BRACHISTOCHRONE PROBLEM SOLVED BY INVARIANT IMBEDDING, DYNAMIC PROGRAMMING, AND QUASILINEARIZATION METHODS

A Thesis

Presented to

the Faculty of the Department of Mechanical Engineering

University of Houston

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Mechanical Engineering

---

by

Moo-Zung Lee

June, 1966

ACKNOWLEDGEMENT

The author wishes to express his sincere gratitude to his adviser Dr. D. Muster, Professor and Chairman of the Department of Mechanical Engineering, University of Houston, for his encouragement, guidance and careful arrangements of discussions with several people during the study and writing of this thesis. Among these, the author is particularly in debt to Dr. R. E. Kalaba of the Rand Corporation who suggested this problem with keys to the solution and contributed many valuable references. Assistance received from Drs. I. Organick, S. R. Parker, and S. B. Childs, (all of the University of Houston), is greatly appreciated.

BRACHISTOCHRONE PROBLEM SOLVED BY INVARIANT IMBEDDING, DYNAMIC PROGRAMMING, AND QUASILINEARIZATION METHODS

An Abstract of a Thesis

Presented to

the Faculty of the Department of Mechanical Engineering

University of Houston

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Mechanical Engineering

by

Moo-Zung Lee

June, 1966

ABSTRACT

In such fields of current interest as optimal control and orbit determination, non-linear two-point boundary-value problems arise, the numerical solutions for which are difficult to obtain. In this thesis, some of the useful tools for treating problems of this nature - invariant imbedding, dynamic programming, and quasilinearization are studied by means of the brachistochrone problem. The three approaches are used separately and in combination. Computer programs using MAD language are presented. The results are compared with the classical solutions.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF FIGURES (con't)

LIST OF TABLES

LIST OF PROGRAMS

LIST OF SYMBOLS

Symbol	Definition
a	Initial position along x-axis
A	Starting point
b_1, b_2	Constants
B	Terminal point
c	Initial state
d	Interpolated value of state variable
delx, dx, Δx	Small increment of x
dely, dy, Δy	Small increment of y
ds	Infinitesimal chord length
dt	Infinitesimal time
f	Optimal function
F	Functional
g	Constant of gravitational acceleration
G	Functional
h_1, h_2	Homogeneous solution
i	State counter
j	State counter
k	Stage counter
l,m,n	Integer constants
O	The origin
p	Particular solution
q	State counter
Qk	Stage counter in quasilinearization

LIST OF SYMBOLS (con't)

Symbol	Definition
r	Slope function (text)
r	Radius of base circle (appendix)
t	Time
u	State variable
u_o	Starting value of u
u_T	Terminal value of u
V	Velocity
w	Slope
x	Independent variable
x_o	Starting value of x
x_T	Terminal value of x
y	Dependent variable
y_o	Starting value of y
y_T	Terminal value of y
θ	Angular displacement of base circle
ω	Angular velocity of base circle

CHAPTER I

INTRODUCTION

1.1 INITIAL-VALUE PROBLEM AND BOUNDARY-VALUE PROBLEM

Consider a second order ordinary differential equation

$$y'' = G(y,y') \tag{1.1-1}$$

with initial conditions

$$\begin{aligned} y(0) &= c_1 \qquad \text{(a)} \\ y'(0) &= c_2 \qquad \text{(b)} \end{aligned} \tag{1.1-2}$$

The determination of a solution to Eq.(1.1-1) subject to conditions Eq.(1.1-2) is known as an initial-value problem. By putting $u=y$, $w=y'$, Eqs.(1.1-1) and (1.1-2) become

$$\begin{aligned} u' &= w, \qquad & u(0) &= c_1 \qquad \text{(a)} \\ w' &= G(u,w), \qquad & w(0) &= c_2 \qquad \text{(b)} \end{aligned} \tag{1.1-3}$$

which are integrable directly.

Modern electronic computers provide the means for obtaining numerical solutions of systems of simultaneous non-linear (or linear) ordinary differential equations subject to a set of initial conditions, with accuracy and speed. However, in some fundamental problems the constraints are not initial values but are in the form

$$\begin{aligned} u' &= w, \qquad & u(0) &= c_1 \qquad \text{(a)} \\ w' &= G(u,w), \qquad & w(x_T) &= c_3 \qquad \text{(b)} \end{aligned} \tag{1.1-4}$$

where x_T is the terminal value of the independent variable x.

The problem is called a two-point boundary-value problem, since values are prescribed at two distinct points, $x=0$ and $x=x_T$.

1.2 THE BRACHISTOCHRONE PROBLEM [1]

As an example of a two-point boundary-value problem, the differential equation of brachistochrone problem is derived as follows:

Given two points in a space containing a constant gravitational force field, we wish to find a frictionless path from a higher point to a lower point along which a particle will slide in minimum time.

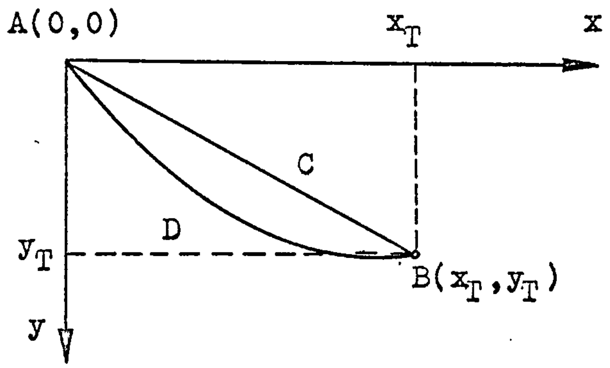


Figure 1.2-1

Possible Paths for the Least Time

In Fig. 1.2-1, It is obvious that the particle will

---

[1] From Greek, βράχιστος, shortest and χρόνος, time, a term invented by Jean Bernoulli (1667-1748) in 1694 to denote a curve along which a body passes from one fixed point to another in the shortest time. When the directive force is constant, the curve is a cycloid.

traverse minimum distance along the straight-line path ACB. Along the curved path ADB the particle picks up speed sooner, but travels a longer route. The optimal path of least time may be found by balancing these considerations properly.

Let us denote the initial point as the origin, set up a coordinate system as shown in Fig. 1.2-1 and call the terminal point (x_T, y_T). We know that the particle velocity, V, in the plane of the field, is equal to $\sqrt{2gy}$ at any position in the field, independent of its horizontal position. Since an infinitesimal arc length, ds is given by

$$ds = \left[(dx)^2 + (dy)^2\right]^{1/2} = \sqrt{1+(y')^2}\cdot dx,$$

the time of descent is expressed by

$$T = \int_0^{x_T} \frac{ds}{V} = \int_0^{x_T} \left[\frac{1+y'^2}{2gy}\right]^{1/2} dx \tag{1.2-1}$$

where g is the gravitational constant. We seek a function $y=y(x)$ which satisfies the constraint conditions $y(0)=0$, $y(x_T)=y_T$, and which minimizes the integral T.

The Euler equation for Eq.(1.2-1) is

$$2yy'' + y'^2 + 1 = 0 \tag{1.2-2}$$

or in the form of Eq.(1.1-1)

$$y'' = -\frac{1+y'^2}{2y} \tag{1.2-3}$$

subject to the boundary conditions

$$y(0) = 0 \quad \text{(a)}$$
$$y(x_T) = x_T \quad \text{(b)} \qquad (1.2\text{-}4)$$

1.3 A NUMERICAL SOLUTION OF TWO-POINT BOUNDARY-VALUE PROBLEM

In order to solve an n-th-order ordinary differential equation numerically, ordinary computing techniques call for a knowledge of y, y', y", ... $y^{(n-1)}$ at either the starting point x=0 or the terminal point $x=x_T$. In the brachistochrone problem, we have one value at one end and another at the other.

In order to solve a problem of this nature, we may choose a value of y'(0), say c_4, and integrate the equation using $y(0)=c_1$, $y'(0)=c_4$ as initial values. If the value at the terminal point, $y=y(x_T)$ obtained in this way agrees sufficiently closely with the desired value y_T, we accept this as the solution. Otherwise, we vary the value of c_4 and recompute the terminal value until agreement at the boundary is satisfactory.

This is not an ideal procedure for a number of reasons. First, it is difficult to estimate in advance the required amount of computing time which will be needed. Second, stipulating a certain accuracy at the end point does not guarantee equal accuracy throughout whole range of x, from x=0 to $x=x_T$. Third, the results obtained from the i-th iteration

$$y(k)_i = y\left[x(k)\right]_i \qquad \text{for } 0 \leq x(k)=k\cdot\Delta x \leq x_T \qquad (1.3\text{-}1)$$

are not utilized to improve the solution in the (i+1)-th try. In addition, a proper first estimate of the solution may be difficult to establish.

1.4 RECENT APPROACHES

As we shall see in the following chapters, theories of invariant imbedding and dynamic programming transform boundary-value problems to initial-value problems by introducing new state variables, and imbedding a specific problem in a family of similar problems. Invariant imbedding provides information of initial slopes from given terminal slopes in a very short computing time. The Euler equations obtained in the course of applying calculus of variations are, in most cases, difficult to solve; dynamic programming provides a means of by-passing this hurdle. On the other hand, quasilinearization attacks these problems by linear approximation techniques combined with a concept analogous to making approximations in policy space [14].[2] The approximations are constructed to yield rapid and monotone convergence.

The theory and techniques mentioned above were developed mainly by Bellman, Kalaba and their colleagues [3-21,24].

[2] Number in bracket refers to identically numbered references in the bibliography.

CHAPTER II

INVARIANT IMBEDDING

2.1 PRINCIPLE OF INVARIANT IMBEDDING

In 1943, Ambarzumian introduced a new approach to the study of atmospheric scattering problems [1]. This approach was extended by Chandrasekhar who gave it the name "principle of invariance" [2]. In recent years, Bellman and Kalaba generalized this methodology and called it "the principle of invariant imbedding" [3]. It can be stated as follows:

> "Given a physical system, S, whose state at any time t is specified by a state vector, x, we consider a process which consists of a family of transformations applied to this state vector.
>
> Suitably enlarging the dimension of the original vector by means of additional components, the state vectors are made elements of a space which is mapped into itself by the family of transformations. In this way we obtain an invariant process, by imbedding the original process within the new family of processes. The functional equations governing the new process are the analytic expression of this invariance."

In other words, we derive equations for the values of the dependent variables at a fixed value of the independent variable as a function of interval on which the boundary value problems are specified.

Many applications of this theory in such diverse areas as radiative transfer, neutron transport, diffusion and heat conduction, scattering and random walk, and wave propagation can be found in recent literature [3,5,6,7,8]. In this report, the fundamental technique is applied to a problem well-known in classical calculus of variations.

2.2 IMBEDDING PARTICULAR PROBLEM IN A FAMILY OF PROBLEMS

In the study of a spring-mass system, customarily we write $y=y(t)$, indicating the dependence of the solution upon t. More generally, the solution is also a function of c, the initial value of y; hence, we write $y=y(c,t)$. This implies that the study of a particular solution of a differential equation may be carried out by studying a family of solutions. It also constitutes the keystone of the theory of invariant imbedding and forms the base for the theory of dynamic programming.

Although imbedding a particular problem in a family of problems appears to complicate rather than simplify the problem, its justification lies in the fact that we can construct a bridge spanning the particular problem and other members of the family, which is utilized to determine the characteristics of the particular member of the family.

2.3 BRACHISTOCHRONE PROBLEM WITH FREE-END CONDITIONS

A brachistochrone path connecting the initial point $A(0,c)$ and any point on the terminal line $x=B$ is characterized by minimizing the functional

$$T = \int_0^B \sqrt{\frac{1 + (y')^2}{2gy}}\, dx \tag{2.3-1}$$

where the dependent variable is subject to the initial condition

$$y(0) = c \qquad (2.3\text{-}2)$$

and y is free at the terminal line x=B. Such a problem is said to have one variable end point.

From Eq.(1.2-3), the optimal path is the solution of the Euler equation

$$y'' = -\frac{1+y'^2}{2y} \qquad (2.3\text{-}3)$$

subject to initial condition $y(0)=c$. The other boundary value is not given explicitly; however, from the statement of the problem and the fact that the minimum-time path from any point on the terminal line to the terminal line itself is equal to zero, we have the so-called natural boundary condition[14]

$$y'(B) = 0 \qquad (2.3\text{-}4)$$

We seek to find the missing initial value $y'(0)$. so that we can integrate Eq.(2.3-3) directly to obtain a solution. In the following section we show how to compute, by invariant imbedding, the missing initial slopes from the given terminal slopes.

2.4 DERIVATION OF EQUATIONS [18]

We rewrite Eq.(1.1-3) with $c_1=0$, $c_2=0$. that is,

$$\begin{aligned} u' &= w, \qquad & u(0) &= c \qquad & \text{(a)} \\ w' &= G(u,w), \qquad & w(x_T) &= 0 \qquad & \text{(b)} \end{aligned} \qquad (2.4\text{-}1)$$

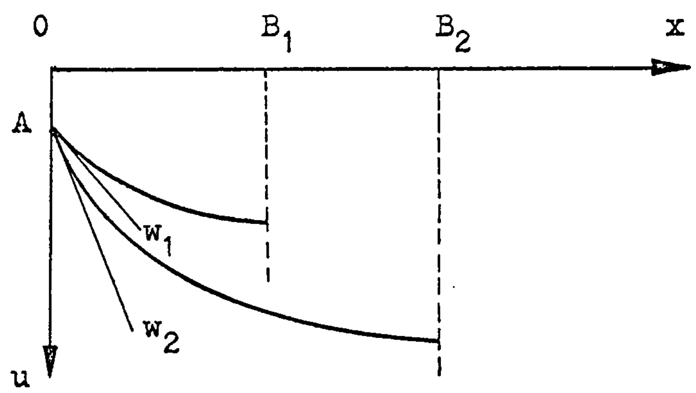


Figure 2.4-1

Initial Slope and the Range of

Independent Variable

From Fig. 2.4-1 we can see that, for similar problems, the initial slopes depend upon the range of the independent variable x. Initial slope $u'(0)=w_1$ is optimum for $x_T=B_1$, while $u'(0)=w_2$ is proper for $x_T=B_2$ [3]. If we fix x_T at B, and consider various starting points at x=a along x-axis, then the initial slope at x=a is a function of a (Fig.2.4-2). We write

$$u'(a) = r(a) \qquad \text{for } 0 \leq a \leq x_T \qquad (2.4\text{-}2)$$

By permitting the parameter a to vary from x_T to 0, we construct a family of similar problems with different range of x for each member of the family. Furthermore, for a particular value of a, say $a=a_1$, the initial slopes differ

---

[3] At the cusps of a cycloid the slope is infinitely large, but here we must choose finite values for use in the computation. On this base we assume w(0) to be finite but large at the cusps.

according to the starting position $c=u(0)$. Therefore we write

$$u'(a) = w(a) = r(c,a) \qquad (2.4\text{-}3)$$

realizing that the correct slope depends upon the starting value of x as well as the initial position $u(x)$. By permitting c or a to vary, or c and a simultaneously, we actually investigate a family of problems of similar nature.

Let us assume the process begins at $x=a$, with slope b_1. After moving along the optimal path to $x=a+\Delta x$ the slope becomes b_2 (as is shown in Figs.2.4-3 and 2.4-4), and

$$w(a+\Delta x) = w(a) + w'(a)\cdot\Delta x + 0\left[(\Delta x)^2\right] \qquad (2.4\text{-}4)$$

Recall Eq.(2.4-3) and replace $w(a)$ by $r(c,a)$; we obtain

$$w(a+\Delta x) = r(c,a) + w'(a)\cdot\Delta x + 0\left[(\Delta x)^2\right] \qquad (2.4\text{-}5)$$

On the other hand, the general functional relationship Eq.(2.4-3) holds equally well for $x=a+\Delta x$, that is

$$w(a+\Delta x) = r(d,a+\Delta x) \qquad (2.4\text{-}6)$$

where d is the value of dependent variable u at $x=a+\Delta x$, which may be expressed by

$$\begin{aligned} d &= u(a+\Delta x) \\ &= u(a) + u'(a)\cdot\Delta x + 0\left[(\Delta x)^2\right] \\ &= c + w(a)\cdot\Delta x + 0\left[(\Delta x)^2\right] \\ &= c + r(c,a)\cdot\Delta x + 0\left[(\Delta x)^2\right] \end{aligned} \qquad (2.4\text{-}7)$$

We substitute Eq.(2.4-7) into Eq.(2.4-6) introduce the second

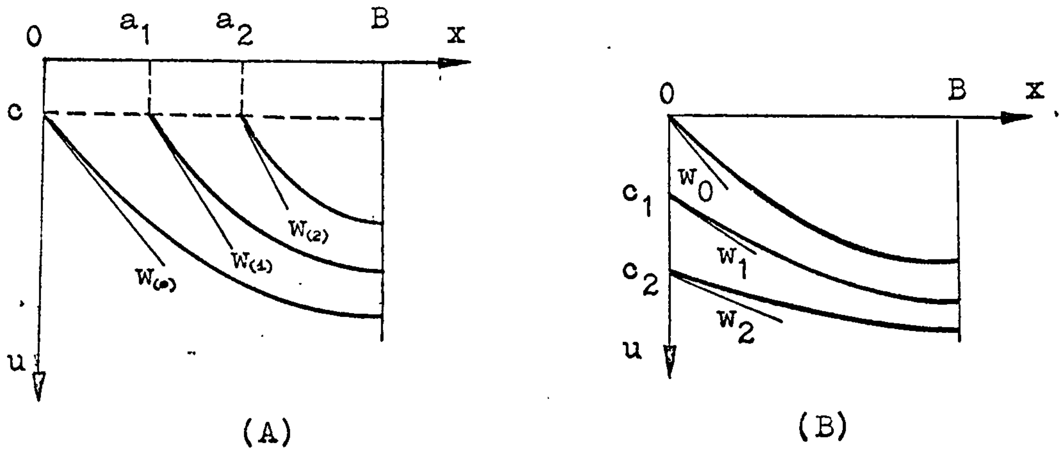


Figure 2.4-2

(A) w as a function of a

(B) w as a function of c

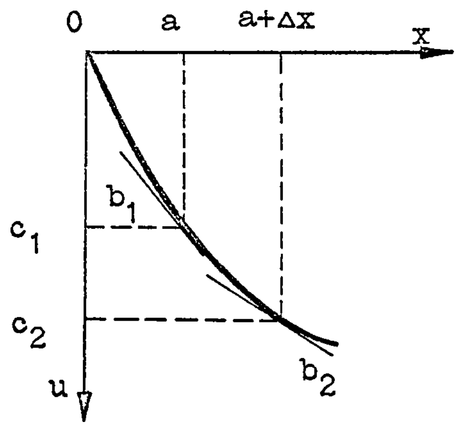


Figure 2.4-3

Slopes Along the Optimal Path in x-u Plane

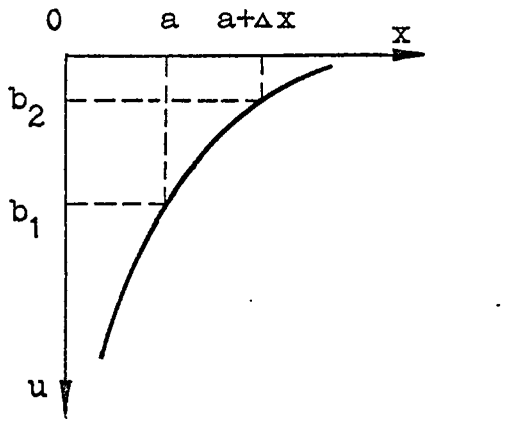


Figure 2.4-4

Slopes Along the Optimal path as a function of x expression of the slope at $x=a+\Delta x$ and obtain

$$w(a+\Delta x) = r\left[c+r(c,a),\ a+\Delta x\right] \tag{2.4-8}$$

By equating the right-hand sides of Eq.(2.4-5) and Eq.(2.4-8) we obtain

$$r(c,a) + w'(a)\cdot\Delta x = r\left[c+r(c,a)\cdot\Delta x,\ a+\Delta x\right] \tag{2.4-9}$$

In order to express $r(c,a)$ as a function of $r(c,a+\Delta x)$, let us take Δx sufficiently small and for the first approximation

$$r\left[c+r(c,a)\cdot\Delta x,\ a+\Delta x\right] \cong r\left[c+r(c,a+\Delta x)\cdot\Delta x,\ a+\Delta x\right] \tag{2.4-10}$$

to rewrite Eq.(2.4-9) as

$$r(c,a) = r\left[c+r(c,a+\Delta x)\cdot\Delta x,\ a+\Delta x\right] - w'(a+\Delta x)\cdot\Delta x \tag{2.4-11}$$

From the geometry of Fig. 2.4-5, if the slopes of curves passing through all grid points at $x=a+\Delta x$ are known, the slopes of different curves passing through grids at $x=a$ are computed as follows.

1. Take the slope at p, $w=r(c_i,a+\Delta x)$ as the first approximation of the slope at q.
2. Locate d by equation $d=c_i+r(c_i,a+\Delta x)\cdot\Delta x$.
3. Compute the slope of curve at d by linear interpolation of $r(c_i,a+\Delta x)$ and $r(c_{i+1},a+\Delta x)$.
4. Compute $r(c_i,a)$ using Eq.(2.4-11).
5. Repeat steps 1~4 for all other points at $x=a$.

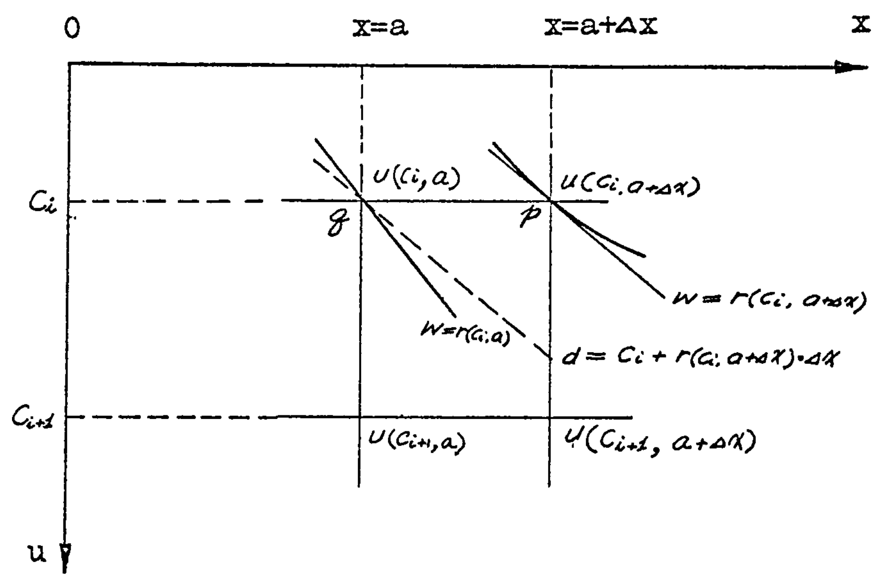


Figure 2.4-5

Geometry of Eq.(2.4-11)

6. Repeat steps 1~5 to regenerate the slopes for all grid points at the neighboring stage in the left-hand side.

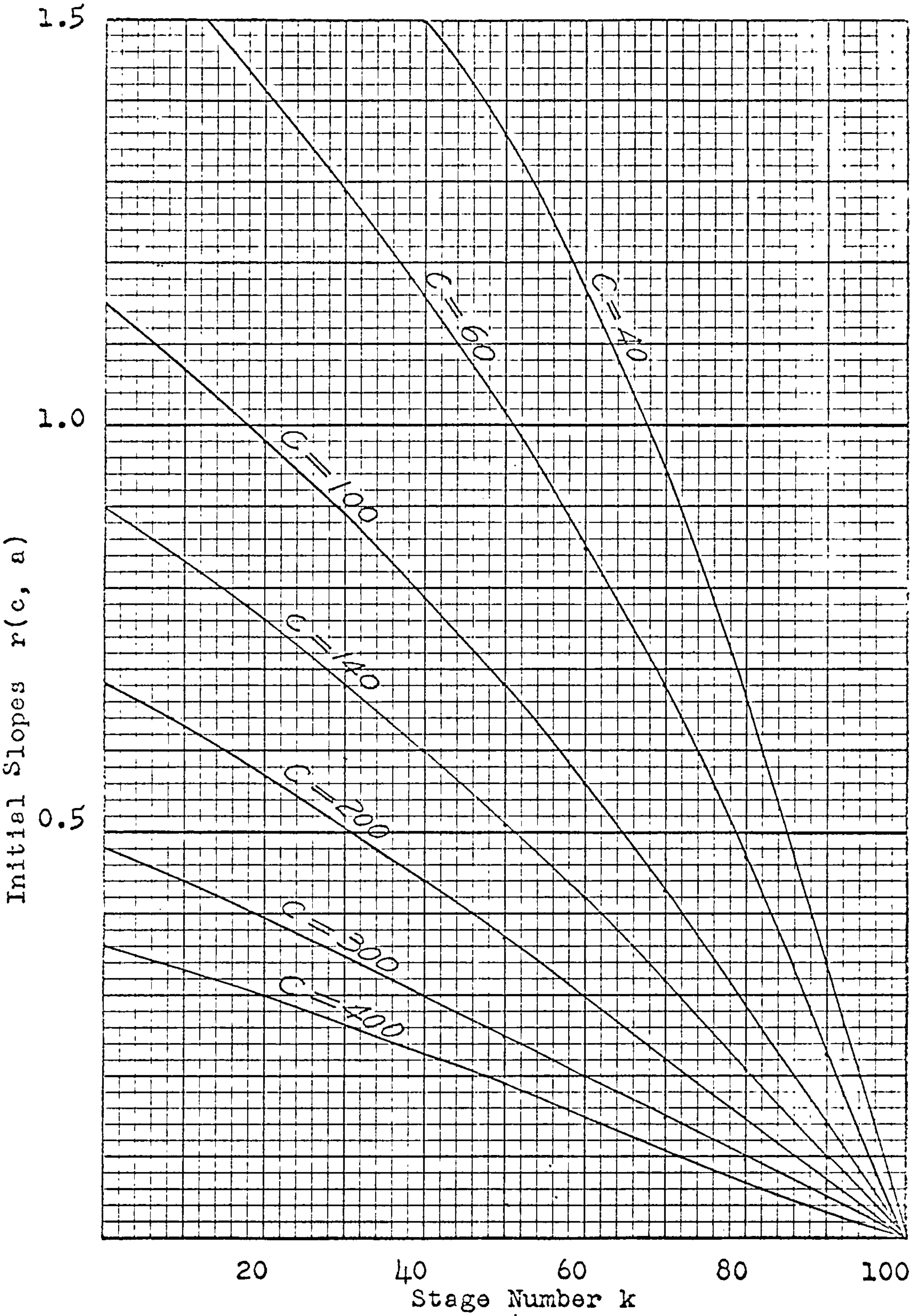
Using Eq.(2.4-11) with the free-end conditions $r(c_i, x_T)=0$, we can determine the slope function r at all grid points at $a = x_T - \Delta x$, $a = x_T - 2\Delta x$ and so on.

Consider the computing procedures outlined above. In step 2, we assigned $r(c_i, a+\Delta x)$ in predicting d; in step 3, both $r(c_i, a+\Delta x)$ and $r(c_{i+1}, a+\Delta x)$ contribute to the estimation of the slope of optimum curve passing through d. The position of d and its slope combined with Eq.(2.4-11) make estimation of $r(c_i, a)$ possible. The roles of the neighboring members of the family of the problems are obvious.

It is not wasteful to expand the dimension of the problem by invariant imbedding, because we imbed a difficult or unsolvable problem in a family of similar problems which become easier to handle after the mutual relations existing between the members of the group are used. As a byproduct, a series of problems are solved in one stroke instead of just obtaining a particular solution for a single problem. This series of results also supplies a more complete picture of the effect of each parameter on the resulting function.

As an example, a group of brachistochrone problems with $x=0\sim314.15926$, $u_T=0\sim400$ and with natural boundary conditions at terminal line were solved by taking 100 grids in both x and u axes. Computation of the initial slopes at various starting points of u at x=0 takes 6.1 sec execution time [4] using IBM 7094 computer. The results of 20 cases of initial slopes are compared with the analytical solution in Table 2-1. The computer program in MAD language used to obtain these results is shown in Program 2-1. In Fig.2.4-6 the initial slopes r(c,a) obtained from invariant imbedding are shown.

[4] In this thesis all computing times were obtained with programs using the same approach and philosophy. Change in either of these could produce significant changes in absolute computing times. On this basis, we have considered computing times as a criterion of comparison.



Initial Instants a = 100π(k/100)

Fig. 2.4-6 Initial Slopes Obtained from Invariant Imbedding

Table 2-1

Initial Slopes Obtained by Invariant Imbedding

Taking 100x100 grid points between
$x=0\sim100\pi$, $y=0\sim400$ feet

Grid Number	Starting Points	Initial Slopes (Invariant Imbedding)	Initial Slopes (Classical)
I	u(I)	w(I)	w(I)
5	.20000000E 02	.35818700E 01	.30228241E 01
10	.40000000E 02	.21314888E 01	.20489414E 01
15	.59999999E 02	.16331606E 01	.16062053E 01
20	.80000000E 02	.13481355E 01	.13373163E 01
25	.10000000E 03	.11561425E 01	.11514445E 01
30	.12000000E 03	.10146379E 01	.10131552E 01
35	.14000000E 03	.90489530E 00	.90529212E 00
40	.16000000E 03	.81680938E 00	.81835540E 00
45	.18000000E 03	.74431062E 00	.74657554E 00
50	.20000000E 03	.68348686E 00	.68620315E 00
55	.22000000E 03	.63167808E 00	.63467290E 00
60	.24000000E 03	.58699879E 00	.59015734E 00
65	.26000000E 03	.54806749E 00	.55131213E 00
70	.28000000E 03	.51384442E 00	.51712201E 00
75	.30000000E 03	.48352921E 00	.48680358E 00
80	.32000000E 03	.45648604E 00	.45974129E 00
85	.34000000E 03	.43213662E 00	.43544406E 00
90	.36000000E 03	.40979266E 00	.41351479E 00
95	.38000000E 03	.38868529E 00	.39362881E 00
100	.40000000E 03	.36815135E 00	.37551792E 00

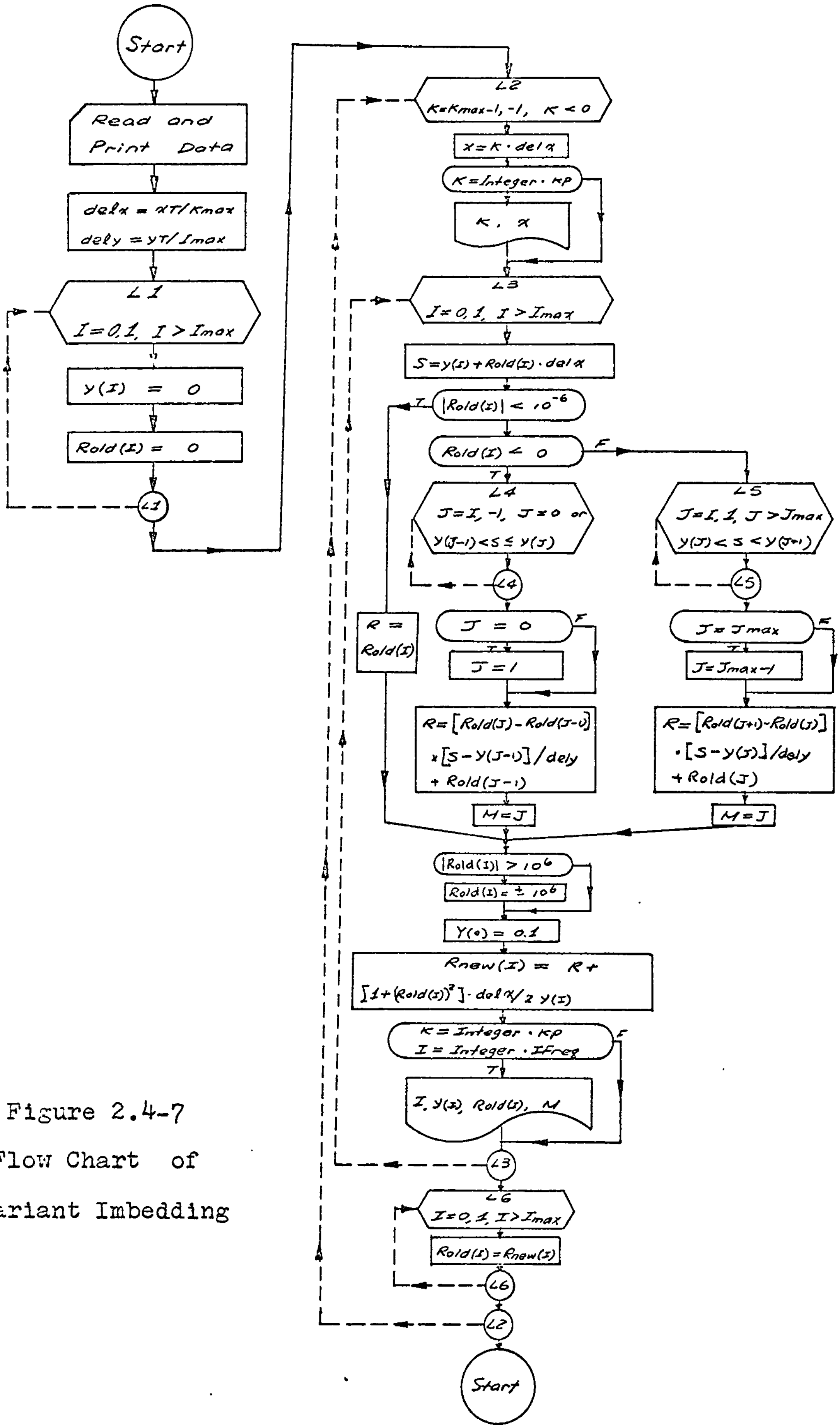


Figure 2.4-7

Flow Chart of

Invariant Imbedding

```
          R              P R O G R A M   2 - 1

          R    BRACHISTOCHRONE PROBLEM WITH FREE END CONDITIONS SOLVED BY
          R                       INVARIANT IMBEDDING

 $  COMPILE MAD, EXECUTE, PRINT OBJECT, DUMP
           INTEGER I, J, K, IMAX, JMAX, KMAX, KP, M, IFREQ
           DIMENSION Y(1000), ROLD(1000), RNEW(1000)
           EQUIVALENCE (IMAX, JMAX)
 START
           READ AND PRINT DATA     IMAX, KMAX, YT, XT, IFREQ
           DELX = XT/KMAX
           DELY = YT/IMAX
           THROUGH L1, FOR I=0,1,I.G.IMAX
           Y(I) = I*DELY
           ROLD(I) = 0.
 L1

           THROUGH L2, FOR K = (KMAX-1), -1, K.L. 0
           X = K*DELX
           WHENEVER K .E. 0
           PRINT RESULTS K , X
           PRINT COMMENT $              I                        Y(I)
          1 SLOPE          M $
           END OF CONDITIONAL

           THROUGH L3, FOR I=0, 1, I.G. IMAX
           S = Y(I) + ROLD(I)*DELX
           WHENEVER .ABS.(ROLD(I)).L. 1E-6
           R = ROLD(I)
           M = I
           OR WHENEVER ROLD(I) .L.0.
           THROUGH L4,FOR J=I,-1, J.E.0 .OR. (S.G.Y(J-1) .AND.S.LE.Y(J))
 L4
           WHENEVER J .E. 0
           J = 1
           END OF CONDITIONAL
           R = (ROLD(J)-ROLD(J-1))*(S-Y(J-1))/DELY + ROLD(J-1)
           M = J

           OTHERWISE
           THROUGH L5,FOR J=I,1,J.E.IMAX .OR.(S.G.Y(J) .AND.S.LE.Y(J+1))
 L5
           WHENEVER J.E. JMAX
           J = JMAX-1
           END OF CONDITIONAL
           R = (ROLD(J+1)-ROLD(J))*(S-Y(J))/DELY + ROLD(J)
           M = J
           END OF CONDITIONAL
```

```
            WHENEVER .ABS.(ROLD(I)) .G. 1E6
            ROLD(I) = 1E6*(ROLD(I)/(.ABS.(ROLD(I))))
            END OF CONDITIOANL
            Y(0) = 1.
            RNEW(I) = R+(1+ROLD(I)*ROLD(I))*DELX/(2*Y(I))
            WHENEVER K .E. 0 .AND. (I/IFREQ)*IFREQ .E. I
            PRINT FORMAT IMBED, I, Y(I), ROLD(I), M
            END OF CONDITIONAL
  L3
            THROUGH L6, FOR I = 0,1, I .G. IMAX
            ROLD(I) = RNEW(I)
  L6
  L2
            TRANSFER TO START
            VECTOR VALUES IMBED = $ 1I10, 2E20.8, 1I10   *$
            END OF PROGRAM
  $ DATA
   IMAX = 100, KMAX= 100,  YT=400., XT=314.15926, IFREQ=5*
```

CHAPTER III

DYNAMIC PROGRAMMING

3.1 DISCRETE MULTISTAGE TWO-DECISION PROCESS

A problem with the property that, at each of a finite set of times t_1, t_2, ...t_n, a decision is to be chosen from a finite set of possible decisions, is called a discrete multistage decision process. If one of m possible decisions must be chosen at each time and the process consists of n such stages, there are $(m)^n$ possible different sequences of n decisions. Our aim is to find the optimal sequence of decisions among these $(m)^n$ possible cases.

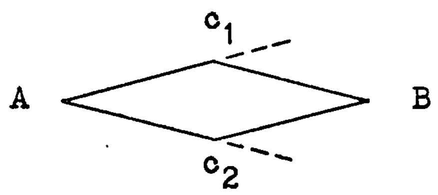


Figure 3.1-1

Two-decision, Two-stage Process.

Let us look at a two-decision two - stage minimum-cost problem. We define the term minimum "cost" as the minimum expenditure (in dallars), or minimum travelling time (in sec). At starting point A we must choose between the paths Ac_1B and Ac_2B, depending upon which one yields the lesser cost. If the cost of each section of the paths in Fig.3.1-1 are known, the decision to be made at A is a simple matter.

$$\text{Cost AB} = \min \begin{cases} \text{cost } Ac_1 + \text{cost } c_1B \\ \\ \text{cost } Ac_2 + \text{cost } c_2B \end{cases} \qquad (3.1\text{-}1)$$

In the multistage two-decision process shown in Fig.3.1-2, suppose the optimal decision is found to be Ac_1 in the first stage; we ask for another decision at c_1. One path should be chosen out of two possible paths c_1d_1B and c_1d_2B. The cost of c_1B is given by

$$\text{Cost } c_1B = \min \begin{bmatrix} \text{cost } c_1d_1 + \text{cost } d_1B \\ \text{cost } c_1d_2 + \text{cost } d_2B \end{bmatrix} \qquad (3.1\text{-}2)$$

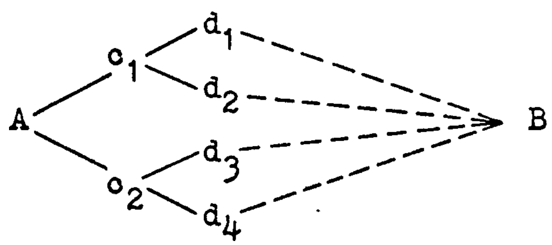


Figure 3.1-2

Two-decision, Multistage Process.

If cost c_1d_2B is found to be less than that of c_1d_1B, next decision must be made at d_2. The same procedure is repeated at each stage in all subsequent stages.

3.2 MARKOVIAN-TYPE PROCESSES

We introduce an assumption concerning the cost property of a network in order to make valid the statements of the previous section. In effect, we assume that the cost of any established path of a network does not change after it has been combined with the later stages of the network. A formal statement of this assumed property is due to Markov and given in [12]:

> "After any number of decisions, say k, we wish the effect of the remaining n-k stages of the decision process upon the total return to depend only upon the state of the system at the end of the k-th decision and the subsequent decisions."

3.3 MULTISTAGE MULTI-DECISION PROCESSES

In a multistage multi-decision process, if one of m possible paths must be chosen at each decision time, the problem is still intrinsically the same as for a two-decision process (Fig.3.3-1). That is,

$$\text{cost } AB = \min\,(\text{cost } Ac_i + \text{cost } c_iB) \qquad (3.3\text{-}1)$$

For a more general illustration, let us construct a grid of points in x-y plane as shown in Fig. 3.3-2. As shown in Fig.3.3-3 the optimum path c_id_o is found by considering costs determined as follows:

$$\text{cost } c_id_o = \min \begin{cases} c_id_j + d_jd_o \\ c_ic_j + c_jd_k + d_kd_o \\ c_ic_j + c_jd_o \end{cases} \qquad (3.3\text{-}2)$$

$$(j,\ k = 0,\ 1,\ 2,\ \ldots\ i)$$

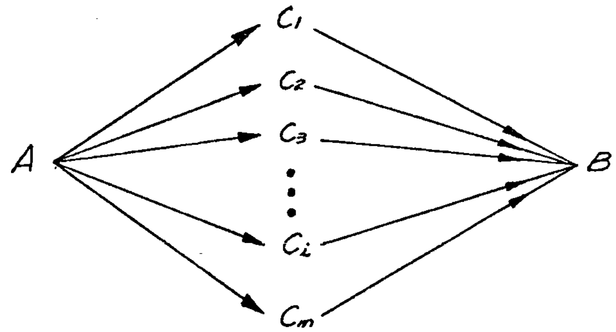


Figure 3.3-1

Multi-decision Process

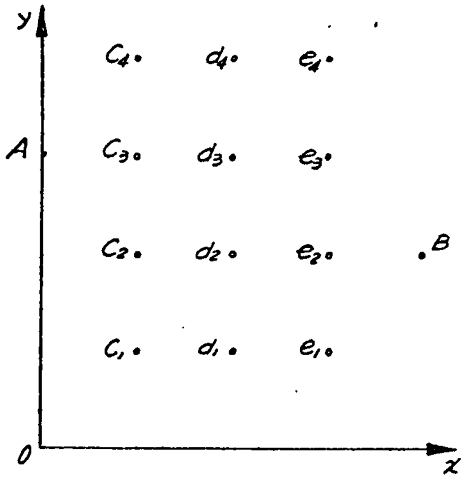


Figure 3.3-2

Grid points in x-y Plane

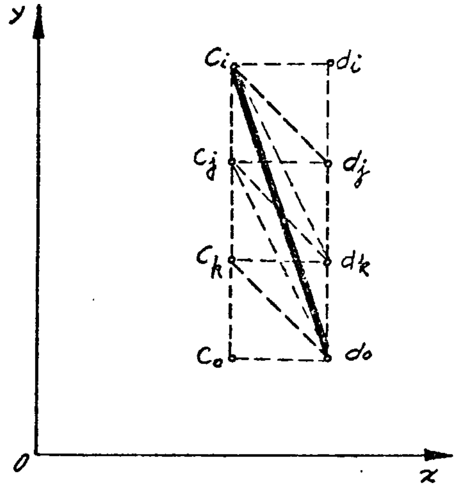


Figure 3.3-3

Optimum Path c_i-d_o

In the brachistochrone problem, by taking grid sizes sufficiently small, we may approximate the optimum path from c_i to d_j on the nearest neighboring stage as the diagonal $\overline{c_i d_j}$.

3.4 THE PRINCIPLE OF OPTIMALITY

Recall Eq.(3.3-2) and Fig.3.3-1, if there exists at least one stage between c_i and B, then the costs of c_iB for $i=0,1,2,\ldots m$, should be completely known before making decision at A. For a multistage process, we start the decision making at the stage nearest to B. After the costs f_iB at the stage $k=n-1$ have been found (as shown in Fig.3.4-1), the cost from any grid e_i at stage $k=n-2$ is expressed by

$$\text{cost } e_iB = \min\,(\text{cost } e_if_j + \text{cost } f_jB) \qquad (3.4\text{-}1)$$
$$j = 0,1,2, \ldots m.$$

Similar but more lengthy procedures are repeated for the points d_i at stage $k=n-3$, with the cost d_iB expressed as

$$\text{cost } d_iB = \min\,(\text{cost } d_ie_j+\text{cost } e_jf_q+\text{cost } f_qB) \quad (3.4\text{-}2)$$
$$j,q = 0,1,2, \ldots m.$$

Consider the right hand side of Eq.(3.4-2). It contains m^2 number of cases. The (cost e_jf_q+cost f_qB) has been computed at the previous stage $k=n-2$; therefore, Eq.(3.4-2) may be simplified as

$$\begin{aligned}\text{cost } d_iB &= \min\left[\text{cost } d_ie_j+(\text{cost } e_jf_q+\text{cost } f_qB)\right]\\ &= \min\,(\text{cost } d_ie_j+ \text{cost } e_jB)\\ j &= 0,1,2, \ldots m. \qquad (3.4\text{-}3)\end{aligned}$$

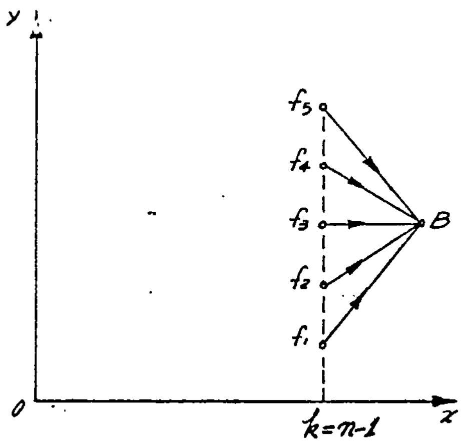


Figure 3.4-1

Stage k = n - 1

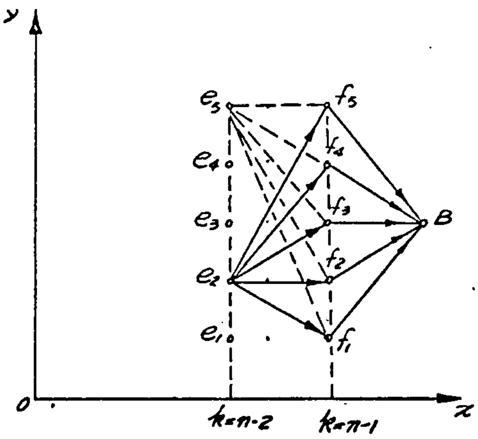


Figure 3.4-2

Stage k = n - 2

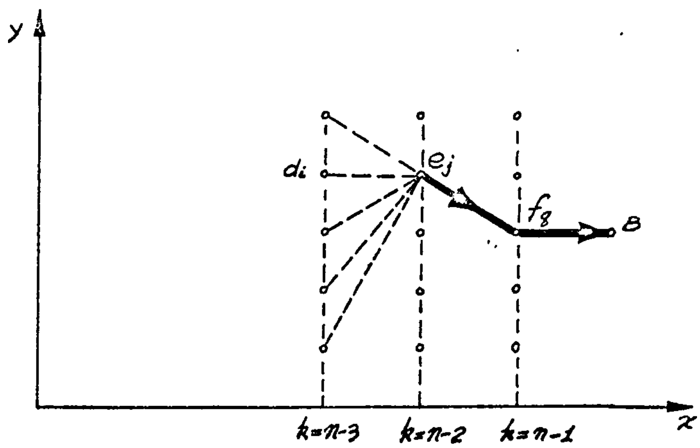


Figure 3.4-3

Geometry of the Principle of Optimality

which reduces the number of cases to be studied from m^2 to m for one grid point d_i. This simplification is legitimate only when cost e_jB is not changed after being combined with the other section d_ie_j; however, our original assumption that the process is to be Markovian satisfies this condition.

For particular point e_j, Eq.(3.4-3) may be written in detail as

$$\underset{(e_j \text{ fixed})}{\text{cost } d_ie_jB = \min} \begin{cases} d_1e_j + e_jB \\ d_2e_j + e_jB \\ \cdots\cdots\cdots \\ d_ie_j + e_jB \\ \cdots\cdots\cdots \\ d_me_j + e_jB \end{cases} \qquad (3.4\text{-}4)$$

Equation (3.4-4) with geometry of Fig.3.4-3 shows that no matter from which point d_i one comes to e_j, the optimum path e_jB found in the previous stage constitutes a part of the optimal path from d_i to B. This basic principle of dynamic programming has been called by Bellman "the principle of optimality" [4, 12, 14], that is,

> "An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision."

On the other hand, for a fixed point d_i, Eq.(3.4-3) may be written as

$$\text{cost } d_i e_j B = \min \begin{cases} d_i e_1 + e_1 B \\ d_i e_2 + e_2 B \\ \cdots\cdots\cdots\cdots \\ d_i e_j + e_j B \\ \cdots\cdots\cdots\cdots \\ d_i e_m + e_m B \end{cases} \qquad (3.4\text{-}5)$$

$(d_i \text{ fixed})$

It is important to note that Eq.(3.4-5) does not mean

$$\text{cost } d_i B = \min(\text{cost } d_i e_j) + \min(\text{cost } e_j B) \qquad (3.4\text{-}6)$$

For arbitrary given cost on each chord shown in Fig.3.4-5, if we apply Eq.(3.4-5) we obtain

$$\text{cost } d_i B = \min \begin{Bmatrix} d_i e_1 B = 1+8 = 9 \\ d_i e_2 B = 2+5 = 7 \\ d_i e_3 B = 4+4 = 8 \end{Bmatrix} = 7 \qquad (3.4\text{-}7)$$

However, applying Eq.(3.4-6) in two ways we have

$$\begin{aligned} &\min d_i e_j + \min e_j B = 1+8 = 9, \quad (j=1,2,3) \\ &\min B e_j + \min e_j d_i = 4+4 = 8, \quad (j=1,2,3) \end{aligned} \qquad (3.4\text{-}8)$$

For a three-stage process shown in Fig.3.4-6

$$\text{cost } d_i B = \min \begin{Bmatrix} 1+6+10 = 17 \\ 1+8+5 = 14 \\ 2+3+10 = 15 \\ 2+4+5 = 11 \end{Bmatrix} = 11 \qquad (3.4\text{-}9)$$

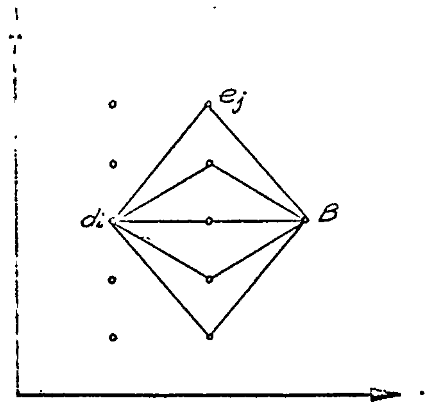


Figure 3.4-4

Possible Paths from d_1 to B

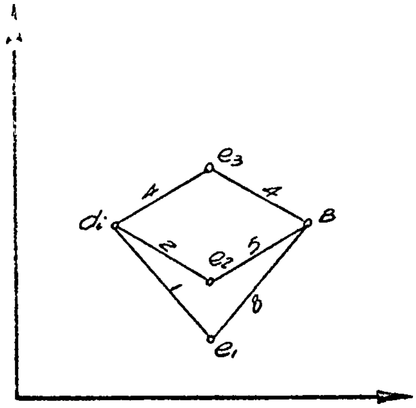


Figure 3.4-5

Figure of an Example

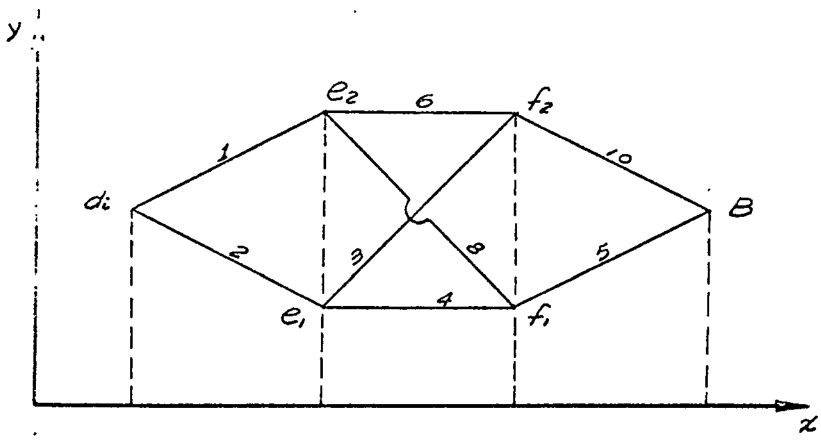


Figure 3.4-6

Figure of an Example

while for j, k = 1, 2

$$\min d_1 e_j + \min e_j f_k + \min f_k B = 1+6+10 = 17 \qquad (3.4\text{-}10)$$

Obviously a multistage decision process problem cannot be solved by making optimal single decisions sequentially. It is not the cost value of each section but the composite effect that is calculated.

3.5 INVARIANT IMBEDDING AND DYNAMIC PROGRAMMING

In computing the optimum costs from f_i to B or from e_j to B, in effect, we imbedded a particular problem in a family of similar problems. Each member of the family has the same terminal point B, with different initial values. This leads to a recursive solution working backward from the terminal point and eventually including point A. It is called a backward solution.

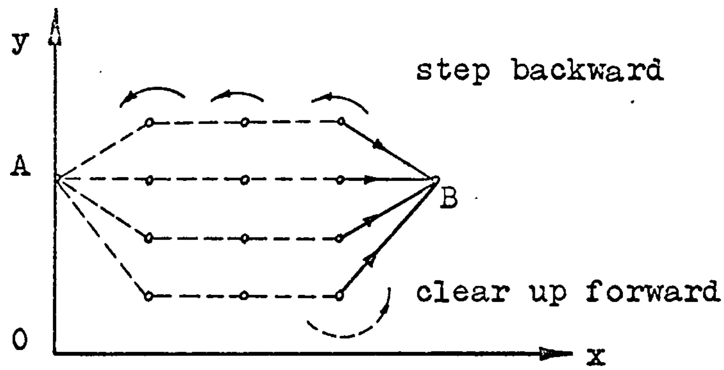


Figure 3.5-1

Backward Scheme

By Eq.(3.4-1) above we cannot actually make a proper decision at stage k=n-2 unless the costs f_iB, for i=0, 1, 2, ... m, are known. On the other hand, we do not know which member of the family of optimum paths f_iB will finally constitute the optimum path AB we are seeking. This is to say, the results of the process stream at all intermediate stages are unknown before the problem is completely solved. The cost equations cannot become immediately useful in solving multistage problems. This difficulty is overcome by employing invariant imbedding techniques in two steps [22].

In the first step, we start from the last stage proceeding backward to the initial stage, construct a table for each stage, relating the optimal decisions to the corresponding values of the objective function for each value of the state variable entering any particular stage. The stage for which the table is to be constructed is considered as the initial stage. At the k-th stage in the n-stage decision process, all downstream stages are considered as an (n-k)-stage process for which the optimum decision and the optimum objective function are already obtained and listed in the table constructed in the previous stage.

The second step is to determine the optimum policy-optimal sequence of decisions, for the entire process by means of table-entry techniques utilizing all the tables constructed. For example, if at the initial stage we found that Ac_5B is optimum among ac_iB, the optimum decision at A is Ac_5, from the table made at the stage k=1 we pick up the optimum decision at state c_5, say c_5d_3. The decision at state d_3 is found from the list made at k=2. In this way, we finally get a series of decisions as $A-c_5-d_3-e_2 \ldots f_4-B$.

3.6 REVERSE PRINCIPLE OF OPTIMALITY

If we imbed the specific problem in a family of problems with fixed initial point A and various terminal points which include the objective point B, the solution is called a forward solution.

As shown in Fig.3.6-1,

$$\text{cost } Ac_j = \text{cost } Ac_j \quad \text{(diagonal path)} \tag{3.6-1}$$

$$\text{cost } Ad_i = \min (Ac_j + c_jd_j) \tag{3.6-2}$$

In Fig.3.6-3. if the optimum path from A to d_3 is found to be Ac_3d_3, then instead of investigating

$$\text{cost } Ac_j + \text{cost } c_jd_3 + \text{cost } d_3e_i \tag{3.6-3}$$

for $j = 1, 2, 3, \ldots m$.

cost Ad_3e_i is given by

$$\begin{aligned} \text{cost } Ad_3e_i &= \min (\text{cost } Ac_j + \text{cost } c_jd_3 + \text{cost } d_3e_i) \\ &= \min (\text{cost } Ad_3 + \text{cost } d_3e_i) \end{aligned} \tag{3.6-4}$$

If we continue to proceed in this way, we have used the principle of optimality in reverse order. Dreyfus calls this "reversed principle of optimality" [21] stating:

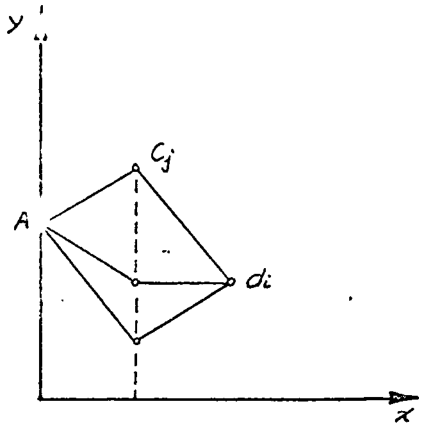


Figure 3.6-1

Possible Paths from A to d_i

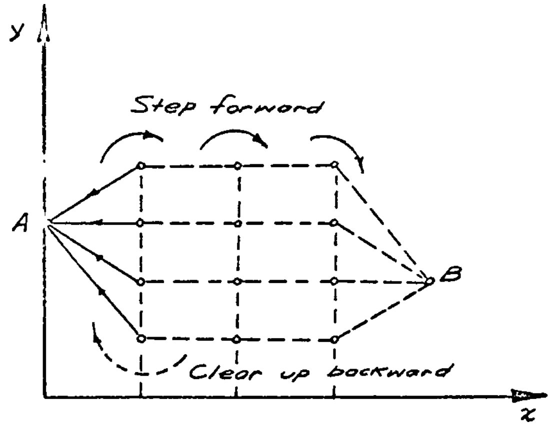


Figure 3.6-2

Forward Scheme

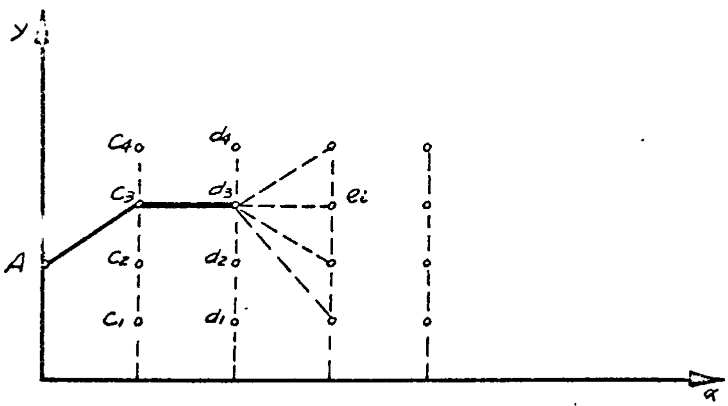


Figure 3.6-3

Geometry of the Reverse Principle

Of Optimality

"An optimal sequence of decisions in a multistage decision process problem has the property that whatever the final decision and state preceding the terminal one, the prior decisions must constitute an optimal sequence of decisions leading from the initial state to that state preceding the terminal one."

3.7 EULER EQUATION DERIVED FROM DYNAMIC PROGRAMMING

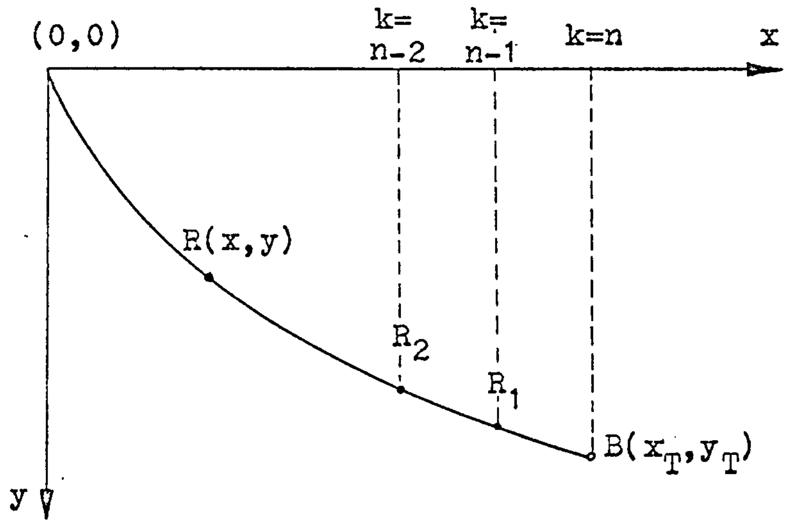


Figure 3.7-1

Figure for Equation (3.7-1)

Let $f(x,y)$ = the minimum time required to travel from $R(x,y)$ on the optimal path to the final point $B(x_T,y_T)$. (3.7-1)

Divide (x_T-0) into n equal segments with grid size

$$x = (x_T-0)/n \qquad (3.7\text{-}2)$$

Suppose r(x,y) is at the last stage with k=n-1, then

$$f_{n-1}(x,y) = \min \left[\sqrt{\frac{1+y'^2}{2gy}} \cdot \Delta x \right] \qquad (3.7\text{-}2)$$

Consider the left-neighboring stage with k=n-2

$$f_{n-2}(x,y) = \text{minimum time for travelling from } R_2 \text{ to } B$$

$$= \min_{y'} \left[\sqrt{\frac{1+y'^2}{2gy}} \cdot \Delta x + f_{n-1}(x,y) \right] \qquad (3.7\text{-}2)$$

Generally

$$f_k(x,y) = \min_{y'} \left[\sqrt{\frac{1+y'^2}{2gy}} \cdot \Delta x + f_{k-1}(x,y) \right] \qquad (3.7\text{-}5)$$

Since

$$x_{k+1} = x_k + \Delta x \qquad (3.7\text{-}6)$$

and

$$y_{k+1} = y_k + \Delta y, \qquad (3.7\text{-}7)$$

Eq.(3.7-5) may be written as

$$f(x,y) = \min_{y'} \left[\sqrt{\frac{1+y'^2}{2gy}} \cdot \Delta x + f(x+\Delta x, y+\Delta y) \right] \qquad (3.7\text{-}8)$$

This recurrence relation is equivalent to those developed in Section 3.4, and is the key to the solution.

Let

$$F = \sqrt{\frac{1+y'^2}{2gy}} \qquad (3.7\text{-}9)$$

and expand Eq.(3.7-9) in Taylor's series

$$f(x,y) = \min_{y'} \left[F\cdot\Delta x + f(x,y) + f_x\cdot\Delta x + f_y\cdot\Delta y + 0\ (\Delta x)^2 \right]$$
$$= \min_{y'} \left[F\cdot\Delta x + f(x,y) + f_x\cdot\Delta x + f_y(y'\cdot\Delta x) + 0\ (\Delta x)^2 \right]$$
$$= f(x,y) + \min_{y'} \left[F\cdot\Delta x + f_x\cdot\Delta x + f_y\cdot y'\cdot\Delta x + 0\ (\Delta x)^2 \right] \tag{3.7-10}$$

Here the term f(x,y) in the right-hand side is taken from the bracket because it is defined as the minimum time of path obtained from the optimally chosen y'. In addition, minimum over y' is equivalent to minimum over y since the grid sizes are chosen constant for all stages throughout the process.

Neglecting high-order terms, Eq.(3.7-10) becomes

$$0 = \min_{y} (F + f_x + y'\ f_y) \tag{3.7-11}$$

This non-linear partial differential equation governing the optimum path is equivalent to two equations. For optimally chosen y',

$$0 = F + f_x + y'\ f_y \tag{3.7-12}$$

To extremize the right-hand side of Eq.(3.7-11), its differentiation with respect to y' must vanish, that is,

$$0 = F_{y'} + f_y \tag{3.7-13}$$

If we differentiate Eq.(3.7-12) with respect to y, we have

$$F_y + f_{xy} + y'\ f_{yy} = 0 \tag{3.7-14}$$

Similarly, if we differentiate Eq.(3.7-12) with respect to x, we have

$$\frac{d}{dx} F_{y'} + f_{xy} + y' f_{yy} = 0 \qquad (3.7\text{-}15)$$

By subtracting Eq.(3.7-14) from Eq.(3.7-15), we finally obtain Euler's equation

$$\frac{d}{dx} F_{y'} - F_y = 0 \qquad (3.7\text{-}16)$$

For our particular case, F is defined in Eq.(3.7-9), and we substitute

$$F_y' = \frac{y'}{\sqrt{2gy \quad (1+y'^2)}} \qquad (3.7\text{-}17)$$

$$F_y = -\frac{1}{2} \frac{\sqrt{1+y'^2}}{\sqrt{2g} \quad (y)^{1.5}} \qquad (3.7\text{-}18)$$

in Eq.(3.7-16). With some manipulation, this yields

$$1 + y'^2 = c/y \qquad (3.7\text{-}19)$$

which is identical to the results derived by the classical method[5].

3.8 BRACHISTOCHRONE PROBLEM SOLVED BY DYNAMIC PROGRAMMING

A family of brachistochrone problems starting at x = 0, y = 0 and terminating at different point on x=100ℜ are solved by using the forward method of dynamic programming. Taking 100

[5] Appendix Eq.(A-5)

grid points in the y direction, we first construct a matrix whose elements represent the costs of diagonal paths of a channel with two nearest neighboring columns as the edges of the channel. For a 20-stage process with 10 sets of solutions printed out, the execution takes 35.1 sec using IBM 7094 computer. In this 20-stage 100-decision process, we actually solved 20 x 100 = 2000 similar problems. In Table 3-1, the minimum travelling times obtained by this method are compared with those obtained by classical solution methods.

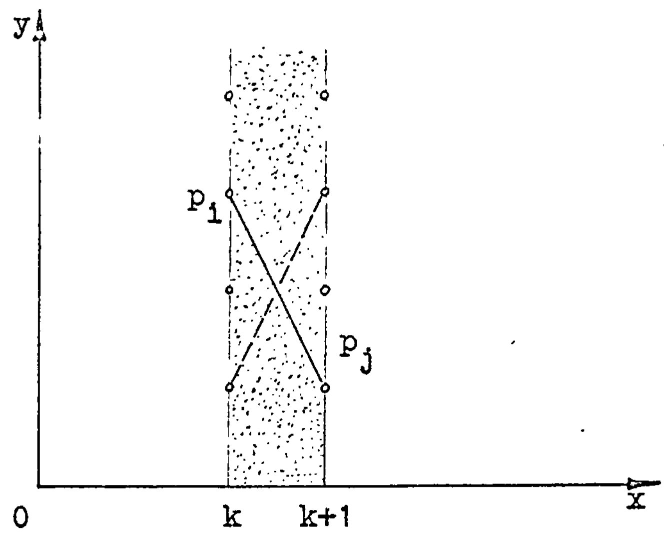


Figure 3.8-1

Elements of Cost Matrix

As can be seen in Table 3-2, the accuracy of the solution depends greatly upon the number of grid points chosen. A large number of grid points not only increases the computing time but also introduces memory problems. For instance, a 40-stage, 150-decision process requires 22500 memory locations for the cost matrix and 6000 for the policy matrix. Memory overlapping was experienced when 28800 memory locations were assigned for arrays in a program run by IBM 7094 computer which has 32768 such locations available. This implies a sufficient number of memory locations were not reserved for execution.

In Fig.3.8-2 the optimal paths for a 20-stage, 80-decision process are shown.

Let us suppose the problem is to find the path of least-travelling time from the origin to the terminal line $x = x_T$, where y_T is unspecified, as mentioned in Section 2.3, this free-end condition only changes one boundary condition from position constraint to slope constraint. If forward method is used, we choose the curve which gives the minimum-time of travelling among all 100 cases with different terminal points on the same terminal line. If backward scheme is employed, the optimal slopes are zero at the stage nearest to the terminal line. This approach is demonstrated in Program 3-2.

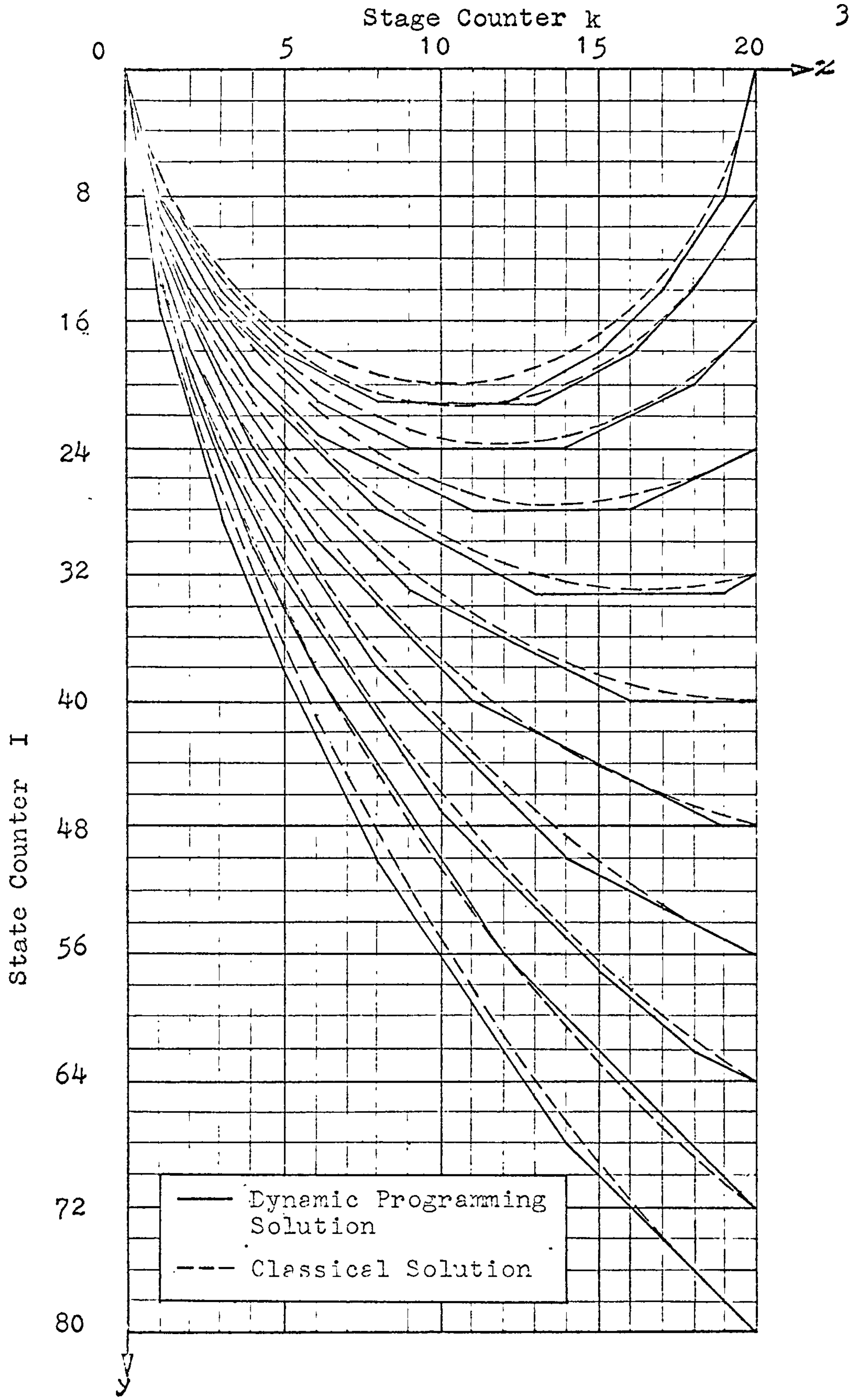


Figure 3.8-2

Optimal curves Obtained by Dynamic Programming
($x=0\sim 100\pi$, $y=0\sim 400$ feet)

Table 3-1

Minimum Travelling Time Obtained by Dynamic Programming

$x=0\sim100\pi$, $y=0\sim400$ feet
Taking 20 grid points in x-direction, 100 in y-direction

		D. P.	Classical	
I	y(I) (feet)	T(I) (sec)	Y(I) (sec)	Error (%)
0	0	7.90703	7.82955	0.98
10	40	6.40467	6.36233	0.67
20	80	5.95519	5.91442	0.69
30	120	5.71579	5.67980	0.63
40	160	5.60058	5.56763	0.59
50	200	5.56509	5.53633	0.52
60	240	5.58637	5.56104	0.46
70	280	5.64761	5.62525	0.40
80	320	5.73690	5.71746	0.34
90	360	5.84633	5.82950	0.29
100	400	5.97084	5.95554	0.27

Table 3-2

Grid Number and Accuracy in Dynamic Programming

From (0,0) to $(100\pi,400)$ feet
Classical Solution T=5.95554 sec

Grid Number		Computing Time	Minimum Time of Trav.	Error
x	y	(sec)	(sec)	(%)
20	20	8.4	6.06087	1.76
20	40	11.8	5.98005	0.41
20	60	17.5	5.97555	0.34
20	80	25.4	5.97141	0.27
20	100	35.1	5.97084	0.27
40	20	9.5	6.29473	5.70
40	40	15.1	6.05224	1.61
40	60	26.1	5.97666	0.35
40	80	40.3	5.97303	0.29
40	100	58.5	5.97186	0.27

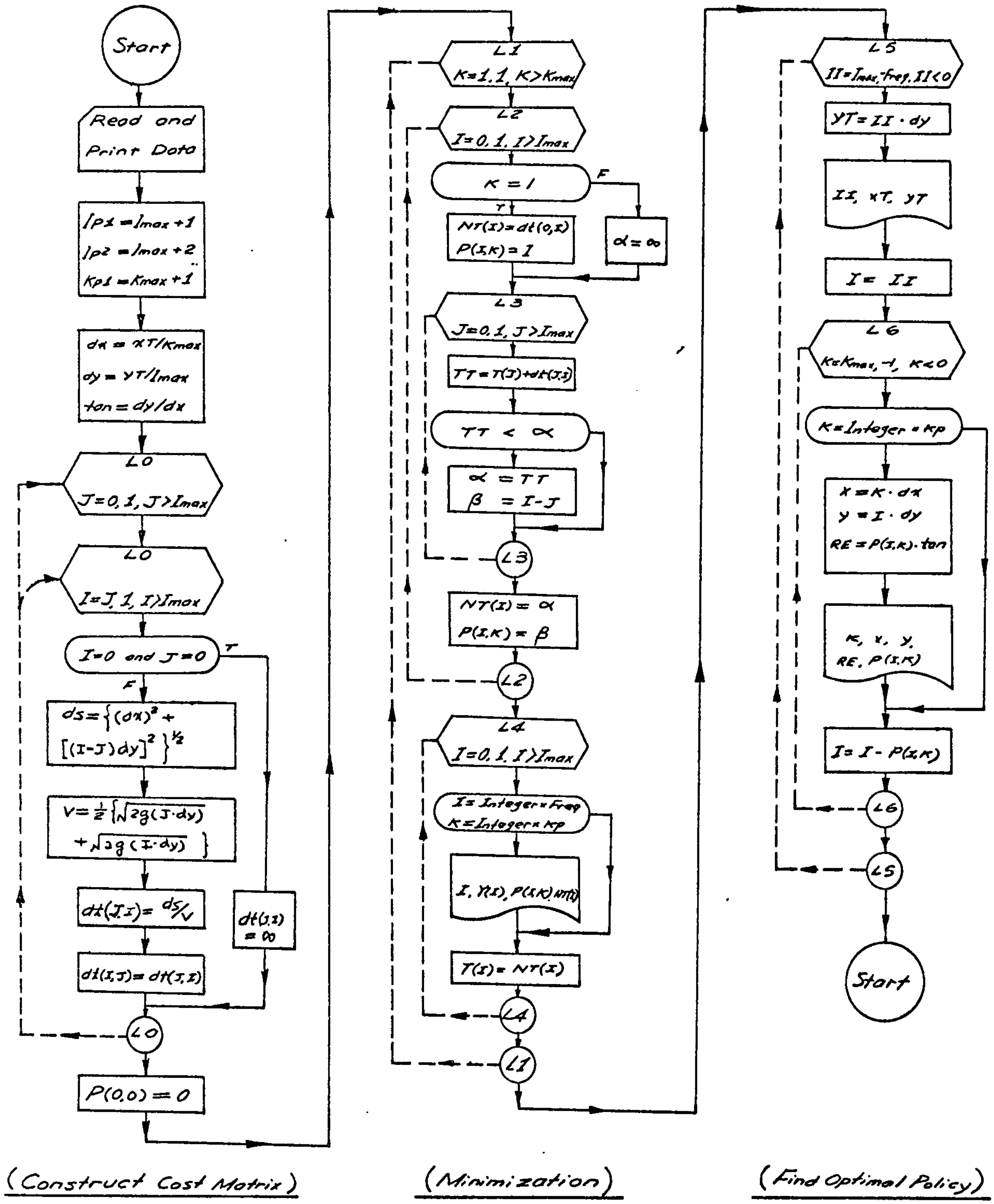


Figure 3.8-3

Flow Chart: Forward Method of Dynamic Programming

```
          R               P R O G R A M   3 - 1

          R   BRACHISTOCHRONE PROBLEM WITH TWO-POINT CONSTRAINT SOLVED BY
          R               FORWARD METHOD OF DYNAMIC PROGRAMMING

$  COMPILE MAD, EXECUTE, PRINT OBJECT, DUMP

           DIMENSION Y(101), T(101), NT(101), P( 6200,DIM),
          1 DT(10300, TIME)
           VECTOR VALUES DIM = 2,0,0
           VECTOR VALUES TIME = 2,0,0
           EQUIVALENCE (DIM(1),KP1), (DIM(2),KMAX), (TIME(1),IP2),
          1(TIME(2), IP1)
           INTEGER I, J, K, IMAX, KMAX, P, BETA, IP1, IP2, KP1, RI, II,
          1 FREQ, KP

START      READ AND PRINT DATA XT, YT, IMAX, KMAX, FREQ, KP
           IP1 = IMAX + 1
           IP2 = IMAX + 2
           KP1 = KMAX + 1
           DX = XT/KMAX
           DY = YT/IMAX
           THROUGH L0, FOR J = 0,1, J.G.IMAX
           THROUGH L0, FOR I = J, 1, I.G. IMAX
           WHENEVER I .E. 0 .AND. J .E. 0
           DT(J,I) = 1E5
           OTHERWISE
           DS = SQRT.(((I-J)*DY).P.2 + DX*DX)
           V = 4.013 * (SQRT.(J*DY) + SQRT.(I*DY))
           DT(J,I) = DS/V
           DT(I,J) = DT(J,I)
           END OF CONDITIONAL
L0
           P(0,0) = 0
           THROUGH L1, FOR K = 1, 1, K .G. KMAX
           THROUGH L2, FOR I = 0,1, I.G.IMAX
           WHENEVER K .E. 1
           NT(I) = DT(0,I)
           P(I,K) = I
           OTHERWISE
           ALPHA = 1E37
           THROUGH L3, FOR J = 0, 1, J .G. IMAX
           TT = T(J) + DT(J,I)
           WHENEVER TT .L. ALPHA
           ALPHA = TT
           BETA = I-J
           END OF CONDITIONAL
L3
           NT(I) = ALPHA
           P(I,K) = BETA
           END OF CONDITIONAL
L2
```

```
          WHENEVER (K/KP)*KP .E. K
          PRINT COMMENT $0$
          PRINT RESULTS K
          PRINT COMMENT     $0          I                          Y(I)
         1    P(I,K)                       T(I)   $
          END OF CONDITIONAL

          THROUGH L4, FOR I = 0,1, I.G.IMAX
          WHENEVER (I/FREQ)*FREQ .E. I .AND. (K/KP)*KP .E. K
          Y(I) = I*DY
          PRINT FORMAT BRACHI, I, Y(I), P(I,K),NT(I)
          END OF CONDITIONAL
          T(I) = NT(I)
L4
L1
          TAN = DY/DX
          PRINT COMMENT $1 ..... THE BEST POLICY .....   *$

          THROUGH L5, FOR II = IMAX,-FREQ, II .L. 0
          YT = II*DY
          PRINT COMMENT $0$
          PRINT COMMENT  $0 THE TERMINAL COMDITION IS $
          PRINT RESULTS II, XT, YT
          PRINT COMMENT   $0    K                  X                   Y
         1                  SLOPE            P(I,K) $
          I = II
          THROUGH L6, FOR K = KMAX, -1, K.L. 0
          WHENEVER (K/KP)*KP .E. K
          RE = P(I,K)*TAN
          X = K*DX
          Y = I*DY
          PRINT FORMAT POLICY, K, X, Y, RE, P(I,K)
          END OF CONDITIONAL
          I = I-P(I,K)
L6
L5
          VECTOR VALUES BRACHI =  $ I10, E30.6,   I10, E30.6   *$
          VECTOR VALUES POLICY = $  1I10, 3E20.8, 1I10   *$
          TRANSFER TO START
          END OF PROGRAM
$ DATA
 XT = 314.15926, YT= 400., IMAX=100, KMAX= 20, FREQ =10, KP=2*
```

```
          R              P R O G R A M   3 - 2

          R   BRACHISTOCHRONE PROBLEM WITH FREE END CONDITIONS SOLVED BY
          R              BACKWARD METHOD OF DYNAMIC PROGRAMMING

$  COMPILE MAD, EXECUTE, PRINT OBJECT, DUMP

           DIMENSION Y(100), T(100), NT(100), P(2200,DIM),DT(10300,TIME)
           VECTOR VALUES DIM = 2, 0, 0
           VECTOR VALUES TIME = 2, 0, 0
           EQUIVALENCE (DIM(1),KP1), (DIM(2),KMAX), (TIME(1),IP2),
          1(TIME(2),IP1)
           INTEGER I, II, IP1, IP2, IMAX, IS, J,
          1K, KP1, KMAX, P, BETA, FREQ
START      READ AND PRINT DATA XT, YT, IMAX, KAMX, FREQ
           IP1 = IMAX + 1
           IP2 = IMAX + 2
           KP1 = KMAX + 1
           DX  = XT/KMAX
           DY  = YT/IMAX
           THROUGH L0, FOR J = 0, 1, J .G. IMAX
           THROUGH L0, FOR I = J, 1, I .G. IMAX
           WHENEVER I .E. 0 .AND. J .E. 0
           DT(J, I) = 1E5
           OTHERWISE
           DS = SQRT.(((I-J)*DY).P.2 + DX*DX)
           V  = 4.013*(SQRT.(J*DY) + SQRT.(I*DY))
           DT(J,I) = DS/V
           DT(I,J) = DT(J,I)
           END OF CONDITIONAL
L0
           THROUGH L1, FOR I = 0, 1, I .G. IMAX
           P(I,KMAX) = 0
           T(I) = 0.
           Y(I)= I*DY
L1
           THROUGH L2, FOR K = KMAX-1, -1, K .L. 0
           THROUGH L3, FOR I = 0, 1, I .G. IMAX
           ALPHA = 1E37
           T(0) = 1E5
           THROUGH L4, FOR J = 0, 1, J .G. IMAX
           TT = T(J) + DT(I,J)
           WHENEVER TT .L. ALPHA
           ALPHA = TT
           BETA  = J-I
           END OF CONDITIONAL
L4
           NT(I)  = ALPHA
           P(I,K) = BETA
L3
```

```
            PRINT COMMENT $0$
            PRINT RESULTS K
            PRINT COMMENT     $                I                                  Y(I)
           1    P(I,K)                                           NT(I)  $
            THROUGH L5, FOR I = 1,1, I .G. IMAX
            WHENEVER (I/FREQ)*FREQ .E. I
            PRINT FORMAT BRACHI, I, Y(I), P(I,K), NT(I)
            END OF CONDITIONAL
            T(I) = NT(I)
L5
L2
            PRINT COMMENT $ THE BEST POLICY$
            THROUGH L6, FOR II = FREQ, FREQ, II .G. 80
            YO = II*DY
            PRINT COMMENT $0$
            PRINT COMMENT $ THE STARTING POINT IS $
            PRINT RESULTS II, YO
            PRINT COMMENT $0      K                          NT(I)           Y
           1                  SLOPE     $
            I = II
            THROUGH L7, FOR K = 0, 1, K .G. KMAX
            PRINT FORMAT POLICY, K, NT(I), Y(I), P(I,K)
            I = I + P(I,K)
L7
L6
            VECTOR VALUES BRACHI = $ 1I10, 1E30.8, 1I10, 1E30.8  *$
            VECTOR VALUES POLICY = $ 1I10, 2E20.8, 1I10  *$
            TRANSFER TO START
            END OF PROGRAM
$ DATA
 XT = 314.15926, YT=400., IMAX=100, FREQ=10, KMAX=20*
```

CHAPTER IV

QUASILINEARIZATION

4.1 NEWTON-RAPHSON-KANTOROVICH METHOD

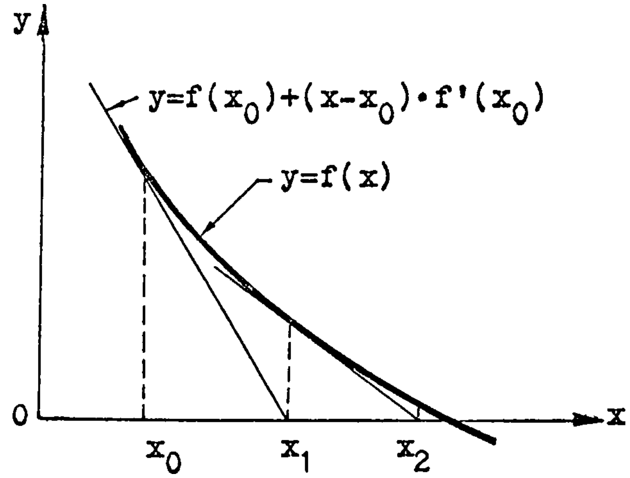


Figure 4.1-1

Newton-Raphson Method

Consider a monotone decreasing, convex function f(x), we approximate f(x) by a linear function of x determined by the value and slope of the function f(x) at $x = x_0$.

$$f(x) = f(x_0) + (x-x_0)\cdot f'(x_0) \tag{4.1-1}$$

Putting f(x) = 0, we obtain for the first approximation

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} \tag{4.1-2}$$

The process is repeated at x_1 leading to a new value x_2, and so on. The general recurrence relation is

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \qquad (4.1\text{-}3)$$

This sequence of approximation yields the root of

$$f(x) = 0 \qquad (4.1\text{-}4)$$

It has been shown that the convergence is monotonic and quadratic [19].

Replacing y by u, and y' by w, Eq.(1.2-3) may be rewritten as

$$u'' = - \frac{1 + w^2}{2\,u} = G(u,w) \qquad (4.1\text{-}5)$$

Let $u_0(x)$ be some initial approximation and consider the sequence $u_n(x)$. Applying Newton-Raphson technique we construct the recurrence relationships

$$u''_{n+1} = G(u,w) + (u_{n+1}-u_n)\frac{\partial G}{\partial u_n} + (w_{n+1}-w_n)\frac{\partial G}{\partial w_n} \qquad (4.1\text{-}6)$$

$$u_{n+1}(0) = y_0, \qquad u_{n+1}(x_T) = y_T \qquad (4.1\text{-}7)$$

Our aim is to produce a sequence of functions $u_1(x)$, $u_2(x)$, ... $u_n(x)$ which converges to the solution of the original function $u(x)$.

The concept characterized by Eq.(4.1-6) is an extension of the Newton-Raphson method to functional space which has been introduced by Kantorovich and is called Newton-Raphson-Kantorovich (NRK) technique [19]. It is essentially the first-order terms in power-series expansion of function $G(u,w)$ about the point u_n.

4.2 QUASILINEARIZATION

Consider a differential equation of the form

$$A(x)\,u'' + B(x)\,u' + C(x) = 0 \tag{4.2-1}$$

Because of its linearity, the principle of superposition holds. If p is the particular solution of the non-homogeneous equation

$$A(x)\,u'' + B(x)\,u' + C(x) = G(u,w) \tag{4.2-2}$$

It can be shown that the linear combination $p + c_1h_1 + c_2h_2$, where c_1 and c_2 are constants and h_1 and h_2 are solutions of the homogeneous equation, also satisfies Eq.(4.2-2), that is

$$u = p + c_1h_1 + c_2h_2 \tag{4.2-3}$$

For an m-order differential equation, the general solution may be written in the form

$$u = \sum_{k=1}^{m} c_kh_k + p \tag{4.2-4}$$

The m conditions imposed on the m unknown functions may be expressed as

$$\sum_{k=1}^{m} c_k h_k^{(\ell)} = u^{(\ell)} - p^{(\ell)} \tag{4.2-5}$$

$(\ell = 0,\ 1,\ 2,\ \ldots m-1.)$

If we substitute Eq.(4.2-5) in Eq.(4.1-6), we obtain

$$p'' + c_1 h_1'' + c_2 h_2''$$

$$= G + (p+c_1 h_1+c_2 h_2)\frac{\partial G}{\partial u} + (p'+c_1 h_1'+c_2 h_2')\frac{\partial G}{\partial w} \tag{4.2-6}$$

By equating the coefficients of Eq.(4.2-6), we obtain

$$p'' = G + (p-u_n)\frac{\partial G}{\partial u} + (p'-w)\frac{\partial G}{\partial w} \tag{4.2-7}$$

$$h_1'' = h_1\frac{\partial G}{\partial u} + h_1'\frac{\partial G}{\partial w} \tag{4.2-8}$$

$$h_2'' = h_2\frac{\partial G}{\partial u} + h_2'\frac{\partial G}{\partial w} \tag{4.2-9}$$

Let us choose the initial conditions

$$p(0) = 0, \qquad p'(0) = 0 \tag{4.2-10}$$

and the conditions on the homogeneous solutions of

$$h_1(0) = 1, \qquad h_1'(0) = 0 \tag{4.2-11}$$

$$h_2(0) = 0, \qquad h_2'(0) = 1 \tag{4.2-12}$$

which insures that the Wronskian

$$W(x) = \begin{vmatrix} h_1(x) & h_2(x) \\ \\ h_1'(x) & h_2'(x) \end{vmatrix} \neq 0 \qquad (4.2\text{-}13)$$

Thus we have a set of initial value problems whose solutions and their derivatives are readily produced numerically on the interval of $x = 0 \sim x_T$. The solution of Eq.(4.1-6) subject to boundary conditions Eq.(4.1-7) and their derivatives is expressed by

$$u(x) = p(x) + c_1 h_1(x) + c_2 h_2(x) \qquad (4.2\text{-}14)$$

$$w(x) = p(x) + c_1 h_1'(x) + c_2 h_2'(x) \qquad (4.2\text{-}15)$$

where c_1 and c_2 are constants to be determined from the linear algebraic equations obtained by substituting $x = 0$, and $x = x_T$ respectively into Eq.(4.1-7)

$$p(0) + c_1 h_1(0) + c_2 h_2(0) = y_0 \qquad (4.2\text{-}16)$$

$$p(x_T) + c_1 h_1(x_T) + c_2 h_2(x_T) = y_T \qquad (4.2\text{-}17)$$

In other words, we produce a particular solution and two independent homogeneous solutions on the interval $x = 0 \sim x_T$ and determine the constants c_1 and c_2 to satisfy the boundary conditions of Eq.(4.1-7). The process of Eqs.(4.2-7) to (4.2-17) is repeated to compute a new approximation of $u(x)$.

In the derivation of Eq.(4.2-7) to Eq.(4.2-8), equation

(4.2- 6), the NRK technique is applied in the abstract plane perpendicular to the x-axis at each point of x.

The computational scheme is shown in Fig.4.2-1 and the computer program follows.

The computational results of two brachistochrone curves using straight-line initial approximations are compared with analytical solutions in Table 4-1 and Table 4-2. In Table 4-1 an error can be seen near the singularity point $x = 0$, $y = 0$. Elsewhere, accuracy to five digits or more was obtained by 3-iteration of quasilinearization in the problem of Table 4-2.

Straight-line approximations failed to converge for the cycloidal paths of range greater than half of a complete cycle. Since the constant multipliers c_1 and c_2 are determined solely at the two end points, a complete cycle of the cycloidal path with singularities at both ends cannot be solved by this method.

Table 4-1

Convergency of $u_n(x)$ to $u(x)$ by Quasilinearization

Take 800 discrete points

k	$u_0(x)$	$u_1(x)$	$u_2(x)$	$u_3(x)$	$u(x)$
0	.000000E 01	.000000E 00	.000000E 00	.000000E 00	.000000E 00
40	.100000E 02	.147406E 02	.415132E 01	.454858E 02	.457040E 02
80	.200000E 02	.446946E 02	.656419E 02	.700734E 70	.702014E 02
40	.100000E 02	.247406E 02	.415132E 02	.454858E 02	.457040E 02
80	.200000E 02	.446946E 02	.656419E 02	.700734E 02	.702714E 02
120	.300000E 02	.622185E 02	.848638E 02	.893294E 02	.895121E 02
160	.400000E 02	.779257E 02	.101082E 03	.105424E 03	.105593E 03
200	.500000E 02	.921473E 02	.115130E 03	.119275E 03	.119430E 03
240	.600000E 02	.105095E 03	.127470E 03	.131380E 03	.131523E 03
280	.700000E 02	.116920E 03	.138396E 03	.142051E 03	.142181E 03
320	.800000E 02	.127734E 03	.148105E 03	.151495E 03	.151614E 03
360	.900000E 02	.137624E 03	.156744E 03	.159863E 03	.159971E 03
400	.100000E 03	.146668E 03	.164420E 03	.167264E 03	.167361E 03
440	.110000E 03	.154919E 03	.171212E 03	.173782E 03	.173870E 03
480	.120000E 03	.162429E 03	.177189E 03	.179483E 03	.179560E 03
520	.130000E 03	.169240E 03	.182400E 03	.184417E 03	.184484E 03
560	.140000E 03	.175390E 03	.186886E 03	.188625E 03	.188682E 03
600	.150000E 03	.180911E 03	.190680E 03	.192140E 03	.192187E 03
640	.160000E 03	.185830E 03	.193807E 03	.194986E 03	.195024E 03
680	.170000E 03	.190176E 03	.196289E 03	.197182E 03	.197211E 03
720	.180000E 03	.193974E 03	.198142E 03	.198744E 03	.198764E 03
760	.190000E 03	.197242E 03	.199376E 03	.199682E 03	.199691E 03
800	.200000E 03	.200000E 03	.200000E 03	.200000E 03	.200000E 03

Table 4-2

Convergency of $u_n(x)$ to $u(x)$ by Quasilinearization

Take 400 discrete points

k	$u_0(x)$	$u_1(x)$	$u_2(x)$	$u_3(x)$	$u(x)$
0	.200000E 03	.200000E 03	.200000E 03	.200000E 03	.200000E 03
20	.204709E 03	.210149E 03	.210341E 03	.210341E 03	.210341E 03
40	.209417E 03	.219541E 03	.219860E 03	.219860E 03	.219859E 03
60	.214126E 03	.228225E 03	.228626E 03	.228627E 03	.228626E 03
80	.218835E 03	.236242E 03	.236698E 03	.236698E 03	.236698E 03
100	.223544E 03	.243629E 03	.244122E 03	.244122E 03	.244121E 03
120	.228254E 03	.250417E 03	.250936E 03	.250936E 03	.250935E 03
140	.232961E 03	.256637E 03	.257173E 03	.257173E 03	.257172E 03
160	.237670E 03	.262313E 03	.262861E 03	.262861E 03	.262860E 03
180	.242379E 03	.267468E 03	.268023E 03	.268023E 03	.268023E 03
200	.247087E 03	.272121E 03	.272680E 03	.272680E 03	.272680E 03
220	.251796E 03	.276291E 03	.276849E 03	.276849E 03	.276849E 03
240	.265505E 03	.279993E 03	.280544E 03	.280544E 03	.280544E 03
260	.261214E 03	.283241E 03	.283777E 03	.283778E 03	.283777E 03
280	.265922E 03	.286049E 03	.286560E 03	.286560E 03	.286560E 03
300	.270631E 03	.288426E 03	.288901E 03	.288901E 03	.288901E 03
320	.275340E 03	.290383E 03	.290807E 03	.290807E 03	.290807E 03
340	.280049E 03	.291929E 03	.292284E 03	.292284E 03	.292284E 03
360	.284757E 03	.293072E 03	.293335E 03	.293335E 03	.293335E 03
380	.289464E 03	.293818E 03	.293965E 03	.293965E 03	.293965E 03
400	.294175E 03	.294175E 03	.294175E 03	.294175E 03	.294175E 03

Table 4-3

Minimum Travelling Time Obtained by Quasilinearization

($u_o = 0$)

Terminal Points	Trav. Time (Q.L)	Trav. Time (Classical)	Error
	iter=5		
$u(x_T)$	T(I)	T(I)	(%)
200	5.53719	5.53633	0.016
240	5.56174	5.56104	0.013
280	5.62580	5.62525	0.010
320	5.71787	5.71746	0.007
360	5.82979	5.82950	0.005
400	5.95571	5.95554	0.003

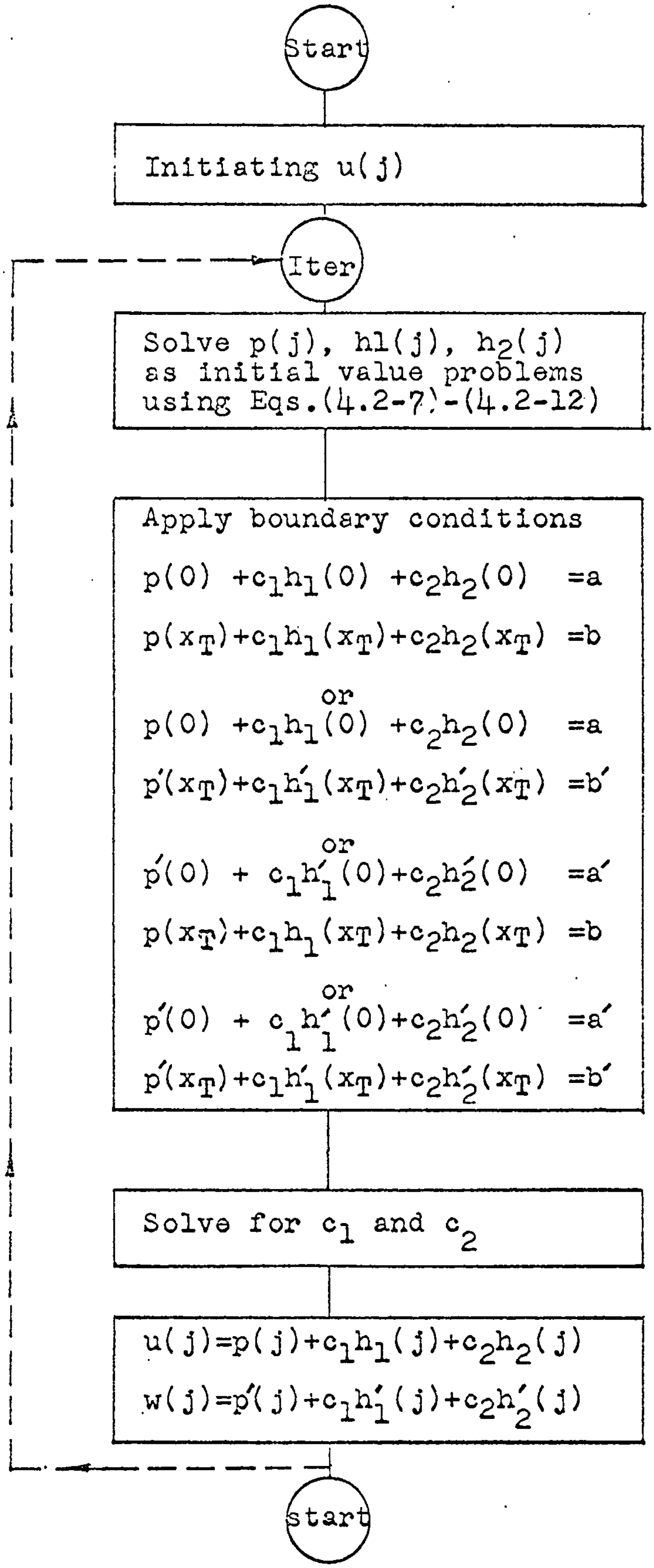

Start
Initiating u(j)
Iter
Solve p(j), hl(j), $h_2(j)$ as initial value problems using Eqs.(4.2-7)-(4.2-12)
Apply boundary conditions
$p(0) + c_1h_1(0) + c_2h_2(0) = a$
$p(x_T) + c_1h_1(x_T) + c_2h_2(x_T) = b$
or
$p(0) + c_1h_1(0) + c_2h_2(0) = a$
$p'(x_T) + c_1h_1'(x_T) + c_2h_2'(x_T) = b'$
or
$p'(0) + c_1h_1'(0) + c_2h_2'(0) = a'$
$p(x_T) + c_1h_1(x_T) + c_2h_2(x_T) = b$
or
$p'(0) + c_1h_1'(0) + c_2h_2'(0) = a'$
$p'(x_T) + c_1h_1'(x_T) + c_2h_2'(x_T) = b'$
Solve for c_1 and c_2
$u(j) = p(j) + c_1h_1(j) + c_2h_2(j)$
$w(j) = p'(j) + c_1h_1'(j) + c_2h_2'(j)$
start

Fig. 4.2-1

```
           R                 P R O G R A M   4 - 1

           R   BRACHISTOCHRONE PROBLEM SOLVED BY QUASILINEARIZATION

$  COMPILE MAD, EXECUTE, PRINT OBJECT, DUMP

            DIMENSION Y(10), F(10), Q(10),PA(800), H1(800), H2(800),
           1U(800), W(800),DPA(800), DH1(800), DH2(800), QT(800)
            INTEGER ITER, ITMAX, K, KP, KMAX, COUNT
START
            PRINT COMMENT $  DATA$
            READ AND PRINT DATA UO, UT, ITMAX, KMAX, XT, EPS,KP
            DX = XT/KMAX
            DY = (UT-UO)/KMAX
            TAN = (UT-UO)/XT
            THROUGH LO, FOR K = 1,1, K.G.KMAX
            X = K*DX
            U(K) = UO+DY*K
            W(K) = TAN
LO
            THROUGH L1, FOR ITER = 1,1, ITER .G. ITMAX
            PA(0) = 0.
            H1(0) = 1.
            H2(0) = 0.
            DPA(0) = 0.
            DH1(0) = 0.
            DH2(0) = 1.
            Y(1) = PA(0)
            Y(2) = DPA(0)
            Y(3) = H1(0)
            Y(4) = DH1(0)
            Y(5) = H2(0)
            Y(6) = DH2(0)
            X = 0.
            EXECUTE SETRKD.(6,Y(1),F(1),Q,X,DX)
            THROUGH LRK, FOR K = 1,1, K.G.KMAX
CALLRK      S = RKDEQ.(0)
            WHENEVER S .E. 1.
            F(1) = Y(2)
            WHENEVER F(1) .G. EPS
            F(1) = EPS
            END OF CONDITIONAL
            F(3) = Y(4)
            WHENEVER F(3) .G. EPS
            F(3) = EPS
            END OF CONDITIONAL
            F(5) = Y(6)
            WHENEVER F(5) .G. EPS
            F(5) = EPS
            END OF CONDITIONAL
```

```
         GU = (1.+W(K)*W(K))/(2*U(K)*U(K))
         WHENEVER GU .G. 1E6
         GU = 1E6
         END OF CONDITIONAL
         GW = -W(K)/U(K)
         WHENEVER .ABS.(GW) .G. 1E6
         GW = 1E6*(GW/(.ABS.(GW)))
         END OF CONDITIONAL
         F(2) = GU*(Y(1)-2.*U(K)) + GW*(Y(2)-W(K))
         WHENEVER .ABS.(F(2)) .G. EPS
         F(2) = EPS*(F(2)/(.ABS.(F(2))))
         END OF CONDITIONAL
         F(4) = GU*Y(3) + GW*Y(4)
         WHENEVER .ABS.(F(4)) .G. EPS
         F(4) = EPS*(F(4)/(.ABS.(F(4))))
         END OF CONDITIONAL
         F(6) = GU*Y(5) + GW*Y(6)
         WHENEVER .ABS.(F(4)) .G. EPS
         F(6) = EPS*(F(6)/(.ABS.(F(6))))
         END OF CONDITIONAL
         TRANSFER TO CALLRK

         OTHERWISE
         PA(K)  = Y(1)
         H1(K)  = Y(3)
         H2(K)  = Y(5)
         DPA(K) = Y(2)
         DH1(K) = Y(4)
         DH2(K) = Y(6)
         END OF CONDITIONAL
  LRK
         DIN = H1(0)*H2(KMAX) - H1(KMAX)*H2(0)
         AP = UO - PA(0)
         BP = UT - PA(KMAX)
         C1 = ( AP*H2(KMAX)-BP*H2(0))/DIN
         C2 = (-AP*H1(KMAX)+BP*H1(0))/DIN
         PRINT COMMENT  $0$
         PRINT COMMENT  $0$
         PRINT RESULTS ITER, C1, C2
         PRINT COMMENT    $   K        X                 PA
        1  H1                 H2               U                W
        2          QT  $
         THROUGH L2, FOR K = 0, 1, K .G. KMAX
         U(K) =  PA(K) + C1* H1(K) + C2* H2(K)
         W(K) = DPA(K) + C1*DH1(K) + C2*DH2(K)
         X = K*DX
         WHENEVER K .E. 0
         QT    = 0.
         OTHERWISE
         DS = SQRT.( (U(K)-U(K-1)).P.2+ DX*DX )
         V = 4.013*(SQRT.(U(K))+SQRT.(U(K-1)))
         QT = QT + DS/V
         END OF CONDITIONAL
```

```
          WHENEVER (K/KP)*KP .E. K
          PRINT FORMAT LINEAR, K, X,PA(K),H1(K),H2(K),U(K),W(K),QT
L2        END OF CONDITIONAL
          U(0) = 0.01
L1
          VECTOR VALUES LINEAR = $ 1I5, 1E12.4, 6E17.8  *$
          TRANSFER TO START
          END OF PROGRAM
$ DATA
U0=200., UT=294.17495,  ITMAX=3, KMAX=400, XT=314.15926, EPS=100,KP=20*
```

CHAPTER V

COMPARISONS AND COMBINATIONS

5.1 COMPARISONS

As we have seen in the previous chapters invariant imbedding, dynamic programming and quasilinearization, each has some powerful characteristics. Quasilinearization is the most accurate technique at the expense of relatively long computing time. Invariant imbedding requires very short computing time but gives only initial slopes and the results may be only approximately correct. Dynamic programming ranks between invariant imbedding and quasilinearization in accuracy and computing costs.

The size of problems which can be handled by dynamic programming is limited by the memory available in a computer. Invariant imbedding and quasilinearization have no memory problem, but the former should be combined with another method to produce state and cost functions; the latter converges only when a proper initial guess to the solution has been made.

Invariant imbedding and quasilinearization make use of the differential equation obtained from Euler's equation of the calculus of variations. Dynamic programming completely bypasses this derivation, although we showed that Euler's equation may be obtained from recurrence relations based on the principle of optimality. However, no differential equation which characterizes the optimum path was used in the minimization process. This powerful feature of dynamic programming is especially useful in the case where Euler's equation does not exist or is difficult to solve.

Another significant aspect is that invariant imbedding and quasilinearization are not suited to handle computations which include such features as the cusps of a cycloid where the slopes are infinity. Dynamic programming which treats continuous systems as discrete multi-stage processes is free of this trouble because the slopes are found between adjoining stages instead of at values of the state variable.

5.2 DYNAMIC PROGRAMMING WITH SEARCHING OVER A RESTRICTED REGION

As mentioned above, dynamic programming bypasses Euler's equation. In the brachistochrone problem, Euler's equation which characterizes the optimum path is known. We seek to find a way to utilize the differential equation obtained from Euler's equation to minimize the searching required in dynamic programming. We note that Eq.(1.2-3)

$$y'' = -\frac{1+y'^2}{2y} < 0, \quad \text{for } y > 0 \tag{5.2-1}$$

implies the slope is monotone decreasing. It can be seen that Eq.(5.2-1) with boundary conditions

$$y(0) = c_1, \qquad y(x_T) = c_2 \tag{5.2-2}$$

or

$$y(0) = c_1, \qquad y(x_T) = c_3 \tag{5.2-3}$$

describes cycloids which are single-valued functions. Let us consider a forward-scheme of dynamic programming. If the slope at state q_i in the k-th stage is greater than (or equal to) zero (as is shown in Fig.5.2-1 (A)), then point p_j (where the optimum curve crosses (k-1)-th stage) must lie below or at a level with q_i. It follows that in minimizing the time of travel from the initial point O to point q_i in the k-th stage, we have only to search over the region $y \leq q_i$, that is

$$\begin{aligned} \text{cost } Oq_i &= \min\,(Op_j+p_jq_i) \qquad j=1,2,3,\ldots m \\ &= \min\,(Op_j+p_jq_i) \qquad j=1,2,3,\ldots i \\ &= \min\,(Op_j+p_jq_i) \qquad j=i,i-1,\ \ldots 2,1. \end{aligned} \tag{5.2-4}$$

Furthermore, since the function is single-valued, the search may be terminated where the minimized cost function begins to increase. Then, Eq.(5.2-4) becomes

$$\text{cost } Oq_i = \min\,(Op_j+p_jq_i), \quad j=i,i-1,\ \ldots\ i\ell. \tag{5.2-5}$$

where $i\ell$ is the lower limit of the grid counter in the region to be searched. Similarily, for the slope at $q_i < 0$, the region to be searched is restricted to

$$j = i,\ i+1,\ i+2,\ \ldots ih \tag{5.2-6}$$

where ih is the upper limit.

A forward-solution using the partial-search technique described above is shown in Program 5-1. It reduced the

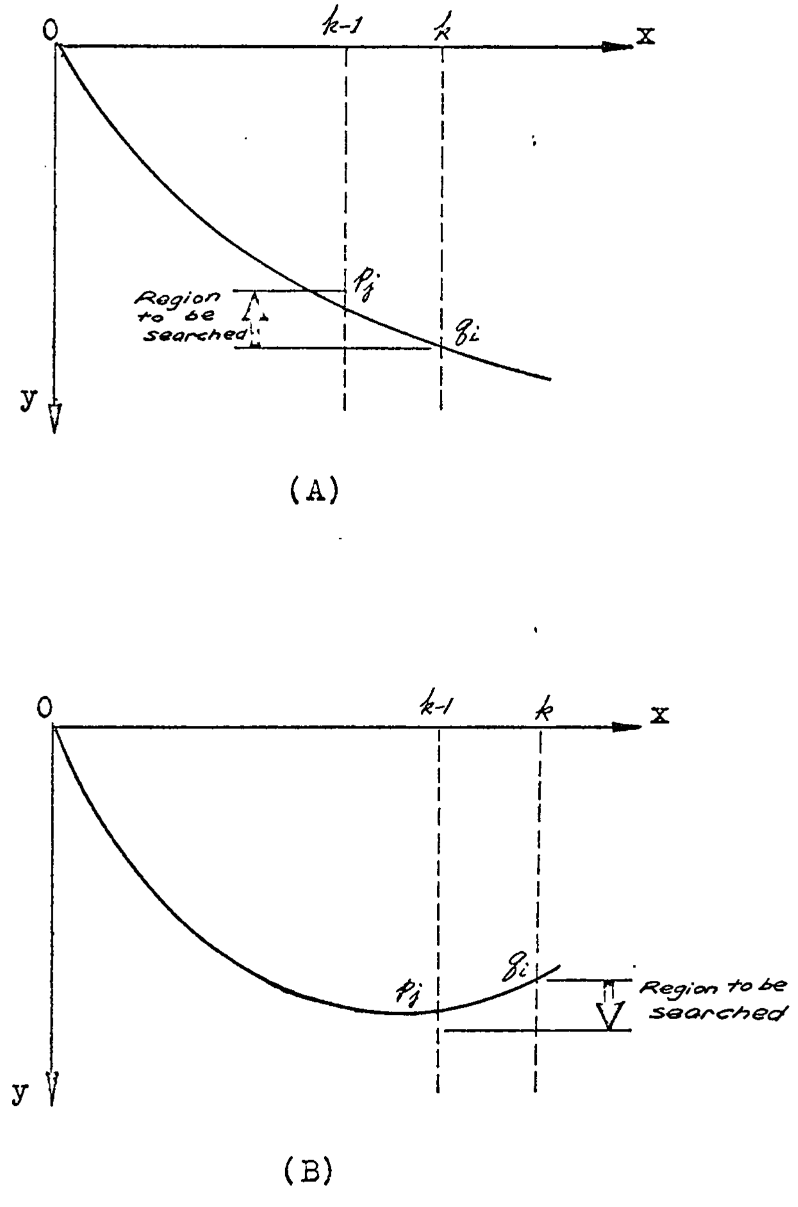


Figure 5.2-1

Slope Characteristics and
Searching Region

computing time from 35.1 sec to 15 sec in solving a 20-stage, 100-decision process with 10 sets of the solutions printed out.

5.3 COMBINATION OF INVARIANT IMBEDDING AND DYNAMIC PROGRAMMING

The technique of searching over a restricted region is effective especially where the absolute values of slopes are small. For the steep curves shown in Fig.5.3-1 (B) and (C), the usefulness of the feature is not as significant. Since dynamic programming is a marching process, the optimum slopes at p_ℓ (for $\ell=1,2,\ldots m$) are known a priori. We may take advantage of this information. Locate p_j from q_i using the slope at p_i, then search several grids in the neighborhood of this predicted position to obtain the optimum value p_j (Fig. 5.3-1 (D)). This can be accomplished successfully by joint use of invariant imbedding and dynamic programming[18], that is, predicting the slopes by invariant imbedding and then searching in the neighborhood by dynamic programming.

For a 20-stage, 100-decision process with 10 sets of solutions printed out, the computing time using this combination was 14.1 sec in comparison with 35.1 sec by dynamic programming only, and 15 sec using the partial-searching method. Searching was restricted to ± 2 grids in the vicinity of the predicted point.

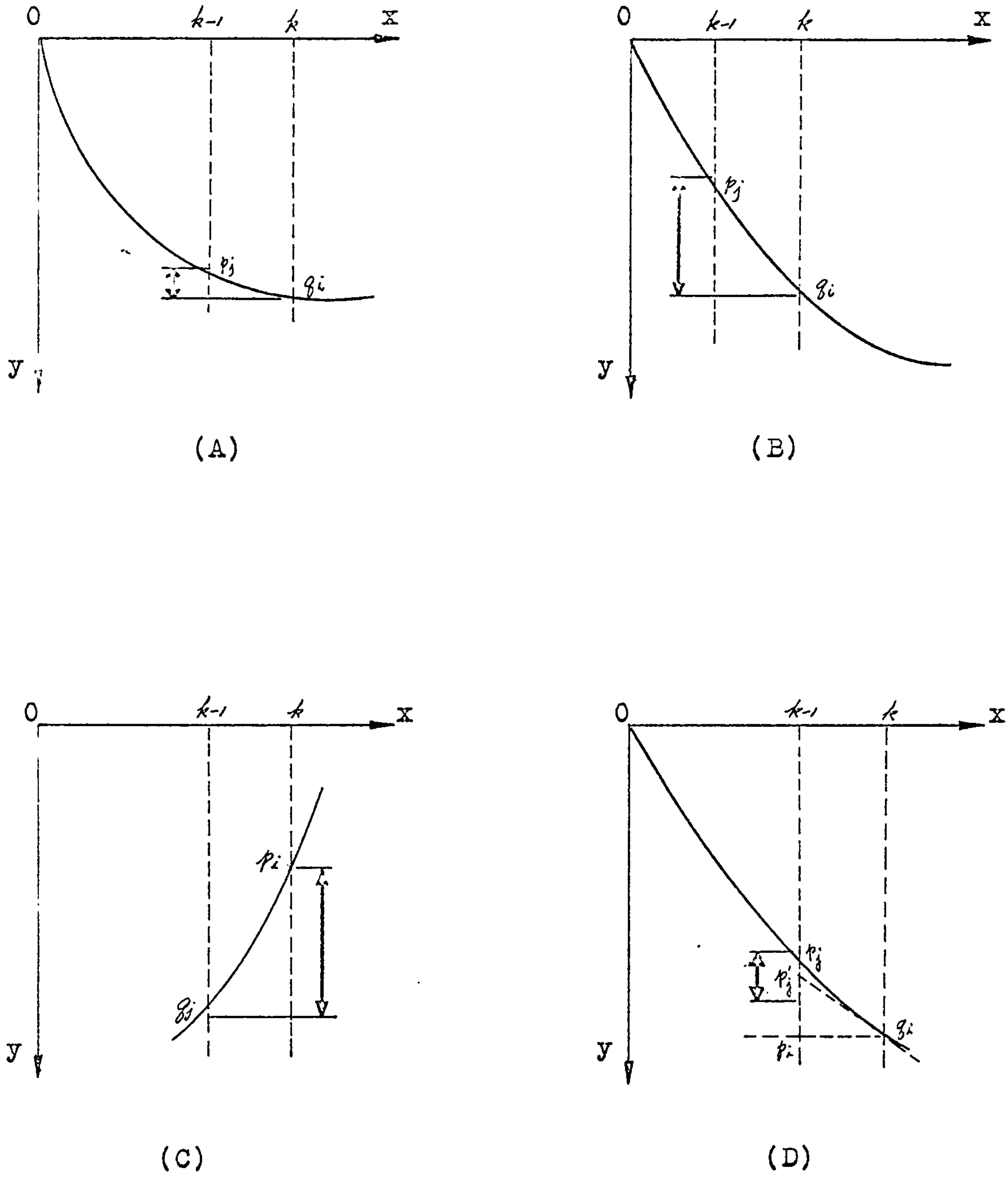


Figure 5.3-1

Regions to be Searched in Various Cases

5.4 DYNAMIC PROGRAMMING AND QUASILINEARIZATION

As mentioned previously, the coarse grids used in dynamic programming result in polygonal curves which may deviate significantly from what we know to be exact solution. Finer grids may improve the accuracy of the solution but a too-fine grid introduces a memory problem with the computer. On the other hand, quasilinearization yields very accurate results but is expensive and its convergence depends greatly upon near-correctness of the initial estimate of the solution. In general, a straight line is the simplest initial estimation; however, in the brachistochrone problem the solution converges only where the boundary point does not exceed a half-cycle of a cycloid.

Combined use of dynamic programming and quasilinearization compensates for the weaknesses of each. By this predictor-corrector method, we solve the problem approximately by first using the dynamic programming procedure with very coarse grids, and then take this solution as the initial guess to the solution whose accuracy is improved by a few applications of quasilinearization.

Program 5-3 uses dynamic programming in the main program and quasilinearization as a corrector in external function. In Table 5-1 the results of taking 20x40 grids in dynamic programming, and 2 applications of quasilinearizations for each solution are shown. Computing time was 50.5 sec which would be less than that for quasilinearization.

5.5 INVARIANT IMBEDDING AND QUASILINEARIZATION

Another predictor-corrector scheme combines invariant imbedding (used to predict the slopes) and quasilinearization (used to correct the solution resulting from the first and to produce the cost and state functions simultaneously) [18].

Consider a problem beginning at point (c,a). If the starting point at x=a is close to the terminal line $x=x_T$, the slopes at all initial points c_i may be estimated as zero and after a few iterations of quasilinearization it converges to the correct value r(c,a). The same procedure is repeated at $x=a-\Delta x$, $x=a-2\Delta x$, and so on. In effect, we solve 2000 problems for a 20-stage, 100-decision process. If the range of the independent variable is sufficiently small, we may use invariant imbedding in a straight-forward manner to produce the initial slopes at all initial values in x=0. Using these initial slopes and the other given initial conditions, the differential equation is integrated numerically by the Runge-Kutta method to produce the first estimate, which may be corrected by quasilinearization. This eliminates the time-consuming quasilinearization steps at the intermediate stages. Of course, by using this procedure no knowledge of the solutions at the intermediate stage can be extracted.

This combination was used in Program 5-4 with one application of quasilinearization. Solutions of a problem with initial value c=200 and free-end conditions were compared with those obtained by quasilinearization with a straight-line initial estimate in Table 5-2.

Table 5-1

Minimum Travelling Time Obtained by Joint Use of

Dynamic Programming and Quasilinearization

$x_T=0$, $y_T=0$, $x_T=100\pi$, $y_T=0 \sim 400$ feet

y_T	D.P.	D.P. and Q.L. iter=2	Classical
40	6.40467	6.36369	6.36233
80	5.95519	5.91569	5.91442
120	5.71579	5.68095	5.67980
160	5.60058	5.56864	5.56763
200	5.56509	5.53718	5.53633
240	5.58637	5.56173	5.56104
280	5.64761	5.62579	5.62525
320	5.73690	5.71786	5.71746
360	5.84633	5.82978	5.82950
400	5.97084	5.95570	5.95554

Table 5-2

u(x) Obtained by Joint Use of Invariant Imbedding and Quasilinearization

Take 100x100 grid points in invariant imbedding
400 discrete points in Q.L.

k	Q.L.iter=1	Q.L.iter=2	I.I.and Q.L. iter=1	Classical
40	.21954105E 03	.21985933E 03	.21985918E 03	.21985937E 03
80	.23624176E 03	.23669814E 03	.23669769E 03	.23669809E 03
120	.25041730E 03	.25093545E 03	.25093470E 03	.25093532E 03
160	.26231297E 03	.26286060E 03	.26285956E 03	.26286044E 03
200	.27212100E 03	.27268004E 03	.27267870E 03	.27267995E 03
240	.27999292E 03	.28054369E 03	.28054209E 03	.28054369E 03
180	.28604860E 03	.28656031E 03	.28655839E 03	.28656035E 03
320	.29038306E 03	.29080698E 03	.29080476E 03	.29080707E 03
360	.29307187E 03	.29333534E 03	.29333279E 03	.29333540E 03
400	.29417494E 03	.29417494E 03	.29417201E 03	.29417495E 03

```
          R                  P R O G R A M   5 - 1

          R               FORWARD METHOD OF DYNAMIC PROGRAMMING
          R                SEARCHING WITHIN RESTRICTED REGIONS

$ COMPILE MAD, EXECUTE
           INTEGER JSTART, JSTEP, SW
          R ..........
          R  SAME AS PROGRAM 3 - 1
          R ..........
           THROUGH L2, FOR I = 0, 1, I .G. IMAX
           WHENEVER K .E. 1
           NT(I) = DT(0,I)
           P(I,K) = I
           OTHERWISE
           ALPHA = 1E36
           WHENEVER P(I,K-1) .GE. 0
           JSTEP = -1
           JSTART = I
           WHENEVER JSTART .G. IMAX
           JSTART = IMAX
           END OF CONDITIONAL
           OTHERWISE
           JSTEP = 1
           JSTART = I
           WHENEVER JSTART .L. 0
           JSTART = 0
           END OF CONDITIONAL
           END OF CONDITIONAL
           SW = 1
           THROUGH L3, FOR J = JSTART, JSTEP, SW .E. 2 .OR. J.L.0
          1.OR. J.G. IMAX
           TT = T(J) + DT(J,I)
           WHENEVER TT .L. ALPHA
           ALPHA = TT
           BETA = I-J
           OTHERWISE
           SW = 2
           END OF CONDITIONAL
L3
           NT(I) = ALPHA
           P(I,K) = BETA
           END OF CONDITIONAL
L2
          R ..........
          R  SAME AS PROGRAM 3 - 1
          R ..........
           END OF PROGRAM
```

```
         R              P R O G R A M   5 - 2

         R        BRACHISTOCHRONE PROBLEM WITH FREE END CONDITIONS
         R                  SOLVED BY JOINT USE OF
         R            DYNAMIC PROGRAMMING AND INVARIANT IMBEDDING  .....

$  COMPILE MAD, EXECUTE, PRINT OBJECT, DUMP

          DIMENSION Y(100), T(100), NT(100), JPRED(100), ROLD(100),
         1P(2200,DIM), DT(10300,TIME)
          VECTOR VALUES DIM  = 2,0,0
          VECTOR VALUES TIME = 2,0,0
          EQUIVALENCE (DIM(1),KP1), (DIM(2),KMAX), (TIME(1), IP2),
         1(TIME(2), IP1)
          INTEGER I, II, IP1, IP2, IMAX, J, JL, JH, JPRED, IS,
         1K, KP1, KMAX, P, BETA, FREQ

START     READ AND PRINT DATA XT, YT, IMAX, KMAX, FREQ
          IP1 = IMAX + 1
          IP2 = IMAX + 2
          KP1 = KMAX + 1
          DX  = XT/KMAX
          DY  = YT/IMAX
          TAN = DY/DX
          THROUGH L0, FOR J = 0, 1, J .G. IMAX
          THROUGH L0, FOR I = J, 1, I .G. IMAX
          WHENEVER I .E. 0 .AND. J .E. 0
          DT(J,I) = 1E5
          OTHERWISE
          DS = SQRT.(((I-J)*DY) .P.2 + DX*DX)
          V  = 4.013 * (SQRT.(J*DY) + SQRT.(I*DY))
          DT(J,I) = DS/V
          DT(I,J) = DT(J,I)
          END OF CONDITIONAL
L0
          THROUGH L1, FOR I = 0, 1, I .G. IMAX
          P(I,KMAX)= 0
          ROLD(I)  = 0
          T(I)     = 0.
          Y(I)     = I*DY
L1
          THROUGH L2, FOR K = KMAX-1, -1, K .L. 0
          EXECUTE IMBED.    (Y,ROLD,DX,DY,IMAX,JPRED)
          THROUGH L3, FOR I = 0, 1, I .G. IMAX
          JL = I + JPRED(I) - 2
          WHENEVER JL .L. 0
          JL = 0
          END OF CONDITIONAL
          JH = JL + 4
          WHENEVER JH .G. IMAX
          JH = IMAX
          END OF CONDITIONAL
```

```
          ALPHA = 1E37
          T(0) = 1E5
          THROUGH L4, FOR J = JL, 1, J .G. JH
          TT = T(J) + DT(I,J)
          WHENEVER TT .L. ALPHA
          ALPHA = TT
          BETA  = J-I
          END OF CONDITIONAL
L4
          NT(I)  = ALPHA
          P(I,K) = BETA
          ROLD(I)= P(I,K)*TAN
L3
          PRINT COMMENT $0$
          PRINT COMMENT $0$
          PRINT RESULTS K
          PRINT COMMENT      $0          I                        Y(I)
         1    P(I,K)                     NT(I)              JPRED $
          THROUGH L5, FOR I = 1, 1, I .G. IMAX
          WHENEVER (I/FREQ)*FREQ .E. I
          PRINT FORMAT BRACHI, I, Y(I), P(I,K), NT(I), JPRED(I)
          END OF CONDITIONAL
          T(I) = NT(I)
L5
L2
          PRINT COMMENT $0 THE BEST POLICY $
          THROUGH L6, FOR II = FREQ, FREQ, II .G. 80
          YO = II*DY
          PRINT COMMENT $0$
          PRINT COMMENT $ THE STARTING CONDITIONAL IS$
          PRINT RESULTS II, YO
          PRINT COMMENT $0      K                  NT(I)             Y
         1             SLOPE            SLOPE(INTEGER)$
          I = II
          THROUGH L7, FOR K = 0, 1, K .G. KMAX
          RE = P(I,K)*TAN
          PRINT FORMAT POLICY, K, NT(I), Y(I), RE, P(I,K)
          I = I + P(I,K)
L7
L6
          VECTOR VALUES BRACHI = $ 1I10, 1E30.8, 1I10, 1E30.8, 1I15*$
          VECTOR VALUES POLICY = $ 1I10, 3E20.8, 1I10  *$
          TRANSFER TO START
          END OF PROGRAM
```

```
$ COMPILE MAD, PRINT OBJECT, DUMP
           EXTERNAL FUNCTION (Y,ROLD,DX,DY,IMAX,JPRED)
           DIMENSION RNEW(100)
           INTEGER I, IMAX, J, JPRED, P
           ENTRY TO IMBED.
           Y(0) = 0.1
           TAN  = DY/DX
           THROUGH L1, FOR I = 0, 1, I .G. IMAX
           S = Y(I) + ROLD(I)*DX
           WHENEVER.ABS. (ROLD(I)).L. 1E-6
           R = ROLD(I)
           OR WHENEVER ROLD(I) .L. 0.
           THROUGH L2,FOR J=I,-1,J.E.0 .OR.(S.G.Y(J-1) .AND.S.LE.Y(J))
L2
           WHENEVER J .E. 0
           J = 1
           END OF CONDITIONAL
           R = (ROLD(J)-ROLD(J-1))*(S-Y(J-1))/DY + ROLD(J-1)
           OTHERWISE
           THROUGHL3,FOR J=I,1,J.E.IMAX .OR.(S.G.Y(J).AND.S.LE.Y(J+1))
L3
           WHENEVER J .E. IMAX
           R = ROLD(IMAX)
           OTHERWISE
           R = (ROLD(J+1)-ROLD(J))*(S-Y(J))/DY + ROLD(J)
           END OF CONDITIONAL
           END OF CONDITIONAL
           WHENEVER .ABS.(ROLD(I)) .G. 1E6
           ROLD(I) = 1E6*(ROLD(I)/(.ABS.(ROLD(I))))
           END OF CONDITIONAL
           RNEW(I) = R+(1.+ROLD(I)*ROLD(I))*DX/(2.*Y(I))
           JPRED(I) = RNEW(I)/TAN
L1
           THROUGH L4, FOR I = 0, 1, I .G. IMAX
           ROLD(I) = RNEW(I)
L4
           FUNCTION RETURN
           END OF FUNCTION
$ DATA
 XT=314.15926, YT=400., IMAX=100, FREQ=10, KMAX=20*
```

```
          R              P R O G R A M   5 - 3

          R              BRACHISTOCHRONE PROBLEM SOLVED BY JOINT USE OF
          R              DYNAMIC PROGRAMMING AND QUASILINEARIZATION

$    COMPILE MAD, EXECUTE, PRINT OBJECT, DUMP

           DIMENSION Y(80), T(80), NT(80), P(1800,DIM), DT(6600,TIME),
          1YR(6), FR(6), QR(6), PA(800), H1(800), H2(800), DPA(800),
          2DH1(800), DH2(800), U(800), W(800)
           VECTOR VALUES DIM  = 2,0,0
           VECTOR VALUES TIME = 2,0,0
           EQUIVALENCE (DIM(1),KP1), (DIM(2),KMAX), (TIME(1),IP2),
          1(TIME(2), IP1)

           INTEGER I, IMAX, IFREQ, IP1, IP2, II, ITER, ITMAX,
          1J,
          2 K, KK, KMAX,  QK, QKMAX, KP1, KP,
          3P, BETA, R
START
           READ AND PRINT DATA XT, YT, YO, IMAX, KMAX, KK, ITMAX, IFREQ
           QKMAX = KK*KMAX
           KP = QKMAX/20
           IP1 = IMAX + 1
           IP2 = IMAX + 2
           KP1 = KMAX + 1
           DX  = XT/KMAX
           DY = (YT-YO)/IMAX
           H = DX/KK
           TAN = DY/DX
           EPS = 100.

          R   CONSTRUCTING  MATRIX FOR DELTA T
           THROUGH LO, FOR J = 0,1, J.G.IMAX
           THROUGH LO, FOR I = J, 1, I.G. IMAX
           WHENEVER I .E. 0 .AND. J .E. 0
           DT(J,I) = 1E5
           OTHERWISE
           DS = SQRT.(((I-J)*DY).P.2 + DX*DX)
           V = 4.013 * (SQRT.(J*DY) + SQRT.(I*DY))
           DT(J,I) = DS/V
           DT(I,J) = DT(J,I)
           END OF CONDITIONAL
LO

          R   DYNAMIC PROGRAMMING - FORWARD  SOLUTION
           P(0,0) = 0
           PRINT COMMENT $0                   I                         Y
          1 P(I,KMAX)                    NY  $
```

```
         THROUGH L1, FOR K = 1, 1, K .G. KMAX
         THROUGH L2, FOR I = 0, 1, I .G. IMAX
         WHENEVER K .E. 1
         NT(I) = DT(0,I)
         P(I,K) = I
         OTHERWISE
         ALPHA = 1E37
         THROUGH L3, FOR J = 0, 1, J .G. IMAX
         TT = T(J) + DT(J,I)
         WHENEVER TT .L. ALPHA
         ALPHA = TT
         BETA = I-J
         END OF CONDITIONAL
L3
         NT(I)  = ALPHA
         P(I,K) = BETA
         END OF CONDITIONAL
L2
         THROUGH L4, FOR I = 0,1, I.G.IMAX
         WHENEVER K .E. KMAX .AND. (I/IFREQ)*IFREQ .E.I
         Y(I) = I*DY
         PRINT FORMAT BRACHI, I, Y(I), P(I,K), NT(I)
         END OF CONDITIONAL
         T(I) = NT(I)
L4
L1

        R   IDENTIFY THE BEST POLICY AND PREPARE FOR Q.L. CORRECTION
         THROUGH L5, FOR II=IMAX, -IFREQ, II .L. IFREQ
         UT = II*DY
         UO = 0.
         PRINT COMMENT $1 SOLUTION WITH END POINT AT  $
         PRINT RESULTS II,     UT
         PRINT COMMENT  $0       K                X                 Y
        1                        SLOPE          P(I,K)     $
         I = II
         W(QKMAX) = P(I,KMAX)*TAN
         THROUGH L6, FOR QK = QKMAX, -1, QK .L. 0
         WHENEVER (QK/KK)*KK .E. QK
         K = QK/KK
         SF = P(I,K)*TAN
         U(QK) = I*DY
         WHENEVER QK .NE. 0
         W(QK-1) = SF
         END OF CONDITIONAL
         I = I-P(I,K)
         OTHERWISE
         W(QK-1) = SF
         U(QK) = U(QK+1)-W(QK)*H
         END OF CONDITIONAL
         WHENEVER (QK/KP)*KP .E. QK
         XA = QK*H
```

```
      PRINT FORMAT POLICY, QK, XA, U(QK), W(QK), P(I,K)
      END OF CONDITIONAL
L6
R     QUASILINEARIZATION CORRECTOR
      EXECUTE QUASI.     (U,W,QT,PA,H1,H2,QKMAX,EPS,ITMAX,H,UO,UT)
      PRINT COMMENT $0$
      PRINT COMMENT     $  QK          X                 PA
1  H1                    H2                 U                       V   $
      THROUGH L9, FOR QK = 0, KP, QK .G. QKMAX
      X = H*QK
      PRINT FORMAT LINEAR, QK,X,PA(QK),H1(QK),H2(QK),U(QK), W(QK)
L9
      PRINT RESULTS QT
L5
      TRANSFER TO START
      VECTOR VALUES BRACHI = $ 1I10,  E30.8, 1I10,  E30.8  *$
      VECTOR VALUES POLICY = $ 1I10, 3E20.8, 1I10, 1E20.8  *$
      VECTOR VALUES LINEAR = $ 1I5, 1E14.4, 5E17.8  *$
      END OF PROGRAM
```

```
$   COMPILE MAD, EXECUTE, PRINT OBJECT, DUMP
            EXTERNAL FUNCTION (U,W,QT,PA,H1,H2,QKMAX,EPS,ITMAX,H,UO,UT)
            DIMENSION DPA(800), DH1(800), DH2(800), FR(10), YR(10),QR(10)
            INTEGER I,IMAX,IFREQ,ITER, ITMAX,K,KK,KMAX,QK,QKMAX
            ENTRY TO QUASI.
           R    ITER-TH APPROXIMATION
            THROUGH L7, FOR ITER = 1,1, ITER .G. ITMAX
            U(0) = 0.01
            PA(0) = 0.
            H1(0) = 1.
            H2(0) = 0.
            DPA(0)= 0.
            DH1(0)= 0.
            DH2(0)= 1.
            YR(1) =  PA(0)
            YR(2) = DPA(0)
            YR(3) =  H1(0)
            YR(4) = DH1(0)
            YR(5) =  H2(0)
            YR(6) = DH2(0)
            X = 0.
            EXECUTE SETRKD.(6,YR(1),FR(1),QR,X,H)
            THROUGH L8, FOR QK = 1,1, QK .G. QKMAX
CALLRK      S = RKDEQ.(0)

            WHENEVER S .E. 1.0
            FR(1) = YR(2)
            WHENEVER FR(1) .G. EPS
            FR(1) = EPS
            END OF CONDITIONAL
            FR(3) = YR(4)
            WHENEVER FR(3) .G. EPS
            FR(3) = EPS
            END OF CONDITIONAL
            FR(5) = YR(6)
            WHENEVER FR(5) .G. EPS
            FR(5) = EPS
            END OF CONDITIONAL
            GU = (1.+W(QK)*W(QK))/(2.*U(QK)*U(QK))
            WHENEVER GU .G. 1E6
            GU = 1E6
            END OF CONDITIONAL
            GW = -W(QK)/U(QK)
            WHENEVER .ABS.(GW) .G. 1E6
            GW = 1E6*(GW/(.ABS.(GW)))
            END OF CONDITIONAL
            FR(2) = GU*(YR(1)-2.*U(QK)) + GW*(YR(2) - W(QK))
            WHENEVER .ABS.(FR(2)) .G. EPS
            FR(2) = EPS*(FR(2)/(.ABS.(FR(2))))
            END OF CONDITIONAL
            FR(4) = GU*YR(3) + GW*YR(4)
```

```
            WHENEVER .ABS.(FR(4)) .G. EPS
            FR(4) = EPS*(FR(4)/(.ABS.(FR(4))))
            END OF CONDITIONAL
            FR(6) = GU*YR(5) + GW*YR(6)
            WHENEVER .ABS.(FR(6)) .G. EPS
            FR(6) = EPS*(FR(6)/(.ABS.(FR(6))))
            END OF CONDITIONAL
            TRANSFER TO CALLRK

            OTHERWISE
            PA(QK)  = YR(1)
            H1(QK)  = YR(3)
            H2(QK)  = YR(5)
            DPA(QK) = YR(2)
            DH1(QK) = YR(4)
            DH2(QK) = YR(6)
            END OF CONDITIONAL
L8
            DIN = H1(0)*H2(QKMAX) - H1(QKMAX)*H2(0)
            AA = UO - PA(0)
            BB = UT - PA(QKMAX)
            C1 = ( AA*H2(QKMAX) - BB*H2(0))/DIN
            C2 = (-AA*H1(QKMAX) + BB*H1(0))/DIN
            PRINT RESULTS C1, C2
            THROUGH L10, FOR QK = 0,1, QK .G. QKMAX
            W(QK) = DPA(QK) + C1*DH1(QK) + C2*DH2(QK)
            U(QK) =  PA(QK) + C1* H1(QK) + C2* H2(QK)
            WHENEVER QK .E. 0
            QT = 0.
            OTHERWISE
            DS = SQRT.((U(QK)-U(QK-1)).P.2 + H*H)
            V = 4.013*(SQRT.(U(QK)) + SQRT.(U(QK-1)))
            QT = QT + DS/V
            END OF CONDITIONAL
L10
L7
            FUNCTION RETURN
            END OF FUNCTION
$ DATA
YO = 0., XT = 314.15926, YT = 400., IMAX = 40, KMAX = 20,  IFREQ = 4,
KK =20,  ITMAX = 2*
```

```
R                 P R O G R A M   5 - 4

R   BRACHISTOCHRONE PROBLEM WITH FREE END CONDITIONS SOLVED BY
R ... JOINT USE OF INVARIANT IMBEDDING AND QUASILINEARIZATION

$ COMPILE MAD, EXECUTE, PRINT OBJECT, DUMP

            INTEGER I, IMAX, ITER, ITMAX, IFREQ, J, JMAX, K, KK, KP,KMAX,
           1 M, KK
            DIMENSION Y(100), ROLD(800), RNEW(800), YR(6), FR(6), QR(6),
           1PA(800), H1(800), H2(800), DPA(800), DH1(800), DH2(800),
           2U(800), W(800)
            EQUIVALENCE (IMAX, JMAX)
START
            READ AND PRINT DATA XT, YO,OYT, IMAX, ITMAX, IFREQ, KMAX, KP,
           1KK, EPS
            DX = XT/KMAX
            DY = (YT-YO)/IMAX
            THROUGH L1, FOR I=0,1,I.G.IMAX
            Y(I) = I*DY
            ROLD(I) = 0.
L1

           R FIND INITIAL SLOPE BY INVARIANT IMBEDDING
            THROUGH L2, FOR K = (KMAX-KK), -KK, K.L. 0
            X = K*DX
            WHENEVER K .E. 0
            PRINT COMMENT $0INITIAL CONDITIONS $
            PRINT RESULTS K , X
            PRINT COMMENT  $          I                    Y(I)
           1 SLOPE       M  $
            END OF CONDITIONAL
            THROUGH L3, FOR I=0, 1, I.G. IMAX
            S = Y(I) + ROLD(I)*DX*KK
            WHENEVER .ABS.(ROLD(I)).L. 1E-6
            R = ROLD(I)
            M = I
            OR WHENEVER ROLD(I) .L.0.
            THROUGH L4,FOR J=I,-1, J.E.0 .OR. (S.G.Y(J-1) .AND.S.LE.Y(J))
L4
            WHENEVER J .E. 0
            J = 1
            END OF CONDITIONAL
            R = (ROLD(J)-ROLD(J-1))*(S-Y(J-1))/DY + ROLD(J-1)
            M = J
            OTHERWISE
            THROUGH L5,FOR J=I,1,J.E.IMAX .OR.(S.G.Y(J) .AND.S.LE.Y(J+1))
L5
            WHENEVER J.E. JMAX
            J = JMAX-1
            END OF CONDITIONAL
            R = (ROLD(J+1)-ROLD(J))*(S-Y(J))/DY + ROLD(J)
            M = J
            END OF CONDITIONAL
```

```
              WHENEVER .ABS.(ROLD(I)) .G. 1E6
              ROLD(I) = 1E6*(ROLD(I)/(.ABS.(ROLD(I))))
              END OF CONDITIOANL
              Y(0) = 0.1
              RNEW(I) = R+(1+ROLD(I)*ROLD(I))*DX*KK/(2.*Y(I))
              WHENEVER K.E.0  .AND. (I/IFREQ)*IFREQ .E. I
              PRINT FORMAT IMBED, I, Y(I), ROLD(I), M
              END OF CONDITIONAL
 L3
              THROUGH L6, FOR I = 0, 1, I .G. IMAX
              ROLD(I) = RNEW(I)
 L6
 L2
            R   INITIAL INTEGRATION
              THROUGH L7, FOR I = IFREQ, IFREQ, I .G. IMAX
              UO = Y(I)
              YR(1) = Y(I)
              YR(2) = ROLD(I)
              X = 0.
              EXECUTE SETRKD.(2,YR(1),FR(1),QR,X,DX)
              THROUGH LRK1, FOR K = 1,1, K .G. KMAX
 RK1          S = RKDEQ.(0)
              WHENEVER S .E. 1.
              FR(1) = YR(2)
              FR(2) = -(1. + FR(1)*FR(1))/(2.*YR(1))
              TRANSFER TO RK1

              OTHERWISE
              U(K) = YR(1)
              W(K) = YR(2)
              END OF CONDITIONAL
 LRK1

            R USE Q. L. AS A CORRECTOR
              THROUGH L8, FOR ITER = 1,1, ITER .G. ITMAX
              PA(0) = 0.
              H1(0) = 1.
              H2(0) = 0.
              DPA(0)= 0.
              DH1(0)= 0.
              DH2(0)= 1.
              YR(1) =  PA(0)
              YR(2) = DPA(0)
              YR(3) =  H1(0)
              YR(4) = DH1(0)
              YR(5) =  H2(0)
              YR(6) = DH2(0)
              X = 0.
              EXECUTE SETRKD.(6,YR(1),FR(1),QR,X,DX)
              THROUGH LRK, FOR K = 1,1, K.G.KMAX
 CALLRK       S = RKDEQ.(0)
```

```
         WHENEVER S .E. 1.0
         FR(1) = YR(2)
         WHENEVER FR(1) .G. EPS
         FR(1) = EPS
         END OF CONDITIONAL
         FR(3) = YR(4)
         WHENEVER FR(3) .G. EPS
         FR(3) = EPS
         END OF CONDITIONAL
         FR(5) = YR(6)
         WHENEVER FR(5) .G. EPS
         FR(5) = EPS
         END OF CONDITIONAL
         GU = (1.+W(K)*W(K))/(2.*U(K)*U(K))
         WHENEVER GU .G. 1E6
         GU = 1E6
         END OF CONDITIONAL
         GW = -W(K)/U(K)
         WHENEVER .ABS.(GW) .G. 1E6
         GW = 1E6*(GW/(.ABS.(GW)))
         END OF CONDITIONAL
         FR(2) = GU*(YR(1)-2.*U(K)) + GW*(YR(2) - W(K))
         WHENEVER .ABS.(FR(2)) .G. EPS
         FR(2) = EPS*(FR(2)/(.ABS.(FR(2))))
         END OF CONDITIONAL
         FR(4) = GU*YR(3) + GW*YR(4)
         WHENEVER .ABS.(FR(4)) .G. EPS
         FR(4) = EPS*(FR(4)/(.ABS.(FR(4))))
         END OF CONDITIONAL
         FR(6) = GU*YR(5) + GW*YR(6)
         WHENEVER .ABS.(FR(6)) .G. EPS
         FR(6) = EPS*(FR(6)/(.ABS.(FR(6))))
         END OF CONDITIONAL
         TRANSFER TO CALLRK

         OTHERWISE
         PA(K)  = YR(1)
         H1(K)  = YR(3)
         H2(K)  = YR(5)
         DPA(K) = YR(2)
         DH1(K) = YR(4)
         DH2(K) = YR(6)
         END OF CONDITIONAL
LRK
         DIN = H1(0)*DH2(KMAX) - DH1(KMAX)*H2(0)
         AA = UO - PA(0)
         C1 = ( AA*DH2(KMAX) + DPA(KMAX)*H2(0))/DIN
         C2 = (-AA*DH1(KMAX) - DPA(KMAX)*H1(0))/DIN
         PRINT COMMENT $0$
         PRINT RESULTS I, UO, ITER
         PRINT RESULTS C1, C2
         PRINT COMMENT    $    K          X                   PA
        1  H1                 H2                 U                  V
        2          QT   $
```

```
         THROUGH L9, FOR K = 0, 1, K .G. KMAX
         U(K) =  PA(K) + C1* H1(K) + C2* H2(K)
         W(K) = DPA(K) + C1*DH1(K) + C2*DH2(K)
         X = K*DX
         WHENEVER K .E. 0
         QT = 0.
         OTHERWISE
         DS = SQRT.( (U(K)-U(K-1)).P.2+ DX*DX )
         V = 4.013*(SQRT.(U(K))+SQRT.(U(K-1)))
         QT = QT + DS/V
         END OF CONDITIONAL
         WHENEVER (K/KP)*KP .E. K
         PRINT FORMAT LINEAR, K, X,PA(K),H1(K),H2(K),U(K),W(K),QT
         END OF CONDITIONAL
L9
         U(0) = 0.001
L8
         PRINT COMMENT $0$
L7
         TRANSFER TO START
         VECTOR VALUES IMBED =  $  1I10, 2E20.8, 1I10    *$
         VECTOR VALUES LINEAR = $ 1I5, 1E12.4, 6E17.8   *$
         END OF PROGRAM

$ DATA
 XT = 314.15926, YO=0., YT=400., IMAX=100,ITMAX=1, KMAX=400,IFREQ =10,
 KP=20, KK=4, EPS=100*
```

CONCLUSIONS

Modern digital computers can solve a great number of initial-value problems with accuracy and speed. The conventional method of solving two-point boundary-value problems by estimating initial slopes does not make efficient use of their capabilities. In addition, the accuracy achieved at the boundary points does not guarantee equal accuracy throughout at intermediate points. The first difficulty may be mitigated by using the technique of invariant imbedding or dynamic programming, while the accuracy in the interval may be improved significantly by quasilinearization.

The convergence of solution obtained by quasilinearization depends solely upon the suitability and closeness of the initial estimate to the solution. This original estimate may be obtained by invariant imbedding or dynamic programming. A major difficulty in applying quasilinearization arises in obtaining the multipliers from high-dimensional systems of linear algebraic equations. Serious errors may result when inaccurately determined multipliers are used in combinations of solutions. Invariant imbedding eliminates this difficulty by producing functions which yield the unknown initial values directly [18].

Dynamic programming reduces, in large scale, the labor of searching for optimal paths. Since it bypasses the requirement for knowing the differential equation governing the optimal curve, it is particularly suited for solving multi-stage multi-decision problems where the differential equation does not exist. If the differential equation governing the optimal path can be derived or a continuous problem giving differential equation is solved as a discrete multistage multidecision process, the computing time may further be reduced by using the technique of searching over a restricted region either by utilizing the slope characteristics of the differential equation or by joint use with invariant imbedding. Accuracy of dynamic programming depends upon the fineness of the selected grid, but the size of the problem is limited by the available memory of a computer. Combining dynamic programming and quasilinearization avoids this difficulty while producing accurate results.

APPENDIX

CLASSICAL SOLUTION OF BRACHISTOCHRONE PROBLEM

The brachistochrone problem requires that we find the path of least-time between two points in a gravitational field. Since gravitational force is the only force acting on the mass, the travelling time may be expressed as

$$T = \int_0^{t_B} dt = \int_0^{s_B} \frac{ds}{V} = \int_0^b \sqrt{\frac{1+y'^2}{2gy}}\, dx$$

$$= \int_0^b F(y,y')\, dx \qquad \text{(A-1)}$$

where ds stands for the infinitesimal chord length, V is the velocity, and g is the constant of gravitational acceleration. In order to minimize T, we apply Euler's equation to the integrand F, that is,

$$\frac{\partial F}{\partial y} - \frac{d}{dx}\left[\frac{\partial F}{\partial y'}\right] = 0 \qquad \text{(A-2)}$$

where

$$F = \sqrt{\frac{1+y'^2}{2gy}} \qquad \text{(A-3)}$$

By performing the operation required by Eq.(A-2) we are led to the equation

$$y'' = -\frac{1+y'^2}{2y} \qquad \text{(A-4)}$$

which may be integrated to yield

$$1 + y'^2 = \frac{c_1}{y} \tag{A-5}$$

where c_1 is a constant of integration.

In turn, by manipulation of the terms and performing a second integration, we obtain

$$x = \frac{c_1}{2}(u - \sin u) + c_2 \tag{A-6}$$

where $u = \cos^{-1}(1-2y/c_1)$ and c_2 is the second constant of integration. Since the path starts at the origin, at $x = y = 0$, $u = 0$, which implies that $c_2 = 0$. Thus, we are led to the solution

$$x = \frac{c_1}{2}(u - \sin u) \qquad \text{(a)}$$

$$\tag{A-7}$$

$$y = \frac{c_1}{2}(1 - \cos u) \qquad \text{(b)}$$

which we recognize as the parametric form of the equation for a cycloid, that is

$$x = r(\theta - \sin\theta) \qquad \text{(a)}$$

$$\tag{A-8}$$

$$y = r(1 - \cos\theta) \qquad \text{(b)}$$

where r ($=c_1/2$) is the radius of the base circle, and θ ($=u$) is the angular displacement of the base circle.

It can be shown that the travelling time along a cycloidal path is given by

$$t = \sqrt{r/g}\ \theta = \frac{\theta}{\omega} \qquad \text{(A-9)}$$

where $\omega = \sqrt{g/r}$ is a constant for particular cycloidal path.

In summary:

The path of least-time in a gravitational field is a part of a cycloid. The travelling time along any section of the cycloid is proportional to the angular displacement of the base circle by which that section of the curve is generated. The angular velocity of the base circle ω is constant ($=\sqrt{g/r}$), where r is the radius of the base circle and g is the constant of gravitational acceleration.

BIBLIOGRAPHY

[1] Ambarzumiam, V. A. "On the Scattering of Light by a Diffuse Medium," Compt. rend. Doklady Acad. Sci. U.R.S.S. V.38, p. 257, 1943.

[2] Chandrasekhar, S., Radiative Transfer, Oxford University Press, London, 1950.

[3] Bellman, R. E. and R. E. Kalaba, "On the Principle of Invariant Imbedding and Propagation Through Inhomogeneous Media," Proc. Nat. Acad. Sci. USA V. 42 (1956), pp. 629-632.

[4] Bellman, R. E., Dynamic Programming, Princeton University Press, Princeton, New Jersey, 1957.

[5] Bellman, R. E. and R. E. Kalaba, "On the Principle of Invariant Imbedding and Diffuse Reflection from Cylindrical Regions," Proc. Nat. Acad. Sci. USA, V. 43 (1957), pp. 514-517.

[6] Bellman, R. E., R. E. Kalaba, and G. M. Wing, "On the Principle of Invariant Imbedding and One-dimensional Neutron Multiplication," Proc. Nat. Acad. Sci. USA, V. 43 (1957), pp. 517-520.

[7] Bellman, R. E. and R. E. Kalaba, "Random Walk, Scattering, and Invariant Imbedding 1. One-dimensional Discrete Case," Proc. Nat. Acad. Sci. USA, V. 43 (1957), pp. 930-933.

[8] Bellman, R. E., R. E. Kalaba, and G. M. Wing, "Invariant Imbedding and Mathematical Physics-I: Particle Processes," J. of Mathematical Physics, V. 1 (1960), pp. 280-308.

[9] Bellman, R., R. E. Kalaba, and G. M. Wing, "Dissipation Function and Invariant Imbedding, 1", Proc. Nat. Acad. Sci. USA, V. 46 (1960), pp. 1145-1147.

[10] Bellman, R. E., R. E. Kalaba, and G. M. Wing, "Invariant Imbedding, Conservation Relations, and Non-linear Equations with Two-point Boundary Values," Proc. Nat. Acad. Sci. USA, V. 46 (1960), pp. 1258-1260.

[11] Bellman, R. E., R. E. Kalaba, and G. M. Wing, "Invariant Imbedding and Reduction of Two-point Boundary Values Problems to Initial Value Problems," Proc. Nat. Acad. Sci. USA, V. 46 (1960) pp. 1646-1649.

[12] Bellman, R. E., Adaptive Control Process, A guided tour, Princeton University Press, Princeton, New Jersey, 1960.

[13] Bellman, R. E. and R. E. Kalaba, "On the Fundamental Equations of Invariant Imbedding-I," Proc. Nat. Acad. Sci. USA, V. 47 (1961), pp. 336-338.

[14] Bellman, R. E. and S. Dreyfus, Applied Dynamic Programming, Princeton University Press, Princeton, New Jersey, 1962.

[15] Bellman, R. E., H. Kagiwada, and R. E. Kalaba, "A Computational Procedure for Optimal System Design and Utilization," Proc. Nat. Acad. Sci. USA, V. 48 (1962), pp. 1524-1528.

[16] Bellman, R. E. and R. E. Kalaba, Dynamic Programming, Invariant Imbedding and Quasilinearization, Comparisons and Interconnections, the Rand Corporation, Santa Monica, California, 1964.

[17] Bellman, R. E., H. Kagiwada, R. E. Kalaba, and R. Spidhar, Invariant Imbedding and Nonlinear Filtering Theory, the Rand Corporation, Santa Monica, California, 1964.

[18] Bellman, R. E., H. Kagiwada, and R. E. Kalaba, Numerical Studies of A Two-point Nonlinear Boundary Value Problem Using Dynamic Programming, Invariant Imbedding, and Quasilinearization, the Rand Corporation, Santa Monica, California, 1964.

[19] Bellman, R. E. and R. E. Kalaba, Quasilinearization and Boundary Value Problems. American Elsevier Publishing Co., New York, 1965.

[20] Bellman, R. E., H. Kagiwada, and R. E. Kalaba, Invariant Imbedding and the Numerical Integration of Boundary Value Problem for Unstable Systems of Ordinaty Differential Equations, the Rand Corporation, Santa Monica, California, 1965.

[21] Dreyfus, Stuart E., Dynamic Programming and the Calculus of Variations, the Rand Corporation, Santa Monica, California, 1965.

[22] Fan, Lian-Tsen, and Chiu-Sen Wan, The Discrete Minimum Principle-A Study of Multistage System Optimization, John Wiley and Sons, New York, 1964.

[23] Hildebrand, Francis B., Advanced Calculus for Applications, Prentice-Hall, Inc., New Jersey, 1962.

[24] Kalaba, R. E., "Computational Considerations for Some Deterministic and Adaptive Control Processes," Optimization Techniques, Edited by George Leitman, Academic Press, 1962.

[25] Tou, Jurious, Modern Control Theory, McGraw-Hill, New York, 1965.