality and Equality Constraints", *Mechanism and Machine Theory*, vol. 10 (1975): pp. 327-336

8. Alizade, R. I., Rao, A. V. and G. N. Sandor, "Optimum Synthesis of Four-Bar and Offset Slider-Crank Planar and Spatial Mechanisms Using the Penalty Function Approach with Inequality and Equality Constraints", *ASME Journal of Engineering for Industry*, (August 1975): pp. 785-790

9. Kramer, S. N. and G. N. Sandor, "Selective Precision Synthesis - A General Method of Optimization for Planar Mechanisms", *ASME Journal of Engineering for Industry* (May 1975): pp. 689-701

10. Pugh, J. T., "Synthesis of Pareto Optimal Four-Bar Function Generators with Optimum Structural Error and Optimum Transmission Angles", *ASME Journal of Mechanisms, Transmissions and Automation in Design*, vol. 106 no. 4 (December 1984): pp. 437-443

11 Aviles, R., Ajuria, M. B. and J. Garcia de Jalon, "A Fairly General Method for Optimum Synthesis of Planar Mechanisms", *Mechanism and Machine Theory*, Vol. 20 No. 4 (1985): pp. 321-328

12. Ravani, B. and B. Roth, "Motion Synthesis Using Kinematic Mappings", *ASME Journal of Mechanisms, Transmissions and Automation in Design*, Vol 105 (September 1983): pp. 460-467

13. Wu, Y. K. and I. Fischer, "Motion Synthesis of Mechanisms Using Constraint Manifolds", *DE vol. 70 Mechanism Synthesis and Analysis* (1994): pp. 441-448

14. Williamson, Crowell and Trotter, *Calculus of Vector Functions*, 3rd edition, Prentice-Hall, Englewood Cliffs, NJ (1972)

15. Isaacson, E. and H. B. Kelly, *Analysis of Numerical Methods*, John Wiley, New York (1966)

16. Press, Vetterling, Teukolsky and Flannery, *Numerical Recipes in FORTRAN*, Cambridge Press, New York (1992): pp. 376-386

17. Freudenstein, F., "Four-Bar Function Generators", *Machine Design*, (November 27, 1958) pp. 119-123