AD-A239 230

ATION PAGE

Form Approved
OMB No. 0704-0188

...ORT DATE	3. REPORT TYPE AND DATES COVERED ~~THESIS~~/DISSERTATION

4. TITLE AND SUBTITLE
A New Approach for the Solution of Optimal Control Problems on Parallel Machines

5. FUNDING NUMBERS

6. AUTHOR(S)
Daniel J. Stech, Captain

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
AFIT Student Attending: University of Colorado

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/CI/CIA-91-002d

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
AFIT/CI
Wright-Patterson AFB OH 45433-6583

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for Public Release IAW 190-1
Distributed Unlimited
ERNEST A. HAYGOOD, 1st Lt, USAF
Executive Officer

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

14. SUBJECT TERMS			**15. NUMBER OF PAGES** 93 **16. PRICE CODE**
17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**

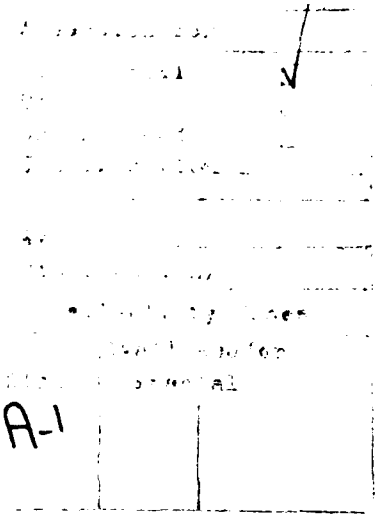
A NEW APPROACH FOR THE SOLUTION OF OPTIMAL CONTROL PROBLEMS ON PARALLEL MACHINES

by

DANIEL J. STECH

B.S., United States Air Force Academy, 1981

M.S., Air Force Institute of Technology, 1985



A thesis submitted to the

Faculty of the Graduate School of the

University of Colorado in partial fulfillment

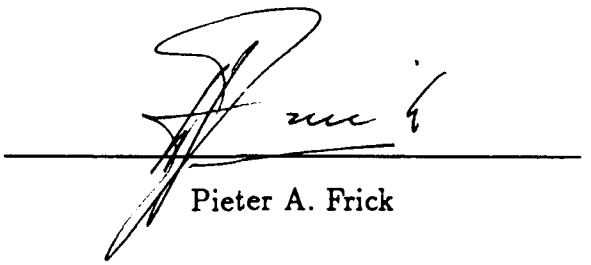
of the requirements for the degree of

Doctor of Philosophy

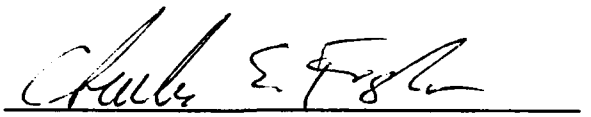
Department of Electrical Engineering

1990

This thesis for the Doctor of Philosophy degree by
Daniel J. Stech
has been approved for the
Department of
Electrical Engineering
by



Pieter A. Frick



Charles E. Fosha

Date 19 NOV 93

Stech, Daniel J. (Ph.D., Electrical Engineering)

A New Approach for the Solution of Optimal Control Problems on Parallel Machines

Thesis directed by Professor Pieter Frick

This thesis develops a highly parallel solution method for nonlinear optimal control problems. Balakrishnan's epsilon method is used in conjunction with the Rayleigh-Ritz method to convert the dynamic optimization of the optimal control problem into a static optimization problem. Walsh functions and orthogonal polynomials are used as basis functions to implement the Rayleigh-Ritz method. The resulting static optimization problem is solved using matrix operations which have well defined massively parallel solution methods. To demonstrate the method, a variety of nonlinear optimal control problems are solved. The nonlinear Raleigh problem with quadratic cost and nonlinear van der Pol problem with quadratic cost and terminal constraints on the states are solved in both serial and parallel on an eight processor *Intel Hypercube*. The solutions using both Walsh functions and Legendre polynomials as basis functions are given. In addition to these problems which are solved in parallel, a more complex nonlinear minimum time optimal control problem and nonlinear optimal control problem with an inequality constraint on the control are solved. Results show the method converges quickly, even from relatively poor initial guesses for the nominal trajectories. Parallel results show the method also offers significant speedup and rapid solution times even on the small hypercube computer. Potential parallelization is much greater than eight and significantly greater speed-up might be realized on larger parallel machines.

This work is dedicated to my wonderful family, Bernadette, Joey, and especially to Anna who was born during this project. Being with them reminds me that I must be doing something right.

ACKNOWLEDGEMENTS

The author wishes to thank his thesis advisor, Dr. Pieter Frick. His generosity with both his ideas and his time made the on time completion of this research a reality.

CONTENTS

TABLES

Table

FIGURES

Figure

CHAPTER 1

INTRODUCTION

The theory of nonlinear optimal control is relatively well developed. Unfortunately, due to the high computational cost, nonlinear optimal control theory is rarely implemented in actual control hardware. Instead, nonlinear systems are linearized and linear control laws are used, or results using full nonlinear system equations are calculated off line and then stored in memory for actual implementation. Both methods result in suboptimal forms of control. In recent years, parallel processors have become well developed. The fastest computers are now parallel processing computers. As parallel processing technology develops, it should offer parallel processing machines with enormous computational power for very little cost. With inexpensive parallel processing power it may become reasonable to consider solving nonlinear optimal control problems in real time. Real time solution of nonlinear optimal control problems is the motivation behind developing nonlinear optimal control methods which may be implemented on parallel processing machines.

There have been relatively few investigations into parallel methods of solving optimal control problems. This is probably due to the slow development of parallel computers and the lack of massively parallel methods of solving the system dynamical equations. The following is a review of parallel methods which have been developed to solve optimal control problems

The first research into parallel processing for optimal control was done by Larsen and Tse [1]. Their method was to give a parallel implementation of dynamic programming and use this parallel dynamic programming method to solve the nonlinear optimal control problem.

Travassos and Kaufman [2] chose to develop a parallel implementation of the shooting method (variation of extremals). They used parallel initial costate searches along with a parallel state equation solver to formulate a parallel method for solving nonlinear two point boundary value problems. They also proposed the type of parallel computer architecture needed to run the algorithm.

Finally, Meyer and Podrazik [3] looked at parallel gradient - based methods for solving the discrete optimal control problem. They used parallel gradient search methods along with a parallel linear recurrence solver to formulate a parallel gradient method for discrete systems.

Two major issues are not adequately addressed in the cited literature; interprocessor communication and discretization of continuous systems. None of the above researchers actually implemented their method on a parallel machine. Because of this, the very important issue of communications between processors is not adequately addressed. Excessive or untimely communication between processors can severely limit the speed-up obtainable on a parallel machine. Secondly, all the above methods require that the system be discrete or the system be simulated in some discrete manner (state equation solving, etc.). For some very nonlinear problems this is a significant drawback since the computational load to obtain a discrete system from a continuous one can be excessive. This dissertation suggests a new parallel approach to the solution of the optimal control problem which addresses both issues.

The fundamental idea of the new method is simple. Most methods for solving two point boundary value problems involve solving the system dynamical equations many times. This is typically the most computationally intensive part of these methods. The solution of ordinary differential equations on a computer is inherently serial. For most numerical methods of solving ordinary differential equations, calculation of the next point is dependent on current points or past points. As a result, solution of the dynamical equations introduces a type of serial bottleneck into parallelizing these types of methods. In order to obtain a solution method which is massively parallel, the dynamic optimization problem is converted into a static optimization problem. This conversion is obtained by use of Balakrishnan's epsilon method which avoids solving the dynamical equations. The new static problem can be optimized using

massively parallel methods. Computational results are very promising. The method has very good convergence properties and has highly parallel implementations.

Chapter 2 develops background necessary to obtain the parallel solutions for optimal control problems. Some key ideas for parallel processing for numerical computation are presented, Walsh functions are reviewed, and an introduction to Balakrishnan's epsilon method is given. In Chapter 3, the linear optimal control problem with quadratic cost is formulated as the epsilon problem and is solved using the Rayleigh - Ritz method with Walsh functions as a basis. This conversion gives a problem formulation in terms of matrix operations which can be implemented in parallel. One parallel implementation for an eight node *Intel Hypercube* is given and computational results for a linear example are presented. In Chapter 4, the nonlinear problem is solved by quasilinearization. Computational results for a nonlinear optimal control problem with quadratic cost are given. Also in chapter 4 the method is extended to include the case of terminal constraints on the state. Chapter 5 reviews orthogonal polynomials, specifically the Legendre polynomials, and gives some of their properties. The Legendre polynomials are then used as basis functions to solve the two nonlinear examples given in Chapters 4 and 5. Chapter 6 shows how a variety of nonlinear operations can be performed on the Walsh coefficients which approximate a function. A nonlinear minimum time optimal control problem is then solved. Chapter 7 uses some results of Chapter 6 to solve nonlinear optimal control problems with inequality constraints. A numerical example is given. Chapter 8 draws final conclusions and offers some additional areas of research which may be of interest. Appendix A reviews some systems notation used throughout the thesis.

CHAPTER 2

BACKGROUND

This chapter covers the background necessary to develop the parallel solution of the optimal control problem.

Parallel Processing for Numerical Computation

Much research has been done in the area of parallel processing for numerical computation. In this section an approach is given for developing parallel implementations for numerical computation problems. Briefly, the idea is to formulate a given numerical problem in terms of smaller operations which are known to have efficient, well defined parallel implementations. It is then possible to combine these smaller operations into a larger parallel scheme which may retain the desirable characteristics of the smaller operations.

There are three properties which are desirable in a parallel implementation, efficiency, scalability, and flexibility. Each of these will now be defined and addressed. Consider first some performance measures for parallel implementations. Let $T^*(n)$ be the time to solve a problem serially, where n is some measure of the size of the problem. For example, if multiplying two $n \times n$ matrices, $T^*(n) = O(n^3)$. Let $T_p(n)$ be the time to solve a problem with p processors. Then the speed-up is defined as

$$S_p(n) = \frac{T^*(n)}{T_p(n)} \tag{2.1}$$

And the efficiency is

$$E_p(n) = \frac{T^*(n)}{pT_p(n)} = \frac{S_p(n)}{p} \tag{2.2}$$

so $E_p(n)$ is a measure of the time a processor is doing work. If $E_p(n)$ is bounded away from zero as n and p increase, then as n increases, p can be increased and $S_p(n)$ will continue to increase. For example, let

$$E_p(n) = c$$

where c is some constant so the efficiency is bounded away from zero. Then

$$\frac{S_p(n)}{p} = c$$

and

$$S_p(n) = pc$$

so if $p \uparrow$, then $S_p(n) \uparrow$. If $c \downarrow 0$, efficiency is not bounded away from zero and speed-up may not increase. If efficiency is bounded away from zero as n is increased without bound, $S_p(n)$ will continue to increase indicating that the number of processors which can be used to solve the problem can be increased without bound. If efficiency goes to zero, $S_p(n)$ may not increase indicating that adding more processors may not speed up solution and therefore might be a wasteful endeavor. A problem which can be solved in parallel and has efficiency bounded away from zero will be called scalable.

One very important characteristic of parallel computers which effects scalability is communication time. Communication time is the time to send one message, typically a real number, from one processor to another. It is very important to consider communication time when the solution time of a particular problem is of interest. A parallel solution which demands the passing of many messages can quickly become communication bound. A problem which is communication bound spends most of the time communicating and little time computing. Extra time spent in excessive communications can quickly render a problem unscalable. Therefore, a problem will only be considered scalable if it is scalable including communication time.

Finally, consider the parallel architecture which is necessary to run a particular numerical problem. Very often, solution of a particular problem is bound strictly to a particular type of parallel architecture. A problem which can be solved on a variety of parallel architectures will be said to have a flexible parallel implementation.

Problem	Time	Topology
Inner Product	O(log n)	Hypercube w/ p=n processors
Matrix-Vector Multiplication	O(n)	Linear Array w/ p=n processors
Matrix-Matrix Multiplication	O(n)	Mesh w/ $p = n^2$ processors
Q R Factorization	O(n)	Mesh w/ $p = n^2$ processors
Backsubstitution	O(n)	Linear Array w/ p=n processors

Table 2.1: Times for Parallel Matrix Calculations

It is obviously most desirable for a problem to have a flexible parallel implementation since then the problem may be solved on a variety of parallel computers or parallel processing arrays. The goal then is to strive for a parallel implementation which is efficient, flexible, and scalable.

Many simple matrix operations have the desirable properties of efficiency, flexibility, and scalability. For example, matrix multiplication and the solution of systems of linear equations have a variety of implementations both on parallel computers and array processors such as systolic arrays or wavefront arrays and have been shown to be scalable [4, 5]. Table 2.1 (adapted from [4]) gives some examples of other simple matrix operations which are known to have efficient, flexible, and scalable parallel implementations.

The idea then is to break down solution of the optimal control problem into smaller operations which have well defined parallel implementations such as simple matrix operations. In this way, it is possible to form parallel implementations of optimal control problems which may retain the desirable parallel characteristics of the simple matrix operations.

Walsh Functions for Optimal Control

Walsh functions have been used for systems analysis, systems identification, and optimal control [6, 7, 8]. The following is a review of some of their useful properties for use in the parallel solution of optimal control problems.

The Walsh functions are a set of square wave functions which are orthogonal.

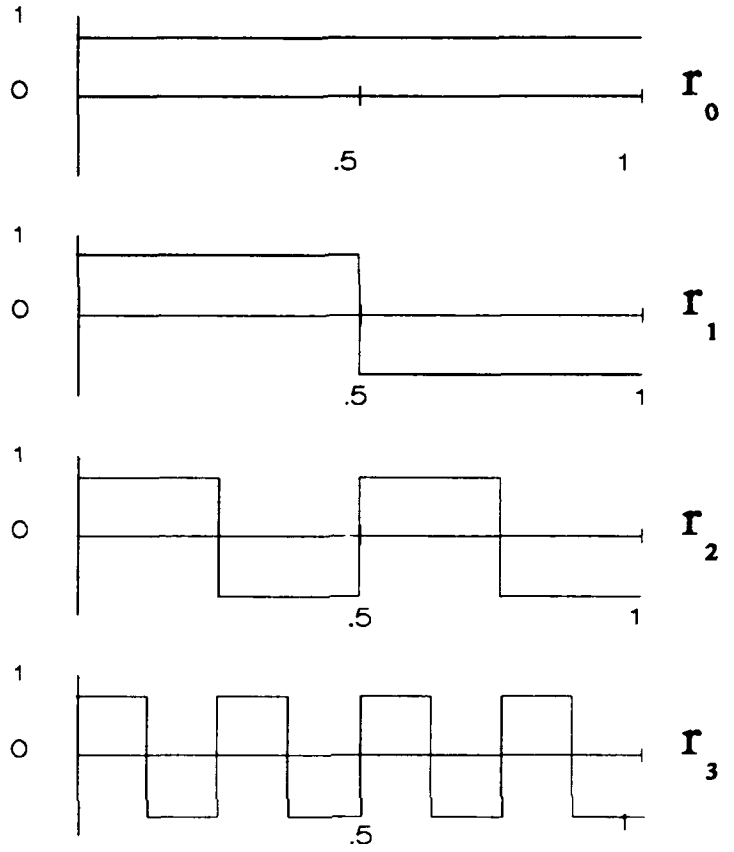


Figure 2.1: Rademacher Functions

The Walsh functions are generated from a more elementary set of square waves known as the Rademacher functions [9]. Rademacher functions are a set of square waves of unit height which have periods of $1, \frac{1}{2}, \frac{1}{4}, \ldots, 2^{1-k}$. The first four Rademacher functions are shown in Figure 2.1. Rademacher functions have odd symmetry about $t = 0$ and $t = \frac{1}{2}$. As a result, the functions are incomplete since it is not possible to approximate a function which has even symmetry about $t = 0$ and $t = \frac{1}{2}$ using Rademacher functions. Walsh combined the Rademacher functions to form a complete, orthogonal set of rectangular waves [10].

The Walsh functions can be generated as the product of n Rademacher functions. Let $a_n \cdots a_2 a_1$ be an n digit binary number, so the a_i's are either 0 or 1

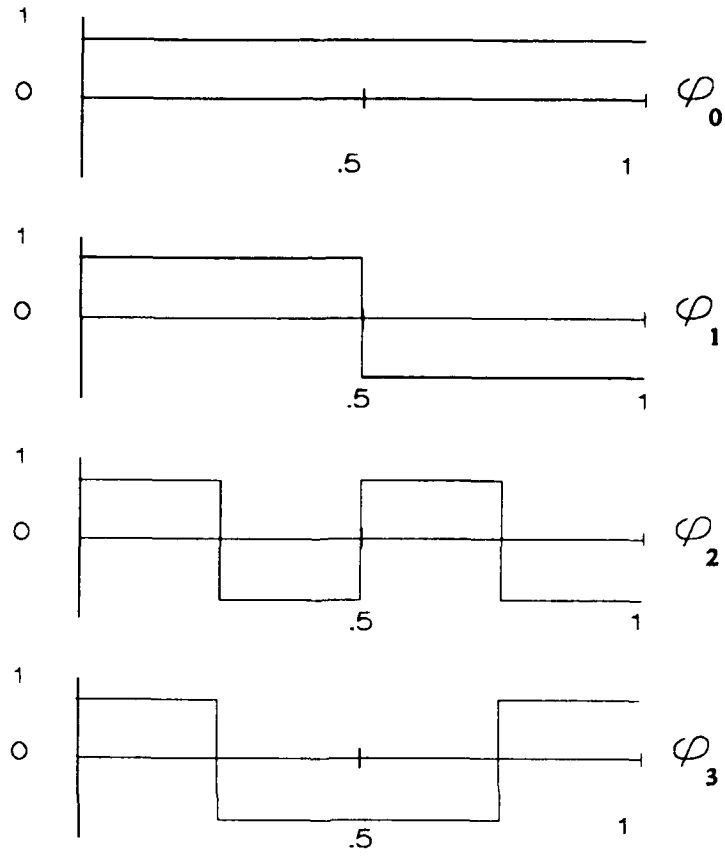


Figure 2.2: Walsh Functions

and a_n is 1. The Walsh functions are obtained by

$$\phi_{a_n \cdots a_2 a_1}(t) = [r_n(t)]^{a_n} \cdots [r_2(t)]^{a_2} [r_1(t)]^{a_1} \tag{2.3}$$

EXAMPLE:

$$\begin{aligned} \phi_0(t) &= r_0(t) & \phi_{100}(t) &= r_3(t) \\ \phi_1(t) &= r_1(t) & \phi_{101}(t) &= r_3(t) r_1(t) \\ \phi_{10}(t) &= r_2(t) & \phi_{110}(t) &= r_3(t) r_2(t) \\ \phi_{11}(t) &= r_2(t) r_1(t) & \phi_{111}(t) &= r_3(t) r_2(t) r_1(t) \end{aligned}$$

Figure 2.2 shows the first four Walsh functions $\phi_0(t)$ to $\phi_3(t)$. Just as a function $f(t)$ may be expanded into a Fourier series, a function $f(t)$ which is integrable in (0,1]

may be expanded into a Walsh series.

$$f(t) = c_0\phi_0(t) + c_1\phi_1(t) + \cdots + c_{N-1}\phi_{N-1}(t) + \cdots \tag{2.4}$$

Walsh functions have very useful integration properties. Integration of a vector of Walsh functions can be performed by a matrix multiplication, specifically [11]:

$$\int_0^t \Psi(\tau)d\tau = P\Psi(t) \tag{2.5}$$

where

$$\Psi(\tau) = \begin{pmatrix} \phi_0(\tau) \\ \phi_1(\tau) \\ \vdots \\ \phi_{N-1}(\tau) \end{pmatrix} \tag{2.6}$$

P is called the integration matrix and has the following general structure:

$$P_1 = \frac{1}{2} \quad P_N = \begin{bmatrix} P_{N/2} & -(\frac{1}{2N})I_{N/2} \\ (\frac{1}{2N})I_{N/2} & 0 \end{bmatrix} \quad \begin{matrix} N = 2^n \\ n = 1, 2, \ldots \end{matrix} \tag{2.7}$$

EXAMPLE: Let $N = 4$ and $f(t) = 1$. Find

$$\int_0^t f(\tau)d\tau$$

Then

$$P = \begin{bmatrix} \frac{1}{2} & -\frac{1}{4} & -\frac{1}{8} & 0 \\ \frac{1}{4} & 0 & 0 & -\frac{1}{8} \\ \frac{1}{8} & 0 & 0 & 0 \\ 0 & \frac{1}{8} & 0 & 0 \end{bmatrix}$$

Let the Walsh approximation for $f(t)$ be

$$f(t) = F\Psi(t)$$

where

$$F = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}$$

then

$$\int_0^t F\Psi(\tau)d\tau = F\int_0^t \Psi(\tau)d\tau = FP\Psi(t)$$

where

$$FP = \begin{bmatrix} \frac{1}{2} & -\frac{1}{4} & -\frac{1}{8} & 0 \end{bmatrix}$$

and it can be shown that $FP\Psi(t)$ is the four term Walsh series approximation for $g(t) = t$.

Walsh functions have interesting multiplication properties. The product of two Walsh functions is found by the mod 2 addition of the binary form of the Walsh function subscripts. Thus the Walsh functions form a closed set under multiplication. The multiplication of two Walsh functions never results in a Walsh function whose subscript is greater than the functions being multiplied.

EXAMPLE:

$$\phi_{10}(t) \times \phi_{11}(t) = \phi_{01}(t)$$

Finally, the Walsh functions form an orthogonal system on $0 \leq t < 1$, or

$$\int_0^1 \phi_i(t)\phi_j(t)dt = \begin{cases} 0 & i \neq j \\ 1 & i = j \end{cases} \tag{2.8}$$

The Epsilon Method

In order to introduce the general mathematical structure for using the epsilon method consider the following general optimal control problem. It is desired to control some system described by:

$$\dot{x}(t) = f(x(t), u(t), t) \qquad x(t_0) = x_0 \tag{2.9}$$

where $x(t) \in R^n, u(t) \in R^m$ and $u(t)$ represents the control. A cost functional V describes some performance measure to be met. In general,

$$V = \int_0^{t_f} G(x(t), u(t), t)dt \tag{2.10}$$

and

$$\psi(x(t_f), t_f) = 0 \tag{2.11}$$

is a terminal constraint on the state which must be met exactly. The problem is to find $u^*(t)$ which minimizes the cost function while meeting the dynamic constraint of (2.9).

A commonly used approach to solve the continuous optimal control problem is to employ the method of Lagrange multipliers [12] to adjoin (2.9) and (2.11) to the cost functional

$$V' = \nu^T \psi(x(t_f), t_f) + \int_0^{t_f} [G(x(t), u(t), t) + \lambda^T(t)\{f(x(t), u(t), t) - \dot{x}(t)\}]dt \quad (? \ 12)$$

Since (2.9) must hold at each $t \in [0, t_f]$ the multiplier $\lambda(t) \in R^n$ is a function of time and is known as the costate. Equation (2.11) holds only at one time and so ν is a constant multiplier.

Now define the Hamiltonian function as

$$H(x(t), u(t), \lambda(t), t) = G(x(t), u(t), t) + \lambda^T(t)[f(x(t), u(t), t)] \quad (2.13)$$

Then

$$V' = \nu^T \psi(x(t_f), t_f) + \int_0^{t_f} [H(x(t), u(t), \lambda(t), t) - \lambda^T(t)\dot{x}(t)]dt \quad (2.14)$$

Now minimizing V subject to the constraint of equation (2.9) is equivalent to the unconstrained optimization problem of minimizing V'. It can be shown that the following conditions must be met to obtain a minimum of V'.

$$\dot{x}(t) = \frac{\partial H(x(t), u(t), \lambda(t), t)}{\partial \lambda(t)} = f(x(t), u(t), t) \quad t \geq 0 \quad (2.15)$$

$$-\dot{\lambda}(t) = \frac{\partial H(x(t), u(t), \lambda(t), t)}{\partial x(t)} \quad t \leq t_f \quad (2.16)$$

$$\frac{\partial H(x(t), u(t), \lambda(t), t)}{\partial u(t)} = 0 \quad (2.17)$$

$$x(0) = x_0$$

$$(\psi_x^T \nu - \lambda(t))^T) \mid_{t_f} dx(t_f) + (\psi_x^T \nu - H(x(t), u(t), \lambda(t), t)) \mid_{t_f} dt_f = 0 \quad (2.18)$$

In (2.18) the subscripts denote partials and the bar means the quantity is evaluated at t_f.

The continuous optimal control problem has been converted into a two point boundary value problem (TPBVP). The state and costate form a system of differential equations in which the initial conditions on the state and the final conditions on the costate are known. The structure of the TPBVP suggests solution methods

which involve numerically solving the state and costate equations many times. An example of this is given in Chapter 6. Since numerical methods for solving ordinary differential equations are very difficult to parallelize, this introduces a serial bottleneck into parallelization of the problem. A method which avoids solving the state equations was formulated by Balakrishnan [13].

Balakrishnan's epsilon method treats the dynamic equations as a penalty equality constraint. More specifically, in the optimal control problem it is desired to minimize some cost function

$$V(x(t), u(t)) = \int_0^{t_f} G(x(t), u(t), t) dt \tag{2.19}$$

while meeting the constraints imposed by some dynamical system

$$\dot{x}(t) = f(x(t), u(t), t)dt, \quad x(0) = x_s, \tag{2.20}$$

where $t \in [0, t_f], t_f < \infty$. By treating the dynamical equations as a penalty constraint, the problem can be reformulated into a nondynamic form:

$$J(x(t), u(t); \varepsilon) = \frac{1}{2\varepsilon} \int_0^{t_f} \| e(t; \varepsilon) \|^2 \, dt + V(x(t), u(t)) \tag{2.21}$$

where

$$e(t; \varepsilon) = \dot{x}(t) - f(x(t), u(t), t) \tag{2.22}$$

The composite cost functional (2.21) is minimized with respect to $x(t)$ and $u(t)$ for a given $\varepsilon > 0$. If necessary, the process is performed with a sequence $\{\varepsilon_j\}$ which decreases monotonically. The process is stopped when the error in the system dynamics is small. This formulation has been called the differential epsilon method. Balakrishnan made a thorough study of the differential epsilon method and gave the conditions where the solution to the epsilon problem approaches the optimal solution to the optimal control problem as $\varepsilon \to 0$ [13].

An integral formulation of the epsilon method was developed by Frick [14] By adjoining an error function described by

$$e(t; \varepsilon) = x(t) - x_s - \int_0^t f(x(\tau), u(\tau), \tau) d\tau \tag{2.23}$$

to the cost function, the composite cost functional is obtained:

$$J(\varepsilon, x(t), u(t)) = \int_0^{t_f} \frac{1}{2\varepsilon} \parallel e(t;\varepsilon) \parallel^2 dt + V(x(t), u(t), t) \tag{2.24}$$

This composite cost function is minimized in the same fashion as the differential epsilon method. Convergence for linear optimal control problems with quadratic performance indices was shown by Frick [14]. These results are summarized next.

Convergence Result:

Consider the sequence of scalars $\{\varepsilon_j\} \downarrow 0$ monotonically decreasing. For the corresponding sequence of minimizing solutions to the integral ε problem denoted by $\{x_o(\varepsilon_j), u_o(\varepsilon_j)\}$ and the associated $\{e_o(\varepsilon_j)\}$, we have:

$$u_o(\varepsilon_j) \rightarrow u^* \quad \text{while} \quad x_o(\varepsilon_j) \rightarrow x^* \quad \text{and} \quad \frac{e_o(\varepsilon_j)}{\varepsilon_j} \rightarrow \lambda^* \tag{2.25}$$

$$e_o(\varepsilon_j) \downarrow 0 \quad \text{while} \quad J(\varepsilon_j, x_o(\varepsilon_j), u_o(\varepsilon_j)) \uparrow V(x^*, u^*) \tag{2.26}$$

as $\varepsilon \downarrow 0$. Here x^*, u^* and $V(x^*, u^*)$ are the optimal control, state, and cost for the case of linear dynamical equations and quadratic cost.

For practical application, the Rayleigh-Ritz method has been used to minimize the composite cost functional of the differential epsilon method [18]. Gradient techniques have been used to minimize the composite cost functional of the integral epsilon method [14]. In the next chapter, the integral epsilon problem for the case of linear time-varying dynamical equations and quadratic cost function is solved using the Rayleigh-Ritz method with Walsh functions as basis functions. As a result of the integration properties of the Walsh functions, the solution of the integral epsilon problem using the Rayleigh-Ritz method is an approach which allows for a high degree of parallelism in the solution.

CHAPTER 3

PARALLEL SOLUTION OF LINEAR OPTIMAL CONTROL PROBLEMS

In this chapter, the epsilon problem for the case of linear time-varying dynamics and quadratic cost is solved by the Ritz method using a Walsh function basis set. The resulting static optimization problem can be solved using matrix operations. A parallel implementation for a linear array embedded in a hypercube computer is then developed.

The Rayleigh-Ritz Method using a Walsh Basis

To minimize a cost functional using the Rayleigh-Ritz method, the time-varying functions in the cost functional are approximated by a series of basis functions. Approximation by a Walsh series will be used. Once the time-varying functions in the cost functional have been replaced by the parameterized functions, the new cost functional can be minimized with respect to the constants of the approximated functions. In this manner minimization of the cost functional is replaced by a static optimization problem which is very amenable to parallel implementation.

It is desired to control some dynamical system which is described by a set of linear time varying differential equations.

$$\dot{x}(t) = A(t)x(t) + B(t)u(t) + C(t), \quad x(0) = x_s, \tag{3.1}$$

with $x(t) \in R^n; u(t) \in R^m$. A performance index or "cost function" is selected to force the dynamical system to follow some desired performance specification. For

example,

$$V(x(t),u(t)) = \int_0^{t_f} \frac{1}{2} < x(t), Qx(t) > + \frac{1}{2} < u(t), Ru(t) > \} dt \tag{3.2}$$

Here $< x(t), Qx(t) >$ denotes the inner product. Q is positive semi-definite and R is positive definite. This is a quadratic cost function. It penalizes deviations from zero in the state and the control, especially if Q and R have large magnitude. The linear time-varying optimal control problem is to find the optimal control $u^*(t)$ which drives the states of the dynamical system along a trajectory $x^*(t)$ such that the cost functional is minimized and the dynamical equations are met. This linear time-varying optimal control problem can be solved using the integral epsilon method and convergence of the method is assured. Consider the integral form of (3.1)

$$x(t) = x_s + \int_0^t A(\tau)x(\tau)d\tau + \int_0^t B(\tau)u(\tau)d\tau + \int_0^t C(\tau)d\tau \tag{3.3}$$

These dynamical equations can be viewed as a dynamical constraint and can be adjoined to the cost functional by forming an error function

$$e(t;\varepsilon) = x(t) - x_s - \int_0^t A(\tau)x(\tau)d\tau - \int_0^t B(\tau)u(\tau)d\tau - \int_0^t C(\tau)d\tau \tag{3.4}$$

Then the composite cost functional can be formed

$$J(\varepsilon, x(t), u(t)) = \int_0^{t_f} \{ \frac{1}{2\varepsilon} \parallel e(t;\varepsilon) \parallel^2 + \frac{1}{2} < x(t), Qx(t) > + \frac{1}{2} < u(t), Ru(t) > \} dt \tag{3.5}$$

The first step is to expand the time varying functions in the composite cost function into a Walsh series. Since for computational application the series must be truncated, the Walsh series represents an approximation to the original function. Once the problem is in approximation form, it is solved by finding the coefficients to the state and control which minimize the composite cost function. The Walsh functions form an orthogonal base on [0,1) so assume (without loss of generality) that $t_f = 1$. Now approximate $x(t), u(t), A(t)$, and $B(t)$ with N Walsh functions where $N = 2^k, k > 0$. For example:

$$x(t) = c_0\phi_0(t) + c_1\phi_1(t) + \cdots + c_{N-1}\phi_{N-1}(t)$$

The following approximations can be made

$$\begin{aligned} x(t) &\approx X\Psi(t) \\ u(t) &\approx U\Psi(t) \\ A(t) &\approx \alpha\Psi(t) \\ B(t) &\approx \beta\Psi(t) \end{aligned} \tag{3.6}$$

where X, U, α, and β are matrices of coefficients and $\Psi(t) = [\phi_0(t), \phi_1(t), \ldots \phi_{N-1}(t)]^T$. Also, the error term in the composite cost functional can be approximated:

$$e(t;\varepsilon) \approx E\Psi(t) \tag{3.7}$$

Equation (3.5) can be written in approximation form:

$$J = \int_0^1 \{\frac{1}{2\varepsilon}\Psi^T(t)E^TE\Psi(t) + \frac{1}{2}\Psi^T(t)X^TQX\Psi(t) + \frac{1}{2}\Psi^T(t)U^TRU\Psi(t)\}dt \tag{3.8}$$

The matrix product terms have the form $\Psi^T(t)M\Psi(T)$ which can be written in summation notation

$$\Psi^T(t)M\Psi(t) = \sum_{i=0}^{N-1}\phi_i(t)\sum_{j=0}^{N-1} m_{ij}\phi_j(t) \tag{3.9}$$

and therefore

$$\int_0^1 \Psi^T(t)M\Psi(t)dt = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} m_{ij}\int_0^1 \phi_i(t)\phi_j(t)dt \tag{3.10}$$

As noted in Chapter 2, for the Walsh functions the above integral is the identity matrix and so

$$\int_0^1 \Psi^T(t)M\Psi(t)dt = \sum_{i=0}^{N-1} m_{ii} = trace(M) \tag{3.11}$$

Using (3.11), (3.8) can be written

$$J = trace\{\frac{1}{2\varepsilon}E^TE + \frac{1}{2}X^TQX + \frac{1}{2}U^TRU\} \tag{3.12}$$

Now it is possible to take advantage of some powerful systems notation standardized by Brewer [15] (See Appendix A). Using *vec* notation (3.12) can be written

$$J = \frac{1}{2\varepsilon}vec(E)^Tvec(E) + \frac{1}{2}vec(X)^Tvec(QX) + \frac{1}{2}vec(U)^Tvec(RU) \tag{3.13}$$

or

$$J = \frac{1}{2\varepsilon} vec(E)^T vec(E) + \frac{1}{2} vec(X)^T (I_N \otimes Q) vec(X) + \frac{1}{2} vec(U)^T (I_N \otimes R) vec(U) \quad (3.14)$$

It is desired to find the Walsh coefficients of the state and control $vec^*(X)$ and $vec^*(U)$ which minimize the above cost function. Since $vec(E)$ is a function of $vec(X)$ and $vec(U)$, it must be found next.

To approximate $e(t;\varepsilon)$ in terms of Walsh functions, the time-varying integrals in (3.4) must be approximated. This can be done using the multiplication and integral properties of Walsh functions. Consider first the scalar case where n=1. Let $A(t) \approx \alpha\Psi(t)$ or $\Psi^T\alpha^T$, α is a vector of coefficients, $\alpha = [\alpha_0, \alpha_1, \cdots \alpha_{N-1}]$ and $x(t) \approx X\Psi(t)$ Then

$$A(t)x(t) \approx X\Psi(t)\Psi^T(t)\alpha^T \quad (3.15)$$

Using the rule given earlier for multiplying Walsh functions, the matrix $\Psi(t)\Psi^T(t)$ can be found. For example if $N = 4$

$$\Psi_{(4)}(t)\Psi^T_{(4)}(t) = \begin{bmatrix} \phi_0(t) & \phi_1(t) & \phi_2(t) & \phi_3(t) \\ \phi_1(t) & \phi_0(t) & \phi_3(t) & \phi_2(t) \\ \phi_2(t) & \phi_3(t) & \phi_0(t) & \phi_1(t) \\ \phi_3(t) & \phi_2(t) & \phi_1(t) & \phi_0(t) \end{bmatrix}$$

Chen [6] gives the general form of the product of these two matrices as

$$\Psi_{(N)}(t)\Psi^T_{(N)}(t) = \begin{bmatrix} \Psi_{(\frac{N}{2})}(t)\Psi^T_{(\frac{N}{2})}(t) & \Psi_{(\frac{N}{2})+\frac{N}{2}}(t)\Psi^T_{(\frac{N}{2})+\frac{N}{2}}(t) \\ \Psi_{(\frac{N}{2})+\frac{N}{2}}(t)\Psi^T_{(\frac{N}{2})+\frac{N}{2}}(t) & \Psi_{(\frac{N}{2})}(t)\Psi^T_{(\frac{N}{2})}(t) \end{bmatrix} \quad (3.16)$$

where the subscript $(\frac{N}{2}) + \frac{N}{2}$ denotes that the subscript of each element is increased by $\frac{N}{2}$. Notice that the elements of this matrix have a single subscript. For matrices of this type, the following is true

$$\Psi(t)\Psi^T(t)\alpha = \Lambda_\alpha \Psi(t) \quad (3.17)$$

Here α is some vector of coefficients and Λ_α is a matrix of the elements of α which

have the same subscripts as $\Psi(t)\Psi^T(t)$. For example if $N = 4$:

$$\begin{bmatrix} \phi_0(t) & \phi_1(t) & \phi_2(t) & \phi_3(t) \\ \phi_1(t) & \phi_0(t) & \phi_3(t) & \phi_2(t) \\ \phi_2(t) & \phi_3(t) & \phi_0(t) & \phi_1(t) \\ \phi_3(t) & \phi_2(t) & \phi_1(t) & \phi_0(t) \end{bmatrix} \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix} = \begin{bmatrix} \alpha_0\phi_0(t) + \alpha_1\phi_1(t) + \alpha_2\phi_2(t) + \alpha_3\phi_3(t) \\ \alpha_1\phi_0(t) + \alpha_0\phi_1(t) + \alpha_3\phi_2(t) + \alpha_2\phi_3(t) \\ \alpha_2\phi_0(t) + \alpha_3\phi_1(t) + \alpha_0\phi_2(t) + \alpha_1\phi_3(t) \\ \alpha_3\phi_0(t) + \alpha_2\phi_1(t) + \alpha_1\phi_2(t) + \alpha_0\phi_3(t) \end{bmatrix}$$

and

$$\begin{bmatrix} \alpha_0\phi_0(t) + \alpha_1\phi_1(t) + \alpha_2\phi_2(t) + \alpha_3\phi_3(t) \\ \alpha_1\phi_0(t) + \alpha_0\phi_1(t) + \alpha_3\phi_2(t) + \alpha_2\phi_3(t) \\ \alpha_2\phi_0(t) + \alpha_3\phi_1(t) + \alpha_0\phi_2(t) + \alpha_1\phi_3(t) \\ \alpha_3\phi_0(t) + \alpha_2\phi_1(t) + \alpha_1\phi_2(t) + \alpha_0\phi_3(t) \end{bmatrix} = \begin{bmatrix} \alpha_0 & \alpha_1 & \alpha_2 & \alpha_3 \\ \alpha_1 & \alpha_0 & \alpha_3 & \alpha_2 \\ \alpha_2 & \alpha_3 & \alpha_0 & \alpha_1 \\ \alpha_3 & \alpha_2 & \alpha_1 & \alpha_0 \end{bmatrix} \begin{bmatrix} \phi_0(t) \\ \phi_1(t) \\ \phi_2(t) \\ \phi_3(t) \end{bmatrix}$$

Now (3.15) can be rewritten

$$A(t)x(t) \approx X\Lambda_\alpha\Psi(t) \tag{3.18}$$

then for the scalar case

$$\int_0^t A(\tau)x(\tau)d\tau \approx X\Lambda_\alpha P\Psi(t) \tag{3.19}$$

and

$$vec(X\Lambda_\alpha P) = [(\Lambda_\alpha P)^T]vec(X) \tag{3.20}$$

Now consider the case where $n > 1$, then

$$A(t) = \begin{bmatrix} a_{11}(t) & a_{12}(t) & \cdots & a_{1n}(t) \\ a_{21}(t) & & & \\ \vdots & & & \vdots \\ a_{n1}(t) & \cdots & & a_{nn}(t) \end{bmatrix}; \quad x(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \\ \vdots \\ x_n(t) \end{bmatrix} \tag{3.21}$$

Using (3.19), the time-varying integral for $n > 1$ can be written as

$$\int_0^t A(\tau)x(\tau)d\tau \approx \begin{bmatrix} X_1\Lambda_{\alpha_{11}}P\Psi(t) + & \cdots & +X_n\Lambda_{\alpha_{1n}}P\Psi(t) \\ \vdots & & \vdots \\ X_1\Lambda_{\alpha_{n1}}P\Psi(t) + & \cdots & +X_n\Lambda_{\alpha_{nn}}P\Psi(t) \end{bmatrix} \tag{3.22}$$

where $\alpha_{11} = [\alpha_{11}^0, \alpha_{11}^1, \cdots \alpha_{11}^{N-1}]$. Here the superscript denotes the Walsh function number so $a_{11}(t)x_1(t) = \alpha_{11}\Psi(t)\Psi^T X_1$. Now $\Psi(t)$ is integrated out of (3.22) as shown in (3.11) when the cost function was approximated with Walsh functions. Equation (3.22) can now be written,

$$[(I_n \otimes P^T)\Lambda'_\alpha] \begin{bmatrix} vec(X_1) \\ vec(X_2) \\ \vdots \\ vec(X_n) \end{bmatrix} \tag{3.23}$$

where

$$\Lambda'_\alpha = \begin{bmatrix} \Lambda_{\alpha_{11}} & \Lambda_{\alpha_{12}} & \cdots & \Lambda_{\alpha_{1n}} \\ \Lambda_{\alpha_{21}} & \Lambda_{\alpha_{22}} & & \\ \vdots & & & \vdots \\ \Lambda_{\alpha_{n1}} & \cdots & & \Lambda_{\alpha_{nn}} \end{bmatrix} \tag{3.24}$$

and this can, in turn, be written in vec notation:

$$vec(X\Lambda_\alpha^{(3dim)}P) = [(P^T \otimes I_n)\Lambda_\alpha^{(2dim)}]vec(X) \tag{3.25}$$

where (if N=4)

$$\Lambda_\alpha = \begin{bmatrix} \Lambda_\alpha^0 & \Lambda_\alpha^1 & \Lambda_\alpha^2 & \Lambda_\alpha^3 \\ \Lambda_\alpha^1 & \Lambda_\alpha^0 & \Lambda_\alpha^3 & \Lambda_\alpha^2 \\ \Lambda_\alpha^2 & \Lambda_\alpha^3 & \Lambda_\alpha^0 & \Lambda_\alpha^1 \\ \Lambda_\alpha^3 & \Lambda_\alpha^2 & \Lambda_\alpha^1 & \Lambda_\alpha^0 \end{bmatrix}$$

Here Λ_α^0 denotes a matrix of the zeroth Walsh coefficients. The order of subscripts of Λ_α is the same as the scalar case. Now $vec(E)$ can be written:

$$\begin{aligned} vec(E) = \; & \{vec(X) - vec(G_s) - [(P^T \otimes I_n)\Lambda_\alpha]vec(X) \\ & -[(P^T \otimes I_n)\Lambda_\beta]vec(U) - (P^T \otimes I_n)vec(C)\} \end{aligned} \tag{3.26}$$

To minimize J the gradient is calculated simultaneously with respect to both $vec(X)$ and $vec(U)$ and set equal to zero.

$$\begin{aligned} \nabla_{vec(X)}J = \; & \tfrac{1}{\epsilon}(I_{Nn} - (P^T \otimes I_n)\Lambda_\alpha)^T \Big\{(I_{Nn} - (P^T \otimes I_n)\Lambda_\alpha)vec(X) \\ & -([P^T \otimes I_n]\Lambda_\beta)vec(U) - vec(G_s)\Big\} + (I_N \otimes Q)vec(X) = \underline{0} \end{aligned} \tag{3.27}$$

$$\begin{aligned}\nabla_{vec(U)}J = \; & -\tfrac{1}{\epsilon}[P^T \otimes I_n]\Lambda_\beta)^T \Big\{(I_{Nn} - (P^T \otimes I_n)\Lambda_\alpha)vec(X) \\ & -([P^T \otimes I_n]\Lambda_\beta)vec(U) - vec(G_s)\Big\} + (I_N \otimes R)vec(U) = \underline{0}\end{aligned} \tag{3.28}$$

The resulting system of equations can be written in very simple form:

$$\begin{bmatrix} K_{11} & K_{12} \\ K_{21} & K_{22} \end{bmatrix} \begin{pmatrix} vec^*(X) \\ vec^*(U) \end{pmatrix} = \begin{pmatrix} D_1 \\ D_2 \end{pmatrix} \tag{3.29}$$

where

$$\begin{aligned} K_{11} = \; & \tfrac{1}{\epsilon}(I_{Nn} - (P^T \otimes I_n)\Lambda_\alpha)^T(I_{Nn} - (P^T \otimes I_n)\Lambda_\alpha) + (I_N \otimes Q) \\ K_{12} = \; & -\tfrac{1}{\epsilon}(I_{Nn} - (P^T \otimes I_n)\Lambda_\alpha)^T([P^T \otimes I_n]\Lambda_\beta) \\ K_{21} = \; & -\tfrac{1}{\epsilon}([P^T \otimes I_n]\Lambda_\beta)^T(I_{Nn} - (P^T \otimes I_n)\Lambda_\alpha) \\ K_{22} = \; & \tfrac{1}{\epsilon}([P^T \otimes I_n]\Lambda_\beta)^T([P^T \otimes I_n]\Lambda_\beta) + (I_N \otimes R) \\ D_1 = \; & \tfrac{1}{\epsilon}(I_{Nn} - (P^T \otimes I_n)\Lambda_\alpha)^T(vec(G_s) + [P^T \otimes I_n]vec(C)) \\ D_2 = \; & -\tfrac{1}{\epsilon}([P^T \otimes I_n]\Lambda_\beta)^T(vec(G_s) + [P^T \otimes I_n]vec(C)) \end{aligned} \tag{3.30}$$

Here the subscripts on the identity matrices indicates their dimensions. K can be written more concisely as follows, if

$$M = \begin{bmatrix} I_{Nn} - (P^T \otimes I_n)\Lambda_\alpha & -(P^T \otimes I_n)\Lambda_\beta \\ 0 & 0 \end{bmatrix} \tag{3.31}$$

and

$$L = \begin{bmatrix} I_N \otimes Q & 0 \\ 0 & I_N \otimes R \end{bmatrix} \tag{3.32}$$

Then

$$K = \frac{1}{\epsilon}M^T M + L \tag{3.33}$$

The linear time varying optimal control problem has been reduced to Kronecker products, matrix multiplications, and the solution to a system of linear algebraic equations.

As mentioned in Chapter 2, these types of operations have desirable properties for parallel implementations. To conclude this section the method is illustrated by a simple numerical example.

EXAMPLE:

Solve the following optimal control problem using the epsilon method and the first 4 Walsh functions as basis functions.

$$\dot{x} = x + u \quad x(0) = 1.0$$

$$V(x,u) = \int_0^1 x^2 + u^2 dt$$

So $N = 4$ and the following is true

$$P = \begin{bmatrix} \frac{1}{2} & -\frac{1}{4} & -\frac{1}{8} & 0 \\ \frac{1}{4} & 0 & 0 & -\frac{1}{8} \\ \frac{1}{8} & 0 & 0 & 0 \\ 0 & \frac{1}{8} & 0 & 0 \end{bmatrix}$$

$$\alpha = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} \quad \beta = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} \quad c = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}$$

$$Q = 1 \quad R = 1 \quad \Lambda_\alpha = I_4 \quad \Lambda_\beta = I_4$$

$$G_s = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Thus the following is true,

$$I_{Nn} - (P^T \otimes I_n)\Lambda_\alpha = I_4 - P^T$$

and

$$-(P^T \otimes I_n)\Lambda_\beta = -P^T$$

Finally,

$$M = \begin{bmatrix} I_4 - P^T & -P^T \\ 0 & 0 \end{bmatrix} =$$

$$\begin{bmatrix} 0.5000 & -0.2500 & -0.1250 & 0.0000 & -0.5000 & -0.2500 & -0.1250 & 0.0000 \\ 0.2500 & 1.0000 & 0.0000 & -0.1250 & 0.2500 & 0.0000 & 0.0000 & -0.1250 \\ 0.1250 & 0.0000 & 1.0000 & 0.0000 & 0.1250 & 0.0000 & 0.0000 & 0.0000 \\ 0.0000 & 0.1250 & 0.0000 & 1.0000 & 0.0000 & 0.1250 & 0.0000 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \end{bmatrix}$$

$$L = \begin{bmatrix} I_4 & 0 \\ 0 & I_4 \end{bmatrix}$$

Now choose epsilon small to ensure small dynamical equation error and calculate K,

$$K = \frac{1}{\varepsilon} M^T M + L = \frac{1}{.001}$$

$$\begin{bmatrix} 0.3291 & 0.1250 & 0.0625 & -0.0313 & -0.1719 & -0.1250 & -0.0625 & -0.0313 \\ 0.1250 & 1.0791 & 0.0313 & 0.0000 & 0.3750 & 0.0781 & 0.0313 & -0.1250 \\ 0.0625 & 0.0313 & 1.0166 & 0.0000 & 0.1875 & 0.0313 & 0.0156 & 0.0000 \\ -0.0313 & 0.0000 & 0.0000 & 1.0166 & -0.0313 & 0.1250 & 0.0000 & 0.0156 \\ -0.1719 & 0.3750 & 0.1875 & -0.0313 & 0.3291 & 0.1250 & 0.0625 & -0.0313 \\ -0.1250 & 0.0781 & 0.0313 & 0.1250 & 0.1250 & 0.0791 & 0.0313 & 0.0000 \\ -0.0625 & 0.0313 & 0.0156 & 0.0000 & 0.0625 & 0.0313 & 0.0166 & 0.0000 \\ -0.0313 & -0.1250 & 0.0000 & 0.0156 & -0.0313 & 0.0000 & 0.0000 & 0.0166 \end{bmatrix}$$

$$D = \begin{bmatrix} I_4 - P^T \\ -P^T \end{bmatrix} G_s = \frac{1}{.001} \begin{bmatrix} .500 \\ -.250 \\ -.125 \\ 0. \\ -.500 \\ -.250 \\ -.125 \\ 0. \end{bmatrix}$$

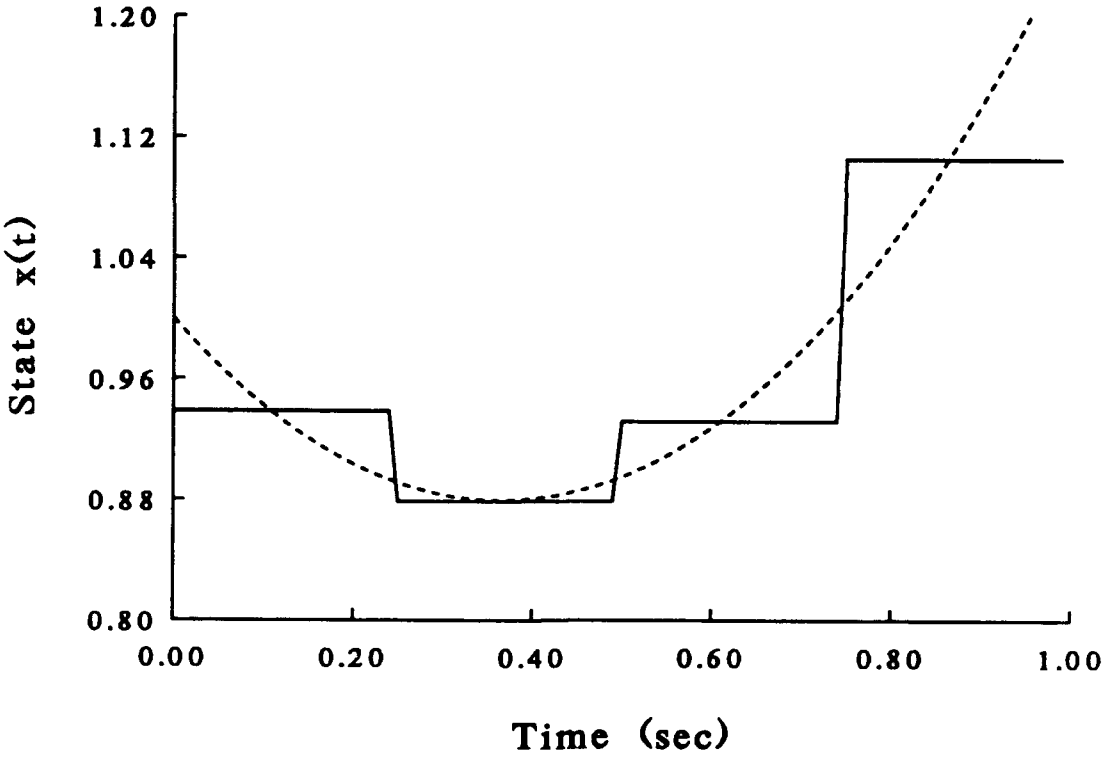


Figure 3.1: Numerical Example - Four Walsh Functions

Solving the system of equations which have been formulated gives

$$vec(X) = \begin{bmatrix} 0.9638 \\ -0.0549 \\ -0.0283 \\ 0.0584 \end{bmatrix} \qquad vec(U) = \begin{bmatrix} -0.7387 \\ -0.4127 \\ -0.2128 \\ -0.0447 \end{bmatrix}$$

The resulting Walsh approximations for the state and control are plotted against the continuous solutions in Figure 3.1.

In the next section, a parallel implementation is given for the solution of optimal control problems on an *Intel Hypercube* with eight processors.

Parallel Implementation

In this section, a specific parallel implementation for solution of optimal control problems on an eight processor *Intel Hypercube* is developed. This is done by first considering parallel solution of each of the necessary matrix operations on a linear array of processors. Once this is done, it is a straightforward task to develop a specific *Hypercube* implementation since a set of processors connected in a linear array is a subset of a set of processors connected in a hypercube configuration.

The matrix operations of interest are Kronecker products, matrix multiplication, and the solution to a set of linear algebraic equations. To solve the set of equations, QR triangularization and backsubstitution are used because of the stable numerical properties of QR triangularization. The serial time to perform a matrix multiplication of two $n \times n$ matrices and the serial time to perform a QR triangularization on an $n \times n$ matrix are the same, $T^*(n^3)$. Serial time for a backsubstitution is $T^*(n^2)$. On a linear array of processors, matrix multiplication and QR triangularization have execution times of $T^*(n^2)$. Since the parallel QR triangularization and matrix multiplication have solution times on the same order as serial backsubstitution, it is not necessary to reduce execution time of the backsubstitution by performing it in parallel. It may be performed serially and not significantly effect overall execution time. The Kronecker product for two $n \times n$ matrices is an $T^*(n^4)$ time operation. The matrices used in the Kronecker products for solution of the optimal control problem are typically sparse. This makes performing the full Kronecker products very inefficient. Consequently, for an efficient implementation on an eight node *Hypercube*, the Kronecker products are done in serial since they are not computationally intensive enough to justify parallel execution. The main computational burden of the problem lies in the matrix multiplication and QR triangularization. Parallel execution of these operations is considered next.

Parallel Matrix Multiplication

It is desired to do some matrix multiplication $AB = C$. Note that B can be partitioned by columns

$$B = [B_1\ B_2 \cdots B_n] \tag{3.34}$$

$B_1, B_2, \ldots B_n$ are the columns of B or groups of columns. The columns of C can be found as the product of A and columns of B.

$$C = [AB_1\ AB_2 \ldots AB_n] \tag{3.35}$$

These matrix multiplications are independent and can be done in parallel. The matrix A and appropriate column or group of columns are passed to a processor which calculates the column or group of columns of C. If B has n columns, then up to n processors can be used.

Parallel Solution to Linear System of Equations

Typical noniterative methods for solving systems of equations $Ax = b$ consist of triangularizing the augmented matrix $[A|b]$ and using backsubstitution to find x. Bojanczyk gave a parallel implementation for QR triangularization on a 2 dimensional mesh of processors [16]. This implementation can be modified for a linear array of processors.

Assume

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1j} & \cdots & a_{1p} & \cdots \\ a_{21} & a_{22} & \cdots & a_{2j} & \cdots & a_{2p} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{i1} & a_{i2} & \cdots & a_{ij} & \cdots & a_{ip} & \cdots \\ a_{i+1,1} & a_{i+1,2} & \cdots & a_{i+1,j} & \cdots & a_{i+1,p} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

and is nonsingular. An orthogonal matrix Q will rotate A to a triangular matrix R. Q is formed as the product of plane rotations, each which zero out an entry in A. So

$$QA = R \tag{3.36}$$

The algorithm can be written concisely as

$$b_{i,j} = (a_{i,j}^2 + a_{i+1,j}^2)^{\frac{1}{2}} \tag{3.37}$$

$$c_i = \frac{a_{i,j}}{b_{i,j}} \; ; \; s_i = \frac{a_{i+1,j}}{b_{i,j}} \tag{3.38}$$

Rows i and $i+1$ are the result of a plane rotation and are given by

$$\begin{aligned} a_{i,p} &= \quad c_i a_{i,p} + s_i a_{i+1,p} \\ a_{i+1,j} &= \; 0 \\ a_{i+1,p} &= \; -s_i a_{i,p} + c_i a_{i+1,p} \end{aligned} \tag{3.39}$$

for $p \neq j$. The above is used iteratively until A is reduced to triangular form. From the above algorithm a data dependence graph can be constructed. This is shown in Figure 3.2. The arcs indicate which way data must flow in order to have the data necessary to calculate the entry in each node. This implies the QR triangularization could be performed on a two dimensional mesh of processors. In this case the entries in each node would be calculated by a processor. This is the implementation given by Bojanczyk. Notice in Figure 3.2 that if a column of the nodes is assigned to a processor, then the algorithm can be performed by a linear array of processors, where s_i and c_i are the data which must be passed between processors. Since for the linear array implementation both the matrix multiplication and the QR triangularization operate on columns of the matrix, they can be joined together very efficiently. If the number of columns each operate on are chosen the same, then all the data needed for the QR factorization is resident in the node after the multiplication. No communication needs to take place. Next the detailed implementation for an *Intel Hypercube* is given.

Intel Hypercube Implementation

A set of processors connected in a hypercube configuration can be formed into a linear array. The particular parallel computer of interest is an *Intel Hypercube*

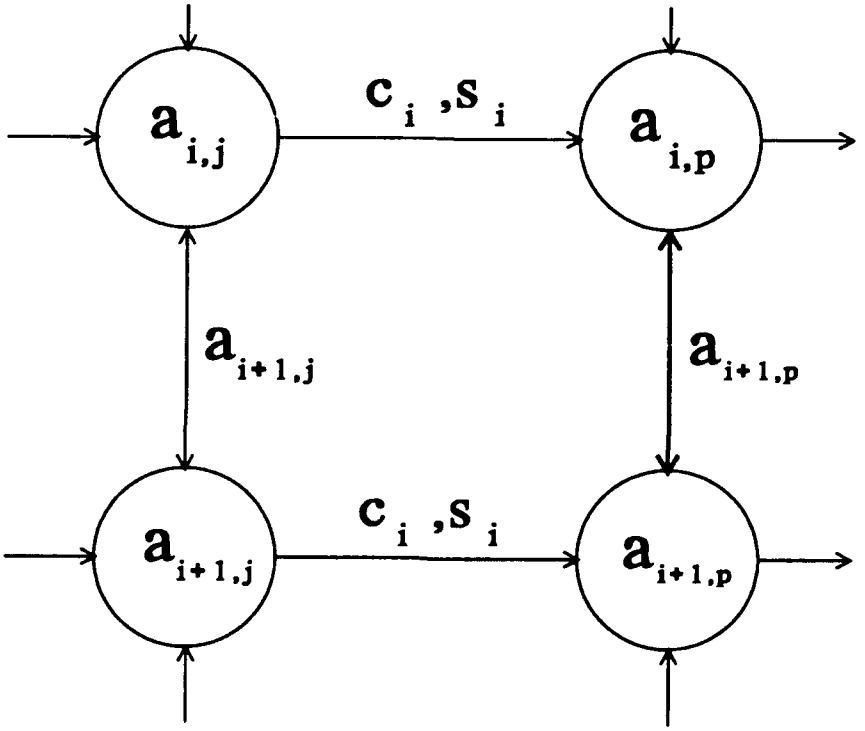


Figure 3.2: Data Dependencies for QR Triangularization

with eight processors. The *Intel Hypercube* is a multiple instruction multiple data computer. In this architecture each processor has its own memory and is connected to its nearest neighbor by a communication line. The example problems to be considered consist of two states and one control. Eight Walsh functions will be used to approximate the state and control. This gives an M matrix which is 24×24 and an augmented matrix $[K|D]$ which is 24×25 on which to perform a QR triangularization. Now solution on the eight node *Hypercube* is as follows:

- Form the matrix M by

$$M = \begin{bmatrix} I_{Nn} - (P^T \otimes I_n)\Lambda_\alpha & -(P^T \otimes I_n)\Lambda_\beta \\ 0 & 0 \end{bmatrix}$$

- M is sent to each processor and each processor calculates three different columns of $M^T M$ and adds the appropriate piece of L.

- The d vector is calculated by

$$D_1 = \frac{1}{\varepsilon}(I_{Nn} - (P^T \otimes I_n)\Lambda_\alpha)^T(vec(G_s) + [P^T \otimes I_n]vec(C))$$

$$D_2 = -\frac{1}{\varepsilon}([P^T \otimes I_n]\Lambda_\beta)^T(vec(G_s) + [P^T \otimes I_n]vec(C))$$

  which are matrix vector multiplies. Matrix vector multiplies require only $T^*(n^2)$ and therefore are performed serially.

- D is augmented to K to form $[K|D]$.

- QR triangularization and backsubstitution are performed on $[K|D]$ yielding the Walsh coefficients.

Figure 3.3 summarizes the implementation

The given parallel implementation is probably not the fastest or the most efficient possible implementation. Because of the nature of triangularization, it is difficult to place the computational load evenly across the linear array. The processors which compute less zero elements do the most work. The implementation does however show how the method can be matched to a particular parallel architecture

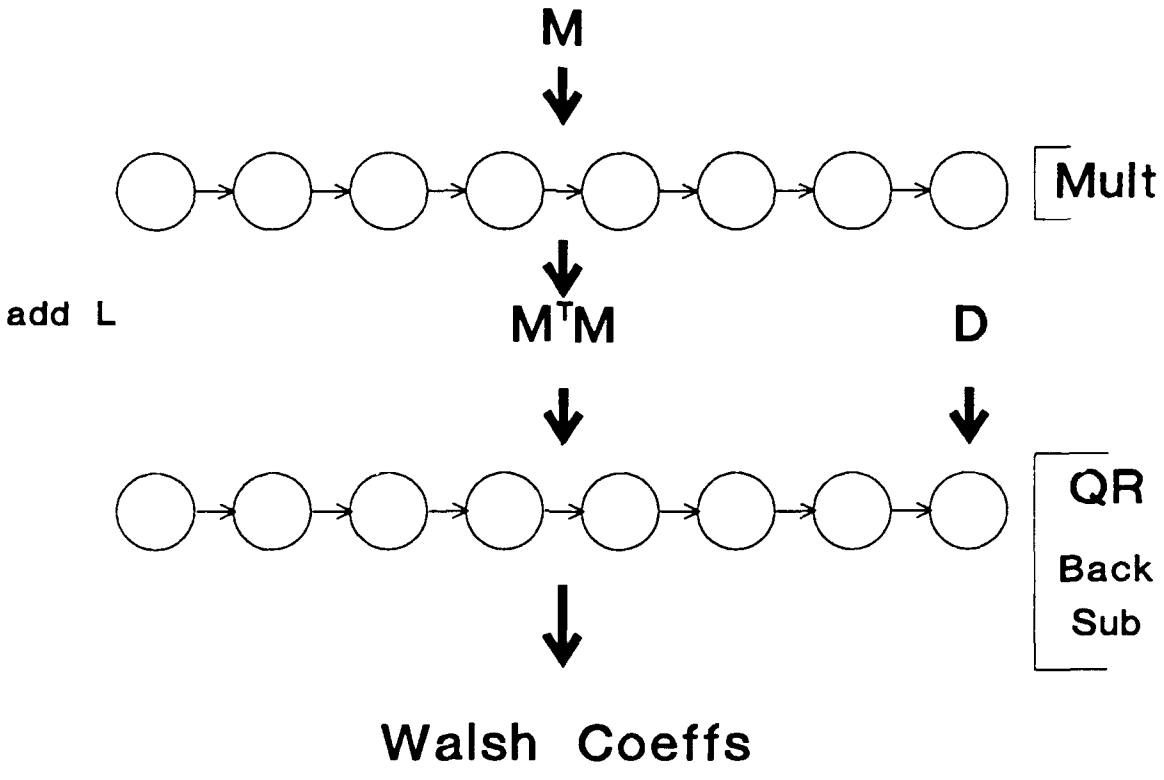


Figure 3.3: Parallel Implementation on a Linear Array

or number of processors. It also shows parallel implementation can be very straightforward when the problem has been broken down into simple matrix operations.

Although a parallel implementation for a linear array of eight processors has been given, the potential parallelizability of the problem should be pointed out. Both matrix multiplication and QR triangularization may be implemented on a two dimensional mesh of processors. For the problems outlined above, up to 576 processors may be used for the matrix multiplication and 300 for the QR triangularization. A larger parallel computer which has comparable communication vs computation time would offer significant speed-up over the linear array. The next section gives results for a numerical example.

Computational Results

The following linear optimal control problem was implemented on the eight node *Intel Hypercube*,

PROBLEM 3.1:

$$\begin{aligned} \dot{x}_1 &= x_2 & x_1(0) &= -4 \\ \dot{x}_2 &= 2x_1 - x_2 + u & x_2(0) &= 4 \end{aligned} \tag{3.40}$$

$$J = \int_0^5 (2x_1^2 + x_2^2 + 0.5u^2)dt \tag{3.41}$$

A Walsh series using eight Walsh functions is used to approximate the state and control. Since the problem is linear, the cost functional of (3.14) is quadratic and can be minimized in one iteration. The error in the dynamic constraint penalty term can be made small by setting epsilon to 0.0001. The plots in Figure 3.4 show the Walsh function approximations against the continuous solutions. Solution time on one node was 2.5 seconds while on all eight solution time was 0.7 seconds for a speed-up of 3.5.

The linear optimal control problem has been solved in many ways. It is not known for being computationally intensive. Solutions to nonlinear optimal control problems on the other hand are considerably more complex and computationally intensive. A goal of this research is to solve such problems in a highly parallel fashion, paving the way for greatly reduced solution times on massively parallel computers or

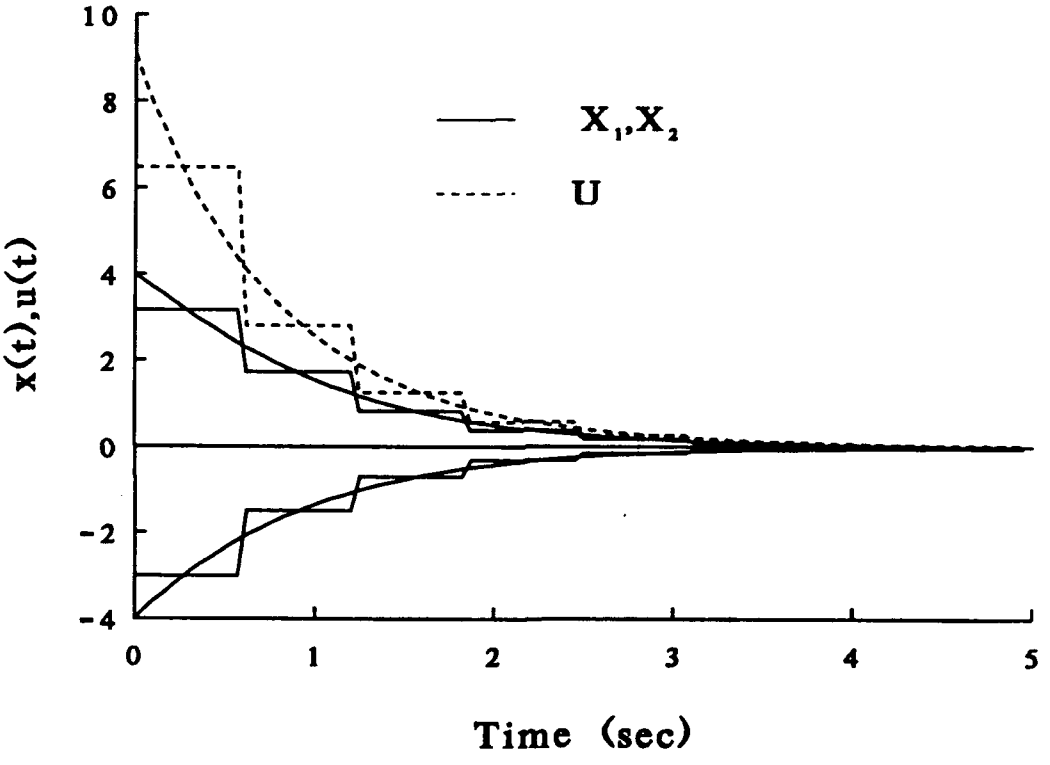


Figure 3.4: Linear Problem Solution

parallel processing arrays. The parallel solution of the nonlinear optimal control problem is the subject of the next chapter.

CHAPTER 4

PARALLEL SOLUTION OF NONLINEAR OPTIMAL CONTROL PROBLEMS

This chapter shows how the solution to the linear time-varying problem can be used to solve nonlinear problems. Two specific types of nonlinear problems are addressed: nonlinear dynamic equations with quadratic cost and nonlinear dynamic equations with quadratic cost and terminal constraints on the states.

Optimal Control Problems with Quadratic Cost

The method used to solve the linear time varying optimal control problem can be extended to nonlinear problems. Quasilinearization is one approach which can be used [17]. For example, consider the following nonlinear problem

$$\dot{x}(t) = f(x(t), u(t), t) \qquad x(0) = x_s \tag{4.1}$$

$$V(x(t), u(t)) = \int_0^{t_f} \{< x(t), Qx(t) > + < u(t), Ru(t) >\} dt \tag{4.2}$$

where $x(t) \in R^n, u(t) \in R^m$. Let $x^*(t)$ and $u^*(t)$ denote the optimal state and control. Now choose some specific $x(t)$ and $u(t)$ which are close to the optimal state and control. These nominal values of the state and control will be called $x_0(t)$ and $u_0(t)$. Since $x^*(t)$ and $u^*(t)$ and $x_0(t)$ and $u_0(t)$ are close together, then the difference between them can be modeled by local linearization. For example let

$$\delta x(t) = x^*(t) - x_0(t) \tag{4.3}$$

Now local linearization is achieved by performing a Taylor series expansion of (4.1) and keeping only first order terms.

$$\delta\dot{x}(t) = \nabla_x f(x_0(t), u_0(t), t)\ \delta x(t) + \nabla_u f(x_0(t), u_0(t), t)\ \delta u(t) \tag{4.4}$$

So $\delta x(t)$ and $\delta u(t)$ can be viewed as some perturbation about a nominal. Now use (4.3) with (4.4) and get

$$\begin{aligned}(\dot{x}(t) - \dot{x}_0(t)) = \ & \nabla_x f(x_0(t), u_0(t), t)(x(t) - x_0(t)) \\ & + \nabla_u f(x_0(t), u_0(t), t)(u(t) - u_0(t))\end{aligned} \tag{4.5}$$

or multiplying out

$$\begin{aligned}\dot{x}(t) = \ & \nabla_x f(x_0(t), u_0(t), t)\ x(t) + \nabla_u f(x_0(t), u_0(t), t)\ u(t) \\ & + (-\dot{x}_0(t) - x_0(t) - u_0(t))\end{aligned} \tag{4.6}$$

which is of the form

$$\dot{x}(t) = A(t)x(t) + B(t)u(t) + C(t) \tag{4.7}$$

where

$$\begin{aligned}A(t) = \ & \nabla_x f(x_0(t), u_0(t), t) \\ B(t) = \ & \nabla_u f(x_0(t), u_0(t), t) \\ C(t) = \ & -f(x_0(t), u_0(t), t) - \nabla_x f(x_0(t), u_0(t), t)\ x_0(t) \\ & -\nabla_u f(x_0(t), u_0(t), t)\ u_0(t)\end{aligned} \tag{4.8}$$

This is the form of the linear time-varying optimal control problem solved in Chapter 3. Notice that if the local linearization accurately modeled the difference between the optimal and nominal, if the system was linear, then the solution could be found in one step. For nonlinear systems, however, the higher order terms are ignored. As a result, $\delta x(t)$ does not give the change to the optimal, but rather the change to move closer to the optimal. So once the linearized problem is solved it is possible to move $\delta x(t)$ closer to the optimal solution by letting

$$x_0^{k+1}(t) = x^k(t) \quad k = 1, 2, 3, ... \tag{4.9}$$

As the optimal solution is approached, then obviously $\delta x(t) \to 0$. In this way it is possible to solve the nonlinear optimal control problem by solving a series of linear

time-varying optimal control problems. The explicit quasilinearization approach is given as follows. Minimize the following

$$J(\varepsilon, x^k(t), u^k(t)) = \int_0^{t_f} \{\frac{1}{2\varepsilon} \parallel e^k(t;\varepsilon) \parallel^2 + \frac{1}{2} < x^k(t), Qx^k(t) > + \frac{1}{2} < u^k(t), Ru^k(t) > \} dt \tag{4.10}$$

for each k = 1,2,3,..... where

$$e^k(t;\varepsilon) = x^k(t) - x_s^k - \int_0^t A^k(\tau)x^k(\tau)d\tau - \int_0^t B^k(\tau)u^k(\tau)d\tau - \int_0^t C^k(\tau)d\tau \tag{4.11}$$

Since the problem is linearized, $A(t)$, $B(t)$, and $C(t)$ are given in equation (4.7). After each k let

$$x_0^{k+1}(t) = x^k(t) \tag{4.12}$$

At each iteration, $\delta x(t)$ and $\delta u(t)$ become smaller indicating that the series of linear solutions is converging to the nonlinear solution. For computational purposes the problem has converged when $\parallel \delta x(t) \parallel < \epsilon$ and $\parallel \delta u(t) \parallel < \epsilon$ where epsilon is some small number.

So the nonlinear optimal control problem can be solved by solution of a series of linear time varying optimal control problems, each of which can be solved in parallel. Computational results are given in the next section.

Computational Results

The Raleigh servo problem was chosen to test the method. This problem is known to cause numerical instabilities when solving the dynamical equations. Thus the Raleigh problem is very difficult to solve using standard methods. An example of the problem is given as follows:

PROBLEM 4.1:

$$\begin{aligned} \dot{x_1} &= x_2 & x_1(0) &= -5 \\ \dot{x_2} &= -x_1 + 1.4\ x_2 - 0.14\ x_2^3 + 4\ u & x_2(0) &= -5 \end{aligned} \tag{4.13}$$

$$V = \int_0^{0.5} (x_1^2 + u^2)dt \tag{4.14}$$

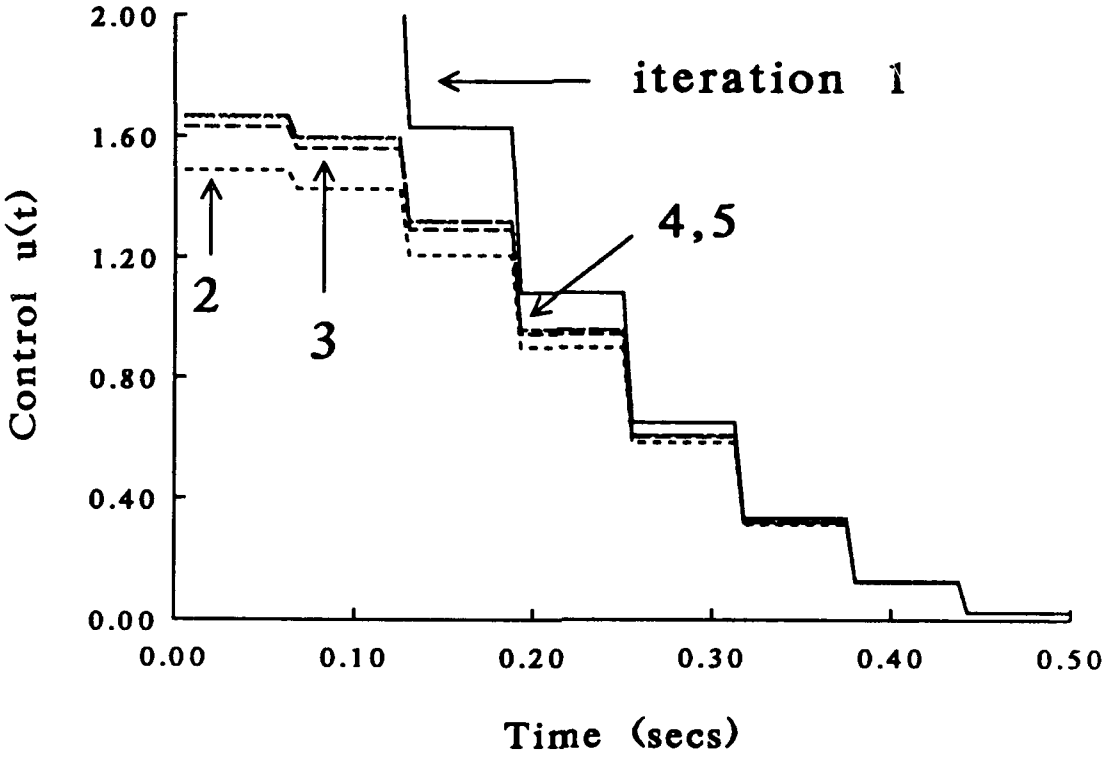


Figure 4.1: Raleigh Problem Solution with Eight Walsh Functions

To solve the problem, eight Walsh functions were used resulting in a parallel implementation as given in Chapter 3. The initial nominal trajectories were set to 1. Epsilon was set to 0.0001. After 5 iterations, $\| \delta x(t) \|_{max} < 0.0055$ and the algorithm was considered to have converged. The cost was within .3 % of the optimal cost of $V^* = 17.00$. Figure 4.1 shows the control for the first five iterations. Table 4.1 shows the error function $e(t;\varepsilon)$ and the cost function J for each iteration. Figure 4.2 shows the Walsh approximation of the optimal control plotted against a gradient based solution to the epsilon problem.

As in the linear case, solution times are of interest. The solution time on one node was 13.6 seconds. On all eight nodes solution time was about 3.9 seconds, for a speed-up of 3.5. This speed-up is expected since the solution to the nonlinear problem is just a series of linear problems. The next section examines how to incorporate

Iteration	$\| \delta x(t) \|_{max}$	$\frac{\|e(t;\varepsilon)\|^2}{\varepsilon}$	$J(\varepsilon, x(t), u(t))$
1	3.7722	0.0009	18.1386
2	0.2845	0.0008	16.9811
3	0.0897	0.0008	16.9585
4	0.0181	0.0008	16.9574
5	0.0034	0.0008	16.9574

Table 4.1: Iterations 1-5 for the Raleigh Solution with Eight Walsh Functions

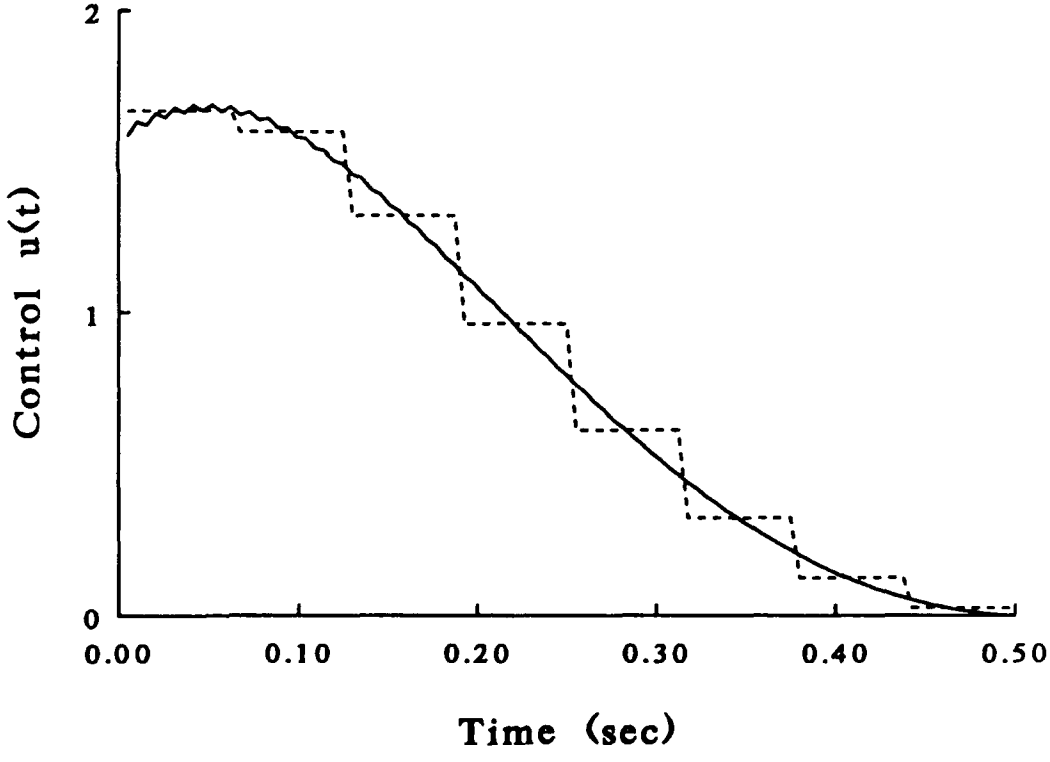


Figure 4.2: Raleigh Problem Solution by Rayleigh-Ritz Method vs Gradient Solution

terminal constraint on the state into the given parallel solution.

Optimal Control Problems with Terminal Constraints

The epsilon method can be used to solve a variety of optimal control problems [18, 14]. The last section addressed one specific type of nonlinear problem. The Raleigh problem has nonlinear state dynamics and quadratic cost. In this type of problem the state is free to take on any value at the final time. Some problems may demand that the state take on a specific value at the final time. This section shows how the epsilon method may be used to solve other forms of optimal control problems, specifically, those with terminal constraints on the state. As before, the solution has a highly parallel implementation. Finally, to show the effectiveness of the method the well known van der Pol problem is solved.

As shown in Frick [14], the optimal control problem can be forced to meet some final state constraint by adding another penalty term which forces the final state to some desired value:

$$J(\varepsilon, x(t), u(t)) = \frac{1}{2\varepsilon}\int_0^{t_f} \| e(t;\varepsilon) \|^2 \, dt + \| \rho(\varepsilon) \|^2 + V(x(t), u(t)) \tag{4.15}$$

As before,

$$e(t;\varepsilon) = x(t) - x_s - \int_0^t f(x(\tau), u(\tau), \tau) d\tau \tag{4.16}$$

and

$$V(x(t), u(t)) = \int_0^{t_f} \{< x(t), Qx(t) > + < u(t), Ru(t) >\} dt \tag{4.17}$$

Approximation of the composite cost functional is simpler if the new penalty term is brought inside the integral. Now as a quadratic term it is simplified similar to the dynamical constraint penalty term. The new penalty term must force the state to some desired final condition so let

$$\rho(\varepsilon) = x(t_f) - x_f \tag{4.18}$$

Here x_f is the desired final state. Since

$$x(t_f) = x_s + \int_0^{t_f} f(x(t), u(t), t) dt \tag{4.19}$$

then

$$\rho(\varepsilon) = x_s + \int_0^{t_f} f(x(t), u(t), t)dt - x_f \tag{4.20}$$

Now consider the case of linear time varying state dynamics. Then

$$\rho(\varepsilon) = x_s + \int_0^{t_f} A(t)x(t)dt + \int_0^{t_f} B(t)u(t)dt + \int_0^{t_f} C(t)dt - x_f \tag{4.21}$$

Note that this is a constant vector that can be approximated by

$$\rho(\varepsilon) = \xi\Psi(t) \tag{4.22}$$

Since the vector to be found is constant the Walsh approximation will be exact and only the coefficients of the zeroth Walsh function will be non-zero. The cost function J must be put in *vec* form so $vec(\xi)$ must be found.

Consider first the scalar case. Then

$$\int_0^{t_f} A(t)x(t)dt \approx \int_0^1 \alpha\Psi(t)\Psi^T(t)X^T dt \tag{4.23}$$

Using the orthogonal property of Walsh functions this can be written

$$\int_0^1 \alpha\Psi(t)\Psi^T(t)X^T dt = \alpha X^T \tag{4.24}$$

For the case where $n > 1$

$$\int_0^1 A(t)x(t)dt = \begin{bmatrix} \int_0^1 a_{11}(t)x_1(t)dt + \cdots + \int_0^1 a_{1n}(t)x_n(t)dt \\ \vdots \\ \int_0^1 a_{n1}(t)x_1(t)dt + \cdots + \int_0^1 a_{nn}(t)x_n(t)dt \end{bmatrix} \tag{4.25}$$

The above matrix can be written

$$\begin{bmatrix} \alpha_{11} & \cdots & \alpha_{1n} \\ \vdots & & \vdots \\ \alpha_{n1} & \cdots & \alpha_{nn} \end{bmatrix} \begin{bmatrix} X_1^T \\ \vdots \\ X_n^T \end{bmatrix} \tag{4.26}$$

Here the X vector is not in standard vec form, but it can be rearranged into standard

vec form

$$\begin{bmatrix} \left[vec \begin{pmatrix} \alpha_{11} \\ \alpha_{12} \\ \vdots \\ \alpha_{1n} \end{pmatrix} \right]^T \\ \left[vec \begin{pmatrix} \alpha_{21} \\ \alpha_{22} \\ \vdots \\ \alpha_{2n} \end{pmatrix} \right]^T \\ \vdots \\ \left[vec \begin{pmatrix} \alpha_{n1} \\ \alpha_{n2} \\ \vdots \\ \alpha_{nn} \end{pmatrix} \right]^T \end{bmatrix} vec(X) \tag{4.27}$$

This matrix vector multiply gives the coefficients of the zeroth Walsh function, the rest are zero. To get a matrix of coefficients for the entire Walsh vector, a matrix Γ_α is formed. The first n rows are obtained from (4.26) and the rest of the matrix is zero. Using Γ_α it is possible to write

$$\int_0^1 \alpha \Psi(t) \Psi^T(t) X dt = \Gamma_\alpha vec(X) \tag{4.28}$$

The same approach can be used to approximate the integral with $B(t)u(t)$. For the last integral a slightly different approach is used. Let

$$C(t) = C\Psi(t) \tag{4.29}$$

Then

$$\int_0^1 C\Psi(t)dt = C \int_0^1 \Psi(t)dt \tag{4.30}$$

The above integral is a vector with the first element being a one and the remaining N are zero. The integral can be written in vec form

$$\int_0^1 C\Psi(t)dt = \Gamma_c vec(C) \tag{4.31}$$

Now $vec(\xi)$ can be written

$$vec(\xi) = vec(G_s) + \Gamma_\alpha vec(X) + \Gamma_\beta vec(U) + \Gamma_c vec(C) - vec(G_f) \tag{4.32}$$

To get the system of equations to be solved, the cost function is minimized as before with the additional penalty term included. The result is to add another term to the K and D matrix.

$$\begin{aligned}
K_{11} &= \tfrac{1}{\epsilon}(I_{Nn} - (P^T \otimes I_n)\Lambda_\alpha)^T(I_{Nn} - (P^T \otimes I_n)\Lambda_\alpha) + (I_N \otimes Q) + \tfrac{1}{\epsilon}\Gamma_\alpha^T\Gamma_\alpha \\
K_{12} &= -\tfrac{1}{\epsilon}(I_{Nn} - (P^T \otimes I_n)\Lambda_\alpha)^T([P^T \otimes I_n]\Lambda_\beta) + \tfrac{1}{\epsilon}\Gamma_\alpha^T\Gamma_\beta \\
K_{21} &= -\tfrac{1}{\epsilon}([P^T \otimes I_n]\Lambda_\beta)^T(I_{Nn} - (P^T \otimes I_n)\Lambda_\alpha) + \tfrac{1}{\epsilon}\Gamma_\beta^T\Gamma_\alpha \\
K_{22} &= \tfrac{1}{\epsilon}([P^T \otimes I_n]\Lambda_\beta)^T([P^T \otimes I_n]\Lambda_\beta) + (I_N \otimes R) + \tfrac{1}{\epsilon}\Gamma_\beta^T\Gamma_\beta
\end{aligned} \tag{4.33}$$

$$\begin{aligned}
D_1 &= \tfrac{1}{\epsilon}(I_{Nn} - (P^T \otimes I_n)\Lambda_\alpha)^T(vec(G_s) + [P^T \otimes I_n]vec(C)) \\
&\quad + \tfrac{1}{\epsilon}\Gamma_\alpha^T\{G_f - G_s - \Gamma_c vec(C)\} \\
D_2 &= -\tfrac{1}{\epsilon}([P^T \otimes I_n]\Lambda_\beta)^T(vec(G_s) + [P^T \otimes I_n]vec(C)) \\
&\quad + \tfrac{1}{\epsilon}\Gamma_\beta^T\{G_f - G_s - \Gamma_c vec(C)\}
\end{aligned} \tag{4.34}$$

As done previously, the above can be written more concisely as

$$M = \begin{bmatrix} I_{Nn} - (P^T \otimes I_n)\Lambda_\alpha & -(P^T \otimes I_n)\Lambda_\beta \\ 0 & 0 \end{bmatrix} \tag{4.35}$$

$$L = \begin{bmatrix} I_N \otimes Q & 0 \\ 0 & I_N \otimes R \end{bmatrix} \tag{4.36}$$

$$F = \begin{bmatrix} \Gamma_\alpha & \Gamma_\beta \\ 0 & 0 \end{bmatrix} \tag{4.37}$$

Then

$$K = \frac{1}{\epsilon}M^T M + L + \frac{1}{\epsilon}F^T F \tag{4.38}$$

The extra term only slightly affects the computational load of the problem since F is made up of predominantly zero elements. Since the additional operations are simple matrix types, the parallelism of the solution is not effected.

Computational Results

Now a problem with van der Pol oscillator dynamics is solved on the eight processor *Hypercube*. The problem is similar to the Raleigh problem in that it has a reputation for being difficult to solve as a result of numerical instabilities encountered when solving the dynamical equations.

PROBLEM 4.2:

$$\begin{aligned} \dot{x_1} &= x_2 & x_1(0) &= 1 \quad x_1(5) = -.97 \\ \dot{x_2} &= -x_1 + x_2 - x_1^2 x_2 + u & x_2(0) &= 0 \quad x_2(5) = -.96 \end{aligned} \tag{4.39}$$

$$V = \int_0^5 (x_1^2 + x_2^2 + u^2) dt \tag{4.40}$$

Since the gamma matrices are nearly all zero, they can be generated quickly and it is not really necessary to do a full matrix-matrix multiplication. The additional term in generating the D vector can also be found quickly as a result of the sparse gamma matrices. The parallel implementation is basically the same as given previously, except F is added when creating K after the parallel matrix multiply and an extra term is added to D before it is used in solution of the system of equations.

To be able to compare results to previous solutions, eight Walsh functions were used to solve the problem. Epsilon was set to 0.0001 and the nominal trajectories were set to unity. As in the Raleigh problem, after about five iterations the method had converged. Table 4.2 gives statistics for the first five iterations. Figure 4.3 shows the resulting approximated optimal state and control. As expected, solution time was also similar to the Raleigh problem. Solution time on one node averaged 14.2 seconds, on eight nodes it averaged 4.1 seconds, for a speed-up of around 3.4.

The next chapter examines other orthogonal functions which may be used effectively to solve the epsilon problem. The motivation is to find a set of basis

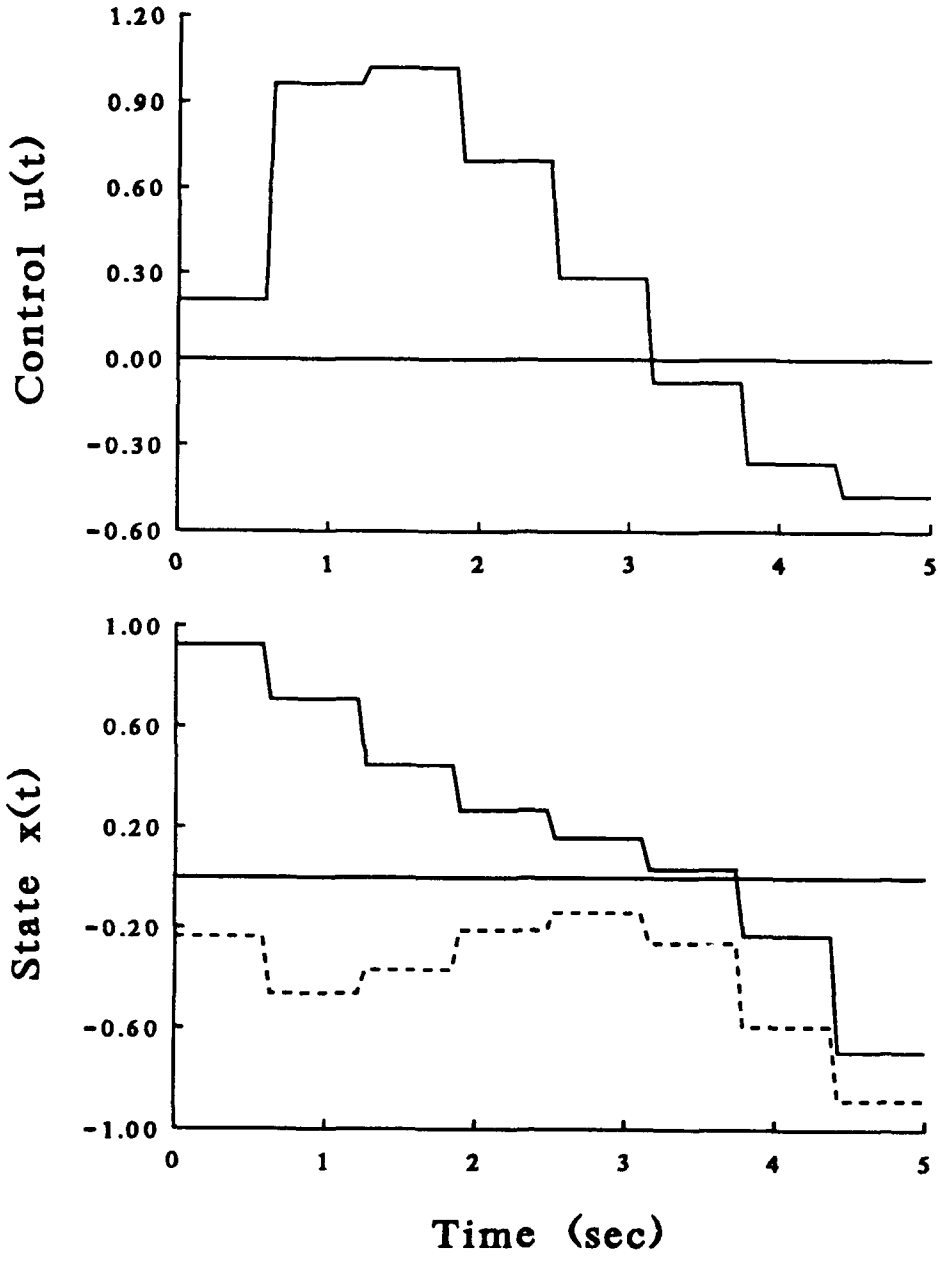


Figure 4.3: van der Pol Problem Solution with Eight Walsh Functions

Iteration	$\| \delta x(t) \|_{max}$	$\frac{\|e(t;\varepsilon)\|^2}{\varepsilon}$	$\frac{\|\rho(\varepsilon)\|^2}{\varepsilon}$	$J(\varepsilon, x(t), u(t))$
1	1.7326	0.0307	0.0034	12.5862
2	0.9912	0.0025	0.0001	3.4257
3	0.2668	0.0040	0.0002	4.2447
4	0.0057	0.0041	0.0002	4.2490
5	0.0004	0.0041	0.0002	4.2490

Table 4.2: Iterations 1-5 for the van der Pol Solution with Eight Walsh Functions

functions which might give closer, smoother approximations to the optimal while using less functions. This would mean a closer approximation with less computation.

CHAPTER 5

PARALLEL SOLUTION OF NONLINEAR OPTIMAL CONTROL PROBLEMS USING OTHER BASIS FUNCTIONS

Walsh functions have been applied to many areas of science and engineering. Of more specific interest here, they have been applied to problems of identification, analysis, and control. More recently, orthogonal polynomials have also been employed in optimal control methods [19, 20, 21, 22]. Orthogonal polynomials have the same integration properties as Walsh functions and may be used effectively as basis functions to solve the epsilon problem when using the Ritz method.

In general, orthogonal polynomials have the following property,

$$\int_a^b w(t)\phi_i(t)\phi_j(t)dt = \begin{cases} 0, & i \neq j \\ \| \phi_i \|, & i = j \end{cases} \tag{5.1}$$

A polynomial is orthogonal with respect to some weight function $w(t)$. The weight function for Legendre polynomials is unity, similar to Walsh functions. In the following section some specific properties of shifted Legendre polynomials are reviewed.

Legendre Polynomials

Legendre polynomials are defined on the interval $[-1, 1]$. For practical application the polynomials are shifted to some interval of interest, $t \in [0, t_f]$. The shifted Legendre polynomials are given by the recurrence relationship [20]

$$(i+1)s_{i+1}(t) = (2i+1)(\frac{2t}{t_f} - 1)s_i(t) - is_{i-1}(t) \quad i = 1, 2, 3, ... \tag{5.2}$$

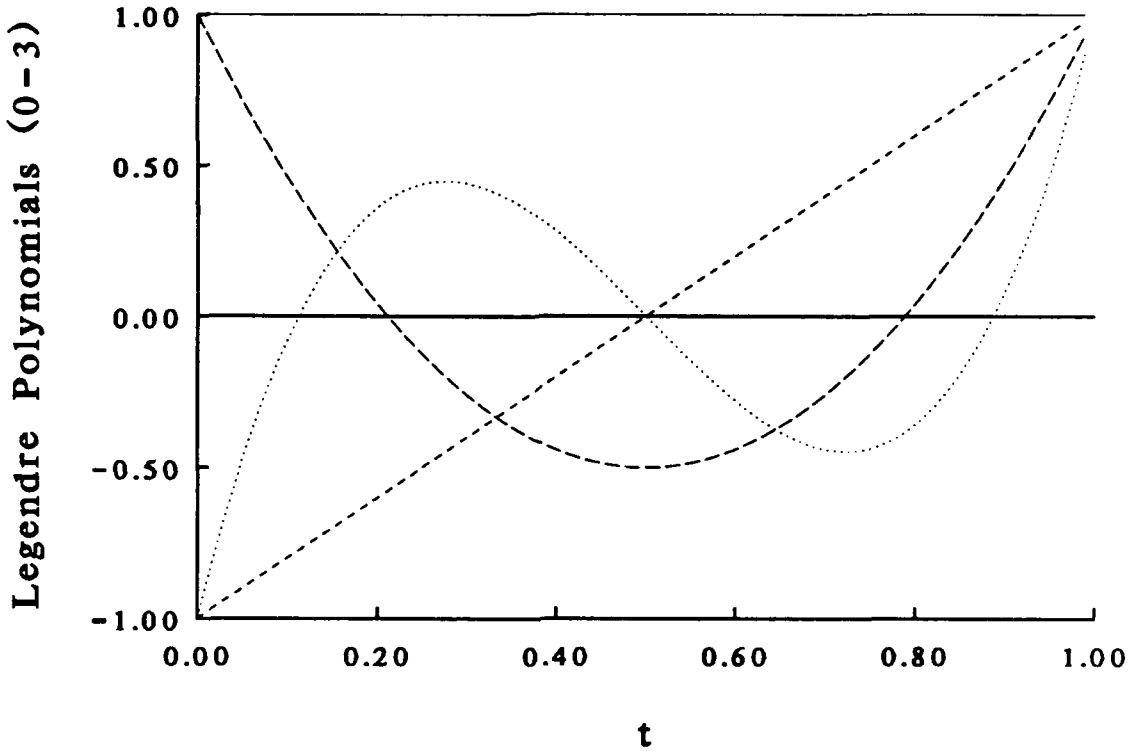


Figure 5.1: Shifted Legendre Polynomials

with

$$s_0(t) = 1$$
$$s_1(t) = \frac{2t}{t_f} - 1$$

Figure 5.1 shows the first four shifted Legendre polynomials on $[0, 1]$. They are orthogonal on $[0, t_f]$ with

$$\int_0^{t_f} s_i(t)s_j(t)dt = \begin{cases} 0, i \neq j \\ \frac{t_f}{2i+1}, i = j \end{cases} \tag{5.3}$$

A function $f(t)$ that is square integrable on the interval $[0, t_f]$ may be expanded into a shifted Legendre series

$$f(t) = \sum_{i=0}^{\infty} c_i s_i(t) \tag{5.4}$$

Hwang and Chen [20] showed that shifted Legendre polynomials have inte-

gration properties similar to Walsh functions, for example

$$\int_0^t S_N(\tau)d\tau = H_N S_N(t) \tag{5.5}$$

where $S_N(t) = [s_0(t)s_1(t)s_2(t)\cdots s_{N-1}(t)]^T$. Thus $S_N(t)$ is a vector of shifted Legendre polynomials and

$$H_N = \frac{t_f}{2}\begin{bmatrix} 1 & 1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ -\frac{1}{3} & 0 & \frac{1}{3} & 0 & \cdots & 0 & 0 & 0 \\ 0 & -\frac{1}{5} & 0 & \frac{1}{5} & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \frac{-1}{2N-3} & 0 & \frac{1}{2N-3} \\ 0 & 0 & 0 & 0 & \cdots & 0 & \frac{-1}{2N-1} & 0 \end{bmatrix} \tag{5.6}$$

The product of two shifted Legendre polynomials is a polynomial and can therefore be approximated by a truncated shifted Legendre series. Hwang and Chen give an approximation to the cross-product of two shifted Legendre vectors as

$$S(t)S^T(t) = \begin{bmatrix} s_0(t)s_0(t) & s_0(t)s_1(t) & \cdots & s_0(t)s_{N-1}(t) \\ s_1(t)s_0(t) & s_1(t)s_1(t) & \cdots & s_1(t)s_{N-1}(t) \\ \vdots & & & \vdots \\ s_{N-1}(t)s_0(t) & s_{N-1}(t)s_1(t) & \cdots & s_{N-1}(t)s_{N-1}(t) \end{bmatrix} \tag{5.7}$$

where

$$\begin{aligned} s_0(t)s_0(t) &= \textstyle\sum_{k=0}^{N-1} \frac{2k+1}{t_f} g_{00k} s_k(t) \\ s_0(t)s_1(t) &= \textstyle\sum_{k=0}^{N-1} \frac{2k+1}{t_f} g_{01k} s_k(t) \\ s_1(t)s_0(t) &= \textstyle\sum_{k=0}^{N-1} \frac{2k+1}{t_f} g_{10k} s_k(t) \\ s_1(t)s_1(t) &= \textstyle\sum_{k=0}^{N-1} \frac{2k+1}{t_f} g_{11k} s_k(t) \\ s_0(t)s_{N-1}(t) &= \textstyle\sum_{k=0}^{N-1} \frac{2k+1}{t_f} g_{0,N-1,k} s_k(t) \\ s_{N-1}(t)s_1(t) &= \textstyle\sum_{k=0}^{N-1} \frac{2k+1}{t_f} g_{N-1,0,k} s_k(t) \\ s_{N-1}(t)s_{N-1}(t) &= \textstyle\sum_{k=0}^{N-1} \frac{2k+1}{t_f} g_{N-1,N-1,k} s_k(t) \end{aligned} \tag{5.8}$$

and

$$g_{ijk} = \int_0^{t_f} s_i(t)s_j(t)s_k(t)dt \tag{5.9}$$

Now the same properties which were defined for the Walsh functions are defined for the Legendre polynomials paving the way for their use as basis functions for the Rayleigh-Ritz method.

Rayleigh-Ritz Method using a Legendre Polynomial Basis

There are two major differences between the previously outlined properties of the Legendre polynomials as compared to the Walsh functions. The first is the orthogonal property. For Walsh functions the result of the orthogonal property integral is the identity matrix.

$$\int_0^1 \phi_i(t)\phi_j(t)dt = \begin{cases} 0, & i \neq j \\ 1, & i = j \end{cases} \tag{5.10}$$

For Legendre polynomials the matrix is diagonal, but it is not the identity matrix:

$$\int_0^{t_f} s_i(t)s_j(t)dt = \begin{cases} 0, & i \neq j \\ \frac{t_f}{2i+1}, & i = j \end{cases} \tag{5.11}$$

or the result can be written as a matrix

$$O_l = \begin{bmatrix} t_f & 0 & 0 & 0 & 0 \\ 0 & \frac{t_f}{3} & 0 & 0 & 0 \\ 0 & 0 & \frac{t_f}{5} & 0 & 0 \\ 0 & 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & 0 & \frac{t_f}{2i+1} \end{bmatrix} \quad i = 0, 1, 2, \ldots N-1 \tag{5.12}$$

Second, the cross-product of two vectors of the functions is obtained differently. For Walsh functions it is obtained by simply multiplying the Walsh functions out (see Chapter 3). The group property of Walsh functions guarantees a resultant matrix in which all the resultant Walsh functions are in the same group as the original product matrices. For Legendre polynomials, however, the cross product of the two function vectors is a matrix of approximations of the product of the individual polynomial multiplications. This means the result of the product of two Walsh functions can be described exactly by another Walsh function whose subscript is no higher than the two

Walsh functions being multiplied. The product of two Legendre polynomials, however, must be approximated by another Legendre polynomial which has the higher order terms truncated to obtain a polynomial of the same order as those being multiplied. Now the solution using Walsh functions can be modified to allow the use of Legendre polynomials and only the above mentioned differences need to be taken into account.

The composite cost functional (3.5) in Chapter 3 must be approximated with Legendre polynomials.

$$J = \int_0^1 \{\frac{1}{2\varepsilon} S^T(t) E^T E S(t) + \frac{1}{2} S^T(t) X^T Q X S(t) + \frac{1}{2} S^T(t) U^T R U S(t)\} dt \qquad (5.13)$$

The quadratic terms in (5.13) can be simplyfied

$$S^T(t) M S(t) = \sum_{i=0}^{N-1} s_i(t) \sum_{j=0}^{N-1} m_{ij} s_j(t) \qquad (5.14)$$

and therefore

$$\int_0^{t_f} S^T(t) M S(t) dt = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} m_{ij} \int_0^{t_f} s_i(t) s_j(t) dt \qquad (5.15)$$

Since

$$\int_0^{t_f} s_i(t) s_j(t) dt = O_l \qquad (5.16)$$

and if O_l is diagonal, then

$$\sum_{i=0}^{N-1} \sum_{j=0}^{N-1} m_{ij} \int_0^{t_f} s_i(t) s_j(t) dt = \sum_{i=0}^{N-1} m_{ii} o_{ii} = trace(M O_l) \qquad (5.17)$$

Now J can be written in vec notation

$$J = \frac{1}{2\varepsilon} vec(E)^T (O_l \otimes I_n) vec(E) + \frac{1}{2} vec(X)^T (O_l \otimes Q) vec(X) + \frac{1}{2} vec(U)^T (O_l \otimes R) vec(U) \qquad (5.18)$$

The cost function can now be minimized by taking the gradient simultaneously with respect to both $vec(X)$ and $vec(U)$ and setting them equal to zero. By collecting terms, the result can be written in the same system of equations format as previously

given, where

$$\begin{aligned}
K_{11} &= \tfrac{1}{\epsilon}(I_{Nn} - (P^T \otimes I_n)\Lambda_\alpha)^T(O_l \otimes I_n)(I_{Nn} - (P^T \otimes I_n)\Lambda_\alpha) + (O_l \otimes Q) \\
K_{12} &= -\tfrac{1}{\epsilon}(I_{Nn} - (P^T \otimes I_n)\Lambda_\alpha)^T(O_l \otimes I_n)([P^T \otimes I_n]\Lambda_\beta) \\
K_{21} &= -\tfrac{1}{\epsilon}([P^T \otimes I_n]\Lambda_\beta)^T(O_l \otimes I_n)(I_{Nn} - (P^T \otimes I_n)\Lambda_\alpha) \\
K_{22} &= \tfrac{1}{\epsilon}([P^T \otimes I_n]\Lambda_\beta)^T(O_l \otimes I_n)([P^T \otimes I_n]\Lambda_\beta) + (O_l \otimes R)
\end{aligned} \tag{5.19}$$

$$\begin{aligned}
D_1 &= \tfrac{1}{\epsilon}(I_{Nn} - (P^T \otimes I_n)\Lambda_\alpha)^T(O_l \otimes I_n)(vec(G_s) + [P^T \otimes I_n]vec(C)) \\
D_2 &= -\tfrac{1}{\epsilon}([P^T \otimes I_n]\Lambda_\beta)^T(O_l \otimes I_n)(vec(G_s) + [P^T \otimes I_n]vec(C))
\end{aligned} \tag{5.20}$$

If

$$M = \frac{1}{\epsilon}\begin{bmatrix} I_{Nn} - (P^T \otimes I_n)\Lambda_\alpha & -(P^T \otimes I_n)\Lambda_\beta \\ 0 & 0 \end{bmatrix} \tag{5.21}$$

and

$$L = \begin{bmatrix} O_l \otimes Q & 0 \\ 0 & O_l \otimes R \end{bmatrix} \tag{5.22}$$

and now an additional matrix is necessary,

$$G = \begin{bmatrix} O_l \otimes I_n & 0 \\ 0 & 0 \end{bmatrix} \tag{5.23}$$

then K is

$$K = M^T G M + L \tag{5.24}$$

So the linear time varying optimal control problem can be solved using Legendre polynomials as basis functions. As a result of the above development, the nonlinear optimal control problem can be solved using Legendre polynomials. Only minor modifications must be made to the parallel implementation used for the Walsh functions. The matrix G is a diagonal matrix, so it just scales the columns of M^T and the full three matrix multiply remains a two matrix multiply.

Consider now the addition of terminal constraints on the state. The development of Chapter 4 can be followed with modification for Legendre polynomials. The penalty term

$$\rho(\varepsilon) = x_s + \int_0^{t_f} A(t)x(t)dt + \int_0^{t_f} B(t)u(t)dt + \int_0^{t_f} C(t)dt - x_f \tag{5.25}$$

must be approximated with Legendre polynomials in order to find $vec(\xi)$. As in Chapter 4 consider the scalar case first.

$$\int_0^{t_f} A(t)x(t)dt = \int_0^{t_f} \alpha S(t)S^T(t)X^T dt \tag{5.26}$$

Which using the orthogonal property of Legendre polynomials can be written

$$\int_0^{t_f} \alpha S(t)S^T(t)X^T dt = \alpha O_l X^T \tag{5.27}$$

Now for the case where $n > 1$

$$\int_0^{t_f} A(t)x(t)dt = \begin{bmatrix} \int_0^{t_f} a_{11}(t)x_1(t)dt + \cdots + \int_0^{t_f} a_{1n}(t)x_n(t)dt \\ \vdots \\ \int_0^{t_f} a_{n1}(t)x_1(t)dt + \cdots + \int_0^{t_f} a_{nn}(t)x_n(t)dt \end{bmatrix} \tag{5.28}$$

The above matrix can be written

$$\begin{bmatrix} \alpha_{11}O_l & \cdots & \alpha_{1n}O_l \\ \vdots & & \vdots \\ \alpha_{n1}O_l & \cdots & \alpha_{nn}O_l \end{bmatrix} \begin{bmatrix} X_1^T \\ \vdots \\ X_n^T \end{bmatrix} \tag{5.29}$$

This is not in standard vec form, but as shown in Chapter 4, it can be rearranged into *vec* notation and Γ_α can be formed. This allows the integral to be written,

$$\int_0^{t_f} \alpha S(t)S^T(t)X^T dt = \Gamma_\alpha vec(X) \tag{5.30}$$

The same approach can be used to approximate the integral with $B(t)u(t)$, and the approximation for the integral with $C(t)$ is the same as in chapter 4. Now $vec(\xi)$ can be written

$$vec(\xi) = vec(G_s) + \Gamma_\alpha vec(X) + \Gamma_\beta vec(U) + \Gamma_c vec(C) - vec(G_f) \tag{5.31}$$

The cost function becomes

$$\begin{aligned} J = \; & \tfrac{1}{2\varepsilon} vec(E)^T (O_l \otimes I_n) vec(E) + \tfrac{1}{2\varepsilon} vec(\xi)^T (O_l \otimes I_n) vec(\xi) \\ & + \tfrac{1}{2} vec(X)^T (O_l \otimes Q) vec(X) + \tfrac{1}{2} vec(U)^T (O_l \otimes R) vec(U) \end{aligned} \tag{5.32}$$

Minimization gives the system of equations

$$\begin{aligned} K_{11} = \; & \tfrac{1}{\varepsilon}(I_{Nn} - (P^T \otimes I_n)\Lambda_\alpha)^T (O_l \otimes I_n)(I_{Nn} - (P^T \otimes I_n)\Lambda_\alpha) + (O_l \otimes Q) \\ & + \tfrac{1}{\varepsilon}\Gamma_\alpha^T (O_l \otimes I_n)\Gamma_\alpha \end{aligned}$$

$$K_{12} = \; -\tfrac{1}{\varepsilon}(I_{Nn} - (P^T \otimes I_n)\Lambda_\alpha)^T (O_l \otimes I_n)([P^T \otimes I_n]\Lambda_\beta) + \tfrac{1}{\varepsilon}\Gamma_\alpha^T (O_l \otimes I_n)\Gamma_\beta$$

$$K_{21} = \; -\tfrac{1}{\varepsilon}([P^T \otimes I_n]\Lambda_\beta)^T (O_l \otimes I_n)(I_{Nn} - (P^T \otimes I_n)\Lambda_\alpha) + \tfrac{1}{\varepsilon}\Gamma_\beta^T (O_l \otimes I_n)\Gamma_\alpha$$

$$K_{22} = \; \tfrac{1}{\varepsilon}([P^T \otimes I_n]\Lambda_\beta)^T (O_l \otimes I_n)([P^T \otimes I_n]\Lambda_\beta) + (O_l \otimes R) + \tfrac{1}{\varepsilon}\Gamma_\beta^T (O_l \otimes I_n)\Gamma_\beta \tag{5.33}$$

$$\begin{aligned} D_1 = \; & \tfrac{1}{\varepsilon}(I_{Nn} - (P^T \otimes I_n)\Lambda_\alpha)^T (O_l \otimes I_n)(vec(G_s) + [P^T \otimes I_n] vec(C)) \\ & + \tfrac{1}{\varepsilon}\Gamma_\alpha^T (O_l \otimes I_n)\{G_f - G_s - \Gamma_c vec(C)\} \end{aligned} \tag{5.34}$$

$$\begin{aligned} D_2 = \; & -\tfrac{1}{\varepsilon}([P^T \otimes I_n]\Lambda_\beta)^T (O_l \otimes I_n)(vec(G_s) + [P^T \otimes I_n] vec(C)) \\ & + \tfrac{1}{\varepsilon}\Gamma_\beta^T (O_l \otimes I_n)\{G_f - G_s - \Gamma_c vec(C)\} \end{aligned}$$

If

$$M = \begin{bmatrix} I_{Nn} - (P^T \otimes I_n)\Lambda_\alpha & -(P^T \otimes I_n)\Lambda_\beta \\ 0 & 0 \end{bmatrix} \tag{5.35}$$

$$L = \begin{bmatrix} O_l \otimes Q & 0 \\ 0 & O_l \otimes R \end{bmatrix} \quad G = \begin{bmatrix} O_l \otimes I_n & 0 \\ 0 & 0 \end{bmatrix} \tag{5.36}$$

$$F = \begin{bmatrix} \Gamma_\alpha & \Gamma_\beta \\ 0 & 0 \end{bmatrix} \tag{5.37}$$

then

$$K = \frac{1}{\varepsilon} M^T G M + L + \frac{1}{\varepsilon} F^T G F \tag{5.38}$$

Now both the Raleigh and van der Pol problems can be solved using Legendre polynomials.

Iteration	$\| \delta x(t) \|_{max}$	$\frac{\|e(t;\epsilon)\|^2}{\epsilon}$	$J(\epsilon, x(t), u(t))$
1	4.1064	0.0009	18.1504
2	0.3242	0.0008	16.9802
3	0.1082	0.0008	16.9553
4	0.0224	0.0008	16.9541
5	0.0043	0.0008	16.9540

Table 5.1: Raleigh Problem Solution with Eight Legendre Polynomials

Computational Results

In this section the Raleigh problem (Problem 4.1) and van der Pol problem (Problem 4.2) are solved using Legendre polynomials. For comparison purposes, eight Legendre polynomials are used. As shown in the previous section once the product matrix $S(t)S^T(t)$ is determined, the only other difference is the addition of the non-identity orthogonal matrix O_l. Since it is a diagonal matrix, full matrix-matrix multiplications are not necessary.

Raleigh Problem Solution with Legendre Polynomials

For this problem the initial nominal trajectories were set to unity. Epsilon was set to 0.0001. As with the Walsh functions, after five iterations the cost was within .3 % of the optimal cost. and no noticeable improvement in the control was observed. Figure 5.2 shows the control for the first five iterations. Table 5.1 shows the error function and the cost function J for each iteration. Figure 5.3 shows the Legendre approximation of the optimal control plotted against the Walsh approximation. The solution time on one node was 13.6 seconds. On all eight nodes the solution time was about 3.9 seconds, for a speed-up of 3.5.

Actually, with the Legendre polynomials it may be possible to get a very close approximation to the solution of the Raleigh problem with much less computation. Figure 5.4 shows the optimal control approximated by four Legendre polynomials plotted against the solution using four Walsh functions.

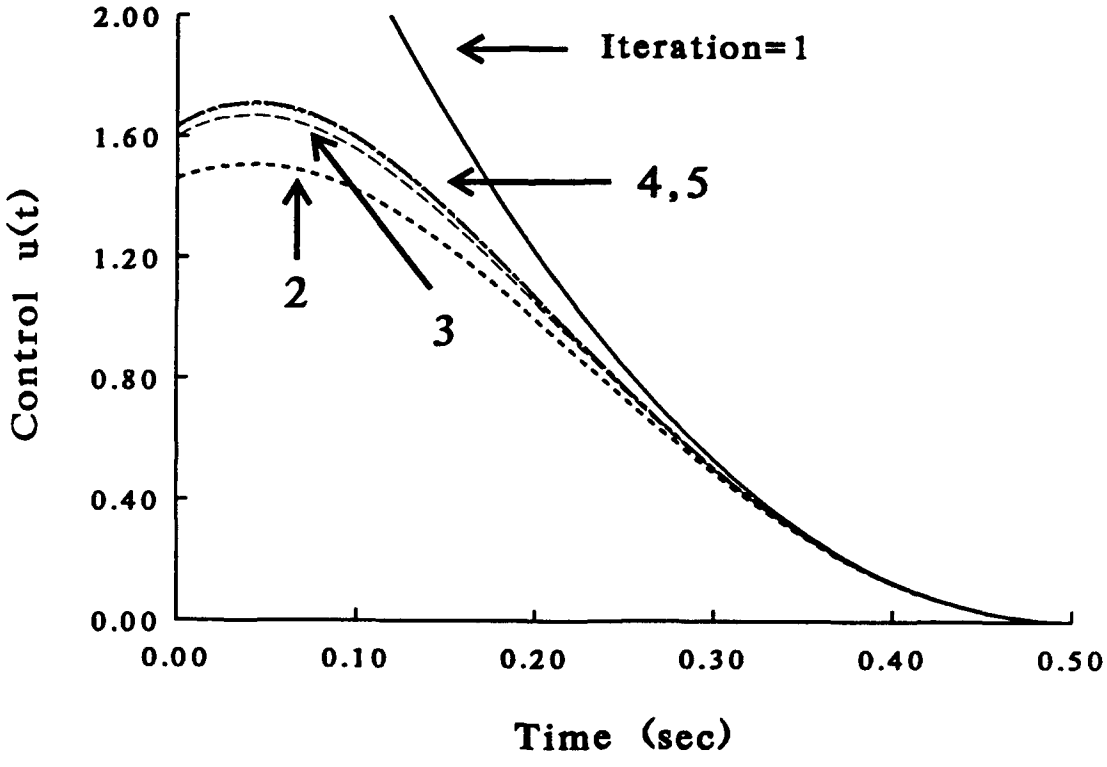


Figure 5.2: Raleigh Problem Solution with Eight Legendre Polynomials

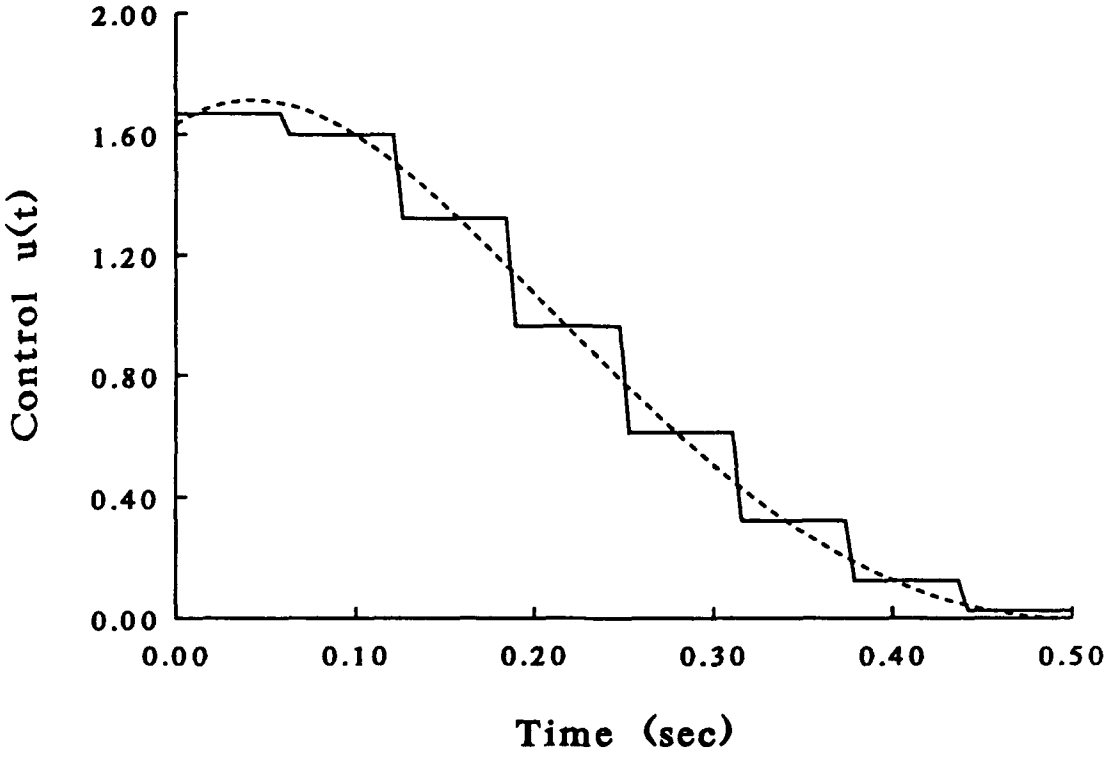


Figure 5.3: Raleigh Solution : Eight Walsh Functions versus Eight Legendre Polynomials

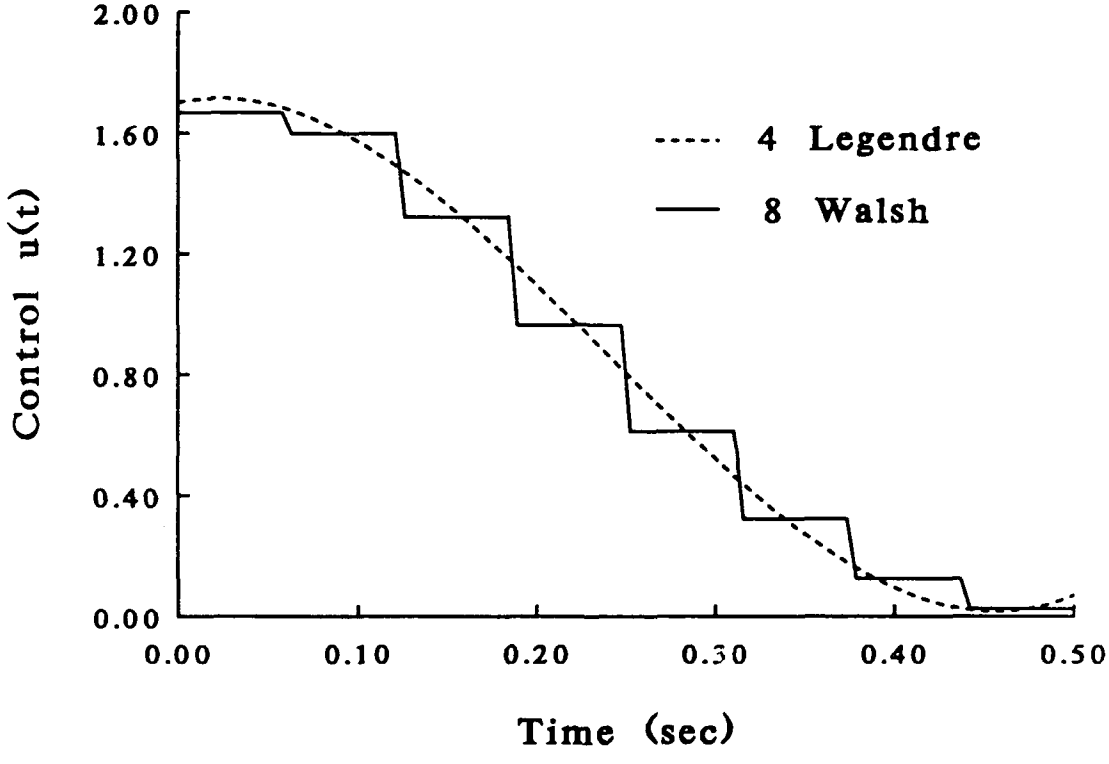


Figure 5.4: Raleigh Solution: Eight Walsh Functions versus Four Legendre Polynomials

Iteration	$\| \delta x(t) \|_{max}$	$\frac{\|e(t;\epsilon)\|^2}{\epsilon}$	$\frac{\|\rho(\epsilon)\|^2}{\epsilon}$	$x_1(5)$	$x_2(5)$	$J(\epsilon, x(t), u(t))$
1	2.8098	0.0270	0.0031	-.9551	-.9492	12.2379
2	2.3719	0.0028	0.0001	-.9387	-.7050	3.8899
3	0.9453	0.0040	0.0002	-.9723	-.9561	4.2096
4	0.0196	0.0040	0.0002	-.9707	-.9662	4.2169
5	0.0004	0.0040	0.0002	-.9707	-.9658	4.2166

Table 5.2: van der Pol Solution with Eight Legendre Polynomials

van der Pol Problem Solution with Legendre Polynomials

As with the Walsh functions, results for the van der Pol problem (Problem 4.2) are very similar to results from the Raleigh problem. The nominal trajectories were set to unity. Epsilon was set to 0.0001. A solution was obtained in 4-5 iterations. Table 5.2 gives the statistics of interest for these iterations. Figure 5.5 shows the Legendre approximation of the control and states plotted against the Walsh approximation. Notice also from Table 5.2 that the terminal constraint is almost met exactly. As expected, the solution times for the case of Legendre polynomials are not significantly different than for Walsh functions. Solution time on one node was 14.3 seconds and on all eight nodes was 4.2 for a speed-up of 3.4. Legendre polynomials are only one type in the class of orthogonal polynomials. It is possible to use other orthogonal polynomials to solve the epsilon problem. In the next section the shifted Chebyshev polynomials are examined.

Chebyshev Polynomials as Basis Functions

It is evident from Chapter 3 and 5 that three key properties of orthogonal basis functions are used in the solution of the epsilon problem using the Ritz method. These properties result in matrices which are used in the solution to the epsilon problem. The matrices which must be found are the orthogonal matrix, the matrix of integration and the product of two vectors of the orthogonal functions. For Walsh functions, these matrices were defined as O_w, P, and $\Psi(t)\Psi^T(t)$. For Legendre

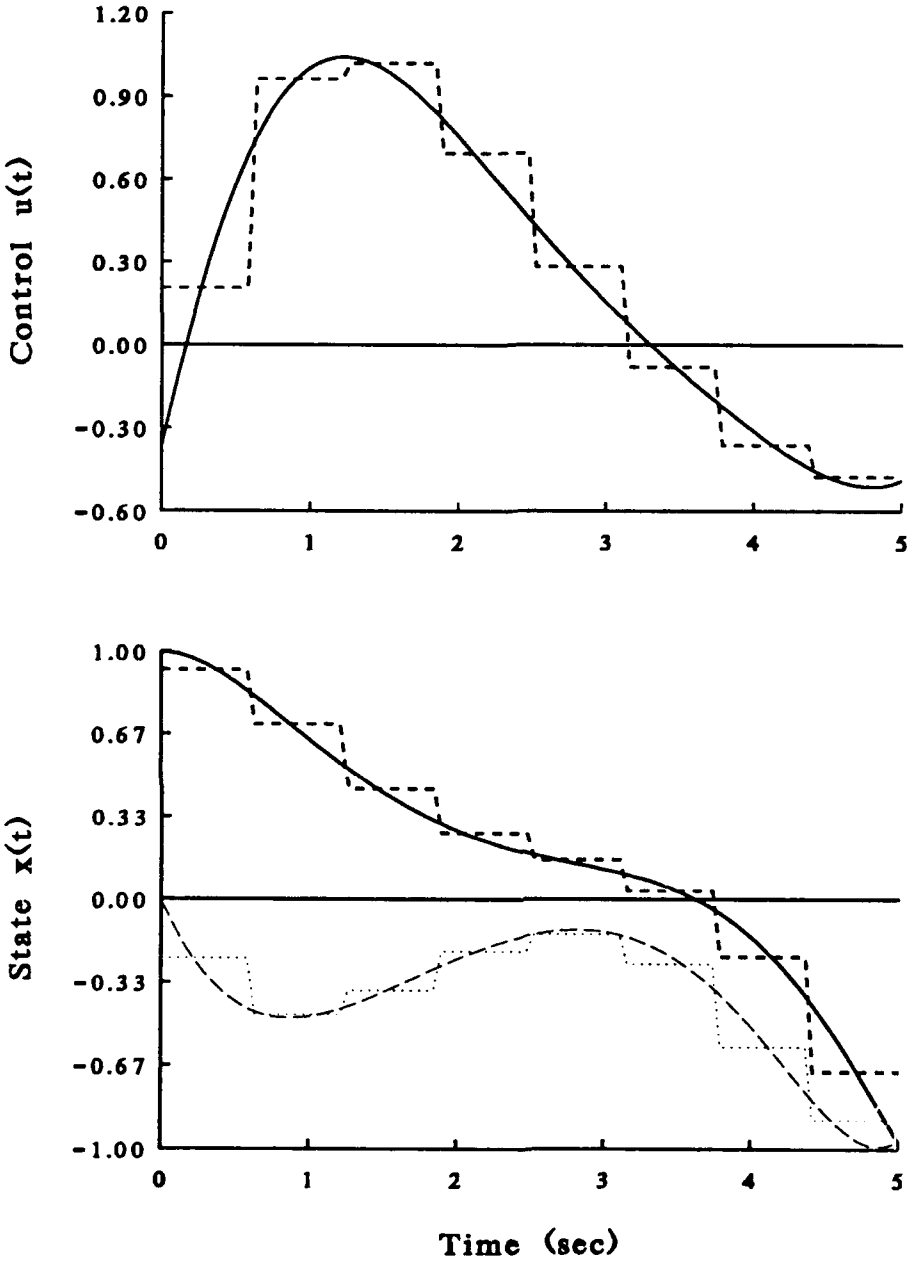


Figure 5.5: van der Pol Solution : Eight Walsh Functions versus Eight Legendre Polynomials

polynomials, they were defined as O_l, H, and $S(t)S^T(t)$. Once these matrices are found for a particular set of basis functions it is straightforward to use the functions as basis functions to solve the epsilon problem. This is demonstrated with Chebyshev polynomials which are in the class of orthogonal polynomials.

The Chebyshev polynomials are defined between interval -1 and 1. They can be shifted onto the interval [0,1] and are given as

$$\begin{aligned} T_0(t) &= 1 \\ T_1(t) &= 1 - 2t \\ T_2(t) &= 8t^2 - 8t + 1 \\ T_3(t) &= -32t^3 + 48t^2 - 18t + 1 \\ &\vdots \\ T_{i+1}(t) &= (2 - 4t)T_i(t) - T_{i-1}(t) \end{aligned} \tag{5.39}$$

Multiplication of Chebyshev polynomials is not difficult. The cross product of two Chebyshev vectors is given as

$$T_i(t)T_j(t) = \frac{1}{2}[T_{i+j}(t) + T_{i-j}] \tag{5.40}$$

Obviously, to keep the number of polynomials used as a basis the same, the terms in the cross product matrix need to be truncated. For example if $N = 3$

$$T(t)T^T(t) = \begin{bmatrix} T_0(t) & T_1(t) & T_2(t) \\ T_1(t) & \frac{1}{2}[T_2(t) + T_0(t)] & \frac{1}{2}T_1(t) \\ T_2(t) & \frac{1}{2}T_1(t) & \frac{1}{2}T_0(t) \end{bmatrix}$$

where the $(3,1)$,$(3,2)$, and $(3,3)$ terms have been truncated so there are no polynomials of higher order than $T_2(t)$.

The integration matrix for shifted Chebyshev polynomials was derived by Liu and Shih [22]. It is given as follows,

$$\int_0^t T(t)dt = H_c T(t) \tag{5.41}$$

where H_c is the integration matrix

$$H_c = \begin{bmatrix} \frac{1}{2} & -\frac{1}{2} & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ \frac{1}{8} & 0 & -\frac{1}{8} & 0 & 0 & \cdots & 0 & 0 & 0 \\ -\frac{1}{6} & \frac{1}{4} & 0 & -\frac{1}{12} & 0 & \cdots & 0 & 0 & 0 \\ -\frac{1}{16} & 0 & \frac{1}{8} & 0 & -\frac{1}{16} & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ -\frac{1}{2N(N-2)} & 0 & 0 & 0 & 0 & \cdots & \frac{1}{4(N-2)} & 0 & -\frac{1}{4N} \\ -\frac{1}{2(N+1)(N-1)} & 0 & 0 & 0 & 0 & \cdots & 0 & \frac{1}{4(N-1)} & 0 \end{bmatrix} \tag{5.42}$$

Finally the orthogonality condition for shifted Chebyshev polynomials is given as

$$\int_0^1 \frac{T_i(t)T_j(t)}{\sqrt{t-t^2}}dt = \begin{cases} 0, & i \neq j \\ \pi/2, & i = j \neq 0 \\ \pi, & i = j = 0 \end{cases} \tag{5.43}$$

The weight function for Chebyshev polynomials is not one. To remain consistent with the parallel solution developed with Legendre polynomials, the integral of the cross product of two Chebyshev vectors is needed. This only needs to be calculated once, so for the given example it is found simply by numerical integration. For example, if $N = 4$

$$O_c = \begin{bmatrix} 1.0000 & 0.0000 & -0.3333 & 0.0000 \\ 0.0000 & 0.3333 & 0.0000 & -0.2000 \\ -0.3333 & 0.0000 & 0.4667 & 0.0000 \\ 0.0000 & -0.2000 & 0.0000 & 0.4857 \end{bmatrix}$$

Raleigh Problem Solution with Chebyshev Polynomials

Using the properties given in the last section, the Raleigh problem (Problem 4.1) can be solved using Chebyshev polynomials. For this problem, 4 Chebyshev polynomials are used. Epsilon is set to 0.0001 and the nominal trajectories are set to unity. Table 5.3 gives the solution using Chebyshev polynomials. After five iterations the solution cost is very close to the optimal cost of 17. Figure 5.6 shows the Chebyshev approximated control plotted against Legendre and Walsh approximations. Just

Iteration	$\| \delta x(t) \|_{max}$	$\frac{\|e(t;\varepsilon)\|^2}{\varepsilon}$	$J(\varepsilon, x(t), u(t))$
1	4.1974	0.0009	18.1493
2	0.2928	0.0008	16.9647
3	0.1164	0.0008	16.9344
4	0.0260	0.0008	16.9327
5	0.0052	0.0008	16.9327

Table 5.3: Raleigh Problem Solution with Chebyshev Polynomials

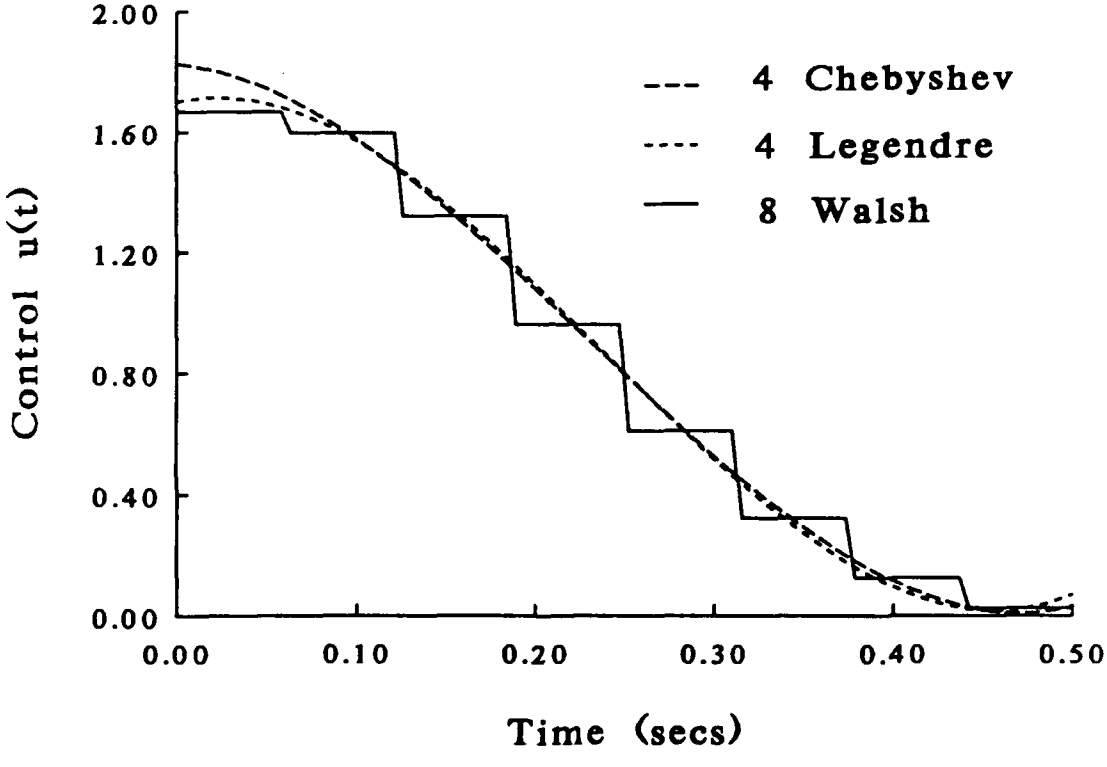


Figure 5.6: Raleigh Problem Solution : Chebyshev Polynomials

as in the case of Legendre polynomials, the Chebyshev polynomials give a very close approximation to the optimal solution.

CHAPTER 6

NONLINEAR MINIMUM TIME OPTIMAL CONTROL PROBLEMS

In the previous chapters, only polynomial nonlinearities in the dynamical equations were considered. In this chapter, some interesting computational properties of Walsh functions are examined which allow a broader class of nonlinearities to be dealt with. These properties of the Walsh functions also allow the solution of complex problems of a more practical nature, e.g., nonlinear minimum time optimal control problems which contain more than simple polynomial nonlinearities in the dynamical equations. Results from this chapter will also allow the solution of problems which involve inequality constraints on the state or more typically the control. Solution of problems with inequality constraints will be dealt with in chapter 7.

Nonlinear Operations on Walsh Functions

In previous chapters, polynomial nonlinearities in the dynamical equations were dealt with by using the multiplication properties of Walsh functions. Other nonlinear operations such as the inverse, exponential, or sin and cos functions cannot be dealt with in the same fashion. An alternate approach must be used.

Consider some function $f(t)$ approximated by N Walsh functions. The approximation contains the value of $f(t)$ at N specific intervals and as such can be described by a vector whose N elements give the value of $f(t)$ at each of the N intervals. This is demonstrated by considering the Walsh function $\phi_2(t)$ (See Figure 2.2). This Walsh function can be completely described by the vector

$$p_2 = \begin{bmatrix} 1 & -1 & 1 & -1 \end{bmatrix}$$

where each of the elements gives the value of $\phi_2(t)$ on each of its four different intervals. As another example, consider $\phi_3(t)$. It is described by

$$p_3 = \begin{bmatrix} 1 & -1 & -1 & 1 \end{bmatrix}$$

It is possible to describe first 4 Walsh functions by a vector with 4 elements, the first 8 Walsh functions by vectors with 8 elements and so on. In this manner, the vector $\Psi(t)$ can be described by a matrix,

$$\Psi(t) = \begin{bmatrix} \phi_0(t) \\ \phi_1(t) \\ \vdots \\ \phi_N(t) \end{bmatrix} = \begin{bmatrix} p_0 \\ p_1 \\ \vdots \\ p_N \end{bmatrix} = \mathcal{P} \tag{6.1}$$

And, just as it is possible to approximate a function by

$$f(t) \approx F\Psi(t) \tag{6.2}$$

the alternate form can be used

$$f(t) \approx F\mathcal{P} \tag{6.3}$$

where $\mathcal{P}$ now is a matrix. The following example will help clarify the process.

EXAMPLE:

Let $f(t) = t$ and $N = 4$, then $f(t)$ can be approximated by a Walsh series

$$f(t) \approx F\Psi(t)$$

where

$$F = \begin{bmatrix} \frac{1}{2} & -\frac{1}{4} & -\frac{1}{8} & 0 \end{bmatrix}$$

Thus

$$f(t) \approx F_t = F\mathcal{P} = \begin{bmatrix} .125 & .375 & .625 & .875 \end{bmatrix}$$

and F_t describes the Walsh series approximation for $f(t)$.

The operation just described amounts to an efficient numerical method of obtaining the actual Walsh series from the Walsh coefficients. It also gives a technique

for getting the Walsh coefficients from the Walsh series. Since the Walsh functions are orthogonal, then $\mathcal{P}$ is invertable and it is possible to write

$$F_t\mathcal{P}^{-1} = F \tag{6.4}$$

where F_t is some Walsh approximation to a time function given in the vector form and F then is the matrix of Walsh coefficients of the Walsh series. By use of the $\mathcal{P}$ matrix it is possible to generate Walsh functions from Walsh coefficients and Walsh coefficients from Walsh functions. Performing nonlinear operations on Walsh functions described by a set of Walsh coefficients now becomes straightforward. Nonlinear operations cannot be performed directly on a set of Walsh coefficients, but they can be performed elementwise on a Walsh series in the vector form. An example is given to clarify this point.

EXAMPLE:

It is desired to find $g(t) = \sin(f(t))$ with only the Walsh coefficients of $f(t)$ are known. In other words ,

$$f(t) \approx F\Psi(t)$$

and F is known. It is desired to find G where

$$g(t) \approx G\Psi(t) = \sin(F\Psi(t))$$

This is straightforward using the $\mathcal{P}$ matrix. The above is equivalent to

$$\sin(F\mathcal{P}) = G\mathcal{P} = G_t$$

so

$$G = \sin(F\mathcal{P})\mathcal{P}^{-1}$$

Let $N = 4$ and assume

$$F = \begin{bmatrix} 1.5 & -.75 & -.375 & 0 \end{bmatrix}$$

which gives the Walsh approximation for $f(t) = 3t$. From F, the Walsh coefficients for $\sin(3t)$ are easily found

$$G = \sin(F\mathcal{P})\mathcal{P}^{-1} = \begin{bmatrix} .679 & -.045 & -.019 & -.249 \end{bmatrix}$$

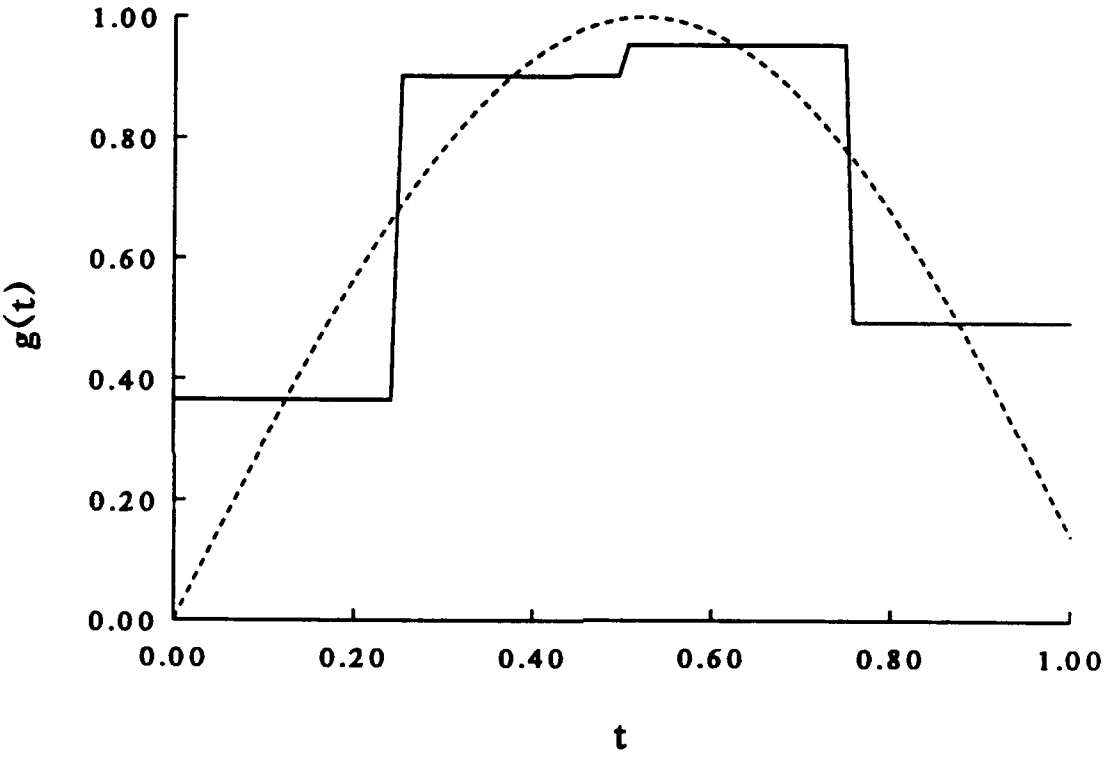


Figure 6.1: Walsh Approximation to $\sin(3t)$

Figure 6.1 shows the Walsh approximation using G against the actual function $\sin(3t)$. Other nonlinear functions can be performed on Walsh coefficients in the same manner. By use of the $\mathcal{P}$ matrix, it is possible to solve nonlinear optimal control problems which have many types of nonlinearities other than just polynomial nonlinearities. In the next section this is demonstrated by numerical solution of a nonlinear minimum time optimal control problem. This problem has other than polynomial nonlinearities in the system dynamical equations and is intended to demonstrate the practical value of the method.

Solution of a Nonlinear Minimum Time Optimal Control Problem

In the previous section, the tools were developed to allow the parallel solution of nonlinear optimal control problems with other than polynomial nonlinearities. This will be demonstrated by solution of a nonlinear minimum time optimal control problem. The particular problem of interest involves determination of the control which generates a minimum time transfer trajectory from Earth orbit to Mars orbit. The following problem is a variation of a problem solved by Taylor [23]. The equations of motion which govern a spacecraft with constant thrust throughout the time of flight are given as

PROBLEM: 6.1:

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= \frac{x_3^2}{x_1} - \frac{\mu}{x_1^2} + \frac{T\sin\theta}{m_o + \dot{m}t} \\ \dot{x}_3 &= -\frac{x_2 x_3}{x_1} + \frac{T\cos\theta}{m_o + \dot{m}t} \end{aligned} \tag{6.5}$$

where x_1 is radial position, x_2 is radial velocity, x_3 is tangential velocity, T is thrust, θ is thrust angle, μ is the gravitational constant, m_o is the initial mass, $\dot{m}$ is mass flow rate, and t is time. The problem is normalized to astronomical units. The initial

values for the state variables are

$$\begin{aligned} x_1(0) &= 1.0 \\ x_2(0) &= 0.0 \\ x_3(0) &= 1.0 \end{aligned} \tag{6.6}$$

The above initial conditions define an initial orbit which is the same as Earth's orbit. The final values for the state variables must be

$$\begin{aligned} x_{1_{t_f}} &= 1.525 \\ x_{2_{t_f}} &= 0.0 \\ x_{3_{t_f}} &= 0.8098 \end{aligned} \tag{6.7}$$

The final conditions define a final orbit which is the same as Mars's orbit. Parameter values are

$$\begin{aligned} \mu &= 1.000 \\ T &= 0.1405 \\ m_o &= 1.000 \\ \dot{m} &= 0.0749 \end{aligned}$$

and 1 time unit equals 58.18 days. Finally, the cost function to be minimized is

$$J = \int_0^{t_f} 1 \; dt \tag{6.8}$$

This defines the problem as a minimum time problem. In other words, the optimal time t_f^* must be found which minimizes J. The problem just given can be summarized as follows: given the equations of motion, find the thrust angle trajectory which transfers a spacecraft from Earth orbit to Mars orbit in the minimum time.

Solution by Steepest Descent

To obtain a reference for later comparisons, the problem is first solved using the method of steepest descent. Steepest descent is basically a method for solving the two point boundary value problem resulting from (2.15) thru (2.19) in Chapter 2. The form of these equations naturally lend themselves to solution by steepest descent.

First the TPBVP which describes the Earth to Mars problem is given. The equations of motion are repeated for convenience.

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= \frac{x_3^2}{x_1} - \frac{\mu}{x_1^2} + \frac{T \sin u}{m_o + \dot{m}t} \\ \dot{x}_3 &= -\frac{x_2 x_3}{x_1} + \frac{T \cos u}{m_o + \dot{m}t} \end{aligned} \tag{6.9}$$

The Hamiltonian is

$$H = t_f + \lambda_1 x_2 + \lambda_2 (\frac{x_3^2}{x_1} - \frac{\mu}{x_1^2} + \frac{T \sin u}{m_o + \dot{m}t}) + \lambda_3 (-\frac{x_2 x_3}{x_1} + \frac{T \cos u}{m_o + \dot{m}t}) \tag{6.10}$$

From this the costate equations can be written

$$\begin{aligned} \dot{\lambda_1} &= [\frac{x_3^2}{x_1^2} - \frac{2\mu}{x_1^3}]\lambda_2 - [\frac{x_2 x_3}{x_1^2}]\lambda_3 \\ \dot{\lambda_2} &= -\lambda_1 + [\frac{x_3}{x_1}]\lambda_3 \\ \dot{\lambda_3} &= -[\frac{2x_3}{x_1}]\lambda_2 + [\frac{x_2}{x_1}]\lambda_3 \end{aligned} \tag{6.11}$$

Finally

$$\frac{\partial H}{\partial u} = \frac{T}{m_o + \dot{m}t}[\lambda_2 \cos u - \lambda_3 \sin u] \tag{6.12}$$

Note that the terminal conditions can be met explicitly, or the following cost function can be used

$$J = \int_0^{t_f} 1 \; dt + S(x_1(t_f) - x_{1_{t_f}})^2 + S(x_2(t_f) - x_{2_{t_f}})^2 + S(x_3(t_f) - x_{3_{t_f}})^2 \tag{6.13}$$

which forces the states to be very close to the required terminal state values if S is chosen very large. In this case, the boundary conditions are

$$\begin{aligned} x_1(0) &= 1.0 \\ x_2(0) &= 0.0 \\ x_3(0) &= 1.0 \end{aligned} \tag{6.14}$$

$$\begin{aligned} \lambda_1(t_f) &= s(x_1(t_f) - x_{1_{t_f}}) \\ \lambda_2(t_f) &= s(x_2(t_f) - x_{2_{t_f}}) \\ \lambda_3(t_f) &= s(x_3(t_f) - x_{3_{t_f}}) \end{aligned} \tag{6.15}$$

Now a control trajectory $u(t)$ is chosen which is close to the optimal control trajectory $u^*(t)$. The state equations can be integrated using this $u(t)$ and the final conditions on the state are used to find the final conditions on the costate using (6.14). The costate equations are then integrated backwards. Now the costate trajectories can be used to determine $\frac{\partial H}{\partial u}$. If $\frac{\partial H}{\partial u} = 0$ then $u(t) = u^*(t)$, otherwise $u(t)$ can be corrected by

$$u_{i+1}(y) = u_i(t) - K\frac{\partial H}{\partial u} \tag{6.16}$$

where $K < 1$ and is a small number chosen for best convergence. The new $u(t)$ replaces the old one and the process is repeated until

$$| \frac{\partial H}{\partial u} | < \epsilon \tag{6.17}$$

where ϵ is some arbitrarily small number. This is the method of steepest descent.

In minimum time problems, the cost function must also be minimized with respect to the final time t_f. Since the cost function is a function of t_f, one way to minimize the cost function is to solve a series of steepest descent problems. For each consecutive problem choose the new t_f by some line search optimization technique, for example quadratic curve fitting (See Luenberger [24]). This technique was used to solve the Earth to Mars minimum time problem. The starting control was chosen to be 1.2 radians for the first half of flight and -1.2 radians for the second half of flight. For $S = 10000$, the solution is

$$t_f = 3.9515$$
$$J_{final} = 3.9589$$

Figures 6.2-6.5 show the optimal control and states.

Solution by the Epsilon Method

The integral epsilon formulation of the Earth to Mars minimum time problem is given as

$$J(\varepsilon, x(t), u(t)) = \frac{1}{2\varepsilon}\{\int_0^{t_f} \| e(t;\varepsilon) \|^2 \, dt + \| \rho(\varepsilon) \|^2\} + \int_0^{t_f} 1 dt \tag{6.18}$$

as before

$$e(t;\varepsilon) = x(t) - x_s - \int_0^t f(x(\tau), u(\tau), \tau)d\tau \tag{6.19}$$

and

$$\rho(\varepsilon) = x_s + \int_0^{t_f} f(x(t), u(t), t)dt - x_f \tag{6.20}$$

This problem can be solved as shown in Chapters 3 and 4. First the problem must be linearized

$$\dot{x}(t) = A(t)x(t) + B(t)u(t) + C(t) \tag{6.21}$$

where

$$A(t) = \begin{bmatrix} 0 & 1 & 0 \\ -\frac{x_{3_o}^2}{x_{1_o}^3} + \frac{2\mu}{x_{1_o}^3} & 0 & 2\frac{x_{3_o}}{x_{1_o}} \\ \frac{x_{2_o}x_{3_o}}{x_{1_o}} & -\frac{x_{3_o}}{x_{1_o}} & -\frac{x_{2_o}}{x_{1_o}} \end{bmatrix} \tag{6.22}$$

$$B(t) = \begin{bmatrix} 0 \\ a(t)\cos u_o \\ -a(t)\sin u_o \end{bmatrix} \tag{6.23}$$

$$C(t) = \begin{bmatrix} 0 \\ -\frac{3}{x_{1_o}^2} + a(t)\sin u_o - (a(t)\cos u_o)u_o \\ a(t)\cos u_o + (a(t)\sin u_o)u_o \end{bmatrix} \tag{6.24}$$

Here $x_{1_o}, x_{2_o}, x_{3_o}$, and u_o are some nominal trajectories and

$$a(t) = \frac{T}{m_o + \dot{m}t}$$

The Walsh coefficients needed to approximate the above matrices can be found using the $\mathcal{P}$ matrix once the initial nominal trajectories have been chosen. Once the matrices have been approximated in terms of the Walsh coefficients, the problem can be solved as before in Chapter 4. To find the optimal t_f, some line search optimization is performed as done with the steepest descent technique. The problem solution is summarized as follows:

- Pick initial nominal trajectories.
- Pick an initial t_f.

Walsh Functions	t_f	J_{final}
4	4.018	4.042
8	3.949	3.975
16	3.921	3.946

Table 6.1: Minimum Time versus Number of Walsh Functions

- Solve the linearized problem iteratively until

$$\| x - x_o \|_{\max} < \epsilon_1$$

- Change t_f using some optimal search scheme and repeat until

$$\| t_f^{i+1} - t_f^i \| < \epsilon_2$$

where ϵ_1 and ϵ_2 are some small numbers.

The above scheme was used to solve the Earth to Mars problem using 4,8 and 16 Walsh functions. The nominal trajectories were chosen as follows:

$$x_{1_o} = 1.0$$
$$x_{2_o} = 0.0$$
$$x_{3_o} = 1.0$$
$$u_o = .5$$

Table 6.1 gives the minimum time results as a function of the number of Walsh functions. Results of a single epsilon problem using eight Walsh functions, the minimum time, and the given initial nominal trajectories are summarized in table 6.2. As expected, speed of convergence is dependent on the initial pick of the nominal trajectories. Since u_0 is not very close to $u^*(t)$, it takes a number of iterations to converge. It does show, however, that the method converges even for relatively poor initial guesses. To test the actual effectiveness of the method, the approximated control was used to drive the nonlinear equations of motion (6.9) from the given initial conditions. The results of the simulation are plotted against the solution by steepest descent in Figures 6.2 thru 6.5. Apparently the method gives very good results even for as few as eight Walsh functions.

Iteration	$\| \delta x(t) \|_{max}$	$\frac{\|e(t;\varepsilon)\|^2}{\varepsilon}$	$\frac{\|\rho(\varepsilon)\|^2}{\varepsilon}$	$J(\varepsilon, x(t), u(t))$
1	1.1262	0.4500	0.0067	4.4067
2	0.5494	0.0326	0.0399	4.0225
3	0.3177	0.1363	0.0162	4.1024
4	0.2159	0.1670	0.0211	4.1381
5	0.1251	0.1946	0.0254	4.1700
6	0.1022	0.1439	0.0166	4.1105
7	0.0983	0.0959	0.0115	4.0574
8	0.1063	0.0563	0.0104	4.0167
9	0.1151	0.0280	0.0124	3.9904
10	0.1136	0.0130	0.0156	3.9785
11	0.0931	0.0078	0.0176	3.9754
12	0.0592	0.0067	0.0181	3.9748
13	0.0289	0.0067	0.0180	3.9747
14	0.0118	0.0068	0.0179	3.9747
15	0.0046	0.0069	0.0178	3.9747
16	0.0019	0.0070	0.0178	3.9747
17	0.0009	0.0070	0.0178	3.9748
18	0.0004	0.0070	0.0178	3.9748
19	0.0002	0.0070	0.0178	3.9748
20	0.0001	0.0070	0.0178	3.9748
21	0.0001	0.0070	0.0178	3.9748

Table 6.2: Solution - Minimum Time Problem

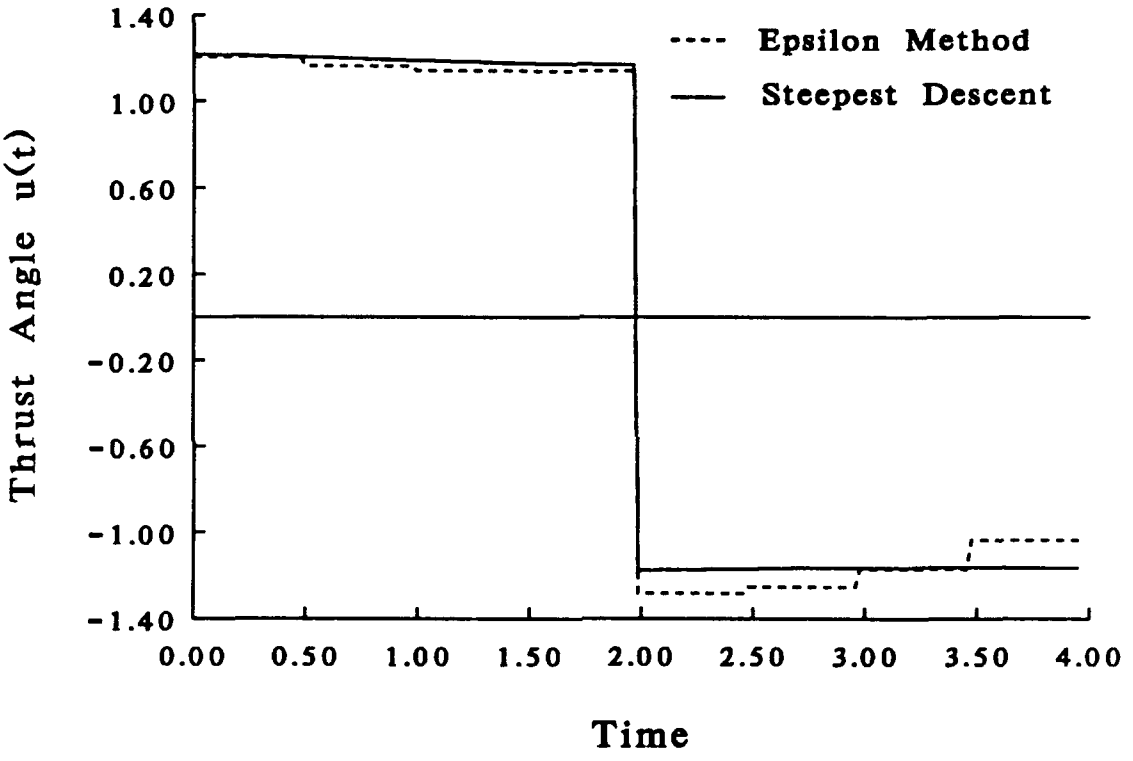


Figure 6.2: Optimal Thrust Angle - Minimum Time Solution

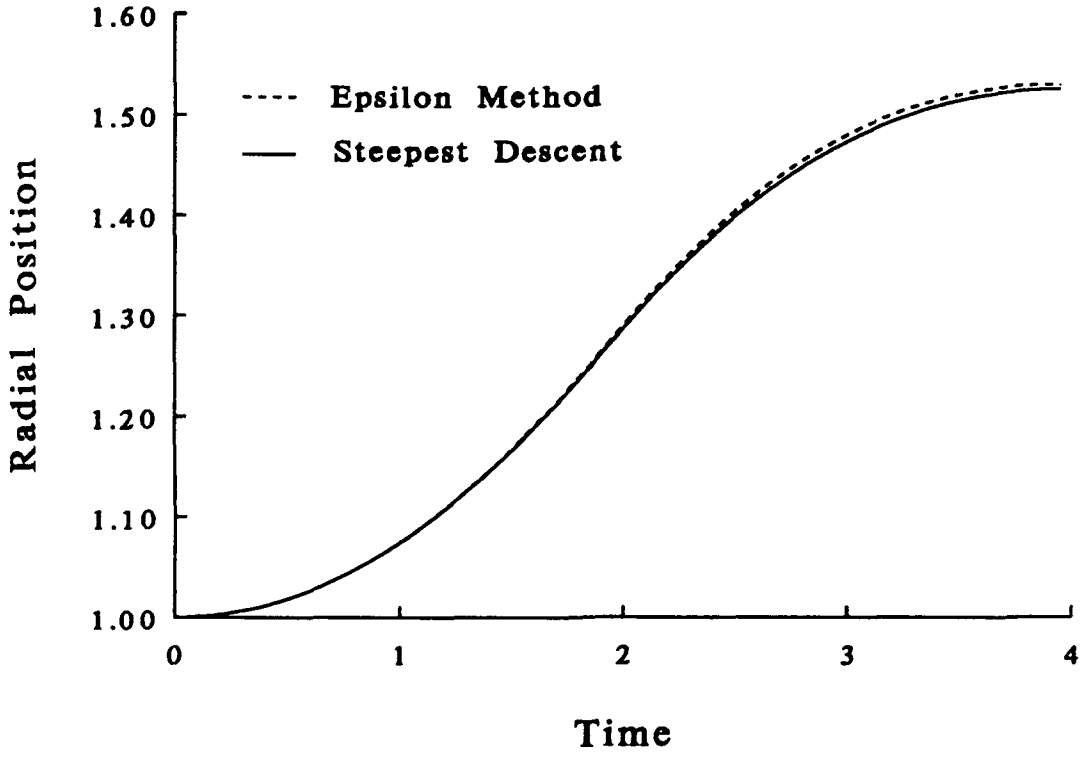


Figure 6.3: Radial Position - Minimum Time Solution

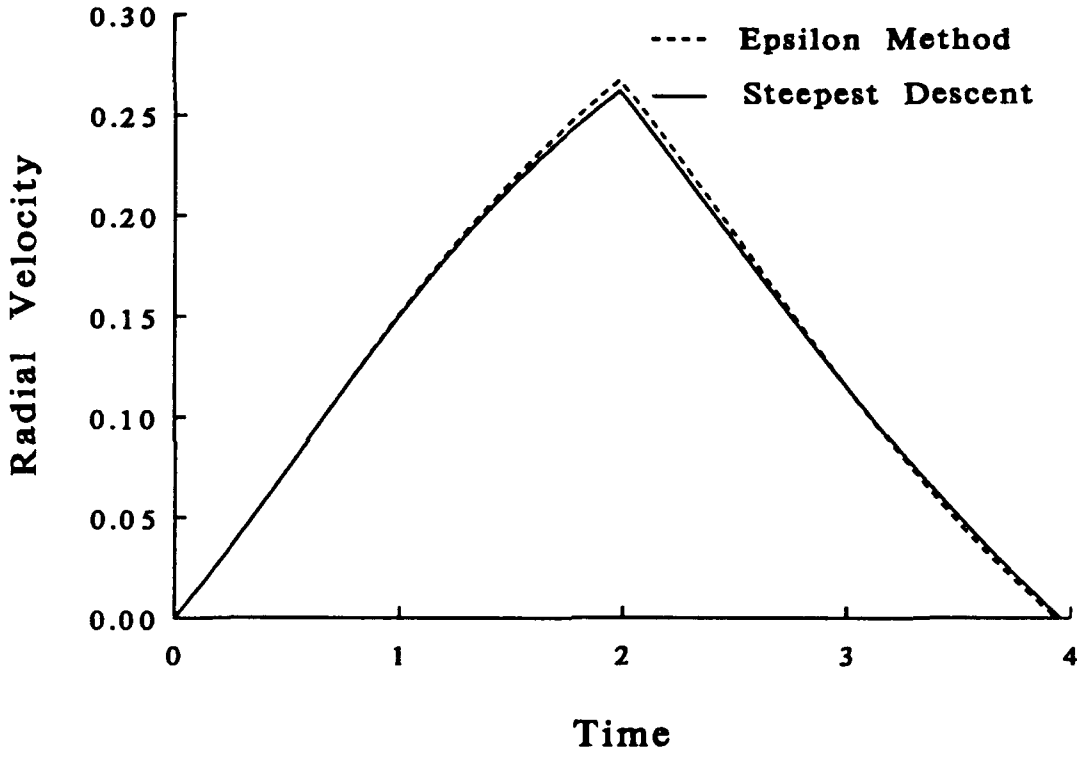


Figure 6.4: Radial Velocity - Minimum Time Solution

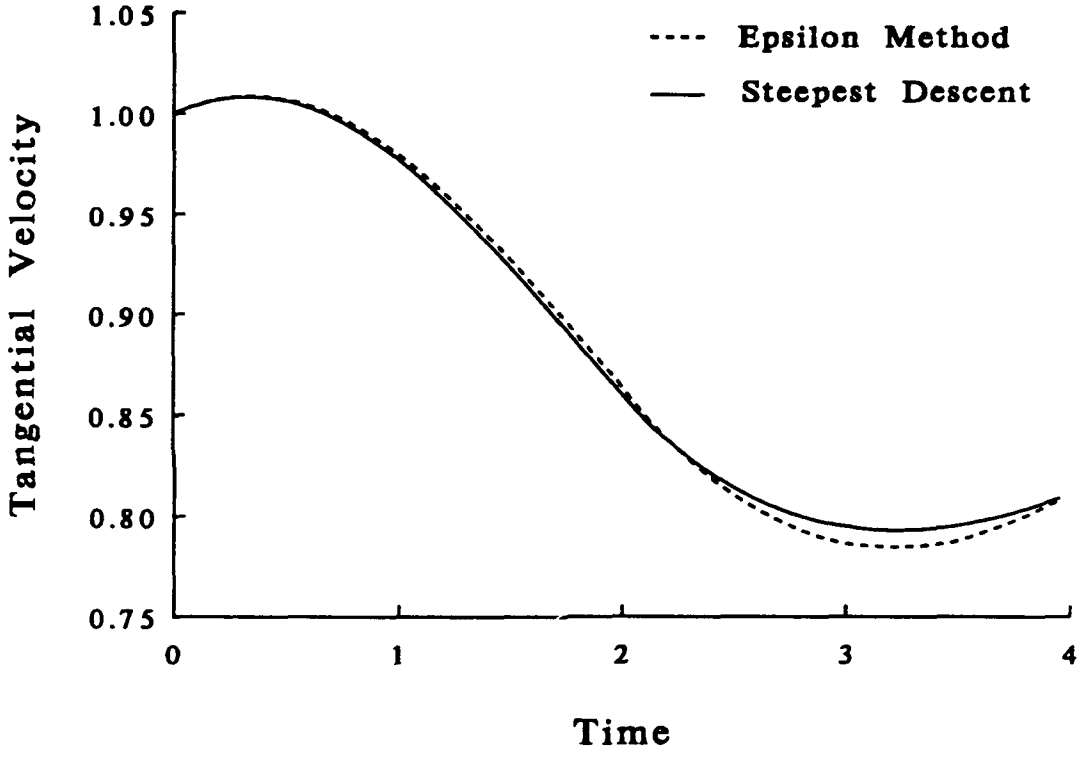


Figure 6.5: Tangential Velocity - Minimum Time Solution

Although the minimum time example was solved on a serial machine, implementation of the problem in parallel poses no new problems and speed-ups similar to previous parallel examples can be expected.

CHAPTER 7

OPTIMAL CONTROL PROBLEMS WITH INEQUALITY CONSTRAINTS

In many optimal control problems, the state or control must be constrained to stay within certain limits. Consider the following problem

$$\dot{x}(t) = f(x(t), u(t), t) \qquad x(t_0) = x_0 \tag{7.1}$$

$$V(x(t), u(t)) = \int_0^{t_f} G(x(t), u(t), t)dt \tag{7.2}$$

and

$$| u(t) | \leq c \tag{7.3}$$

where c is some constant. In this case, it is desired to find the control which minimizes V while meeting the dynamic constraint of (7.1) and the inequality constraint of (7.3). The control is constrained to remain within some specified limits, so this is a constrained dynamic optimization problem. Contrary to the previously solved problems, there are two constraints which must be taken into account: the equality constraint of the dynamical equations and the inequality constraint on the control. The linear time-varying optimal control problem with no inequality constraints was solved in chapter 3. By conversion into the integral epsilon problem and solution using the Rayleigh-Ritz method, the dynamic optimization problem was converted into a static optimization with the following cost function:

$$J = \frac{1}{2\varepsilon} vec(E)^T vec(E) + \frac{1}{2} vec(X)^T (I_N \otimes Q) vec(X) + \frac{1}{2} vec(U)^T (I_N \otimes R) vec(U) \tag{7.4}$$

where

$$\begin{aligned} vec(E) = \; & \{vec(X) - vec(G_s) - [(P^T \otimes I_n)\Lambda_\alpha] vec(X) \\ & - [(P^T \otimes I_n)\Lambda_\beta] vec(U) - (P^T \otimes I_n) vec(C)\} \end{aligned} \tag{7.5}$$

This cost function is minimized simultaneously with respect to both the Walsh coefficients for the state $vec(X)$ and the Walsh coefficients for the control $vec(U)$. If there are inequality constraints on the state or the control, then the above cost function must be minimized while meeting the inequality constraint. Since the cost function is minimized with respect to the Walsh coefficients, the inequality constraint must be formulated in terms of the Walsh coefficients. Once this is done the problem becomes a static constrained optimization problem. Similar to the epsilon method, this problem can be solved by adjoining the inequality constraint to the cost function as a penalty term and then minimizing with respect to the Walsh coefficients. This essentially converts the static constrained optimization problem into an unconstrained problem. The key then is to be able to formulate the inequality constraint as given in (7.3) into an equivalent inequality constraint on the Walsh coefficients. The next section shows how an inequality constraint on the control or state can be formulated in terms of the Walsh coefficients.

Formulating Walsh Coefficient Inequality Constraints

Consider an optimal control problem with the following inequality constraint,

$$| u(t) | \leq c \tag{7.6}$$

where c is some constant. This inequality constraint must be converted into a constraint on the Walsh coefficients of the control. The inequality constraint must be given in terms of the Walsh coefficients. This can be done using the $\mathcal{P}$ matrix given in chapter 6. For example, c can be described by a vector of N elements of value c. Let

$$c = C_t = \left[\begin{array}{cccc} c_t^0 & c_t^1 & \cdots & c_t^{N-1} \end{array} \right]$$

where C_t is a vector of N values for which $c_t^i = c$. In the same manner

$$u(t) = U_t = \left[\begin{array}{cccc} u_t^0 & u_t^1 & \cdots & u_t^{N-1} \end{array} \right]$$

Now the inequality constraint can be written

$$| u_t^i | \leq c_t^i \quad i = 0, 1, 2, \ldots, N-1$$

Since

$$U_t = U\mathcal{P} = Up_i \quad i = 0, 1, 2, \ldots, N-1 \tag{7.7}$$

the inequality can be written

$$| Up_i | \leq c_t^i \quad i = 0, 1, \ldots, N-1 \tag{7.8}$$

In (7.8), p_i are the vectors which describe the Walsh functions as shown in (6.1). The next section gives one method which may be used to solve the constrained optimization problem given by (7.4) and (7.8).

Solution of Optimal Control Problems with Inequality Constraints

In a manner similar to the epsilon problem, the constrained optimization problem given by (7.4) and (7.8) can be converted into an unconstrained problem by adjoining the inequality constraints to the cost function in the form of a penalty term. The following penalty term may be used,

$$\frac{1}{2\gamma} \| \max(0, | Up_i | - c_{t_i}) \|^2 \tag{7.9}$$

In this case, only the constraints which are violated will appear in the penalty term. A good approach then is to determine at each iteration the constraints which are active and form a matrix $\mathcal{P}'$ and C_t' which consist of only the active constraints. This matrix and vector are then used to form the penalty term which must be in *vec* form.

$$\frac{1}{2\gamma} \| (\mathcal{P}'^T \otimes I_n) vec(U) - vec(C_t') \|^2 \tag{7.10}$$

This term is added to the cost function of equation (7.4). Let

$$vec(\mathcal{E}) = (\mathcal{P}'^T \otimes I_n) vec(U) - vec(C_t') \tag{7.11}$$

then the cost function becomes

$$\begin{aligned} J = \; & \tfrac{1}{2\varepsilon} vec(E)^T vec(E) + \tfrac{1}{2} vec(X)^T (I_N \otimes Q) vec(X) \\ & + \tfrac{1}{2} vec(U)^T (I_N \otimes R) vec(U) + \tfrac{1}{2\gamma} vec(\mathcal{E})^T vec(\mathcal{E}) \end{aligned} \tag{7.12}$$

Now the cost function can be minimized as before and the system of equations

$$\begin{bmatrix} K_{11} & K_{12} \\ K_{21} & K_{22} \end{bmatrix} \begin{pmatrix} vec^*(X) \\ vec^*(U) \end{pmatrix} = \begin{pmatrix} D_1 \\ D_2 \end{pmatrix} \tag{7.13}$$

is obtained by letting

$$M = \begin{bmatrix} I_{Nn} - (P^T \otimes I_n)\Lambda_\alpha & -(P^T \otimes I_n)\Lambda_\beta \\ 0 & 0 \end{bmatrix} \tag{7.14}$$

and

$$L = \begin{bmatrix} I_N \otimes Q & 0 \\ 0 & I_N \otimes R \end{bmatrix} \tag{7.15}$$

and

$$F = \begin{bmatrix} 0 & 0 \\ 0 & \mathcal{P}'^T \otimes I_n \end{bmatrix} \tag{7.16}$$

Then

$$K = \frac{1}{\varepsilon} M^T M + L + \frac{1}{\gamma} F^T F \tag{7.17}$$

and a term must also be added to D

$$D_2 = D_2 + \frac{1}{\gamma} (\mathcal{P}'^T \otimes I_n)^T vec(C_t') \tag{7.18}$$

In (7.18), D_2 is the same as developed in Chapter 3. To incorporate inequality constraints, at each iteration the active constraints are determined. This involves checking N intervals of the state or control. If the value of the state or control in an interval exceeds c_t^i, that constraint must become active. Once all the active constraints are determined, the cost function is minimized by solving the system of equations given by including the modifications above. The next section gives a numerical example to demonstrate the method.

Computational Results

To demonstrate the method given in the previous section, following problem was solved.

PROBLEM 7.1:

$$\begin{aligned} \dot{x_1} &= x_2 & x_1(0) &= 0 \\ \dot{x_2} &= -x_1 + x_2 - x_1^2 x_2 + u & x_2(0) &= 1 \end{aligned} \tag{7.19}$$

$$V = \int_0^5 (x_1^2 + x_2^2 + u^2) dt \tag{7.20}$$

This is a van der Pol problem with no terminal constraints. Now Problem 7.2 is obtained by adding an inequality constraint to the control.

PROBLEM 7.2:

$$\begin{aligned} \dot{x_1} &= x_2 & x_1(0) &= 0 \\ \dot{x_2} &= -x_1 + x_2 - x_1^2 x_2 + u & x_2(0) &= 1 \end{aligned} \tag{7.21}$$

$$V = \int_0^5 (x_1^2 + x_2^2 + u^2) dt \tag{7.22}$$

$$| u(t) | \leq .8 \tag{7.23}$$

To solve both problems, eight Walsh functions were used and epsilon and gamma were set to 0.0001. Figure 7.1 shows the optimal control and state for Problem 7.1. Figure 7.2 shows the solution when the control is constrained to stay between $-.8$ and .8

Although the above example was solved on a serial computer, implementation of the problem in parallel would involve only minor modifications to the parallel implementation given in Chapter 3. The only addition would be the determination of the active constraints. Since each of the N intervals are independent of each other this may be done in parallel. As shown with the addition of the penalty term for terminal constraints in Chapter 4, addition of another penalty term will not effect parallelization of the problem.

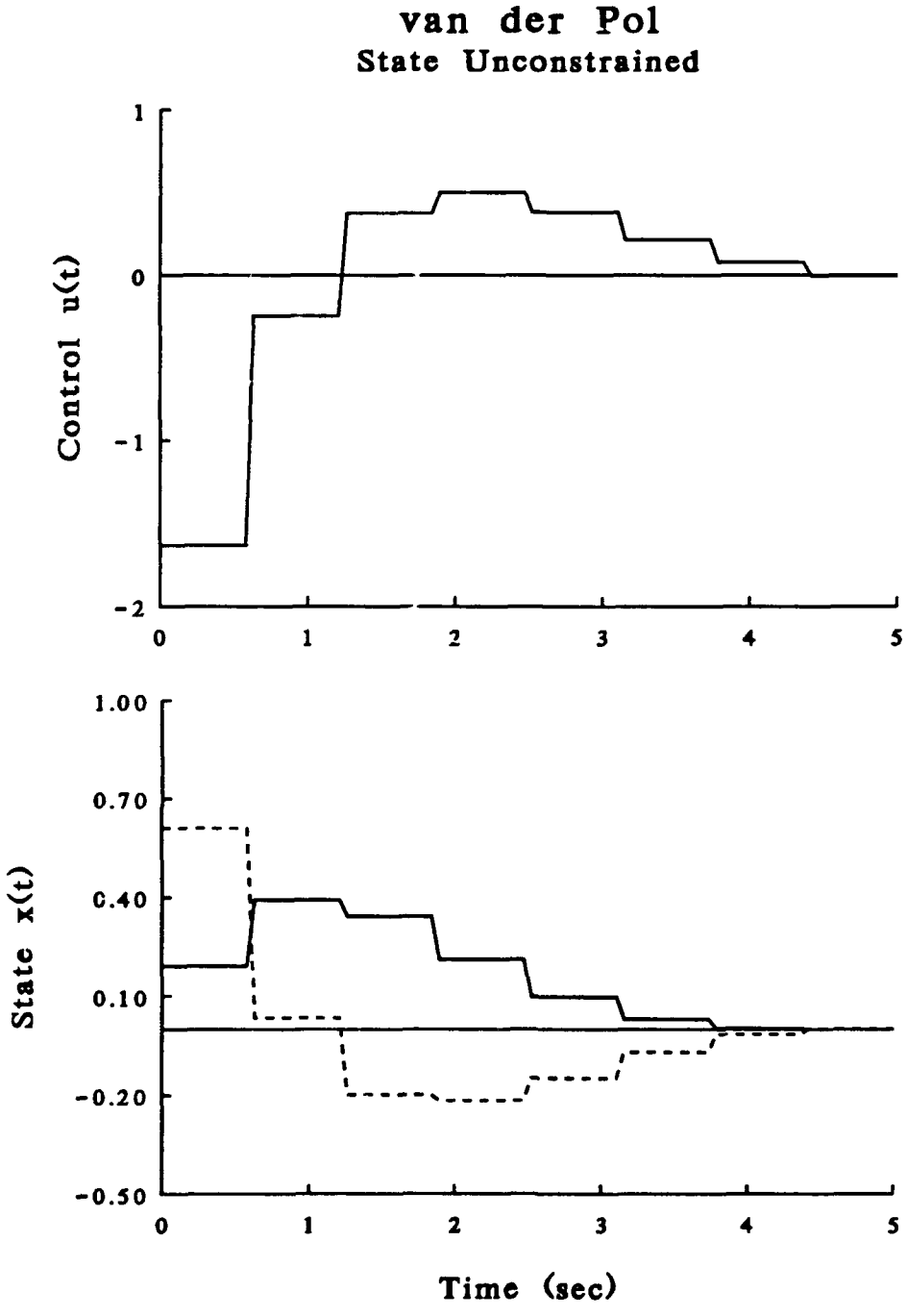


Figure 7.1: Problem 7.1 Solution with Eight Walsh Functions

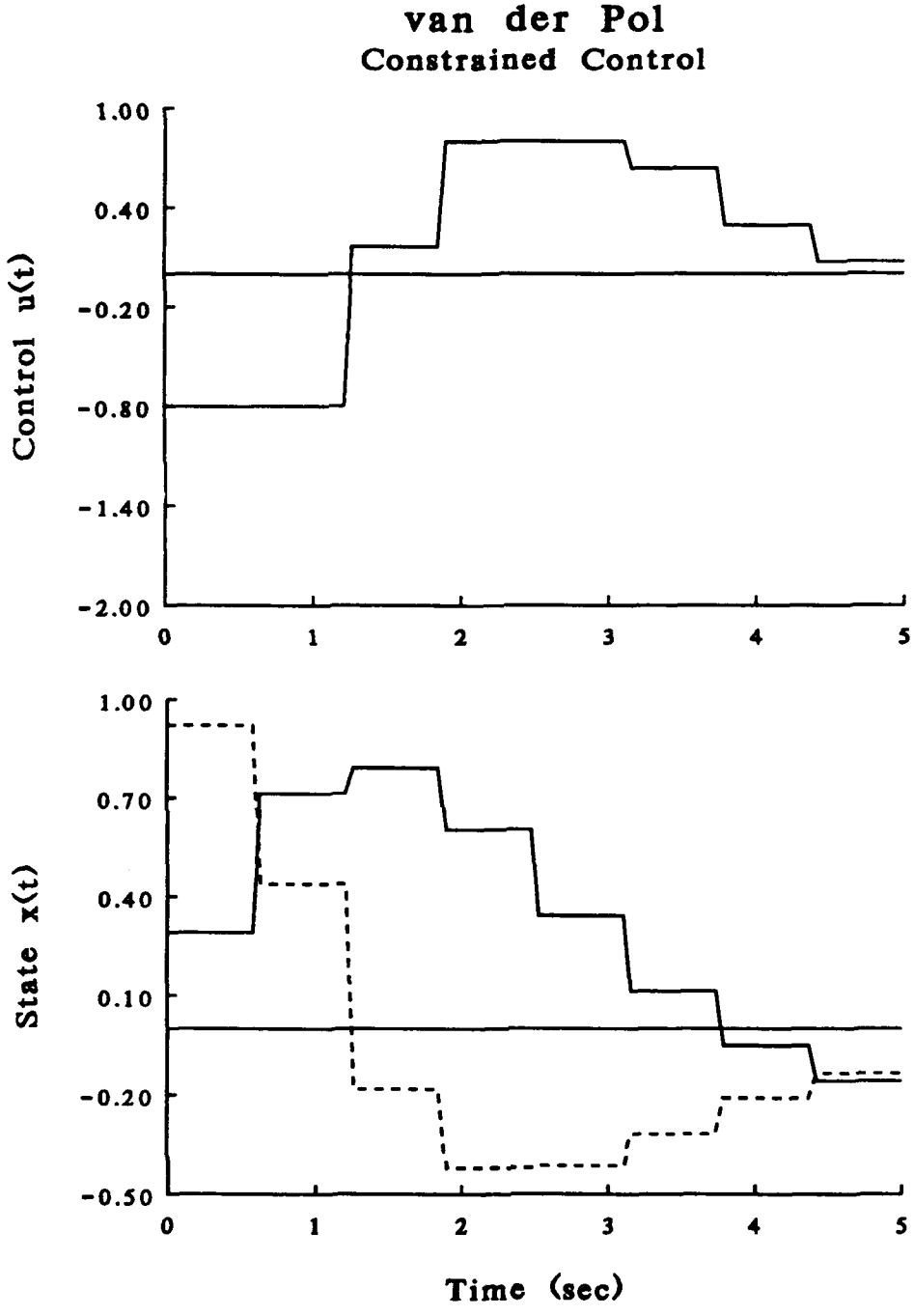


Figure 7.2: Problem 7.2 Solution with Eight Walsh Functions

CHAPTER 8

CONCLUSIONS

This research has developed and implemented a parallel solution method for nonlinear optimal control problems. Balakrishnan's epsilon method converts the two point boundary value problem into a nonlinear programming problem thus avoiding the serial bottlenecks inherent in methods which solve the dynamical equations. The solution to the nonlinear programming problem can be found using simple matrix operations which have parallel implementations with the desirable properties of flexibility and scalability.

To test the method, a variety of nonlinear optimal control problems were solved. The Raleigh problem represents the case of nonlinear dynamical equations with a quadratic cost and the van der Pol problem represents the case of nonlinear dynamical equations with a quadratic cost and terminal constraints on the state. Using eight Walsh functions or Legendre polynomials to approximate the state and control, the problems could be solved in four to five iterations from relatively poor initial guesses for the nominal trajectories. On an eight node *Intel Hypercube*, solution time was 4-5 seconds and speed-up on the order of 3.4-3.5.

The Raleigh problem and van der Pol problem contain only polynomial nonlinearities in the system dynamical equations. It is simple to perform complex nonlinear operations on Walsh functions, making them ideal for the solution of problems with nonlinearities which are more complex than just polynomial nonlinearities. This was demonstrated by solution of a nonlinear minimum time optimal control problem and nonlinear optimal control problem with an inequality constraint on the control. Results for the nonlinear minimum time problem were compared to results

obtained by solution of the same problem using the technique of steepest descent. Results show the method gives accurate results and is, therefore, a feasible technique for the rapid solution of complex nonlinear control problems.

Since it is not apparent how to deal with other than polynomial nonlinearities when using orthogonal polynomials, it would appear that the Walsh functions have a broader range of applications. They would also be more effective when considering problems with discontinuities in the state or control. On the other, hand preliminary results show that for problems with continuous state and control a good approximation may be obtained using very few orthogonal polynomials. As a result, the best basis functions to solve a problem will be a function of the particular problem to be solved.

Finally, it should be pointed out that the parallel implementations given in this thesis were geared toward a specific parallel computer. Matrix operations have a variety of parallel implementations. As a result, it is possible to solve the optimal control problem on a variety of parallel computers or parallel processing arrays. In addition to this implementation flexibility, potential parallelization is very high. For example using eight Walsh functions up to 560 processors might be used to solve the minimum time problem given in Chapter 6. The method offers the potential for very fast solution times of complex nonlinear optimal control problems.

Future Research Areas

This research paves the way to many additional research areas. Only a small parallel machine was available to solve the example problems. It would be a powerful demonstration to solve a complex nonlinear optimal control problem, such as the minimum time problem, on a very large parallel computer ($p > 500$). The goal may be solution times approaching real-time.

Inherent in the given parallel solution technique is a highly parallel solution technique for nonlinear ordinary differential equations. There are other areas of estimation and control in which a parallel solution method for solving ordinary differential equations would be useful. For example, it may allow a highly parallel

solution method for extended Kalman filters or parallel solution of Riccati equations.

Frick showed that Walsh functions could be used to approximate noise processes [25]. It would be interesting to investigate parallel solution methods for stochastic optimal control problems using these ideas.

Finally, the idea of approximating functions with a set of basis functions to formulate a parallel solution method is a useful one. It is possible the method of approximating functions may be used to reformulate other problems which seem very serial in nature into problems which are highly parallel in nature.

BIBLIOGRAPHY

[1] R. E. Larson and E. Tse, *Parallel Processing Algorithms for the Optimal Control of Nonlinear Dynamic Systems*, **IEEE Transactions on Computers**, Vol. C-22, No. 8, pp. 777-785, Aug 1973.

[2] R. Travassos and H. Kaufman, *Parallel Algorithms for Solving Nonlinear Two-Point Boundary Value Problems which arise in Optimal Control*, **Journal of Optimization Theory and Applications**, Vol. 30, No. 1, pp. 53-71, Jan 1980.

[3] G. G. Meyer and L. J. Podrazik, *Parallel Implementations of Gradient Based Iterative Algorithms for a Class of Discrete Optimal Control Problems*, **Proceedings of the International Conference on Parallel Processing**, pp. 491-494, August 1987.

[4] D. P. Bertsekas and J. N. Tsitsikis, **Parallel and Distributed Computation - Numerical Methods**, Prentice Hall, New Jersey, 1989.

[5] S. Y. Kung, **VLSI Array Processors**, Prentice Hall, New Jersey, 1988.

[6] W. Chen and Y. Shih, *Analysis and Optimal Control of Time Varying Linear Systems via Walsh Functions*, **International Journal of Control**, Vol. 27, p. 917, 1978.

[7] C. Chen and C. Hsiao , *Design of Piecewise Constant Gains for Optimal Control via Walsh Functions*, **IEEE Transactions on Automatic Control**, Vol. 20, No. 5, pp. 596-602, Oct 1975.

[8] V. R. Karanam, P. A. Frick and R. R. Mohler, *Bilinear System Identification by Walsh Functions*, **IEEE Transactions on Automatic Control**, Vol. 27, No. 4, pp. 709-713, Aug 1978.

[9] H. Rademacher, *Einge Satze Uber Reihen Von Allgemen Orthogonal Function*, **Ann. Math.**, Vol 87, p. 712, 1922.

[10] J. L. Walsh, *A Closed Set of Orthogonal Functions*, **American Journal of Mathematics**, Vol. 45, p. 5, 1923.

[11] N. J. Fine, *On the Walsh Functions*, **Transactions of the American Mathematical Society**, Vol. 65, 1949.

[12] F. L. Lewis, **Optimal Control**, John Wiley and Sons, 1986.

[13] A. V. Balakrishnan, *On a new Computing Technique in Optimal Control*, **SIAM Journal on Control**, Vol. 6, No. 2, 1968.

[14] P. A. Frick, *An Integral Formulation of the Epsilon problem and a New Computational Approach to Control Function Optimization*, **Journal of Optimization Theory and Applications**, Vol. 13, pp. 553-581, 1974.

[15] J. W. Brewer, *Kronecker Products and Matrix Calculus in System Theory*, **IEEE Trans. Circuits and Systems**, Vol CAS-25, No 9, pp.772-781, 1978.

[16] A. Bojanczyk, R. P. Brent and H. T. Kung, *Numerically Stable Solution of Linear Equations using Mesh-Connected Processors*, **SIAM Journal of Scientific and Statistical Computing**, Vol. 5, pp. 95-104, 1984.

[17] D. E. Kirk, *Optimal Control Theory*, **Prentice-Hall**, New Jersey 1970.

[18] A. V. Balakrishnan, *On a new Computing Technique in Optimal Control and its Application to Minimal Time Flight Profile Optimization*, **Journal of Optimization and Applications**, Vol. 4, No. 1, 1968, pp. 1-21.

[19] P. N. Paraskevopoulos, *Legendre Series Approach to Identification and Analysis of Linear Systems*, **IEEE Transactions on Automatic Control**, Vol. 30, No. 6, pp. 585-589, 1985.

[20] C. Hwang, and M. Chen, *Analysis and Optimal Control of Time-Varying Linear Systems via Shifted Legendre Polynomials*, **International Journal of Control**, Vol. 41, No. 5, pp. 1317-1330, 1985.

[21] J. Chow, and I. Horng *Application of Chebyshev Polynomials to Optimal Control of Time-Varying Linear Systems*, **International Journal of Control**, Vol. 41, No. 1, pp. 135-144, 1985.

[22] C. Liu, and Y. Shih, *Analysis and Parameter Estimation of Bilinear Systems via Chebyshev Polynomials*, **Journal of the Franklin Institute**, Vol. 317, No. 6, pp. 373-382, 1984.

[23] J. M. Taylor, *Optimization: Application of the Epsilon Method*, University of Wyoming, PhD Thesis, 1970.

[24] D. G. Luenberger, **Linear and Nonlinear Programming**, Addison-Wesley, 1984.

[25] P. A. Frick, *On Walsh Noise: Its Properties and Use in Dynamic Stochastic Systems*, **IEEE Transactions on Systems, Man, and Cybernetics**, Vol. SMC-9, No. 8, pp. 411-419, August 1979.

Appendix A

Notation

This appendix reviews some very powerful systems notation as given from a paper by Brewer [15]. The notation is used in the solution of the epsilon problem using the Ritz method with some orthogonal functions as a basis.

Let A be a matrix of dimension $p \times q$, B is dimension $s \times t$ and D is dimension $q \times s$. Then vec of A is a vector of dimension $pq \times 1$ defined as

$$vec(A) = \begin{pmatrix} A_{\cdot 1} \\ A_{\cdot 2} \\ \vdots \\ A_{\cdot q} \end{pmatrix} \tag{A.1}$$

Here the $A_{\cdot 1}$ notation refers to the first column of A and $A_{\cdot 2}$ refers to the second and so on. So the $vec(A)$ amounts to stacking the columns of A on top of each other into a large vector.

The symbol $\otimes$ denotes the Kronecker product. It is defined as:

$$A \otimes B = \begin{bmatrix} a_{11}B & a_{12}B & \cdots & a_{1q}B \\ a_{21}B & a_{22}B & & \vdots \\ \vdots & & & \\ a_{p1}B & \cdots & & a_{pq}B \end{bmatrix} \tag{A.2}$$

So $A \otimes B$ is a $ps \times qt$ matrix.

Finally, it can be shown that

$$vec(ADB) = (B^T \otimes A)vec(D) \tag{A.3}$$

Using this identity the *vec* of the product of two matrices can be found.

$$\begin{aligned} vec(AD) &= vec(ADI_s) = (I_s \otimes A)vec(D) \\ vec(AD) &= vec(I_pAD) = (D^T \otimes I_p)vec(A) \\ vec(AD) &= vec(AI_qD) = (D^T \otimes A)vec(I_q) \end{aligned} \tag{A.4}$$